Izrada horor igre preživljavanja u programskom alatu Unity

Alilović, Dario

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Dario Alilović

IZRADA HOROR IGRE PREŽIVLJAVANJA U PROGRAMSKOM ALATU UNITY

ZAVRŠNI RAD

Varaždin, 2020.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Dario Alilović

Matični broj: 45885/17–R

Studij: Informacijski sustavi

IZRADA HOROR IGRE PREŽIVLJAVANJA U PROGRAMSKOM ALATU UNITY

ZAVRŠNI RAD

Mentor:

Doc. dr. sc. Konecki Mario

Varaždin, rujan 2020.

Dario Alilović

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/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

U ovom radu upoznati ćemo se sa programskim alatom Unity, te izraditi vlastitu horor igru preživljavanja. Napraviti ćemo detaljan opis programskog alata Unity, njegovu povijest i gdje se on danas koristi. Također ćemo opisati žanr horor igara preživljavanja, te navesti nekoliko primjera igara iz tog žanra. Proći ćemo kroz nekoliko tehnika, metoda i algoritama koji se koriste prilikom izrade računalnih igara te ih primijeniti prilikom izrade vlastite igre.

Ključne riječi: Unity; Video igra; Horor; Programiranje;

Sadržaj

1. Unity

U ovom poglavlju opisati ćemo programski alat Unity. Unity je program za izradu računalnih igara (game engine) koji podržava razvoj video igara na većini platformi uključujući desktop računala, računalne konzole, virtualnu stvarnost, proširenu stvarnost, mobilne uređaje (android, IOS), TV platforme, Web platforme i ostale[1]. David Helgason, Nicholas Francis i Joachim Ante razvili su prvu verziju Unity-a koja je izdana u javnost 2005. godine. Nastavili su ga razvijati i poboljšavati sve do 2008. kada je Unity počeo rasti u popularnosti, što je omogućilo širenje tvrtke. Te iste godine Apple je uveo svoj App Store koji je privukao pozornost programera Unity-a. Ubrzo su razvili podršku za iPhone, a pošto su bili prvi u industriji koji su to izveli, popularnost Unity-a je samo porasla. Velike tvrtke, poput Cartoon Network-a, počele su koristiti Unity za razvoj vlastitih igara što je još više povećalo popularnost Unity-a. Unity je danas među vodećim razvojnim okolinama za izradu računalnih igara, ako ne i najveća[2].

1.1. Unity korisničko sučelje

Kod razvojnih alata korisničko sučelje je jako bitno i sam poredak i izgled grafičkog sučelja direktno utječe na utrošeno vrijeme i produktivnost. Unity korisničko sučelje raspoređeno je u nekoliko prozora:

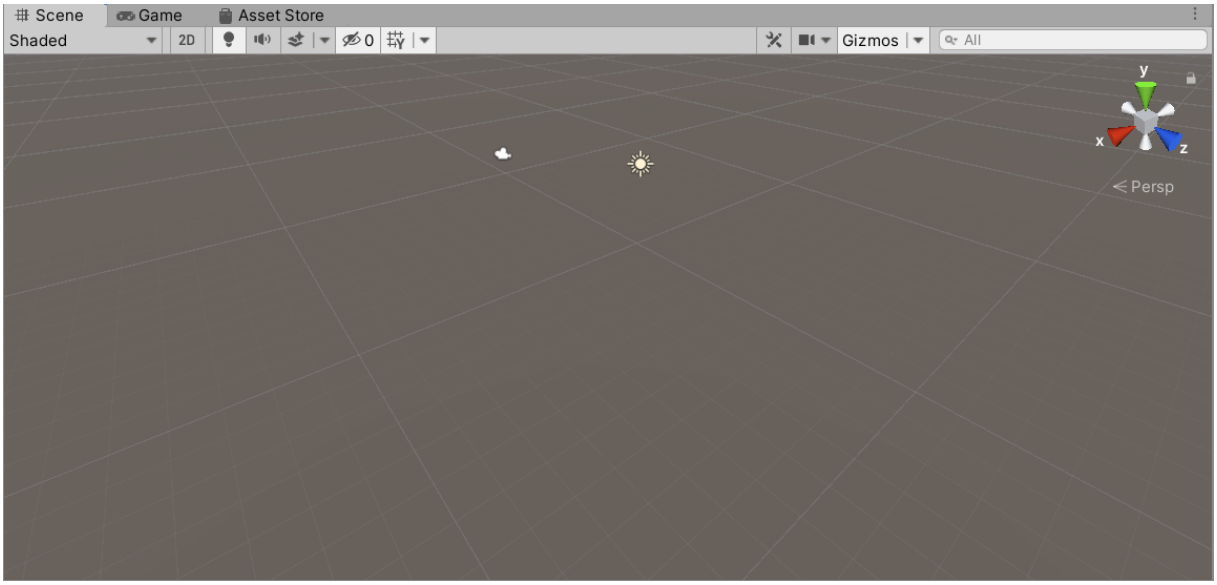
- Project
- Scene view
- Game view
- Hierarchy
- Inspector
- Toolbar
- Dodatni

Ovi prozori služe za upravljanje i manipulaciju programskog okruženja, te lagano snalaženje u alatu što nam omogućava brz i efikasan rad na projektima[3].

Project window (slika 1) prikazuje sve datoteke koje su vezane za trenutni projekt i koristimo ga za brzu i laganu navigaciju u projektu. Najčešće se nalazi u donjem dijelu prozora, ali ga možemo premjestiti ovisno o vlastitim potrebama. Na lijevoj strani prikazana je hijerarhija mapa i datoteka u projektu, dok na desnoj strani vidimo sadržaj neke mape. Također sadrži alatnu traku za pretraživanje, alat za dodavanje novih datoteka te gumb za skrivanje ili pokazivanje skrivenih paketa[4].

Slika 1: Project window

Scene view (slika 2) je interaktivni prozor u alatu Unity koji nam služi za kreiranje, pregled i generalnu interakciju sa svijetom koji stvaramo. U njega možemo postavljati objekte (game objects) kao što su kamera, svjetla, oblici, igrači i slično. Ovo je jedan od glavnih prozora koji se koriste za kreiranje izgleda same igre[5].



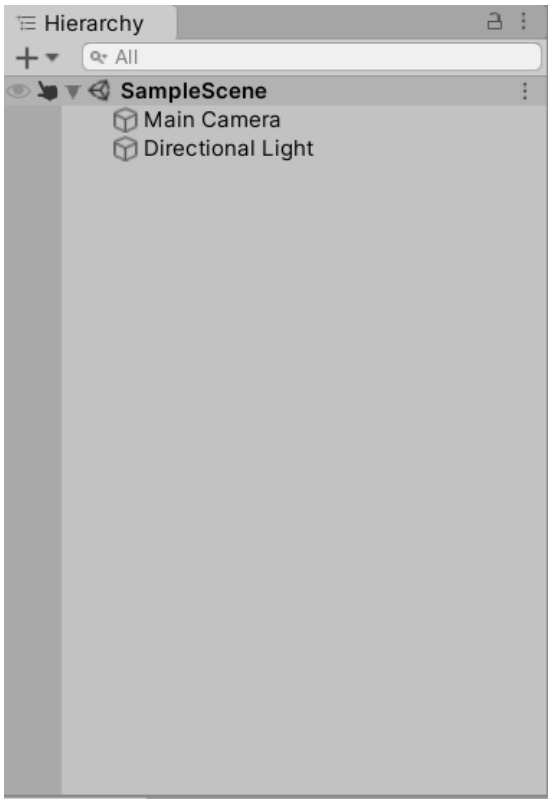
Slika 2: Scene view

U ovom prozoru nalazi se nekoliko alata koji omogućuju lagano kretanje unutar scene prozora. Jedan od tih alata je takozvani „Scene Gizmo“ koji prikazuje trenutnu orijentaciju, te se koristi za promjenu kuta gledanja, a nalazi se u gornjem desnom kutu scene prozora. Sa lijeve strane **alatne trake** koja se nalazi na vrhu prozora nalaze se scene view kontrole (prikazane na slici 3) koje nam omogućavaju pomicanje, rotiranje, povećavanje i smanjivanje

objekata i ostale slične funkcionalnosti koje koristimo za manipulaciju objekata unutar scene prozora[6].

Slika 3: Alati za manipulaciju

Za odabir određenog objekta potrebno je jednostavno kliknuti na njega ili ga pronaći u **hierarchy** prozoru koji se najčešće nalazi sa lijeve strane Unity alata (slika 4). Unutar prozora hierarchy možemo isključiti mogućnost odabira određenog objekta koristeći „pickability“ ikonu koja se nalazi u gornjem lijevom kutu (ruka) ili sakriti određeni objekt koristeći „visibility“ ikonu koja se isto tako nalazi u gornjem lijevom kutu (oko)[7][8].

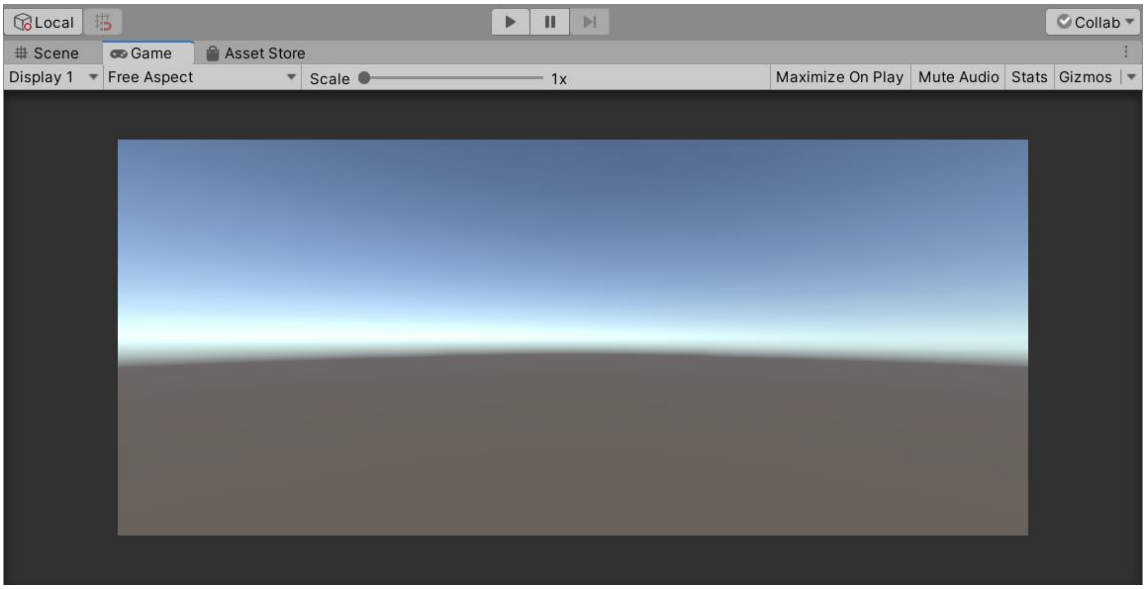


Slika 4: Hierarchy

Kada smo odabrali objekt možemo mijenjati njegovu veličinu, poziciju, izgled, rotaciju, itd. koristeći alate za manipulaciju (slika 3). Kada odaberemo željeni alat, kontrole nam se

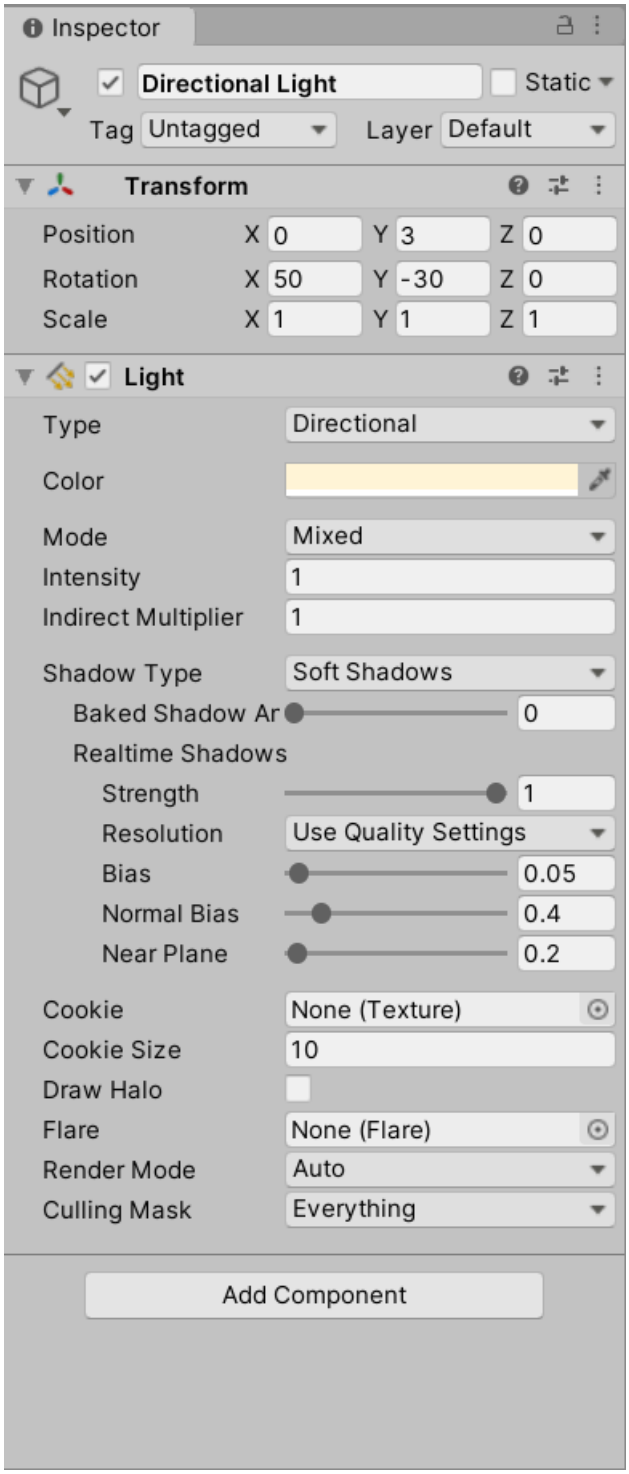
prikažu u scene prozoru unutar selektiranog objekta. Ove mogućnosti ćemo koristiti kako bi smo izradili adekvatnu mapu na kojoj će se igra odvijati[9].

Game view (slika 5) je pogled iz kamera koje smo postavili u scene view, on nam daje ideju kako će finalni proizvod izgledati, te u njemu možemo testirati igru. U alatima iznad imamo gumbe za upravljanje game view prozorom, gdje možemo paliti, gasiti, pauzirati ili ići korak naprijed u igri[10]. Unutar ovog prozora, provoditi će se testiranje performansi igre kako bi mogli pronaći dijelove koda koji možda stvaraju probleme, te testiranje same igre.



Slika 5: Game view

Inspector (slika 6) je prozor koji ćemo najviše koristiti tijekom izrade ove igre. U njemu se nalaze sve informacije o selektiranom objektu, te sve eventualne komponente koje su pridodane tom elementu. O komponentama ćemo reći nešto više u daljnjem tekstu. Unutar inspector prozora možemo mijenjati razne postavke našeg objekta kako bi smo ga prilagodili našim potrebama. Neke od tih postavki su veličina, rotacija, pozicija, boja, sjena, materijal i slično. U inspektoru također možemo zaključati trenutni objekt ako ga ne želimo izgubiti dok radimo sa drugim objektima. To možemo napraviti klikom na lokot koji se nalazi u gornjem desnom kutu inspector prozora. Pojedinim objektima je isto tako moguće i promijeniti ikonu koja je vidljiva u scene view prozoru kako bi mogli lakše razlikovati određene objekte koji možda imaju neke sličnosti[11].



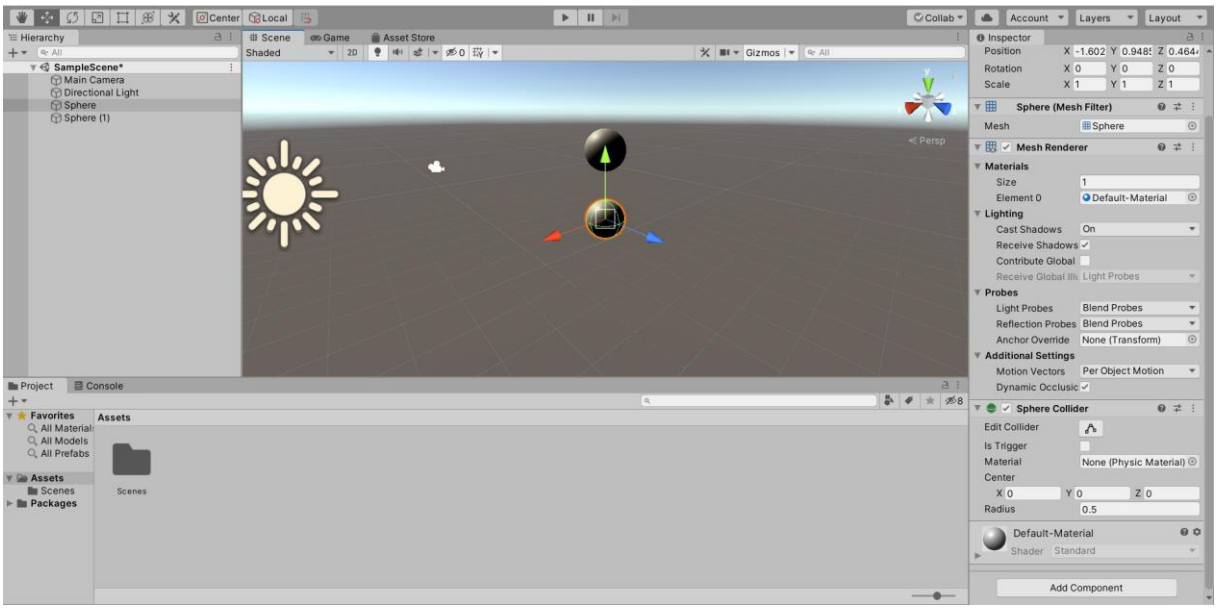
Slika 6:Inspector

1.2. Osnovni koncepti izrade računalnih igara

1.2.1. Manipulacija objekata

Sada kada znamo osnove Unity korisničkog sučelja možemo početi sa teorijom izrade računalnih igara. Da bi izradili igru moramo znati neke osnovne koncepte izrade računalnih igara. U ovom poglavlju biti će objašnjeni neki osnovni koncepti koje ćemo kasnije koristiti u izradi naše igre.

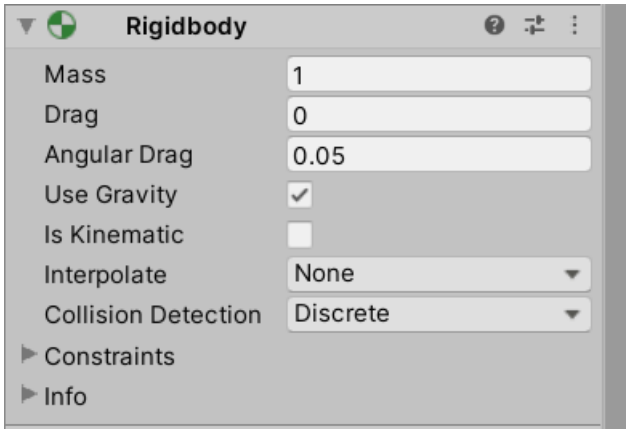
Prilikom izrade računalnih igara potrebno je izraditi objekte koji će biti postavljeni u naš svijet. Kako bismo stvorili objekt koji odgovara našim potrebama moramo koristiti inspector, te pomoću njega uređivati objekt i dodavati mu različite komponente. Uzmimo za primjer kuglu sa slike 7. Vidimo da ova kugla ima određene parametre kao što su pozicija, rotacija, veličina i ostali, ali uz to ima još dodane komponente collider i material. Kugla iz našeg primjera kao collider ima definiran sphere colider koji zapravo definira fiziku ove kugle. Ovaj collider zapravo definira čvrste rubove kroz koje ostali objekti ne mogu prolaziti, te tako dobivamo čvrstu kuglu. Materijal koji je postavljen na ovu kuglu je predefinirani materijal koji se kasnije može promijeniti.



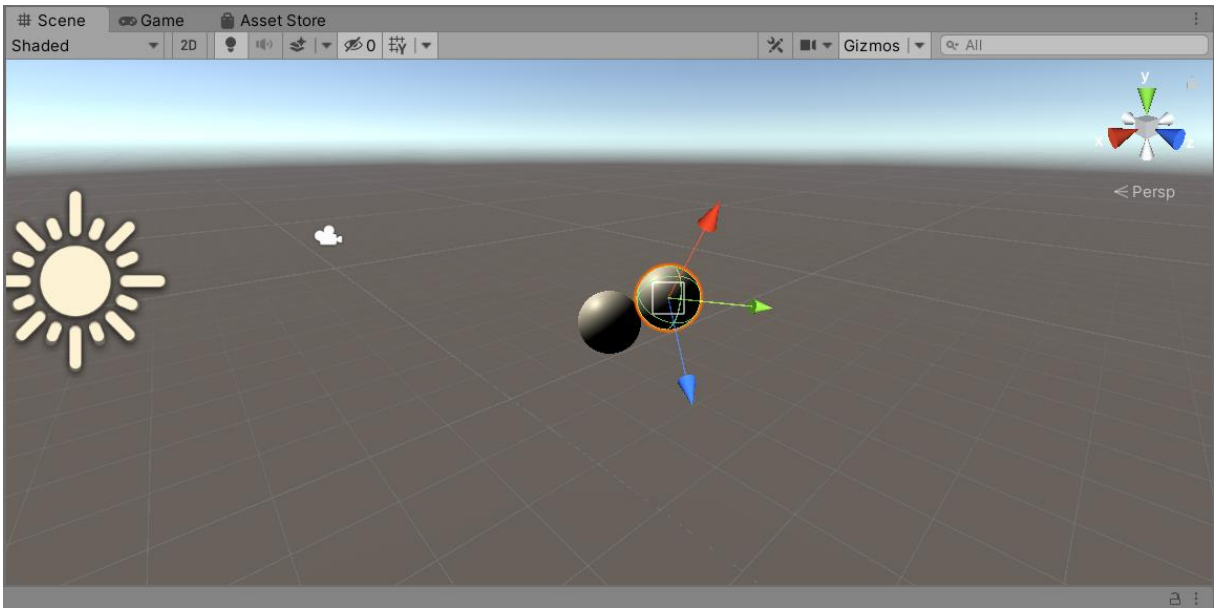
Slika 7: Manipulacija objekata

Ako sada gornjoj kugli dodamo komponentu „rigid body“ i označimo opciju „Use Gravity“ (slika 8), na nju će početi utjecati gravitacija te će se sudariti sa donjom kuglom koja

nema gravitaciju (slika 9)[12]. Objekti koji koriste komponentu „rigid body“ se ne smiju upravljati kroz skriptu koristeći trasform metodu, nego mu se dodaje force na temelju kojega physics engine računa rezultat[19].

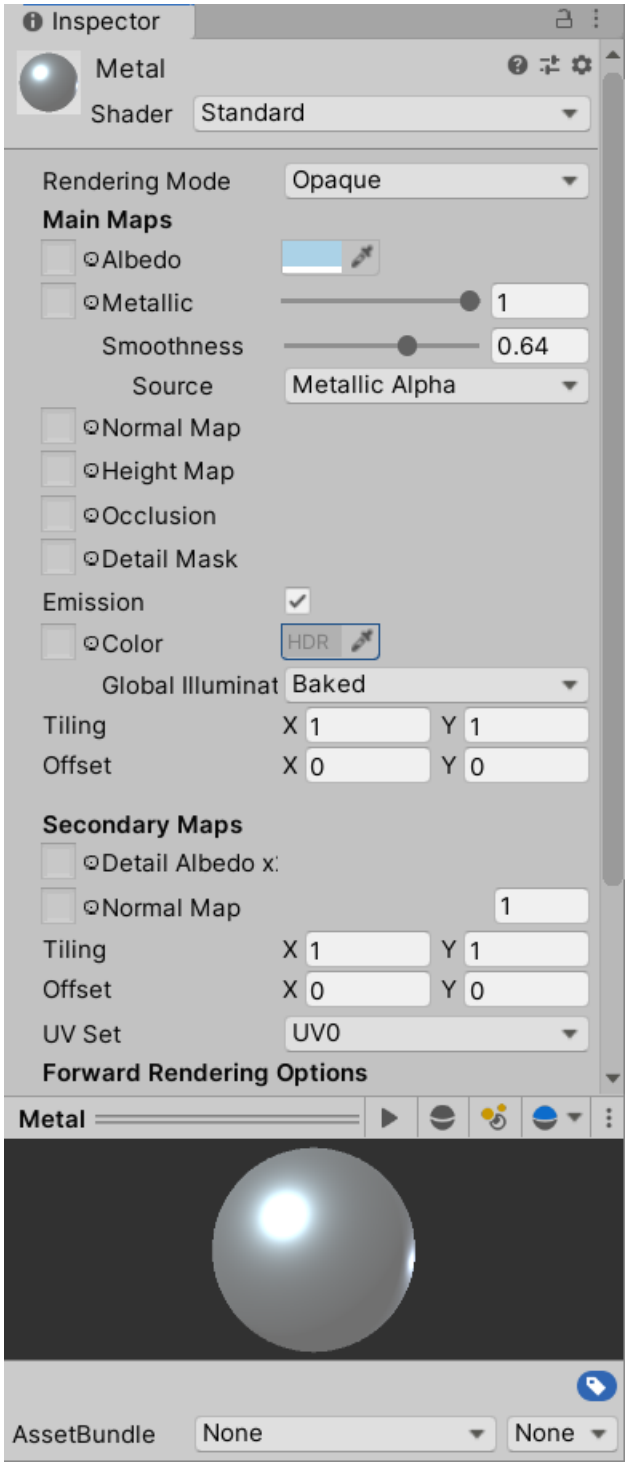


Slika 8: Rigid body komponenta



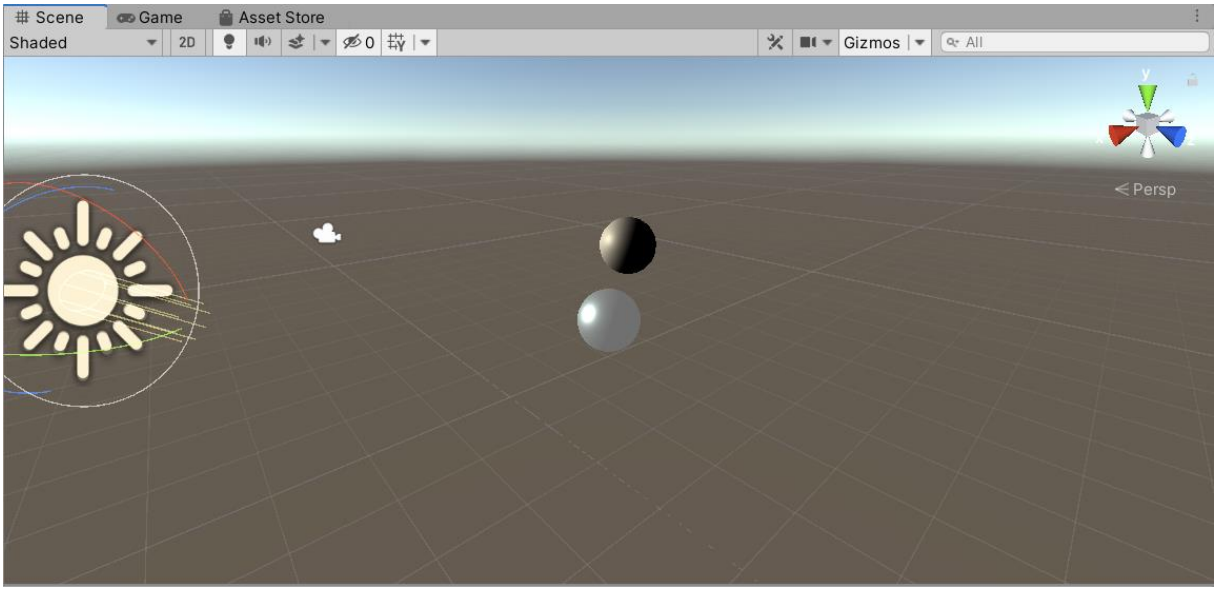
Slika 9: Sudar dvije kugle

Kako bi nekom objektu dodijelili materijal, potrebno ga je prvo napraviti. Pozicioniramo se u mapu assets, unutar project prozora, kliknemo desni klik miša te odaberemo opciju **Create>Material**. Nakon toga dodijelimo ime našem materijalu, te ga nakon toga uređujemo njegove postavke u novootvorenom prozoru[13].



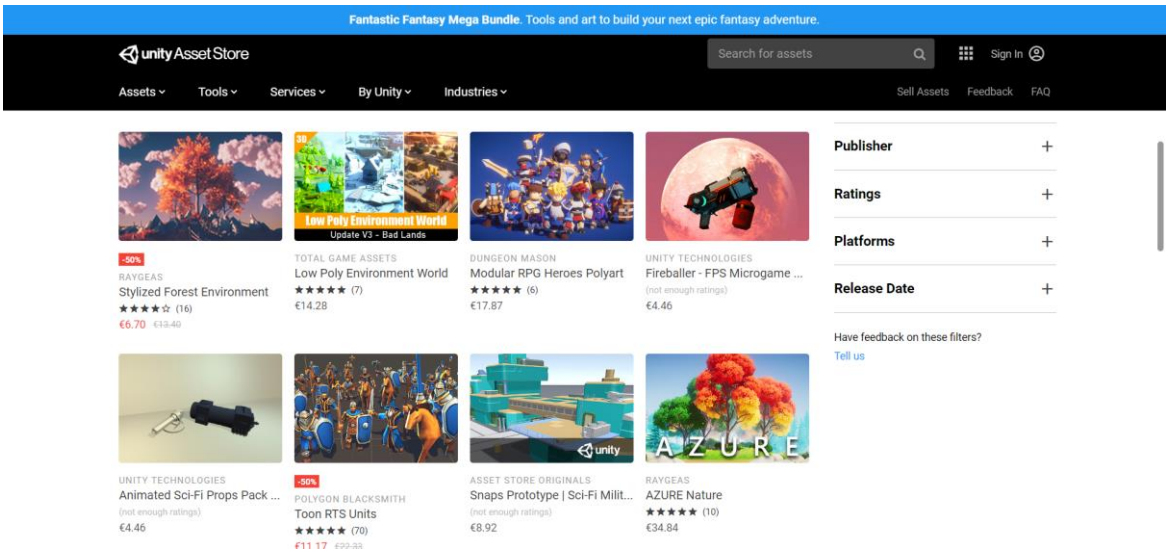
Slika 10: Stvaranje materijala

Kada smo kreirali željeni materijal možemo ga dodati u naš objekt tako da ga povučemo unutar inpector prozora ili uz pomoć opcije „Add component“.



Slika 11: Kugla nakon dodavanja materijala

Naravno ako ne želimo sami izrađivati modele objekata, možemo pronaći već napravljene modele drugih korisnika koje možemo preuzeti besplatno ili kupiti unutar „Asset store“.



Slika 12: Asset store

Na sličan način objektima, kroz opciju „Add component“, možemo dodavati zvukove, efekte, događaje, video sadržaje, fiziku, navigaciju i slično, ali jedna od najvažnijih komponenti za nas je skripta. Pomoću skripti ćemo zapravo definirati ponašanje objekata u igri.

1.2.2. Osvjetljenje

Unity podržava 3 vrste osvjetljenja koje različito utječu na teksture i performanse, pa tako imamo osvjetljenje u realnom vremenu (realtime light), takozvano „baked“ osvjetljenje i kombinacija tih osvjetljenja.

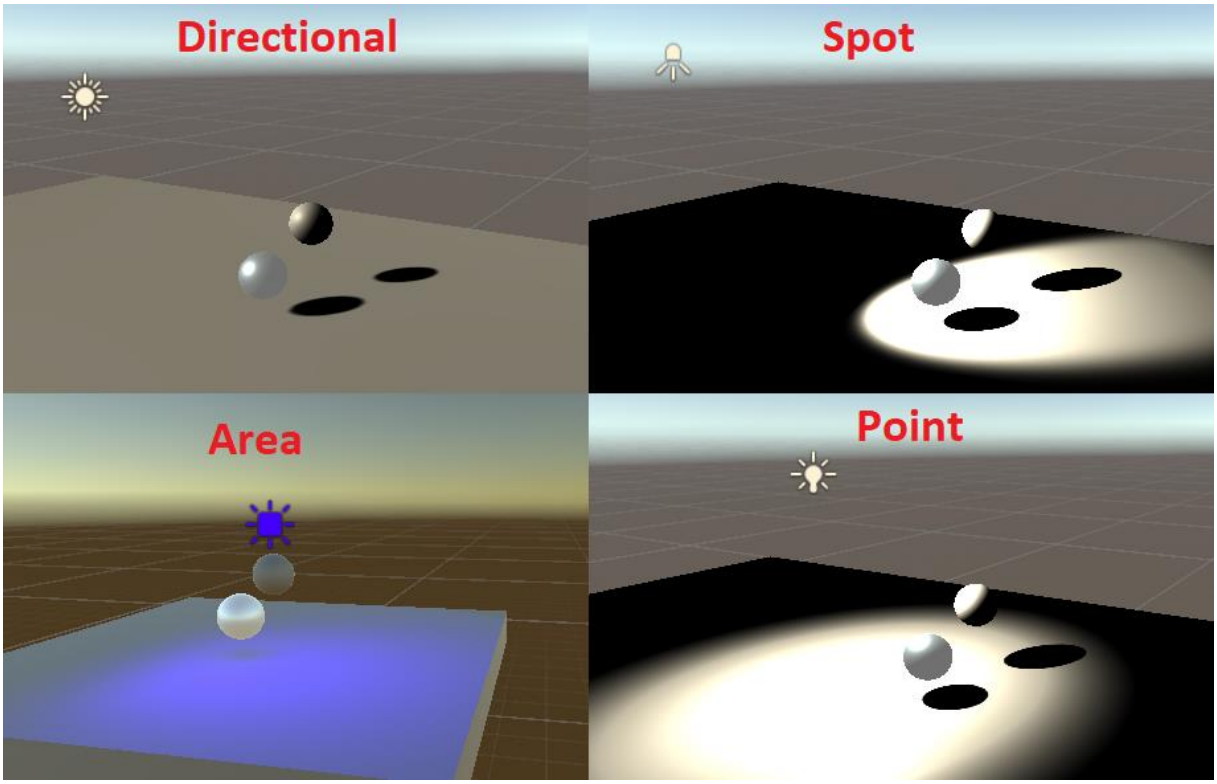
Realtime osvjetljenje je vrsta osvjetljenja koja se osvježava prilikom svakog osvježavanja slike, pa je moguće pomicati objekte, a da se njihova sjena mijenja ovisno o njihovom položaju. Nedostatak ove vrste osvjetljenja je taj da objekti nemaju refleksiju boja, što znači da su sjene potpuno crne jer samo izvor svjetla utječe na njih, pa zbog toga scene mogu izgledati nerealistično. Za ovu metodu je potrebno više procesorskih kalkulacija ovisno o veličini i kompleksnosti scene[21].

Baked osvjetljenje je vrsta osvjetljenja koje je unaprijed izračunato. To znači da se uopće ne osvježava, što dovodi do toga da jednom izračunate sjene na objektu ostaju na onoj poziciji na kojoj su inicijalno bile izračunate. Prednost ove vrste osvjetljenja je ta da objekti zapravo imaju refleksiju što dovodi do indirektnog osvjetljenja i realističnijih sjena[22]. Ova metoda osvjetljenja je korisna kod stacionarnih objekata kojima osvježavanje osvjetljenja nije potrebno.

Mixed osvjetljenje je kombinacija prethodne dvije vrste osvjetljenja, u smislu da uzima dobre strane obje metode i kombinira ih. Ova metoda omogućava osvjetljenje u realnom vremenu s tim da objekti imaju refleksiju, te se time dobiva najrealističnije osvjetljenje. Postoje različite opcije miješanog osvjetljenja koje variraju u performansama i krajnjem izgledu. Kako ova metoda kombinira sve dobre strane prethodne dvije metode, tako zapravo kombinira i neke njihove loše strane, pa je tako za ovu metodu potrebno više procesorskih kalkulacija od realtime metode i više memorije od baked metode[23].

Unutar Unity alata imamo nekoliko vrsta svjetla koja imaju različita ponašanja i primjene. Vrste svjetla koje postoje su point, spot, directional i area. **Point** svjetlo je vrsta svjetla koje se nalazi u jednoj točki u prostoru i emitira svjetlost u svim smjerovima, a intenzitet svjetla se smanjuje s udaljenošću[24]. Ovu vrstu svjetla možemo koristiti kao simulaciju svijeće, lampe i slično, pošto se ponaša veoma slično pravom svjetlu. **Spot** svjetlo je vrsta svjetla koja se također nalazi u jednoj točki u prostoru, ali za razliku od point svjetla ono emitira svjetlost u jednom smjeru, u obliku stošca[24]. Ovu vrstu svjetla možemo koristiti, na primjer, za simuliranje svjetla na autu. **Directional** svjetlo, za razliku od prijašnja dva, nema fiksnu točku na kojoj se nalazi, odnosno nije bitno gdje se nalazi u sceni. Svojstvo ovog svjetla je da se ponaša kao neki izvor svjetlosti koji je neograničeno daleko, a emitira svjetlost samo u jednom smjeru. Intenzitet ove vrste svjetla je uvijek isti pa je možemo koristiti za simulaciju

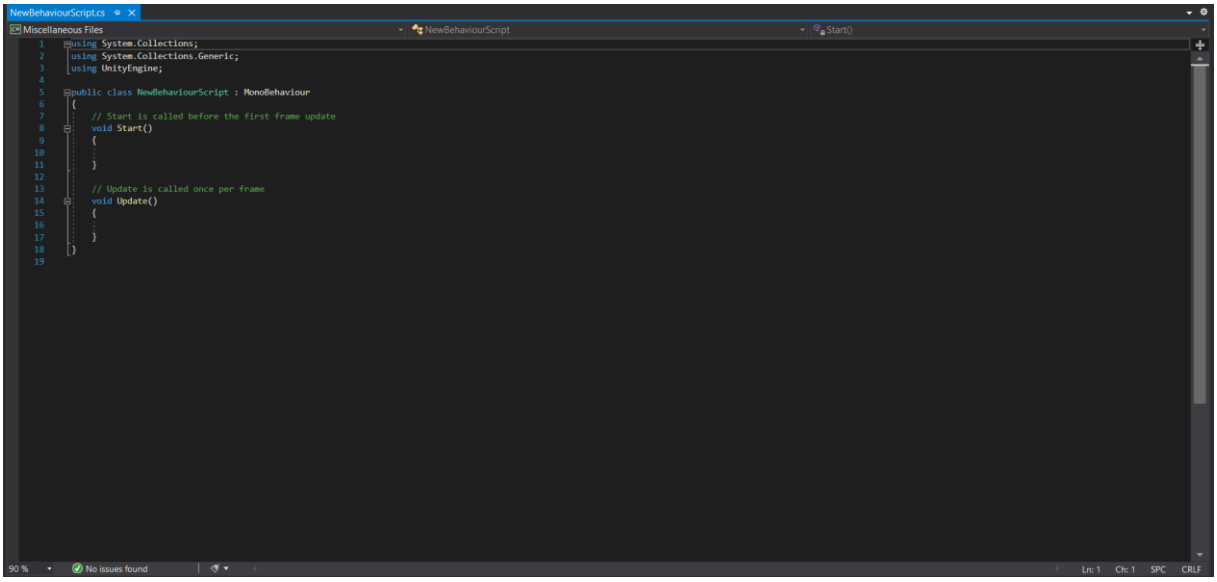
sunca i slično[24]. **Area** svjetlo je vrsta svjetla koja je definirana četverokutom u prostoru iz kojeg se svjetlost emitira u svim smjerovima jednako, ali samo sa jedne strane četverokuta. Pošto ova vrsta svjetla emitira svjetlost iz više različitih izvora na četverokutu, rezultat je glađa i realističnija osvijetljenost. Iz tog istog razloga, ova vrsta osvjetljenja stavlja velik napor na procesor, pa stoga nije dostupna kao realtime već samo kao baked vrsta osvjetljenja[24].



Slika 13: Vrste svjetla

1.2.3.Skriptiranje

Za pisanje skripti unutar Unity alata koristi se porgramski jezik C#. Kako bi kreirali skriptu, pozicioniramo se unutar mape „assets“, kliknemo desni klik i odeberemo opciju **Create>C# Script**. Nakon toga dodjeljujemo ime toj skripti, te ju otvaramo u editoru. Zadani editor koda je Visual Studio, ali to se može promijeniti u postavkama alata. Kada otvorimo skriptu u editoru, dobijemo već napravljeni kostur programa koji možemo vidjeti na slici ispod[14].



Slika 14: Kostur skripte

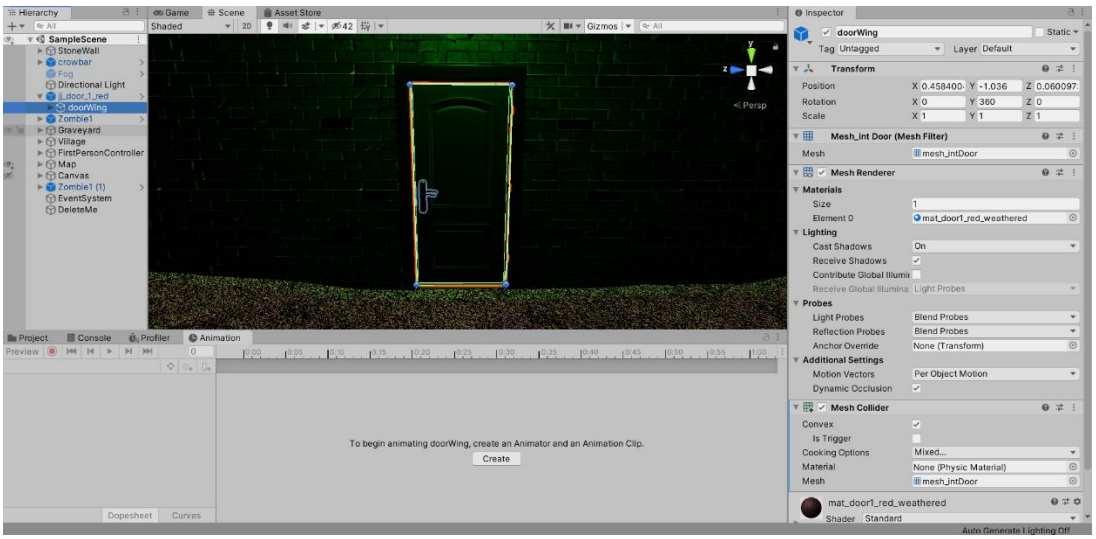
Unutar metode *Start()* stavljamo kod koji će se izvršiti na početku samo jednom prije izvršavanja ostatka koda, dok unutar metode *Update()* stavljamo kod koji će se aktivirati za svako ažuriranje slike, što nam je pogodno za pisanje koda za kretnje ili neke akcije koje čekaju određeni okidač[14].

Ako unutar skripte definiramo javnu globalnu varijablu (public), ona će nam se priakzati unutar inspector prozora. Ovako definirane varijable je moguće mijenjati unutar editora, a promjene će biti prihvaćene i dok je igra pokrenuta[15]. Ovo nam je korisno ako želimo u stvarnom vremenu testirati kako će se igra ponašati za određenu vrijednost neke varijable. Kroz skriptu također možemo i dodavati komponente bez da ih dodajemo unutar editora[16]. Ova funkcionalnost nam može poslužiti ako u početku ne želimo da je neka komponenta dodana željenom objektu, nego ju želimo naknadno dodati na temelju određenog okidača i slično. Okidače, kako je već rečeno, dodajemo u metodu *Update()*, ali ako želimo točnije rezultate, stavljamo ih u *FixedUpdate()* metodu. Razlog zašto se koristi *FixedUpdate()* metoda je taj da se ažuriranje fizike i ažuriranje slike ne odvija istom frekvencijom, pa nam to može stvarati probleme. Nadalje, funkcija *LateUpdate()* omogućava nam izvršavanje naredbi koje bi htjeli da se pokrenu nakon *Update()* i *FixedUpdate()* funkcija. Unuter funkcije *OnGUI()* pišemo kod koji će rukovati funkcijama grafičkog sučelja, kao na primjer glavni izbornik. Osim toga imamo i funkcije koje reagiraju na kolizije. Te funkcije se pozivaju kada dođe do kontakta (*OnCollisionEnter()*), kada traje kontakt (*OnCollisionStay()*) i kada se kontakt prekine (*OnCollisionExit()*). Ako je *collider* objekta postavljen kao okidač, tada se koriste funkcije *OnTriggerEnter()*, *OnTriggerStay()* i *OnTriggerExit()*[17].

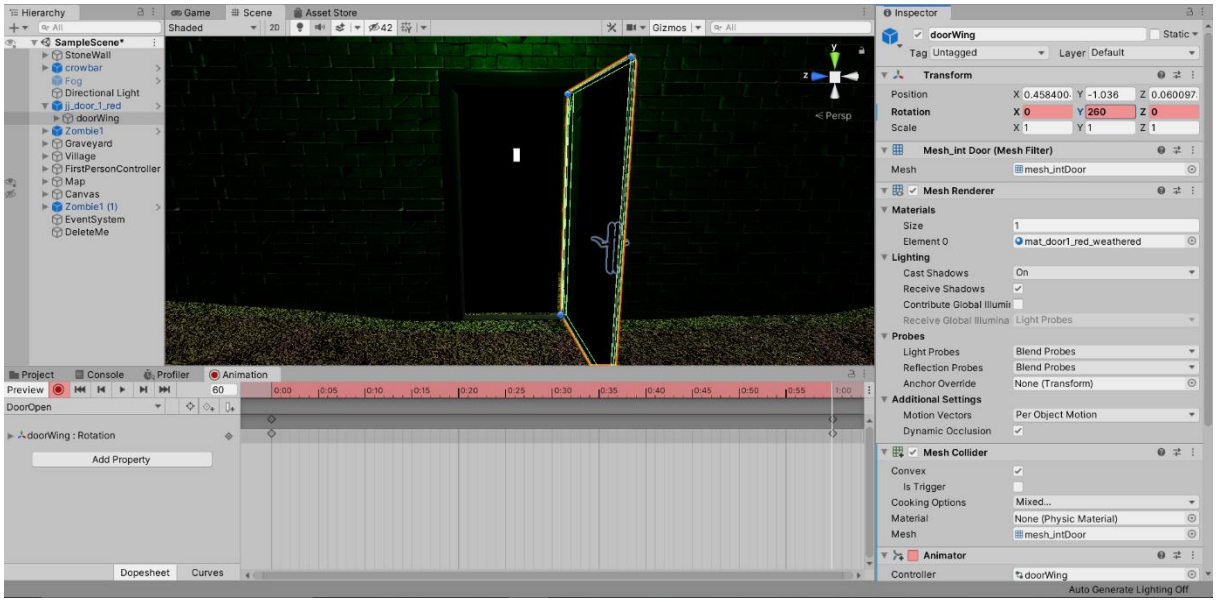
Kroz skripte također možemo stvarati i brisati objekte. To činimo uz pomoć funkcije *Instantiate()*[18]. Ovaj način dodavanja i brisanja objekata nam je koristan ako želimo dinamički stvarati objekte poput neprijatelja za koje ne želimo da budu u sceni odmah pri pokretanju igre.

1.2.4. Animacije

U Unity alatu imamo mogućnost izrade animacija nad našim objektima, a za to koristimo prozor Animator. Unity animacije radi na temelju animacijskih isječaka (animation clips) koji većinom dolaze iz vanjskih izvora. Ti isječci mogu biti kreirani unutar animatora, u nekom drugom vanjskom softveru ili snimanjem pokreta u studiju. Ove animacije određuju ponašanje naših objekata tako što utječu na njihovu poziciju, rotaciju i slično. Nakon što uvezemo (ili kreiramo) animacije, uz pomoć animatora određujemo kako i kada će se one pokretati. To definiramo unutar kontrolera animatora (animator controller), koji ima oblik dijagrama toka, a omogućava nam spajanje raznih isječaka uz određene uvjete, na primjer prijelaz animacije hodanja u animaciju trčanja[20]. Uz pomoć ove opcije mi dajemo realističnost našoj igri. Umjesto da su objekti stacionarni oni imaju nekakve kretnje koje manje ili više daju realizam igri, pa tako umjesto da naš objekt (igrač, neprijatelji, …) „klizi“ po površini on ima animaciju hodanja ili trčanja koja se aktivira prilikom kretanja tog lika. Naravno, ako ne želimo sami izrađivati objekt, možemo preuzeti ili kupiti objekte koji već imaju definirane animacije unutar asset trgovine. Za izradu animacija možemo koristiti i prozor Animation u kojem definiramo položaj, rotaciju, veličinu i slične postavke objekta u određenom vremenskom trenutku. Nakon što imamo definirano više „položaja“ u vremenu, Unity će napraviti glatke prijelaze između tih prijelaza te tako stvoriti animaciju[37].

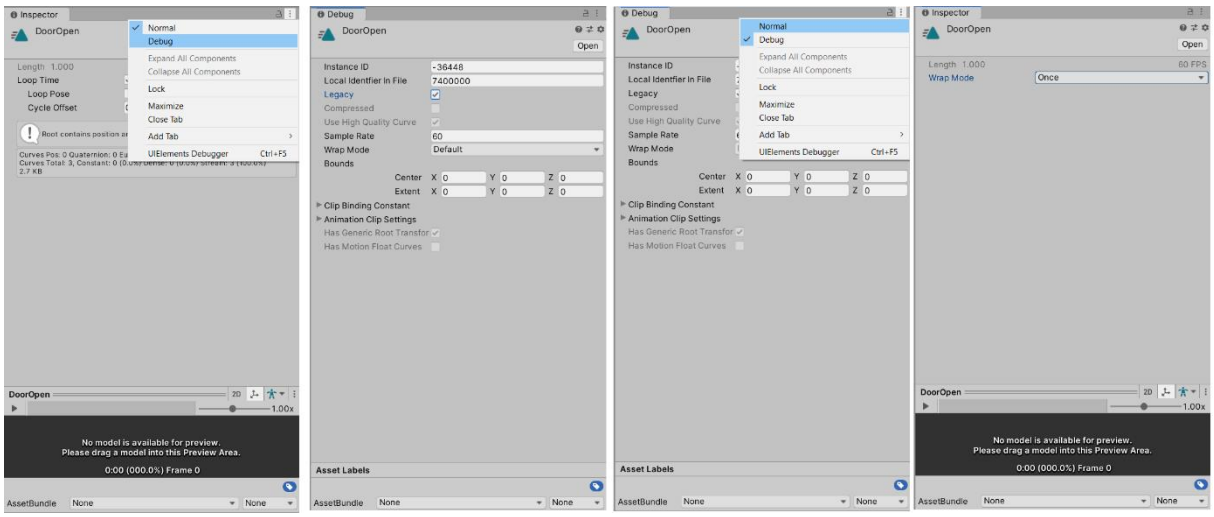


Slika 15: Prozor Animation

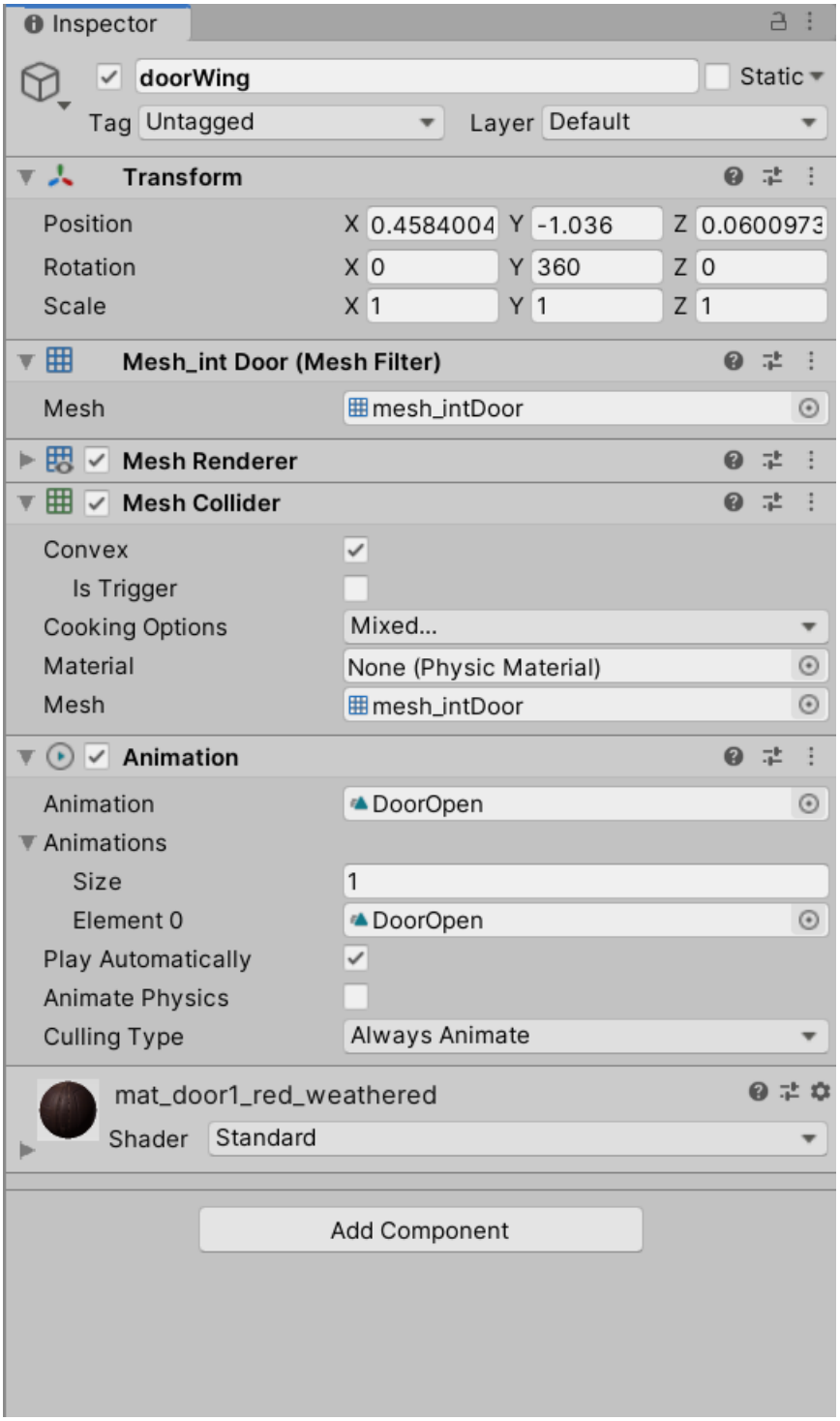


Slika 16:Stvaranje animacije

Kao što vidimo na slici 16, u točki 0 (0 sekundi) smo postavili rotaciju vrata po Y osi na 0 (vrata potpuno zatvorena), a u točki 60 (1 sekunda) postavili smo rotaciju na 260 (vrata potpuno otvorena). Time smo stvorili animaciju koju onda dodajemo na objekt vrata, nakon što definiramo potrebne postavke animacije kako bi se izvršila samo jednom. To postižemo tako da se u inspector prozoru postavimo u debug mod, ukjučimo opciju „Legacy“, vratimo se u normal mod i postavimo „Wrap Mode“ na „Once“[37].



Slika 17: Promjena postavki animacije



Slika 18: Umetanje komponente Animation

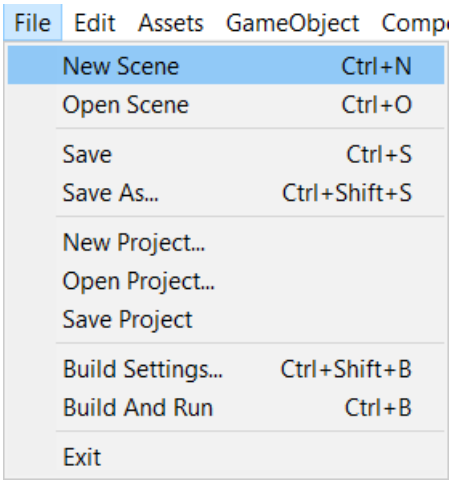
1.2.5.Zvuk

Unity nam omogućava uvoz zvukova u MP3, Ogg, WAV i AIFF formatima, te nam omogućava dodavanje tih zvukova određenim objektima. Ti zvukovi putuju od objekta kojem su pridodani do komponente za osluškivanje zvuka (Audio Listener), što stvara iluziju da zvuk dolazi iz smjera tog objekta. Slušač zvuka se najčešće stavlja na glavnu kameru kako bi zvuk dopirao do našeg lika. Unity također ima i alat za manipulaciju zvuka (Audio Mixer) koji omogućava ručno uređivanje zvuka i dodavanje efekata[25]. U razvoju računalnih igara, zvuk se često koristi za postavljanje atmosfere, pogotovo u horor igrama.

Slika 19: Audio Source

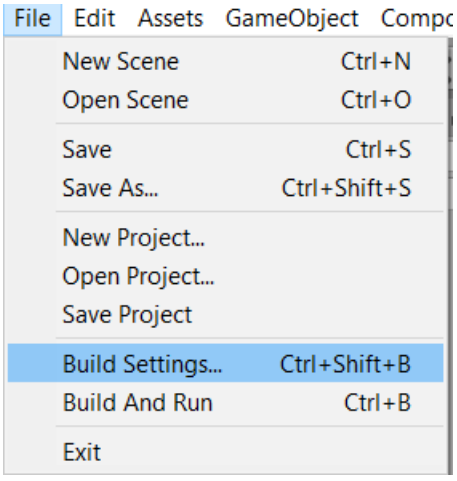
1.2.6.Scene

U Unity, dodavanje novih scena je vrlo jednostavno. U izborniku odaberemo *File > New scene* pri čemu nam se otvara nova scena u kojoj možemo izraditi novi level ili stacionarne scene kao „game over“ scena i slično. Scenama možemo upravljati kroz skripte koristeći biblioteku UnityEngine.SceneManagement i funkciju SceneManager.LoadScene() [44].

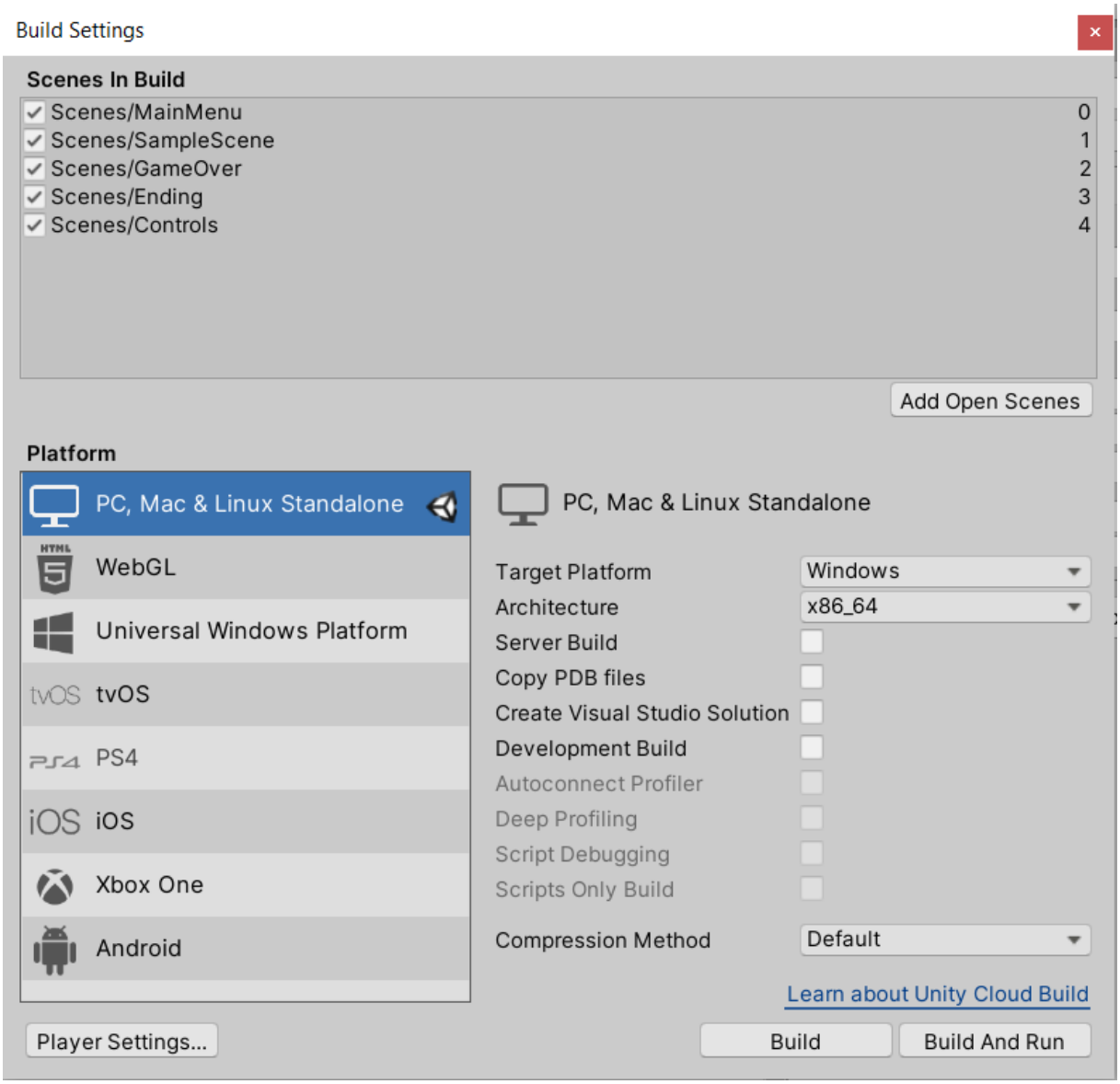


Slika 20: Nova scena

Redoslijed scena slažemo u prozoru „Build Settings“, gdje ih možemo dodavati i raspoređivati kako želimo, a u skriptama ih referenciramo po indeksu [44].



Slika 21: Build Settings



Slika 22: Build settings prozor

2. Horor igre preživljavanja

Horor igra preživljavanja (survival horror) je žanr video igara koji koristi elemente horor igara, ali sa naglaskom na oskudnost resursa[26]. Prva igra koja je predstavljena kao horor preživljavanja 1996. je bila Resident Evil koja je danas među najpoznatijim horor igrama. Ono što definira horor preživljavanja su zagonetke, korištenje često malog inventara i borba sa neprijateljima. Borba sa neprijateljima nije u svakoj igri jednaka. U nekima borba jednostavno ne postoji, nego se više fokusira na izbjegavanje neprijatelja[27]. Horor igre najčešće kreću polako, te s vremenom stvaraju sve veću napetost dok ne dođemo do točke kada priča dođe do vrhunca. U tom trenutku često dolazi do nekog velikog obračuna ili nekog velikog preokreta u priči[28].

Dakle, horor preživljavanja je zapravo vrsta horor igre koja nas prisiljava na pametno iskorištavanje resursa i snalaženje u okolini, a sve to uz konstantni ili periodični element straha.

2.1. Usporedba nekoliko naslova

Kako bi bolje razumjeli ovaj žanr video igara, napraviti ćemo kratak opis nekoliko naslova, te njihovu usporedbu.

Resident Evil, kao što je već bilo rečeno, je prva igra koja se smatra horor igrom preživljavanja. Priča je smještena u napuštenoj kući, koja je zapravo bila postrojenje za testiranje virusa, kojeg jedan od članova našeg tima želi pretvoriti u oružje. Kroz cijelu igru borimo se sa mutiranim stvorenjima, dok se na kraju ne pokrene samouništenje cijelog laboratorija, a mi kao igrač pobjegnemo helikopterom[29]. Ono što Resident Evil implementira, da potakne igrače na planiranje unaprijed, je to da zombiji koje ubijemo, nakon nekog vremena „ožive“ i postanu puno teži za rješavati kasnije[30]. Sve se to naravno odvija pod malim osvjetljenjem i uz prisutnost jezivih zvukova, što je ikonično za horor igre općenito.

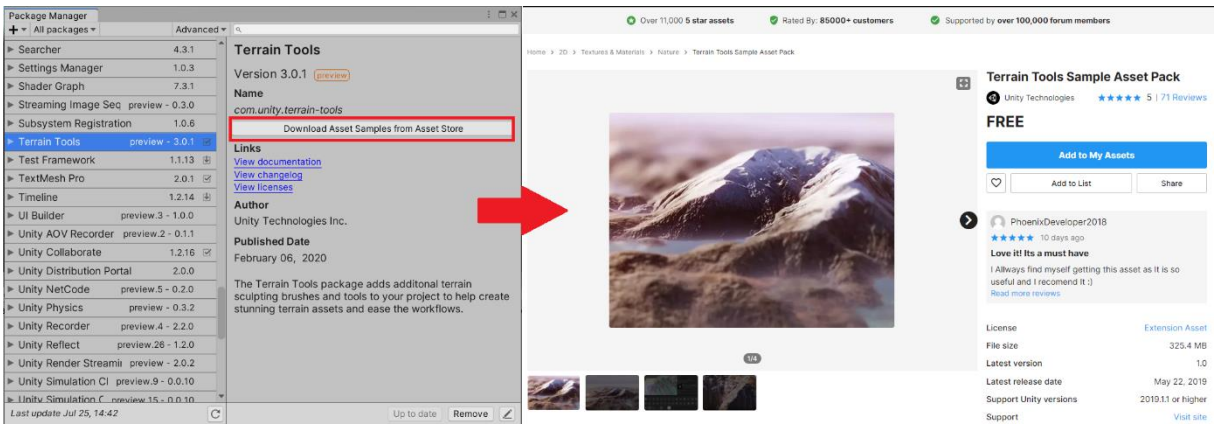
Još jedan naslov koji je značajno utjecao na popularnost horor preživljavanja je Silent Hill. Ukratko priča ove igre je da otac i kćer odlaze u grad po imena Silent Hill. Kćer Cheryl nestaje nakon automobilske nesreće, nastale zbog žene koja je prelazila cestu, nakon čega otac Harry kreće u potragu za njom. Dok traje potraga, upoznaje policajku koja mu daje pištolj, nakon čega nastavlja potragu. Priča prati Harryja kroz napušteni grad, boreći se sa čudovištima i prolazeći kroz razne scenarije pune uznemirujućih prizora. S vremenom dolazimo do kraja priče koji varira ovisno o tome kako smo od početka igrali. Magla je velik dio ove igre, dijelom zbog ograničenja tadašnje konzole, ali i dijelom radi stvaranja efekta i postavljanja raspoloženja[31].

Dakle, iz ova dva naslova možemo dobiti grubu ideju kako bi naša igra zapravo trebala izgledati, te što sve definira ovaj stil i vrstu horor igara. Generalno, potrebno je uz pomoć raznih vizualnih i zvukovnih elemenata postaviti raspoloženje, te igri dodati element straha i napetosti. Zatim ne dati igraču previše slobode, odnosno oduzeti mu osjećaj sigurnosti u već napetim situacijama. Potrebno je dodati zagonetke u dijelove igre, kako bi igraču povratili taj osjećaj sigurnosti, jer se u ove dvije igre vidi da oscilacija količine napetosti pridonosi sveukupnom efektu.

3. Izrada igre

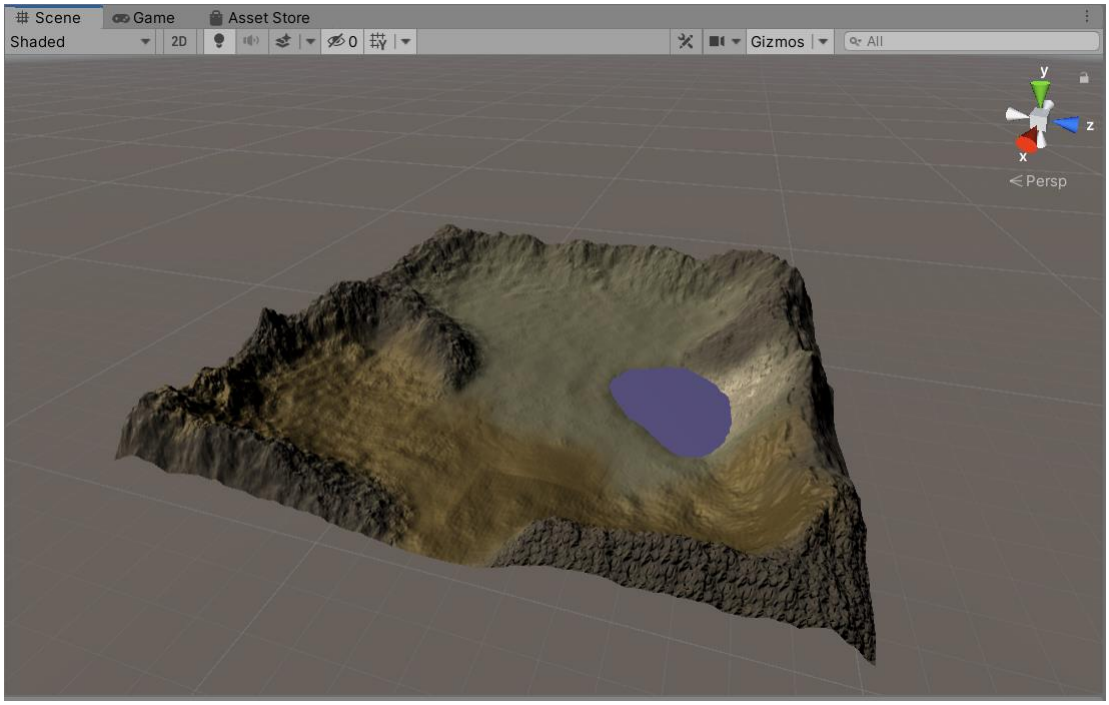
3.1. Izrada mape

Sada kada znamo neke osnove možemo početi sa izradom naše igre. Za početak ćemo izraditi mapu ne kojoj će igra biti smještena. Unutar package manager-a preuzeti ćemo paket „Terrain tools“ koji će nam olakšati posao izrade mape, a nakon što se paket preuzme package manager nas vodi u asset store gdje preuzimamo asset u naš projekt[32].



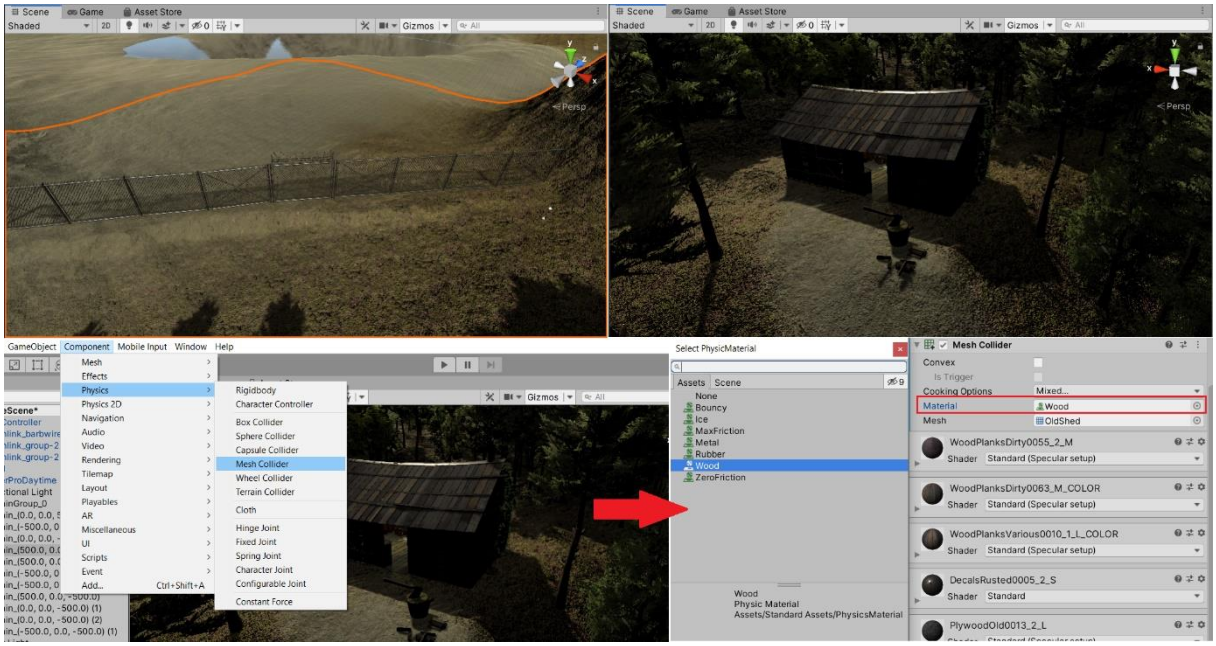
Slika 23: Instalacija paketa "Terrain tools"

Nakon nekoliko sati rada imamo grubi oblik mape na koji je sada potrebno dodati detalje, mjesta, ograničenja i slično.

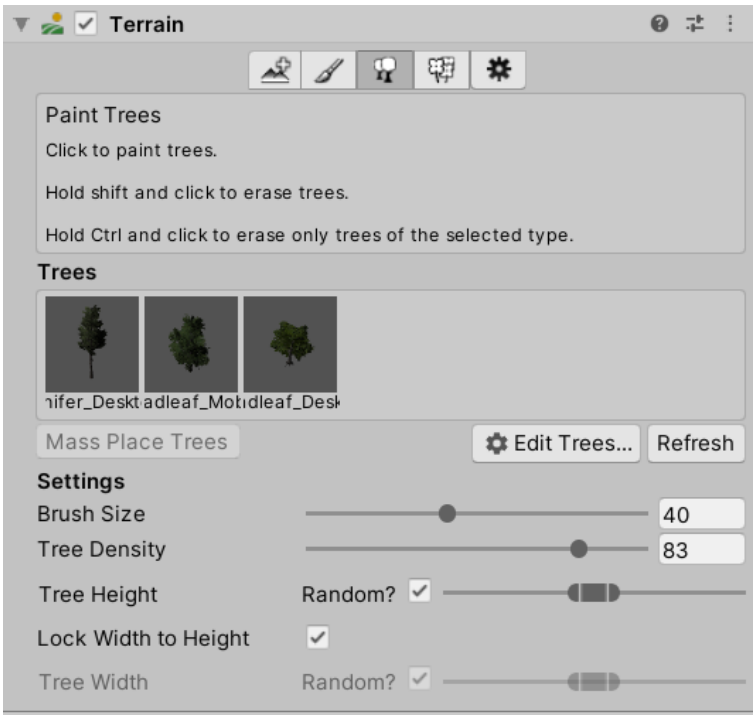


Slika 24: Grubi oblik mape

Kako bi uredili mapu i dodali neke tražene detalje, možemo ih preuzeti iz asset store prozora. Ti gotovi modeli mogu se samo umetnuti u našu scenu i podesiti našim potrebama. Primjer dodavanja gotovih modela možemo vidjeti na slici ispod. Ti modeli nemaju takozvani collider pa igrač može kroz njih prolaziti, zato im dodajemo mesh collider koji stvara „čvrsta“ područna na rubovima modela. Također možemo preuzeti i modele drveća i trave koje nanosimo na mapu uz pomoć ugrađenih kistova.

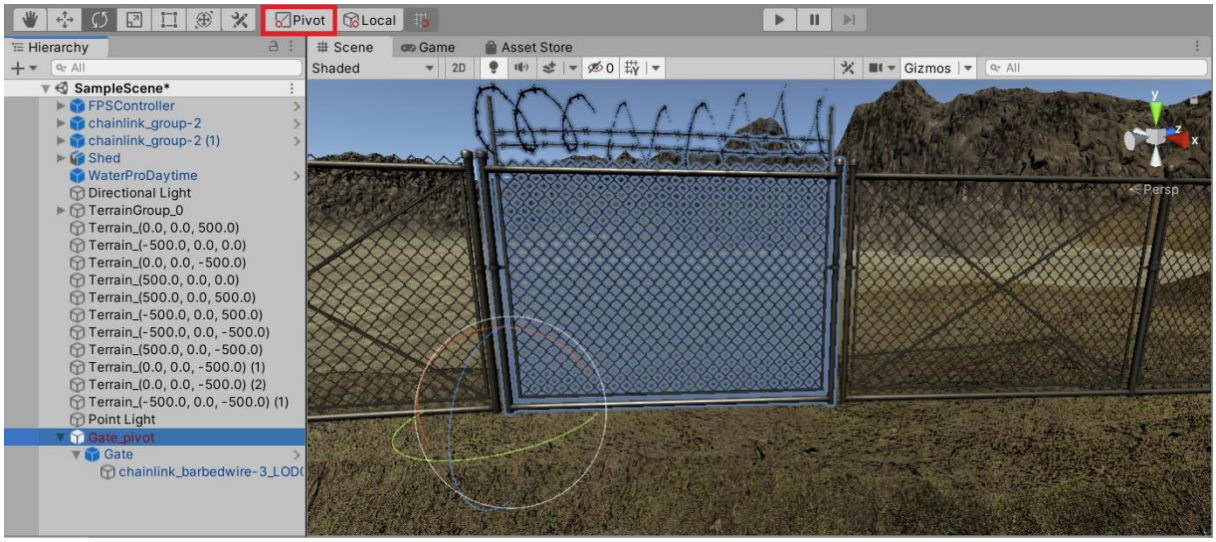


Slika 25: Dodavanje mesh collidera u uvezene modele



Slika 26: Postavke za dodavanje drveća

Kako bi kasnije mogli otvarati i zatvarati ogradu, moramo postaviti novi pivot na model pošto je trenutni u sredini modela. To ćemo napraviti tako da ćemo stvoriti novi prazan objekt koji ćemo pozicionirati na željeno mjesto (kut ograde), odabrati ćemo opciju pivot iz trake u gornjem dijelu prozora, te umetnuti naš objekt ograde unutar tog novog objekta. Tom novom objektu ćemo dodijeliti ime „Gate_pivot“ kako bi smo ga lakše pronašli kasnije.



Slika 27: Stvaranje novog pivota za ogradu

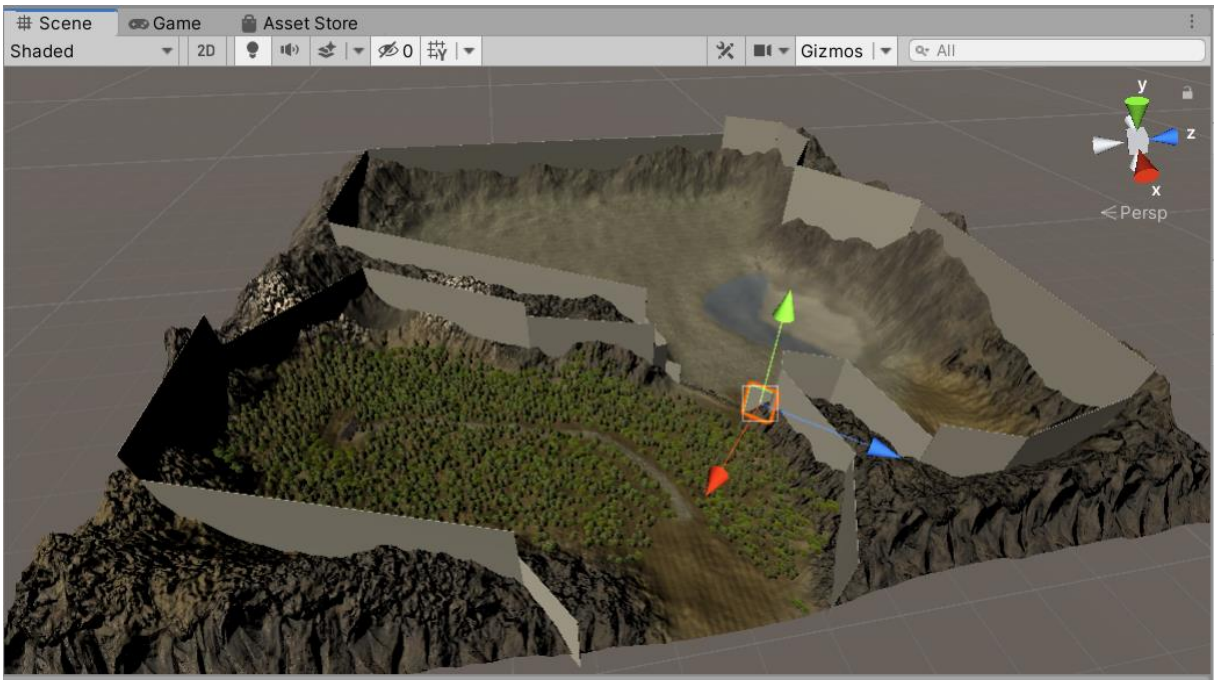
Ogradu ćemo otvarati koristeći skriptu „OpenGate“, koja će provjeravati postoji li pila u sceni, te ako ne postoji (pokupljena je), omogućiti otvaranje ograde rotiranjem po dodanom pivotu. Rotiranje postižemo stvaranjem animacije u prozoru animation. Po istom principu otvaramo i sva ostala vrata ili objekte koji se otvaraju rotacijom[38].

OpenGate

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class OpenGate : MonoBehaviour
{
    public float Distance;
    public GameObject Fog;
    public GameObject DisplayActionKey;
    public GameObject DisplayText;
    public GameObject ActionCrosshair;
    public GameObject Gate;
    public AudioClip GateSound;
    public GameObject GateLock;
    public GameObject Saw;
    private Text text;
    private AudioSource audioSource;

    private void Start()
    {
        audioSource = GetComponent<AudioSource>();
    }
    void Update()
    {
        Distance = DetectDistance.TargetDistance;
    }

    private void OnMouseOver()
    {
        if(Distance < 10)
        {
            DisplayActionKey.SetActive(true);
            DisplayText.SetActive(true);
            ActionCrosshair.SetActive(true);
            if (Saw.activeInHierarchy)
```

```
            {
                text = DisplayText.GetComponent<Text>();
                text.text = "It appears to be locked";
            }
            else
            {
                text = DisplayText.GetComponent<Text>();
                text.text = "Use saw";
            }
        }
        if (Distance >= 10)
        {
            DisplayActionKey.SetActive(false);
            DisplayText.SetActive(false);
            ActionCrosshair.SetActive(false);
        }
        if (Input.GetKey(KeyCode.E) && Distance < 10)
        {
            if (!Saw.activeInHierarchy)
            {
                this.GetComponent<BoxCollider>().enabled = false;
                DisplayActionKey.SetActive(false);
                DisplayText.SetActive(false);
                ActionCrosshair.SetActive(false);
                Gate.GetComponent<Animation>().Play("GateAnimation");
                Fog.SetActive(true);
                audioSource.clip = GateSound;
                audioSource.Play();
                GateLock.SetActive(false);
            }
        }
    }

    private void OnMouseExit()
    {
        DisplayActionKey.SetActive(false);
        DisplayText.SetActive(false);
        ActionCrosshair.SetActive(false);
    }
}
```

Potrebno je dodati i nevidljive zidove kako igrač ne bi mogao izlaziti izvan mape i kako ne bi mogao prelaziti u dijelove mape u koje još uvijek ne smije ići.

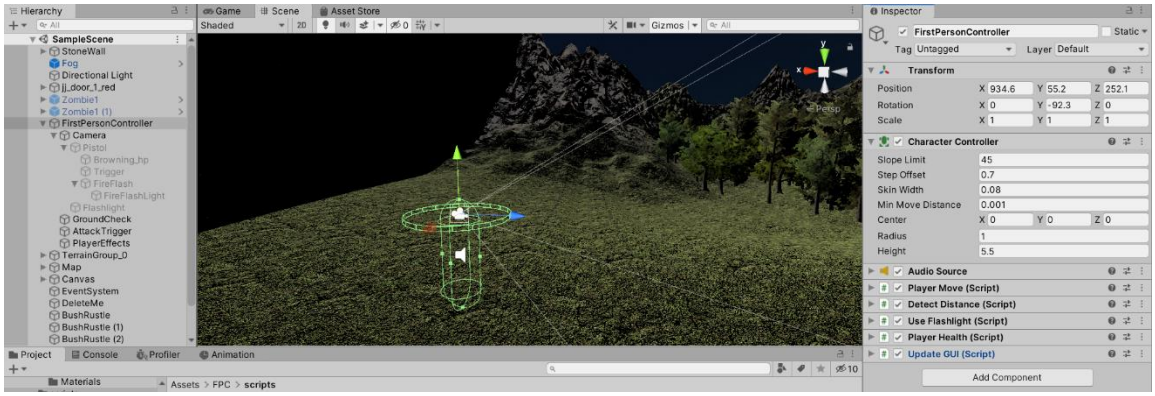


Slika 28: Granice unutar kojih je dozvoljeno kretanje

3.2. Izrada „First Person“ kontrolera

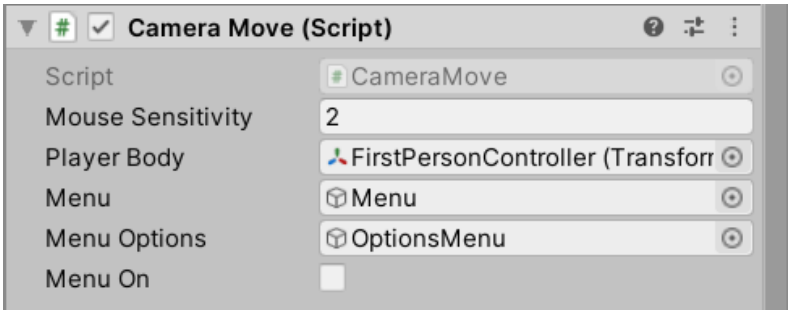
U alatu Unity moguće je na dva načina stvoriti „igrača“. Igrač može biti stvoren kao „rigid body“ koji ima već ugrađenu gravitaciju, trenje i interakciju sa objektima, ili kao „character controller“ koji ima bolju interakciju sa stepenicama i kosinama, te dobru interakciju sa zidovima, u smislu da se igrač ne zaglavljuje u njima[33]. U našoj implementaciji koristiti ćemo „character controller“.

Za početak u scenu ćemo dodati novi prazan objekt i nazvati ga „FirstPersonController“, dodati mu „character controller“ i postaviti dimenzije koje najbolje pašu našim potrebama. Na taj kontroler ćemo dodati kameru, element za zvukove, te skripte koje će nam omogućiti kretanje, pomicanje kamere, aktivaciju zvukova i slično.



Slika 29: First person controller

Prva skripta koju ćemo dodati je „CameraMove“ u kojoj ćemo definirati logiku pomicanja kamere uz pomoć miša. Za horizontalno pomicanje rotirati ćemo cijelog lika, a za vertikalno pomicanje rotirati ćemo samo kameru. Vertikalno pomicanje ćemo također ograničiti (dodati vrat) kako igrač ne bi mogao okrenuti kameru (glavu) naopako. Ovu skriptu dodajemo kao komponentu u kameru na našem liku, te u „Player Body“ dodajemo čitav element „FristPersonController“.



Slika 30: CameraMove skripta kao komponenta

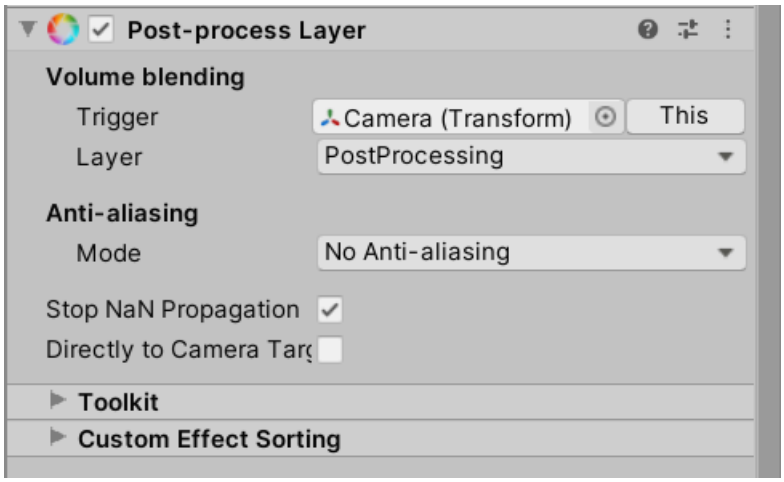
CameraMove:

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CameraMove : MonoBehaviour
{
    public float mouseSensitivity = 2f; // Osjetljivost miša
    public Transform playerBody;

    public GameObject menu;
    public GameObject menuOptions;

    public bool menuOn = false;

    float xRotation = 0f;

    void Start()
    {
        Cursor.lockState = CursorLockMode.Locked;
        Cursor.visible = false;
    }

    private void Update()
    {
        if (Input.GetKeyDown(KeyCode.Escape))
        {
            ToggleMenu();
        }
    }

    //Uključuje ili isključuje menu
    private void ToggleMenu()
    {
        if (menuOn)
        {
            menuOn = false;
```

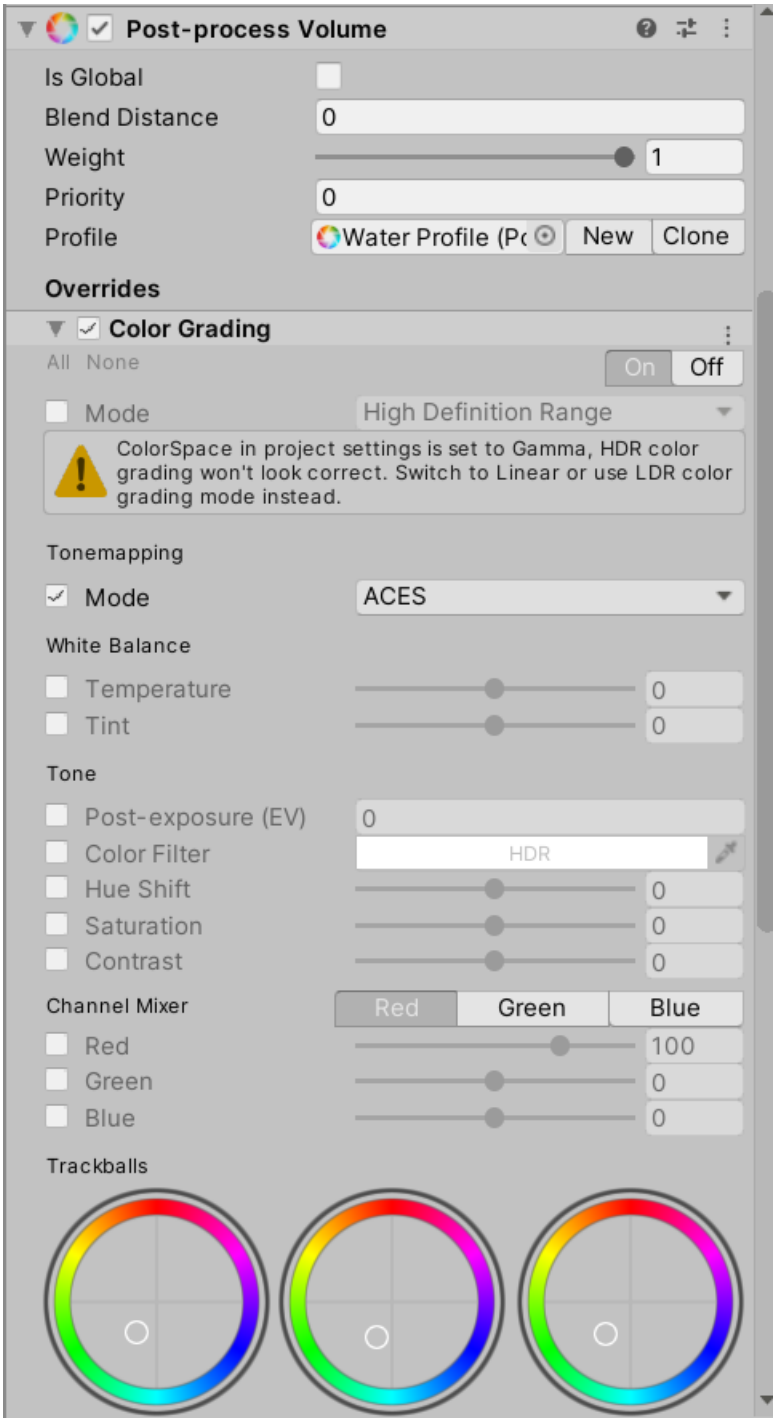
```
            menu.SetActive(false);
            menuOptions.SetActive(false);
            Cursor.lockState = CursorLockMode.Locked;
            Cursor.visible = false;
        }
        else
        {
            menuOn = true;
            menu.SetActive(true);
            Cursor.lockState = CursorLockMode.None;
            Cursor.visible = true;
        }
    }

    //Prebacivanje iz menu u postavke
    public void ToggleOptions()
    {
        if (menu.activeInHierarchy)
        {
            menu.SetActive(false);
            menuOptions.SetActive(true);
        }
        else
        {
            menu.SetActive(true);
            menuOptions.SetActive(false);
        }
    }

    void LateUpdate()
    {
        if (!menuOn)
        {
            // Dohvaća trenutnu rotaciju miša na X osi. "Mouse X" je
predefinirani naziv za X miša u alatu Unity.
            float mouseX = Input.GetAxis("Mouse X") * mouseSensitivity;
            float mouseY = Input.GetAxis("Mouse Y") * mouseSensitivity;
//Radi isto što i prethodna linija samo za Y os.

            xRotation -= mouseY;
```

```
            xRotation = Mathf.Clamp(xRotation, -90f, 90f); // Ograničava
rotaciju tako da se može gledati do ravno gore ili ravno dolje i ne više.


            transform.localRotation = Quaternion.Euler(xRotation, 0f, 0f);
// Rotiramo kameru ovisno o xRotation varijabli.


            playerBody.Rotate(Vector3.up * mouseX); // Rotiramo kontroler
ovisno o mouseX varijabli.
        }
    }
}
```

Na kameru ćemo još dodati i Post-process koji će nam postaviti zamućeni efekt i određenu boju dok se nalazimo u određenom području (u ovom slučaju u vodi). To ćemo dodati tako da na kameru dodamo Post-process Layer, a u trigger u kojem želimo efekt dodamo Post-process Volume [36].



Slika 31: Post-process Layer



Slika 32: Post-process Volume

Sljedeća skripta koju ćemo dodati je „PlayerMove“. Ova skripta omogućiti će nam kretanje po mapi korištenjem tipki W, A, S i D, koje su predefinirane tipke za kretanje u alatu Unity. Svaka tipka ima svoju vrijednost za određenu os: W = 1, S = -1, A = -1, D = 1. Također ćemo dodati i neke druge kontrole kao skakanje, trčanje i slično. Gravitaciju ćemo dodati po uzoru na formulu slobodnog pada kako igrač ne bi padao konstantnom brzinom nego uračunavamo i akceleraciju[33]. Također u ovoj skripti definirati ćemo koje zvukove treba pokrenuti u kojem trenutku, pa ćemo tako imati različit zvuk koraka za različitu podlogu, različit zvuk skakanja za određenu podlogu i zvukove u vodi. Ovu skriptu dodajemo u „FirstPersonController“ kao cjelinu. Formula slobodnog pada[34]:

$$s = \frac{1}{2} g * t^2$$

g –akceleracija gravitacije

t – vrijeme

PlayerMove

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Security.AccessControl;
using System.Threading;
using UnityEngine;
using UnityEngine.Experimental.Rendering;

public class PlayerMove : MonoBehaviour
{
    [SerializeField] private AudioClip[] dirtFootsteps;
    [SerializeField] private AudioClip[] rockFootsteps;
    [SerializeField] private AudioClip[] grassFootsteps;
    [SerializeField] private AudioClip[] sandFootsteps;
    [SerializeField] private AudioClip[] gravelFootsteps;
    [SerializeField] private AudioClip[] waterFootsteps;
    [SerializeField] private AudioClip[] woodFootsteps;
    [SerializeField] private AudioClip dirtJump;
    [SerializeField] private AudioClip rockJump;
    [SerializeField] private AudioClip grassJump;
    [SerializeField] private AudioClip sandJump;
    [SerializeField] private AudioClip gravelJump;
```

```
[SerializeField] private AudioClip waterJump;
[SerializeField] private AudioClip woodJump;
[SerializeField] private AudioClip JumpUp;
[SerializeField] private AudioClip DivingSound;
[SerializeField] private AudioClip SwimmingSound;

private int lastIndex = 0;
private int newIndex = 0;
public int surfaceIndex = 0;
public Terrain terrain;
private TerrainData terrainData;
private Vector3 terrainPos;
private TerrainLayer[] terrainLayers;
private AudioSource audioSource;
private int currentDelay;
private Vector3 lastPosition;
private Terrain[] _terrains;
public CharacterController controller;
public float walkingSpeed = 12f;
public float runningSpeed = 24f;
public float lastGravity;
public float gravity = -9.81f;
public float waterGravity = -300f;
public float jumpHeight = 3f;
public int stepDelay = 20;
public int runDelay = 15;
public Transform groundCheck;
public float groundDistance = 0.4f;
public LayerMask groundMask;

Vector3 velocity;
Vector2 input;
bool inWater;
bool inBuilding;
bool inRuinedBuilding;
public bool isSwimming;
bool isDiving;
bool isJumping;
bool isWalking;
[SerializeField] bool isGrounded;
```

```
    [SerializeField] bool wasGrounded;
    bool isMoving;

    void Start()
    {
        _terrains = Terrain.activeTerrains;

        lastGravity = gravity;
        inWater = false;
        lastPosition = new Vector3(0, 0, 0);
        currentDelay = 0;
        audioSource = GetComponent<AudioSource>();

        isJumping = false;
        wasGrounded = true;

        QualitySettings.vSyncCount = 0;

        PlayerHealth.currentHealth = 20;
    }

    void Update()
    {
        terrain = GetClosestCurrentTerrain(groundCheck.transform.position);
        terrainData = terrain.terrainData;
        terrainPos = terrain.transform.position;
        surfaceIndex = GetMainTexture(transform.position);
        terrainLayers = terrain.terrainData.terrainLayers;

        wasGrounded = isGrounded;
        isGrounded = Physics.CheckSphere(groundCheck.position,
groundDistance, groundMask);

        if (!wasGrounded && isGrounded && !isDiving)
        {
            PlayLandingSound();
            velocity.y = -2f;
            isJumping = false;
            wasGrounded = true;
        }
```

```
    if (wasGrounded && !isGrounded && !isDiving)
    {
        audioSource.clip = JumpUp;
        audioSource.Play();
    }

    if (!isGrounded && !isJumping && wasGrounded)
    {
        velocity.y = -2f;
    }
}

//Kada igrač dotakne površinu uključuje zvuk skoka ovisno o podlozi
private void PlayLandingSound()
{
    string textureName = GetTextureName();

    if (inWater)
    {
        audioSource.clip = waterJump;
        audioSource.Play();
        return;
    }

    if (inBuilding)
    {
        audioSource.clip = woodJump;
        audioSource.Play();
        return;
    }

    switch (textureName)
    {
        case "moss":
            audioSource.clip = grassJump;
            audioSource.Play();
            break;
        case "dirt":
            audioSource.clip = dirtJump;
```

```
                audioSource.Play();
                break;
            case "rock":
                audioSource.clip = rockJump;
                audioSource.Play();
                break;
            case "scree":
                audioSource.clip = rockJump;
                audioSource.Play();
                break;
            case "sand":
                audioSource.clip = sandJump;
                audioSource.Play();
                break;
            case "gravel":
                audioSource.clip = gravelJump;
                audioSource.Play();
                break;
            default:
                break;
        }
    }

    private void FixedUpdate()
    {
        float speed;
        Calculate(out speed);

        Vector3 move = transform.right * input.x + transform.forward *
input.y; // Vektor koji upravlja kretnjama

        RaycastHit hitInfo;
        Physics.SphereCast(transform.position, controller.radius,
Vector3.down, out hitInfo,
            controller.height / 2f, Physics.AllLayers,
QueryTriggerInteraction.Ignore); //upisuje u hitInfo informaciju o tome što
je ispod nas
        move = Vector3.ProjectOnPlane(move, hitInfo.normal).normalized;

        controller.Move(move * speed * Time.deltaTime); //pomiče kontroler
```

```
isSwimming = CheckIfSwimming();

//definira kontrole kada je kontroler u vodi
if (isSwimming)
{
    gravity = 0;

    if (Input.GetKey(KeyCode.LeftShift) && !isDiving)
    {
        velocity.y = -50;
        controller.Move(velocity * Time.deltaTime);
        isDiving = true;
    }

    if (isDiving)
    {
        if (Input.GetKey(KeyCode.Space))
        {
            velocity.y = -waterGravity * Time.deltaTime;
        }
        else
        {
            velocity.y = 0;
        }

        if (Input.GetKey(KeyCode.LeftShift))
        {
            velocity.y = waterGravity * Time.deltaTime;
        }

        if (controller.transform.position.y > 38.35)
        {
            velocity.y = 0;
            isDiving = false;
        }
    }
    else
    {
        if (controller.transform.position.y > 38.3)
        {
```

```
                velocity.y = 0;
            }
            else
            {
                velocity.y = 5;
            }
        }
    }
    else
    {
        gravity = lastGravity;
        velocity.y += gravity * Time.deltaTime;
    }

    controller.Move(velocity * Time.deltaTime);

    isMoving = CheckIfMoving();

    if ((isGrounded || isSwimming) && isWalking && isMoving)
    {
        PlayFootstepSound(stepDelay);
    }

    if ((isGrounded || isSwimming) && !isWalking && isMoving)
    {
        PlayFootstepSound(stepDelay);
    }

    if (isDiving)
    {
        PlayFootstepSound(stepDelay);

    }

    if (Input.GetButtonDown("Jump") && isGrounded)
    {
        velocity.y = Mathf.Sqrt(jumpHeight * -2 * gravity);
        isJumping = true;
    }
}
```

```
    //provjerava pliva li igrač
    private bool CheckIfSwimming()
    {
        if (controller.transform.position.y < 38.4)
        {
            return true;
        }
        return false;
    }

    //detektira ulazak u vodu
    private void OnTriggerEnter(Collider other)
    {
        if (other.gameObject.name == "WaterProDaytime")
        {
            inWater = true;

            runningSpeed -= 5;
            walkingSpeed -= 5;

            stepDelay += 3;
            runDelay += 3;
        }
    }

    //detektira jesmo li u šupi
    private void OnTriggerStay(Collider other)
    {
        if (other.gameObject.name == "OldShed" || other.gameObject.name ==
"aband_house")
        {
            inBuilding = true;
        }

        if (other.gameObject.name == "RuinedBuilding")
        {
            inRuinedBuilding = true;
        }
    }
```

```
    //detektira izlaz iz vode ili šupe
    private void OnTriggerExit(Collider other)
    {
        if (other.gameObject.name == "WaterProDaytime")
        {
            inWater = false;

            runningSpeed += 5;
            walkingSpeed += 5;

            stepDelay -= 3;
            runDelay -= 3;
        }

        if (other.gameObject.name == "OldShed" || other.gameObject.name ==
"aband_house")
        {
            inBuilding = false;
        }

        if (other.gameObject.name == "RuinedBuilding")
        {
            inRuinedBuilding = false;
        }
    }

    //provjerava krećemo li se
    private bool CheckIfMoving()
    {
        if (transform.position != lastPosition)
        {
            lastPosition = transform.position;
            return true;
        }
        return false;
    }

    //računa brzinu trčanja i hodanja
    private void Calculate(out float speed)
```

```
{
    float horizontal = Input.GetAxis("Horizontal");
    float vertical = Input.GetAxis("Vertical");

    isWalking = !Input.GetKey(KeyCode.LeftControl);

    if (isWalking)
    {
        speed = walkingSpeed;
    }
    else
    {
        speed = runningSpeed;
    }

    input = new Vector2(horizontal, vertical);

    if (input.sqrMagnitude > 1)
    {
        input.Normalize();
    }
}

//pokreće zvuk hodanja ovisno o površini
public void PlayFootstepSound(int delay)
{
    currentDelay++;
    if (currentDelay < delay)
    {
        return;
    }
    currentDelay = 0;
    string textureName = GetTextureName();

    System.Random rand = new System.Random();

    CheckIfMoving();

    if (inWater && isMoving && !isGrounded && !isDiving)
    {
```

```
        audioSource.clip = SwimmingSound;
        if (!audioSource.isPlaying)
        {
            audioSource.Play();
        }
        return;
    }

    if (isDiving)
    {
        audioSource.clip = DivingSound;
        if (!audioSource.isPlaying)
        {
            audioSource.Play();
        }
        return;
    }

    if (inWater && isGrounded)
    {
        while (newIndex == lastIndex)
        {
            newIndex = rand.Next(0, waterFootsteps.Length);
        }
        lastIndex = newIndex;
        audioSource.clip = waterFootsteps[newIndex];
        audioSource.Play();
        return;
    }

    if (inBuilding)
    {
        while (newIndex == lastIndex)
        {
            newIndex = rand.Next(0, woodFootsteps.Length);
        }
        lastIndex = newIndex;
        audioSource.clip = woodFootsteps[newIndex];
        audioSource.Play();
        return;
```

```
}

if (inRuinedBuilding)
{
    while (newIndex == lastIndex)
    {
        newIndex = rand.Next(0, sandFootsteps.Length);
    }
    lastIndex = newIndex;
    audioSource.clip = sandFootsteps[newIndex];
    audioSource.Play();
    return;
}

if (!inBuilding)
{
    switch (textureName)
    {
        case "moss":
            while (newIndex == lastIndex)
            {
                newIndex = rand.Next(0, grassFootsteps.Length);
            }
            lastIndex = newIndex;
            audioSource.clip = grassFootsteps[newIndex];
            audioSource.Play();
            break;
        case "dirt":
            while (newIndex == lastIndex)
            {
                newIndex = rand.Next(0, dirtFootsteps.Length);
            }
            lastIndex = newIndex;
            audioSource.clip = dirtFootsteps[newIndex];
            audioSource.Play();
            break;
        case "rock":
            while (newIndex == lastIndex)
            {
                newIndex = rand.Next(0, rockFootsteps.Length);
```

```
                    }
                    lastIndex = newIndex;
                    audioSource.clip = rockFootsteps[newIndex];
                    audioSource.Play();
                    break;
                case "scree":
                    while (newIndex == lastIndex)
                    {
                        newIndex = rand.Next(0, rockFootsteps.Length);
                    }
                    lastIndex = newIndex;
                    audioSource.clip = rockFootsteps[newIndex];
                    audioSource.Play();
                    break;
                case "sand":
                    while (newIndex == lastIndex)
                    {
                        newIndex = rand.Next(0, sandFootsteps.Length);
                    }
                    lastIndex = newIndex;
                    audioSource.clip = sandFootsteps[newIndex];
                    audioSource.Play();
                    break;
                case "gravel":
                    while (newIndex == lastIndex)
                    {
                        newIndex = rand.Next(0, gravelFootsteps.Length);
                    }
                    lastIndex = newIndex;
                    audioSource.clip = gravelFootsteps[newIndex];
                    audioSource.Play();
                    break;
                default:
                    break;
            }
        }

    }
//dohvaća naziv teksture pod nama
    private string GetTextureName()
```

```
    {
        return terrainLayers[surfaceIndex].name;
    }

    //dohvaća terrain koji je trenutno pod nama
    Terrain GetClosestCurrentTerrain(Vector3 playerPos)
    {
        //dohvaća najbliži terrain
        var center = new Vector3(_terrains[0].transform.position.x +
_terrains[0].terrainData.size.x / 2, playerPos.y,
_terrains[0].transform.position.z + _terrains[0].terrainData.size.z / 2);
        float lowDist = (center - playerPos).sqrMagnitude;
        var terrainIndex = 0;

        for (int i = 0; i < _terrains.Length; i++)
        {
            center = new Vector3(_terrains[i].transform.position.x +
_terrains[i].terrainData.size.x / 2, playerPos.y,
_terrains[i].transform.position.z + _terrains[i].terrainData.size.z / 2);

            //Traži postoji li manja udaljenost
            var dist = (center - playerPos).sqrMagnitude;
            if (dist < lowDist)
            {
                lowDist = dist;
                terrainIndex = i;
            }
        }
        return _terrains[terrainIndex];
    }

    //dohvaća sve teksture ispod nas
    private float[] GetTextureMix(Vector3 WorldPos)
    {
        int mapX = (int)(((WorldPos.x - terrainPos.x) / terrainData.size.x)
* terrainData.alphamapWidth);
        int mapZ = (int)(((WorldPos.z - terrainPos.z) / terrainData.size.z)
* terrainData.alphamapHeight);

        if (mapX > 511 || mapX < 0 || mapZ > 511 || mapZ < 0)
        {
```

```
            mapX = 0;
            mapZ = 0;
        }
        float[,,] splatmapData = terrainData.GetAlphamaps(mapX, mapZ, 1,
1);

        float[] cellMix = new float[splatmapData.GetUpperBound(2) + 1];

        for (int n = 0; n < cellMix.Length; n++)
        {
            cellMix[n] = splatmapData[0, 0, n];
        }
        return cellMix;
    }
    //dohvaća najzastupljeniju teksturu
    private int GetMainTexture(Vector3 WorldPos)
    {
        float[] mix = GetTextureMix(WorldPos);
        float maxMix = 0;
        int maxIndex = 0;

        for (int n = 0; n < mix.Length; n++)
        {
            if (mix[n] > maxMix)
            {
                maxIndex = n;
                maxMix = mix[n];
            }
        }
        return maxIndex;
    }
}
```

Dio koda zaslužan za detektiranje koji terrain se nalazi pod nama razvio je stack overflow korisnik „miralong“[48], dio za detektiranje toče teksture Unity forum korisnik „ExDeaDguY“[35], a kretnje su inspirirane kodom youtube korisnika „Brackeys“[33].

Sljedeća skripta je skripta „PlayerHealth“ koja sadrži informaciju o trenutnom stanju života igrača, te pokreće povezane zvukove i animacije. Ova skripta dodana je na čitav „FirstPersonController“.

PlayerHealth

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Security.Cryptography;
using UnityEngine;
using UnityEngine.SceneManagement;

public class PlayerHealth : MonoBehaviour
{
    public static int currentHealth = 20;
    public int myHealth;
    public GameObject fadeOut;
    public AudioClip PlayerHurtSound1;
    public AudioClip PlayerHurtSound2;
    public AudioClip PlayerDeadSound;
    public int randomNum;
    public AudioSource audioSource;
    private bool died;
    private System.Random rand;

    private void Start()
    {
        died = false;
        rand = new System.Random();
    }

    void Update()
    {
        if(currentHealth < myHealth)
        {
            PlayHurtSound();
        }
        myHealth = currentHealth;
```

```
        if(currentHealth <= 0)
        {
            StartCoroutine(Wait());
        }
        //Pokreće animaciju i pali Game Over scenu
        IEnumerator Wait()
        {
            PlayDyingSound();
            fadeOut.GetComponent<Animation>().Play("FadeOutAnimation");
            yield return new WaitForSeconds(1);
            SceneManager.LoadScene(2);
        }
    }
    //Pokreće zvuk umiranja
    private void PlayDyingSound()
    {
        if (!died)
        {
            audioSource.clip = PlayerDeadSound;
            audioSource.Play();
            died = true;
        }
    }

    //pokreće zvuk ozljeđivanja
    private void PlayHurtSound()
    {
        randomNum = rand.Next(0, 2);
        if (randomNum == 1)
        {
            audioSource.clip = PlayerHurtSound1;
        }
        else
        {
            audioSource.clip = PlayerHurtSound2;
        }
        audioSource.Play();
    }
}
```

Za interakciju sa objektima trebamo znati našu udaljenost od tog objekta. Za to koristimo skriptu „DetectDistance“ koju pozivamo svaki put kada trebamo znati udaljenost od nečega[38].

DetectDistance

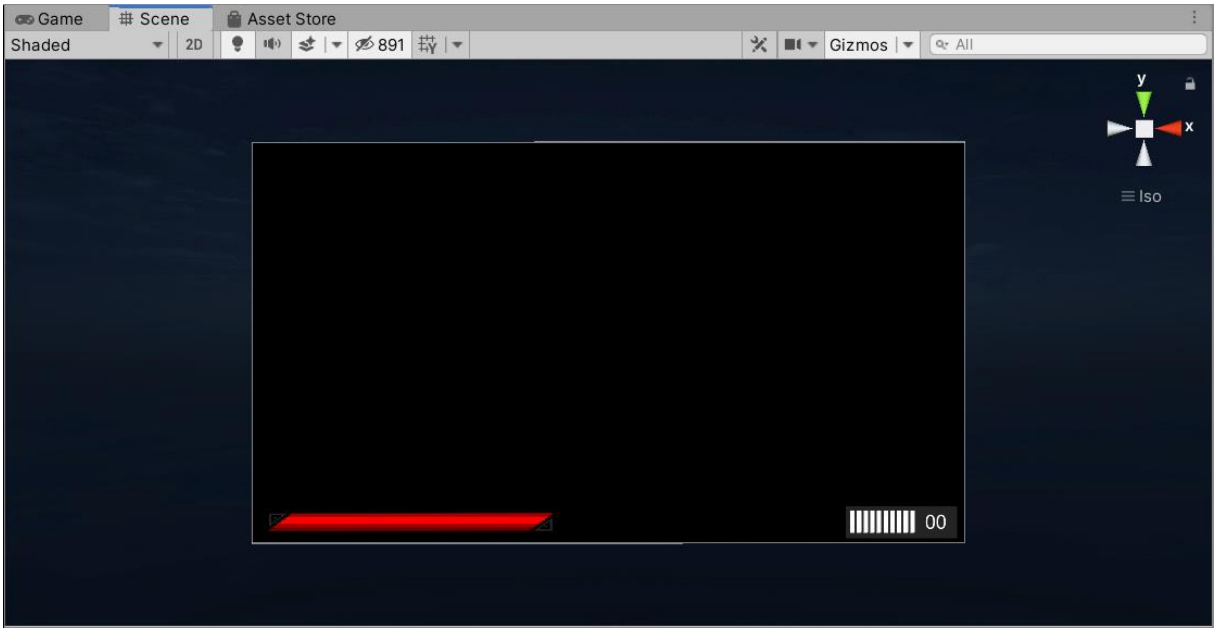
```
using System.Collections;
using System.Collections.Generic;
using System.Xml.Schema;
using UnityEngine;

public class DetectDistance : MonoBehaviour
{
    public static float TargetDistance;
    void Update()
    {
        RaycastHit hit;
        if(Physics.Raycast (transform.position,
transform.TransformDirection (Vector3.forward), out hit))
        {
            TargetDistance = hit.distance;
        }
    }
}
```

Sljedeća funkcija u igri je korištenje ručne svjetiljke. Za to koristimo skriptu „UseFlashlight“ koja provjerava postoji li svjetiljka u sceni, te ako ne postoji (pokupljena je) omogućava paljenje i gašenje svjetla tipa „spot light“ koje je usmjereno od igrača prema naprijed kako bi nalikovalo svjetiljki.

UseFlashlight

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class UseFlashlight : MonoBehaviour
{
    public GameObject Flashlight;
    public GameObject FlashlightInHand;
    private bool FlashlightWorking;

    private void Start()
    {
        FlashlightWorking = false;
    }

    void Update()
    {
        if(!Flashlight.activeInHierarchy && Input.GetKeyDown(KeyCode.F))
        {
            ToggleFlashlight();
        }
    }
    private void ToggleFlashlight()
    {
        if (FlashlightWorking)
        {
            FlashlightWorking = false;
            FlashlightInHand.SetActive(false);
        }
        else
        {
            FlashlightWorking = true;
            FlashlightInHand.SetActive(true);
        }
    }
}
```

Igrač mora imati dostupne informacije o svom liku, kao što su život ili metci. Ako uredimo tzv. „Canvas“, odnosno platno koje se uvijek nalazi na ekranu, tako da dodamo elemente koje je moguće manipulirati, pomoću njih ćemo moći prikazati sve potrebne informacije. Na slici ispod prikazano je jednostavno sučelje sa praznim okvirom za život ispod kojeg je prazna crvena slika na koju je dodan „slider“ [42]. Sučelje za metke je napravljeno samo uz pomoć praznih slika i jednog tekstnog okvira. Skripta „UpdateGUI“ upravlja tim elementima i prikazuje vrijednosti dohvaćene iz drugih skripti, kao na primjer „PlayerHealth“.



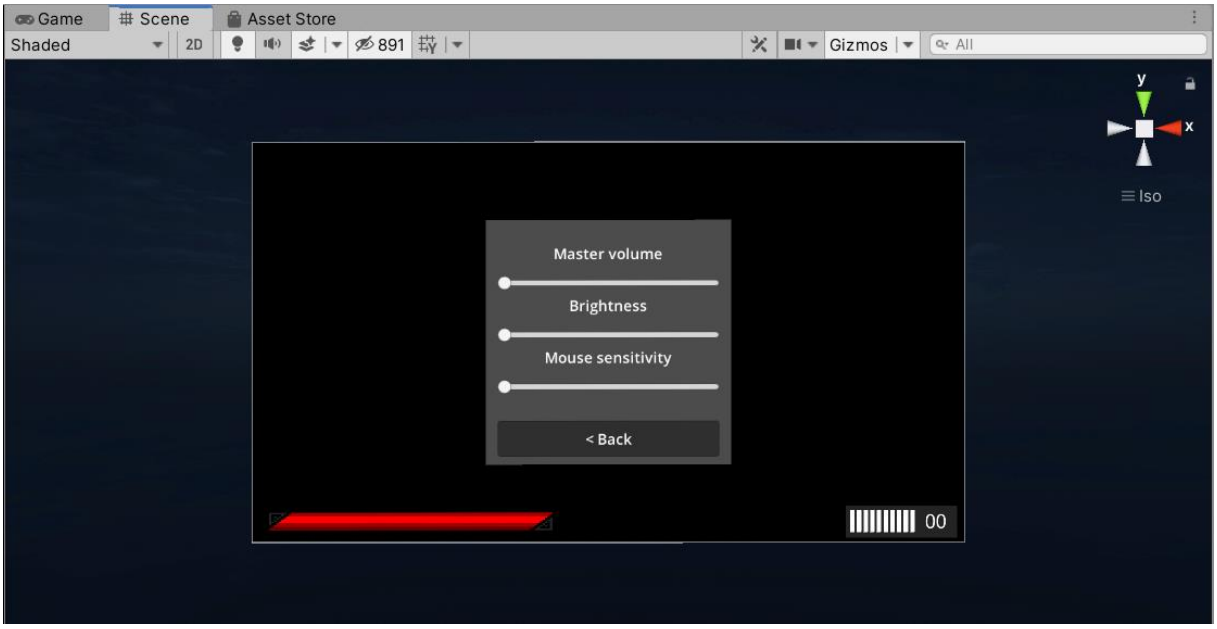
Slika 33: Player GUI

UpdateGUI

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class UpdateGUI : MonoBehaviour
{
    public Slider healthBar;
    public GameObject bullet1;
    public GameObject bullet2;
    public GameObject bullet3;
    public GameObject bullet4;
    public GameObject bullet5;
    public GameObject bullet6;
    public GameObject bullet7;
    public GameObject bullet8;
    public GameObject bullet9;
    public GameObject bullet10;
    public Text guiAmmo;
    public GameObject AmmoBlock;
    public GameObject Player;
    public GameObject Gun;

    private int health;
    private int ammoInGun;
    private int ammoInReserve;

    private void Start()
    {
        healthBar.maxValue = 20;
        healthBar.minValue = 0;
    }

    void Update()
    {
        health = Player.GetComponent<PlayerHealth>().myHealth;
        ammoInGun = Gun.GetComponent<FireGun>().ammoInGun;
```

```
        ammoInReserve = Gun.GetComponent<FireGun>().ammoInReserve;

        healthBar.value = health;

        AmmoBlock.SetActive(Gun.activeInHierarchy);

        if (Gun.activeInHierarchy)
        {
            bullet1.SetActive(ammoInGun >= 1);
            bullet2.SetActive(ammoInGun >= 2);
            bullet3.SetActive(ammoInGun >= 3);
            bullet4.SetActive(ammoInGun >= 4);
            bullet5.SetActive(ammoInGun >= 5);
            bullet6.SetActive(ammoInGun >= 6);
            bullet7.SetActive(ammoInGun >= 7);
            bullet8.SetActive(ammoInGun >= 8);
            bullet9.SetActive(ammoInGun >= 9);
            bullet10.SetActive(ammoInGun >= 10);

            if (ammoInReserve < 10)
            {
                guiAmmo.text = "0" + ammoInReserve;
            }
            else
            {
                guiAmmo.text = ammoInReserve.ToString();
            }
        }
    }
}
```

Igraču je potrebno dodijeliti i kontrole nad nekim opcijama, kao što su svjetlina, glasnoća, i osjetljivost miša. Svaka od tih mogućnosti ima zasebnu skriptu koja uzima vrijednosti iz klizača i primjenjuje ju na potrebnim mjestima. Imamo tako skripte „VolumeControl“, „BrightnessControl“ i „SensitivityControl“ [47].



Slika 34: Postavke

VolumeControl

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class VolumeControll : MonoBehaviour
{
    public Slider Volume;

    private void Start()
    {
        Volume.value = 0.5f;
    }
    private void Update()
    {
```

```
        AudioListener.volume = Volume.value;
    }
}
```

BrightnessControl

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class BrightnessControll : MonoBehaviour
{
    public Slider Brightness;
    public Light Lighting;

    private void Start()
    {
        Lighting.intensity = 0.05f;
        Brightness.value = 0.05f;
    }

    private void Update()
    {
        Lighting.intensity = Brightness.value;
    }

}
```

SensitivityControl

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class SensitivityControll : MonoBehaviour
{
    public Slider MouseSensitivity;

    private void Start()
    {
        MouseSensitivity.value = 2f;
    }

    private void Update()
    {
        GetComponent<CameraMove>().mouseSensitivity =
MouseSensitivity.value;
    }
}
```

Igraču je potreban nekakav način obrane, pa mu dodajemo pištolj na koji dodajemo skriptu „FireGun“ koja pokreće animacije dodane na pištolj, smanjuje količinu metaka, omogućuje punjenje pištolja metcima, definira koliko štete će nanijeti jedan metak, te šalje poruku „DamageEnemy“ sa vrijednošću štete koju smo definirali, te koju ćemo kasnije prihvatiti u drugoj skripti.

FireGun

```
using System.Collections;
using System.Collections.Generic;
using System.Runtime.CompilerServices;
using UnityEngine;

public class FireGun : MonoBehaviour
{
    public GameObject Pistol;
    public GameObject FireFlash;
    public GameObject FireFlashLight;
    public AudioClip PistolFire;
    public AudioClip PistolEmpty;
    public AudioClip Reload;

    public GameObject playerCammera;

    public int ammoInGun;
    public int ammoInReserve;

    public bool isFiring;
    public bool isReloading;

    public float TargetDistance;
    public int Damage = 5;

    private AudioSource audioSource;

    void Start()
    {
        audioSource = GetComponent<AudioSource>();
        isFiring = false;
        ammoInGun = 10;
    }

    void Update()
    {
        if (!playerCammera.GetComponent<CameraMove>().menuOn)
        {
```

```
            if (!audioSource.isPlaying)
            {
                isReloading = false;
            }
            if
(!FireFlash.GetComponent<Animation>().IsPlaying("PistolFireAnimation"))
            {
                FireFlashLight.SetActive(false);
            }
            if
(FireFlash.GetComponent<Animation>().IsPlaying("PistolFireAnimation"))
            {
                isFiring = true;
            }
            else
            {
                isFiring = false;
            }

            if (Pistol.activeInHierarchy && !isFiring && !isReloading)
            {
                if (Input.GetKeyDown(KeyCode.Mouse0))
                {
                    if (ammoInGun == 0)
                    {
                        audioSource.clip = PistolEmpty;
                        audioSource.Play();
                    }
                    else
                    {
                        RaycastHit Fire;
                        if (Physics.Raycast(transform.position,
transform.TransformDirection(Vector3.forward), out Fire))
                        {
                            TargetDistance = Fire.distance;
                            Fire.transform.SendMessage("DamageEnemy",
Damage, SendMessageOptions.DontRequireReceiver);
                        }

Pistol.GetComponent<Animation>().Stop("PistolAnimation");
                        audioSource.clip = PistolFire;
```

```
                        audioSource.Play();

Pistol.GetComponent<Animation>().Play("PistolAnimation");

FireFlash.GetComponent<Animation>().Play("PistolFireAnimation");
                        FireFlashLight.SetActive(true);
                        ammoInGun--;
                    }
                }
                if (Input.GetKeyDown(KeyCode.R))
                {
                    int ammoMissing = 10 - ammoInGun;
                    if (ammoInReserve != 0)
                    {
                        if (ammoInReserve <= ammoMissing)
                        {
                            ammoInGun += ammoInReserve;
                            ammoInReserve = 0;
                            isReloading = true;
                            audioSource.clip = Reload;
                            audioSource.Play();
                        }
                        if (ammoInReserve > ammoMissing)
                        {
                            ammoInGun += ammoMissing;
                            ammoInReserve -= ammoMissing;
                            isReloading = true;
                            audioSource.clip = Reload;
                            audioSource.Play();
                        }
                    }
                }
            }
        }
    }
}
```

Još jedna bitna stvar za igrača je da može kupiti stvari, kao na primjer pištolj, svjetiljku, pilu, itd. Sve skripte rade na istom principu, a primjer takve skripte je skripta „PickUpPistol“. Ova skripta koristi skriptu „DetectDistance“ da odredi udaljenost igrača od pištolja, te ako je

dovoljno blizu na ekran ispisuje „[E]“, kao tipku za interakciju, „Pick up gun“, te postavlja „ActionCrosshair“ koji je samo dodatan ukras postojećem kako bi naglasili da je moguća interakcija. Kupljenje svega ostalog realizirano je po istom principu[39]. Također ćemo dodati i efekt sjaja na stvari koje može pokupiti koristeći „particle system“ kako bi ih igrač mogao primijetiti[43].

Slika 35: Kupljenje pištolja

PickUpPistol

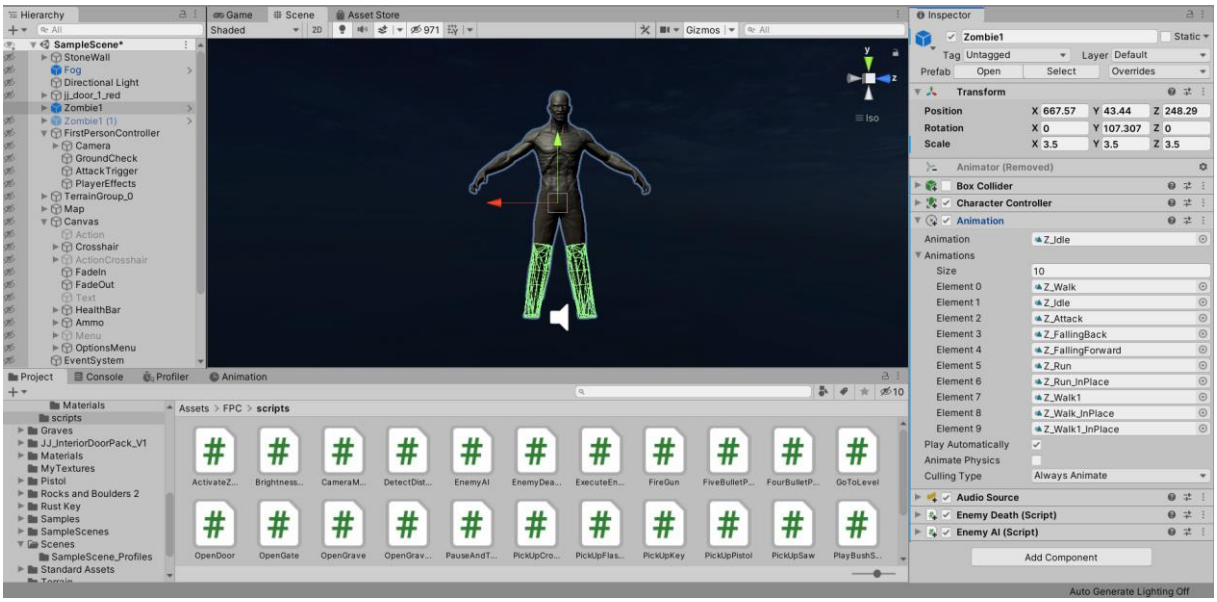
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class PickUpPistol : MonoBehaviour
{
    public float Distance;
    public GameObject DisplayActionKey;
    public GameObject DisplayText;
    public GameObject ActionCrosshair;
    public GameObject PistolPickup;
    public GameObject Pistol;
```

```
private Text text;

void Update()
{
    Distance = DetectDistance.TargetDistance;
}
private void OnMouseOver()
{
    if (Distance < 5)
    {
        DisplayActionKey.SetActive(true);
        DisplayText.SetActive(true);
        ActionCrosshair.SetActive(true);
        text = DisplayText.GetComponent<Text>();
        text.text = "Pick up pistol";
    }
    if (Distance >= 5)
    {
        DisplayActionKey.SetActive(false);
        DisplayText.SetActive(false);
        ActionCrosshair.SetActive(false);
    }
    if (Input.GetKey(KeyCode.E) && Distance < 5)
    {
        this.GetComponent<BoxCollider>().enabled = false;
        PistolPickup.SetActive(false);
        Pistol.SetActive(true);
        DisplayActionKey.SetActive(false);
        DisplayText.SetActive(false);
        ActionCrosshair.SetActive(false);
        text = DisplayText.GetComponent<Text>();
        text.text = "";
        Pistol.GetComponent<FireGun>().ammoInGun = 10;
    }
}
private void OnMouseExit()
{
    DisplayActionKey.SetActive(false);
    DisplayText.SetActive(false);
    ActionCrosshair.SetActive(false);
```

```
    }
}
```

3.3. Izrada neprijatelja

Za neprijatelje koristit ćemo zombije čiji model je preuzet u „asset store“. Ovaj model ima već izrađene animacije koje je samo potrebno uključiti u određenom trenutku. Također je potrebno i dodati nekakvu „umjetnu inteligenciju“. U skripti „EnemyAI“ realizirana je vrlo jednostavna „umjetna inteligencija“ koja računa položaj neprijatelja i igrača, stvara vektor koji povezuje to dvoje i pomiče neprijatelja prema igraču. Kada neprijatelj dođe u kontakt sa okidačem „AttackTrigger“, koji je zapravo samo disk pozicioniran na naš „FirstPersonController“, pokreće se animacija napadanja, a inače pokrećemo animaciju trčanja [40], [41]. Ova skriptra manipulira javnom varijablom „currentHealth“ iz skripte „PlayerHealth“ tako da oduzima definiranu vrijednost.



Slika 36: Model zombija

EnemyAI

```
using System;
using System.Collections;
using System.Collections.Generic;
using Unity.Collections.LowLevel.Unsafe;
using UnityEditor;
using UnityEngine;

public class EnemyAI : MonoBehaviour
{
    public GameObject Player;
    public GameObject PlayerController;
    public GameObject Enemy;
    public float enemySpeed = 0.01f;
    public int enemyDamage = 5;
    public bool attackTrigger = false;
    public bool isAttacking = false;
    public bool isDead = false;
    public AudioClip ZombieSound;
    public AudioClip ZombieAttack;
    public CharacterController controller;
    Vector3 velocity;
    private float speed;
    private AudioSource audioSource;

    private void Start()
    {
        speed = enemySpeed;
        audioSource = GetComponent<AudioSource>();
    }
    void Update()
    {
        if (isDead)
        {
            audioSource.Stop();
        }
```

```
        if (!isDead && Vector3.Distance(Enemy.transform.position,
Player.transform.position) <= 130)
        {
            Enemy.GetComponent<Animation>().Stop("Z_Idle");
            controller.Move(velocity * Time.deltaTime);
            if (!Enemy.GetComponent<Animation>().IsPlaying("Z_Attack") &&
isAttacking)
            {
                isAttacking = false;
            }
            transform.LookAt(Player.transform);
            if (!attackTrigger && !isAttacking)
            {
                if (PlayerController.GetComponent<PlayerMove>().isSwimming)
                {
                    enemySpeed = 0f;
                    Enemy.GetComponent<Animation>().Play("Z_Idle");
                }
                else
                {
                    enemySpeed = speed;
                    Enemy.GetComponent<Animation>().Play("Z_Run_InPlace");
                    if (!Enemy.GetComponent<AudioSource>().isPlaying)
                    {
                        PlayZombieSound();
                    }
                    else
                    {
                        if (Enemy.GetComponent<AudioSource>().clip !=
ZombieSound && Player.GetComponent<PlayerHealth>().myHealth > 0)
                        {
                            PlayZombieSound();
                        }
                    }
                }
                velocity.y = -4f;
                controller.Move(velocity * Time.deltaTime);
                transform.position =
Vector3.MoveTowards(transform.position, Player.transform.position,
enemySpeed * Time.deltaTime);
            }
```

```
            if (attackTrigger && !isAttacking)
            {
                enemySpeed = 0f;
                isAttacking = true;
                Enemy.GetComponent<Animation>().Play("Z_Attack");
                if (PlayerController.GetComponent<PlayerHealth>().myHealth
> 0)
                {
                    PlayAttackSound();
                }
                PlayerHealth.currentHealth -= enemyDamage;
            }
        }
        else
        {
            if (!isDead)
            {
                enemySpeed = 0f;
                Enemy.GetComponent<Animation>().Play("Z_Idle");
                velocity.y = -4f;
                controller.Move(velocity * Time.deltaTime);
            }
        }
    }
    private void PlayZombieSound()
    {
        audioSource.clip = ZombieSound;
        audioSource.Play();
    }
    private void PlayAttackSound()
    {
        audioSource.clip = ZombieAttack;
        audioSource.Play();
    }
    private void OnTriggerEnter(Collider other)
    {
        if (other.gameObject.name == "AttackTrigger" && !isDead)
        {
            attackTrigger = true;
        }
```

```
    }
    private void OnTriggerExit(Collider other)
    {
        if (other.gameObject.name == "AttackTrigger" && !isDead){
            attackTrigger = false;
        }
    }
}
```

Potrebno je pokrenuti animacije umiranja kada igrač ubije neprijatelja, te maknuti sve kolizije iz modela kako ne bi imali nevidljive prepreke. Te mogućnosti ćemo definirati u skripti „EnemyDeath“ [40].

EnemyDeath

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class EnemyDeath : MonoBehaviour
{
    public int EnemyHealth = 20;
    public GameObject Enemy;
    public GameObject Leg1;
    public GameObject Leg2;
    public int StatusCheck;
    public int randomNumber;
    private System.Random rand;

    void DamageEnemy(int damage)
    {
        rand = new System.Random();
        EnemyHealth -= damage;
    }
    void Update()
    {
        if(EnemyHealth <= 0 && StatusCheck == 0)
        {
```

```
            Enemy.GetComponent<EnemyAI>().isDead = true;
            StatusCheck = 2;
            Enemy.GetComponent<Animation>().Stop();
            randomNumber = rand.Next(0, 2);
            Enemy.GetComponent<CharacterController>().enabled = false;

            Leg1.GetComponent<MeshCollider>().enabled = false;
            Leg2.GetComponent<MeshCollider>().enabled = false;

            Enemy.GetComponent<AudioSource>().Stop();
            if (randomNumber == 1){
                Enemy.GetComponent<Animation>().Play("Z_FallingBack");
            }
            else{
                Enemy.GetComponent<Animation>().Play("Z_FallingForward");
            }
        }
    }
}
```

U početku neprijatelji će biti onemogućeni, sve dok igrač ne napravi određenu akciju (prođe kroz okidač, pokupi pilu i slično), a omogućujemo ih kroz skripte ovisno o potrebama. Primjer skripte koja omogućuje neprijatelja kada igrač dođe u kontakt sa nevidljivim okidačem je „ActivateZombie“.

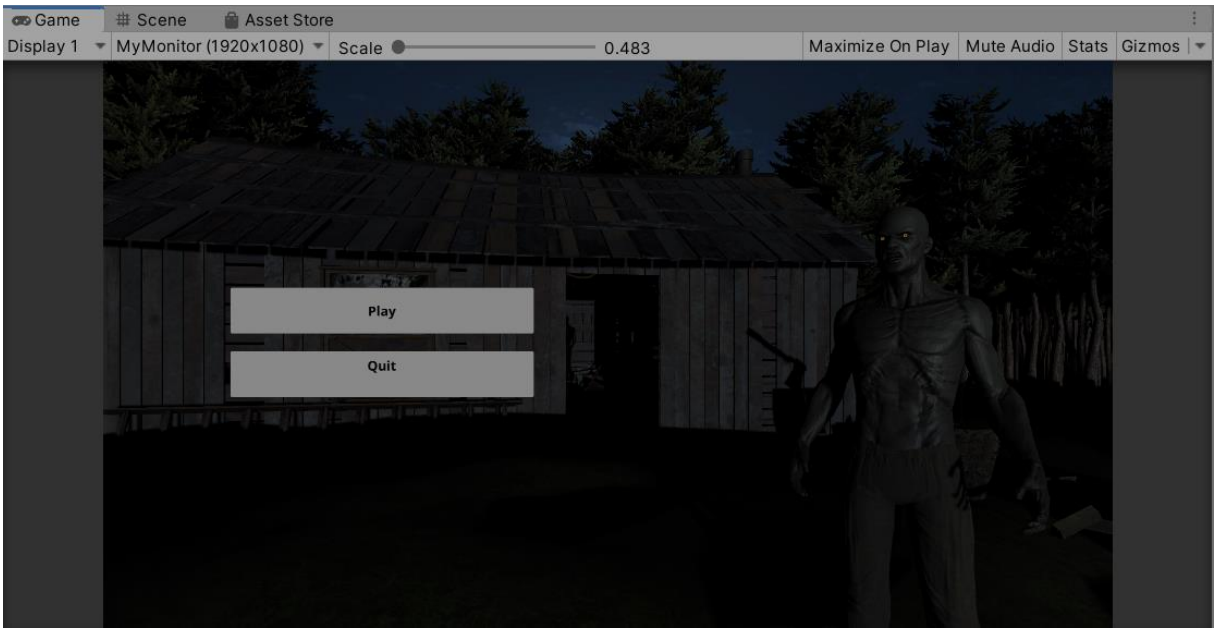
ActivateZombie

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ActivateZombie : MonoBehaviour
{
    public GameObject Zombie;

    private void OnTriggerEnter(Collider other)
    {
        this.GetComponent<BoxCollider>().enabled = false;
```

```
            Zombie.SetActive(true);
        }
}
```

3.4. Scene

U naš projekt dodati ćemo 4 nove scene: „MainMenu“, „GameOver“, „Ending“ i “Controls“. Main menu imati će postavljenu scenu kao ukras i dodana dva gumba, „Play game“ i „Quit“ [45], [46]. Pritiskom na „Quit“ izlazimo iz igre. Pritiskom na „Play game“ pokreće se skripta „PlayButton“ koja pokreće animaciju zatamnjivanja ekrana nakon čega prelazi na scenu „Controls“. Scena također ima i skriptu „StartMenu“ koja prikazuje miš i omogućava njegovo kretanje ako je prije oduzeto.



Slika 37: Main menu

StartMenu

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class StartMenu : MonoBehaviour{
    private void Start(){
```

```
        Cursor.visible = true;
        Cursor.lockState = CursorLockMode.None;
    }
}
```

PlayGame

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;

public class PlayButton : MonoBehaviour
{
    public GameObject fade;
    public void PlayGame()
    {
        fade.SetActive(true);
        fade.GetComponent<Animation>().Play("FadeInMenu");
        StartCoroutine(StartGame());
    }

    IEnumerator StartGame()
    {
        yield return new WaitForSeconds(1);
        SceneManager.LoadScene(4);
    }
}
```

QuitGame

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;

public class QuitGame : MonoBehaviour{
```

```
    public void ExitGame(){
        Application.Quit();
    }
}
```

Scena „Controls“ samo sadrži informacije o kontrolama i skriptu „GoToLevel“ koja čeka 5 sekundi te prebacuje na scenu u kojoj je igra.

Slika 38: Controls scena

GoToLevel

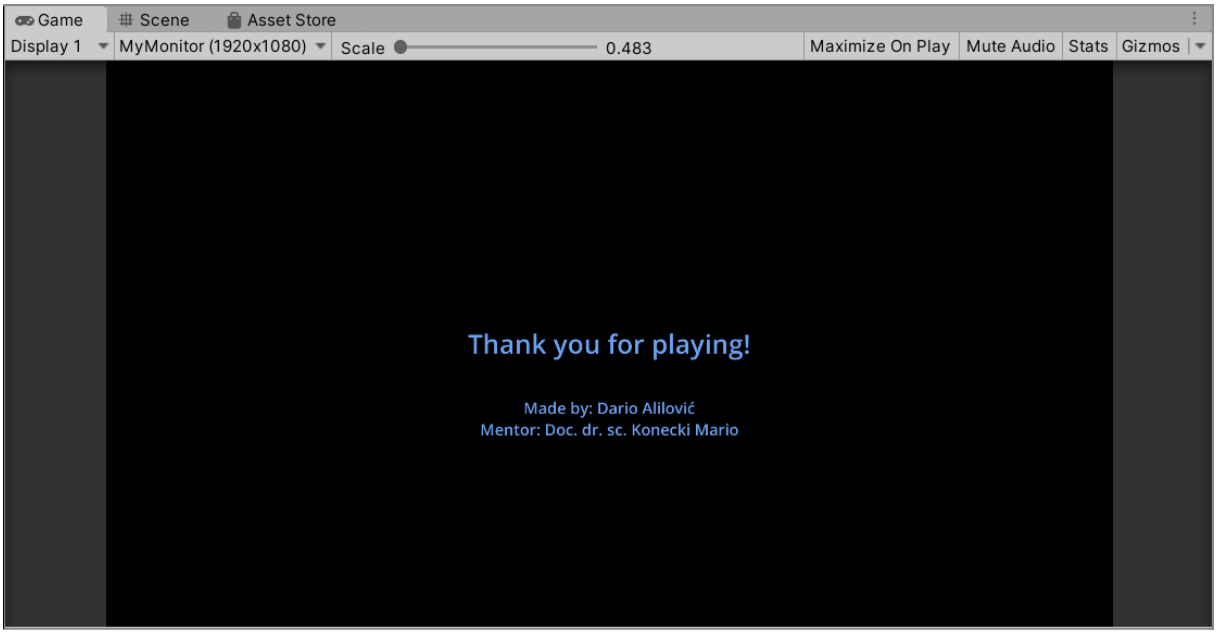
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class GoToLevel : MonoBehaviour
{
    private void Start()
    {
        StartCoroutine(StartLevel());
    }
```

```
    IEnumerator StartLevel()
    {
        yield return new WaitForSeconds(5);
        SceneManager.LoadScene(1);
    }
}
```

Scena „GameOver“ pokreće se kada igrač umre. Ova scena sadrži samo tekst „Game over“ i skriptu „ToMainMenu“ koja vodi na scenu „MainMenu“.

Slika 39: Game over scena

Slična je i scena „Ending“ koja se pokreće kada igrač pređe igru, a sadrži tekst i skriptu ToMainMenu.



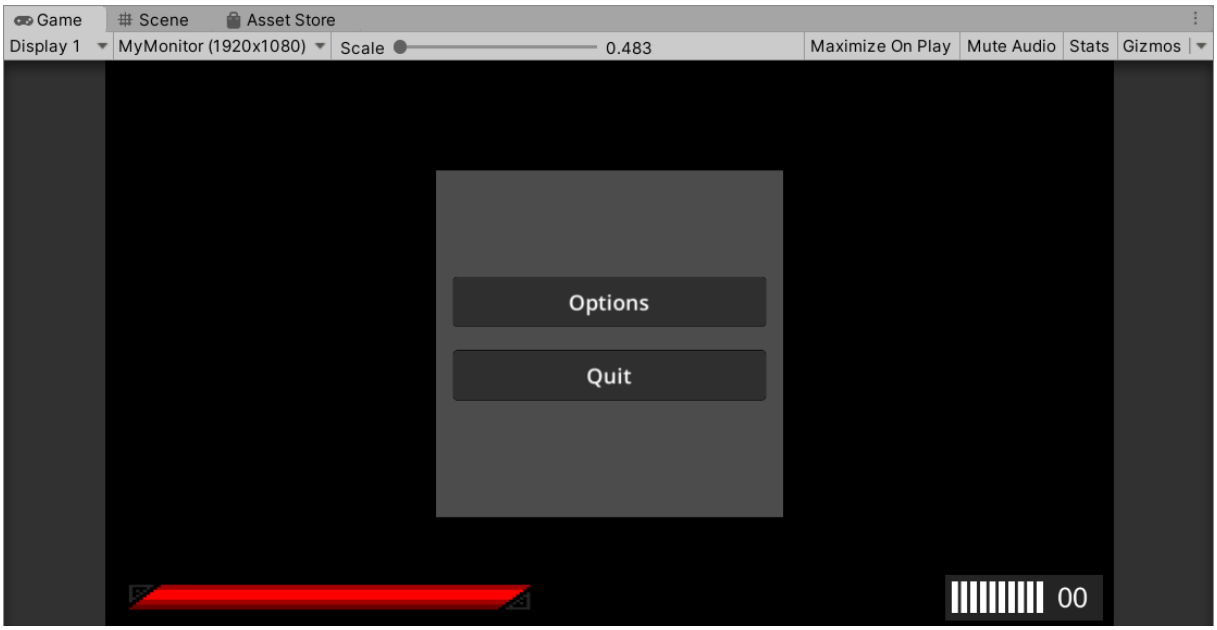
Slika 40: Ending scena

PauseAndTransition

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class PauseAndTransition : MonoBehaviour
{
    private void Start()
    {
        StartCoroutine(Transition());
    }

    IEnumerator Transition()
    {
        yield return new WaitForSeconds(5);
        SceneManager.LoadScene(0);
    }
}
```

3.5. Ostale funkcije

U In-Game izborniku pritiskom na gumb „Quit“ vraćamo se u glavni izbornik. Za to koristimo skriptu „ToMainMenu“.



Slika 41: In-game izbornik

ToMainMenu

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class ToMainMenu : MonoBehaviour
{
    public void BackToMainMenu()
    {
        SceneManager.LoadScene(0);
    }
}
```

Postaviti ćemo i nekoliko okidača koji, kada igrač prođe kroz njih, pokreću određene zvukove. Skripta za tu funkciju je „PlayBushSound“.

PlayBushSound

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PlayBushSound : MonoBehaviour
{
    public AudioClip BushRustle;
    private AudioSource audioSource;
    private void OnTriggerEnter(Collider other)
    {
        this.GetComponent<BoxCollider>().enabled = false;
        audioSource = GetComponent<AudioSource>();
        audioSource.clip = BushRustle;
        audioSource.Play();
    }
}
```

I posljednja skripta je skripta koja se pokreće kada igrač dođe u kontakt sa okidačem na kraju igre. Ta skripta je „ExecuteEnding“, a njezina funkcija je da pokreće scenu „Ending“.

ExecuteEnding

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class ExecuteEnding : MonoBehaviour
{
    public GameObject fadeOut;
    private void OnTriggerEnter(Collider other)
    {
```

```
        fadeOut.GetComponent<Animation>().Play("FadeOutAnimation");
        StartCoroutine(Wait());
    }

    IEnumerator Wait()
    {
        yield return new WaitForSeconds(1);
        SceneManager.LoadScene(3);
    }
}
```

4. Zaključak

Unity je vrlo moćan alat za razvoj video igara koji uvelike olakšava čitav proces razvoja. Neke od najvažnijih mogućnosti alata Unity su lagana manipulacija objekata, velik broj postavki i tipova osvjetljenja, stvaranje animacija, lagano modeliranje površina, lagana integracija skripti i slično. Jedna od vrlo korisnih stvari u alatu Unity je „Asset store“ preko kojega možemo preuzimati modele, osvjetljenja, skybox i razna druga rješenja drugih korisnika, koja možemo integrirati u vlastitu igru. Korisničko sučelje je vrlo kompaktno i lagano za korištenje, što omogućuje brzo i lagano podešavanje željenih postavki. Za razvoj koda, Unity kao zadani program koristi Visual Studio.

Kao početniku u razvoju video igara, razvoj vlastite horor igre koristeći alat Unity bio je vrlo izazovan, ali vrlo brz i sve u svemu lagan. Koristeći mnoge alate lagano se prilagoditi i ispraviti pogreške. Prilikom razvoja ove igre, naučio sam mnogo toga o Unity alatu, metodama razvoja, pa čak i neke nove metode kodiranja skripti. Naravno za razvoj neke potpune igre bilo bi potrebno puno više vremena i puno više pažnje posvećeno sitnim detaljima.

Žanr horor igre preživljavanja bio je vrlo zanimljiv za istraživanje i razvoj iako je zbog svoje prirode stvorio neke probleme koje je trebalo riješiti. Ova vrsta igre ima vrlo velik potencijal jer možemo biti kreativni u načinu izrade, te nismo ograničeni nikakvim preprekama.

Osobno, vrlo sam zadovoljan Unity alatom, kao i sa Unity zajednicom gdje ljudi dijele znanje i materijale sa drugima. Uz Unity zajednicu i Unity priručnik vrlo je lagano naučiti osnove razvoja video igara, čak i bez prijašnjeg znanja o razvoju igara ili programiranju. Unity bih preporučio svakome tko se želi iskušati u razvoju video igara.

Cijeli projekt dostupan je na sljedećoj poveznici:

https://drive.google.com/file/d/17YVOfBjZJo0tQaXzf9FOnJoyrXkj5tug/view?usp=sharing

5. Popis literature

[1] Unity Technologies (bez dat.) *Multiplatform* [Na internetu]. Dostupno: https://unity.com/features/multiplatform [pristupano 22.06.2020.].

[2] Jon Brodkin, „How Unity3D Became a Game-Development Beast“, 2013. [Na internetu]. Dostupno: https://insights.dice.com/2013/06/03/how-unity3d-become-a-game-development-beast/ [pristupano 22.06.2020.].

[3] Unity Technologies (22.06.2020.) *Unity's interface* [Na internetu]. Dostupno: https://docs.unity3d.com/Manual/UsingTheEditor.html [pristupano 22.06.2020.].

[4] Unity Technologies (23.06.2020.) *The project window* [Na internetu]. Dostupno: https://docs.unity3d.com/Manual/ProjectView.html [pristupano 23.06.2020.].

[5] Unity Technologies (23.06.2020.) *The scene view* [Na internetu]. Dostupno: https://docs.unity3d.com/Manual/UsingTheSceneView.html [pristupano 23.06.2020.].

[6] Unity Technologies (23.06.2020.) *Scene view navigation* [Na internetu]. Dostupno: https://docs.unity3d.com/Manual/SceneViewNavigation.html [pristupano 23.06.2020.].

[7] Unity Technologies (24.06.2020.) *Picking and selecting GameObjects* [Na internetu]. Dostupno: https://docs.unity3d.com/Manual/ScenePicking.html [pristupano 24.06.2020.].

[8] Unity Technologies (24.06.2020.) *Scene visibility* [Na internetu]. Dostupno: https://docs.unity3d.com/Manual/SceneVisibility.html [pristupano 24.06.2020.].

[9] Unity Technologies (24.06.2020.) *Positioning GameObjects* [Na internetu]. Dostupno: https://docs.unity3d.com/Manual/PositioningGameObjects.html [pristupano 24.06.2020.].

[10] Unity Technologies (24.06.2020.) *The Game view* [Na internetu]. Dostupno: https://docs.unity3d.com/Manual/GameView.html [pristupano 24.06.2020.].

[11] Unity Technologies (25.06.2020.) *The Inspector window* [Na internetu]. Dostupno: https://docs.unity3d.com/Manual/UsingTheInspector.html [pristupano 25.06.2020.].

[12] HowToMakeMobileGames (03.07.2020.) *How to add physics objects to unity*, Youtube [Video datoteka]. Dostupno: https://www.youtube.com/watch?v=ag95Ivs3M4s [pristupano 25.06.2020.].

[13] Unity Technologies (25.06.2020.) *Creating and Using Materials* [Na internetu]. Dostupno: https://docs.unity3d.com/Manual/Materials.html [pristupano 25.06.2020.].

[14] Unity Technologies (25.06.2020.) *Creating and Using Scripts* [Na internetu]. Dostupno: https://docs.unity3d.com/Manual/CreatingAndUsingScripts.html [pristupano 25.06.2020.].

[15] Unity Technologies (25.06.2020.) *Variables and the Inspector* [Na internetu]. Dostupno: https://docs.unity3d.com/Manual/VariablesAndTheInspector.html [pristupano 25.06.2020.].

[16] Unity Technologies (25.06.2020.) *Controlling GameObjects using components* [Na internetu]. Dostupno: https://docs.unity3d.com/Manual/ControllingGameObjectsComponents.html [pristupano 25.06.2020.].

[17] Unity Technologies (26.06.2020.) *Event Functions* [Na internetu]. Dostupno: https://docs.unity3d.com/Manual/EventFunctions.html [pristupano 26.06.2020.].

[18] Unity Technologies (26.06.2020.) Creating and Destroying GameObjects [Na internetu]. Dostupno: https://docs.unity3d.com/Manual/CreateDestroyObjects.html [pristupano 26.06.2020.].

[19] Unity Technologies (26.06.2020.) *Rigidbody overview* [Na internetu]. Dostupno: https://docs.unity3d.com/Manual/RigidbodiesOverview.html [pristupano 26.06.2020.].

[20] Unity Technologies (26.06.2020.) *Animation System Overview* [Na internetu]. Dostupno: https://docs.unity3d.com/Manual/AnimationOverview.html [pristupano 26.06.2020.].

[21] Unity Technologies (26.06.2020.) *Light Mode: Realtime* [Na internetu]. Dostupno: https://docs.unity3d.com/Manual/LightMode-Realtime.html [pristupano 26.06.2020.].

[22] Unity Technologies (26.06.2020.) *Light Mode: Baked* [Na internetu]. Dostupno: https://docs.unity3d.com/Manual/LightMode-Baked.html [pristupano 26.06.2020.].

[23] Unity Technologies (27.06.2020.) *Light Mode: Mixed* [Na internetu]. Dostupno: https://docs.unity3d.com/Manual/LightMode-Mixed.html [pristupano 27.06.2020.].

[24] Unity Technologies (27.06.2020.) *Types of light* [Na internetu]. Dostupno: https://docs.unity3d.com/Manual/Lighting.html [pristupano 27.06.2020.].

[25] Unity Technologies (27.06.2020.) *Audio Overview* [Na internetu]. Dostupno: https://docs.unity3d.com/Manual/AudioOverview.html [pristupano 27.06.2020.].

[26] LoveToKnow, Corp. (bez dat.) *survival-horror* [Na internetu]. Dostupno: https://www.yourdictionary.com/survival-horror [pristupano 27.06.2020.].

[27] Jason Ashman (25.10.2015.) *What Makes a Survival Horror Game?* [Na internetu]. Dostupno: https://techraptor.net/gaming/opinion/what-makes-survival-horror-game [pristupano 27.06.2020.].

[28] World of Level Design LLC by Alex Galuzin (29.06.2009.) *Horror/Survival Level Design: Part 2 – Anticipation* [Na internetu]. Dostupno: https://www.worldofleveldesign.com/categories/level_design_tutorials/horror-fear-level-design/part2-survival-horror-level-design-anticipation-pacing.php [pristupano 27.06.2020.].

[29] Lauren Relph (22.01.2017.) *Resident Evil Recap — the entire story explained and summarized* [Na internetu]. Dostupno: https://www.windowscentral.com/recap-resident-evil-story [pristupano 27.06.2020.].

[30] Game Wisdom (14.07.2018.) *How Resident Evil 1 Remastered Improved Horror Game Design,* Youtube [Video datoteka]. Dostupno: https://www.youtube.com/watch?v=PXNeogvH0f4 [pristupano 27.06.2020.].

[31] Keith Stuart (31.01.2019.) *Silent Hill at 20: the game that taught us to fear ourselves* [Na internetu]. Dostupno: https://www.theguardian.com/games/2019/jan/31/silent-hill-at-20-the-game-that-taught-us-to-fear-ourselves [pristupano 27.06.2020.].

[32] Sykoo (14.07.2019.) *How to make Terrains with Unity 2019! (Tutorial),* Youtube [Video datoteka]. Dostupno: https://www.youtube.com/watch?v=2Vvwjfp-hg8 [pristupano 27.06.2020.].

[33] Brackeys (27.10.2019.) *FIRST PERSON MOVEMENT in Unity - FPS Controller,* Youtube [Video datoteka]. Dostupno: https://www.youtube.com/watch?v=_QajrabyTJc [pristupano 27.06.2020.].

[34] Slobodni pad (bez dat.) u Wikipedia, the Free Encyclopedia. Dostupno: https://hr.wikipedia.org/wiki/Slobodni_pad [pristupano 27.06.2020.].

[35] ExDeaDguY (02.08.2019.) *Detecting terrain texture at position* [Na internetu]. Dostupno: https://forum.unity.com/threads/detecting-terrain-texture-at-position.94723/ [pristupano 27.06.2020.].

[36] Romi Fauzi (06.10.2019.) *Underwater FX with Unity 2019,* Youtube [Video datoteka]. Dostupno: https://www.youtube.com/watch?v=zmDDR9mOVTs [pristupano 27.06.2020.].

[37] Jimmy Vegas (31.10.2017.) *How To Make A Survival Horror Game - Unity Tutorial 005 - UI / HUD,* Youtube [Video datoteka]. Dostupno: https://www.youtube.com/watch?v=mhJwrcpXxUM&list=PLZ1b66Z1KFKiaTYwyayb8-L7D6bdiaHzc&index=5 [pristupano 27.06.2020.].

[38] Jimmy Vegas (10.11.2017.) *How To Make A Survival Horror Game - Unity Tutorial 006 - RAYCAST & SOUND,* Youtube [Video datoteka]. Dostupno: https://www.youtube.com/watch?v=HhFoFWjR2WA&list=PLZ1b66Z1KFKiaTYwyayb8-L7D6bdiaHzc&index=6 [pristupano 27.06.2020.].

[39] Jimmy Vegas (29.01.2018.) *How To Make A Survival Horror Game - Unity Tutorial 009 - WEAPONS,* Youtube [Video datoteka]. Dostupno: https://www.youtube.com/watch?v=s3WW1lZICuQ&list=PLZ1b66Z1KFKiaTYwyayb8-L7D6bdiaHzc&index=9 [pristupano 27.06.2020.].

[40] Jimmy Vegas (20.04.2018.) *How To Make A Survival Horror Game - Unity Tutorial 012 - AI & HEALTH,* Youtube [Video datoteka]. Dostupno: https://www.youtube.com/watch?v=wC8hDGdA7pc&list=PLZ1b66Z1KFKiaTYwyayb8-L7D6bdiaHzc&index=12 [pristupano 27.06.2020.].

[41] Jimmy Vegas (07.05.2018.) *How To Make A Survival Horror Game - Unity Tutorial 013 - SEQUENCING,* Youtube [Video datoteka]. Dostupno: https://www.youtube.com/watch?v=sOJ4NwAhUII&list=PLZ1b66Z1KFKiaTYwyayb8-L7D6bdiaHzc&index=13 [pristupano 27.06.2020.].

[42] Brackeys (09.02.2020.) *How to make a HEALTH BAR in Unity!,* Youtube [Video datoteka]. Dostupno: https://www.youtube.com/watch?v=BLfNP4Sc_iA [pristupano 27.06.2020.].

[43] Jimmy Vegas (23.10.2019.) *How To Make A Survival Horror Game - Unity Tutorial 030 - GLOWING OBJECTS + KEYS,* Youtube [Video datoteka]. Dostupno: https://www.youtube.com/watch?v=wNIpyJ_x3qk&list=PLZ1b66Z1KFKiaTYwyayb8-L7D6bdiaHzc&index=31 [pristupano 27.06.2020.].

[44] Jimmy Vegas (27.10.2018.) *How To Make A Survival Horror Game - Unity Tutorial 014 - GAME OVER,* Youtube [Video datoteka]. Dostupno: https://www.youtube.com/watch?v=14xAyMToVZo&list=PLZ1b66Z1KFKiaTYwyayb8-L7D6bdiaHzc&index=14 [pristupano 27.06.2020.].

[45] Jimmy Vegas (06.01.2019.) *How To Make A Survival Horror Game - Unity Tutorial 020 - MAIN MENU CREATION,* Youtube [Video datoteka]. Dostupno: https://www.youtube.com/watch?v=kfPo_t6frXY&list=PLZ1b66Z1KFKiaTYwyayb8-L7D6bdiaHzc&index=21 [pristupano 27.06.2020.].

[46] Jimmy Vegas (27.01.2019.) *How To Make A Survival Horror Game - Unity Tutorial 021 - SPLASH SCREEN | CREEPY MENU | LINKING,* Youtube [Video datoteka]. Dostupno: https://www.youtube.com/watch?v=yxHqErWhVew&list=PLZ1b66Z1KFKiaTYwyayb8-L7D6bdiaHzc&index=22 [pristupano 27.06.2020.].

[47] CodewithFin (15.04.2016.) *Controlling Master Volume with one slider?* [Na internetu]. Dostupno: https://answers.unity.com/questions/1171611/controlling-master-volume-with-one-slider.html [pristupano 27.06.2020.].

[48] miralong (07.09.2019.) *Unity, get the "actual" current Terrain?* [Na internetu]. Dostupno: https://stackoverflow.com/questions/52345522/unity-get-the-actual-current-terrain [pristupano 27.06.2020.].

6. Popis slika

7. Korišteni resursi

1. KBH Toon Skeleton, Team Hushkal Studios, Dostupno: https://assetstore.unity.com/packages/3d/characters/kbh-toon-skeleton-36700
2. Rust Key, Alexn09, Dostupno: https://assetstore.unity.com/packages/3d/props/rust-key-167590
3. Crowbar, RRFreelance / PiXelBurner, Dostupno: https://assetstore.unity.com/packages/3d/props/tools/crowbar-20500
4. Grave 18, luchin192, Dostupno: https://assetstore.unity.com/packages/3d/environments/grave-18-81974
5. Cathedral and Cemetery Kit, Aquarius Max, Dostupno: https://assetstore.unity.com/packages/3d/environments/dungeons/cathedral-and-cemetery-kit-29240
6. Abandoned buildings, Aleksey Kozhemyakin, Dostupno: https://assetstore.unity.com/packages/3d/environments/abandoned-buildings-62875
7. AllSky Free - 10 Sky / Skybox Set, rpgwhitelock, Dostupno: https://assetstore.unity.com/packages/2d/textures-materials/sky/allsky-free-10-sky-skybox-set-146014
8. Classic Interior Door Pack 1, Jan Fidler, Dostupno: https://assetstore.unity.com/packages/3d/props/interior/classic-interior-door-pack-1-118744
9. Tileable Bricks Wall, Game-Ready Studios, Dostupno: https://assetstore.unity.com/packages/2d/textures-materials/brick/tileable-bricks-wall-24530
10. Zombie, Pxltiger, Dostupno: https://assetstore.unity.com/packages/3d/characters/humanoids/zombie-30232
11. Flashlight PRO, Indie_G, Dostupno: https://assetstore.unity.com/packages/3d/props/tools/flashlight-pro-53053
12. [PBR] Pistol, Game-Ready Studios, Dostupno: https://assetstore.unity.com/packages/3d/props/guns/pbr-pistol-33838
13. Rock and Boulders 2, Manufactura K4, Dostupno: https://assetstore.unity.com/packages/3d/props/exterior/rock-and-boulders-2-6947
14. Chainlink Fences, Kobra Game Studios, Dostupno: https://assetstore.unity.com/packages/3d/chainlink-fences-73107

15. The Shed, Blackant Master Studio, Dostupno: https://assetstore.unity.com/packages/3d/environments/urban/the-shed-10303
16. Terrain Tools Sample Asset Pack, Unity Technologies, Dostupno: https://assetstore.unity.com/packages/2d/textures-materials/nature/terrain-tools-sample-asset-pack-145808
17. Standard Assets (for Unity 2018.4), Unity Technologies, Dostupno: https://assetstore.unity.com/packages/essentials/asset-packs/standard-assets-for-unity-2018-4-32351