Metode i tehnike izrade računalnih igara

Ivan, Bogović

Master's thesis / Diplomski rad

2020

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Organization and Informatics / Sveučilište u Zagrebu, Fakultet organizacije i informatike**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:211:953584

Rights / Prava: Attribution-NonCommercial 3.0 Unported/Imenovanje-Nekomercijalno 3.0

Download date / Datum preuzimanja: **2024-04-23**

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Repository / Repozitorij:

Faculty of Organization and Informatics - Digital Repository

zir.nsk.hr

dabar
DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Ivan Bogović

METODE I TEHNIKE IZRADE RAČUNALNIH IGARA

DIPLOMSKI RAD

Varaždin, 2020.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Ivan Bogović

Matični broj: 0016116682

Studij: Informacijsko i programsko inženjerstvo

METODE I TEHNIKE IZRADE RAČUNALNIH IGARA

DIPLOMSKI RAD

Mentor/Mentorica:

Doc. dr. sc. Mario Konecki

Varaždin, travanj 2020.

Ivan Bogović

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

Tijekom izrade ovog rada, usmjerit ćemo se prema tehnikama izrade računalnih igara. Jedan od važnijih početnih koraka je odabir izdavača (eng. Publisher) koji će kao takav financirati sami razvoj igre. Prilikom stvaranja različitih dogovora s izdavačem, potrebno je odrediti troškove razvoja same igre tj. pojedinih funkcionalnosti i komponenti igre te će cijeli proces na primjeru biti opisan u ovom radu.

Nadalje, u radu će na kratkim primjerima poznatih igara biti prikazani troškovi razvoja te sami prihodi pojedinih igara kako bi se dobio osjećaj veličine i kompleksnosti razvoja igara.

Također, bit će prikazana povijest tehnologija koje su se koristile u samom početku razvoja različitih računalnih igara te će one biti uspoređene s današnjim aktualnim tehnologijama koje se koriste.

Kroz određeni programski primjer će biti prikazan razvoj jednostavne računalne igre te će biti prikazan programski kod same igre kako bi bilo lakše razumjeti sami proces razvoja.

Ključne riječi: računalna, igra, programski, kod, razvoj, tehnike, metode, tehnologija

Sadržaj

Sadržaj iii
1. Uvod 1
2. Razvoj video igara 2
2.1. Životni ciklus razvoja igara 2
2.2. Elementi koji utječu na trošak razvoja igara 6
2.3. Tehnologije i pristupi razvoju igara 10
2.3.1. Pokretači igara (eng. Game engine) 10
2.3.2. Programi za 3D modeliranje 13
2.3.3. Programi za grafički dizajn 14
2.3.4. Metode za postizanje većeg realizma u igrama 16
2.3.5. Kreiranje 3D objekta i njegova upotreba u već postojećoj igri 21
3. Izrada 2D igre za PC 30
3.1. Priprema okruženja za rad 30
3.2. Prikaz igrača i postavljanje osnovnih mogućnosti 31
3.3. Postavljanje platformi za kretanje 34
3.4. Povezivanje kamere sa igračem te generiranje platformi 36
3.5. Skakanje igrača i uklanjanje platformi 42
3.6. Animacija skakanja i postava razmaka između platformi 45
3.7. Generiranje i sakupljanje bodova 50
3.8. Pozadina igre i zvučni efekti 54
3.9. Sakupljanje bodova 58
3.10. Smrt igrača i dovršetak razvoja igre 60
4. Zaključak 66
Popis literature 67
Popis slika 72
Popis tablica 74
Prilozi 75

1. Uvod

Računalne igre u moderno vrijeme predstavljaju jedan od najraširenijih oblika zabave. Sami razvoj video igara kroz povijest je potakao razvoj sve naprednijih i naprednijih tehnologija, bilo to programskih ili hardverskih. Početci razvoja industrije igara kreću od ranih 70-tih godina prošlog stoljeća kada su razvijene igre kao što je „Pong“ od strane Atari-a koji je isto tako nedugo zatim stvorio jednu od prvih igraćih konzola. Navedena konzola se je zvala VCS (Video Computer System) 2600 i pojavila se 1977. godine [1].

Daljnjim nastavkom razvoja industrija igara su se počele pojavljivati sve naprednije igre te se je tržište širilo. Uz razvoj igara za računala, pojavio i veći broj konzola kao što je Nintendo, PlayStation, Xbox itd. Opseg samih igara je rastao te troškovi razvoja kreću u milijunima dolara te isto tako i prihodi ostvareni prodajom. Pojavili su se neki od najpoznatijih razvojnih studija kao što su: Rockstar Games, Ubisoft, DICE, Electronic Arts, Valve Company, Activision Blizzard, BioWare itd.

U ovome radu ćemo se orijentirati na tehnologije i metode koje se koriste za razvoj video igara te ćemo opisati i neke faktore tj. elemente koji naviše utječu na troškove razvoja. Uz to, prikazati ćemo jednostavan primjer kreiranja 3D modela u odabranom programu te njegov prijenos u pokretač igre (eng. Game engine). Na kraju ćemo taj model iskoristiti kako modifikaciju koju će igrač moći koristiti u igri. Uz navedeni 3D model koji će se moći koristiti kao element već postojeće igre, kreirati ćemo i 2D igru platformerskog dizajna te ćemo opisati tehniku kreiranja i programski kod koji je napravljen da bi igra mogla ostvariti određene funkcionalnosti.

2. Razvoj video igara

Unutar ovog poglavlja ćemo se ukratko osvrnuti na životni ciklus razvoja igara unutar kojega ćemo ukratko opisati pojedine faze. Nakon toga ćemo prikazati troškove razvoja i poneke elemente koji oni obuhvaćaju. Na kraju ćemo se orijentirati na tehnologije i pristupe koji se danas koriste za razvoj igara. Opisat ćemo ukratko i neke naprednije tehnike razvoja. Na kraju će biti prikazan pristup izradi 3D modela te tehniku prebacivanja 3D modela u pokretač igre Giants Editor. Bit će omogućeno korištenje modela u već postojećoj igri, Farming Simulator, kao korisnička modifikacija.

2.1. Životni ciklus razvoja igara

Prilikom razvoja igara nije dovoljno samo usvojiti i pratiti klasični pristup razvoja programa tj. njegov životni ciklus (eng. Software development life cycle). Klasični pristup se odnosi najvećim dijelom na razvoj programskog koda te njegovo testiranje. Prilikom razvoja igara, razvoj programskog koda je samo jedan dio slagalice koju moramo posložiti. Ostatak se odnosi na stvari kao što je rad sa: grafičkim dizajnom, zvukovima, simulacijom utjecaja fizike na elemente igre, umjetnom inteligencijom, animacijama, kontrolama igre itd.

Kako bi se riješio problem same konstrukcije postupka razvoja, orijentiramo se prema životnom ciklusu razvoja igara znanom skraćeno kao GDLC (eng. Game development life cycle) [2]. Navedeni životni ciklus se sastoji od nekoliko faza: ideja igre, konceptualna analiza, planiranje igre, sastavljanje tima, konceptualni dizajn, razvoj, testiranje, alpha i beta testiranje, produkcija, prodaja i marketing. Sve navedene faze možemo vidjeti na sljedećoj slici:

Slika 1: Životni ciklus razvoja igre [2]

Prvi korak razvoja igre je priča tj. ideja same igre. Kako bismo mogli postaviti ostale ciljeve tijekom daljnjeg razvoja, moramo krenuti od same osnove tj. ideje koju želimo ostvariti te ona kao takva predstavlja realni neizostavni dio životnog ciklusa razvoja. Početne ideje ne predstavljaju cijelu igru koja će biti napravljena već samo osnovne tj. temeljne dijelove koje je potrebno postaviti kako bi se moglo krenuti s razvojem, a onda će se tijekom razvoja dodavati nove ideje te isto tako izmjenjivati već postojeće kako bi se zaobišle prepreke koje će nastati.

Sljedeća faza razvoja se odnosi na konceptualnu analizu koja obuhvaća mogućnost ostvarivanja postavljenih ciljeva. Procjenu izvodljivosti je potrebno napraviti prije samog razvoja igre. Na temelju postavljene ideje igre u prijašnjoj fazi, potrebno je izdvojiti pojmove koji će se pojavljivati prilikom razvoja te iz njih procijeniti elemente kao što su [2]:

- Stvarni zahtjevi
- Trošak razvoja
- Trenutne tehničke mogućnosti potrebne za razvoj
- Mogući organizacijski, kulturalni ili legalni problemi i rješenja
- Veličina projekta
- Potrebne vještine

Ova faza razvoja je bitna, te ju ne treba ignorirati s obzirom da bi se njenim ignoriranjem ili lošom procjenom moglo trajno utjecati na daljnji razvoj projekta.

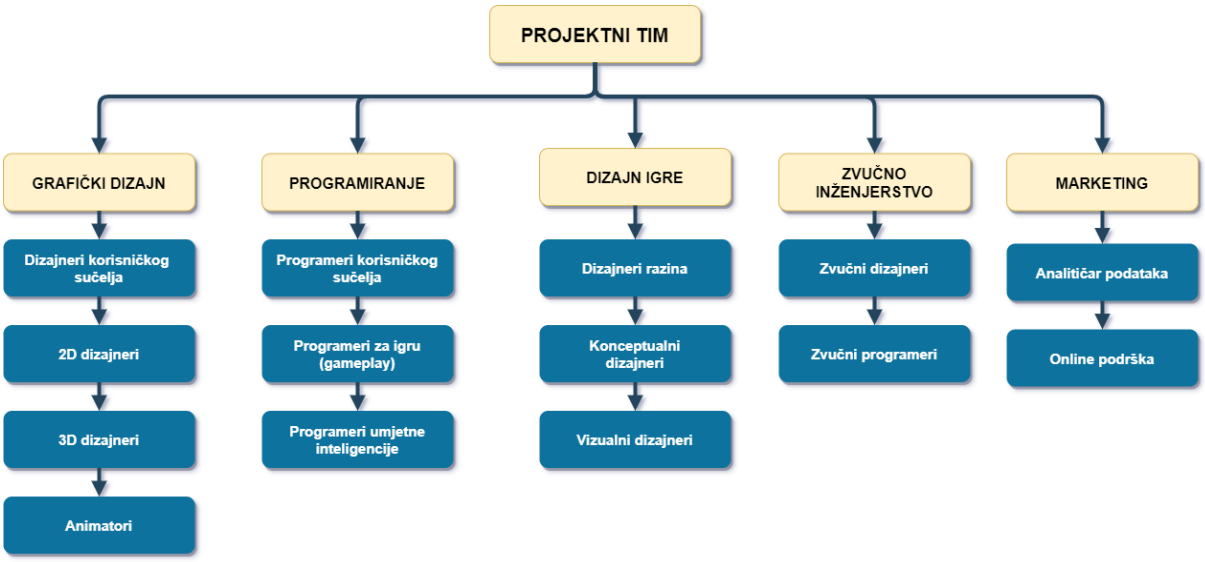
Naredna faza se odnosi na samo planiranje. Nakon što smo u prijašnjoj fazi sakupili sve potrebne zahtjeve koje moramo ostvariti kako bi naša igra bila realizirana prema našim željama, potrebno je kreirati projektni plan razvoja koji će sadržavati neke osnovne stavke kao što su [2]:

- Lista zadataka (grafički dizajn, animacije, zvučni efekti itd.)
- Vremenski plan i procjena potrebnog vremena prema pojedinim zadacima
- Grafikon toka svih zadataka
- Plan testiranja tj. skup testnih slučajeva
- Plan izdavanja igre nakon završetka razvoja
- Plan dorade igre nakon izdavanja i popravljanja grešaka

Navedeni plan razvoja može sadržavati dodatne stavke koje ovdje nisu navedene, te to ovisi o veličini projekta te o samim mogućnostima razvojnog tima.

Kako bi se mogao ostvariti postavljeni rad, potrebno je formirati tim te raspodijeliti dužnosti ovisno o ulozi pojedinih članova. Članovi razvojnog tima se mogu svrstati u nekoliko kategorija: grafički dizajneri, programeri, dizajneri igara, zvučni inženjeri, osiguranje kvalitete

te marketing. Svaka od navedenih kategorija se može podijeliti na nekoliko različitih potkategorija ovisno o sposobnostima tj. vještinama koje posjeduju pojedini članovi. Navedene potkategorije mogu, ali i ne moraju predstavljati stvarnu podjelu tima te je sama podjela u stvarnim okolnostima podložna opsegu razvojnog tima, veličini igre, mogućnostima ostvarivanja postavljenih ciljeva itd. Primjer podjela u potkategorije možemo vidjeti na sljedećoj slici:



Slika 2: Primjer moguće podjele uloga u timu prije početka razvoja [2]

Sljedeća faza se odnosi na konceptualni dizajn odnosno na dizajniranje osnova same igre na koje će se vezati ostale ideje prilikom razvoja. Ova faza predstavlja ključni dio za početak rada na igri jer definira sve potrebne elemente igre koji će biti potrebno raspodijeliti po članovima tima te razviti i spojiti u cjelinu. Razvojem konceptualnog dizajna se obično bavi konceptualni dizajner. On pomoću svih prikupljenih ideja stvara GDD (eng. Game design document) tj. dokument za razvoj igre koji sadrži sve potrebne elemente za razvoj te koji će biti raspodijeljeni članovima tima s obzirom na ulogu [2]. Primjeri nekih najčešćih elemenata su: grafičko sučelje, podaci igre, karakteristike igrača, dizajn razina, mehanike igre, umjetna inteligencija, animacije, zvučni efekti itd.

Nakon postavljanja dizajna igre, potrebno je krenuti na sami razvoj elemenata navedenih u dizajnu tj. GDD dokumentu. Prije početka razvoja je potrebno odabrati grafički pokretač (eng. Game engine) koji će biti korišten te također sve potrebne module i ostale dodatke. Svaki grafički pokretač ima svoje određene prednosti i mane te s obzirom na to, potrebno je pravilno postaviti odabir sukladno samom dizajnu igre te financijskim mogućnostima ako se kupuje/unajmljuje pokretač tj. ako tim već ne posjeduje svoj vlastiti pokretač. Programski razvoj se znatno ne razlikuje od razvoja ostalih oblika programskih proizvoda. Ukratko, svaki programer odrađuje svoje zadatke te ih implementira rješenja

zajedno sa implementacijama ostalih programera. Nadzor kvalitete rješenja vrši glavni programer (eng. Lead programmer) te on daje povratnu informaciju ostalim programerima o samoj kvaliteti rješenja te potrebnim promjenama ako postoje. No prilikom razvoja igara, suočavamo se s više elementa koje je potrebno kontrolirati: kvaliteta zvučnih efekata, vizualni dizajn, animacije i slično te svaki od navedenih elemenata predstavlja dodatno opterećenje s kojim se je potrebno suočiti. Primjer toga bi bio vizualni dizajn. Kako bi se ostvario kvalitetan i igraču primamljiv dizajn, potrebno je npr. Koristiti teksture visoke rezolucije. Time se ostvaruje dizajn koji će privlačiti igrača, no javljaju se problemi kao što je povećanje ukupnog prostora kojeg će igra zauzeti na računalu igrača, opterećenost na grafičkoj kartici tj. potreba za većom količinom radne memorije kartice. Navedeni negativni elementi mogu odbiti određenu skupinu igrača koja nije u mogućnosti financijski si priuštiti bolju opremu za igranje. Zbog toga je vizualni dizajn, isto kao i svaki drugi element igre, potrebno balansirati. Primjer toga je korištenje tekstura visoke rezolucije za samo one elemente s kojima će se igrač direktno i najviše susretati prilikom igranja dok će ostali elementi koristiti teksture niže rezolucije kako bi se povećale performanse igre. To je samo jedan od primjera elemenata s kojima se je potrebno suočiti prilikom razvoja igre.

Nakon završetka razvoja igre, potrebno je vršiti temeljito unutarnje testiranje svih njenih elemenata. Testiranje je neizostavan dio životnog ciklusa razvoja igre te ono predstavlja ogledalo gotovog proizvoda. Unutarnje testiranje se odnosi na testiranje igre koja još nije došla u ruke korisnika tj. to je testiranje koje provodi tim za osiguranje kvalitete (eng. Quality assurance team). Cilj testera igre nije samo igrati igru već ju probatI namjerno „srušiti“ na bilo koji mogući način i pronaći greške (eng. Bug) u njoj [3]. Prilikom testiranja igre, svaku pronađenu grešku je potrebno zabilježiti tj. dokumentirati na već prije dogovoreni način kako bi ostao trajni trag o postojanju greške tj. kako bi se ta greška od strane programera, koji je zadužen za dio igre gdje se nalazi greška, mogla ispraviti [2]. Primjeri mogućih elemenata koje je potrebno dokumentirati su: općeniti podaci o grešci, način kako reproducirati događaj koji je doveo do te greške, moduli koje je greška zahvatila, učestalost pojavljivanja, vrijeme kada je greška pronađena i zabilježena, slike (eng. Screenshots) greške itd. Također, postoje različite vrste testiranja kao što su: funkcionalno testiranje, testiranje modula, testiranje performansi, testiranje zauzeća memorije, testiranje opterećenosti na sustav itd.

Opisano testiranje tj. testiranje od strane zaposlenika se još naziva i alpha testiranje. Kao što je već opisano, to je testiranje koje provode zaposlenici na samom završetku razvoja igre gdje nastoje oponašati korisnikovo ponašanje. Sljedeći oblik testiranja u životnom ciklusu razvoja se odnosi na beta testiranje. Za razliku od alpha testiranja, beta testiranje vrše sami krajnji korisnici u stvarnom okruženju. S obzirom da korisnici prilikom igranja jedino mogu uočiti greške koje nastaju prilikom npr. rušenja igre i s obzirom na to da korisnici nemaju pristup

programskom kodu igre te ostalim elementima korištenim u razvoju, to testiranje se još naziva i testiranje crne kutije (eng. Black box testing) [4].

Nakon završetka alpha i beta testiranja, prelazi se na produkciju i marketing. Produkcija predstavlja proces izdavanja igre kupcima, dok se marketing odnosi na samo promoviranje igre pomoću različitih metoda kao što je reklamiranje po društvenim mrežama, izradom kratkih videa o samoj igri (eng. Trailer) i slično. Tim za marketing također sakuplja povratne podatke o samoj igri od strane kupaca kao što su ocjene igre i komentari o kvaliteti proizvoda, te prosljeđuje navedene podatke ostatku tima kako bi se mogle napraviti potrebne promjene ili povećati kvaliteta budućih proizvoda [2].

2.2. Elementi koji utječu na trošak razvoja igara

Trošak razvoja igre ovisi o pojedinim slučajevima, no ovdje ćemo troškove generalizirati tako što ćemo navesti elemente troškova koji se odnose na većinu igara prilikom razvoja. Na primjer, postoje znatne razlike između manjih „indie“ igara, velikih „AAA“ igara te „MMO“ igara. Najveći faktor koji utječe na cijenu razvoja igre je sama kompleksnost igre. Na primjer, razvoj manje igre za mobilne platforme može biti u rasponu do 500 dolara, dok velika igra kao što je GTA 5, koja se razvija za više različitih platformi, može rasti do 265 milijuna dolara [5]. S obzirom da ćemo u ovom poglavlju ukratko opisati neke moguće elemente koji povećavaju troškove razvoja, trebamo ih podijeliti u nekoliko kategorija:

- Razvoj tj. plaće razvojnog tima
- Licence programa potrebnih za razvoj
- Intelektualno vlasništvo tj. cijena kupnje prava na korištenje određenih brendova, poznatih likova itd.
- Kupnja odgovarajuće opreme za rad
- Eksponencijalni rast troška prilikom razvoja s obzirom na kompleksnost igre

Kao što je već navedeno, ovaj popis kategorija je okviran te ne predstavlja u potpunosti stvarnu situaciju s obzirom da je nemoguće obuhvatiti troškove razvoja svih igara te smo se zbog toga orijentirali samo na najčešće elemente koji utječu na troškove dok stvarni ukupni troškovi ovise od slučaja do slučaja.

Prva stavka na popisu je plaća razvojnog tima. Kao što je već prije u radu opisano, razvojni tim se dijeli na nekoliko kategorija. Tako na primjer, dizajneri igre se mogu poistovjetiti s projekt menadžerima s obzirom na njihovu važnost u timu. Na manjim projektima je dovoljno da sudjeluje jedan dizajner, dok će za veće projekte postojati cijeli tim čime će se i troškovi

znatno povećati. Kako bi pravilno i efikasno koristili grafičke pokretače kao što je to Unreal Engine ili Unity Game Engine, potrebni su kvalitetni programeri koji će moći razvijati programske skripte za rad. Ukoliko želimo razvijati igre za više platformi, potrebno je više programera koji se specijaliziraju za pojedine platforme kao što je Android, iOS, Xbox itd. U pogledu vizualnog dizajna su nam potrebni 2D i 3D dizajneri. Uz navedene dizajnere, u timu je potrebno imati i animatora koji će kvalitetno i efikasno kreirati sve potrebne animacije. Na manjim projektima je moguće zaposliti jednu osobu koja ima dovoljno vještine da odradi sve navedene poslove, dok je na većim projektima potrebno zaposliti specijalizirane ljude ovisno o pojedinim područjima rada. Svaka igra također ima postavljen zvučni dizajn. Pojedini manji projekti mogu kupiti gotove efekte za manju cijenu dok veći projekti ipak zahtijevaju specijalizirane zvučne inženjere i kreatore koji će omogućiti da se igra izdvaja od ostatka, no naravno, i to će znatno podići cijenu razvoja. Nakon završetka razvoja, potrebno je provesti testiranje. Na jako malim projektima možda nije potrebno zaposliti zasebne testere, no ako je projekt srednjeg ili većeg obujma, potrebno je temeljito testirati sve elemente igre kako bi se prije same produkcije uklonile greške [5].

Sljedeća kategorija se odnosi na cijenu licenci za programe potrebne za razvoj. Ukoliko se radi na manjim projektima, postoji vjerojatno da će se moći koristiti besplatne alternative za rad. No u bilo kakvim većim projektima to nije slučaj te je potrebno kupiti odgovarajuće programe kao što su npr. Maya, 3D Max, Photoshop itd. Kao primjer možemo uzeti Unity3D Pro verziju koja košta 125 dolara mjesečno [6]. Također, moguće je i korištenje određenih usluga u igrama kao što je npr. Google Firebase čije cijene variraju ovisno o količini korištenja pojedinih stavki tj. omogućuju usputno plaćanje (eng. Pay as you go) [7]. Prilikom korištenja takvih usluga je potrebno uzeti u obzir najgoru situaciju tj. situaciju koja će imati najviše financijskog tereta. Uz same programe, moguće je kupiti i određene biblioteke, predloške i ostali sadržaj koji bi olakšao te ubrzao rad [5].

Ukoliko želimo koristiti određeni brand iz stvarnog života u igri, morat ćemo kupiti pravo na to. Npr. ukoliko je igra orijentirana na auto-moto utrke, logično je da će igrači željeti vidjeti stvarne brendove automobila u igri, te samim time se javlja znatan trošak na strani razvoja igre. Takvi troškovi nisu prikladni za manje igre tj. manje razvojne studije te se oni nastoje izbjegavati. Čak i razvojni studio čije igre možda nisu orijentirane u potpunosti na prikaz brendova u igri, pokušavaju izbjeći korištenje stvarnih naziva. Primjer toga bi bio GTA serijal igara razvojnog studija Rockstar games koji, unatoč velikoj popularnosti serijala, ni danas ne koristi stvarne nazive automobila u svojim igrama iako je puno puta u igrama očito o kojim se autima iz stvarnog svijeta radi.

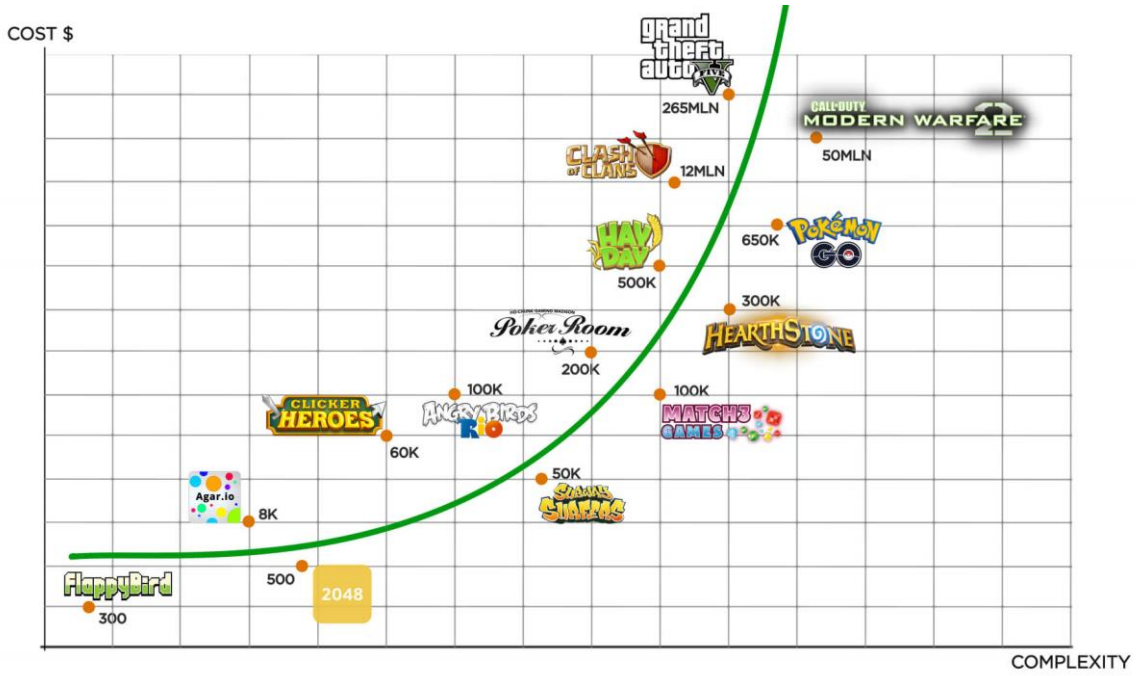
Za razvoj elemenata igre potrebna nam je i odgovarajuća fizička oprema uz već navedenu digitalnu. Kako bi mogli ispravno testirati proizvod za platforme na kojima ga želimo distribuirati, potrebno je ujedno i posjedovati više primjeraka odabranih platformi kao što je PC s potrebnom minimalnom konfiguracijom, PlayStation konzola, Xbox konzola itd. Za kreiranje sadržaja je potrebna oprema kao što su kamere, osvjetljenja, oprema za snimanje pokreta (eng. Motion tracker) itd. Količina opreme i njena kvaliteta ovisi o samom obujmu igre na kojoj se radi [5].

Zadnja kategorija na našoj listi se odnosi na trošak razvoja koji raste s vremenom. Vrijeme potrebno za konačan razvoj igre se može kretati od nekoliko mjeseci pa do nekoliko godina te se samim time može i ukupni trošak kretati od otprilike recimo, 200 dolara za manje igre pa do više od 200 milijuna dolara. Prilikom razvoja se pojavljuju dodatni troškovi. Recimo da krenemo s manjom idejom koja će se razvijati s vremenom, te time zaposlimo jednog programera i jednog dizajnera. S vremenom prilikom razvoja će se pojavljivati sve veći zahtjevi koje će trebati zadovoljiti. Pojavit će se potreba za više programera te možda i potreba za iznajmljivanjem dodatnog prostora za njihov rad. Isto vrijedi i za dizajnere. Možda će se pojaviti programski ili dizajnerski problem koji će moći riješiti samo novi član tima. Prilikom rada će se pojavljivati nove ideje koje možda odabrani grafički pokretač neće moći zadovoljiti te će njegove funkcionalnosti prvo trebati proširiti kako bi se mogao nastaviti daljnji rad. Vizualni dizajn je također podložan promjenama te isto tako količina grafičkih elemenata koji će biti korišteni u igri. Zbog toga se prilikom razvoja može javiti potreba za smanjenjem kvalitete elemenata, ili suprotno, povećanjem, što zahtjeva primjenu naprednih metoda optimizacije koje je potrebno prvo istražiti i proučiti kako bi se mogle primjenjivati sukladno tome. Još jedan problem su poslužitelji, koje treba optimizirati za rad s većom količinom podataka, ukoliko igra ovisi o njima, te se njena kompleksnost s vremenom povećava. Testiranje na manjim projektima je izvedivo pomoću par članova, no ako projekt raste s vremenom, potrebno je zapošljavati nove testere koji će ujedno i izrađivati dokumentaciju o testnim slučajevima i rezultatima. Javlja se i problem prilagodbe programskog koda za više platformi ukoliko se ne radi o grafičkom pokretaču koji ne podržava rad za više platformi (eng. Cross platform) [5].

Kao što je već prije navedeno, elementi koji utječu na troškove razvoja igara ovise o pojedinim projektima, te ih je nemoguće u potpunosti generalizirati. Ovdje smo sada naveli elemente koji su najčešći na većini projekata. Usporedbu troškova razvoja nekih popularnijih igara možemo vidjeti na sljedećoj tablici:

Naziv igre	Cijena (dolar)
Flappy bird	300
2048	500
Agrar.io	8.000
Clicker Heroes	60.000
Subway Surfers	50.000
Soccer Stars	300.000
Angry Birds Rio	100.000
Match 3	100.000
Poker Room	200.000
HearthStone	300.000
Hay Day	500.000
Pokemon Go	650.000
Clash of Clans	12.000.000
Call of duty: Modern Warfare	50.000.000
GTA V	265.000.000

Tablica 1: Usporedba troškova razvoja igara [5]



Slika 3: Trošak razvoja igara [8]

2.3. Tehnologije i pristupi razvoju igara

U ovom poglavlju ćemo se orijentirati na metode i tehnike koje se odnose na sami razvoj igara u modernom okruženju. Igre mogu biti različitog sadržaja, namjene i kompleksnosti te ćemo se prema tome orijentirati na različite pristupe i tehnologije koje se koriste za rad na razvoju igara. U modernom svijetu, industrija igara predstavlja jedan od najznačajnijih oblika zabave te je kao takva jako raširena. Svaki dan se razvijaju novi pristupi radu na igrama te nove tehnologije kako bi ostvarile željene karakteristike. Prema tome, s obzirom na veliku količinu tehnologija, nije moguće ih sve istražiti i obuhvatiti u ovom radu, te ćemo se ovdje orijentirati samo na neke od najkorištenijih tehnologija.

2.3.1. Pokretači igara (eng. Game engine)

Prilikom razvoja bilo kojeg oblika igre, potrebno je odabrati odgovarajući grafički pokretač (eng. Game engine). Tržište nudi veliku selekciju pokretača te svaki od njih ima svoje prednosti i mane. Grafički pokretač predstavlja okvir (eng. Framework) za kreiranje video igara. On kao takav nudi različite mogućnosti, sve od rada s animacijama do rada s umjetnom inteligencijom. Grafički pokretači su odgovorni za prikazivanje grafičkih elemenata, rad s kolizijama objekata, upravljanje memorijom i te nude mnoge druge mogućnosti. Većina grafičkih pokretača se sastoji od nekoliko osnovnih elemenata [9]:

1. Glavni program koji sadrži logiku igre
2. Grafički pokretač koji služi za prikaz 2D/3D prostora scene za rad
3. Pokretač za zvučne efekte sa svim potrebnim zvučnim algoritmima
4. Pokretač za simuliranje fizičkog ponašanja objekata
5. Modul za rad s umjetnom inteligencijom

Kako bi se započeo razvoj igara, potrebno je odabrati odgovarajući pokretač. Samo neki primjeri od popularnijih pokretača bi bili Unity, Unreal Engine, RPG Maker, OGRE Engine, Gadot, Game Maker studio. Ukratko ćemo opisati prednosti i nedostatke pojedinih pokretača kako bi se lakše usvojile njihove razlike.

Prvi na listi je Unity. Prednost Unity pokretača je u tome što je dobar za početnike te će kasnije u ovom radu i biti korišten kako bi se napravila jednostavna 2D platformer igra. Podržava izvođenje igara na više različitih platformi (eng. Cross platform), ima odličan repozitorij već gotovih elemenata spremnih za korištenje prilikom razvoja (eng. Asset), besplatan je za korištenje u početku za jednostavne projekte, te cijena kasnije ne raste značajno (najviše 150 dolara mjesečno). No isto tako postoje određene mane. Projekti tj. igre na kojima se radi pomoću Unity-a zauzimaju puno prostora na računalu te čak i male igre imaju

velike .exe izvršne datoteke. Još je jedan nedostatak potreba za obaveznim plaćanjem „Pro“ verzije ukoliko izrađena igra zarađuje više od 100.000 dolara godišnje. Iako cijena „Pro“ verzije nije velika, ovo može predstavljati prepreku studiju koji ne želi plaćati naknade za korištenje pokretača u svojim igrama [10].

Sljedeći pokretač na listi je Unreal Engine. Jedna od velikih prednosti navedenog pokretača je njegova popularnost. Koristi se za izradu velikih i popularnih igara kao što su „Unreal Tournament“ i „Squad“ među ostalima. Sadrži velik broj alata tj. mogućnosti za jednostavnije kreiranje sadržaja. Koristi efikasan 3D pokretač za prikaz scena. Nudi se novčana podrška u obliku stipendija (eng. Grants) manjim razvojnim studijima koji su odlučili koristiti Unreal Engine na svojim projektima. Nedostatak se javlja u obliku potrebe za licenciranom kopijom programa tj. kreatori koji namjeravaju koristiti pokretač za financijsku dobit moraju se odreći 5% od ukupne dobiti [11]. Nadalje, iako ovaj pokretač nudi velike mogućnosti, to predstavlja problem za početnike zbog same kompleksnosti, te je kao takav više namijenjen timskom razvoju [10].

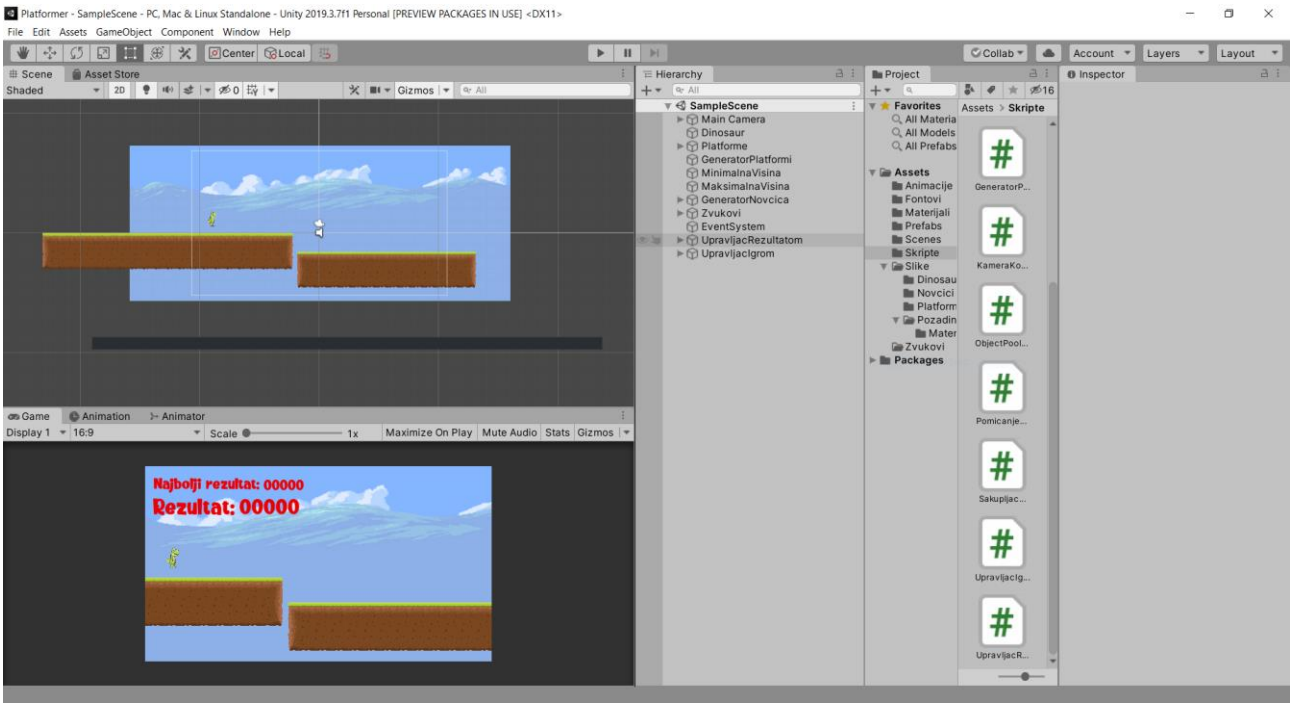
RPG Maker predstavlja jednostavan alat za učenje na području razvoja igara. Naspram Unity-a i Unreal Engine-a, RPG Maker je stariji pokretač čije je prvo izdanje objavljeno 17. prosinca 1992. godine. Također, naspram prijašnje opisanih pokretača, RPG Maker je namijenjen izradi samo 2D RPG (eng. Role playing game) igara. Njegova prednost je u tome što je jako jednostavan te je odličan izbor za početnike koji žele razvijati jednostavne 2D RPG igre. Nedostaci se javljaju u cijeni. Naime, iako je star i namijenjen za jednostavne projekte, nije besplatan. Cijene pojedinih izdanja se kreću i do 80 dolara [12]. Također, drugi noviji pokretači mogu napraviti čak i bolji posao jer nude više mogućnosti [10].

OGRE Engine je isto nešto stariji pokretač otvorenog koda (eng. Open source) čije je prvo izdanje objavljeno u veljači 2005. godine. Za razliku od ostalih pokretača igara, OGRE Engine je primarno namijenjen da se koristi samo kao grafički pokretač te kao takav ne nudi sve mogućnosti koje nude ostali opisani pokretači igara. Zbog svoje starosti je više programski otvoren te zahtjeva više pisanja programskog koda kako bi se ostvarile određene mogućnosti. Zbog toga se javlja prednost fleksibilnosti prilikom ostvarivanja željenih funkcionalnosti, ali većina modernih pokretača ionako nudi već gotov programski kod za skoro sve osnovne a i neke naprednije funkcionalnosti. Također je težak za učenje i nije jednako dobro dokumentiran kao ostali pokretači. Jedna od prednosti je u tome što je u potpunosti besplatan [10].

Isto kao i OGRE Engine, Gadot predstavlja pokretač otvorenog koda. Omogućuje izradu 2D i 3D igara te pruža veliku fleksibilnost. Prednost se javlja u korištenju MIT licence [13] koja omogućuje kreatoru da zadrži sva prava na svoj rad i ne mora plaćati nikakve naknade kasnije te je sav rad u pokretaču besplatan. Prvo izdanje pokretača je objavljeno u

veljači 2014. godine te je redovito ažuriran. Zadnje stabilno izdanje za vrijeme pisanja ovog rada je objavljeno 10. ožujka 2020. s verzijom 3.2.1. Nedostatak predstavlja njegova kompleksnost i kao takav nije jednostavan za učenje. Zajednica koja ga koristi je mala s obzirom na to da je star tek nekoliko godina te dokumentacija još nije na razini nekih većih pokretača [10].

Zadnji pokretač na našoj listi je Game Maker studio. Primarni fokus ovog pokretača je razvoj 2D igara. Pisan je u JavaScript-u i jednostavan je za učenje. Prvo izdanje se je pojavilo 1999. godine no i dalje se redovito ažurira. Pruža mogućnost izvoza (eng. Export) igara za skoro sve popularne platforme. Nedostatak se javlja u cijeni. Nije besplatan za korištenje te njegova cijena bez obzira na starost, raste i do 1500 dolara mjesečno. I baš zato što je namijenjen radu s 2D igrama, cijena predstavlja još veći problem s obzirom na to da su 2D igre nešto manje popularne naspram 3D. Također, iako je pokretač star, zajednica koja ga koristi je i dalje mala naspram pokretača kao što su Unity ili Unreal Engine [10].



Slika 4: Vlastiti projekt u Unity pokretaču prilagođenom za rad sa 2D igrama

2.3.2. Programi za 3D modeliranje

U prijašnjem poglavlju smo naveli kako neki pokretači u sebi sadrže repozitorije gotovih 2D i 3D elemenata koje kreator može koristiti za rad na svojoj igri. No kako bi igra bila jedinstvena, potrebno je kreirati svoje elemente (objekte) za igru. Također, navedeni repozitoriji obično sadrže samo nekoliko primjera elemenata kao što je: lik osobe, poneko vozilo, neki veći objekt kao što je kuća itd. Navedeni primjeri obično sadrže neke osnovne karakteristike za rad u navedenom pokretaču koje služe samo kako bi ih kreator mogao proučiti da bi proširio svoje znanje. Veliki broj manje poznatih pokretača uopće ni ne sadržava repozitorij. Zbog toga je potrebno razviti samostalno 2D/3D elemente za rad iako se u moderno doba većinom radi s 3D-om te ćemo se ovdje fokusirati na programe za 3D. Neki od najpopularnijih programa za razvoj 3D elemenata bi bili: Maya, 3DS Max i Blender.

Maya predstavlja proizvod razvijen od strane Autodesk-a te je inicijalno zamišljen kao program za razvoj animacija i tekstura te je tek kasnije za životnog vijeka programa dodana mogućnost razvoja modela. Također, Autodesk je razvio i 3DS Max čija je primarna namjena bila razvoj modela u odnosu na Mayu gdje je to dodano tek kasnije kao mogućnost. Ukratko, kod 3DS Maxa, situacija je obrnuta u odnosu na Mayu u smislu da je 3DS Maxu dodana mogućnost animiranja ali je primarno fokusiran na rad s modelima. Treći program za 3D modeliranje bi bio Blender razvijen od strane Blender Foundation-a. To je program otvorenog koda namijenjen razvoju modela, animacija, vizualnih efekata itd. [14]

Usporedbu navedenih 3D programa s obzirom na određene elemente možemo vidjeti na sljedećoj tablici:

ELEMENTI USPOREDBE	**MAYA**	**3DS MAX**	**BLENDER**
DEFINICIJA	Razvijen od strane Autodesk-a i primarno namijenjen za rad na animacijama i teksturama modela	Razvijen od strane Autodesk-a i primarno namijenjen modeliranju, dizajnu arhitekture, inženjerstvu i konstrukciji	Razvijen od strane Blender Foundation-a i primarno namijenjen animaciji i vizualnim efektima
KOMPATIBILNOST	Windows i MAC platforme	Windows i MAC platforme	Windows, Linux, MAC i ostale platforme

SUČELJE	Komplicirano sučelje što zahtjeva vođenje prilikom učenja	Naspram Maye, sučelje znatno jednostavnije	Komplicirano učenje te zahtjeva određeno vrijeme za privikavanje
KORISNIČKO ISKUSTVO	Težak proces učenja, ali jedan od najboljih 3D programa	Lakši proces učenja u odnosu na Mayu jer već nakon par koraka su vidljivi rezultati	Nešto lakši i pristupačniji proces učenja u odnosu na Mayu
MOGUĆNOSTI	Sadrži veću količinu mogućnosti za rad sa 3D animacijama i teksturama	Sadrži veću količinu mogućnosti za rad sa 3D modelima i dizajnom	Više mogućnosti orijentirane na vizualne efekte i animaciju
DOSTUPNOST	Licenciran program	Licenciran program	Besplatan program
UPOTREBA	Najvećim dijelom veliki produkcijski studiji	Razvoj arhitekture i igara	Najbolje primjeren malim „start-up“ tvrtkama orijentiranim na 3D animaciju i efekte

Tablica 2: Usporedba nekih od najpopularnijih 3D programa [14]

2.3.3. Programi za grafički dizajn

Prilikom razvoja modela i animacija za igre, potrebno je također odabrati odgovarajuće teksture za kreirane modele. Mnoge web stranice nude mogućnost preuzimanja već gotovih tekstura. Poneke teksture su besplatne, no većina se plaća. Ali puno puta razvojni studiji ne mogu pronaći odgovarajuće teksture kako bi zadovoljili svoje potrebe te naprosto trebaju kreirati svoje teksture. Također, neke kupljene i preuzete teksture mogu samo djelomično zadovoljiti potrebe te ih je pomoću odgovarajućih programa potrebno urediti te ćemo ovdje navesti neke od najkorištenijih programa za kreiranje i uređivanje tekstura. Teksture ukratko predstavljaju slike koje „omataju“ 3D model kako bi on poprimio željeni izgled. Teksture mogu biti kreirane fotografiranjem objekata iz stvarnog svijeta te prijenosom u 3D okruženje uz neke

dorade ili mogu biti kreirane u potpunosti od početka korištenjem skupa alata ili skupa već postojećih slika.

Jedan od najpopularnijih alata za uređivanje slika bi naravno bio Adobe Photoshop. Photoshop je naširoko korišten alat od strane profesionalnih grafičkih dizajnera i fotografa. Prvo izdanje se je pojavilo još 1990. godine te je i danas jedan od najkorištenijih alata. Omogućuje stvaranje različitih ilustracija, izmjene na fotografijama, kreiranje 3D ilustracija te čak omogućuje i uređivanje video zapisa i stvaranje animacija. Nedostatak Photoshopa se javlja u njegovoj cijeni koja funkcionira principu mjesečne pretplate koja se kreće od 9 dolara mjesečno pa sve do 52 dolara, ovisno o odabranom paketu [15]. Navedena mjesečna pretplata može predstavljati ograničenje za manje razvojne studije čiji je budžet daleko manji od budžeta većih studija.

S obzirom da Photoshop zbog svoje cijene može predstavljati ograničenje manjih studijima, kao alternativno rješenje se javlja GIMP koji je u potpunosti besplatan. GIMP predstavlja program za manipulaciju slikama koji ujedno radi na više platformi kao što su: Windows, Linux, BSD, Solaris, macOS. GIMP je program otvorenog koda koji podržava iznimno velik broj dodataka (eng. Plugin) čime se njegove mogućnosti mogu neprestano nadograđivati. Samim time predstavlja jednu od najboljih besplatnih alternativa Photoshopu [16].

Još jedna besplatna alternativa bi bila Inkspace. Inkspace je isto kao i GIMP, program otvorenog koda. Omogućuje lakši rad sa vektorskim slikama. Vektorska grafika predstavlja dizajn kreiran pomoću točaka, linija i krivulja koje se temelje na matematičkim jednadžbama. Prednost vektorske grafike naspram rasterskoj je u tome što, bez obzira razinu promjene veličine slike, ona neće biti zamućena (eng. Blurry), kao što je to slučaj sa rasterskom grafikom [17]. Neke od istaknutijih mogućnosti koje podržava Inkspace bi bile: transformacije objekata, grupiranje objekata, kreiranje slojeva (eng. Layers), poravnanje i raspoređivanje objekata, booleove operacije, tekst unutar oblika itd. [18]

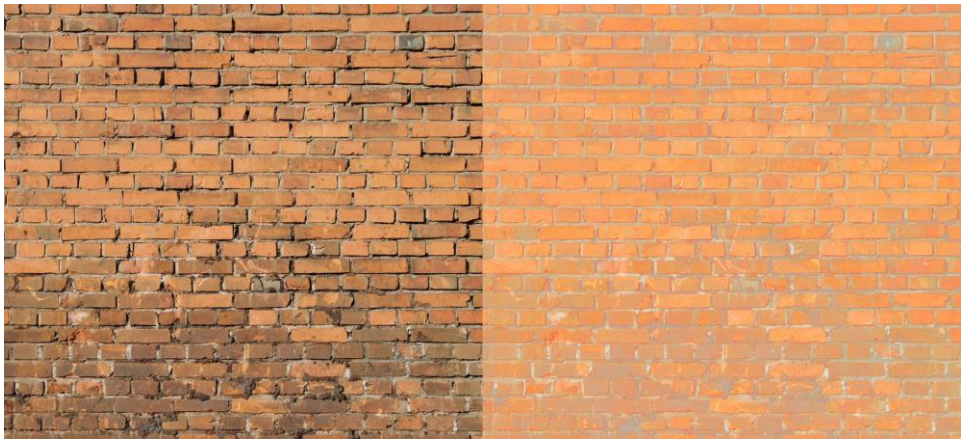
Za profesionalni rad se možemo ponovo okrenuti plaćenim programima kao što je Photoshop. Primjer jednog takvog programa bi bio Pixelmator. Svojom cijenom je ipak nešto jeftiniji od Photoshopa ali i ponešto manjih mogućnost. Postoji još nekoliko verzija Pixelmatora uz običnu: verzija za iOS, Photo i Pro. Cijene se kreću od 4.99 dolara za mobilnu verziju pa sve do 39.99 dolara za „Pro“ verziju [19]. Kao što je navedeno, postoji i mobilna verzija no ona vjerojatno neće biti od interesa ozbiljnijim grafičkim dizajnerima. Također, za razliku od prije navedenih programa, Pixelmator je dostupan samo sa iOS i Mac uređaje.

2.3.4. Metode za postizanje većeg realizma u igrama

U ovom poglavlju ćemo se orijentirati na neke naprednije metode i tehnike koje se koriste prilikom razvoja igara kako bi se olakšao rad na njima.

U ovom radu smo već opisali programe koje koriste dizajneri tekstura u igrama. Prilikom svojeg rada s teksturama, koriste se različiti tipovi mapa tekstura kao slojevi koje se primjenjuju kako bi tekstura dobila realističniji odraz kao jedna od naprednijih tehnika. Postoji veliki broj mapa te ćemo ovdje opisati samo neke koje su najkorištenije. Ovim pristupom dizajnu tekstura se može ostvariti znatno veća realnost uz minimalne troškove na performanse omogućujući stvaranje dodatnih vizualnih efekta. Bitna stvar je da grafički pokretači moraju biti unaprijed pripremljeni za ove mogućnosti, no većina modernih pokretača već je.

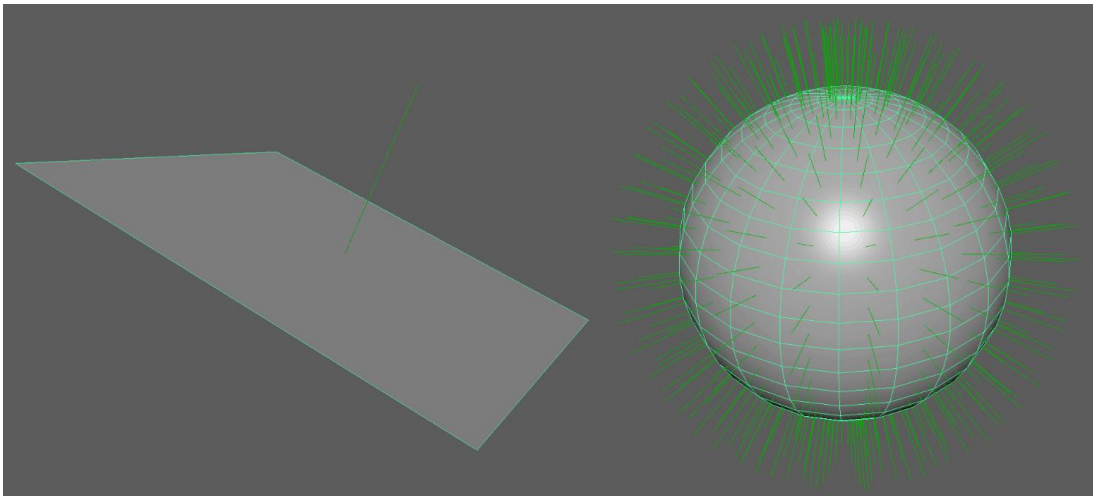
Prva mapa na koju ćemo obratiti pozornost je „Albedo“ mapa. Ona predstavlja osnovu za daljnji rad na materijalu. Definira boju teksture (ili fotografiju) koju želimo koristiti na našem modelu no bez dodatnih znatno naglašenih tamnih ili svijetlih elemenata. Ukratko, „Albedo“ mapa je tekstura kreirana od obične boje ili slike bez ikakvih dodatnih sjena i naglašenih dijelova. Kao takva predstavlja osnovu za jednostavniji rad s drugim mapama koje će se primjenjivati na teksturu modela [20]. Primjer takve mape možemo vidjeti na sljedećoj slici:



Slika 5: Albedo mapa kreirana iz fotografije zida [20]

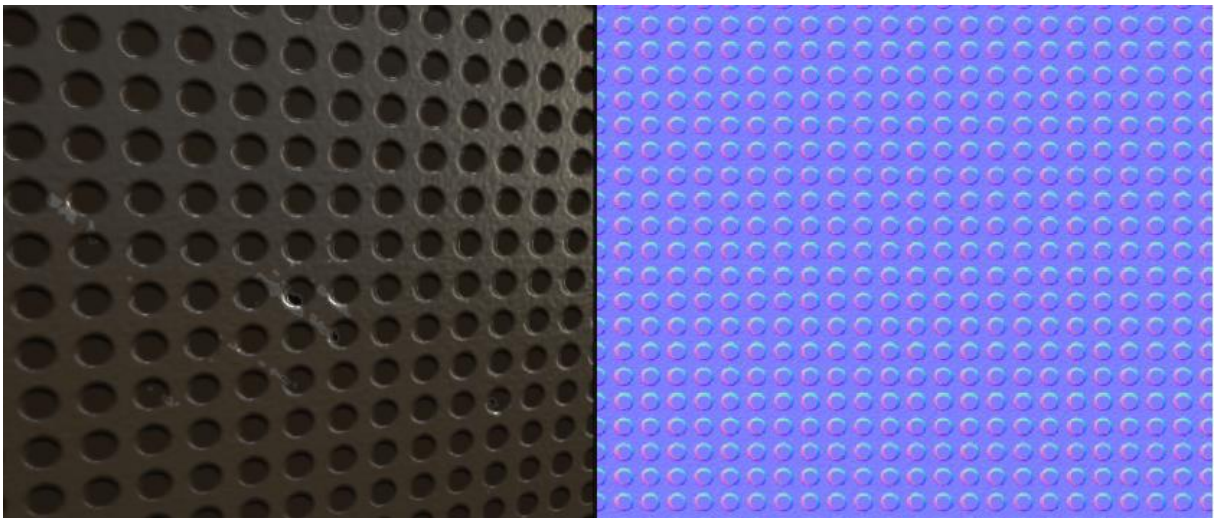
Naredna mapa se naziva „normal map“. Primjena navedene mape omogućuje simulaciju izbočina i udubina na teksturama bez potrebe da se 3D model naknadno oblikuje. Samim time se ne gube dodatne performanse prilikom izrade modela, odnosno ne povećava se broj poligona. U geometriji, riječ „normala“ predstavlja pravac ili vektor koji je okomit na objekt. Svaki poligon u video igrama ima normale na površini koje omogućuju vršenje kalkulacija za osvjetljenje objekta. Kut između normale na površini i smjera odakle dolazi svjetlost u grafičkom pokretaču se koristi kako bi se odredila „jačina“ sjene na modelu tj. koliko

će sjena biti izražena [21]. Primjer normala na površini 3D modela možemo vidjeti na sljedećoj slici:



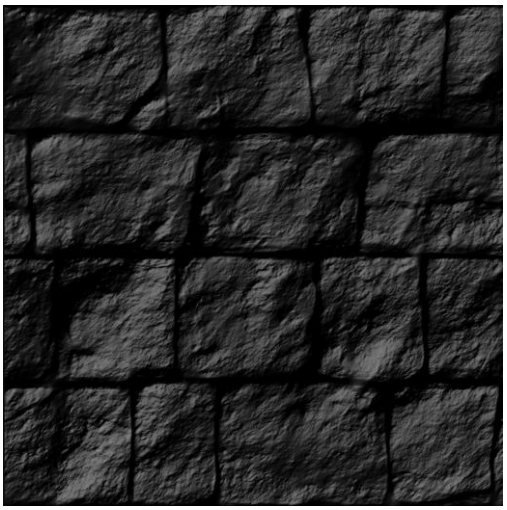
Slika 6: Prikaz normala na površini 3D modela [21]

Iako su navedene mape zapravo 2D slike, one mogu spremiti određene 3D informacije. RGB kanali u „normal“ mapama odgovaraju x,y i z koordinatama normala na površini objekta. Iako se ove mape ne mogu koristiti kako bi se prikazale jako velike udubine i velika izbočenja na modelu, dovoljne su kako bi se prikazala manja ispupčenja na površini te manje udubine. Geometrija modela se pomoću „normal“ mapa ne mijenja te ostaje ista tj. stvara se iluzija navedenih promjena. Kao što je već prije navedeno, grafički pokretači moraju biti spremni za rad s ovakvim oblicima mapa te to isto vrijedi i za „normal“ mape kako bi iz nje mogao očitati vrijednosti RGB kanala kao x,y i z vrijednosti i samim time simulirati odgovarajuće osvjetljenje na pojedine dijelove modela. Primjer primjene „normal“ mape i njenog utjecaja na osvjetljenje možemo vidjeti na sljedećoj slici:



Slika 7: Utjecaj koji „normal“ mapa ima na teksturu [21]

Opisat ćemo ukratko još jednu mapu koja se često koristi prilikom razvoja modela za igre, a to je „specular“ mapa tj. reflektirajuća mapa. Pomoću nje definiramo razinu do koje određene tekstura stvara reflektirajuće efekte pod utjecajem svijetla u grafičkom pokretaču tj. na kraju i u samoj igri. Princip rada se temelji na stvaranju crno-bijele teksture iz već postojeće koju želimo koristiti. Pikseli navedene kreirane teksture sadržane određene vrijednosti (od crnog prema bijelom) te se pomoću toga određuje razina osvjetljenja objekta. Ukratko, što je veća razina bijele boje, to će se to mjesto na teksturi više reflektirati na svjetlosti. Npr. zato će reflektirajuće mape tekstura objekata kao što su kamene površine biti najvećim dijelom crne s obzirom na to da kamene površine jako slabo odbijaju svjetlost [22]. Primjer reflektirajuće mape možemo vidjeti na sljedećoj slici:



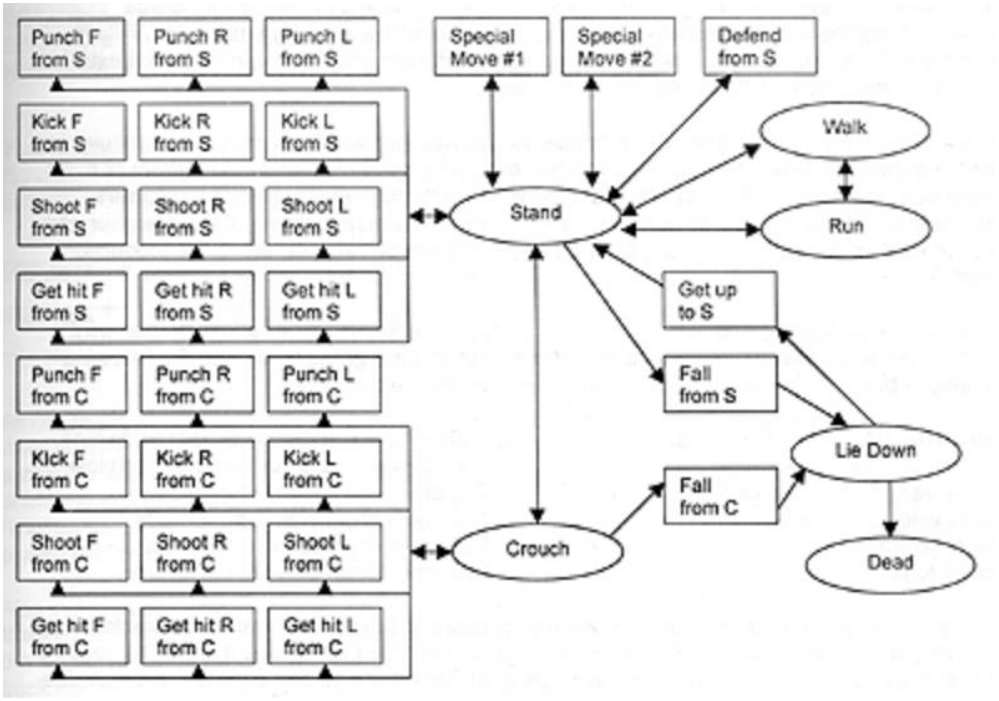
Slika 8: Reflektirajuća („specular“) mapa kamene površine [23]

Sada kada smo opisali neke načine za rad s teksturama u 3D grafičkim pokretačima pomoću kojih je moguće postići veći realizam, opisat ćemo pristup stvaranja animacija pokreta pomoću napredne tehnike hvatanja pokreta (eng. Motion capture). Stvaranje velikog broja pokretnih animacija, naročito animacija pokreta čovjeka, može biti zamoran i skup proces za animatore. Prema tome, razvijena je tehnologija za snimanje pokreta koja je znatno olakšala posao. Navedena tehnologija se ne koristi samo u industriji igara već i u filmskoj, vojnoj, sportskoj itd. U filmskoj industriji i industriji video igara ima još naziv „matchmaking“ [24]. Motion capture nije u potpunosti zamjena za proces animiranja. Tehnologija služi samo za snimanje pokreta koji se može koristiti prilikom animiranja u 3D programima. Također, navedena tehnologija nudi neke prednosti i nedostatke:

PREDNOSTI	NEDOSTATCI
Brzi rezultati	Zahtjeva poseban hardware za snimanje pokreta
Smanjuje trošak klasične animacije	Ukupna cijena hardware-a, programa i osoblja za snimanje nije pogodna za manje studije
Omogućuje lakše testiranje tj. isprobavanje pokreta prije nego što se nastavi sa animacijom	Sustav za hvatanje pokreta može imati dodatne zahtjeve za rad kao što je veća prostorija za snimanje
Omogućuje kreiranje realističnijih pokreta	Ako nastane problem prilikom snimanja, lakše je ponovo snimati scenu nego manipulirati podacima kako bi se problem popravio
Količina podataka za animaciju je znatno veća u odnosu na klasični pristup	Potrebne naknadne dorade u programima kako bi se ostvarila potpuna animacija
Može se koristiti sa različitim neslužbenim programima što može smanjiti troškove	Pokreti su ograničeni zakonima fizike
	Ako je model na računalu različitih proporcija u odnosu na model koji je sniman, mogu se stvoriti artefakti

Tablica 3: Neke prednosti i nedostaci „motion capture“ tehnologije [24]

Kako bi se mogli snimiti pokreti, potrebno je proučiti sadržaje igre na kojoj se raditi te napraviti detaljan tok pokreta koje je potrebno snimiti. Ako se npr. radi o igri sa superherojem, potrebno je rastaviti sve pokrete na pojedinačne dijelove i prema tome napraviti dijagram toka prema kojem će se vršiti snimanje pokreta. Primjer jednog takvog dijagrama toko možemo vidjeti na sljedećoj slici:



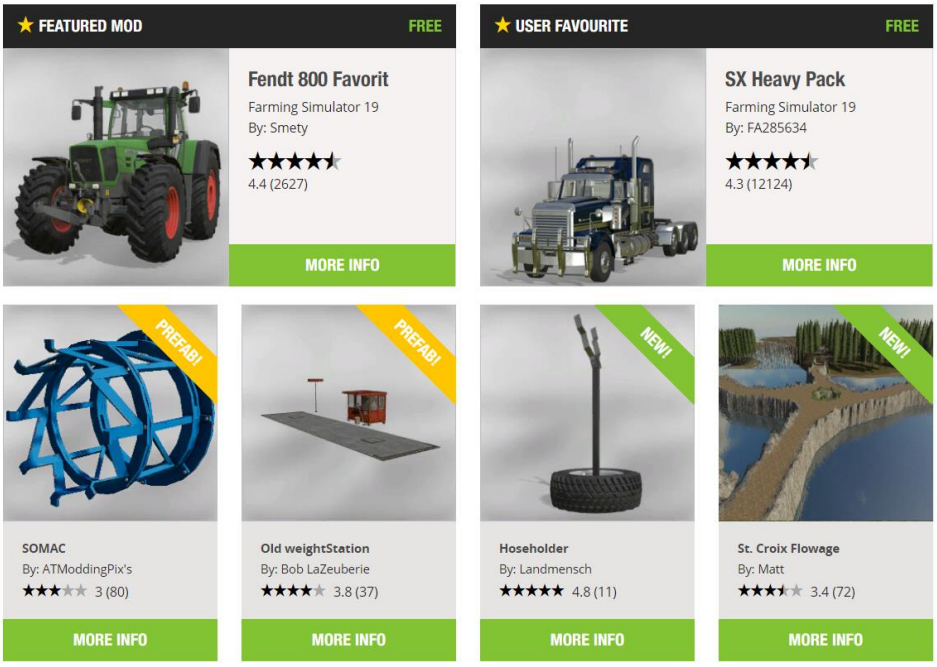
Slika 9: Dijagram toka pokreta koje je potrebno snimiti [25]

Primjer snimanja pokreta scene za akcijsku igru možemo vidjeti na sljedećoj slici. Moguće je vidjeti na glumac drži model puške kako bi se postigla veća realnost pokreta, iako će model puške biti naknadno izrađen u 3D programima nakon što snimanje završi:

Slika 10: Primjer scene snimanja pokreta za akcijsku igru [26]

2.3.5. Kreiranje 3D objekta i njegova upotreba u već postojećoj igri

U ovom poglavlju ćemo ukratko opisati kreiranje i prijenos gotovog 3D elementa i njegov prijenos u već postojeću igru u obliku modifikacije. Zbog ograničenih financijskih mogućnosti, za izradu 3D modela ćemo koristiti već prije opisani program, Bledner, s obzirom na to da je besplatan za korištenje. Najlakši način za prijenos 3D modela u igru i je bilo korištenje igre koja podržava korisničke modifikacije, također znane kao modovi (eng. Mods). Prilikom istraživanja igara odlučeno je da ćemo koristiti igru naziva „Farming Simulator 2019“ tvrtke „Giants Software“. Tema igre se orijentira na poljoprivredu, no to nam trenutno nije toliko bitno. Najbitnija stvar nam je da proizvođač igre službeno podržava korisničke modifikacije. Modifikacije su obično u obliku .zip datoteka koje se jednostavno smjeste u odgovarajuću datoteku naziva „mods“ te ih igra od tamo učitava. Prilikom pokretanja nove igre, igrač ima mogućnost odabira željenih korisničkih modifikacija koje želi koristiti. Modifikacije se mogu preuzeti s velikog broja web stranica, ali najsigurnije mjesto je službeni repozitorij same igre. Korisnici mogu poslati svoje modifikacije timu za testiranje koji onda testira iste provjeravajući određene kriterije koje modifikacije moraju zadovoljiti. Primjeri kriterija bi bili broj grešaka koje modifikacija stvara, broj poligona modela sukladno propisanim standardima itd. Ako modifikacija ne zadovoljava određene kriterije, šalje se nazad kreatoru s popisom problema na prepravljanje. Ako se kriteriji zadovolje, modifikacija se objavljuje sa službenoj web stranici te ju ostali korisnici igre mogu preuzeti i besplatno koristiti:

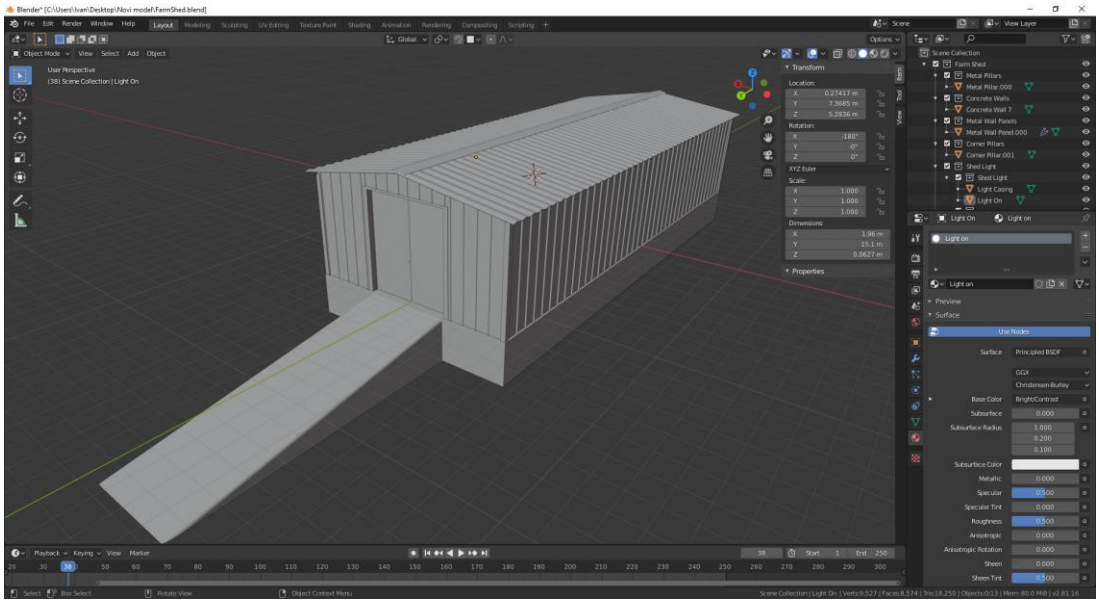


Slika 11: Primjer modifikacija spremnih za preuzimanje sa službene stranice [27]

Naša modifikacija će biti jednostavna i poslužit će samo kao primjer prijenosa korisničkog 3D modela u igru. Kao što je već prije navedeno, za izradu modela će se koristiti

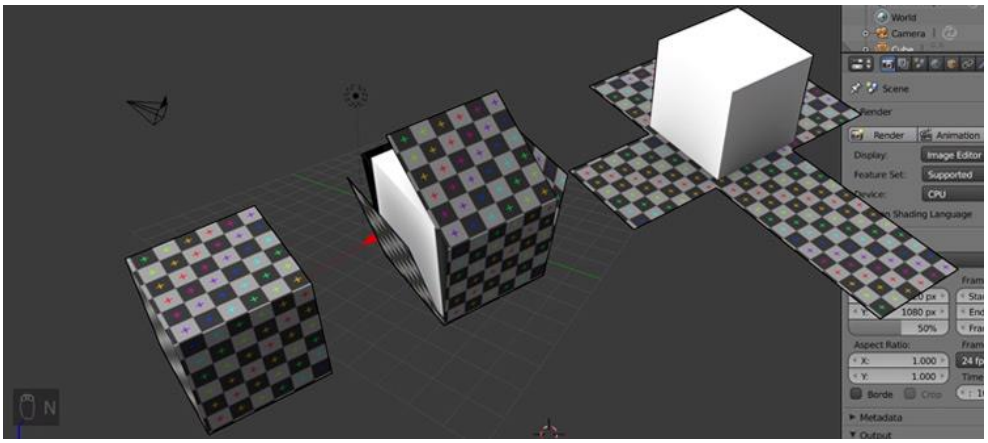
Blender. No s obzirom da bi prenijeli model u samu igru nakon kreiranja, potrebno je koristiti službeni pokretač naziva „Giants Editor“ koji se besplatno može preuzeti na službenoj stranici tvrtke [28]. Sami proces kreiranja modela u Blenderu neće biti opisan, s obzirom da je model jednostavan, te sve što je bilo potrebno za njegovo kreiranje je određeno vrijeme. Jedna od prepreka na koju smo naišli tijekom kreiranja je sami izvoz (eng. Export) kreiranog modela u odgovarajućem formatu. Naime, Blender ima predefinirani skup formata u koji može izvoziti svoje 3D modele. Na samom proizvođaču igre je odgovornost da kreira odgovarajući modul za izvoz (eng. Exporter plugin) za 3D programe koje želi koristiti. Giants Software svoje originalne modele prilikom kreiranja igre radi u Mayi, no s obzirom da želi podržavati korisničke modifikacije, kreirao je i odgovarajući modul za Blender s obzirom na to da je Blender besplatan za korištenje i samim time dostupan svim korisnicima koji nemaju financijske mogućnosti niti želje za plaćanjem profesionalnih 3D programa kao što je Maya. Modul se može također preuzeti sa službene stranice Giants Software-a [28].

S obzirom da je tema igre za koju radimo model poljoprivreda, odlučili smo kreirati jednostavnu ostavu za strojeve. Otišli smo na internet te pregledali nekoliko slika stvarnih objekata te smo odlučili kreirati svoj. Naš objekt ne reflektira u potpunosti niti jedan objekt iz stvarnog svijeta te je plod mašte. S obzirom da se nismo ograničili na taj način te smo mogli raditi što poželimo, sam proces kreiranja nije bio težak. Sve što je bilo potrebno je uzeti malo vremena i naučiti sve potrebne kontrole za modificiranje objekata u 3D prostoru. Prilikom rada na ostavi, kreirali smo nekoliko različitih manjih objekata kao što su: temelji, željezni stupovi, vrata, krovne ploče, betonski blokovi i slično. Pomoću navedenih objekata smo kreirali željeni oblik ostave prikazane na slici:



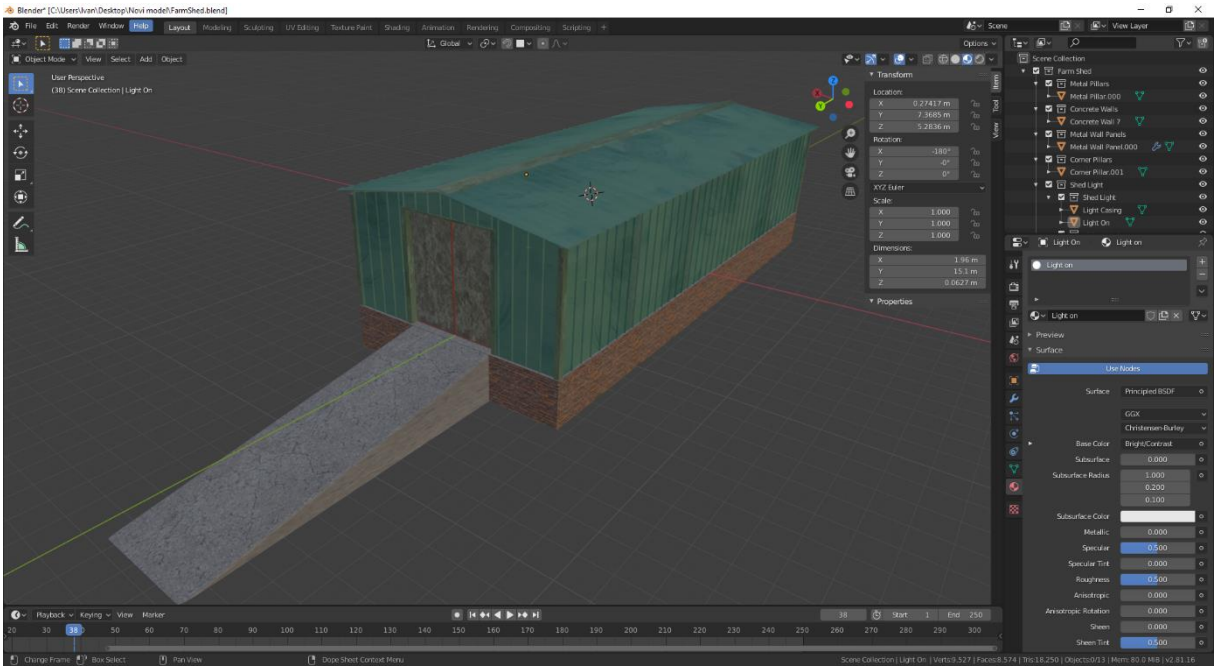
Slika 12: Gotov model ostave bez tekstura

Kao što je moguće vidjeti na gornjoj slici, navedeni model je sive boje što znači da pojedinim dijelovima nisu pridružene teksture. Kako bi dodali teksture pomoću kojih bi model izgledao realno, bilo je potrebno „odmotati“ (eng. Unwrap) pojedine manje 3D objekte od kojih se sastoji ovaj model kako bi se mogle ispravno postaviti željene teksture. Kada se model „odmota“ dobivamo UV mapu. UV mapiranje predstavlja proces prikazivanja tj. reprezentacije 3D modela u 2D prostoru. S obzirom da se 3D modeli prikazuju pomoću koordinatama x,y i z, za reprezentaciju u 3D prostoru, u 2D prostoru ih nije moguće tako prikazati s obzirom da tamo imamo samo x i y koordinate tj. nema treće, z koordinate. Zbog toga ih je potrebno „odmotati“ te se prikazuju pomoću u i v koordinata s obzirom da su x i y već zauzete. Koordinata u se odnosi na horizontalnu os dok se koordinata v odnosi na vertikalnu os [29]. Jednostavnu reprezentaciju UV mapiranja na primjeru 3D objekta obične kocke možemo vidjeti na sljedećoj slici:



Slika 13: Reprezentacija UV mapiranja [30]

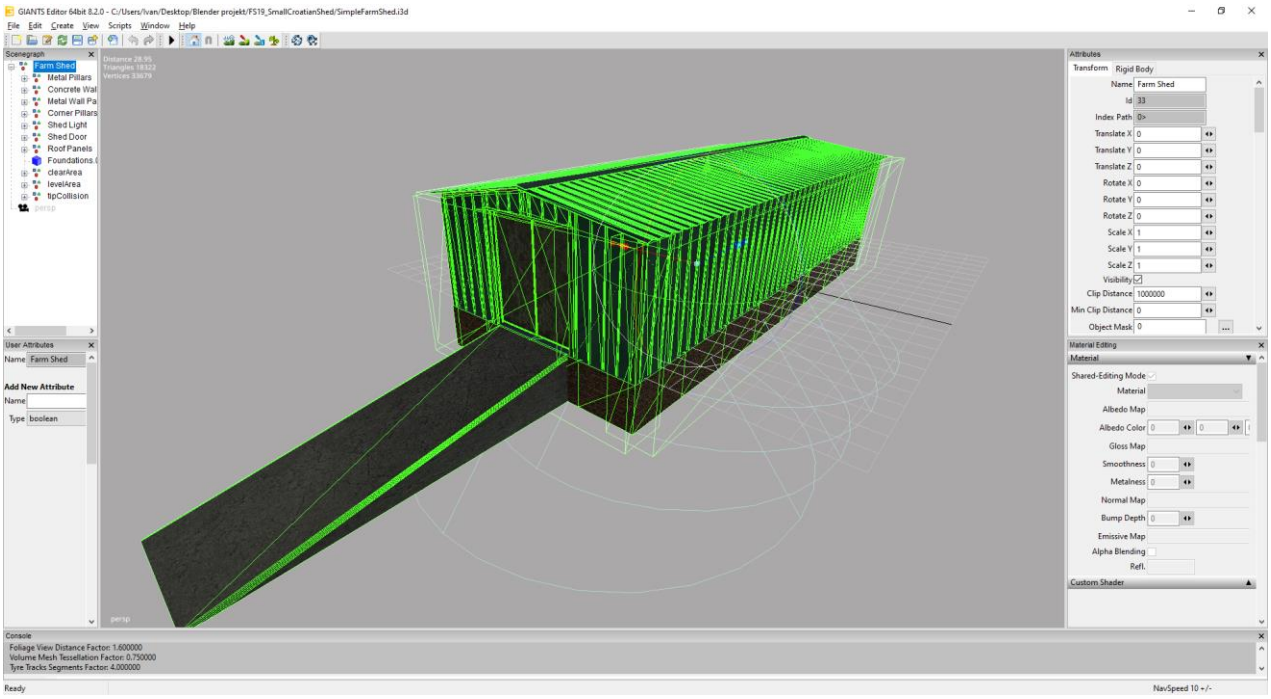
Za dodavanje tekstura smo se poslužili Internetom te smo sve teksture preuzeli besplatno sa istog izvora [31]. Teksture je moguće naknadno uređivati (bojati, potamnjivati, dodavati im efekte osvjetljenja itd.), no u našem slučaju smo bili zadovoljni sa teksturama koje smo odabrali jer su dovoljno realno reprezentirale materijale stvarnog iz stvarne okoline te smo ih primijenili na naš model čime je on dobio sljedeći izgled:



Slika 14: Model nakon dodavanja željenih tekstura

Nakon što smo završili rad na modelu, bilo ga je potrebno izvesti u odgovarajući format koji koristi Giants Editor pokretač. Već smo prije naveli kako smo instalirali potreban modul, te smo ga sad iskoristili kako bi stvorili .i3d datoteku. No kako bi kreirali modifikaciju koja će se moći koristiti u igri, potrebne su nam bile još neke datoteke pomoću kojih bih igra ispravno radila s modelom. Potrebna nam je bila mapa tekstura koje model koristi te koje smo morali naknadno učitati u pokretaču kako bi on znao primijeniti navedene teksture u samoj igri. Nadalje, bila nam je potrebna slika u formatu .dds pomoću koje bi prepoznali naš model u igri i iskoristili ga kao objekt. Također su nam bile potrebna dvije .xml datoteke naziva „modDesc“ i „SimpleFarmShed“. Prva datoteka sadrži osnovne podatke o modifikaciji pomoću koje igra radi s istom, dok druga sadrži ostale programske elemente koje omogućuju rad s animacijom i slično. Radi slabe dokumentacije vezane uz kreiranje modifikacija, preuzeli smo nekoliko sličnih već postojećih modifikacija sa službene stranice koja je već prije navedena u ovom radu kako bi vidjeli na koji način su opisane .xml datoteke postavljene za rad s igrom.

Nakon izvoza modela u .i3d format, učita li smo ga u Giants Editor-u. U mapi gdje se nalazi učitana datoteka, stvorili smo još jednu mapu engleskog naziva „textures“, u koju smo spremili sve teksture korištene u Blenderu, te smo ih u Giants Editoru ponovo učitali s nove lokacije. To je učinjeno kako bi putanja do tekstura bila relativna i navodila direktno do datoteke modifikacije, čime se izbjegavaju problemi s učitavanjem tekstura na drugim računalima koja će koristiti modifikaciju. Izgled modela u Giants Editor-u možemo vidjeti na sljedećoj slici:



Slika 15: 3D model u Giants Editor pokretaču

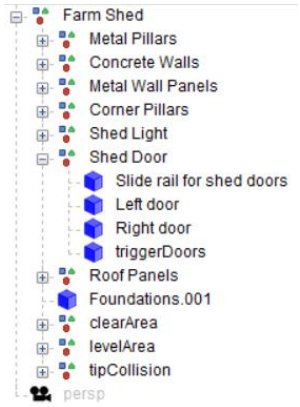
Nakon ponovnog učitavanja tekstura, spremili smo model i započeli modifikaciju .xml datoteka. Prva datoteka koju smo morali modificirati je bila „modDesc.xml“. Unutar nje smo morali definirati naziv datoteke slike moda koji je uobičajeno naziva „store.dds“. Zatim smo dodali naziv objekta koji će se koristiti u igri te opis (eng. Description). Svi elementi su postavljeni na engleskom jeziku s obzirom da je cijela igra sama po sebi na engleskom. Igra u sebi sadržava predefinirane tipove modova kao što su kamioni, prikolice itd. Još jedan tip su objekti koje igrač može slobodno postaviti (eng. Placeable). Prema tome, unutar .xml-a smo morali postaviti vrijednost atributa „rootNode“ na „placeable“ te smo isto tako morali postaviti vrijednost atributa „xmlFilename“ na vrijednost koja odgovara nazivu druge .xml datoteke. Izgled „modDesc.xml“ datoteke je moguće vidjeti na sljedećoj slici:

```
<?xml version="1.0" encoding="utf-8" standalone="no" ?>
<modDesc descVersion="40">
        <author>Ivan Bogovic</author>

    <version>1.0</version>
    <iconFilename>store.dds</iconFilename>
    <title>
        <de>Simple Croatian Farm Shed</de>
        <en>Simple Croatian Farm Shed</en>
    </title>
    <description>
        <de>Simple Croatian farm shed where you can store machinery, bales, your crops etc.</de>
        <en>Simple Croatian farm shed where you can store machinery, bales, your crops etc.</en>
    </description>
    <multiplayer supported="true"/>
    <storeItems>
        <storeItem rootNode="placeable" xmlFilename="SimpleFarmShed.xml"/>
    </storeItems>
</modDesc>
```

Slika 16: XML datoteka „modDesc“

Sad je na red došla druga .xml datoteka postavljenog naziva „SimpleFarmShed“. No prije nego što nju objasnimo, vratit ćemo se na model izrađen u Blender-u. Naime, nakon što smo kreirali više manjih 3D objekata i postavili kako smo željeli, označili smo ih sve redom i spojili u jedan objekt kako bi mogli lakše raditi s njime u Giants Editoru. Sve manje objekte smo spojili osim dva. Ta dva su objekta su predstavljala klizna vrata tj. lijevo i desno krilo vrata. Naime, htjeli smo kreirati animaciju otvaranja i zatvaranja vrata. Unutar Blendera je moguće kreirati animacije, ali ih Giants službeno ne podržava. Podržavaju se samo animacije kreirane u Mayi. Kao alternativno rješenje, moguće je kreirati animacije pomoću navedene „SimpleFarmShed“ xml datoteke. Princip rada je jednostavan te se odnosi na pomicanje objekata po x,y i z koordinatama u određenom definiranom vremenu. No kako bi se objekt kojeg želimo animirati mogao pomicati po navedenim koordinatama, moramo ga ostaviti odvojenog od ostalih objekata te su s obzirom na to, lijevo i desno krilo kliznih vrata odvojeni objekti od ostatka. Unutar Giants Editora smo dodali nevidljivi objekt koji je predstavljao okidač (eng. Trigger) pomoću kojeg je igrač mogao pokrenuti željenu animaciju. Također, pomicali smo objekte vrata po željenim koordinatama te smo zapisivali vrijednosti koordinata kako bi ih mogli prenijeti u .xml datoteku i samim time stvoriti animaciju pokreta. U našem slučaju, kod oba objekta se je mijenjala samo vrijednost x koordinate. Navedene vrijednosti su nam bitne kako bi mogli postaviti vrijednosti atributa „keyFrame“ u .xml datoteci. Općenito, „key frame“ u animaciji predstavlja početnu ili završnu točku tranzicije. Ako postavimo više točaka, animacija će se izvoditi između tih točaka s obzirom na definirano vrijeme [32]. Nadalje, kako bi mogli povezati objekte vrata i okidača s potrebnim animacijama i funkcijama, potrebne su nam njihove pozicije u hijerarhiji elemenata unutar Giants Editora:



Slika 17: Struktura elemenata unutar Giants Editora

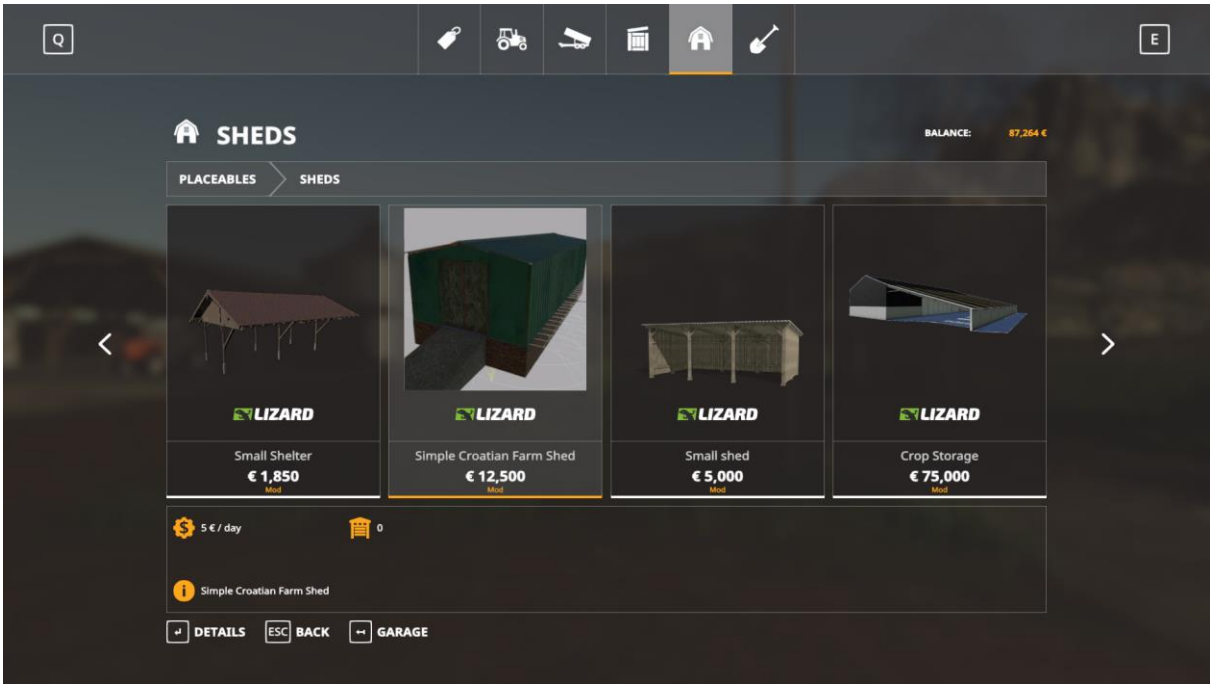
Ako pogledamo prijašnju sliku i traži grupu naziva „Shed Door“, možemo vidjeti da se ona nalazi šesta po redu ali s obzirom da se u ovom slučaju brojanje počinje od nule, njen indeks je pet. Također, unutar grupe brojanje ponovo počinje od početka te vidimo da objekt „Left door“ ima indeks jedan, „Right door“ indeks dva dok okidač „triggerDoors“ ima indeks čija je vrijednost tri. Na temelju tih vrijednosti povezujemo elemente u .xml datoteci i radimo sa njima. Sada se vraćamo na „SimpleFarmShed“ datoteku. Unutar nje postavljamo željene vrijednosti već definiranih xml elemenata kao što je cijena (npr. 12500), mogućnost prodaje, kategorija pod kojom će se pojavljivati u igri, dnevni trošak održavanja itd. Kako bi postavili animaciju otvaranja vrata, bitan nam je bio element naziva „animatedObjects“, unutar kojeg smo postavljali nove elemente „animatedObject“. Unutar njih smo definirali sve potrebne elemente i atribute za animiranje kao što je prikazano na sljedećoj slici:

```
<animatedObjects>
    <animatedObject saveId="Doors">
        <animation duration="1.7">
            <part node="5|1">
                <keyFrame time="0.0" translation="1.051 2.41 -12.21"/>
                <keyFrame time="0.5" translation="1.511 2.41 -12.21"/>
                <keyFrame time="1.0" translation="3.022 2.41 -12.21"/>
            </part>
            <part node="5|2">
                <keyFrame time="0.0" translation="-1.051 2.41 -12.21"/>
                <keyFrame time="0.5" translation="-1.511 2.41 -12.21"/>
                <keyFrame time="1.0" translation="-3.022 2.41 -12.21"/>
            </part>
        </animation>
        <controls triggerNode="5|3" posAction="ACTIVATE_HANDTOOL" posText="action_openGate" negText="action_closeGate"/>
        <sounds>
            <moving file="$data/sounds/prefab/gate/gate_loop.wav" loops="0" linkNode="5|3" volume="0.6" radius="25" innerRadius="3" fadeOut="0.25"/>
            <posEnd file="$data/sounds/prefab/gate/gate_opened.wav" linkNode="5|3" volume="0.6" radius="25" innerRadius="3" />
            <negEnd file="$data/sounds/prefab/gate/gate_closed.wav" linkNode="5|3" volume="0.6" radius="25" innerRadius="3" />
        </sounds>
    </animatedObject>
```

Slika 18: Elementi potrebni za animiranje i zvuk

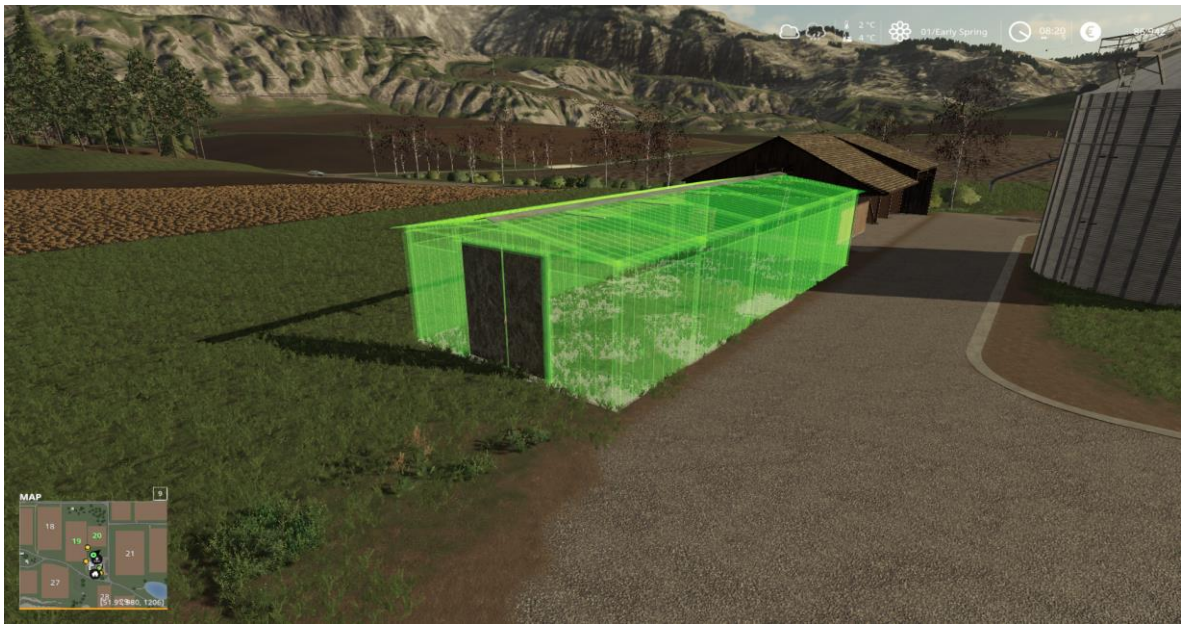
Na prijašnjoj slici je moguće vidjeti elemente „keyFrame“ s atributima „time“ koje smo već prije spominjali. Odredili smo da će svaki objekt vrata imati tri „keyFrame“ elementa zajedno za ukupnim trajanjem animacije od 1.7 sekundi. Kako bi igrač mogao ostvariti interakciju s vratima, potrebno je bilo i dodati „control“ element koji će kontrolirati sve animacije pod željenim „animatedObject“ elementom. Unutar elementa smo ponovo pomoću odgovarajućih indeksa došli do objekta tj. okidača koji će biti zaslužan za interakciju. Radi ostvarivanja boljeg i realističnijeg efekta, pomoću elementa „sounds“ dodali smo i zvučne efekte otvaranja vrata. Efekte smo dodali okidaču te smo koristili zvukove koji već dolaze uz igru kako bi smanjili veličinu modifikacije. Samim time smo završili rad na našoj modifikaciji te ju se sad trebalo ubaciti u igru. Igra prilikom svog učitavanja, učitava sve modifikacije iz datoteke „mods“ kako bi ih korisnik mogao koristiti te smo tamo trebali prebaciti datoteku naše modifikacije. Putanja do datoteke može varirati ovisno o operacijskom sustavu na kojem se nalazi igra. Za testiranje je korišten sustav Windows 10, gdje se navedena datoteka nalazi na

lokaciji datoteke „My Games“ unutar datoteke „Documents“. Kada smo prebacili modifikaciju i pokrenuli igru, pronašli smo naš objekt na odgovarajućem mjestu kao što je prikazano na slici:



Slika 19: Objekt naziva „Simple Croatian Farm Shed“ unutar igre Farming Simulator 19

Nakon odabira objekta u igri, igrač ima mogućnost njegovog postavljanja na željenu lokaciju. Testirali smo animaciju otvaranja vrata te je objekt bi spreman za korištenje kao u potpunosti funkcionalan element igre razvijen u od strane korisnika u Blenderu. Primjer korištenja objekta u igri je moguće vidjeti na sljedećim slikama:



Slika 20: Postavljanje objekta u igri na željenu lokaciju od strane igrača

Slika 21: Mogućnost otvaranja kliznih vrata od strane igrača

3. Izrada 2D igre za PC

U ovom dijelu rada će biti prikazan način izrade 2D platformerske igre. Igra će biti izrađena za prijenosne PC uređaj u Unity razvojnom okruženju. Za izradu skripti će se koristiti C# programski jezik te Visual Studio Code kao razvojno okruženje za pisanje i rad s kodom. Igra će biti jednostavnog platformerskog dizajna, te će se u izradi koristiti besplatni grafički materijali pronađeni na Internetu kako bi se ostvario vizualni dizajn bez potrebe za korištenjem dodatnih programskih alata.

3.1. Priprema okruženja za rad

Prije početka rada na razvoju igre, bilo je potrebno postaviti u potpunosti razvojna okruženja te testirati sve bitne komponente za rad, kako bi se izbjegli budući problemi. Kao što je već prije navedeno, razvoj programskog koda je proveden pomoću Visual Studio Code razvojnog okruženja verzije 1.43.2 [33]. S obzirom da se radi o C# programskom jeziku, moguće je bilo i korištenje Visual Studio razvojnog okruženja, no ono kao takvo zauzima više prostora na računalu a ne postoji određen razlog za njegovim korištenjem. Nakon instalacije, bilo je potrebno skinuti i instalirati zadnju verziju Unity razvojnog okruženja. Zbog stabilnosti rada, izbjegnuta je probna (Beta) verzija te je odabrana zadnja stabilna verzija (2019.3.7f1) [34].

Nakon odabira verzije sustava, došlo je pitanje odabira grafičkih elemenata koji će biti korišteni u igri. Pretraživanjem po Internetu, pronađene su određene stranice koje besplatno nude 2D elemente za razvoj jednostavnih igara. Korišteni elementi će biti prikazani kasnije u radu.

S obzirom da se plan razvoja odnosi na jednostavnu platformersku igru, proučeno je par primjera već postojećih igara te su određena pravila naše igre. Igra neće imati definiran kraj već će kretanje kroz razinu biti beskonačno. Samo kretanje igrača će biti nagrađivano bodovima, ali isto tako će se na razini nalaziti dodatni bodovi koje će igrač moći sakupiti. S obzirom da će kretanje biti beskonačno, potrebno je generirati razine nasumično. Nadalje, tijekom razvoja igre je još donesena odluka o postupnom ubrzanju kretanja igrača kako bi stvari postale teže i predstavljale veći izazov. Također su određeni zvučni efekti koje će igra proizvoditi prilikom igranja. Besplatne kvalitetne zvučne efekte je bilo nešto teže pronaći na Internetu te s obzirom da je njihov opseg bio manji, određena su četiri zvučna efekta koja će igra koristiti: pozadinski zvuk, zvuk skakanja, zvuk sakupljanja bodova te zvuk smrti igrača. Prilikom razvoja je tražen i zvuk kretanja igrača kroz razinu (zvuk trčanja) ali nije pronađena

kvalitetna verzija efekta koji bi se uklapao u igru te je zvučni efekt trčanja izbačen iz igre. Na kraju je bilo potrebno odabrati naziv igre, te je odlučeno da će se igra jednostavno zvati Platformer.

Prije početka razvoja igre je odlučeno da će igra biti na hrvatskom jeziku ali će nazivi određenih datoteka unutar razvojnog okruženja biti ostavljeni na engleskom jeziku radi jednostavnosti snalaženja te traženja pomoći na Internetu. Na primjer, prilikom kreiranja projekta stvorena je datoteka naziva „Assets“ koja sadrži sve potrebne materijale za rad te ona neće biti preimenovana na hrvatski jezik. Ali sve daljnje vlastito kreirane datoteke za rad će sadržavati hrvatski naziv. Isto tako je odlučeno da će svi elementi programskog koda (ili barem većina) koji nisu automatski kreirani sadržavati hrvatske nazive radi jednostavnosti razumijevanja prilikom čitanja ovog rada.

3.2. Prikaz igrača i postavljanje osnovnih mogućnosti

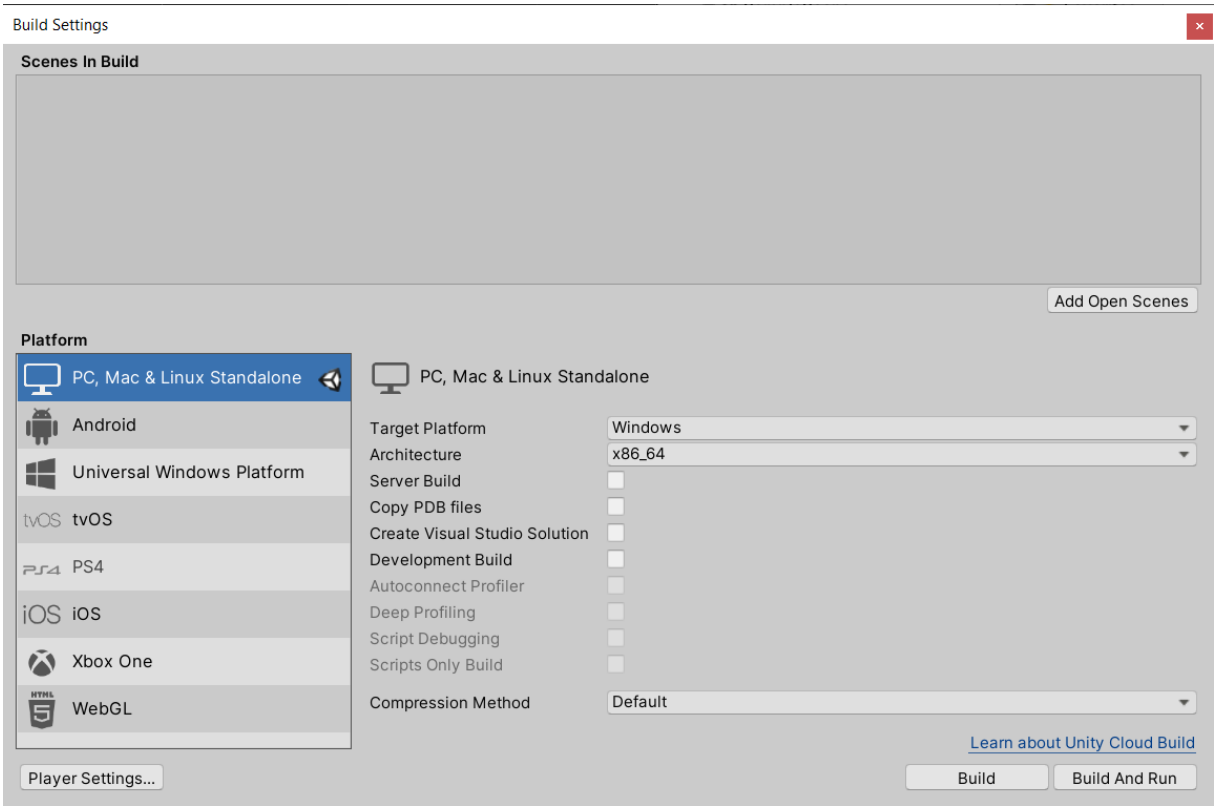
Prva komponenta igre od koje je krenuo razvoj je bio sami prikaz igrača u igri. Odlučeno je da će igra biti opuštenijeg dizajna te prilikom pretrage po Internetu pronađen skup slika za 2D igrača u obliku dinosaura [35]. Skup slika se je sastojao od više stanja igrača: stajanje, hodanje, trčanje, skakanje, smrt. Svako stanje se je sastojalo od više sličica koje su predstavljale cijeli pokret od početka do kraja. Na taj način je omogućeno stvaranje animacije u razvojnim okruženjima. Prilikom rada na projektu je uklonjeno stanje smrti igrača s obzirom da ono nije vidljivo na ekranu. Nadalje, ostavljena je samo jedna sličica skakanja, jer skakanje predstavlja kratku radnju koju nije potrebno animirati u našoj igri. Isto tako je odlučeno da nema potrebe za stanjem mirovanja i stanjem hodanja igrača, te su i te sličice također uklonjene iz datoteke projekta. Primjer jedne sličice stanja trčanja dinosaura je moguće vidjeti na sljedećoj slici:

Slika 22: Primjer stanja trčanja igrača (dinosaura)

Unity koristi mrežni prikaz scene (eng.Grid) za jednostavniji rad tj. jednostavnije postavljanje veličine objekata unutar igre. Mreža se sastoji od više kvadrata od kojih je svaki

veličine 256x256 piksela. Prema tome, radi jednostavnijeg rada je odlučeno da će svi pojedini grafički elementi tj. sve sličice koje će biti korištene u igri biti iste veličine kao i navedeni kvadrati. Prema tome, svim uvezenim slikama je bilo potrebno postaviti postavku „Pixels per unit“ na vrijednost 256. Nakon što smo postavili zadanu vrijednost, odabrali smo jednu sličicu dinosaura te ju postavili na sredinu zaslona kako bi mogli testirati pokretanje.

Da bismo igru prenijeli na računalu, bilo je potrebno otići pod „Build Settings“ opciju, odabrati „PC, Mac & Linux Standalone“ i kliknuti na opciju „Switch Platform“ kako bi promijenili platformu razvoja prije prijenosa igre. Zatim je bilo potrebno odabrati opciju „Build and Run“ kao što je prikazano na sljedećoj slici:



Slika 23: Postavke za izradu i pokretanje projekta na računalu

Nakon prijenosa projekta, igra se automatski pokreće te je na sredini zaslona bilo moguće vidjeti postavljenu sličicu igrača. Samim time je potvrđeno da se projekt uspješno kompajlira te da je moguće nastaviti daljnji razvoj.

Nakon što smo u naš projekt dodali slike za vizualni prikaz, potrebno je postaviti animacije za pokrete igrača. Igrač predstavlja objekt te se kao takav može koristiti u programskom kodu tj. prilikom izrade potrebnih skripti. Za jednostavnije pronalaženje, preimenovali smo objekt igrača u „Dinosaur“. Svakom objektu u Unity-u je moguće dodavati komponente koje će definirati svojstva pojedinih objekata. Kada odaberemo neki objekt, u prozoru naziva „Inspector“ možemo vidjeti njegova svojstva. Primjer jednog svojstva je

„Transform“ grupa koja definira poziciju, rotaciju i veličinu objekta pomoću x,y i z koordinata. Kako bi objekt bio pod utjecajem fizike, potrebno je dodati komponentu naziva „RigidBody 2D“. Navedena komponenta simulira gravitaciju, ubrzanje i ostale slične elemente [36]. Nakon dodavanje navedene komponente, prilikom pokretanja igre je bilo moguće vidjeti da igrač pada.

Za privremeno testiranje ponašanja igrača, dodali smo objekt 3D kocke te ga produžili u privremenu platformu koju ćemo kasnije obrisati i zamijeniti s vlastitim platformama koje će naša igra koristiti. S obzirom da smo dodali 3D kocku a radimo u 2D prostoru, bilo je potrebno ukloniti komponentu naziva „Box Collider“ s obzirom da je ona namijenjena za 3D prostor. Nakon uklanjanja komponente, bilo je potrebno dodati komponentu naziva „Box Collider 2D“ koja se odnosi na rad u 2D prostoru. Navedene komponente omogućuju stvaranje kolizije objekata čime drugi projekti neće prolaziti kroz njih već će se „sudarati“ s njima u prostoru. Kako igrač prilikom pada ne bi prolazio kroz dodanu platformu, i njemu isto treba dodati „Box Collider 2D“ komponentu. Samim time prilikom pokretanja igre će igrač pasti na platformu jer obje komponente, igrač tj. dinosaur i platforma, imaju definirano svojstvo kolizije.

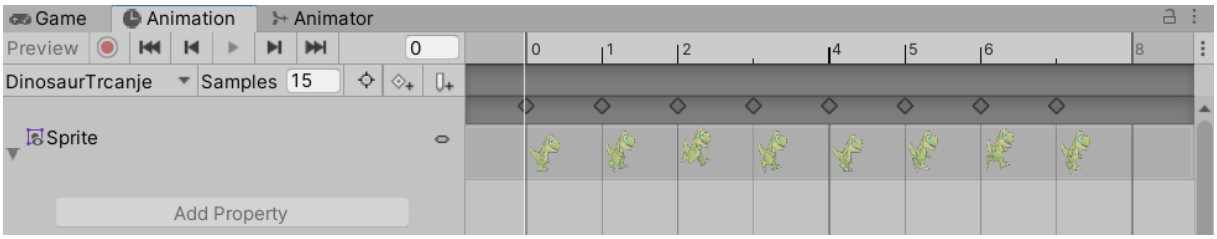
Kako bi se igrač mogao kretati, potrebno je dodati programsku skriptu koja će to omogućiti. Kako bi struktura projekta ostala čista, stvorena je datoteka naziva „Skripte“ u koje će biti dodane sve skripte za rad. Unutar datoteke se jednostavnom opcijom u izborniku može stvoriti nova C# skripta naziva „DinosaurKretanje“. Nakon otvaranja skripte pomoću Visual Studio Code razvojnog okruženja, bilo je moguće vidjeti dvije već kreirane metode naziva „Start()“ i „Update()“. Prva metoda se pokreće samo jednom prilikom pokretanja igre. Unutar nje je potrebno vršiti sve potrebne inicijalizacije. Druga metoda se pokreće jednom svaku sličicu (eng. Frame).

Kako bi mogli raditi sa „Rigid Body2D“ komponentom tj. kako bi pomoću nje omogućili kretanje igrača, potrebno ju je definirati kao privatnu varijablu izvan funkcija. Nakon toga ju je potrebno inicijalizirati, a to se vrši unutar „Start()“ metode kao što je već prije bilo opisano. Inicijalizacija se vrši pomoću poziva metode „GetComponent<RigidBody2D>()“. Kako bi postavili brzinu kretanja igrača, postavljamo javnu varijablu tipa „float“ naziva „brzinaDinosaura“. Unutar metode „Update()“ postavljamo vrijednost brzine (eng. Velocity) prve varijable koju smo kreirali tipa „RigidBody2D“. Postavljanje se vrši pomoću inicijalizacije „Vector2()“ objekta [37] koji omogućuje definiranje brzine kretanja objekta u 2D prostoru. Za definiranje brzine kretanja, potrebno je proslijediti x i y koordinate kao parametre. S obzirom da se u našoj igri igrač kreće s lijeva na desno tj. samo po x osi, kao parametar za x os prosljeđujemo varijablu „brzina dinosaura“, a za y parametar postavljamo vrijednost naše „RigidBody2D“ varijable kao što je prikazano na slici:

```
rigidBody.velocity = new Vector2(brzinaDinosaura,rigidBody.velocity.y);
```

Slika 24: Postavljanje parametara Vector2 objekta

Nakon postavljanja početnog koda, bilo je potrebno dodati skriptu objektu igrača. Skripte također predstavljaju komponente objekata te se dodaju na isti način kao i sve druge komponente. Prvo je potrebno odabrati objekt igrača, nakon toga u „Inspector“ prozoru odabrati opciju „Add Component“ te odabrati željenu skriptu. Nakon dodavanja skripte, u istom prozoru je moguće vidjeti polje za unos brzine kretanja dinosaura koju smo definirali u programskom kodu. Nakon isprobavanja određenih vrijednosti brzine, odlučeno je da će biti unesena brzina s vrijednosti 10 kao idealna početna brzina kretanja. No kada pokrenemo igru, sličica igrača će pokreće, no nema nikakvih animacija kretanja, iako smo skinuli skup slika dinosaura koje predstavljaju kretanje. Kako bismo dodali traženu animaciju, potrebno je otvoriti novi prozor naziva „Animation“ i odabrati objekt igrača. Nakon toga je potrebno stvoriti novu animaciju. Prilikom stvaranja animacije napravljena je nova datoteka naziva „Animacije“ kako bi struktura projekta ostala čista. Animaciju smo nazvali „DinosaurTrcanje“. Kako bismo stvorili iluziju kretanja igrača tj. animaciju, trebamo otvoriti datoteku „Slike“ i odabrati sve slike kretanja dinosaura u pravilnom redoslijedu. Nakon toga ih jednostavno dodamo u prostor za animaciju. Sličice kretanja će se automatski dodati jedna za drugom, te je na nama da samo postavimo brzinu animacije pomoću postavke „Sample“ kao što je prikazano na sljedećoj slici:



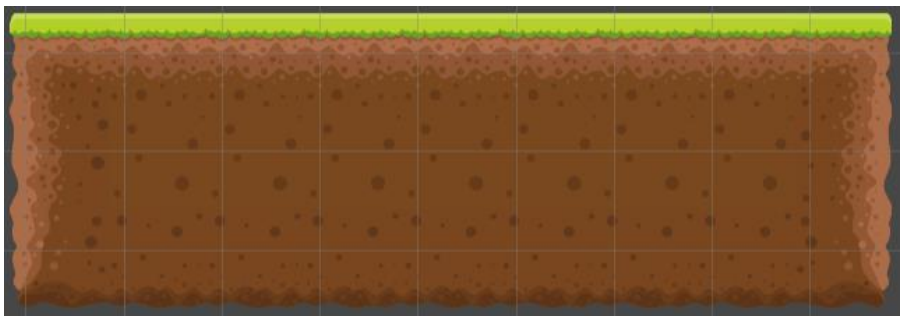
Slika 25: Animacija kretanja igrača

3.3. Postavljanje platformi za kretanje

Prije nego što smo postavili grafički prikaz platformi, bilo je potrebno pronaći odgovarajuće slike za platforme. S obzirom da smo veličine platformi trebali sami postavljati, bilo je potrebno pronaći skup od više slika koje spojene čine jednu platformu. Nakon kraće potrage po Internetu, pronađen je jedan takav skup 2D slika koje su odgovarale temi naše igre [38]. Nakon pronalaska slika, kreirali smo zasebnu datoteku naziva „Platforma“, te smo ubacili slike platforme unutar nje kako bi i dalje zadržali čistoću projekta. Nadalje, svakoj slici posebno smo promijenili postavku „Pixels per Unit“ na 256 kako bi slike odgovarale veličini ostalih elemenata u igri. Za našu igru nam nisu trebale sve slike platforme s obzirom da je preuzeti skup slika omogućavao stvaranje različitih oblika platformi a nama su trebale samo ravne

platforme različitih duljina. Nakon dodavanja slika u datoteku, sve slike smo prebacili u radni prozor „Scene“.

Nakon što smo pripremili slike za sastavljanje platformi, pojavilo se pitanje poravnanja slika u sceni kako bi se stvorile ravne platforme. Jednostavno rješenje za taj problem je bilo korištenje komponenti slika tj. komponente naziva „Transform“, koja je već prije objašnjena. Označili smo sve slike i postavili njihove x i y koordinate na nule kako bi se sve slike zajedno premjestile na sredinu ekrana. Samim time smo omogućili pomicanje slika po x koordinatama kako bi se stvorile savršeno ravne platforme. Prilikom prolaska kroz dokumentaciju Unitiy razvojnog okruženja, uočen je skup prečaca na tipkovnici koje možemo koristiti za jednostavnije pomicanje elemenata [39]. Držanjem tipke Ctrl na tipkovnici prilikom pomicanja elemenata, elementi se pomiču za veličinu jednog bloka na mreži. Samim time je olakšano sastavljanje platformi. Za sada se je samo trebalo poigrati s elementima, kako bi se kreirale platforme. Od odgovarajućih slika su sastavljeni lijevi i desni rubovi platforme te sama sredina. Nakon toga, sve što je bilo potrebno je pomaknuti rubove na željenu udaljenost, sredinu platforme duplicirati (Ctrl + D) te spajati dijelove. Odlučeno je kako će igra sadržavati tri platforme: kratku, srednju i dugačku platformu. Izgled jedne od platformi možemo vidjeti na sljedećoj slici:



Slika 26: Izgled kratke platforme

Prilikom kreiranja platformi, bilo je potrebno svim elementima koje čine pojedinu platformu dodati komponentu „Box Colider 2“, koja omogućuje simuliranje kolizije između objekata. Isto kao što smo i prije imali privremenu platformu za testiranje, tako i ovdje trebamo dodati navedenu komponentu, da objekt igrača ne bi propadao kroz platforme. Prije dodavanja komponente kolizije, stvorili smo nove prazne objekte (eng. Empty object) te smo im pridružili sljedeće nazive: „KratkaPlatforma“, „SrednjaPlatforma“ i „DugackaPlatforma“. Unutar tih objekata smo dodali sve objekte od kojih se sastoje pojedine platforme radi bolje organizacije rada. Na početku igre smo ostavili dvije platforme odakle će igrač uvijek kretati prilikom nove igre, dok će ostali raspored platformi biti nasumično generiran. Sada smo svakoj platformi trebali dodati komponentu kolizije. Ovaj put se naše platforme sastoje od više elemenata te je

zbog toga bilo potrebno ručno postavili veličinu kolizije za pojedinu platformu pomoću promjena x i y koordinata, kako bi kolizija obuhvatila cijelu veličinu pojedine platforme.

Kako bi mogli započeti rad na nasumičnom generiranju platformi, stvorili smo novu datoteku naziva „Prefabs“. „Prefab“ komponente predstavljaju ponovo iskoristive elemente prilikom razvoja igara tj. elemente koji služe kao osnova za daljnju gradnju i razvoj igre [40]. Prednost takvog načina ponovnog iskorištavanja već postojećih elemenata je u tome što se promjene na roditeljskom elementu reflektiraju na sve instance tog elementa. Time se znatno olakšava rad, pogotovo na većim projektima. Također, mogu se stvarati kompleksne hijerarhije ovakvih elementa tj. moguće je ugnježđivanje „prefab“ elemenata. S obzirom da će naša igra sama automatski generirati nasumično nove platforme, želimo koristiti naše tri veličine platformi kao „prefab“ elemente te smo zbog toga premjestili sva tri elementa u novo kreiranu „Prefabs“ datoteku.

3.4. Povezivanje kamere sa igračem te generiranje platformi

Nakon što smo postavili sve elemente, testirali smo igru kako bi utvrdili da li se igra normalno ponaša. Sve je bilo u redu, ali je uočen jedan problem. Prilikom kreiranja novog projekta za 2D igru, automatski se kreira objekt kamere naziva „Main Camera“. Navedeni objekt omogućuje jednostavno postavljanje scene koju će igrač vidjeti tijekom igranja igre. S obzirom na to da se naša igra odnosi na kretanje igrača kroz razine, bilo je potrebno zajedno za objektom igrača (dinosaura) pomicati i objekt kamere jer bi inače objekt dinosaura vrlo brzo nakon pokretanja igre izašao izvan scene te ga igrač više ne bi mogao vidjeti. Zbog toga je bilo potrebno programski povezati ta dva objekta kako bi objekt dinosaura cijelo vrijeme bio vidljiv na zaslonu ekrana. Još jedan od problema koji su uočeni je bio povezan s rotacijom igrača prilikom pada sa platfome tj. prelaska s više platforme na nižu. Iako je igra i dalje radila normalno, ipak nismo htjeli da se objekt dinosaura rotira te smo trebali to ograničiti. S obzirom da se je igrač kretao po x osi tj. s lijeva na desno, prilikom pada sa platforme se je rotirao po osi z. Problem je riješen ponovo korištenjem komponenti objekata te nije bilo potrebe za programskim skriptama. Pod komponentom „Rigidbody 2D“ je bilo moguće postaviti ograničenja (eng. Constraints). Ograničenje koje smo postavili se zove „Freeze Rotation Z“ te je samim time problem popravljen.

Sada se je trebalo vratiti na problem izlaska objekta igrača izvan okvira kamere. Kako bi povezali objekt igrača i objekt kamere, stvorili smo novu C# skriptu unutar datoteke „Skripte“ te smo ju nazvali „KameraKontroler“ kako bi lakše u buduće razaznali samu ulogu skripte. Kao i sa svakom drugom C# skriptom u Unity-u, i ovdje smo dobili automatski generirane metode „Start()“ i „Update()“. Kako bi mogli povezati kameru sa objektom igrača, bilo je potrebno

dohvatiti objekt. Vratili smo se u Unity okruženje, odabrali objekt igrača te smo mu dodali oznaku (eng. Tag) vrijednosti „dinosaurTag“ kako bi lakše dohvatili objekt u kodu. Kasnije tijekom izrade projekta je uočeno da ne postoji potreba za dodavanjem oznake tj. da se objekt igrača može dohvatiti direktnim pozivom skripte za koji je povezan naziva „DinosaurKretanje“ koju smo već prije kreirali. Za to će se koristiti metoda „FindObjectOfType<>()“ Vratili smo se nazad u programski kod te kreirali privatnu varijablu tipa „GameObject“ naziva „dinosaur“. Nadalje, kako bi dohvatili zadnju poziciju igrača, kreirali smo varijablu tipa „Vector3“ naziva „zadnjaPozicija“. Prije u ovom radu je već opisan tip podatka „Vector2“ koji sadrži x i y koordinate određenog objekta, no u ovom slučaju su nam potrebne sve tri koordinate tj. x,y i z te zbog toga koristimo Vector3 [41]. Također smo definirali privatnu varijablu tipa „float“ pomoću koje ćemo određivati udaljenost kamere. Jednom kada smo definirali varijable, u metodi „Star()“ smo dohvatili objekt igrača korištenjem metode „FindGameObjectWithTag("dinosaurTag")“ kao što je prikazano na sljedećoj slici:

```
dinosaur = GameObject.FindGameObjectWithTag("dinosaurTag");
```

Slika 27: Dohvaćanje objekta dinosaura (igrača)

Također, prilikom pokretanja igre je odmah bilo potrebno dohvatiti zadnju poziciju objekta igrača pomoću sljedeće linije koda:

```
zadnjaPozicija = dinosaur.transform.position;
```

Slika 28: Dohvaćanje pozicije igrača

Navedena linija koda dohvaća x,y i z koordinate iz komponente „Transform“ objekta igrača. Nakon dohvaćanja svih potrebnih elemenata prilikom pokretanja igre, unutar metode „Update()“ je prvo trebali izračunati udaljenost kamere od objekta igrača kako bi onda tu vrijednost mogli koristiti prilikom kretanja kamere tj. kako bi omogućili praćenje igrača. S obzirom da se naš igrač kreće s lijeva na desno tj. samo po x osi, za izračun udaljenosti kamere od igrača je i potrebno koristiti samo vrijednosti x osi. S obzirom da se „Update()“ metoda izvršava svaku sličicu tj. da predstavlja petlju (eng. Loop), trebamo uzeti zadnju x poziciju koju smo na početku postavili u „Start()“ metodi i oduzeti je od trenutne pozicije igrača. Navedenu udaljenost računamo pomoću sljedeće linije koda:

```
udaljenost = dinosaur.transform.position.x - zadnjaPozicija.x;
```

Slika 29: Izračunavanje udaljenosti kamere od objekta igrača

Nakon izračunavanja udaljenosti kamere, potrebno je pomoću tog izračuna ažurirati poziciju svaku pojedinu sličicu. Ažuriranje pozicije se vrši stvaranjem novog Vector3 objekta te dohvaćanjem pozicije kamere pomoću komponente „Transform“. No kako bi se kamera pomicala zajedno sa igračem, zadnjoj vrijednosti x pozicije pridružujemo izračunatu vrijednost

udaljenosti koja je spremljena u varijabli „zadnjaPozicija“. Na taj način kamera mijenja svoju poziciju po x osi svaku sličicu zajedno za objektom igrača. Ali da bi svaki sljedeći izračun bio točan, ponovo je potrebno postaviti vrijednost varijable „zadnjaPozicija“ na kraju „Update()“ metode tako što njoj sada pridružujemo trenutnu poziciju objekta igrača. Cijeli kod za promjenu pozicije kamere pomoću „Update()“ metode je moguće vidjeti na sljedećoj slici:

```
void Update()
{
    udaljenost = dinosaur.transform.position.x - zadnjaPozicija.x;
    transform.position = new Vector3(
        transform.position.x + udaljenost,
        transform.position.y,
        transform.position.z
        );
    zadnjaPozicija = dinosaur.transform.position;
}
```

Slika 30: Kod metode „Update()“ za ažuriranje pozicije kamere

Nakon postavljanja programskog koda za pomicanje kamere, vratili smo se u Unity okruženje, odabrali objekt kamere te dodali skriptu kao komponentu kamere. Sada kad je problem s kamerom riješen, možemo se vratiti na rad s platformama koji je prije opisan već.

Ovaj put, rad s platformama se odnosi na automatsko nasumično generiranje platformi. Generiranje smo isto odradili pomoću programskog koda tj. C# skripte. S obzirom na to da nije preporučljivo instancirati cijelo vrijeme nove objekte platformi, potrebno je deaktivirati objekte koje se trenutno ne koriste. Također, da bi radi optimizacije izbjegli stalno instanciranje novih objekata, deaktivirane objekte možemo ponovo iskoristiti tj. aktivirati ih kako bi ponovo bili upotrebljivi. Koncept iskorištavanja već postojećih objekata se naziva „Pooling“ [42]. Kako bi omogućili navedene operacije, stvorit ćemo novu C# skriptu engleskog naziva „ObjectPooler“ kako bi bilo razumljivija uloga skripte. Skripta će se općenito koristiti za objekte koji se trebaju ponovo iskorištavati te neće biti namijenjena samo isključivo platformama. Također će ju koristiti i objekti bodova, ali to će tek kasnije biti opisano u ovom radu. S obzirom da će skripta biti korištena za različite objekte, potrebno je koristiti poopćene nazive. Kreirat ćemo varijablu tipa „GameObject“ naziva „pooledObject“. Zadržan je engleski naziv zbog nemogućnosti pronalaska prikladnog naziva na hrvatskom jeziku, te radi jednostavnijeg razumijevanja uloge objekta. Sada trebamo definirati varijablu pomoću koje će određivati broj objekata koji se trebaju ponovo iskorištavati. Varijabla će biti tipa „int“ naziva „brojObjekata“. Isto tako je potrebno definirali listu objekata tipa „GameObject“ naziva „listaObjekata“ koja će sadržavati instance objekata koji će se ponavljati. Unutar „Start()“ metode trebamo inicijalizirati listu. Nakon toga trebamo inicijalizirati sve objekte s obzirom na postavljeni broj objekata te ih dodati u kreiranu listu kao što je prikazano na sljedećoj slici:

```
for(int i = 0; i < brojObjekata; i++){
    GameObject objekt = Instantiate(pooledObject);
    objekt.SetActive(false);
    listaObjekata.Add(objekt);
}
```

Slika 31: Inicijalizacija i dodavanje objekata u listu

Kako bi se inicijalizirani objekti pojavili na ekranu tj. na sceni, moramo koristiti metodu „Instantiate()“. Navedena metoda klonira objekt i vraća njegov klon [43]. Također, ne želimo aktivirati odmah objekt pa moramo koristiti metodu „SetActive()“ kako bi to promijenili. Na kraju dodajemo svaki pojedini objekt u listu objekata.

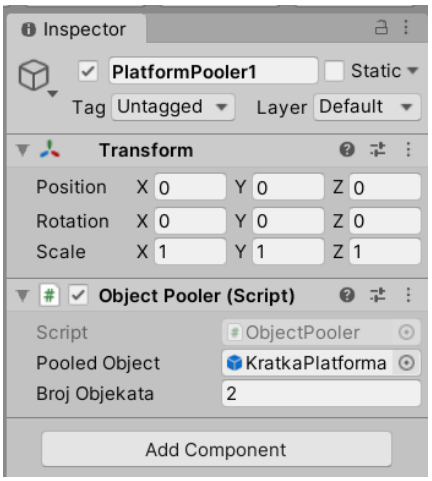
Kako bismo mogli dohvatiti kreirane objekte, trebamo napraviti funkciju koja će se pozivati kada je potrebno prikazati novi objekt. Za to trebamo pomoću petlje prolaziti kroz listu i vraćati objekte koji nisu aktivni kao što je prikazano na slici:

```
public GameObject dohvatiObjekt(){
    foreach(GameObject obj in listaObjekata){
        if(!obj.activeInHierarchy){
            return obj;
        }
    }
```

Slika 32: Dohvaćanje neaktivnih objekata

No prilikom testiranja, primijećeno je da na početku izvršavanja koda, vrlo brzo svi objekti postanu aktivni te više ne postoji neaktivni objekt kojeg možemo dohvatiti iz liste. Da bi se taj problem ispravio, bilo je potrebno stvoriti novi objekt kada se to dogodi i dodati ga u listu na isti način kao što je već prije prikazano. S time smo završili rad na trenutnoj skripti.

Unutar Unity okruženja smo sada dodali novi prazni objekt naziva „Platforme“ u koji smo premjestili dvije platforme s početka igre te u koju ćemo spremati ostatak novih objekata s kojima ćemo raditi. Sada želimo iskoristiti novo kreiranu skriptu za naše platforme. S obzirom na to da je skripta rađena za sve objekte u igri tj. pošto je poopćena i ne odnosi se isključivo na objekte platforme, unutar objekta naziva „Platforme“ smo kreirali novi prazni objekt naziva „PlatformPoolers“. Unutar tog objekta smo dodali prazni objekt naziva „PlatformPooler“ te ga duplicirali još dva puta kako bi na kraju imali tri objekta: „PlatformPooler1“, „PlatformPooler2“, „PlatformPooler3“. Svakom od ta tri objekta smo pridružili kreiranu skriptu kao komponentu. Zatim smo svakome postavili vrijednost „Broj objekata“ koji je definiran u skripti na dva. Mogli smo staviti i više objekata, ali nije bilo potrebe. Također, svakom od tri objekta smo pridružili platforme tj. objektu „PlatformPooler1“ smo pridružili kratku platformu, objektu „PlatformPooler2“ srednju platformu a objektu „PlatformPooler3“ dugačku platformu.



Slika 33: Postavljanje broja objekata i dodavanje platforme

Sada ćemo stvoriti novi objekt naziva „GeneratorPlatformi“ kojem ćemo pridružiti novu skriptu za nasumično generiranje platformi. Objekt pomičemo do desnog ruba zadnje platforme na desnoj strani ekrana, jer od tamo želimo dalje nasumično generiranje novih platformi. Također, kao što je već prije napomenuto, želimo uklanjati tj. deaktivirati objekte nakon što izađu izvan okvira kamere tj. scene koju igrač vidi na svom zaslonu. Isto tako želimo i aktivirati objekte izvan okvira kamere kako sami proces kreiranja igrač ne bi vidio. Zbog toga prvo stvaramo točku kreiranja naziva „TockaKreiranja“ i postavljamo ju unutar objekta kamere te ju pomičemo desno na nešto veću udaljenost od igrača. Samim time između točke generatora platformi i krajnje točke kreiranja postoji razmak koji želimo popuniti s novim platformama. Da bismo to napravili, stvaramo novu C# skriptu naziva „GeneratorPlatformi“. Prvo trebamo definirati javnu varijablu točke kreiranja. Nakon toga trebamo napraviti javno polje (eng. Array) naziva „platformPoolers“ u koje ćemo spremati naše pooler-e koje smo definirali u Unity okruženju. Nadalje, trebamo spremiti širine platformi u privatno polje tipa float naziva „veličinePlatformi“. Unutar „Start()“ metode postavljamo veličinu polja „veličinePlatformi“ korištenjem veličine polja „platformPoolers“. Sada trebamo dohvatiti veličine svih platformi tako što ćemo proći kroz polje „platformPoolers“ te ih postaviti u polje „veličinePlatformi“. Vrijednost veličine dohvaćamo korištenjem komponente „Box Collider 2D“ kao što je prikazano na slici:

```
for(int i = 0; i < platformPoolers.Length; i++){
    velicinePlatformi[i] = platformPoolers[i].pooledObject.GetComponent<BoxCollider2D>().size.x;
}
```

Slika 34: Dohvaćanje veličina platformi korištenjem „Box Collider 2D“ komponente

Sada korištenjem metode „Update()“ želimo generirati platforme. Prvo provjeravamo da li je x pozicija objekta generatora manja od x pozicije krajnje točke kreiranja tj. varijable naziva „točkaKreiranja“. Time provjeravamo da li između te dvije točke postoji „šupljina“ tj. prostor koji treba popuniti. S obzirom da taj prostor treba biti nasumično popunjen, svaku sličicu

stvaramo novi nasumični broj u rasponu od nula pa do veličine „platformPoolers“ polja i spremamo ga u novu varijablu naziva „nasumičnaPlatforma“. Tu se javlja problem pozicije generiranje platformi. Prilikom generiranja, stara tj. već kreirana platforma bi se preklapala s novom platformom jer generator platformi stvara nove platforme na svojoj poziciji, tako da se nakon stvaranja on nalazi u sredini nove platforme. Zbog toga, potrebno je pomaknuti generator u desno po x osi za polovicu nove platforme. Izračun udaljenosti za koju je potrebno napraviti pomak je moguće vidjeti na sljedećoj slici:

```
float udaljenost = velicinePlatformi[nasumicnaPlatforma]/2;
```

Slika 35: Izračun udaljenosti za koju je potrebno pomaknuti generator platformi

Nakon izračuna, koordinati x jednostavno pridružujemo izračunatu vrijednost pomoću „Transform“ komponente slično kao što je već prikazano u prijašnjim primjerima. Nakon promjene pozicije generator, trebamo dohvatiti nasumični deaktivirani objekt, postaviti njegovu poziciju na trenutnu poziciju generatora platformi te ga jednostavno aktivirati kao što je prikazano na sljedećoj slici:

```
GameObject platforma = platformPoolers[nasumicnaPlatforma].dohvatiObjekt();
platforma.transform.position = transform.position;
platforma.SetActive(true);
```

Slika 36: Postavljanje pozicije ponovo iskorištenog objekta i njegovo aktiviranje

Prilikom testiranja se je pojavio problem ponovnog aktiviranja platformi na istoj poziciji tj. bez pomicanja. Problem je u tome što nismo pomaknuli generator platforme nakon aktiviranja objekata. Problem je riješen ponovnim pomicanjem generatora poslije stvaranja objekata na isti način kao što smo prije već to napravili:

```
transform.position = new Vector3(
    transform.position.x + udaljenost,
    transform.position.y,
    transform.position.z
);
```

Slika 37: Pomicanje generatora platformi

Sada se vraćamo i Unity okruženje te pridružujemo objektu „GeneratorPlatformi“ skriptu istog naziva kao komponentu. Odabrano je da će nazivi objekta i skripte koja se odnosi na taj objekt biti isti radi lakšeg razumijevanja. Da bi generator ispravno radio, treba mu dodati tri već prije kreirana „pooler-a“ pomoću kojih će nasumično generirati platforme. Također mu treba dodati krajnju točku kreiranja novih platformi. Točku smo već prije kreirali i postavili u prostoru te ona nosi naziv „TockaKreiranja“.

3.5. Skakanje igrača i uklanjanje platformi

Prilikom testiranja generacije novih platformi, unutar „Hierarchy“ prozora u Unity okruženju je bilo moguće vidjeti stvaranje novih objekata platformi, ali ne i iskorištavanje već postojećih. Kao što je već prije spomenuto, ovakav način beskonačnog stvaranja novih objekata bi s vremenom uzrokovao popunjene memorije uređaja na kojem se igra izvršava te to sigurno želimo izbjeći s obzirom da je igra beskonačna.

Kao što smo za generiranje platformi postavili objekt naziva „TockaKreiranja“ i pomaknuli ga više u desnu stranu po x osi, kako bi stvorili razmak između generatora platformi i navedene točke koji smo trebali popuniti s platformama, tako i sada želimo stvoriti još jednu točku za uklanjanje platformi. Prema tome, unutar objekta kamere naziva „Main Camera“ smo stvorili još jedan prazni objekt naziva „TockaBrisanja“. S obzirom da se igrač kreće po x osi s lijeva na desno, ovu točku smo pomaknuli dalje na lijevu stranu jer želimo deaktivirati sve platforme koje su izašle iz okvira kamere na lijevoj strani. Sada, isto kao što smo za generiranje platformi trebali kreirati skriptu, tako i sada trebamo napraviti novu skriptu za deaktivaciju. Stvorili smo novu C# skriptu naziva „BrisacPlatformi“. Kasnije je uočeno da ovaj naziv skripte baš i ne odgovara točnoj funkciji koju skripta izvršava. Naime, mi ne želimo brisati objekte i kasnije ih ponovo instancirati. Operaciju instantacije želimo izbjeći tj. želimo ostaviti postojeće objekte i samo ih deaktivirati. Prema tome, bolji naziv za ovu skriptu bi bio možda „DeaktivatorPlatformi“ ali s obzirom na to da je problem uočen tek kasnije prilikom rada na projektu, naziv je zadržan kako bi se izbjegla potreba za dodatnim promjenama na više mjesta.

Nakon što smo u Unity okruženju kreirali objekt „TockaBrisanja“, otvaramo C# skriptu i definiramo privatnu varijablu tipa „GameObject naziva „tockaBrisanja“. Sada, da bismo dohvatili u tu točku brisanja koju smo kreirali u Unity okruženju, unutar „Start()“ metode koristimo „Find()“ funkciju za dohvaćanje [44]. S obzirom da želimo brisanje izvršavati cijelo vrijeme dok igra traje, unutra metode „Update()“ koja se izvršava svaku sličicu, želimo provjeriti da li je trenutna pozicija platforme s obzirom na x os manja od pozicije novo kreirane točke brisanja, te ako je, tu platformu želimo deaktivirati. Programski kod za deaktiviranje platformi možemo vidjeti na sljedećoj slici:

```
public class BrisacPlatformi : MonoBehaviour
{
    private GameObject tockaBrisanja;


    void Start()
    {
        tockaBrisanja = GameObject.Find("TockaBrisanja");
    }


    void Update()
    {
        if(transform.position.x < tockaBrisanja.transform.position.x){
            gameObject.SetActive(false);
        }
    }
}
```

Slika 38: Programski kod skripte za deaktivaciju platformi

Jednom kada smo završili s kreiranjem skripte, želimo ju pridružiti objektima u igri koji će ju koristiti tj. objektima platformi. Isto kao i sve druge skripte, i ovu skriptu pridružujemo kao komponentu objekta. Dodavanje vršimo tako što odabiremo objekte platformi iz datoteke „Prefabs“ i svakoj platformi zasebno pridružujemo skriptu. Početnim platformama koje već jesu na sceni ne treba zasebno dodavati skripte s obzirom da nakon što smo kreirali „prefab“ elemente, sa scene smo obrisali postojeće privremene platforme i zamijenili iz sa „prefab“ elementima. Ukratko da pojasnimo proceduru rada. Prije u ovom radu je opisan generator platformi koji uzima iz liste platformi deaktivirane platforme te ih ponovo aktivira. Skripta generatora je radila ali je problem bio u tome što se lista nije popunjavala s deaktiviranim platformama s obzirom da platforme nigdje nismo deaktivirali te se zbog toga lista deaktiviranih platformi nije popunjavala. Zbog toga smo sada dodali skriptu za deaktivaciju koja deaktivira platforme kako bi se one mogle spremati u listu i ponovo aktivirati unutar skripte „GeneratorPlatformi“. Platforme se dohvaćaju nasumično.

Nakon što smo postavili deaktivaciju i ponovnu aktivaciju platformi, vratili smo se nazad na igrača tj. njegove pokrete. Prije smo pomoću skripte naziva „DinosaurKretanje“ već postavili kretanje objekta igrača po x osi s lijeva na desno. No sada je bilo potrebno omogućiti skakanje igrača s obzirom da se radi o platformerskoj igri. Prva stvar koji trebamo provjeriti je da li je igrač na zemlji tj. platformi ili ne jer želimo omogućiti skakanje igrača samo kada dodiruje platformu. Također, radi jednostavnosti igre je odlučeno da igrač neće moći skakati više puta kao što je to slučaj u ponekim drugim igrama gdje igrač može skočiti dva ili više puta dok je objekt igrača iznad platforme tj. „u zraku“. Za provjeru da li je igrač na platformi ili ne, morat ćemo se vratiti u prije već kreiranu skriptu „DinosaurKretanje“. Tamo ćemo iskoristiti metodu „IsTouchingLayers()“ kako bi izvršili provjeru dodira više kolizija [45]. No prije toga, unutar Unity okruženja moramo platformama dodati slojeve (eng. Layers). Navedeni slojevi se obično najviše koriste od strane kamera na sceni kako bi se omogućilo osvjetljavanje samo dijelova scene [46], no u našem slučaju ćemo slojeve koristiti kako bi napravili provjeru nastanka kolizije između objekata. Da bismo napravili slojeve, odlazimo u datoteku „Prefabs“ te

odabiremo bilo koju platformu. U prozoru „Inspector“ možemo vidjeti opciju da za odabir postojećih slojeva. Unutar nje postoji opcija za dodavanje novih slojeva te nju odabiremo. Sada kreiramo sloj jednostavnog naziva „Platforma“. Nakon toga odabiremo ostale dvije platforme i jednostavno im pridružujemo kreirani sloj s obzirom da želimo vršiti provjeru nad svim platformama. No kako bismo pomoću metode „IsTouchingLayers()“ provjerili da li postoji kolizija između platformi i igrača, moramo i objektu igrača dodati novi sloj. Odabiremo objekt igrača naziva „Dinosaur“ te ponavljamo postupak kako bi kreirali sloj naziva „Dinosaur“.

Sada želimo modificirati skriptu „DinosaurKretanje“ koju smo prije već kreirali kao bismo mogli izvršiti provjeru. Kao što smo prije već kreirali varijablu za spremanje brzine kretanja tj. trčanja igrača, sada trebamo stvoriti varijablu za spremanje visine skakanja igrača čiju ćemo vrijednost moći kasnije sami mijenjati unutar Unity okruženja. Prema tome, stvaramo javnu varijabli tipa float naziva „skakanje“. Sada, želimo omogućiti skakanje objekta igrača svaki put kad fizički igrač klikne lijevu tipku miša. Za to koristimo metodu „GetMouseButtonDown()“ tipa „bool“ kojoj prosljeđujemo vrijednost nula kao parametar te koja vraća vrijednost istine tj. „true“ kada igrač klikne postavljenu tipku [47]. U našem slučaju želimo da igrač „skoči“ kada se klikne lijevi klik, pa zato prosljeđujemo vrijednost nule. Provjeru da li je igrač kliknuo lijevu tipku želimo vršiti svaku sličicu te zbog toga provjeru vršimo u metodi „Update()“. Prilikom postavljanja brzine kretanja igrača, mijenjali smo brzinu kretanja po x osi. Sada želimo napraviti istu promjenu, ali po y osi kao što je prikazano na sljedećoj slici:

```
if(Input.GetMouseButtonDown(0)){
    if(igracPrizemljen){
        rigidBody.velocity = new Vector2(rigidBody.velocity.x,skakanje);
    }
}
```

Slika 39: Promjena pozicije igrača s obzirom na y os

Umjesto trenutne vrijednosti y osi, prosljeđujemo vrijednost koju sami postavljamo i spremamo u varijablu „skakanje“. Nadalje, na prošloj slici je moguće vidjeti dodatnu provjeru vezanu za prizemljenost igrača te će ona uskoro biti opisana. Da bi testirali stvari, vraćamo se u Unity okruženje, odabiremo objekt igrača te pod komponentama već pronalazimo skriptu „DinosaurKretanje“ koju smo već prije dodali. Sada unutar postavki komponente vidimo novu postavku naziva „Skakanje“ te trebamo postaviti njenu vrijednost. Nakon kraćeg testiranja, zaključeno je da će vrijednost biti postavljena na 26. No kao što je već prije opisano, ne želimo u igri omogućiti beskonačno skakanje odnosno želimo omogućiti igraču da skoči samo jednom kada se nalazi na površini platforme, te ne želimo da više može skakati dok ne dodiruje platformu. Sada dolazimo do dijela gdje želimo iskoristiti prije kreirane slojeve za izvršavanje provjere dodira objekta igrača i objekta platforme. Unutar koda skripte „ DinosaurKretanje“ definiramo novu varijablu naziva „platforma“ tipa „LayerMask“. Isto tako nam je potrebna

komponenta kolizije (eng. Collider) igrača pa definiramo varijablu tipa „Collider2D“ naziva „dinosaurCollider“. Naziv varijable je samo jednim dijelom na hrvatskom jeziku kako bi se lakše razumjela struktura koda. Sada je bilo potrebno inicijalizirati varijable unutar „Start()“ funkcije. Pomoću „GetComponent<>()“ metode smo dohvatili „Collider2D“ igrača. Ako se sada vratimo na prijašnju sliku u ovom radu, vidjet ćemo dodatnu provjeru stanja varijable naziva „igracPrizemljen“ te ćemo nju sada objasniti. Unutar metode „Update()“ smo postavili navedenu varijablu koja je tipa „bool“. Njenu vrijednost smo definirali pomoću već navedene metode „IsTouchingLayers()“ koja provjerava da li se sloj igrača i sloj platforme dodiruju. Postavljanje vrijednosti varijable je moguće vidjeti na sljedećoj slici:

```
bool igracPrizemljen = Physics2D.IsTouchingLayers(dinosaurCollider,platforma);
```

Slika 40: Postavljanje vrijednosti varijable „igracPrizemljen“

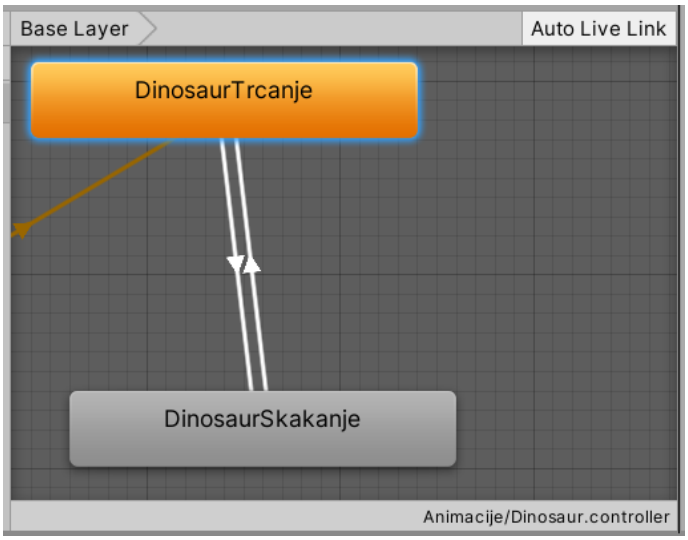
Kao parametre je moguće vidjeti nazive slojeva koje smo već prije postavili unutar Unity okruženja. Metodu pozivamo iz klase „Physics2D“ koja omogućava rad s metodama vezanim za fizička ponašanja objekata [48]. Sada kada je vrijednost varijable postavljena, vršimo dodatnu provjeru koju smo već prije naveli. Kako bismo testirali promjenu, trebamo se vratiti u Unity okruženje te postaviti otvoriti komponentu skripte kretanja igrača. Tamo ćemo vidjeti novu postavku naziva „Platforma“ koju moramo postaviti na vrijednost istog naziva tj. „Platforma“. Postavka je se pojavila zbog toga što smo u kodu dodali javnu varijablu naziva „platforma“ tipa „LayerMask“ kojoj sada u Unity okruženju dodjeljujemo odgovarajući sloj tj. sloj naziva „Platforma“ koji je pridružen svim platformama koje imamo u „Prefabs“ datoteci. Na taj način metoda „IsTouchingLayers()“ zna s kojim slojem da uspoređuje sloj igrača. Nakon testiranja promjene, moguće je bilo vidjeti da igrač sada može skočiti samo jednom prije nego li padne ponovo na platformu.

3.6. Animacija skakanja i postava razmaka između platformi

Nakon što smo uspješno dodali mogućnost skakanja igrača te nakon što smo to testirali, uočili smo da objekt igrača cijeli vrijeme koristi istu animaciju trčanja, što ne želimo. Naš cilj je aktivirati animaciju skakanja ako se igrač odvoji od platforme.

Da bismo kreirali novu animaciju, otvaramo ponovo prozor „Animation“ te dodajemo novu animaciju naziva „DinosaurSkakanje“ i pohranjujemo ju u datoteku naziva „Animacije“ koju smo već prije kreirali. Sada trebamo dodati sliku skakanja. Pošto se ovdje radi o jednostavnoj 2D igri, animacija skakanja se može napraviti korištenjem samo jedne slike te je tako i napravljeno u ovom projektu. Kada smo dodali animaciju, trebamo postaviti uvjete koji definiraju kada će animacija skakanja biti aktivna a kada animacija trčanja. Za to koristimo sučelje naziva „Animator“. Navedeno sučelje služi kao kontrolni mehanizam za animacije čije

se izvršavanje kontrolira pomoću svojstava (eng. Properties) [49]. Sada unutar sučelja trebamo definirati prijelaze (eng. Transition) animacija. U ovom slučaju, postavljanje je jednostavno. Kada igrač trči, moguće je preći u stanje skakanja i samim time se treba pokrenuti animacija skakanja. Kada je igrač u stanju skakanja, moguće je preći u stanje trčanja i samim time se treba pokrenuti animacija trčanja nakon toga. Dodane animacije su sučelju „Animator“ prikazane u obliku članova. Da bismo ostvarili prijelaz, desnim klikom odabiremo jednu animaciju te odabiremo opciju „Make Transition“ te ju nakon toga povezujemo s drugom animacijom. U našem slučaju, na taj način prvo povezujemo animaciju „DinosaurTrcanje“ s animacijom „DinosaurSkakanje“. Nakon toga ponavljamo istu stvar, ali samo u obrnutom redoslijedu. Na ovaj način smo ostvarili upravo ono što je maloprije opisano tj. stvoreni su prijelazi između animacija. Povezane animacije možemo vidjeti na sljedećoj slici:



Slika 41: Ostvareni prijelazi između animacija

Na slici iznad su prijelazi između animacija prikazani bijelim strelicama između njih. Sada trebamo dodati uvjete pomoću parametara. Odabiremo prvi prijelaz tj. strelicu koja povezuje animaciju „DinosaurTrcanje“ s animacijom „DinosaurSkakanje“ i otvaramo prozor „Parameters“. Unutar navedenog prozora dodajemo novi parametar tipa „bool“ naziva „IgracPrizemljen“. Sada u prozoru „Conditions“ dodajemo parametar i njegovu vrijednost postavljamo na „false“. Postavljanje vrijednosti na „false“ znači da će animacija skakanja biti aktivirana samo kada igrač nije na platformi. Na isti način sada dodajemo parametar za prijelaz između animacije „DinosaurSkakanje“ i animacije „DinosaurTrcanje“. Odabiremo prijelaz, odlazimo u prozor „Conditions“ i dodajemo isti parametar „IgracPrizemljen“ ali ovaj put njegovu vrijednost stavljamo na „true“ s obzirom da se animacija trčanja mora pokrenuti kada je objekt igrača na platformi.

Sada kada smo dodali uvjete, moramo otvoriti skriptu „DinosaurKretanje“ kako bi napravili potrebne promjene. Prvo definiramo varijablu tipa „Animator“ naziva „animator“.

Unutar „Start()“ metode pomoću metode „GetComponent()“ dohvaćamo komponentu „Animator“ koja je pridružena objektu igrača. Unutar „Update()“ postavljamo vrijednost varijable „animator“ pomoću parametra „igracPrizemljen“. Sada ako se vratimo u Unity okruženje i pokrenemo igru, moguće je vidjeti da se animacija igrača mijenja prilikom skakanja. Prilikom promatranja prijelaza između animacija, bilo je moguće vidjeti da postoji određeno vrijeme prije nego li se animacije promjene. Da bismo maknuli vrijeme potrebno za promjenu animacije, vratili smo se u prozor „Animator“, odabrali jedan prijelaz te otvorili prozor „Inspector“. Tamo je bilo moguće vidjeti postavku „Transition Duration“ te smo nju postavili na vrijednost nula.Isto smo napravili i s drugim prijelazom.

Jednom kada smo postavili prijelaze između animacija igrača, vratili smo se na problem generiranja platformi po kojima će se igrač kretati. Da bi otežali igru, htjeli smo dodati razmak između platformi. Samim time će nam se omogućiti da stvorimo mogućnost gubitka igrača. Također, odlučeno je kako ćemo dodati i razlike u visini između platformi kako bi igra bila zanimljivija. Ako se vratimo na neke stvari koje smo prije opisali u radu, možemo vidjeti kako smo dodali dvije točke za prostor generiranja i prostor deaktiviranja platformi. Točke su bile pomaknute po x osi na desnu i lijevu stranu ovisno o njihovoj ulozi. Sličan princip rada ćemo iskoristiti i ovdje.

Unutar naše scene ćemo sada dodati dvije točke za visinu naziva „MinimalnaVisina“ i „MaksimalnaVisina“. Točke smo pomaknuli unutar scene na odgovarajuće lokacije te su one kasnije dodatno pomicane prilikom testiranja kako bi se postigao kvalitetniji rezultat. Sada je potrebno modificirati C# skriptu „GeneratorPlatformi“ kako bi mogli ostvariti pomicanje platformi u visinu. Definiramo dvije nove javne varijable istih naziva kao i dodane točke naziva „minimalnaVisina“ i „maksimalnaVisina“ tipa „Transform“ s obzirom da „Transform“ komponenta sadrži koordinate po kojima se platforme pomiču. Pomoću tih varijabli ćemo dohvaćati početne vrijednosti navedenih točki koje smo prije postavili unutar Unity okruženja. Nadalje, sada još dodajemo dvije varijable tipa „float“ naziva „minY“ i „maxY“ u koje ćemo kasnije spremati nasumične visine platformi. Na kraju dodajemo još dvije varijable isto tipa „float“ naziva „minRazmak“ i „maxRazmak“ pomoću kojih ćemo definirati minimalni i maksimalni dopušteni razmak između platformi. Unutar metode „Update()“ trebamo sada postaviti nasumični razmak. Definiramo varijablu „razmak“ tipa „float“ te pomoću metode „Random()“ kojoj prosljeđujemo vrijednosti varijabli „minRazmak“ i „maxRazmak“ generiramo vrijednost razmaka. Vrijednosti varijabli „minRazmak“ i „maxRazmak“ kasnije trebamo postaviti unutar Unity okruženja. Sada dodatno modificiramo već postojeću „Transform“ komponentu tako što vrijednosti x osi pridodajemo vrijednost varijable „razmak“ kao što je prikazano na slici:

```
float razmak = Random.Range(minRazmak,maxRazmak);
transform.position = new Vector3(
    transform.position.x + udaljenost + razmak,
    visina,
    transform.position.z
);
```

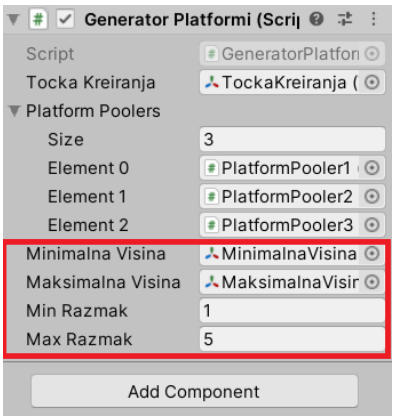
Slika 42: Modificiranje „Transform“ komponente kako bi stvorili razmak

Na prijašnjoj slici je moguće vidjeti primjenu varijable naziva „visina“ koja zamjenjuje y vrijednost. Ta varijabla će biti uskoro naknadno opisana. Sada kada smo između platformi dodali nasumičnu vrijednost razmaka, trebamo se vratiti promjeni visine platformi. Prvo trebamo dohvatiti početne vrijednosti y koordinata točaka koje smo postavili na sceni. Njihovim postavljanjem na sceni tj. pomicanjem po y osi određujemo maksimalnu i minimalnu visinu platformi. Na taj način kasnije prilikom testiranja možemo lakše promijeniti te vrijednosti samim pomicanjem točaka na sceni, te ne moramo mijenjati programski kod. Da bismo dohvatili početne vrijednosti točaka tj. vrijednosti njihovih y koordinata, unutar „Start()“ metode dohvaćamo y vrijednosti „Transform“ komponenti pojedine točke te ih spremamo u varijable „minY“ i „maxY“. Sada, isto kao i sa stvaranjem razmaka između platformi, trebamo stvoriti nasumičnu vrijednost za visinu. Unutar „Update()“ metode definiramo već maloprije spomenutu varijablu naziva „visina“ te pozivom „Random()“ funkcije kojoj prosljeđujemo vrijednosti varijabli „minY“ i „maxY“ postavljamo nasumičnu vrijednost kao što je prikazano na slici:

```
float visina = Random.Range(minY, maxY);
```

Slika 43: Postavljanje nasumične vrijednosti visine

Nakon toga modificiramo vrijednost y „Transform“ komponente tako što umjesto y vrijednosti postavljamo varijablu „visina“ kao što je već prikazano na pretprošloj slici. Ako se sada vratimo u Unity okruženje, odabirom objekta igrača vidimo da su dodane nove stavke koje treba popuniti. Potrebno je dodati objekte točaka visine platformi koje smo prije kreirali te postaviti maksimalnu i minimalnu vrijednost razmaka između platformi kao što je vidljivo na slici:

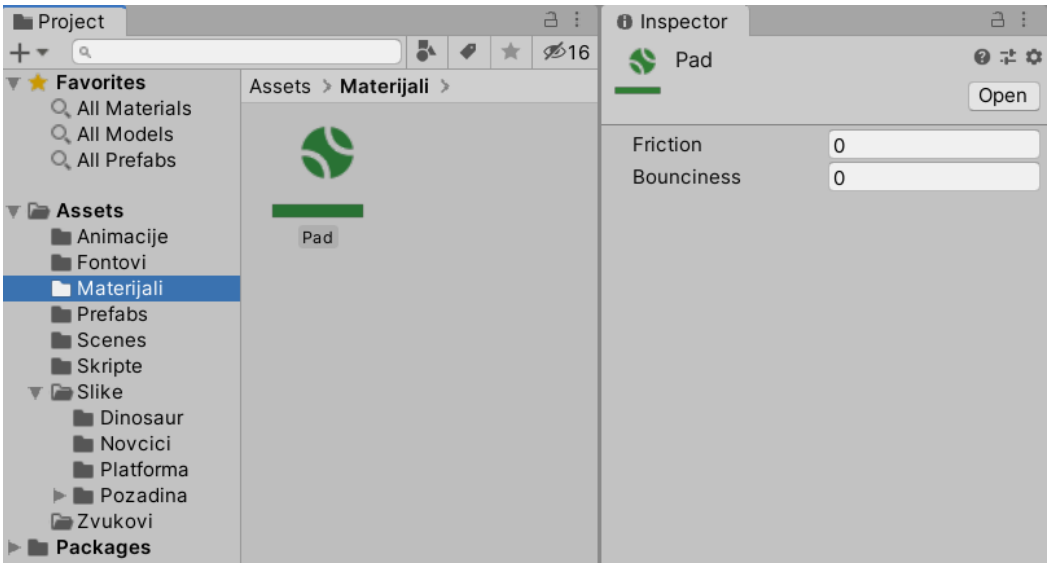


Slika 44: Postavljene vrijednosti razmaka i visine

Nakon kraćeg eksperimentiranja je uočeno da je najbolje postaviti maksimalni razmak na vrijednost 5 te su minimalna i maksimalna visina malo prepravljene unutar scene jednostavnim pomicanjem objekata točki. Nadalje, odmah je prepravljena vrijednost pada igrača s obzirom da je bila premala i igrač je presporo padao. Promjena je napravljena tako što smo odabrali objekt igrača te promijenili vrijednost postavke „Gravitiy Scale“ na pet.

Kao što je već prije spomenuto, jednom kada smo postavili razmak između platformi te generirali nasumičnu visinu, možemo se pozabaviti samim završetkom igre tj. sa „smrti“ igrača. Za sad smo samo radi testiranja u našu scenu dodali novi 3D element kocke naziva „HvatacIgraca“ na koju će objekt igrača pasti. Promijenili smo dimenzije kocke s obzirom na x os kako igrač ne bi pao pokraj objekta tj. kako bi sigurno pao na njega. Objekt smo premjestili unutar objekta kamere kako bi se on kao takav kretao zajedno sa kamerom odnosno sa objektom igrača te smo ga isto tako na sceni pomaknuli niže izvan okvira kamere kako ga pravi tj. fizički igrač ne bi vidio na zaslonu. Sa „smrti“ igrača ćemo se pozabaviti kasnije jer nam je trenutno bio cilj samo testirati pravilno propadanje objekta igrača između platformi i njegovo hvatanje kako bi kasnije mogli kreirati završetak igre.

Tijekom testiranja je uočen još jedan problem. S obzirom da su platforme podosta velike, igrač je mogao ponekad zapeti za lijevi rub platforme na koju je skakao i samim time nije padao dolje tj. igraču je bilo olakšano penjanje a to nismo htjeli. Kao bismo to popravili, kreirali smo novu datoteku naziva „Materijali“. Unutar datoteke smo stvorili novi materijal naziva „Physics material 2D“. Navedeni materijal omogućava stvaranje trenja između objekata isto kao i jačine odbijanja objekata [50]. U našem slučaju, trebamo promijeniti razinu trenja kako bi ostvarili pad igrača te smo samim time dodali materijalu naziv „Pad“ kako bi znali na što se odnosi iako se ispostavilo da nam naziv nije previše bitan s obzirom da je to jedini materijal koji je bio potreban u našem projektu. S obzirom da je problem u objektu igrača, trebamo njemu pridružiti novo kreirani materijal. Odabirem objekt dinosaura te unutar komponente naziva „Rigidbody2D“ vidimo postavku naziva „Materials“ unutar koje postavljamo naš materijal. Sada kada je objektu igrača pridružen materijal, potrebno je promijeniti svojstvo trenja. Odabiremo materijal iz datoteke „Materijali“ i postavku „Friction“ postavljamo na vrijednost nula. Isto tako, postavku „Bounciness“ isto ostavljamo na nuli s obzirom da ne želimo da se naš igrač odbija od drugih objekata. Postavke je moguće vidjeti na sljedećoj slici:



Slika 45: Smještaj i postavke materijala „Pad“

3.7. Generiranje i sakupljanje bodova

Naša igra je u ovom stanju radila dobro. No falio je sami cilj igre. U većini igara ovog tipa postoje bodovi koje igrač ostvaruje kretanjem kroz razinu ili ih sakuplja usput. U našem slučaju je odlučeno da ćemo imati oba pristupa. Igrač će bodove sakupljati s vremenom kako se kreće te će isto tako moći sakupiti dodatne bodove koji će mu se naknadno pribrajati.

Odlučeno je kako će dodatni bodovi izgledati u obliku „novčića“ koje će igrač sakupljati. Na Internetu je pronađen besplatni skup slika novčića [51] koji je vizualnim stilom odgovarao našoj igri. Kreirana je nova datoteka naziva „Novcici“ te su u nju iz preuzetog skupa prebačene sve slike. Sada je kao i s prijašnjim slikama bilo potrebno svakoj slici promijeniti postavku „Pixels per Unit“ na 256 kako bi slike vizualno odgovarale ostatku igre. Skup slika je predstavljao rotaciju jednog novčića te je zbog toga trebalo stvoriti novu animaciju. Isto tako, novčići će biti automatski nasumično generirani na platformama te samim time predstavljaju element koji će se ponovo iskorištavati. Prvo smo postavili jednu sliku novčića na scenu te ju preimenovali u „Novcic“. Kao i ostali objekti u igri, i novčići trebaju imati koliziju s drugim objektima kako bi se mogli sakupljati tj. kako igrač ne bi prolazio kroz njih. Prema tome, pod komponenta ma je ponovo bilo potrebno dodati komponentu kolizije. Prilikom dodavanja komponente, uočeno je da već postoji predefinirana komponenta kolizije za objekte kružnog oblika naziva „Circle Collider 2D“ te smo nju i koristili. Opseg kolizije se automatski detektira i postavlja, te ne postoji potreba više za naknadnim promjenama iako su one moguće. Sada je bilo potrebno napraviti animaciju okretanja novčića na isti način kao što su kreirane i prošle animacije. Unutar prozora „Animation“ smo prebacili pravilnim redoslijedom sve slike novčića

te je animacija automatski generirana te smo mi samo trebali malo smanjiti brzinu rotacije kako bi se ostvario ugodniji prikaz. Naziv animacije je postavljen na „NovcicRotacija“.

Sada smo testirali igru kako bi vidjeli hoće li objekt igrača zapeti za objekt novčića, te je to i ostvareno s obzirom da oba imaju komponente kolizije. No poanta igre je da igrač sakuplja novčiće tj. da oni nestaju kako igrač prolazi kroz njih. Kako bi to ostvarili, bilo je potrebno pokrenuti programsku funkciju svaki put kada se dogodi kolizija između igrača i novčića. Odabirom objekta novčića te proučavanjem njegove komponente „Circle Collider 2D“ je bilo moguće vidjeti postavku naziva „Is Trigger“ koja navedenu komponentu pretvara u okidač funkcije [52]. S obzirom da će se kreirani novčić koristiti više puta tj. bit će automatski generiran, koristit ćemo ga kao „prefab“ element te smo ga zbog toga prebacili u datoteku „Prefabs“.

Kao što smo prije kreirali novi objekt naziva „GeneratorPlatformi“ koji smo slobodno pomicali unutar scene, tako smo i sada stvorili objekt naziva „GeneratorNovcica“ kako bi mogli upravljati njihovim generiranjem. Unutar datoteke skripti smo kreirali novu C# skriptu identičnog naziva kako bi lakše povezali stvari prilikom rada. Unutar skripte smo obrisali automatski generirane metode „Start()“ i „Update()“ jer nam one u ovom slučaju nisu trebale. S obzirom da će se novčići automatski generirati isto kao i platforme, i u ovom slučaju se želimo pozabaviti aktivacijom i deaktivacijom već kreiranih objekata kako bi izbjegli problem pretrpavanja memorije uređaja na kojem je igra pokrenuta. Prema tome, iskoristili smo već postojeću skriptu „ObjectPooler“. Kreirali smo novu javnu varijablu tipa „ObjectPooler“ naziva „novciciPooler“. Sada smo kreirali metodu za stvaranje novčića naziva „StvaranjeNovcica“ koja kao argumente prima poziciju na kojoj će se generirati novčići te kao drugi argument prima širinu platforme s obzirom da ne želimo da novčići generiraju izvan same širine platforme.

Isto kao i kod platformi, i ovdje smo trebali dohvatiti jedan od deaktiviranih objekata pomoću „ObjectPooler“ skripte. Nadalje, želimo postaviti i poziciju novčića korištenjem „Transform“ komponente. Na kraju trebamo samo aktivirati objekt novčića. No s obzirom da želimo generirati skup novčića, trebamo generirati nasumični broj novčića po svakoj platformi. Prema tome, odlučeno je da će minimalni broj novčića generiran po platformi biti tri a maksimalni broj će ovisiti o veličini platforme s obzirom da ne želimo da novčići izlaze sa platforme. Veličina platforme je proslijeđena pomoću parametara metode.

Kako bi generirali veći broj novčića a ne samo jedan, kod za generiranje smo postavili u petlju čiji kraj ovisi o nasumično generiranom broju novčića. Sada smo se vratili u Unity okruženje te smo odabrali objekt „GeneratorNovcica“ i njemu pridružili kreiranu skriptu kao novu komponentu. Kako bi mogli koristiti „ObjectPooler“, unutar spomenutog objekta smo napravili novi prazni objekt naziva „Novcici Pooler“ te smo njemu pridružili skriptu

„ObjectPooler“ kao bi ju mogli koristiti. Sada kada smo dodali skriptu, pojavile su se dvije postavke. Prva se je odnosila na tip objekta koji će se koristiti te smo za to postavili objekt novčića iz „Prefabs“ datoteke. Druga opcija se je odnosila na broj objekata i nju smo postavili na 20 iako je možda moglo i manje. Sada smo ponovo odabrali objekt „Generator novčića“ i te smo pod postavku „Novcic Pooler“ postavili novo kreirani objekt kako bi skripta za generiranje novčića znala koji objekt koristiti. Da bismo pozvali metodu „StvaranjeNovcica()“ iz kreirane skripte, otvorili smo skriptu „GeneratorPlatformi“ te definirali novu varijablu tipa „GeneratorNovicica“ kako bismo mogli pozvati metodu. Unutar „Start()“ metode smo inicijalizirali varijablu. Sada je samo bilo potrebno ispod koda za generiranje platformi pozvati metodu i proslijediti joj potrebne parametre pozicije aktivirane platforme i njene veličine.

U ovom trenutku smo testirali generiranje novčića, te je uočen jedan problem. Svi novčići koji su bili generirani su se nalazili na istoj poziciji. Da bi ispravili taj problem, vratili smo se u programski kod te smo modificirali poziciju po x koordinati jer smo novčiće trebali pomicati u desnu stranu. Problem smo jednostavno popravili dodavanjem vrijednosti varijable „i“ iz petlje unutar koje se nalazi kod za generiranje. Ponovnim testiranjem je uočeno da se novčići generiranju na krivim pozicijama tj. u sredini platformi. Zbog toga smo se vratili u skriptu za generiranje te smo koordinati y pridružili vrijednost jedan kako bi se pozicija novčića podigla malo iznad platforme. Nadalje, uočeno je da se novčići ipak generiraju izvan platforme tj. na desnoj strani prelaze rub s obzirom da se generiranje počinje od sredine tj. polovice platforme. Kako bi to popravili, od x pozicije smo oduzeli polovicu platforme. Nakon testiranja smo još poziciji x odlučili pridružiti vrijednost 4 kako bi pomakli generiranje ipak malo u desnu stranu jer su sada novčići prelazili malo lijevi rub platforme, iako to nije bilo preveliki problem. Također je uočeno da se novčići generiraju za svaku platformu te je odlučeno kako bi to trebalo promijeniti jer nije imalo baš previše smisla u kontekstu same igre. Problem je riješen tako što smo ponovo koristili nasumično generirane brojeve. Ovaj put se je samo trebalo igrati malo s brojevima te smo generatoru dali raspon od 1 do 100 te smo odlučili, da ako je generirani broj manji od recimo 50, da se ostatak metode ne izvodi tj. dio metode koji generira novčiće na trenutnoj platformi. Sada su se novčići generirali samo na nekim platformama, ali su uvijek bili generirani na razini igrača tako da on nije morao skakati kako bi ih pokupio. Prema tome smo i y koordinatu novčića promijenili tako što smo joj pridružili nasumično generiran broj između tri i pet. Programski kod metode je moguće vidjeti na sljedećoj slici:

```
public class GeneratorNovcica : MonoBehaviour
{
    public ObjectPooler novciciPooler;

    public void StvaranjeNovcica(Vector3 pozicija, float velicinaPlatforme){

        int nasumicnoGeneriranje = Random.Range(1,100);

        if(nasumicnoGeneriranje < 50){
            return;
        }

        int brojNovcica = (int)Random.Range(3f,velicinaPlatforme);
        float y = Random.Range(3,5);


        for(int i = 0; i < brojNovcica; i++){
            GameObject novcic = novciciPooler.dohvatiObjekt();

            float x = pozicija.x - (velicinaPlatforme/2) + 4;


            novcic.transform.position = new Vector3(
                x + i,
                pozicija.y + y,
                1
            );

            novcic.SetActive(true);
        }
    }

}
```

Slika 46: Programski kod metode za stvaranje novčića

Sada kada smo generirali novčiće na željeni način, trebamo omogućiti igraču da ih i sakupi i dobije bodove za njih. Jedan od problema koje nismo popravili tijekom generacije novčića je njihovo deaktiviranje kako bi se mogli ponovo iskoristiti. S obzirom da smo već kreirali skriptu za deaktiviranje platformi, odlučeno je kako ćemo nju iskoristiti i za deaktiviranje novčića pošto se princip toga ne mijenja. Kako bismo deaktivirali novčiće, trebamo navedenu skriptu dodati kao komponentu. Iz datoteke „Prefabs“ smo odabrali objekt novčića te mu pridružili skriptu „BrisacPlatformi“ koja je primarno služila s deaktivaciju platformi iako ju i ovdje možemo iskoristiti. Nikakve naknade promjene u kodu nisu bile potrebne te je prilikom testiranja uočeno da se novčići deaktiviraju.

Sada se vraćamo na problem sakupljanja novčića. Da bismo to ostvarili, trebali smo kreirati novu C# skriptu. Unutar datoteke „Skripte“ smo stvorili novu skriptu naziva „SakupljacNovcica“. Otvorili smo skriptu i uklonili „Update()“ metodu pošto nam nije bila potrebna. Kako bi omogućili sakupljanje novčića, trebali smo definirati što se događa kada se objekt igrača „sudari“ s objektom novčića. Za to postoji već definirana metoda naziva „OnTriggerEnter2D()“ unutar koje trebamo definirati svoj kod [53]. Metoda kao argument primat objekt tipa „Collider2D“ pomoću kojeg unutar „if“ metode provjeravamo da li je proslijeđen objekt igrača (dinosaura). Ako je proslijeđen objekt igrača, onda unutar „if“ uvjeta deaktiviramo trenutni objekt koji koristiti ovu skriptu tj. u našem slučaju je to objekt novčića. Deaktiviranjem objekta novčića sa scene stvaramo iluziju „sakupljanja“ koju će igrač vidjeti na ekranu. Sada

je bilo potrebno otići u Unity okruženje, odabrati objekt novčića iz „Prefabs“ datoteke te mu dodati opisanu skriptu kao komponentu. Sada prilikom testiranja smo mogli vidjeti da se objekti novčića deaktiviraju. Programski kod za navedenu skriptu možemo vidjeti na slici ispod:

```
void OnTriggerEnter2D(Collider2D other){
    if(other.gameObject.name == "Dinosaur"){
        gameObject.SetActive(false);

        if(zvukSakupljanjaNovcica.isPlaying){
            zvukSakupljanjaNovcica.Stop();
        }
        zvukSakupljanjaNovcica.Play();

        upravljacRezultatom.rezultat += novcicBodovi;
    }
}
```

Slika 47: Programski kod metode za deaktivaciju novčića

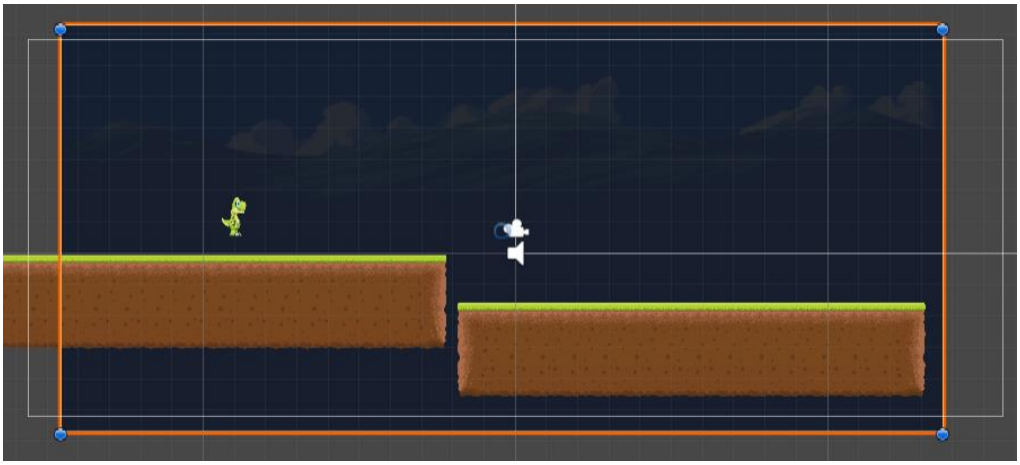
S obzirom da je ovaj rad pisan nakon završetka projekta, unutar metode s prethodne slike je moguće vidjeti još dijelova programskog koda koji se odnosi na pribrajanje bodova prilikom sakupljanja novčića te dodavanja zvukova sakupljanja novčića te će cijeli postupak biti uskoro naknadno opisan. Za sad smo samo deaktivirali novčiće s ekrana kako bi ostvarili vizualni prikaz.

3.8. Pozadina igre i zvučni efekti

Kako bismo obogatili igru s vizualnim stilom, potrebno je dodati sliku pozadine u igru jer je u ovom trenutku za pozadinu bila postavljena samo tamo plava boja i ništa više. Isto kao i s ostalim vizualnim elementima, i ovdje je bilo potrebno pronaći odgovarajuću pozadinu. Pozadinu smo pronašli na stranici za besplatne elemente igre [54]. Prvo smo skinuli skup od više elemenata pozadine te smo trebali odabrati koje elemente ćemo koristiti. S obzirom da naša igra sadrži platforme po kojima se igrač kreće, odabrali smo sliku neba za pozadinu te smo ostale elemente iz skupa odbacili. Odabranu sliku pozadine smo ubacili u datoteku slika u Unity okruženju. Kako se igrač kreće kroz igru, željeli smo da cijelo vrijeme pozadina bude ista ali i da bude pokretna u smislu da ne želimo da samo prati objekt igrača kao što to radi kamera, već da se cijelo vrijeme ponavlja. To je značilo da se je slika pozadine trebala ponavljati. Prema tome, odabrali smo objekt pozadine, te unutar prozora „Inspector“ promijeniti postavku „Wrap Mode“ na „Repeat“.

Sada smo unutar objekta kamere napravili novi objekt četverokuta (eng. Quad). Unutar scene smo promijenili njegovu veličinu tako da bude malo veći od okvira kamere. Pomoću ovog objekta ćemo prikazivati sliku pozadine tj. upravljati s njom s obzirom da će nam biti potrebna nova skripta za pomicanje pozadine. Kako bismo upravljali pozadinom pomoću

objekta četverokuta, dodali smo mu sliku pozadine iz datoteke „Pozadina“ koja se nalazi unutar datoteke „Slike“ kao novu komponentu. Unutar datoteke „Pozadina se je sada automatski generirala nova datoteka naziva „Materials“ te unutar njega materijal pozadine. S obzirom da je naša pozadina bila jako tamna kada smo ju dodali objektu četverokuta, trebali smo promijeniti opciju „Shader“ na „Unlit/Texture“. Navedena opcija, koji je najlakše prevesti kao sjenčanje, se odnosi na promjenu svojstava elemenata na ekranu prilikom promjene njihove pozicije [55]. Prilikom rada sa 3D ili 2D objektima, to se obično odnosi na dodavanje efekta sjene ili zatamnjenja prilikom promjene pozicije elementa na ekranu. S obzirom da u našem projektu nemamo postavljene zasebne postavke za sjenčanje, naša slika je cijela bila zatamnjena tj. „u sjeni“. Odabirom spomenute opcije smo uklonili bilo kakve oblike sjenčanja te se je sada slika pravilno prikazivala na ekranu tj. bila je osvjetljenja. Izgled pozadine s efektom sjenčanja bez naknadnih postavki možemo vidjeti na sljedećoj slici:



Slika 48: Slika pozadine sa sjenčanjem (eng. Shader)

Sada kada smo pravilno dodali sliku pozadine, potrebno je bilo omogućiti njeno pomicanje. Već smo odredili da se pozadina može ponavljati ali nam to u ovom trenutku nije ništa značilo s obzirom da je pozadina ionako bila statična. No glavni razlog dodavanja objekta četverokuta je i bila mogućnost dodavanja skripte za pomicanje slike. Prema tome, napravili smo novu C# skriptu naziva „PomicanjePozadine“. Pregledom unutar Unity okruženja, vidjeli smo da je slika pozadine unutar četverokuta dodana pod komponentu naziva „Mesh Renderer“ te smo prema tome kreirali novu varijablu tipa „Renderer“ kako bi dohvatili pozadinu u kodu. Također smo napravili javnu varijablu tipa „float“ kako bi unutar Unity okruženja mogli kontrolirati brzinu pomicanja pozadine. Unutar „Update()“ metode, definirali smo pomicanje teksture pozadine korištenjem metode „mainTextureOffset()“ [56] kao što je prikazano na sljedećoj slici:

```
void Update()
{
    pozadina.material.mainTextureOffset += new Vector2(brzinaPozadine * Time.deltaTime,0f);
}
```

Slika 49: Kod za pomicanje pozadinske slike

Da objasnimo malo prijašnju metodu. S obzirom da se metoda „Update()“ izvršava svaku sličicu, potrebno je svaku sličicu pomicati pozadinu. Brzinu pomicanja pozadine mijenjamo sami unutar Unity okruženja tj. postavljamo sami vrijednost varijable „brzinaPozadine“. Da bi smo omogućili pomicanje, koristili smo Vector2() objekt koji prima x i y koordinate kao parametre. S obzirom da se je naša pozadina trebala pomicati po x koordinati, brzinu pomicanja smo pomnožili s vremenom koje je proteklo od zadnje sličice tj. sa „Time.deltaTime“ [57] te vrijednost postavili na mjesto parametra x koordinate, a vrijednost parametra y koordinate smo ostavili na nuli s obzirom da pomicanje ne želimo vršiti po y osi. Sada je bilo potrebno samo dodati skriptu objektu četverokuta i poigrati se s vrijednosti brzine pomicanja.

Sada kada smo se dodatno pozabavili vizualnim elementima, odlučili smo dodati i zvučne efekte. Prvo smo kreirali prazan objekt naziva „Zvukovi“ unutar kojega ćemo dodavati sve naše zvučne efekte. Kao što je već prije u ovom radu navedeno, imat ćemo četiri zvučna efekta: zvuk pozadine [58], zvuk sakupljanja novčića [59] te zvukove skakanja i umiranja [60]. Svi paketi efekata su preuzeti s Interneta te su besplatni za korištenje.

Prilikom dodavanja efekata pod objekt „Zvukovi“, trebalo je promijeniti opciju „Play On Awake“ za sve efekte osim za efekt pozadine kako bi se prilikom pokretanja igre čuo samo pozadinski zvuk. Također je za pozadinski zvuk trebalo označiti opciju „Loop“ koja omogućuje ponavljanje pozadinskog zvuka jednom kada završi.

Da bismo definirali kada će se koji zvukovi puštati, bilo je potrebno modificirati skriptu „DinosaurKretanje“ kako bi mu mogli pridružiti efekte skakanja i smrti. Unutar skripte smo dodali dvije nove javne varijable tipa „AudioSource“ naziva „zvukSmrti“ i „zvukSkakanja“. Tim varijablama ćemo pridružiti zvukove unutar Unity okruženja, a unutar metode „Update()“ ćemo ih jednostavno pokrenuti pozivom „Play()“ metode kada je to potrebno tj. zvuk skakanja ćemo pokrenuti u uvjetu za skakanje kao što je vidljivo na slici:

```
if(Input.GetMouseButtonDown(0)){
    if(igracPrizemljen){
        zvukSkakanja.Play();
        rigidBody.velocity = new Vector2(rigidBody.velocity.x,skakanje);
    }
}
```

Slika 50: Pokretanje zvuka skakanja igrača pomoću „Play()“ metode

Pokretanje zvuka smrti ćemo objasniti kasnije kada ćemo opisivati kraj igre. Sada nam je još ostao zvuk sakupljanja novčića, te s obzirom da se taj zvuk treba pokrenuti za svaki pojedini objekt novčića kojeg igrač „sakupi“ moramo otvoriti C# skriptu naziva „SakupljacNovcica“ te tamo stvoriti privatnu novu „AudioSource“ varijablu naziva „zvukSakupljanjaNovcica“. Unutar metode „Start()“ smo pomoću metode „Find()“ pronašli odgovarajući zvučni efekt te smo ga unutar metode „Update()“ pokrenuli. Prilikom testiranja igra uočen je problem sa zvukom sakupljanja novčića tj. navedeni zvuk se nije svaki puta pokretao kada je igrač sakupio novi objekt novčića. Problem je bio u tome što navedeni zvučni efekt traje određeno vrijeme te se prvo mora izvršiti u potpunosti do kraja prije nego li se pokrene ponovo ako igrač sakupi novi objekt novčića za kojeg je taj efekt vezan. Kako bi popravili taj problem, trebalo je provjeriti da li je efekt već pokrenut s metodom „isPlaying()“ te ukoliko je, trebalo ga je prvo zaustaviti s metodom „Stop()“ te iza toga ponovo pokrenuti sa „Play()“. Programski kod skripte „SakupljacNovcica“ možemo vidjeti na sljedećoj slici:

```
public class SakupljacNovcica : MonoBehaviour
{

    private AudioSource zvukSakupljanjaNovcica;
    private float novcicBodovi = 3f;
    private UpravljacRezultatom upravljacRezultatom;

    void Start()
    {
        zvukSakupljanjaNovcica = GameObject.Find("Novcic").GetComponent<AudioSource>();
        upravljacRezultatom = FindObjectOfType<UpravljacRezultatom>();
    }

    void OnTriggerEnter2D(Collider2D other){
        if(other.gameObject.name == "Dinosaur"){
            gameObject.SetActive(false);

            if(zvukSakupljanjaNovcica.isPlaying){
                zvukSakupljanjaNovcica.Stop();
            }
            zvukSakupljanjaNovcica.Play();

            upravljacRezultatom.rezultat += novcicBodovi;
        }
    }
}
```

Slika 51: Programski kod skripte „SakupljacNovcica“

Kao što je već prije spomenuto, ovaj rad je pisan nakon završetka projekta te se na prijašnjoj slici može vidjeti još dodatna varijabla tipa „UpravljacRezultatom“ koja nije još objašnjena te će biti kasnije pojašnjena njena uloga. Prije testiranja je također trebalo inicijalizirati zvuk za objekt igrača unutar Unity okruženja kako bi se efekt izvodio pravilno.

3.9. Sakupljanje bodova

Jedna od stavki koji smo već prije opisali se je odnosila na sakupljanje bodova u igri. Definirano je kako će igrač dobivati bodove samim kretanjem kroz razinu te će moći ostvariti dodatne bodove sakupljanjem novčića.

Za početak je trebalo omogućiti prikaz bodova na ekranu kako bi lakše testirali programsku skriptu za bodove koju ćemo stvoriti te ujedno kako bi i igrač vidio svoje bodove. Za to je trebalo dodati natpise na ekran. Odlučeno je kako će se vidjeti natpis trenutnih bodova koje je igrač ostvario te natpis najvišeg prijašnjeg ostvarenog rezultata kako bi igrač znao da li je uspio napredovati tj. ostvariti bolji rezultat od prijašnjeg najboljeg. Natpise u igri dodajemo isto kao objekte s kojima možemo upravljati. Objekte koje dodajemo su tipa „Text“. Prema tome, dodali smo prvo dva objekta, „NajboljiRezNatpis“ s vrijednosti „Najbolji rezultat:“ te objekt „RezultatNatpis“ s vrijednosti „Rezultat:“. Natpise smo postavili na ekranu u gornji lijevi kut te smo nakon toga dodali još dva objekta, „NajboljiRezTekst“ i „RezultatTekst“ kojima smo početne vrijednosti postavili na „00000“. Njih smo isto smjestili na odgovarajuće pozicije na ekranu te ćemo se dalje pozabaviti s njima jer njihove vrijednosti želimo mijenjati ovisno o rezultatu koji ostvari igrač. Također, fontovi natpisa nisu baš odgovarali vizualnom stilu igre te smo s Interneta preuzeli i iskoristili font naziva „Kero kero“ [61] koji je bolje odgovarao željenom stilu.

Jednom kada smo postavili sve potrebne natpise, bilo je potrebno napraviti novu C# skriptu pomoću koje ćemo upravljati s rezultatom igrača. Nakon kreiranja skripte naziva „UpravljacRezultatom“, kreirali smo i novi objekt identičnog naziva pomoću kojega ćemo kasnije mijenjati vrijednosti bodovanja igrača. Unutar kreiranog objekta smo prebacili sve objekte natpisa tj. teksta koje smo kreirali. Unutar skripte smo definirali prvo varijablu za određivanje količine bodova koje će igrač dobivati svake sekunde. Zatim smo kreirali dvije varijable tipa „float“ pomoću kojih ćemo računati rezultat. Nadalje, trebale su nam varijable tipa „Text“ za promjenu natpisa trenutnog i najboljeg rezultata kojeg smo izračunali.

Nakon definiranja varijabli, unutar metode „Update()“ izračunali smo rezultat dobiven kretanjem kroz razinu tako što smo prijašnjem rezultatu pribrojili umnožak bodova koje igrač dobiva svake sekunde pomnožene s vremenom od prošle sličice. Također je trebalo vršiti provjeru, da li je igrač ostvario veći rezultat od prijašnjeg najboljeg rezultata te ako je, ažurirati vrijednost i tog rezultata. Kako bismo sačuvali najbolji rezultat za svaku sljedeću igru, bilo je potrebno sačuvati rezultat po postavke igrača s određenim imenom. To je ostvareno na sljedeći način:

```
PlayerPrefs.SetFloat("NajboljiRezultatKey",najboljiRezultat);
```

Slika 52: Spremanje najboljeg rezultata pod postavke igrača

Prilikom spremanja je trebalo definirati ključ pod kojim ćemo moći pronaći vrijednost [62]. Naziv ključa smo postavili na „NajboljiRezultatKey“ te smo unutar „Start()“ metode provjerili da li postoji ključ s tim nazivom, te ako postoji dohvatili smo njegovu vrijednost kako bi mogli raditi s njom i mijenjati ju kasnije:

```
void Start()
{
    rezultatSePovecava=true;
    if(PlayerPrefs.HasKey("NajboljiRezultatKey")){
        najboljiRezultat = PlayerPrefs.GetFloat("NajboljiRezultatKey");
    }
}
```

Slika 53: Dohvaćanje vrijednosti najboljeg rezultata

Sad kada smo postavili sve potrebno za izračun rezultata prilikom kretanja igrača, rezultat je bilo potrebno prikazati na ekranu postavljanjem vrijednosti natpisa:

```
rezultatTekst.text = Mathf.Round(rezultat).ToString();
najboljiRezTekst.text = Mathf.Round(najboljiRezultat).ToString();
```

Slika 54: Postavljanje vrijednosti rezultata

S obzirom da su rezultati tipa „float“, pomoću „Round()“ metode smo uklonili decimale te smo rezultate pretvorili u vrijednosti tipa „string“. Sada smo u Unity okruženju objektu „UpravljacRezultatom“ pridružili skriptu koju smo sada kreirali. Nakon toga su se pojavile postavke koje je trebalo namjestiti kao što je prikazano na slici:

Slika 55: Postavke rezultata

Sada kada smo omogućili dobivanje bodova kretanjem kroz razine, trebamo preći na drugi dio bodovanja, a to su dodatni bodovi ostvareni sakupljanjem novčića. Prema tome, otvorili smo već prije kreiranu skriptu „SakupljacNovcica“ i definirali novu varijablu „novciciBodovi“ s kojom smo definirali vrijednost bodova koje će igrač ostvariti sakupljanjem pojedinog novčića. Kako bismo povećali ukupni rezultat, potreban nam je objekt tipa „UpravljacRezultatom“ . Unutar „Start()“ metode smo dohvatili objekt upravljača rezultatom te pomoću njega unutar metode „Update()“ povećali ukupni rezultat:

```
upravljacRezultatom.rezultat += novcicBodovi;
```

Slika 56: Povećanje ukupnog rezultata dodavanjem bodova sakupljanja novčića

3.10. Smrt igrača i dovršetak razvoja igre

U ovom poglavlju ćemo se osvrnuti na postavljanje uvjeta smrti igrača te na promjene koje su naknadno napravljene kako bi igra bila dovršena.

Kao što je već prije u radu navedeno, da bi povećali kompleksnost igre, dodali smo platforme na različitim udaljenostima i visinama. Također smo omogućili igraču da sakuplja dodatne bodove prolaskom kroz razinu. No prilikom testiranja je uočeno kako igra nije zapravo puno teža s time, te je odlučeno kako će se brzina kretanja objekta igrača povećavati kako igrač napreduje sve dalje i dalje.

Kako bismo ostvarili navedenu promjenu, moramo otvoriti prvo skriptu „DinosaurKretanje“. Zatim smo dodali tri nove varijable: „tockaUbrzanja“, „brojacTockiUbrzanja“ i „uvecavacBrzine“. Sve tri varijable su tipa „float“. Prva varijabla, „tockaUbrzanja“, predstavlja udaljenost nakon koje će se brzina kretanja povećavati. Varijabla „brojacTockiUbrzanja“ nam služi kako bi imali ukupan broj točki ubrzanja spremljen te pomoću vrijednosti te varijable uvećavali brzinu svaki put kada igrač dosegne sljedeću točku ubrzanja. Zadnja varijabla, „uvecavacBrzine“, služi kao vrijednost s kojom ćemo pomnožiti točku ubrzanja kako bi ostvarili ubrzanje.

Unutar „Start()“ metode smo postavili vrijednost varijable „brojacTockiUbrznanja“ jednako varijabli „tockaUbrzanja“ s obzirom da na početku igre te dvije vrijednosti moraju biti jednake. Sada smo unutar „Update()“ metode provjerili da li je pozicija igrača po x koordinati veća od vrijednosti varijable „brojacTockiUbrzanja“. Ukoliko je, povećali smo vrijednost varijable „brzinaDinosaura“ tako što smo je pomnožili s vrijednost varijable „uvecavacBrzine“. Bilo je potrebno i povećati vrijednost varijable „brojacTockiUbrzanja“ za vrijednost varijable „tockaUbrzanja“, no isto tako nismo svaki put prilikom dodavanja vrijednosti htjeli istu vrijednost varijable „tockaUbrzanja“ pa smo i njenu vrijednost unutar uvjeta povećali za umnožak stare vrijednost varijable i varijable „uvecavacBrzine“.

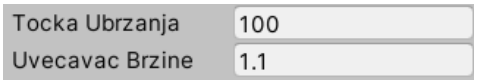
Sada smo se vratili u Unity okruženje te je bilo potrebno inicijalizirati vrijednosti novih varijabli koje smo dodali. S obzirom da je skripta „DinosaurKretanje“ dodana objektu igrača, odabiremo ga te pod komponentama nalazimo navedenu skriptu. Unutar nje vidimo da su se pojavile vrijednosti koje trebamo postaviti. Za udaljenost početne točke ubrzanja postavljamo vrijednost 100. Odlučeno je da se vrijednost ubrzanja poveća za 10% svaki puta kada igrač

dosegne točku ubrzanja. Programski kod za povećanje brzine unutar skripte je moguće vidjeti na sljedećoj slici:

```
if(transform.position.x > brojacTockiUbrzanja){
    brojacTockiUbrzanja += tockaUbrzanja;
    brzinaDinosaura = brzinaDinosaura * uvecavacBrzine;
    tockaUbrzanja += tockaUbrzanja * uvecavacBrzine;
}
```

Slika 57: Uvjet povećanja brzine kretanja igrača

S obzirom da želimo povećati brzinu za 10%, vrijednost varijable „uvecavacBrzine“ mora biti postavljena na 1.1 unutar Unity okruženja kao što je prikazano na slici:



Slika 58: Postavljanje vrijednosti varijabli ubrzanja unutar Unity okruženja

Sada kada smo otežali igru, trebamo se pozabaviti njenim krajem tj. sa „smrti“ igrača. Prije je već opisano kako smo kreirali objekt „HvatacIgraca“ koji slijedi poziciju igrača kako bi igrač pao na taj objekt. Sada ćemo opisati način kako je taj objekt iskorišten za kraj igre.

Prva stvar koji smo trebali napraviti je dodati sloj (eng. Layer) objektu „HvatacIgraca“ kako bi olakšali rad s njime. Način dodavanja slojeva objektima je već opisan u ovom radu, a za naziv sloja smo postavili „KrajIgrePlatforma“ kako bi u kodu olakšali razumijevanje svrhe ovog objekta. Nakon toga smo otvorili skriptu „DinosaurKretanje“ te smo dodali varijablu „smrtPlatforma“ tipa „LayerMask“. Unutar metode „Update()“ smo pomoću metode „IsTouchingLayers()“, koja je već prije u radu opisana, provjerili da li igrač dodiruje postavljeni sloj te smo dobivenu vrijednost spremili u varijablu tipa „bool“ naziva „igracMrtav“ kako bi ju kasnije mogli iskoristiti.

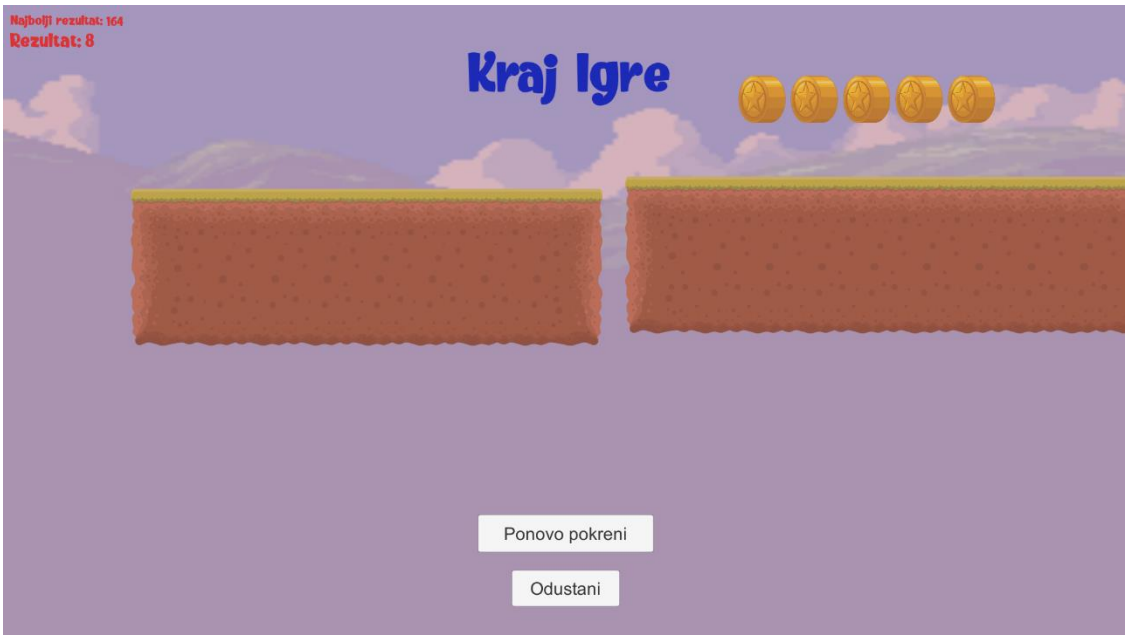
Kako bi se na ekranu igrača moglo vidjeti da je igrač „poginuo“, potrebno je bilo sastaviti odgovarajući prikaz. Vratili smo se nazad u Unity okruženje te smo stvorili novi prazni objekt naziva „UpravljacIgrom“ pomoću kojega ćemo zaustavljati igru kada igrač „pogine“. Unutar tog objekta smo htjeli postaviti sve potrebne elemente za kraj igre te smo zbog toga kreirali još jedan prazan objekt naziva „KrajIgreIzbornik“. Jednom kada igra završi, igrač će moći odabrati dvije opcije, opciju za ponovnu igru ili opciju za izlazak iz igre. Za početak smo htjeli malo zatamniti ekran ukoliko dođe do smrti te smo zbog toga htjeli dodati djelomično transparentnu sliku preko ekrana. Unutar objekta „KrajIgreIzbornik“ smo kreirali novi objekt tipa „Image“ te ju proširili preko cijelog ekrana. Da bi simbolizirali kraj igre, boju slike smo promijenili na svijetlo crvenu te smo uz to promijenili transparentnost, tako da se u pozadini slike vidi igra. Nakon toga je bilo potrebno još dodati novi objekt tipa „Text“ pomoću kojega smo ispisali poruku o

kraju igre te smo primijenili već prije spomenuti font koji smo preuzeli s Interneta kako bi natpis odgovarao ostatku igre.

Sada kada smo stvorili natpis, dodali smo dva gumba tj. objekta tipa „Button“ kako bi igraču omogućili ponovno pokretanja igre ili odustajanje te smo im promijenili veličinu kako bi bili uočljiviji na zaslonu. Sada kad smo postavili zaslon za kraj igre, trebalo ga je prikazivati samo na kraju igre. Prema tome, trebali smo deaktivirati objekt „KrajIgreIzbornik“ unutar kojeg se nalaze svi potrebni elementi kako bi izbjegli njegovo prikazivanje na početku igre. Njegovom deaktivacijom smo deaktivirali i sve ostale objekte kojima je on roditelj:

Slika 59: Deaktivirani objekti zaslona za kraj igre

Izgled konačnog zaslona za kraj igre zajedno sa potrebnim elementima je moguće vidjeti na sljedećoj slici:



Slika 60: Izgled zaslona kraja igre

Kako bi mogli upravljati prikazom zaslona, kreirali smo novu C# skriptu naziva „UpravljacIgrom“ s obzirom da ćemo kasnije navedenu skriptu pridružiti objektu istoga naziva. Unutar skripte smo definirali tri nove metode: „KrajIgre()“, „PonovoIgraj()“ i „Odustani()“. Nakon toga, kako bi mogli stvoriti programski kod za navedene metode, morali smo definirati potrebne varijable. S obzirom da se ovdje radi o skripti koja služi za upravljanje cijelom igrom, bio nam je potreban veliki broj elemenata kao što je objekt igrača itd. Sve potrebne varijable nećemo

opisivati ponovo s obzirom da su već opisane prije u ovom radu te ih je moguće vidjeti na sljedećoj slici:

```
public DinosaurKretanje dinosaur;
public UpravljacRezultatom upravljacRezultatom;
public AudioSource zvukPozadine;
public AudioSource zvukSmrti;
private Vector3 tockaPocetkaIgraca;
private Vector3 tockaPocetkaGeneriranjaPlatformi;
public GeneratorPlatformi generatorPlatformi;
public GameObject velikaPlatforma;
public GameObject srednjaPlatforma;
public GameObject krajIgreIzbornik;
```

Slika 61: Varijable skripte „UpravljacIgrom“

Unutar metode „Start()“ smo inicijalizirali varijablu „tockaPocetkaIgraca“ koja predstavlja početnu poziciju objekta igrača prilikom pokretanja igre. Razlog tome je što kada igrač odabere ponovno pokretanje igre, pozicija objekta igrača se treba vratiti na početak. Ista situacija je i s točkom generiranja platformi koju smo prije kreirali. Kako igra traje, objekt generatora platformi mijenja svoju poziciju tj. prati objekt igrača te zbog toga ju moramo vratiti na početak ako igrač želi ponovo pokrenuti igru. Ukratko opisano, oba objekta na početku igre imaju definirane svoje početne pozicije a s obzirom na to da se metoda „Start()“ izvršava samo jednom tj. na početku igre, trebamo odmah prilikom prvog pokretanja dohvatiti i spremiti pozicije tih objekata. Još jedna stvar koju moramo napraviti prilikom pokretanja igre je deaktivirati objekt „krajIgreIzbornik“ kako se odmah na početku ne bi prikazivao zaslon kraja igre zajedno sa njegovim elementima:

```
void Start()
{
    tockaPocetkaIgraca = dinosaur.transform.position;
    tockaPocetkaGeneriranjaPlatformi = generatorPlatformi.transform.position;
    krajIgreIzbornik.SetActive(false);
}
```

Slika 62: Dohvaćanje pozicija potrebnih objekata te deaktivacija zaslona kraja igre

Kao i prije, vrijednosti svih potrebnih varijabli će biti postavljene unutar Unity okruženja. Jednom kada igra završi tj. kada igrač „pogine“, potrebno je napraviti određene promjene. Prvo je potrebno deaktivirati objekt igrača. Nakon toga, ne želimo da nam se rezultat više povećava te smo deaktivirali i povećanje rezultata postavljanjem vrijednosti varijable „rezultatSePovecava“ koju smo definirali unutar skripte „UpravljacRezultatom“. Sad kada smo deaktivirali potrebne elemente, prikazali smo kreirani zaslon kraja igre te pokrenuli zvuk „smrti“. Tijekom testiranja igre je bio uočen problem preklapanja zvuka smrti sa zvukom pozadine te smo se vratili u programski dok i prije pokretanja zvuka „smrti“ smo zaustavili zvuk pozadine. Programski kod metode za kraj igre je moguće vidjeti na sljedećoj slici:

```
public void KrajIgre()
{
    dinosaur.gameObject.SetActive(false);
    krajIgreIzbornik.SetActive(true);
    upravljacRezultatom.rezultatSePovecava = false;
    zvukPozadine.Stop();
    zvukSmrti.Play();
}
```

Slika 63: Metoda za zaustavljanje tj. kraj igre

Kako bi pozvali kreiranu metodu „KrajIgre()“, vratili smo se u skriptu „DinosaurKretanje()“ i iskoristili vrijednost prije opisane „bool“ varijable naziva „igracMrtav“ koja sadržavala vrijednost istine (eng. True) ako je objekt igrača dodirivao objekt „HvatacIgraca“ tj. ako je igrač pao s platforme. Da bismo pozvali navedenu metodu, definirali smo novu varijablu tipa „UpravljacIgrom“ jer se u njoj nalazi metoda za kraj igre, te smo pomoću uvjeta provjerili da li je vrijednost varijable „igracMrtav“ postavljena na istinu te ako je, pozvali smo metodu za kraj igre. Vrijednost varijable upravljača igrom je postavljena unutar Unity okruženja odabirom komponente „DinosaurKretanje“ koje je pridružena objektu igrača.

Kao što je već opisano, u prošloj metodi smo aktivirali zaslon kraja igre. Na tom zaslonu je igrač imao dva gumba tj. dvije opcije, opciju ponovnog pokretanja igre ili opciju izlaska. Prema tome, već smo stvorili dodatne dvije metode, „Odustani()“ i „PonovoIgraj()“. Unutar metode „Odustani()“ smo dodali jednu liniju koja zatvara cijelu aplikaciju [63]. Prilikom testiranja je uočeno da se metoda može izvršiti samo ako se poziva na fizičkom uređaju te se ne izvršava unutar Unity okruženja, no to nije predstavljalo problem. Nadalje, zadnja metoda se je odnosila na ponovno pokretanje igre. Kako bi razumjeli što je sve potrebno napisati u nju, vratili smo se u metodu „KrajIgre()“ da bi vidjeli što sve od objekata deaktivirali. Također, kada igra završi, ostaju aktivirani objekti platformi te smo zbog trebali dohvatiti sve aktivirane objekte tako što smo kreirali polje naziva „brisac“ tipa „BrisacPlatformi“ i u njega spremili sve objekte. Prošli smo kroz polje pomoću petlje te ih redom sve deaktivirali. Navedeno polje je tipa „BrisacPlatformi“ s obzirom da svi objekti platformi imaju pridruženu skriptu identičnog naziva, kako bi ih se moglo deaktivirati kada izađu izvan okvira kamere. Nadalje, s obzirom da naša igra uvijek na početku ima dvije ručno postavljene platforme, morali smo obje platforme aktivirati sami. Nakon toga smo postavili poziciju igrača te poziciju generatora platformi nazad na početak pomoću vrijednosti varijabli koje smo postavili unutar „Start()“ metode. S obzirom da sada radimo na metodi za ponovno pokretanje igre, zaslon kraja igre će biti aktiviran te ga je potrebno deaktivirati. Aktivirali smo ponovo objekt igrača, postavili rezultat nazad na nulu, te aktivirali povećanje rezultata postavljanjem vrijednosti varijable „rezultatSePovecava“ skripte „UpravljacRezultatom“ na istinu (eng. True). U ovom trenutku se nisu izvršavali nikakvi zvučni efekti, te smo zbog toga samo trebali ponovo pokrenuti zvuk pozadine:

```
public void PonovoIgraj()
{
    BrisacPlatformi[] brisac = FindObjectsOfType<BrisacPlatformi>();
    for(int i = 0; i< brisac.Length; i++){
        brisac[i].gameObject.SetActive(false);
    }
    velikaPlatforma.SetActive(true);
    srednjaPlatforma.SetActive(true);
    dinosaur.transform.position = tockaPocetkaIgraca;
    generatorPlatformi.transform.position = tockaPocetkaGeneriranjaPlatformi;
    krajIgreIzbornik.SetActive(false);
    dinosaur.gameObject.SetActive(true);
    zvukPozadine.Play();
    upravljacRezultatom.rezultat = 0;
    upravljacRezultatom.rezultatSePovecava = true;
}
```

Slika 64: Programski kod metode „PonovoIgraj()“

Sada kad su metode implementirane, vratili smo se nazad u Unity okruženje te smo odabrali naše gumbe i unutar komponenti im pridružili skriptu „UpravljacIgrom“ te iz nje pozvali odgovarajuće metode sukladno željenoj funkcionalnosti pojedinog gumba. Samim time je naša igra bila završena.

4. Zaključak

Prilikom kreiranja ovog rada smo se orijentirali na samo ograničeni skup tehnologija i metoda koje se najčešće koriste prilikom razvoja igara. Prilikom izrade praktičnog dijela sam se upoznao s principima korištenja pokretača igara te 3D programa koji služe za izradu modela i dodjeljivanje tekstura tim modelima. Za izradu modela je korišten besplatan program Blender dok se je za prebacivanje modela u igru koristio pokretač igre Giants Editor. Za uređivanje xml datoteka je korišten uređivač teksta Notepad++.

Nakon izrade modela, napravljena je jednostavna 2D platformerska igra u pokretaču Unity. Za pisanje svih potrebnih skripti je korišten jezik C# te Visual Studio Code IDE. Kao rezultat rada smo dobili jednostavnu igru u kojoj se igrač kreće po platformama te sa lijevom tipkom miša skače kako bi prešao sa jedne platforme na drugu te kako bi sakupio dodatne bodove. Igra je izrađena kako bi se bolje upoznao sa preprekama koje je potrebno prijeći prilikom kreiranja računalnih igara.

Svidio mi se je način na koje određene igre dopuštaju kreiranje vlastitog sadržaja od strane običnih korisnika u obliku modifikacija. Smatram da takav pristup razvoju znatno povećava vrijednost igre jer omogućava veću i dulju zainteresiranost igrača za određenu igru bez značajnih troškova za proizvođača. Također, sadržaj koji kreiraju korisnici može davati dobre smjernice proizvođaču na promjene koje treba napraviti s trenutnom ili nekom budućom igrom koju će razvijati.

Popis literature

[1] J. Tyson, „How video game system work“, 2000. [Na internetu]. Dostupno: https://electronics.howstuffworks.com/video-game2.htm [Pristupano 4.4.2020.]

[2] S. Kotila, „GDLC [Game Development Life Cycle]“, 2018. [Na internetu]. Dostupno: http://www.unity3dtechguru.com/2018/01/gdlc-game-development-life-cycle.html [Pristupano 4.4.2020.]

[3] Get educated (bez dat.), „How to become a video game tester“ [Na internetu]. Dostupno: https://www.geteducated.com/career-center/how-to-become-a-video-game-tester/ [Pristupano 4.4.2020.]

[4] Guru99 (bez dat.), „What is BLACK Box Testing? Techniques, Example & Types“ [Na internetu]. Dostupno: https://www.guru99.com/black-box-testing.html [Pristupano 4.4.2020.]

[5] Vironit, „How much does it cost to make a video game?“, 2018. [Na internetu]. Dostupno: https://vironit.com/how-much-does-it-cost-to-make-a-video-game/ [Pristupano 5.4.2020.]

[6] Unity Store (bez dat.), „Unity Pro“ [Na internetu]. Dostupno: https://store.unity.com/products/unity-pro [Pristupano 5.4.2020.]

[7] Firebase (bez dat.), „Pricing plan“ [Na internetu]. Dostupno: https://firebase.google.com/pricing?hl=hr [Pristupano 6.4.2020.]

[8] Trošak razvoja igara [Slika] (bez dat.) Dostupno: https://vironit.com/wp-content/uploads/2018/06/%5EC7CECB9F2368C44CA15442C459770557AA2EE41077FD69F7E2%5Epimgpsh_fullsize_distr.jpg [Pristupano 7.4.2020.]

[9] Interesting Engineering, „How do Game Engines work ?“, 2016. [Na internetu]. Dostupno: https://interestingengineering.com/how-game-engines-work [Pristupano 7.4.2020.]

[10] IndieGameDev, „Comparison of game engines 2020“, 2020. [Na internetu]. Dostupno: https://indiegamedev.net/2020/02/11/comparison-of-game-engines-2020/ [Pristupano 7.4.2020.]

[11] UnrealEngine, „License options“ [Na internetu]. Dostupno: https://www.unrealengine.com/en-US/get-now/agnostic [Pristupano 7.4.2020.]

[12] RPG Maker, „Products“ [Na internetu]. Dostupno: https://www.rpgmakerweb.com/products/programs/rpg-maker-mv [Pristupano 7.4.2020.]

[13] Open Source Initiative, „MIT License“ [Na internetu] Dostupno: https://opensource.org/licenses/MIT [Pristupano 7.4.2020.]

[14] Educba, „Maya vs 3ds Max vs Blender“ [Na internetu]. Dostupno: https://www.educba.com/maya-vs-3ds-max-vs-blender/ [Pristupano 8.4.2020.]

[15] Adobe, „Creative Cloud Plans & Pricing“ [Na internetu]. Dostupno: https://www.adobe.com/creativecloud/plans.html?plan=individual&filter=all [Pristupano 8.4.2020.]

[16] Gimp, „Frequently Asked Questions“ [Na internetu]. Dostupno: https://www.gimp.org/docs/userfaq.html [Pristupano 8.4.2020.]

[17] S. Johnson, „What is a Vector Image?“ [Na internetu]. Dostupno: https://vimm.com/what-is-a-vector-image/ [Pristupano 8.4.2020.]

[18] InkSpace, „Karakteristike Inkscapea“ [Na internetu]. Dostupno: https://inkscape.org/hr/o-programu/karakteristike/ [Pristupano 8.4.2020.]

[19] Pixelmator, „Pixelmator Pro“ [Na internetu]. Dostupno: https://www.pixelmator.com/pro/ [Pristupano 8.4.2020.]

[20] Cgdirector, „The Albedo Map Mystery Revealed“, 2020. [Na internetu]. Dostupno: https://www.cgdirector.com/albedo-map/ [Pristupano 8.4.2020.]

[21] TreeHouse, „Understanding Normal Maps“, 2015. [Na internetu]. Dostupno: https://blog.teamtreehouse.com/understanding-normal-maps [Pristupano 8.4.2020.]

[22] SplashDamage, „Specular maps“, 2007. [Na internetu]. Dostupno: https://wiki.splashdamage.com/index.php/Specular_Maps [Pristupano 9.4.2020.]

[23] Reflektirajuća („specular“) mapa kamene površine [Slika] (bez dat.) Dostupno: https://cgiknowledge.files.wordpress.com/2013/02/specular_maps_bricks_image.jpg [Pristupano 9.4.2020.]

[24] SlideShare, „Advantages and Disadvantages of Motion Capture“ [Na internetu]. Dostupno: https://www.slideshare.net/animationcoursesahmedabad/advantages-and-disadvantages-of-motion-capture [Pristupano 10.4.2020.]

[25] M. Kines, „Planning and Directing Motion Capture for Games“ [Na internetu]. Dostupno:https://www.gamasutra.com/view/feature/3420/planning_and_directing_motion_.php?print=1 [Pristupano 10.4.2020.]

[26] Primjer scene snimanja pokreta za akcijsku igru [Slika] (bez dat.) Dostupno: https://gameon.studio/wp-content/uploads/2016/04/Mocap_2.jpg [Pristupano 11.4.2020.]

[27] FarmingSimulator, „Mods“ [Na internetu]. Dostupno: https://www.farming-simulator.com/mods.php?lang=en&country=us [Pristupano 11.4.2020.]

[28] Giants Developer Network, „Home“ [Na internetu]. Dostupno: https://gdn.giants-software.com/index.php [Pristupano 11.4.2020.]

[29] T. Denham, „What is UV Mapping & Unwrapping?“ [Na internetu]. Dostupno: https://conceptartempire.com/uv-mapping-unwrapping/ [Pristupano 11.4.2020.]

[30] Reprezentacija UV mapiranja [Slika] (bez dat.) Dostupno: https://cdn.conceptartempire.com/images/02/6627/00-featured-uv-unwrapping-blender-sample.jpg [Pristupano 11.4.2020.]

[31] Textures.com, „Home“ [Na internetu]. Dostupno: https://www.textures.com/ [Pristupano 11.4.2020.]

[32] Computer Hope, „Keyframe“ [Na internetu]. Dostupno: https://www.computerhope.com/jargon/k/key-frame.htm [Pristupano 11.4.2020.]

[33] Microsoft, „Visual Studio Code“ (verzija 1.42.1) (2020) [Na internetu]. Dostupno: https://code.visualstudio.com/download [Pristupano 12.4.2020.]

[34] Unity Technologies, „Unity“ (verzija 2019.3.7f1) (2020) [Na internetu]. Dostupno: https://store.unity.com/download [Pristupano 12.4.2020.]

[35] GameArt2D, „Cute Dino – Free Sprite“ [Na internetu]. Dostupno: https://www.gameart2d.com/free-dino-sprites.html [Pristupano 12.4.2020.]

[36] Unity, „Unity manual – RigidBody“ [Na internetu]. Dostupno: https://docs.unity3d.com/Manual/class-Rigidbody.html [Pristupano 12.4.2020.]

[37] Unity, „Unity manual – Vector2“, [Na internetu]. Dostupno: https://docs.unity3d.com/ScriptReference/Vector2.html [Pristupano 12.4.2020.]

[38] GameArt2D, „Free platformer game tile set“, [Na internetu]. Dostupno: https://www.gameart2d.com/free-platformer-game-tileset.html [Pristupano 12.4.2020.]

[39] Unity, „Unity manual – Unity hotkeys“, [Na internetu]. Dostupno: https://docs.unity3d.com/2017.3/Documentation/Manual/UnityHotkeys.html [Pristupano 13.4.2020.]

[40] Unity, „Unity manual – Prefabs“, [Na internetu]. Dostupno: https://docs.unity3d.com/Manual/Prefabs.html [Pristupano 13.4.2020.]

[41] Unity, „Unity manual – Vector3“, [Na internetu]. Dostupno: https://docs.unity3d.com/ScriptReference/Vector3.html [Pristupano 13.4.2020.]

[42] Mark Placzek, „Object Pooling in Unity“, (23.11.2016), [Na internetu]. Dostupno: https://www.raywenderlich.com/847-object-pooling-in-unity [Pristupano 13.4.2020.]

[43] Unity, „Unity manual – Instantiate“, [Na internetu]. Dostupno: https://docs.unity3d.com/ScriptReference/Object.Instantiate.html [Pristupano 13.4.2020.]

[44] Unity, „Unity manual – Find“, [Na internetu]. Dostupno: https://docs.unity3d.com/ScriptReference/GameObject.Find.html [Pristupano 14.4.2020.]

[45] Unity, „Unity manual – IsTouchingLayers“, [Na internetu]. Dostupno: https://docs.unity3d.com/ScriptReference/Collider2D.IsTouchingLayers.html [Pristupano 14.4.2020.]

[46] Unity, „Unity manual – Layers“, [Na internetu]. Dostupno: https://docs.unity3d.com/Manual/Layers.html [Pristupano 14.4.2020.]

[47] Unity, „Unity manual – GetMouseButtonDown“, [Na internetu]. Dostupno: https://docs.unity3d.com/ScriptReference/Input.GetMouseButtonDown.html [Pristupano 14.4.2020.]

[48] Unity, „Unity manual – Physics2D“, [Na internetu]. Dostupno: https://docs.unity3d.com/ScriptReference/Physics2D.html [Pristupano 14.4.2020.]

[49] Unity, „Unity manual – Animator“, [Na internetu]. Dostupno: https://docs.unity3d.com/ScriptReference/Animator.html [Pristupano 15.4.2020.]

[50] Unity, „Unity manual – Physics Material 2D“, [Na internetu]. Dostupno: https://docs.unity3d.com/Manual/class-PhysicsMaterial2D.html [Pristupano 15.4.2020.]

[51] Free Game Assets, „Free Game Coins Sprite“, [Na internetu]. Dostupno: https://free-game-assets.itch.io/free-game-coins-sprite [Pristupano 15.4.2020.]

[52] Unity, „Unity manual – IsTrigger“, [Na internetu]. Dostupno: https://docs.unity3d.com/ScriptReference/Collider-isTrigger.html [Pristupano 15.4.2020.]

[53] Unity, „Unity manual – OnTriggerEnter2D(Collider2D)“, [Na internetu]. Dostupno: https://docs.unity3d.com/ScriptReference/MonoBehaviour.OnTriggerEnter2D.html [Pristupano 15.4.2020.]

[54] Ansimuz, „Country side platformer“, [Na internetu]. Dostupno: https://ansimuz.itch.io/country-side-platfformer- [Pristupano 15.4.2020.]

[55] The book of shaders, „What is fragment shader“, [Na internetu]. Dostupno: https://thebookofshaders.com/01/ [Pristupano 16.4.2020.]

[56] Unity, „Unity manual – mainTextureOffset“, [Na internetu]. Dostupno: https://docs.unity3d.com/ScriptReference/Material-mainTextureOffset.html [Pristupano 16.4.2020.]

[57] Unity, „Unity manual – deltaTime“, [Na internetu]. Dostupno: https://docs.unity3d.com/ScriptReference/Time-deltaTime.html [Pristupano 16.4.2020.]

[58] OpenGameArt, „Generic 8-bit JRPG Soundtrack“, [Na internetu]. Dostupno: https://opengameart.org/content/generic-8-bit-jrpg-soundtrack [Pristupano 16.4.2020.]

[59] OpenGameArt, „8-Bit Sound Effect Pack (Vol. 001)“, [Na internetu]. Dostupno: https://opengameart.org/content/8-bit-sound-effect-pack-vol-001 [Pristupano 16.4.2020.]

[60] OpenGameArt, „8-Bit Sound Effects Library“, [Na internetu]. Dostupno: https://opengameart.org/content/8-bit-sound-effects-library [Pristupano 17.4.2020.]

[61] Da Font, „Kero kero“, [Na internetu]. Dostupno: https://www.dafont.com/kero-kero.font [Pristupano 17.4.2020.]

[62] Unity, „Unity manual – deltaTime“, [Na internetu]. Dostupno: https://docs.unity3d.com/ScriptReference/PlayerPrefs.html [Pristupano 17.4.2020.]

[63] Unity, „Unity manual – Aplication.Quit“, [Na internetu]. Dostupno: https://docs.unity3d.com/ScriptReference/Application.Quit.html [Pristupano 17.4.2020.]

Popis slika

Slika 1: Životni ciklus razvoja igre [2] 2
Slika 2: Primjer moguće podjele uloga u timu prije početka razvoja [2] 4
Slika 3: Trošak razvoja igara [8] 9
Slika 4: Vlastiti projekt u Unity pokretaču prilagođenom za rad sa 2D igrama 12
Slika 5: Albedo mapa kreirana iz fotografije zida [20] 16
Slika 6: Prikaz normala na površini 3D modela [21] 17
Slika 7: Utjecaj koji „normal“ mapa ima na teksturu [21] 17
Slika 8: Reflektirajuća („specular“) mapa kamene površine [23] 18
Slika 9: Dijagram toka pokreta koje je potrebno snimiti [25] 20
Slika 10: Primjer scene snimanja pokreta za akcijsku igru [26] 20
Slika 11: Primjer modifikacija spremnih za preuzimanje sa službene stranice [27] 21
Slika 12: Gotov model ostave bez tekstura 22
Slika 13: Reprezentacija UV mapiranja [30] 23
Slika 14: Model nakon dodavanja željenih tekstura 24
Slika 15: 3D model u Giants Editor pokretaču 25
Slika 16: XML datoteka „modDesc“ 25
Slika 17: Struktura elemenata unutar Giants Editora 26
Slika 18: Elementi potrebni za animiranje i zvuk 27
Slika 19: Objekt naziva „Simple Croatian Farm Shed“ unutar igre Farming Simulator 19 28
Slika 20: Postavljanje objekta u igri na željenu lokaciju od strane igrača 28
Slika 21: Mogućnost otvaranja kliznih vrata od strane igrača 29
Slika 22: Primjer stanja trčanja igrača (dinosaura) 31
Slika 23: Postavke za izradu i pokretanje projekta na računalu 32
Slika 24: Postavljanje parametara Vector2 objekta 34
Slika 25: Animacija kretanja igrača 34
Slika 26: Izgled kratke platforme 35
Slika 27: Dohvaćanje objekta dinosaura (igrača) 37
Slika 28: Dohvaćanje pozicije igrača 37
Slika 29: Izračunavanje udaljenosti kamere od objekta igrača 37
Slika 30: Kod metode „Update()“ za ažuriranje pozicije kamere 38
Slika 31: Inicijalizacija i dodavanje objekata u listu 39
Slika 32: Dohvaćanje neaktivnih objekata 39
Slika 33: Postavljanje broja objekata i dodavanje platforme 40
Slika 34: Dohvaćanje veličina platformi korištenjem „Box Collider 2D“ komponente 40

Slika 35: Izračun udaljenosti za koju je potrebno pomaknuti generator platformi 41
Slika 36: Postavljanje pozicije ponovo iskorištenog objekta i njegovo aktiviranje 41
Slika 37: Pomicanje generatora platformi 41
Slika 38: Programski kod skripte za deaktivaciju platformi 43
Slika 39: Promjena pozicije igrača s obzirom na y os 44
Slika 40: Postavljanje vrijednosti varijable „igracPrizemljen“ 45
Slika 41: Ostvareni prijelazi između animacija 46
Slika 42: Modificiranje „Transform“ komponente kako bi stvorili razmak 48
Slika 43: Postavljanje nasumične vrijednosti visine 48
Slika 44: Postavljene vrijednosti razmaka i visine 48
Slika 45: Smještaj i postavke materijala „Pad“ 50
Slika 46: Programski kod metode za stvaranje novčića 53
Slika 47: Programski kod metode za deaktivaciju novčića 54
Slika 48: Slika pozadine sa sjenčanjem (eng. Shader) 55
Slika 49: Kod za pomicanje pozadinske slike 56
Slika 50: Pokretanje zvuka skakanja igrača pomoću „Play()“ metode 56
Slika 51: Programski kod skripte „SakupljacNovcica“ 57
Slika 52: Spremanje najboljeg rezultata pod postavke igrača 59
Slika 53: Dohvaćanje vrijednosti najboljeg rezultata 59
Slika 54: Postavljanje vrijednosti rezultata 59
Slika 55: Postavke rezultata 59
Slika 56: Povećanje ukupnog rezultata dodavanjem bodova sakupljanja novčića 60
Slika 57: Uvjet povećanja brzine kretanja igrača 61
Slika 58: Postavljanje vrijednosti varijabli ubrzanja unutar Unity okruženja 61
Slika 59: Deaktivirani objekti zaslona za kraj igre 62
Slika 60: Izgled zaslona kraja igre 62
Slika 61: Varijable skripte „UpravljacIgrom“ 63
Slika 62: Dohvaćanje pozicija potrebnih objekata te deaktivacija zaslona kraja igre 63
Slika 63: Metoda za zaustavljanje tj. kraj igre 64
Slika 64: Programski kod metode „PonovoIgraj()“ 65

Popis tablica

Tablica 1: Usporedba troškova razvoja igara [5] 9
Tablica 2: Usporedba nekih od najpopularnijih 3D programa [14] 14
Tablica 3: Neke prednosti i nedostaci „motion capture“ tehnologije [24] 19

Prilozi

[1] Poveznica za preuzimanje računalne igre i modifikacije za igru:
https://drive.google.com/file/d/1gyv9ar4HMLNdSl8V4VKglu-zjC6SnBHQ/view?usp=sharing