Izrada trodimenzionalne igre istraživanja u programskom alatu Unity

Majcen, Luka

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Luka Majcen

IZRADA TRODIMENZIONALNE IGRE ISTRAŽIVANJA U PROGRAMSKOM ALATU UNITY

ZAVRŠNI RAD

Varaždin, 2021.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Luka Majcen

Matični broj: 45692/16-I

Studij: Primjena informacijske tehnologije u poslovanju

IZRADA TRODIMENZIONALNE IGRE ISTRAŽIVANJA U PROGRAMSKOM ALATU UNITY

ZAVRŠNI RAD

Mentor/Mentorica:

Doc. dr. sc. Mladen Konecki

Varaždin, lipanj 2021

Luka Majcen

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

Ovim završnim radom prikazat će se izrada jednostavne trodimenzionalne igre u programskom alatu Unity. Uz samu izradu cilj je obuhvatiti osnovni set alata, dodati glazbu u igru te prikazati pravila igre korisniku koji će samu i igrati. Uz programski alat Unity koristit će se specijalizirani softver za glazbu - Reaper. Izrada video-igara je poprilično kompleksan proces te ćemo to i prikazati.

Ključne riječi: Unity; igra; softver; Reaper; 3D; glazba

Sadržaj

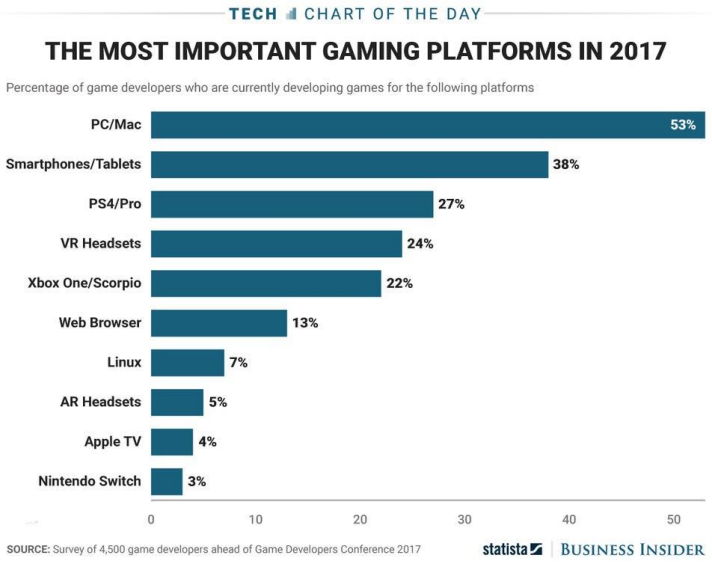
1. Uvod

Danas su video-igre vrlo popularan način zabave za mlade, ali i starije osobe. Postoji mnoštvo žanrova i vrsta igara pa korisnici vrlo lagano mogu izabrati željenu igru.

Neki od žanrova igara su akcijske, RPG (*Role-playing game*), sportske, avanture, strategije, utrke. Bitno je reći da sada skoro svaka osoba na svijetu ima pristup video-igrama te ogromnu paletu različitih žanrova, a uz to mogu se igrati i igrice koje nije potrebno platiti (*Free-To-Play*). [1]

Video-igre izrađuju softverske tvrtke koje se obično specifično time bave, a veličine tih tvrtki mogu poprilično varirati (npr. tim može biti samo jedna osoba pa sve do par tisuća ljudi koji su uključeni u proces izrade). Izrada igara može biti i poprilično skupa jer osim samog tima koji programira/izrađuje igru postoje i timovi koji su zaduženi za neke popratne stvari u igrama kao glazba, film, grafički dizajn, gluma te glas preko kadra *(voice-over*). Video-igre su nekada bile najviše zastupljene na konzolama, ali danas imamo mnogo više platformi i mogućnosti za igranje. [2]

Osim popularnih konzola poput PlayStation-a (PS1, PS2, PS3, PS4 i PS5) i Xbox-a (Xbox, Xbox 360, Xbox One, Xbox Series X), sada se za gaming koristi i PC (*Personal Computer*) te danas najviše popularni – mobilni telefoni. [3]



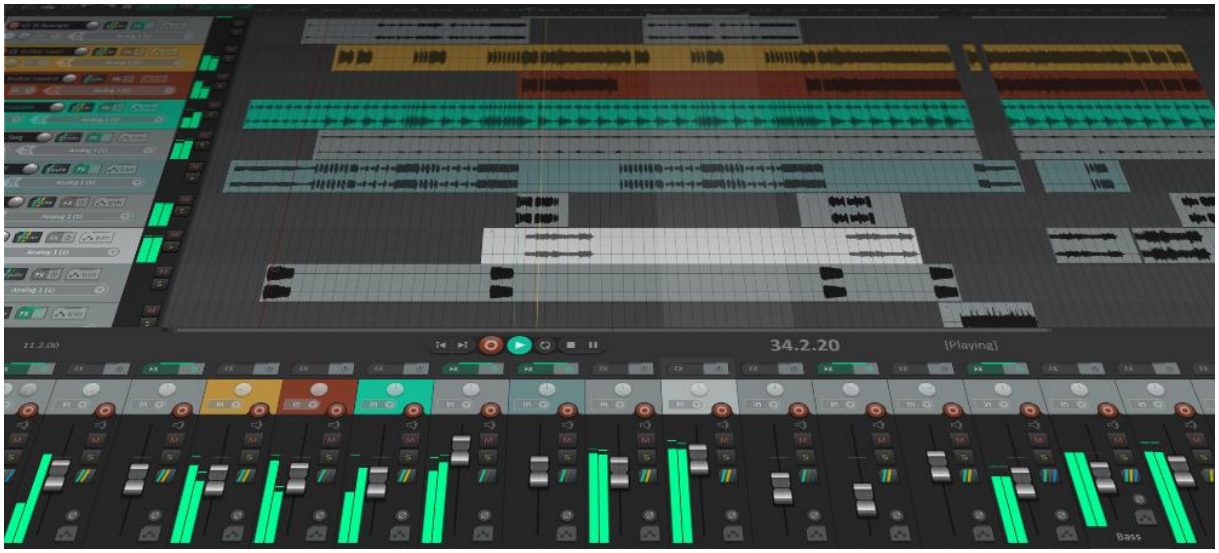
Slika 1. Najpopularnije platforme za igranje igrica 2017. godine (Izvor: Business Insider, 2017.)

U svom uratku ću prikazati izradu jednostavne 3D igre sa prije spomenutim dodatnim elementom glazbe.

2. Alati za izradu video igre

Za potrebe izrade moje video-igre koristit ću Unity za izradu same igre dok ću za kompoziciju, miksanje i masteriranje glazbe koristiti Reaper.

Slika 2. Unity (Izvor: Unity3D, bez dat.)



Slika 3. Reaper (Izvor: Reaper FM, bez dat.)

Osim što video-igra i pjesma moraju biti tehnički ispravne, moraju se i kreativno slagati (mirnija glazba će više odgovarati na sporije igre dok će brža i moćnija glazba više odgovarati pucačinama i igrama gdje korisnik treba brže donositi odluke). Nakon završetka izrade pjesme u Reaperu, ona će se ubaciti u Unity projekt gdje se nalazi video-igra. Svaki proces će biti veoma slikovito prikazan i objašnjen.

2.1. Unity

2.1.1. Što je Unity?

Unity su u Kopenhagenu 2004. godine osnovali Nicholas Francis, Joachim Ante i David Helgason. Njegova je priča započela na OpenGL forumu u svibnju 2002. godine gdje je Francis objavio poziv suradnicima na „open source shader-compileru“ (grafički alat) za populaciju programera igara zasnovanih na Macu.. Odazvao se Ante, tada srednjoškolac u Berlinu. Joachim Ante odlično je komplementirao Francisov fokus na grafiku i gameplay sa vrlo intuitivnim smislom za back-end arhitekturu. Uskoro im se pridružuje i Helgason koji je, kako ga oni vide, više poslovno orijentiran i bolji u sklapanju dogovora. Zbog toga Helgason postaje CEO (glavni izvršni direktor). Koristeći svoje štednje, novac koji je posudio otac Joachima Antea (25 000 eura) te prihod iz Helgasonovog honorarnog posla, trojac osniva kompaniju po imenu Over The Edge Entertainment (OTEE). Prvih nekoliko mjeseci uložili su u izradu igre GooBall no nakon samog launcha, video-igra nije zaživjela jer je bila preteška za igranje no nakon toga odlučuju da se žele fokusirati na game engine. [4]

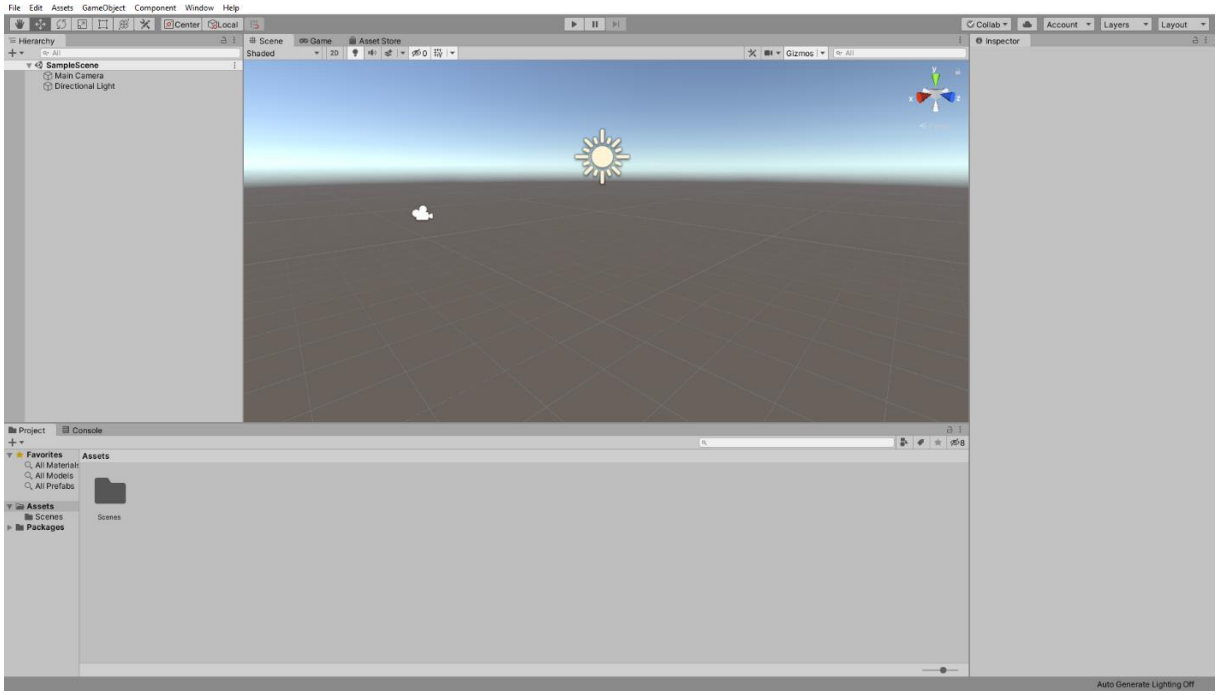
Unity (još poznat kao i Unity 3D) je IDE (Integrirano razvojno okruženje) i game engine (motor igre) napravljen od strane Unity Technologies-a za stvaranje interaktivnog sadržaja, a za moj projekt koristit će se u svrhu izrade 3D video-igre. Unity je prvi puta pušten u javnost u lipnju 2005. godine, a do danas je dobio mnogo različitih softverskih nadogradnji i zakrpa.

Osim za izradu 3D igara, Unity se može koristiti i za izradu 2D i VR video-igara. Unity danas broji oko 1,5 milijun aktivnih korisnika. 2018. godine objavljeno je da je u Unity okruženju napravljeno oko 60-70% svih mobilnih igara. [5]

Slika 4. Kreiranje igara za mobitele u Unity-u (Izvor: YouTube, 2019.)

2.1.2. Korisničko sučelje Unity-a

Nakon prvog pokretanja Unity programa za izradu video-igara program će pokrenuti tvornički zadan poredak komponenata i prozora no korisnik ima mogućnost personalizirati ih po svojim zahtjevima. Za razliku od nekih drugih game engine-a Unity je vrlo intuitivan i jednostavan za korištenje.

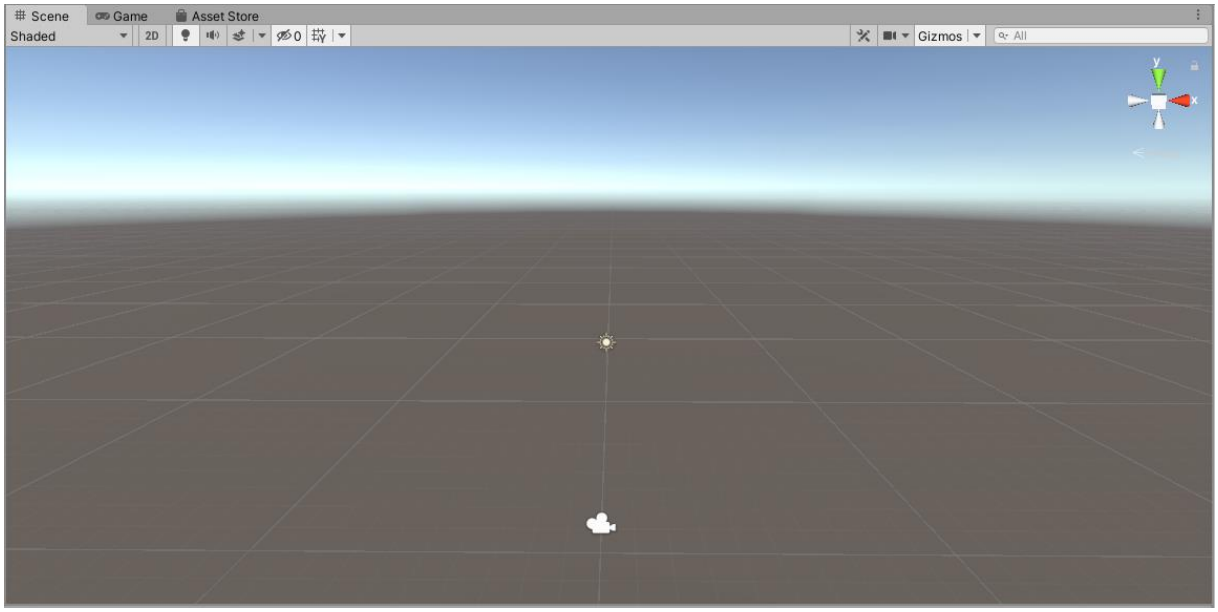


Slika 5. Prvo pokretanje programa

Po zadanim postavkama Unity nam daje nekoliko prozora od kojih svaki ima neku dodijeljenu funkciju. Na zaglavlju programa nalazi se glavni meni koji unutar svakog posebnog izbornika ima i podizbornik za različite funkcije: spremanje projekta, pokretanje novog projekta, personalizacija i poredak prozora, pomoć u korištenju programa i slično. Imamo i vrlo bitan izbornik *Component* u kojem se nalaze podizbornici za efekte, renderiranje, mesheve, fiziku i još mnogi drugi.

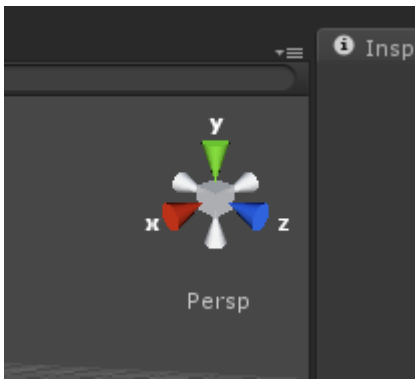
Slika 6. Glavni meni

U glavnom meniju nalazimo još neke alate koji nam omogućavaju različite funkcije unutar glavnog prozora poput *Hand-Tool-a* za „pomicanje rukom“ u 3D prikazu, *Rotate-Tool-a* za rotiranje scene te alat za upravljanje objektima (*Move, Rotate and Scale selected objects*).



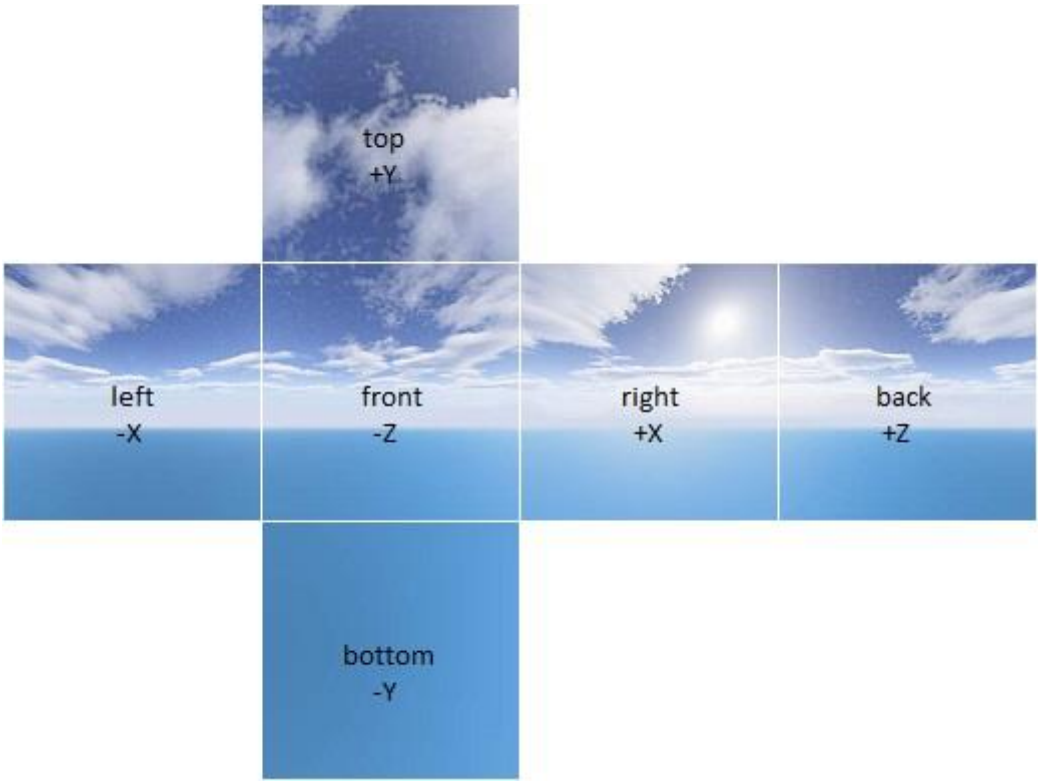
Slika 7. Glavni prozor – Scene

Glavni prozor u programu interaktivni je prikaz svijeta koji se izrađuje. Scena nudi nekoliko različitih perspektiva pregleda svijeta. Nudi pregled svih objekata, likova, kamera i svjetla koje se koristi u svijetu. Scena je prikazana u 3D obliku te sadrži grid za lakšu navigaciju po osima. Osim grida tu je i *Scene Gizmo* koji prikazuje trenutnu orijentaciju i zapravo olakšava korisniku da brže promijeni kut gledanja i projekciju.



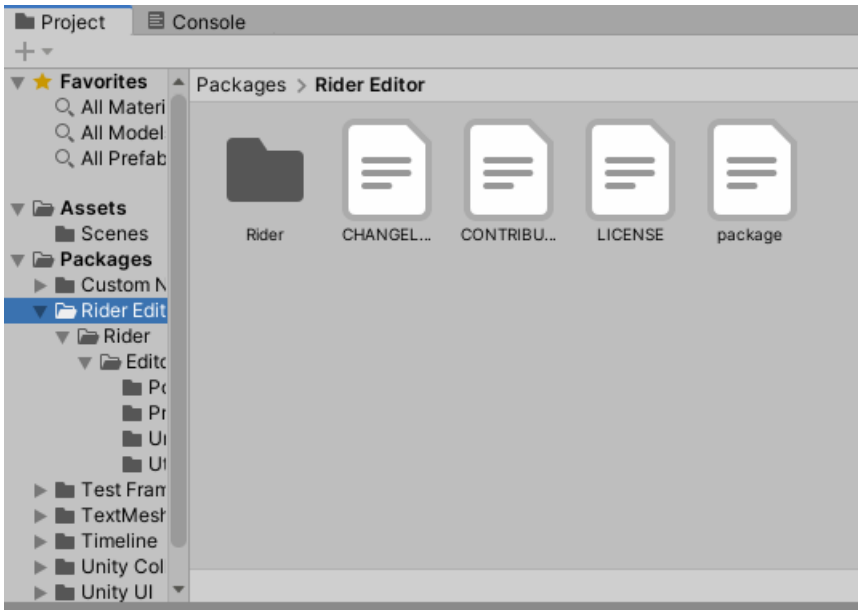
Slika 8. Gizmo

U prozoru se nalazi i oznaka sunca koje predstavlja izvor svjetla u našem svijetu. Prikazuje sa koje strane će svjetlost obasjati objekt, svijet ili nekog specifičnog lika. Svaki obasjan lik ili objekt imat će svoju sjenu. Scena prikazuje i Skybox materijal. Skybox materijal nalazi se u pozadini iza svih objekata. Skybox po zadanoj vrijednosti prikazuje nebo, ali sa interneta se može preuzeti mnoštvo tekstura i gotovih Skybox-eva koji se mogu ukomponirati u korisničke projekte (npr. noćno nebo sa zvijezdama). Skybox se sastoji od šest strana od kojih svaka predstavlja jednu od osi (X, Y, Z) sa negativnim ili pozitivnim prefiksom (+ ili -).



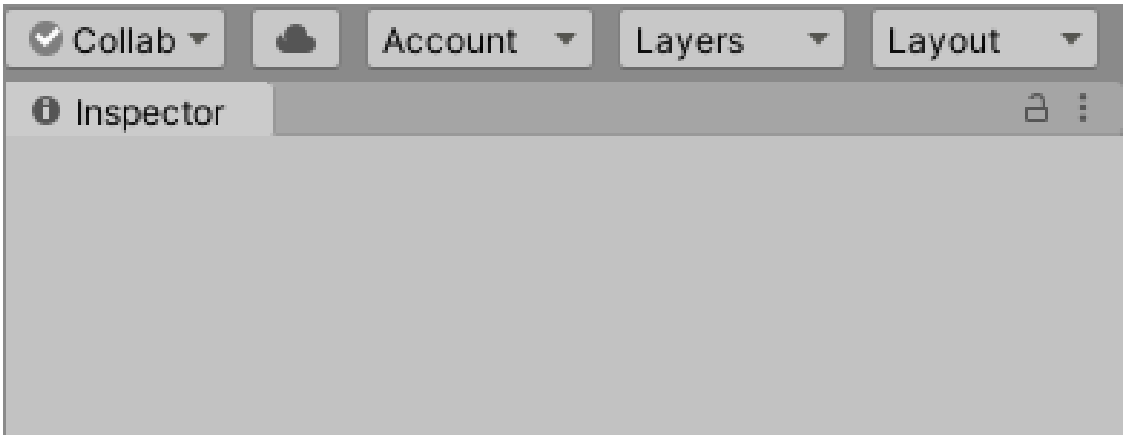
Slika 9. Skybox (Izvor: Packt Subscripton, bez dat.)

Unity sadrži i *Project prozor* u kojem se prikazuju datoteke povezane sa aktualnim projektom. Nudi pregled svih *Asset-a* i modela koji se koriste u projektu.



Slika 10. Project prozor

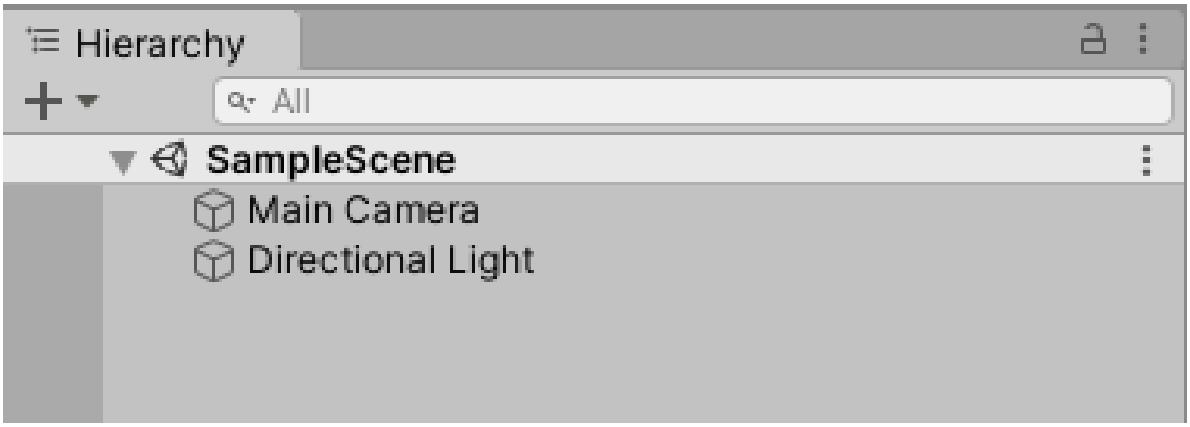
Posljednja dva prozora koja se nalaze na sučelju programa i pomoći će nam kod izrade projekta su *Inspector* i *Hiearchy* prozor. *Inspector* je jedna od najvažnijih komponenata Unity-a jer nam koristi za prikaz svojstava i postavki nekog objekta, komponente, materijala ili asset-a te se koristi za uređivanje istih.



Slika 11. Inspector

Hierarchy prikazuje sve aktualne objekte koji se nalaze na sceni i njihovu hijerarhijsku strukturu. Koncept koji se koristi u programu Unity naziva se roditelj-dijete princip. Roditelj objekt kojem se svojstva modificiraju predaje svojstva svojoj djeci (npr. promjena rotacije ili kretanja roditelj objekta primijeniti će se i na njegovu djecu objekte). Isto tako princip roditelj-dijete možemo i koristi za gašenje vidljivosti. Ako isključimo vidljivost jednog djeteta to neće

utjecati na ostalu djecu ili roditelja već isključivo odabrano dijete, a u slučaju da isključimo vidljivost roditelja nestat će i roditelj i djeca.



Slika 12. Hierarchy

2.2. Reaper

2.2.1. Što je Reaper?

Reaper je DAW (Digital Audio Workstation) za izradu glazbe i glazbenu produkciju. Reaper je zapravo akronim za „Rapid Environment for Audio Production, Engineering and Recording“. Tvrtku Cockos koja je proizvela ovaj DAW, osnovao je Justin Frankel u San Franciscu. [6]

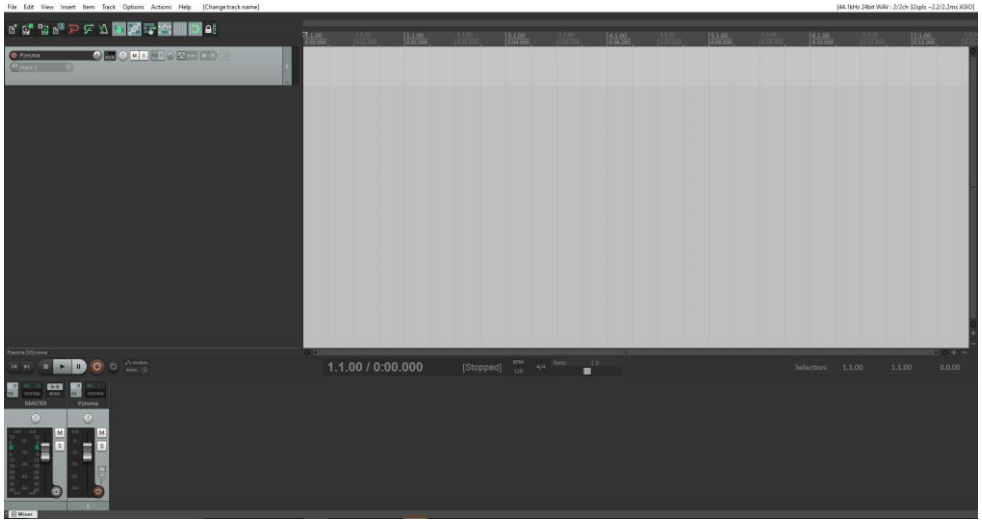
Snimanje i produkcija glazbe u posljednjem su desetljeću postali mnogo jeftiniji i pristupačniji korisnicima diljem svijeta. Analogno studijsko snimanje na magnetne trake zamijenile su zvučne kartice i digitalne audio postaje (DAW) kojima korisnici vrlo brzo i intuitivno mogu snimati iz udobnosti svog doma. Reaper, osim što služi za direktno snimanje materijala u trake, služi i za njihovo miksanje. Miksanje glazbe proces je obrade sirovih traka kako bi odgovarale jedna drugoj u cjelini i za otklanjanje nekih mogućih problema koji su nastali snimanjem materijala. Miksanje uključuje korištenje plug-ina poput EQ-a (*ekvilajzer*) kojim određene frekvencije možemo pojačati ili smanjiti *(boosting* or *cutting)*, kompresije kojom smanjujemo dinamički raspon određene trake (izjednačavamo tiše i glasnije dijelove) te dodatne obrade i simulacije analognog zvuka u digitalnom obliku. Plug-inovi su obično besplatan softverski paket koji se vrlo lako skida i ugrađuje u DAW, ali neki kompleksniji i jači plug-inovi su podosta skupi. Isto tako imamo i potpunu kontrolu nad razinom volume-a određene trake kojom određujemo hoće li neka traka biti ukupno glasnija od ostalih ili tiša od ostalih. Ovaj se proces poprilično razlikuje ovisno o instrumentu koji koristimo za snimanje.

Gitara, bas-gitara, sintetizator zvuka, vokali ili bubanj imaju različite frekvencije pa se tako za svaki od instrumenata mora pristupati drukčije.

Slika 13. Miksanje glazbe (Izvor: Izotope, 2018)

2.2.2. Korisničko sučelje Reaper-a

Prvo pokretanje Reaper-a slično je Unity programu – pokreće se nekoliko manjih prozora od kojih svaki ima određenu funkciju. Reaper sadrži vremensku liniju gdje se snimaju trake i dostupni su vremenski podaci o njima, mikser gdje je najlakše i najbrže kontrolirati glasnoću pojedinih traka te imamo prozor gdje za pojedine trake možemo aktivirati plug-ine, organizirati trake i kreirati roditelj-dijete odnos (*buss trake*).



Slika 14. Reaper - prvo pokretanje

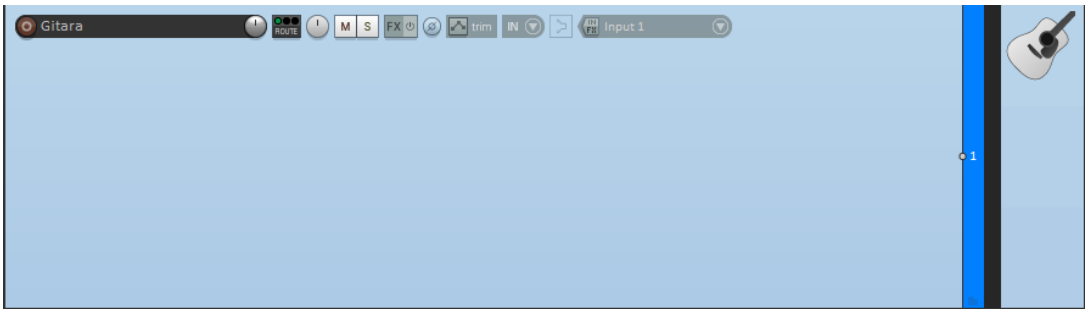
Vremenska linija odvojena je crtama koje predstavljaju brzinu udarca metronoma (*beats per minute*) te ovisno o brzini te će crte biti gušće ili rjeđe. Vremenska linija prikazuje i duljinu svake trake u satima, minutama i sekundama. Na prvom pokretanju Reaper će nativno postaviti četveročetvrtinski takt odnosno četiri četvrtinke u svakom taktu, a na korisniku je da tu mjeru može postaviti ovisno o tome kakvu vrstu glazbe radi.

Slika 15. Vremenska linija

Mikser donosi jednostavnost upravljanja glasnoćom pojedinih traka u projektu. Mikser popisuje sve trenutne trake i nudi klizače (*fader*) koji se pomiču mišem. Također sadrži i brzu opciju za upravljanje pomakom zvuka (*panning*) kojim traku možemo pomaknuti u lijevo ili desno uho ili u centar stereo polja. Mikser donosi i gumbić za gašenje trake (*mute*) i izoliranje trake (*solo*). Mikser ima opciju kojom ga možemo odvojiti od programa i koristiti na nekom drugom izlaznom uređaju za prikazivanje slike (još jedan monitor ili televizor).

Slika 16. Mikser

U programu postoji i drugi mikser koji se nalazi odmah pokraj vremenske linije. On omogućuje jednostavniji i intuitivniji pregled traka jer je u ravnini sa već snimljenim trakama. Ima slične opcije kao i običan mikser, ali se razlikuju po tome što mikser pokraj vremenske linije nudi opcije poput invertiranja faze, bržeg spajanja ulaznog uređaja sa digitalnom audio stanicom (poput sintetizatora ili gitare koji se spajaju na zvučnu karticu) i opciju monitoring-a na izlazni uređaj. Također nudi opciju promjene imena trake, odabir personalizirane boje za traku (postoji i opcija za odabir boje RGB formatu) i dodavanje malih ikonica koje pomažu kod raspoznavanja instrumenata u slučaju velikog broja traka u projektu.



Slika 17.Mikser pokraj vremenske linije

3. Izrada - Unity

3.1. Žanr

Igra koja će se izraditi i dokumentirati pripada žanru psihološke avanture. Psihološka avantura igra je koja nema mnogo interakcije između lika i svijeta već se sama radnja odvija kroz ostale oblike umjetnosti poput glazbe i filma. Mnogo puta se u ovaj žanr uključuje voice-over koji služi za monologe glavnih likova.

Psihološka avantura ima nekoliko izvrsnih predstavnika žanra:

1. Dear Esther

2. Eleusis

3. Call of Cthulhu

Dear Esther igra je u prvom licu koja priča o ljubavi, gubitku, krivnji i iskupljenju. Igra je započela kao kultni mod za Half-Life 2 2007. godine. Dear Esther, uz predivnu grafiku, pomiče granice u dizajnu igara na način da u svoj svijet uključuje pripovijedanje. Kombinacija navedenog donosi melankolično i beskompromisno iskustvo. [6]

Call of Cthulhu uzima inspiraciju iz *„The Call of Cthulhu“,* pisca H.P. Lovecraft-a te dodaje mračnu atmosferu i elemente horora, a Eleusis je igra naizgled slična Dear Esther-u no donosi bolju vizualnu izvedbu sa zagonetkama i šuljanjem (stealth). Eleusis je igra koja ne sadrži mnogo interakcije i oružja, već poneko korištenje *item-a* i prilično kratko trajanje same igre. Igrač je smješten u Eleuzinu, ruralni dio Stare Grčke. Priča je bazirana na Eleuzinskim Misterijama. Eleuzinske Misterije bila su inicijacije koje su označavale ulazak u sektu i održavale su se u čast božice Demeter i Perzefone. Kroz cijelo trajanje igre osjetna je tenzija glavnog lika kojeg prati zabrinutost i zbunjenost. Igra zbog svoje jednostavnosti i usmjerenosti posebnoj publici *gamer-a* često nije dobila dobre recenzije. Eleusis je na tržište izbačen 2013. godine od strane *Nocturnal Works-a,* a izrađen je u *Unreal Engine-u.* Donosi složene slagalice koje polako odmataju pravo lice malo ruralnog mjesta.

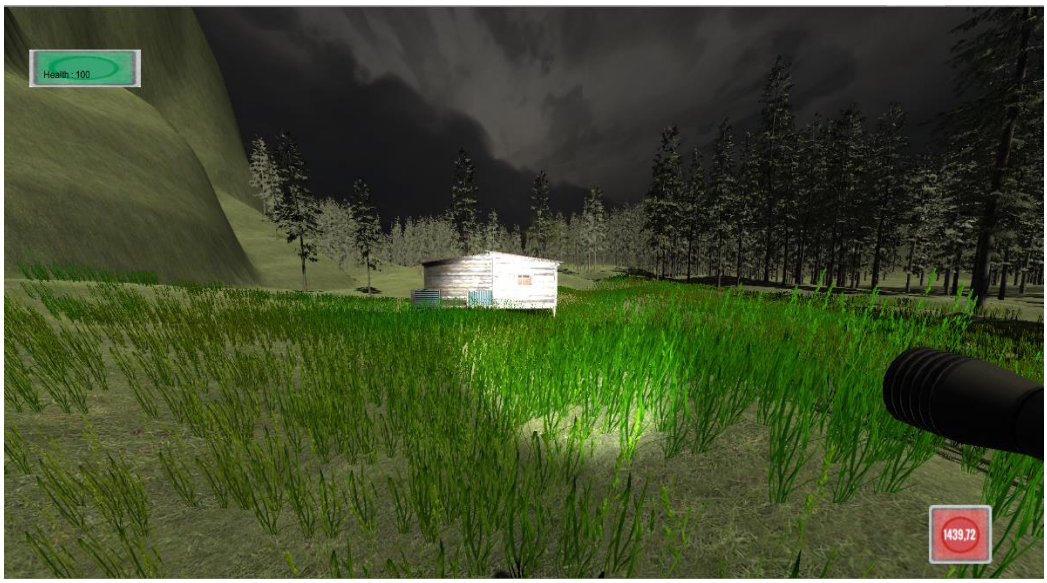
Slika 18. Igra Eleusis (Izvor: Gold-Plated Games, 2017.)

Slika 19. Dear Esther (Izvor: GoodgameHR, 2012.)

Slika 20. Call of Cthulhu (Izvor: Reboot.hr, 2018.)

3.2. Cilj igre

U našoj je igri glavni lik ubačen u nepoznat svijet u kojem vladaju čudovišta. Cilj igre je da igrač otvori kuću kod koje igra započinje i odgonetne tajnu gdje se zapravo nalazi. Igrač za pristup u kuću treba sakupiti brojeve koji su napisani na objekte koji su dio prošlosti glavnog lika, a raštrkani su po svijetu. Igrač pritom mora paziti na stvorenja, koja ako im se previše približi, napadaju i oštećuju životne bodove igrača. Neprijatelji se nalaze u *pack-ovima* te mogu podići igrača. Igrač može pobjeći neprijateljima ako im se dovoljno udalji. Igrač ima mogućnost vraćanja životnih bodova skupljanjem napitka u obliku bačve sa srcem. Igrač može napitak pokupiti jedino ako ima oštećene životne bodove i ne može vratiti više životnih jedinica nego je to maksimalno zadano. Još jedna od mehanika je brojač koji tjera igrača da čim brže pronađe sve brojeve i otkrije istinu. U slučaju da je igrač poražen od strane neprijatelja ili je istekao brojač, igra će se automatski ponovno pokrenuti i postaviti igrača na početnu točku. Zbog tame igrač ima svjetiljku koju može uključiti i isključiti pritiskom na tipku „*F*“. Igru prati hladna i atmosferična glazba koja sadrži poneki monolog glavnog lika. Nakon korištenja skripte za otključavanje vrata, igrač saznaje svoju pravu sudbinu.



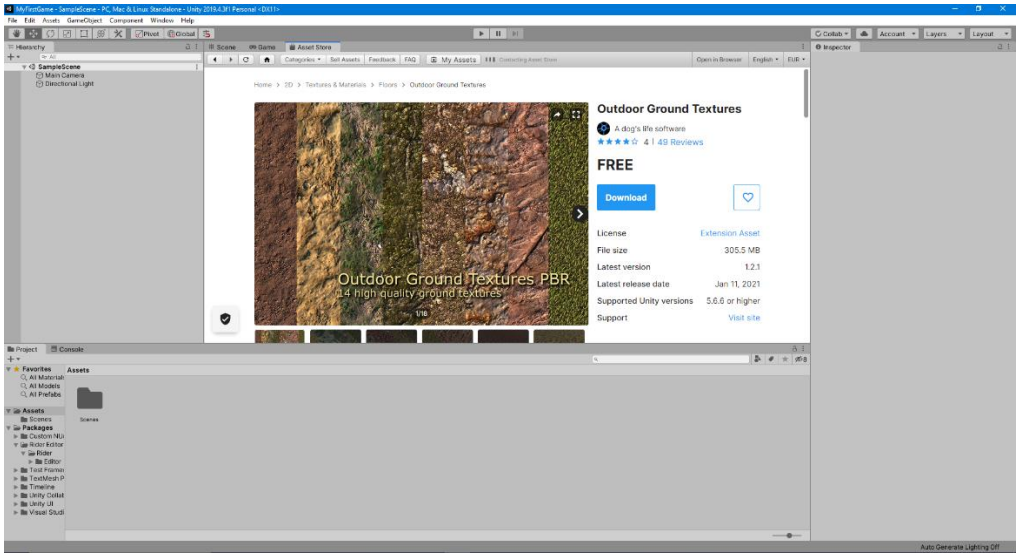
Slika 21. Igra

3.3. Izrada svijeta

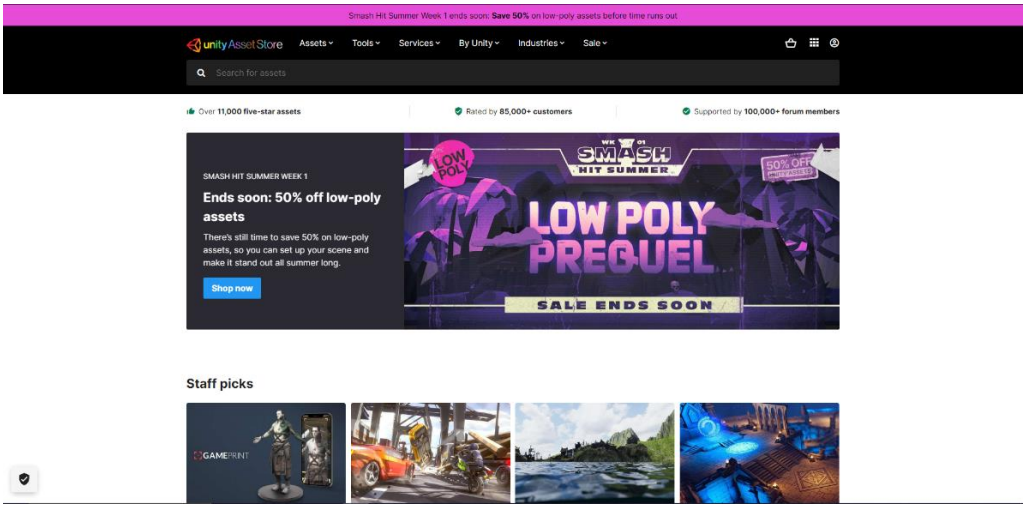
Izradu svijeta započet ćemo sa skidanjem tekstura i modela sa *Unity Asset Store-a*. Tekstura je kvaliteta neke površine i u svijetu video-igara nekom poligonu ili modelu daje vizualni izgled. *Unity Asset Store* nudi mnoštvo besplatnih i plaćenih proizvoda no za potrebe

našeg projekta koristit ćemo isključivo besplatne proizvode. U web dućanu prvo odabiremo teksturu ili model koji želimo skidati, a nakon toga proizvod moramo skinuti na naše računalo i sinkronizirati ga sa Unity okruženjem.

Objekti i teksture veće rezolucije i kvalitete igraču daju osjećaj realizma i imerzivnog igranja video-igara. Takve teksture nazivaju se *Photo Realism Textures*. Za projekt nam je potrebno mnogo vrsta tekstura i modela (npr. drveće) koje ćemo preuzeti sa Unity Asset Store-a.



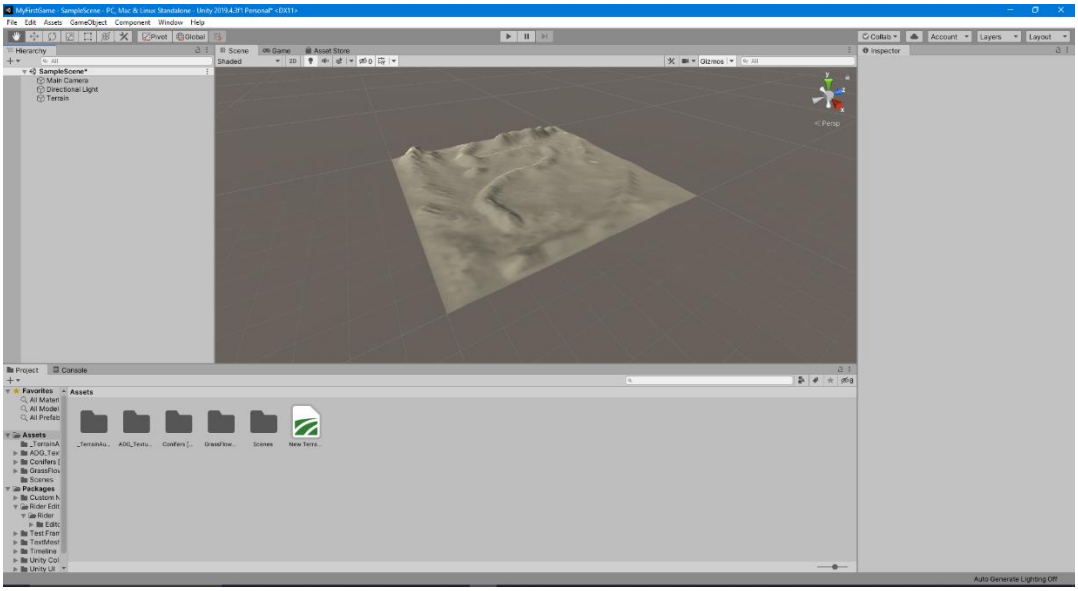
Slika 22. Skidanje tekstura



Slika 23. Skidanje tekstura na web-u

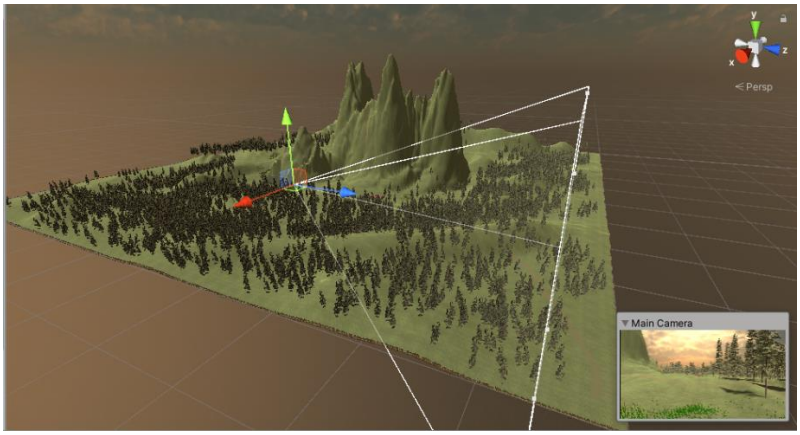
3.3.1. Teren

Prvi korak u izradi video-igre je postavljanje terena na koji će se stavljati svi ostali objekti. Za teren sam odabrao veliku plohu u obliku četverokuta. Teren je u početku ravan no Unity nam nudi opcije za podizanje i spuštanje terena kojim se kreiraju brda i planine, a isto tako i udubine poput jama i špilja i mjesta kamo možemo staviti vodu.



Slika 24. Teren

Veličina našeg svijeta je jedan kvadratni kilometar (Unity za mjernu jedinicu duljine koristi metar). Svijet će biti ispunjen vegetacijom, neravninama i objektima bitnim za odvijanje radnje u igri. Svijet je obojan teksturom zelene boje sa ponekim iscrtanim putem smeđe boje. Stavljen je novi *skybox* koji ima tamnije nebo i mnogo manju ekspoziciju usporedbi sa starim svijetlim koji prikazuje plavo nebo.



Slika 25. Gotov svijet

3.3.2. Objekti

Objekti i modeli u video-igrama obično služe za interakciju s glavnim likom ili da popune atmosferu ovisno o svijetu u kojem se lik nalazi. Objekte i modele za izrađen projekt moguće je skinuti sa Unity Asset Store-a, a moguće je kreirati i u programima poput Autodesk Maya-e no to zahtijeva mnogo više vremena i iskustva.

Objekti mogu biti raznih veličina i tekstura, a za naš projekt korišteni su sljedeći objekti:

1. **Kuća**
2. **Ograda**
3. **Šupa**
4. **Stol**
5. **Kovčeg**
6. **Auto**
7. **Vrata**
8. **Bačva**
9. **Neprijatelji**
10. **Svjetiljka**

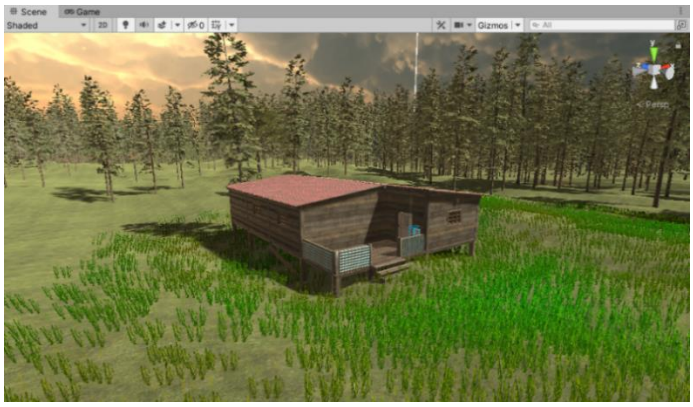
Vrata će na sebi sadržavati kod kojim će igrač morati pogoditi točnu kombinaciju brojeva kako bi nastavio igru. Igrač će pripadajuće brojeve pronalazi po kreiranom svijetu i kada pronađe sva četiri broja potrebna za stvaranje koda, igrač će doći do kuće i stati pred vrata te će mu se prikazati poruka za „otvaranje kutije“ u koju će se moći upisati kod.

Kod koji je bio potreban za interakciju pronađen je na internetu i izmijenjen je za potrebe projekta.

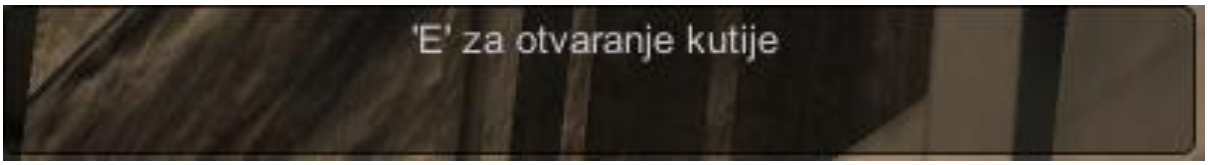
```
using UnityEngine;
using System.Collections;
public class Keypad : MonoBehaviour {
    public string curPassword = "2406";
    public string input;
    public bool onTrigger;
    public bool doorOpen;
    public bool keypadScreen;
    public Transform doorHinge;
    public GameObject bodka;
    public AudioSource Vrzganie;
    void OnTriggerEnter(Collider other) {
        onTrigger = true;}
    void OnTriggerExit(Collider other) {
        onTrigger = false;
        keypadScreen = false;
        input = "";}
    void Update() {
```

```
        if (Input.GetKeyDown(KeyCode.Alpha0) ||
Input.GetKeyDown(KeyCode.Keypad0)) {
            input = input + "0";}
        if (Input.GetKeyDown(KeyCode.Alpha1) ||
Input.GetKeyDown(KeyCode.Keypad1)) {
            input = input + "1";}
        if (Input.GetKeyDown(KeyCode.Alpha2) ||
Input.GetKeyDown(KeyCode.Keypad2)) {
            input = input + "2";}
        if (Input.GetKeyDown(KeyCode.Alpha3) ||
Input.GetKeyDown(KeyCode.Keypad3)) {
            input = input + "3";}
        if (Input.GetKeyDown(KeyCode.Alpha4) ||
Input.GetKeyDown(KeyCode.Keypad4)) {
            input = input + "4";}
        if (Input.GetKeyDown(KeyCode.Alpha5) ||
Input.GetKeyDown(KeyCode.Keypad5)) {
            input = input + "5";}
        if (Input.GetKeyDown(KeyCode.Alpha6) ||
Input.GetKeyDown(KeyCode.Keypad6)) {
            input = input + "6";}
        if (Input.GetKeyDown(KeyCode.Alpha7) ||
Input.GetKeyDown(KeyCode.Keypad7)) {
            input = input + "7";}
        if (Input.GetKeyDown(KeyCode.Alpha8) ||
Input.GetKeyDown(KeyCode.Keypad8)) {
            input = input + "8";}
        if (Input.GetKeyDown(KeyCode.Alpha9) ||
Input.GetKeyDown(KeyCode.Keypad9)) {
            input = input + "9";}
        if (input == curPassword) {
            doorOpen = true;}
        if (doorOpen) {
            var newRot = Quaternion.RotateTowards(doorHinge.rotation,
Quaternion.Euler(0.0f, 90.0f, 0.0f), Time.deltaTime * 250);
            doorHinge.rotation = newRot;
            bodka.SetActive(false);
            Vrzganie.Play();} }
    void OnGUI() {
        if (!doorOpen) {
            if (onTrigger) {
                GUI.Box(new Rect(0, 0, 400, 50), "'E' za otvaranje
kutije");
                if (Input.GetKeyDown(KeyCode.E)) {
                    keypadScreen = true;
                    onTrigger = false;} }
            if (keypadScreen) {
                GUI.Box(new Rect(0, 0, 640, 70), "");
                GUI.Box(new Rect(10, 10, 620, 50), input);
                GUI.Label(new Rect(10, 60, 200, 50), "'Esc' za izlaz");
                if (Input.GetKeyDown(KeyCode.Escape)) {
                    keypadScreen = false;} } } }
```

Navedena programska skripta omogućuje stavljanje lozinke na vrata koja koristimo u igri. Samu vrijednost lozinke postavljamo u *string* unutar programskog koda i u slučaju da igrač unese ispravnu lozinku odvija se skripta za otvaranje vrata. Igrač prvo dolazi do kutije koja ga informira da na pritisak tipke *'E'* pokreće sekvencu za upisivanje lozinke. Nakon toga otvara se prozor za unos koji je moguće ugasiti tipkom '*Esc*'.



Slika 26. Kuća



Slika 27. Otvaranje kutije za upis koda

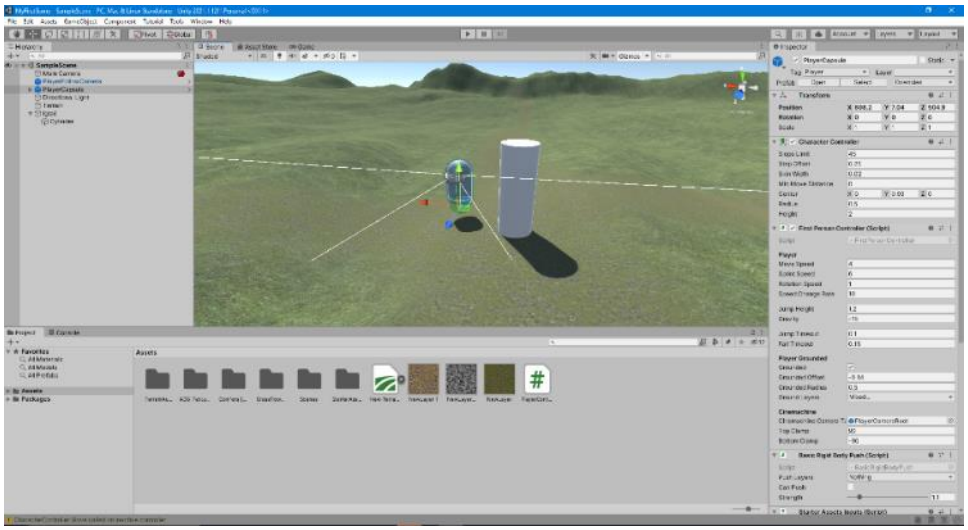
Slika 28. Auto

Slika 29. Kovčeg

Objekti poput auta i kovčega na sebi će sadržavati broj u obliku 3D teksta koji predstavlja dio koda za otključavanje vrata. Navedene modele prvo je potrebno skinuti sa Unity Store-a te ih učitati u program (slično kao i teksture) te ako model na sebi nema gotovu teksturu, ona se mora ručno dodati iz *Project* prozora. Upravo model auta koji je korišten u početku nije imao teksture već je bio prazan model, a u paketu s njim je stiglo nekoliko tekstura koje su se mogle ručno odabrati i odvući na model (boja prozora, boja guma i boja karoserije). Model kovčega ubacio se preko jednostavnog *drag and drop* načina.

3.3.3. Kamera

U Unity okruženju kamera je komponenta kroz koju igrač vidi svijet. Postoji nekoliko vrsta kamera no najbitnija kamera je kamera igrača koja slijedi kontroler igrača. Kontroler igrača zapravo pomiče kameru pomoću tipki za kretnju (*W, A, S, D*) te pogledima mišem – *MouseLook.* Unity u svojim projektnim datotekama ima gotovo rješenje za kontrolu igrača, a moguće je birati između kontrolera iz prvog lica (*FPS*) te kontrolera iz trećeg lica (*TPS*).

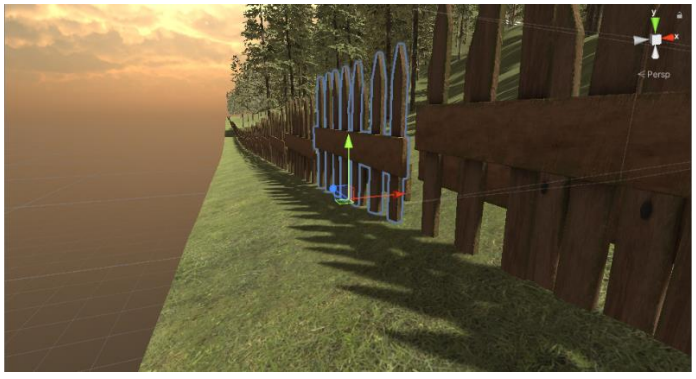


Slika 30. Dodavanje kontrolera igrača

3.3.4. Ograda

Ograda koja se nalazi na rubovima mape sprječava igrača da padne van mape i tako završi u beskonačnom padu. Može se postavljati kao običan model/objekt principom *drag and drop* no to bi moglo dugo potrajati pa se u ovom slučaju koristi programska komponenta koja sadrži skriptu te štedi vrijeme postavljanja ograde. Prvo moramo označiti točku A i točku B, a nakon toga kada se u komponentu stavi model željene ograde, kreira se petlja između točke A i točke B. Igrač će to percipirati kao gotovu spojenu ogradu umjesto pojedinačno složenih panela ograde.

Također komponenta kontrolira veličinu, rotaciju i razmak između pojedinačnih modela u petlji. Nakon ovog postupka ograda će biti vidljiva no postoji problem transparentnosti ponekih modela učitanih u Unity. Igrač zbog toga može proći kroz ogradu bez ikakvih kolizija i pasti van mape. Kako bi to ispravili koristit ćemo komponentu *Box Collider* koja se može pronaći u *Inspector* prozoru nakon selektiranja petlje ograde.



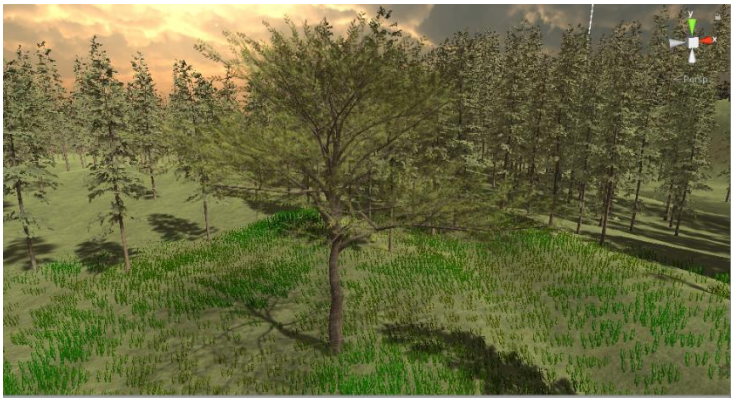
Slika 31. Ograda

```
using System.Collections.Generic;
using UnityEditor;
using UnityEngine;
[ExecuteInEditMode]
public class FenceLayout : MonoBehaviour {
	public List<Vector3> FencePoints = new List<Vector3>();
	public Transform[] FencePrefabs = new Transform[1];
	public Transform PostPrefab;
	public Vector3 FenceEndOffset = new Vector3();
	public float FenceRotation = 0.0f;
	public float FenceLength = 2.0f;
	public Vector3 FenceScale = new Vector3(1, 1, 1);
	public bool CompleteLoop = false;
	public bool ObjectMode = false;
	public Vector2 PositionVariation = new Vector2(0, 0);
	public Vector2 RotationVariation = new Vector2(-180, 180);
	public Vector2 ScaleVariation = new Vector2(1, 1);
	public bool UseShear = true;
	public enum EditMode {
		None,
		Add,
		Delete};
	private EditMode m_currentEditMode = EditMode.None;
	private Vector3 m_mousePos;
	private void OnDrawGizmosSelected() {
		Vector3 lastPos = new Vector3();
		bool gotLast = false;
		int deleteIdx = m_currentEditMode == EditMode.Delete ?
GetDeletePointIdx() : -1;
		for (int i = 0; i < FencePoints.Count; i++) {
			Vector3 p = FencePoints[i] + transform.position;
			if (m_currentEditMode == EditMode.Delete && i ==
deleteIdx) {
				Gizmos.color = new Color(1.0f, 0.0f, 0.0f);
				Gizmos.DrawSphere(p, 0.5f);}
			Else {
				Gizmos.color = m_currentEditMode == EditMode.Add ?
new Color(1.0f, 1.0f, 1.0f) : new Color(0.2f, 1.0f, 0.2f);
				Gizmos.DrawSphere(p, 0.3f);}
			if (gotLast) {
				Gizmos.color = new Color(0.2f, 1.0f, 0.2f);
				Gizmos.DrawLine(lastPos, p);}
			lastPos = p;
			gotLast = true;}
		if (m_currentEditMode == EditMode.Add) {
			Gizmos.color = new Color(1.0f, 0.0f, 0.0f);
			Gizmos.DrawSphere(m_mousePos, 0.3f);
			Gizmos.color = new Color(1.0f, 1.0f, 0.0f);
			if (FencePoints.Count > 0) {
				int addIdx = GetAddPointIdx();
				if (addIdx > 0) {
					Gizmos.DrawLine(m_mousePos,
FencePoints[addIdx - 1] + transform.position);}
				if (addIdx < FencePoints.Count) {
```

```
                              Gizmos.DrawLine(m_mousePos,
FencePoints[addIdx] + transform.position);} } } }
      public void SetEditMode(EditMode mode, Vector3 mousePos) {
            m_currentEditMode = mode;
            m_mousePos = mousePos;}
      public int GetAddPointIdx() {
            if (FencePoints.Count == 0) {
                  return 0;}
            else if (FencePoints.Count == 1) {
                  return 1;}
            else {
                  Vector3 addPos = m_mousePos - transform.position;
                  int closestIdx = 0;
                  float closestDist = float.MaxValue;
                  for (int i = 0; i < FencePoints.Count; i++) {
                        float dist = (FencePoints[i] -
addPos).sqrMagnitude;
                        if (dist < closestDist) {
                              closestIdx = i;
                              closestDist = dist;} }
                  if (closestIdx == 0) {
                        if (Vector3.Dot(addPos - FencePoints[0],
FencePoints[1] - FencePoints[0]) > 0) {
                              return 1;}
                        return 0;}
                  else if (closestIdx == FencePoints.Count - 1) {
                        if (Vector3.Dot(addPos -
FencePoints[FencePoints.Count - 1], FencePoints[FencePoints.Count - 2] -
FencePoints[FencePoints.Count - 1]) > 0) {
                              return FencePoints.Count - 1;}
                        return FencePoints.Count;}
                  else {
                        Vector3 dirPrev = (FencePoints[closestIdx - 1] -
FencePoints[closestIdx]).normalized;
                        Vector3 dirNext = (FencePoints[closestIdx + 1] -
FencePoints[closestIdx]).normalized;
                        Vector3 dirThis = (addPos -
FencePoints[closestIdx]).normalized;
                        if (Vector3.Dot(dirThis, dirPrev) >
Vector3.Dot(dirThis, dirNext)) {
                              return closestIdx;}
                        return closestIdx + 1;} } }
      public int GetDeletePointIdx() {
            Vector3 addPos = m_mousePos - transform.position;
            int closestIdx = -1;
            float closestDist = 10.0f;
            for (int i = 0; i < FencePoints.Count; i++) {
                  float dist = (FencePoints[i] - addPos).sqrMagnitude;
                  if (dist < closestDist) {
                        closestIdx = i;
                        closestDist = dist;} }
            return closestIdx;} }
```

3.3.5. Drveće

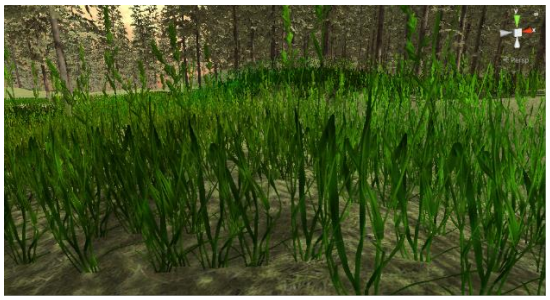
Kako teren na kojem se odvija igra ne izgleda prazno i monotono potrebno je ubaciti drveće. Za postavljanje drveća Unity ima posebnu komponentu *Paint Trees.* Odabiremo model i vrstu drveća te ih jednostavno „bojamo“ po mapi umjesto da ubacujemo jedno po jedno. Od opcija se nudi veličina kista, gustoća i visina naslikanog drveća. Osim drveća koja Unity nudi korisniku od početka, korisnik može i sam skinuti svoje modele sa Unity dućana i ubaciti ih u projekt.



Slika 32. Drveće

3.3.6. Trava

Trava se dodaje na sličan način kao i drveće. Korisnik uzima kist i „boja“ vegetaciju i travu po mapi. Odabire se veličina kista, neprozirnost i gustoća. Trava osim što može biti *mesh,* može biti i obična 2D tekstura. Mnogo vegetacije obično stavlja pritisak na grafičku karticu računala pa se sa opcijama drveća, trave i vegetacije mora postupati lakše radi očuvanja računalnih resursa. Trava sadrži i animaciju gibanja, odnosno pomiče se ovisno o smjeru puhanja vjetra.



Slika 33. Trava

3.4. Mehanike igre

Kako bi igru učinili zanimljivijom u nju stavljamo razne igraće mehanike. Za našu igru iskoristit ćemo mehanike neprijateljskih jedinica koje napadaju glavnog lika, mehanika životnih bodova lika (gubljenje i mogućnost vraćanja životnih bodova) i brojač koji daje igraču informaciju koliko mu je vremena ostalo da riješi problem.

3.4.1. Neprijatelji

Neprijatelji glavnom liku u igri predstavljaju opasnost. Nalaze se raštrkani po terenu na kojem se igra odvija i čekaju igrača da im se približi. Neprijatelji će napasti jedino ako im se igrač približi na određenu udaljenost koja je zadana u programskom kodu. U slučaju da neprijatelj napadne igrača, igrač mora napraviti veću udaljenost nego je zadano u programskom kodu kako bi ga neprijatelj prekinuo slijediti.

Kada neprijatelj dobije alarm da je u određenom radijusu lik igrača on ga prvo počinje slijediti i ako mu se dovoljno približi i nastane kolizija igrač će dobiti penale, odnosno u našem slučaju smanjit će mu se životni bodovi. Model neprijatelja je preuzet s Unity dućana s asset-ima. Imamo 2 vrste neprijatelja od kojih su leteći neprijatelji agresivniji i brži dok su neprijatelji na tlu sporiji. Osim programske skripte na neprijatelja je potrebno dodati *Nav Mesh Agent* komponentu koja daje razne opcije za upravljanje neprijateljskim jedinicama poput brzine kretanja jedinice, akceleracije, zaobilaženje prepreka, upravljanje kvalitetom izgleda neprijatelja. Osim navedenog moramo i „zapeći“ teren uz pomoć *Navigation* taba i opcije *Bake* koji omogućuje provjeravanje svih neravnina na terenu. Ta opcija zapravo šalje sve informacije o tome gdje se može prolaziti svim objektima koji imaju mogućnost kretanja po terenu.

Slika 34. Neprijatelj u zraku

Slika 35. Neprijatelj na tlu

Skripta koja pokreće neprijatelje pronađena je na internetu te je izmijenjena za potrebe projekta.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.AI;
public class EnemyMaster : MonoBehaviour {
    public GameObject PlayerCapsule;
    public float Distance;
    public bool isAngered;
    public NavMeshAgent _agent;
    void Update() {
        Distance = Vector3.Distance(PlayerCapsule.transform.position,
this.transform.position);
        if (Distance <= 20) {
            isAngered = true;}
        if (Distance > 20) {
            isAngered = false;}
        if (isAngered) {
            _agent.isStopped = false;
            _agent.SetDestination(PlayerCapsule.transform.position);}
        if (!isAngered) {
            _agent.isStopped = true;} } }
```

3.4.2. Životni bodovi igrača

Životni bodovi u video-igrama predstavljaju „zdravlje“ pojedinih jedinica koje u slučaju ako dođu na brojku nula, gube život i moraju ponovo pokrenuti igru ili se desi *respawn* jedinice. Životni bodovi mogu biti posloženi na mnogo načina, a mi smo odabrali način da nam glavni

igrač ima 100 životnih bodova koje gubi u dodiru sa neprijateljskim jedinicama. Svaka kolizija sa neprijateljskom jedinicom oduzima određeni dio ukupnih životnih bodova ovisno o tome kako smo to posložili u programskom kodu. Osim programske skripte moramo dodati i vizualnu informaciju o trenutnom stanju životnih bodova jer bez toga igrač ne bi znao koliko mu je životnih bodova još ostalo. Prikaz životnih bodova najčešće je fiksiran u jednom dijelu igračeve kamere. U Unity-u prikaz životnih bodova najlakše je dodati preko opcije *Canvas* koja se nalazi u UI sekciji kod kreiranja novog objekta u prozoru *Hierarchy.* Programskim smo kodom omogućili dvije stvari, a to su skidanje životnih bodova nakon kolizije s neprijateljima i *update* životnog prikaza. Konkretno to znači da će se u našem slučaju smanjivati zelena kutija i da će se brojka životnih bodova promijeniti. Programska skripta za ovu komponentu pronađena je na internetu i promijenjena je za potrebe projekta:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
public class FillStatusBar : MonoBehaviour {
    public Health playerHealth;
    public Image fillImage;
    private Slider slider;
    void Awake() {
        slider = GetComponent<Slider>();}
    void Update() {
        if (slider.value <= slider.minValue) {
            fillImage.enabled = false;}
        if (slider.value > slider.minValue && !fillImage.enabled) {
            fillImage.enabled = true;}
            float fillValue = playerHealth.currentHealth /
playerHealth.maxhealth;
        if(fillValue <= slider.maxValue/3) {
            fillImage.color = Color.white;}
        else if(fillValue > slider.maxValue/3) {
            fillImage.color = Color.red;}
            slider.value = fillValue;} }
```

Skripta *FillStatusBar* omogućuje kretanje *slider-a* po pravokutniku na kojem se nalazi. On se pomiče s desna na lijevo i ispod sebe ostavlja bijelu pozadinu. Sljedeća skripta se nalazi na kontroleru igrača i odgovorna je za provjeravanje kolizije sa neprijateljskim jedinicama i slanje podataka skripti *FillStatusBar* da pomiče slider. U slučaju nula preostalih životnih bodova, restartat će se cijeli nivo i igrač počinje ispočetka. Kako bi skripte mogle zajedno pronaći jedna drugu moraju se koristiti *public* podaci.

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

```
public class PlayerHealth : MonoBehaviour {
    public float maxHealth = 100;
    public float curHealth;
    public Slider slider;
    public Text text;
    void Update() {
        slider.value = curHealth;
        text.text = "Health : " + curHealth;
        if (curHealth >= 100) {
            curHealth = maxHealth;}
        if (curHealth <= 0) {
            Debug.Log("Player is dead");
            Application.LoadLevel(Application.loadedLevel);} }
    void OnCollisionEnter(Collision obj) {
        if (obj.gameObject.tag == "Enemy")
            curHealth = curHealth - 1f;} }
```

Slika 36. Životni bodovi

Kako bi igrač mogao dulje opstati u igri, uvodimo mehaniku vraćanja životnih bodova. Maksimalan broj životnih bodova je 100 jedinica, a skupljanjem objekata za vraćanje životnih bodova igrač vraća 15 jedinica. Mehanika kojom se postiže takav sistem je spajanje programskih skripti sa postojećom skriptom igračevih životnih bodova. Prvo kreiramo željeni objekt koji će predstavljati napitak, a potom mu dodajemo *Box Collider* komponentu i programsku skriptu. Način na koji možemo mijenjati vrijednost napitka je direktan upis u programski kod ili promjena vrijednosti napitka u komponenti gdje se nalazi skripta. Za objekt koristimo model bačve i model srca koje omogućava lakšu vidljivost igraču. Skripta za napitak glasi:

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

```
public class HealthPickup : MonoBehaviour {
    PlayerHealth playerHealth;
    public float HealthBonus = 15f;
    void Awake() {
        playerHealth = FindObjectOfType<PlayerHealth>();}
    void OnTriggerEnter(Collider napitak) {
            if (playerHealth.curHealth <= playerHealth.maxHealth &&
playerHealth.curHealth != playerHealth.maxHealth) {
                Destroy(gameObject);
                playerHealth.curHealth = playerHealth.curHealth +
HealthBonus;} } }
```

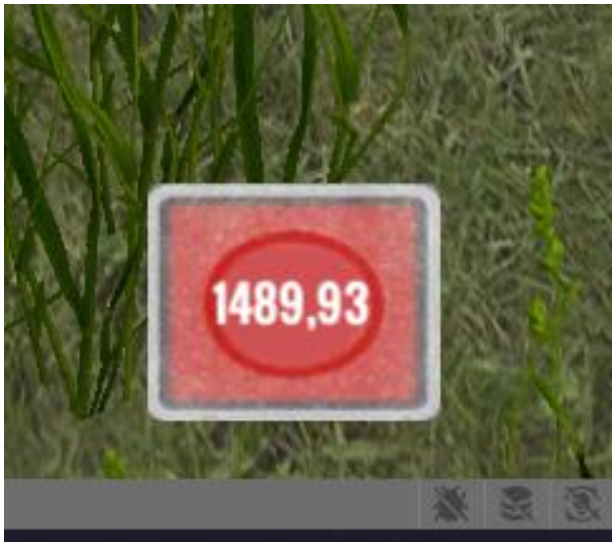
Slika 37. Napitak

3.4.3. Brojač

Osim neprijatelja, brojač u igri služi kao još jedan od načina kako usmrtiti igrača. Brojač se izrađuje na istom principu kao i životni bodovi, a to je dodavanje *Canvas-a*. Brojač počinje odbrojavati od trenutka pokretanja igre i u slučaju da stigne na nulu, resetira cijeli nivo. Brojač se nalazi u crvenom pravokutniku i igraču je konstantno vidljiv u donjem desnom kutu. Odbrojavanje traje 25 minuta, odnosno 1500 sekundi, a prikazuje sekunde i stotinke. Skripta koja pokreće brojač pronađena je na internetu i izmijenjena je za potrebe projekta:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
public class CountdownTimer : MonoBehaviour {
    public string levelToLoad;
    private float timer = 1500f;
    private Text timerSeconds;
    void Start() {
```

```
        timerSeconds = GetComponent<Text>();}
    void Update() {
        timer -= Time.deltaTime;
        timerSeconds.text = timer.ToString("f2");
        if(timer<=0) {
            Application.LoadLevel(Application.loadedLevel);} } }
```



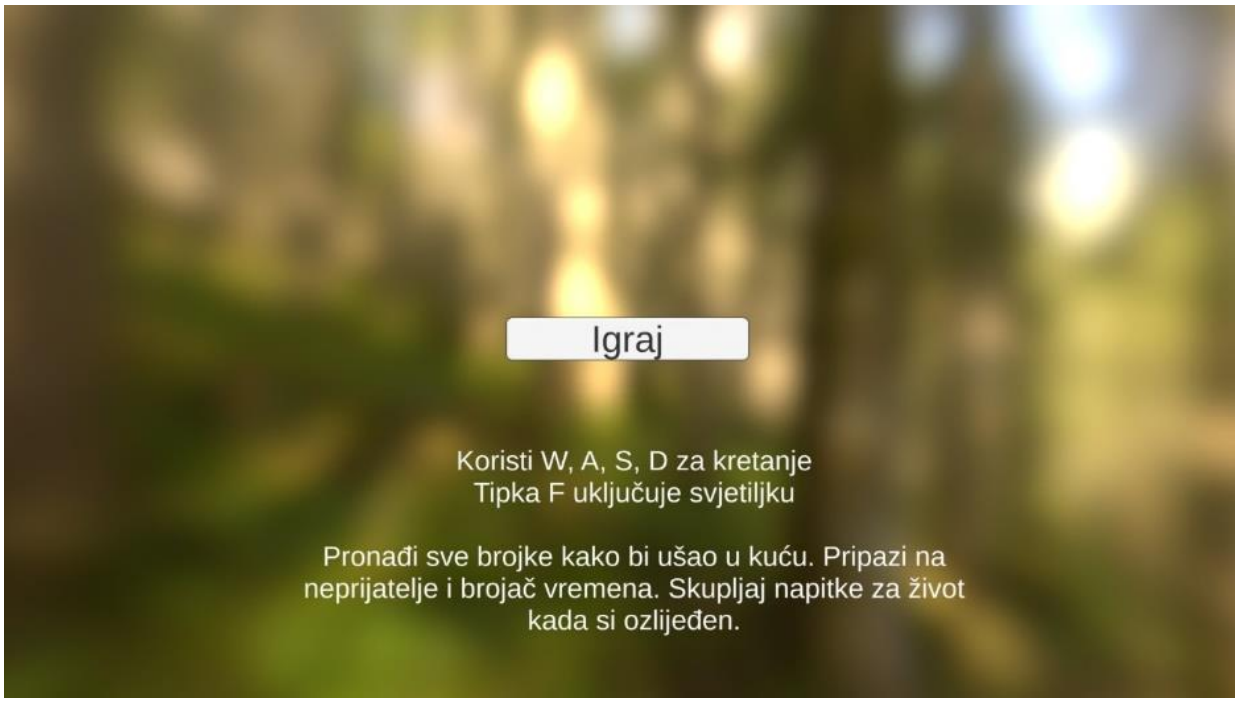
Slika 38. Brojač

3.4.4. Start meni

Start meni iskorišten je za kratku uputu igraču. Sadržava samo jedan gumb, a to je gumb koji pokreće igru. Kratko su opisane mehanike igre i što je igraču činiti kako bi stigao do kraja igre. UI za start meni izrađen je uz pomoć *Canvasa* i korištenja gumba. Jednostavna skripta koja se nalazi na gumbu pokreće drugu zadanu scenu u Unity programu.

Kod za start meni:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
public class MainMenuManager : MonoBehaviour {
    public void startFunction() {
        SceneManager.LoadScene("UnityPrijeFinalGame");
        Cursor.visible = false;} }
```



Slika 39. Start Meni

4. Izrada – Reaper

4.1. Žanr glazbe

Glazba u igrama služi za ispunjavanje atmosfere i upravljanje emocijama igrača. Žanr pozadinske glazbe u igrama uvelike ovisi o žanru igre s kojim se spaja. Za pucačine i trkaće igre se obično koristi brža glazba, a naš projekt zahtijeva sporiji žanr glazbe. Iz tog razloga *drone* žanr glazbe odlično pristaje atmosferi psiholoških avantura. Svojim sporim pulsiranjem i jednostavnim repetitivnim aranžmanom *drone* kreira novu dimenziju odvijanja video-igre koja se zapravo najviše odvija u samom igraču na način da otvori emocionalni kanal i spoji ga sa igrom.

Za takav žanr glazbe može se koristiti mnoštvo različitih instrumenata, od gitare, vokala, sintetizatora zvuka pa sve do didgeridoo-a. Obično se dodaju i razne efekt pedale te plugin-ovi. Za upravo ovaj projekt tu ćemo takvu vrstu glazbe realizirati uz pomoć tih alata.

4.2. Izrada glazbe

Za početak otvaramo novi projekt u programu Reaper i pripremamo program za snimanje naše pjesme. *Beats per minute* postavljamo na 120 udaraca i stavljamo četveročetvrtinski takt. Ta svojstva pomažu nam kod snimanja samih traka kako bi svi instrumenti bili u harmoniji bez odskakanja u tempu. Reaper u svom osnovnom paketu plugin-ova ne sadržava VST plugin-ove koji se obično koriste za simulaciju instrumenata i gitarskih pojačala i kabineta već te ih moramo dodatno skinuti i učitati u program, slično kao učitavanje Asset-a u Unity.

Popis plugin-ova za izradu glazbe:

1. **S-Gear 2 –** simulacija gitarskih pojačala
2. **Tyrell N6 –** simulacija sintetizatora zvuka
3. **ReaEQ –** ekvilajzer zvuka
4. **ReaComp –** kompresija zvuka
5. **Red3 Compressor –** kompresija zvuka
6. **Thrillseeker VBL –** kompresija zvuka na mix bussu
7. **Valhalla VintageVerb -** reverb
8. **DeEsser Mono –** DeEsser za vokale
9. **Tube Delay -** delay

Uz virtualne alate koji su učitani u Reaper, koristit ćemo i hardverske alate. Za MIDI upravljanje sintetizatorima zvuka koristit ćemo Novation Launchkey 25 MKII, za žičani instrument koristit ćemo Gibson SG Standard za P90 pickup-ima i mnoštvo gitarskih efekata i pedala. Za kratke monologe lika koristit ćemo Shure SM57 mikrofon. Sva oprema se može spojiti direktno u zvučnu karticu Focusrite Scarlett 2i4 te zbog korištenja virtualnih simulacija nije nam potrebno korištenje pojačala, analognih mikseta i mikrofona za snimanje instrumenata.

Slika 40. Novation Launchkey

Slika 41. Gibson SG

Slika 42. Pedalboard

Popis gitarskih efekata korištenih izradu glazbe:

1. **Eventide Space –** Reverb
2. **Source Audio Nemesis Delay -** Delay
3. **Fulltone OCD v2 -** Overdrive
4. **Xotic RC Booster –** Booster signala
5. **Polytune 3 -** Štimer
6. **Buzzaround Fuzz –** Fuzz

4.2.1. Sintetizator zvuka

Nakon pripreme za rad prvi instrument koji ćemo koristiti je sintetizator zvuka. Kontrolirat ćemo ga preko Novation MIDI (Musical Instrument Digital Interface) kontrolera. Pritiskom tipki na kontroleru DAW to registrira kao ulazni signal te nakon kratke latencije vrši *playback* zvuka. Ulazna latencija ovisi o jačini računala, postavkama u Reaper-u te kvaliteti zvučne kartice. Za pjesmu ćemo dodati tri različite snimke sintetizatora zvuka: bas sintetizator, sintetizator koji proizvodi šum te sintetizator koji ima vodeću melodiju.

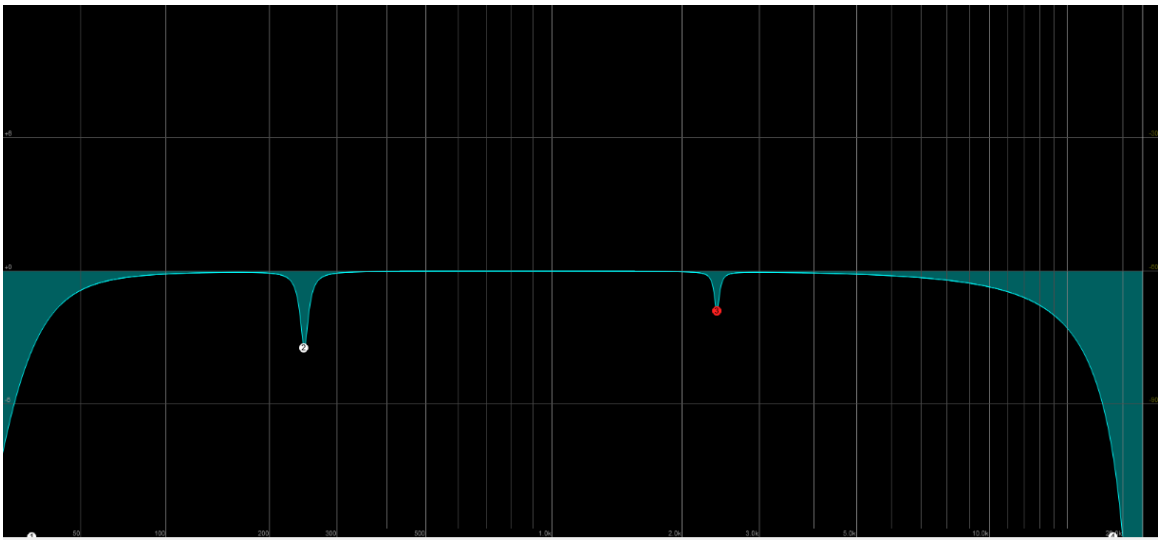
Za prvu traku snimamo niske frekvencije koje pune prostor i počinju pjesmu, a uz pomoć *latching* sistema kod snimanja možemo jedanput pritisnuti tipku i ona će beskonačno trajati. Nakon snimanja trake na vremensku liniju, sirovu traku potrebno je obraditi. Miksanje je uvijek najbolje raditi kada već imamo nekoliko snimljenih traka istih instrumenata ili

kompletno snimljenu pjesmu iz razloga lakšeg pogađanja omjera ukupnih frekvencija u nekoj pjesmi. Nakon snimanja bas sintetizatora dodajemo sintetizator šuma i sintetizator vodeće melodije. Za razliku od *latching* pristupa, iduće trake snimamo uživo uz pomoć metronoma te trake imenujemo i dodajemo ikone i boju trake za lakše snalaženje u procesu miksanja.

Slika 43. Tyrell N6

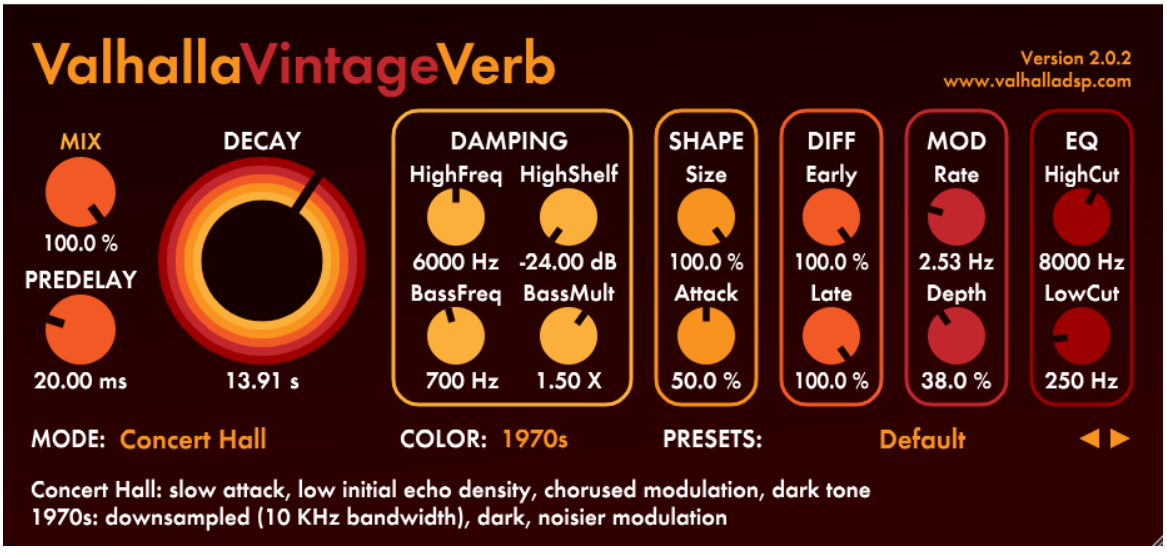
Nakon snimanja i organiziranja svih traka, započinjemo proces miksanja korištenjem ekvilajzera. Odrezat ćemo nepotrebne niske i visoke frekvencije na pojedinačnim trakama da ima manje kolizije u frekvencijama te posebnu pažnju obratiti kod niskih-srednjih i visokih-srednjih frekvencija. Niske-srednje frekvencije u pjesmama mogu izazvati mulj i gubitak definicije instrumenata dok visoke-srednje frekvencije mogu proizvesti oštre frekvencije na koje je ljudsko uho prilično osjetljivo.

Kod kompresije potrebno je pažljivo koristiti *ratio* da traka ne postane previše kompresirana te je rezultat toga gubitak dinamike. Sintetizator ne trebamo previše kompresirati jer instrument sam po sebi već ima mnogo kompresije u zvuku.



Slika 44. Primjer ekvilajzera

Za kraj na posebnu traku dodajemo reverb i spajamo je sa trakom sintetizatora zvuka zato da bi za reverb imali poseban lanac efekata. Reverb je kompleksan efekt koji proširuje zvuk i daje dubinu, ali zato ga treba korigirati pomoću ekvilajzera da se smanji kolizija zvuka s ostalim instrumentima i trakama.



Slika 45. Valhalla VintageVerb

4.2.2. Gitara

U usporedbi sa sintetizatorom zvuka, gitara je instrument koji je lakše snimati i miksati. Za snimanje gitare prvo spajamo gitaru u lanac efekata po želji te na kraju spajamo u zvučnu

karticu preko koje simuliramo gitarsko pojačalo i kabinet. Za gitaru se često koristi *overdubbing* tehnika snimanja kojom se precizno snima više snimaka istog dijela da naposljetku zvuči veće.

Za prvi snimak gitaru spajamo u Fulltone OCD, Xotic RC booster, Nemesis delay i Eventide Space reverb. Za simulaciju gitarskog pojačala uzimamo pojačalo čistog zvuka iz kolekcije pojačala S-Gear. Za drugi snimak koristimo gitaru spojenu u Buzzaround Fuzz i direktno spojenu u zvučnu karticu bez dodatnih prostornih efekata. Time ćemo postići omjer „suhog/mokrog“ *(dry/wet)* da se iz gitare izvuku svi potrebni tranzijenti i harmonici, a da ona zvuči široko. Slično kao i sintetizator zvuka, na gitaru je potrebno staviti ekvilajzer koji će maknuti nepotrebne frekvencije i precizno balansirati frekvencije koje su bitne za ljudsko uho. Nakon toga u lanac plugin-ova stavljamo kompresiju i pažljivo koristimo *Attack* i *Release* opciju. Za simulaciju gitarskog kabineta koristimo 4x12“ za veći frekvencijski raspon.

Umjesto plugin efekta reverba kao kod sintetizatora zvuka, kod gitare koristimo hardverski efekt. Hardverski efekti u većini slučajeva nude organski i nešto kvalitetniji zvuk u usporedbi sa digitalnim simulacijama. Na pedali koristimo širok stereo reverb koji u svako uho šalje posebno procesiran signal te to u konačnici rezultira ambijentalnim zvukom gitare.

Slika 46. S-Gear

4.2.3. Vokali

Vokali u našem projektu imaju ulogu kratkog monologa lika. Vokal je pridružen na početak pjesme kako bi je igrač čuo odmah pri pokretanju igre. Snimljene su jednostavne fraze koje predstavljaju izgubljenost glavnog lika u nepoznatom svijetu. Za snimanje vokala mikrofon je spojen direktno u zvučnu karticu bez dodatnog hardvera.

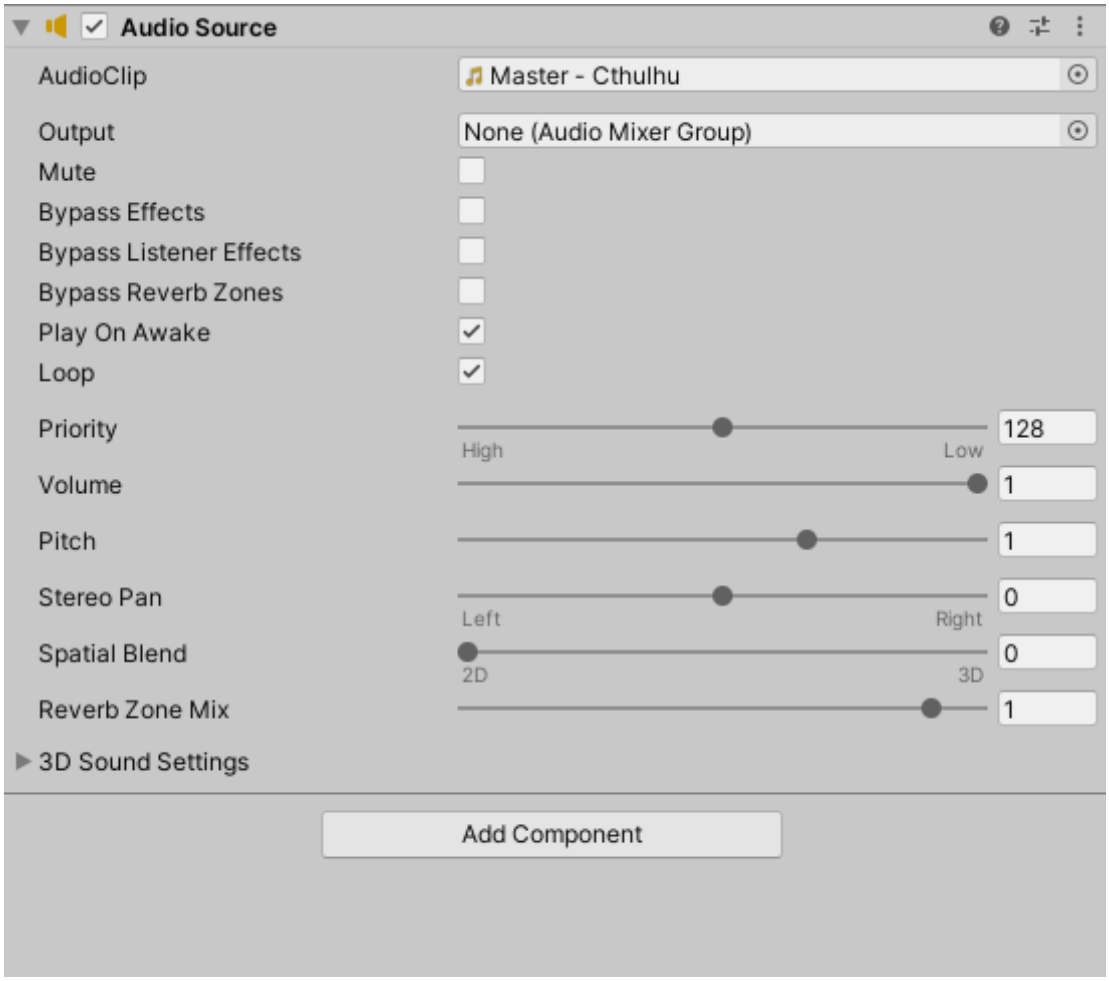
Slično kao i ranije opisani instrumenti, vokali se miksaju na način da sirovu snimku kompresiramo, dodamo ekvilajzer zvuka i reverb, ali se uz to obično dodaje i *DeEsser* koji služi za korekciju pretjerano istaknutih suglasničkih zvukova poput slova „s“ i „z“. Nakon ovog koraka slijedi *Master* trake, odnosno dizanje glasnoće na prihvatljivu razinu za streaming i za CD kvalitetu.

Slika 47. DeEsser

4.3. Učitavanje glazbe u projekt

Za kraj u Unity projekt učitavamo pjesmu koju smo prethodno izvezli iz Reaper-a te tako postižemo audio-vizualnu kombinaciju između svijeta video-igara i svijeta glazbe. U Unity pjesmu učitavamo tako da prvo dodamo prazan objekt u hijerarhijski prozor, a nakon toga tom objektu dodajemo komponentu *Audio Source* koja omogućava uvoz audio zapisa u projekt.

Komponenta omogućava jednostavno upravljanje zvukom i daje opciju korisniku da upravlja glasnoćom trake, upravlja stereo poljem te opcija da se učitana pjesma beskonačno izvodi. Naziv trake koja je učitana u video-igru naziva se *Cthulhu.* U video-igri pjesma počinje odmah na pokretanju igre te traje oko petnaest minuta, a nakon završetka i u slučaju da korisnik još uvijek igra igru, pjesma počinje iznova.



Slika 48. Učitavanje pjesme

5. Zaključak

Video-igre u današnje doba su sve više popularan pojam, a glavni je razlog da ljudi vole zabavu i izazove. Tome pridonosi i velik broj platforma koje podržavaju video-igre i cijela industrija se konstantno razvija u pogledu kvalitete, tehnologija i kompleksnosti igara. Cijeli proces izrade video-igara je prilično zahtjevan jer osim striktno tehničkih vještina, kreatori igara trebaju biti i kreativno potkovani.

Za izradu vlastite igre trebalo mi je nekoliko vještina koje sam morao savladati. Izrada terena za igru, korištenje modela, programiranje skripte za interakciju, upravljanje kamerom lika, samo su neke od vizualnih vještina, dok je za glazbu potrebno znanje o kompoziciji, snimanju, miksanju i masteriranju glazbe.

Naglim porastom broja igara dolazi i do veće saturacije tržišta i žanrovske sličnosti igara. Mnogo timova koji su sa manjim financijskim potporama izradili video-igre dokazali su da video-igre ne moraju biti samo *triple-a* naslovi i da osim kratkotrajne zabave, igre mogu imati jak psihološki učinak na pojedinca. Obično takve video-igre u sebi imaju komponente ostalih umjetnosti poput filma i glazbe. Psihološka avantura pravi je primjer da jednostavnost može imati jači učinak nego složenost. Upravo to me inspiriralo da se okušam u izradi video-igre koja nije samo video-igra, već je i djelo koje spaja komponiranu autorsku glazbu.

Popis literature

[1] – Genres of Video Games [Preuzeto na internetu] . Dostupno na:

https://www.idtech.com/blog/different-types-of-video-game-genres [1.6.2021]

[2] – Video Game Cost [Preuzeto na internetu] . Dostupno na:

https://vgsales.fandom.com/wiki/Video_game_costs [1.6.2021]

[3] – Video Game Platforms[Preuzeto na internetu] . Dostupno na:

https://starloopstudios.com/what-are-the-best-platforms-for-video-games/ [1.6.2021]

[4] - Unity How It Started [Preuzeto na internetu] . Dostupno na:

https://techcrunch.com/2019/10/17/how-unity-built-the-worlds-most-popular-game-engine/ [1.6.2021]

[5] – Unity [Preuzeto na internetu] . Dostupno na: https://techcrunch.com/2018/09/05/unity-ceo-says-half-of-all-games-are-built-on-unity/?guccounter=1&guce_referrer=aHR0cHM6Ly93d3cuZ29vZ2xlLmNvbS8&guce_referrer_sig=AQAAALqxLqftP6B9AeesPVaEgAub1GuwKuebz-MzxMr8VaNSBHB4DTqaWqPzLOGazP3E4Gy0AVZHOpHCPlq8rRlqrWYspQAwirMZrnUg5bDI3GbRehicTyapFpb71CzqpTs-la4Kw-Glt01_ABKBrtc8QIsq6aC9ShWtQyH0o7ICBv6o

[6] – Reaper [Preuzeto na internetu] . Dostupno na: https://happymag.tv/reaper/

[7] – Dear Esther [Preuzeto na internetu]. Dostupno na: https://www.thechineseroom.co.uk/games/dear-estherhttps://www.thechineseroom.co.uk/games/dear-esther [5.6.2021.]

Popis slika