УНИВЕРЗИТЕТ У НОВОМ САДУ
ПРИРОДНО-МАТЕМАТИЧКИ ФАКУЛТЕТ
ДЕПАРТМАН ЗА МАТЕМАТИКУ И ИНФОРМАТИКУ

Santa Run

Projekat iz predmeta Unity - razvoj kompjuterskih igara

Ime i prezime: Tamara Gogić

Sadržaj

1. Uvod

U ovom radu će biti opisana implementacija 3D video-igre “Santa Run” koja je razvijena u višeplatformskom okruženju za razvoj video-igara Unity. Korišćena je Unity verzija 2019.4.6f1. Odabrana platforma za video-igru je Windows. Za implementaciju je korišćen programski jezik C#. Pored implementacije biće opisani osnovni koncepti i način na koji funkcioniše Unity.

2. Unity

2.1 Šta je Unity i čemu služi

Unity je višeplatformsko okruženje za razvoj video-igara za PC, mobilne uredaje, igračke konzole i veb-sajtove sa mogućnošću izvršavanja na preko 20 podržanih platformi. Razvijan je od strane kompanije Unity Technologies od 2004. godine. Unity predstavlja potpuno opremljen alat za razvoj video-igara i to je jedan od razloga njegove velike popularnosti. Unity je vodeće globalno okruženje za razvoj video-igara. Od programskih jezika Unity podržava C#, JavaScript i Boo. Za pisanje koda se standardno koristi Monodevelop ali je moguće korišćenje bilo kog drugog okruženja, npr. Microsoft Visual Studio.

2.2 Lokalni i globalni prostor

Svaka tačka 3D prostora ima tri koordinate - X, Y, i Z, koje se nazivaju i Dekartove koordinate. X je horizontalna, Y vertikalna, a Z komponenta dubine. Ako se tačka nalazi na poziciji (2, 6, 1) to znači da je njena X koordinata jednaka 2, Y koordinata jednaka 6, a Z koordinata jednaka 1. U 3D prostoru uvek postoji tačka koja označava koordinatni početak (0, 0, 0) i pozicija svih objekata globalnog prostora se određuje u odnosu na ovu tačku. Međutim, postoji i lokalni prostor ili prostor objekta koji služi za definisanje pozicije jednog objekta u odnosu na neki drugi objekat. Ova veza među objektima je poznata i kao roditelj-dete veza. Ovakva veza u Unity-u se lako uspostavlja povlačenjem jednog objekta na drugi. Ukoliko je objekat dete na istoj poziciji kao i roditelj njegova pozicija je (0, 0, 0) iako globalna pozicija roditelja ne mora biti nula pozicija. Na osnovu ovakve postavke razdaljina između objekata se računa u lokalnom prostoru gde za svaki dete objekat roditelj objekat uvek ima nula poziciju.

2.3 Vektori

U razvoju igara često se koriste vektori. 3D vektori se opisuju preko Dekartovih koordinata. Predstavljaju veličine koje imaju pravac, smer i intenzitet. Mogu da se pomeraju u 3D prostoru ali se time ne menjaju. Korisni su pri računanju razdaljina, relativnih uglova između objekata, itd. Unity koristi strukturu Vector3 za reprezentaciju vektora ali i tačaka.

2.4 Kamere

Kamere su od suštinske važnosti u 3D prostoru i video-igrama jer se ponašaju kao okviri za prikaz i direktno određuju šta igrač vidi na svom ekranu. Kamere mogu da se postave na bilo kojoj poziciji u prostoru, mogu da se animiraju ili prikače za neki pokretni objekat u igri. Na sceni može biti postavljen veći broj kamera ali u svakom trenutku je samo jedna glavna - Main Camera, koja renderuje ono što igrač vidi. Ovo je razlog zašto Unity automatski dodaje Main Camera objekat pri kreiranju novog projekta ili scene. Postojanje više kamera na istoj sceni igraču pruža mogućnost prikaza nekog drugog dela okruženja ili prikaz okruženja iz nekog drugog ugla kada za tako nešto postoji potreba.

2.5 Temena, Ivice, Poligoni, Mesh-evi

Složeni 3D objekti ne postoje kao takvi već se dobijaju na osnovu jednostavnih 2D poligona koji su međusobno povezani i zajedno čine tzv. **Mesh**. Složeni modeli konstruišu se uz pomoć softvera za 3D

modelovanje i kao takvi se mogu jednostavno uvući u Unity projekat nakon čega je rad sa njima isti kao i sa bilo kojim drugim objektom u Unity-u. Pri uvozu modela iz nekog softvera za modelovanje Unity konvertuje sve poligone u trouglove. Ovako dobijeni trouglovi predstavljaju strane i sastoje se od tri povezane ivice. Mesta spajanja ivica su temena. Uz pomoć ovih podataka Game Engine može da vrši razna izračunavanja vezana za objekat poput onih koja se odnose na određivanje mesta udarca ili kolizije dva objekta. Na osnovu Mesh podataka može se odrediti vizuelno približno isti oblik objekta ali jednostavniji i sa manje detalja koji je pogodniji za izračunavanja. Ovaj pristup doprinosi boljim performansama. Što je više poligona koji čine objekat to objekat ima više detalja i bolji, grafički realističniji izgled ali to vuče intenzivnija izračunavanja i zahteva jače mašine koje bi to podržale.

2.6 Materijali, shader-i, teksture

Materijali su zajednički koncept svih 3D aplikacija i grafike uopšte jer obezbeđuju vizuelni izgled modela. Materijal predstavlja definiciju kako površina treba biti renderovana uključujući reference tekstura koje su korišćene, boja, i drugo. Podešavanjem materijala mogu se dobiti razni efekti od obične boje do reflektujuće površine i ovi efekti se dodeljuju objektima postavljanjem odgovarajućeg materijala. Materijal određuje koji će shader koristiti, a shader definiše koje će opcije biti dostupne za detaljno podešavanje tog materijala.

Shader je mala skripta zadužena za stil renderovanja. Shader sadrži matematičke kalkulacije i algoritme za određivanje boje svakog renderovanog piksela na osnovu osvetljenja i konfiguracije materijala objekta. Npr., kod reflektujućeg shadera materijal će renderovati refleksije objekata okruženja, a pritom će zadržati i prethodno dodatu boju ili teksturu.

Teksture su bitmap slike koje možemo nalepiti na objekat u cilju promene njegove vizuelne pojave. Korišćenjem tekstura se imitira izgled objekata iz stvarnog sveta. Materijal može imati referencu teksture i tada se shader materijala uzima u obzir pri kalkulaciji boje površine objekta. Sem boje, tekstura može predstavljati i druge aspekte površine materijala kao što su na primer refleksija i hrapavost.

Korišćenje materijala u Unity-u je lako. Materijali koji su razvijeni van Unity-a mogu da se uvezu u Unity i koriste u bilo kojem projektu. Takođe, Unity omogućava i kreiranje materijala od početka.

Kreiranje tekstura moguće je u svakom programu za obradu slika poput Photoshop-a ili GIMP-a. Ono o čemu treba voditi računa prilikom kreiranja tekstura je rezolucija slika. Veća rezolucija znači više detalja ali i sporije renderovanje. Teksture u Unity-u se uvek skaliraju na stepen broja 2, npr. 64x64, 128x128, itd.

2.7 Podsistem za izračunavanje fizike

Podsistem za fiziku je najvažniji podsistem u okruženju Unity. Unity koristi podsistem za izračunavanje fizike Box2D u 2D-u i podsistem za izračunavanje fizike NVIDIA PhysX 3.3 u 3D-u. NVIDIA PhysX je moćan podsistem za izračunavanje fizike, omogućava realnu fiziku i olakšava posao programerima nuđenjem osnovnih komponenti mehanike. Fizički pokret simulira ponašanje objekata kako reaguju jedni na druge kad se primeni sila na njih. Sile mogu biti konstantne poput gravitacije ili moment vozila, dok su druge kratke i moćne poput eksplozija. Osnovna komponenta fizike se naziva Rigidbody. Ona omogućuje fizičko ponašanje dodeljenom objektu.

2.8 Detekcija kolizije

Kako je u igrama često potrebno ispitati da li je došlo do kolizije dva objekta Unity ima svoj pristup u rešavanju ovog problema. Oko posmatranih objekata formira se nevidljiva mreža uz pomoć **Collider** komponente koja imitira formu i oblik objekata i na osnovu nje se vrše izračunavanja koja vode ka određivanju kolizije. U Unity-u su na raspolaganju dva tipa Collider-a - Primitive Collider i Mesh Collider.

Primitivni oblici su jednostavni 3D oblici poput kvadra, sfere i kapsule. **Primitive Collider** koji predstavlja kvadar zadržava ovu formu uprkos stvarnom obliku samog objekta kojem pridružujemo ovu komponentu. Primitive Collider-i se koriste kada preciznost nije na prvom mestu, jer su jednostavni i efikasni u smislu izračunavanja.

Mesh Collider se zaniva na 3D Mesh-u koji čini objekat. Što je ovaj Mesh kompleksniji to je Collider precizniji, sa više detalja, ali uz ovakva podešavanja treba očekivati slabije performanse. Zlatna sredina je korišćenje Mesh Collider-a koji prilično verno oslikava glavni objekat ali ipak sa smanjenim nivoom detalja. Ovakav Collider ne pruža potpunu preciznost ali je po ovom pitanju uvek bolji od Primitive Collider-a dok zadržava visoke performanse.

2.9 Osnovni koncepti Unity tehnologije

Prvi osnovni koncept Unity-a je **GameObject**. Korišćenjem Game objekata igra se može podeliti na delove kojima se lako upravlja i koji se jednostavno povezuju i podešavaju. Game objekti se sastoje od komponenti. **Komponente** su zapravo funkcionalnosti koje objekat može da ima. Svaki objekat može imati skoro beskonačno komponenti i dakle, isto toliko funkcionalnosti i osobina. Komponente imaju **promenjive**. To su osobine ili podešavanja koja datu komponentu kontrolišu. Podešavanjem ovih promenljivih dobijamo potpunu kontrolu nad efektima koje data komponenta ima nad objektom.

Sledeći Unity koncept su **Asset**-i. To su svi oni delovi ili blokovi koji grade projekat i objekte - zvukovi, slike, teksture, materijali, animacije, modeli, ali i nadogradnje Unity-a u vidu ekstenzija, dodataka i skripti. Sve to predstavlja asset-e. Upravo iz tog razloga su svi fajlovi koji se koriste u projektu zapamćeni, standardno, u folderu Assets.

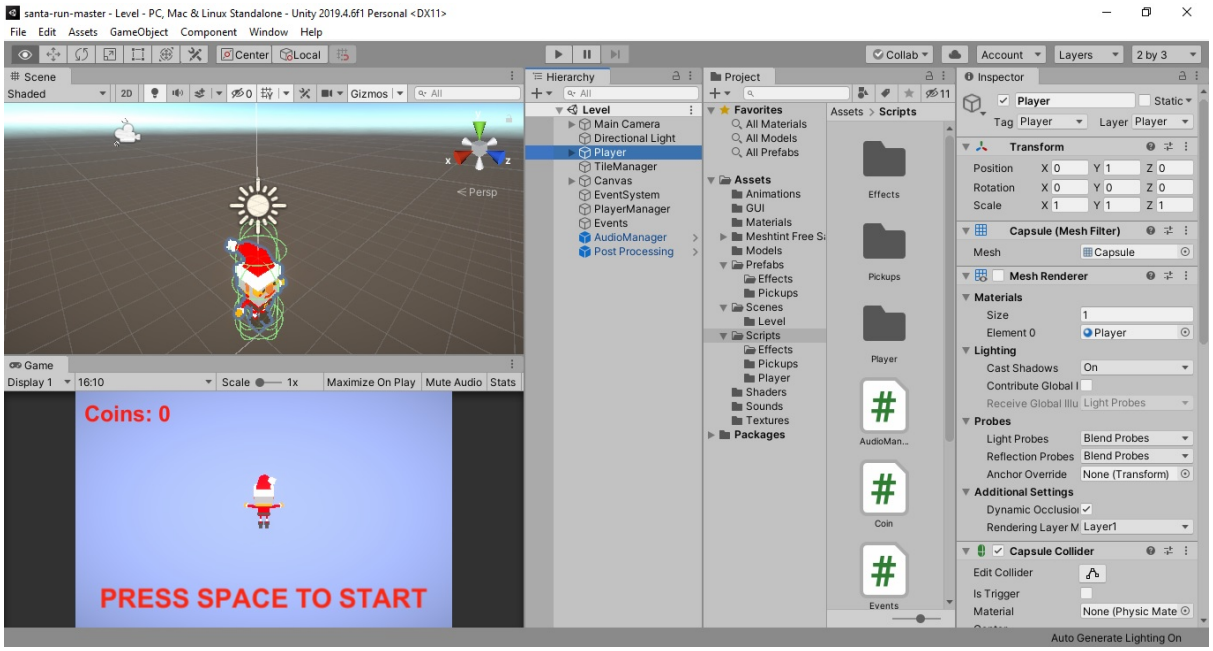
Skripte su sledeća bitna stavka. Nakon dodavanja objekata na scenu i komponenti koje određuju njihov izgled, poziciju i druge osobine, potrebno je ugraditi logiku čime se određuje i njihova uloga, tj. funkcija i ponašanje, što je i poenta celog razvoja igara. Ovo se postiže pisanjem skripti korišćenjem odabranog programskog jezika i njihovim pridruživanjem objektima u vidu komponenti.

Unity omogućava pamćenje kreiranog objekta u vidu prefabrikovanog modela sa svim komponentama, osobinama i logikom koje ga čine onim što jeste. Ovo je koncept Unity-a koji se naziva **Prefabs**, a ovakvi objekti su u potpunosti spremni za kasnije korišćenje u istom projektu ili u drugim projektima jednostavnim uvozom.

Tags, ili tagovi, predstavljaju način identifikovanja objekata u Unity-u. Jedan od načina identifikovanja objekta je korišćenje imena tog objekta. U pojedinim situacijama korisno je imati načina za zajedničko identifikovanje sličnih ili istih objekata.

Layers, ili slojevi, čine sledeći koncept. Layer-i ukazuju na neke funkcionalnosti koje su zajedničke različitim, pa i nesrodnim, objektima. Na primer, slojevi mogu da ukazuju na to koji će objekti biti iscrtani, ili koji će biti sakriveni, ili na koje će moći da se puca, ili jednostavno koji će objekti imati neku specifičnu osobinu.

3. Unity korisničko okruženje



Slika 3.1 – Unity korisničko okruženje

Na slici 3.1 prikazano je kako izgleda Unity okruženje.

3.1 Layout

Raspored komponenti razvojnog okruženja može da se podešava odabirom odgovarajućeg **Layout**-a. Osnovna predefinisana Layout podešavanja nalaze se u gornjem desnom uglu. Klikom na ovu opciju dobija se padajući meni gde se može odabrati neki od postojećih, kreirati novi, ili izvršiti brisanje prethodno kreiranog Layout-a. Layout 2 by 3 se može videti na slici 3.1.

3.2 Scene View

Scene View je mesto gde se razvija vizuelni aspekt scene. Korišćenjem Scene View-a dobijamo pogled u 3D svet video-igre. Na scenu se mogu prevlačiti elementi igre nakon čega im se mogu menjati pozicija, veličina i rotacija i to pomoću strelica oko obeleženog objekta. Kroz Scene View na jednostavan način se može organizovati izgled čitavog nivoa igre. Precizna podešavanja i manipulacija sa objektima scene može se vršiti uz pomoć Inspector-a koji se takođe pojavljuje obeležavanjem objekata bilo iz Scene ili Hierarchy View panela. U gornjem desnom uglu Scene View-a je tzv. Gizmo pomoću kojeg se menja pogled na scenu po X, Y, ili Z osi. Klik po sredini Gizma vodi u perspektivni pogled. Scene View sadrži i polje za pretragu koja može da se vrši po tipu ili nazivu objekta. U slučaju kada ima puno postavljenih objekata na sceni ovo može biti od velike koristi.

3.3 Game View

Game View je komponenta gde se vrši testiranje aplikacije pre krajnjeg bildovanja za neku platformu. Za razliku od Scene View-a gde postoji mogućnost proizvoljnog kretanja po sceni, zumiranja, manipulacije objekima, ovde tih mogućnosti nema i svet igre se posmatra samo kroz prethodno definisani objekat kamere. Dugme koje se ovde često koristi je Maximize on Play i ono omogućava pokretanje aplikacije preko celog ekrana. Tu je i opcija isključivanja zvuka (Mute Audio), Aspect Ratio podešavanje u gornjem levom uglu, ali i dugme Stats koje prikazuje statistiku igre koja se odnosi na zvuk, grafiku (broj temena, trouglova, rezolucija, senke, animacije, fps, i sl.) i zauzeće resursa.

3.4 Hierarchy View

Hierarchy View prikazuje sve objekte koji se trenutno nalaze na aktivnoj sceni bilo da su oni uključeni ili isključeni, vidljivi ili sakriveni. To mogu biti objekti okoline, karaktera, izvori svetlosti, kamere, i drugi. Game objekti mogu da se dodaju i dinamički iz koda. U tom slučaju su prikazani kroz Hierarchy View jedino u aktivnom stanju u toku igre. Uklanjanje objekata sa scene je tako]e moguće iz programskog koda. Dupli klik na objekat iz Hierarchy View-a fokusira i zumira Scene View na dati objekat. Isto se postiže označavanjem objekta pomoću tastature ili miša i pritiskom tastera F na tastaturi. Dakle, Hierarchy View omogućava detaljniji i lakši prikaz i pristup objektima scene, njihovu hijerarhijsku organizaciju, kao i kreiranje novih objekata iz kontekstnog menija koji se dobija pomoću desnog tastera miša.

3.5 Project View

Project View je deo Unity okruženja i služi za prikaz i rad sa folderima i fajlovima projekta. To su fajlovi koje je korisnik kreirao ili uvezao u projekat - scene, skripte, modeli, teksture, zvuci, itd. Project View zapravo predstavlja reprezentaciju stvarnog foldera projekta koji se nalazi na određenoj lokaciji na hard disku. Modifikacijom ili brisanjem fajlova iz Project View panela Unity ažurira veze sa istim, pa ukoliko su prethodne promene dovele do neke greške to se relativno lako i brzo može korigovati. U suprotnom, ukoliko se vrše promene foldera i fajlova iz Windows Explorer-a ili drugog File Manager-a, kidaju se prethodno uspostavljene veze projekta sa fajlovima što dovodi do pojave većih grešaka, pucanja scene, ili neučitavanja projekta. Project View ima dve layout opcije, tj. dve mogućnosti prikaza fajlova, kroz jednu ili dve kolone. Na korisniku je da izabere željeni izgled, mada se prikaz kroz dve kolone uglavnom pokazao kao pregledniji i efektivniji. Prva kolona u tom slučaju služi za odabir foldera, a druga za prikaz fajlova u odabranom folderu i njihovu modifikaciju.

3.6 Inspector

Inspector je bitna, vrlo korisna, i verovatno najkorišćenija komponenta Unity okruženja. Koristi se za pristup svim podešavanjima Unity-a, kako onima koja se odnose na samo razvojno okruženje ili učitani projekat, tako i podešavanjima karakteristika konkretnih objekata scene i njihovih komponenti. Različiti objekti na sceni mogu imati različite karakteristike i podešavanja. Osobine koje se mogu videti u Inspector-u objekta glavne kamere, neće se naći kod, npr. objekta koji predstavlja izvor svetlosti. Svaka komponenta ili

grupa podešavanja objekta u Inspector-u može da se isključi ili uključi uz pomoć check-box-a koji se nalazi pored naziva te komponente. Dobra osobina Unity-a je što Scene View automatski registruje svako ažuriranje podešavanja u Inspector-u i svaku promenu vrednosti atributa komponenti objekata tako da je rezultat promena odmah vidljiv.

3.7 Toolbar

Ispod linije menija nalazi se linija sa alatima, tzv. **Toolbar.** Sačinjen iz nekoliko različitih grupa kontrola koje se odnose na Scene View i manipulaciju sa scenom, ali pored toga sadrži i kontrole za pokretanje igre u editoru. Prva grupa kontrola sastoji se iz alata za navigaciju scenom i transformaciju objekata. Nakon prve grupe alata slede kontrole Center/Pivot i Global/Local koje služe za podešavanje koordinatog sistema i centra u odnosu na koji će se vršiti promene pomoću prethodno pomenutih alata. U centralnom delu se nalaze kontrole za pokretanje i pauziranje igre u toku testiranja. Poslednje dugme u ovom nizu koristi se za bolju kontrolu toga šta se dešava u igri jer pruža mogućnost frame-by-frame prikaza. Slede opcije za podešavanje naloga, Layer-a, i Layout-a. Layer-i se između ostalog koriste radi kontrole kojim kamerama će objekti biti renderovani ili kojim će svetlima biti osvetljeni. Različiti Layer-i mogu imati različite funkcije.

3.8 Meniji

Ostale opcije i funkcionalnosti su raspoređene u padajućim listama glavne linije menija. To su: File, Edit, Assets, GameObject, Component, Window i Help.

Iz **File** menija možemo učitati, kreirati i pamtiti scene i projekte. Bitna podešavanja ovog menija su podešavanja vezana za odabir platforme i Build aplikacije sa dodatnim opcijama u Inspector-u.

Edit meni sadrži očekivane Undo, Redo, Cut, Copy, Paste, Duplicate i Delete komande, kao i nekoliko runtime komandi. Tu su prethodno pomenute Play, Pause i Step opcije, opšta podešavanja - Preferences, i podešavanja projekta - Project Settings.

Asset-i predstavljaju blokove koji grade svaki Unity projekat. Teksture, slike, 3D modeli, zvuci, sve su to asset-i. Podmeni koji se najviše koristi je Create i on služi za kreiranje asset-a, a sem navedenih tu su i skripte, materijali, animacije, fontovi, itd. Ovaj meni služi i za kreiranje novih foldera. Grupisanje fajlova u odgovarajuće foldere puno olakšava organizaciju projekta. Create meni je dostupan i iz kontekstnog menija Project View panela. Veoma korisna mogućnost Unity-a je jednostavan uvoz asset-a iz drugih softvera, poput 3D Studio Max-a, ali i preuzimanje gotovih asset-a iz Unity Asset prodavnice.

Osnovna jedinica građe i funkcije Unity aplikacija jeste **Game Object**. Game objekat može biti prazan objekat, kontejner za druge objekte, skripte, a može biti i kompletni složeni model koji imitira stvarni objekat iz realnog sveta. Game objekat se sastoji iz komponenti. Osnovne komponente svih Game objekata se odnose na lokaciju, rotaciju i veličinu i to su zajedničke osobine svih vizuelnih objekata scene. Između Game objekata može da postoji veza roditelj-dete. Osnovni Unity Game objekti su ravan, kvadar, sfera, cilindar, ali sem ovih pod Game objekte spadaju i sistemi čestica, kamere, svetla, i dr. Iz menija GameObject kreiraju se novi objekti i postavljaju na scenu.

Component meni daje pristup komponentama koje određuju objekat - Collider-i, Mesh-evi, skripte, efekti, itd. Komponente mogu određivati izgled, ponašanje, kao i druge funkcionalnosti i svojstva koje objekat može da ima. Komponente se mogu kreirati, a ukoliko su već gotove mogu se samo prevući na željeni objekat čime će automatski biti dodate. Sve komponente koje su pridružene objektu mogu se menjati i prilagođavati promenom njihovih atributa u Inspector-u. Sem ovih standardnih komponenti koje se vezuju za objekte tu su i UI komponente koje se često koriste u kreiranju menija ili bilo kakvog prikaza dugmića, slika i teksta u 2D ili 3D apikacijama.

Window meni prikazuje komande sa odgovarajućim prečicama na tastaturi za uređenje samih prozora i editora Unity okruženja. Takođe, ovaj meni pruža pristup oficijalnoj Unity Asset prodavnici gde se mogu pronaći razni gotovi asset-i, Unity dodaci, ekstenzije. Mnogi od njih su besplatni, ali se neki, pogotovo oni kvalitetniji, naplaćuju.

Help je meni za pomoć. Tu su Unity uputstva, zatim Unity Answers sa odgovorima na česta pitanja kao i Unity Forum sa velikom podrškom Unity programera.

4. Razvoj 3D video-igre “Santa-Run”

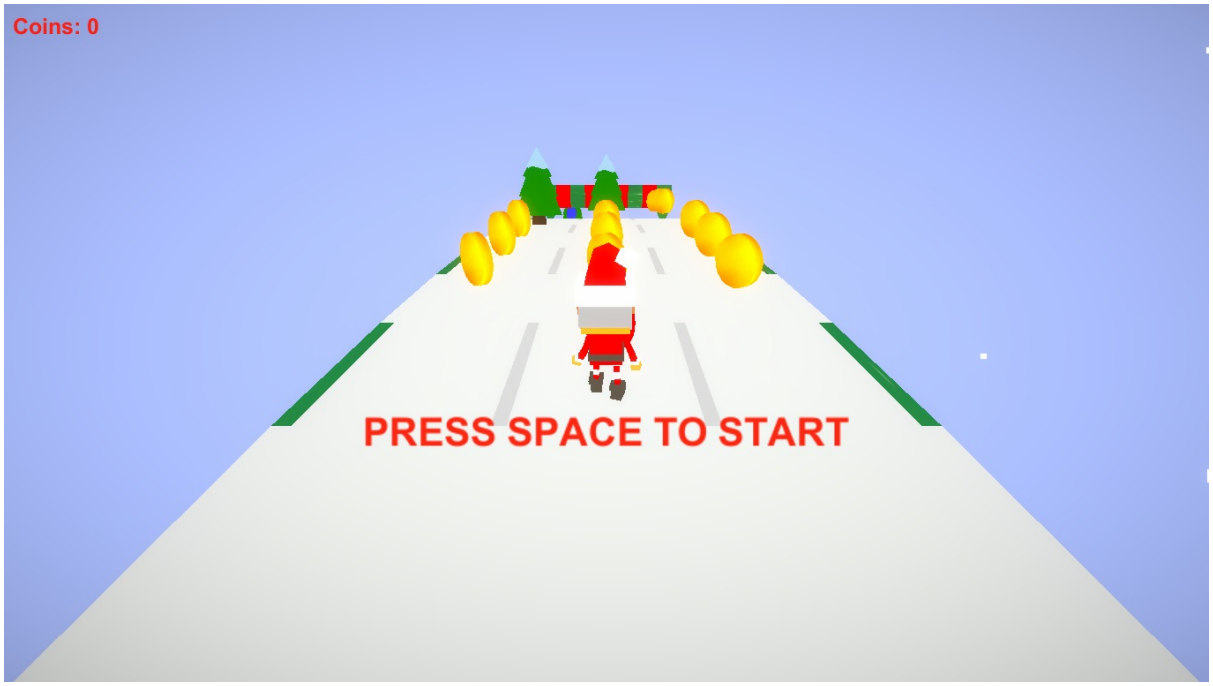
U ovom odeljku će naveća pažnja biti posvećena opisu implementacije video-igre kao i funkcionalnosti od kojih se sastoji, dok će ostali aspekti poput animacija, tekstura, modela i slično biti pomenuti u kratkim odeljcima.

4.1 Opis video-igre “Santa Run”

Ideja je da se Deda Mraz kreće niz snežnu padinu izbegavajući prepreke poput drveća, rampi i slično. Biće tri vrste prepreka koje može zaobići, preskočiti ili se podvući ispod njih. Usput može skupljati novčiće. Kako vreme odmiče, brzina Deda Mraza će se povećavati, pa samim tim i težina igrice. Kada Deda Mraz ne uspe da izbegne prepreku, igrica se završava i prikazuje se postignuti skor koji se računa na osnovu prikupljenih novčića. Pored novčića, Deda Mraz će moći da pokupi određene vrste pomoći poput smanjenja brzine u trajanju od 4 sekunde, zatim štit koji će obezbediti da može prolaziti kroz prepreke u trajanju od 5 sekundi i magnet u trajanju od 5 sekundi uz pomoć kog će Deda Mraz moći da privlači ka sebi novčiće u svojoj neposrednoj blizini.

Pokretanjem igre prikazuje se glavni meni (Slika 4.1.1) koji sadrzi dva dugmeta: “Play” i “Quit”. Klikom na “Quit” se zatvara igra, a klikom na “Play” prikazuje se scena (Slika 4.1.2) iz koje je moguće započeti igru pritiskom tastera “space” na tastaturi.

Slika 4.1.1 – Glavni meni



Slika 4.1.2 – Početak igre

Kontrola glavnog lika igre na Windows platformi vršiće se preko tastature. Strelice levo i desno koristiće se za skretanje, strelica na gore za skok, a strelica na dole za podvlačenje ispod prepreke. Kraj igre se dešava ukoliko Deda Mraz ne uspe da izbegne neku od prepreka. U tom slučaju će se na ekranu prikazati poruka “Game Over”, konačan broj ostvarenih novčića za vreme igre, kao i dva dugmeta: “Replay” i “Quit” (Slika 4.1.3). Klikom na dugme “Replay” se može započeti igra ispočetka.

Slika 4.1.3 – Game Over ekran

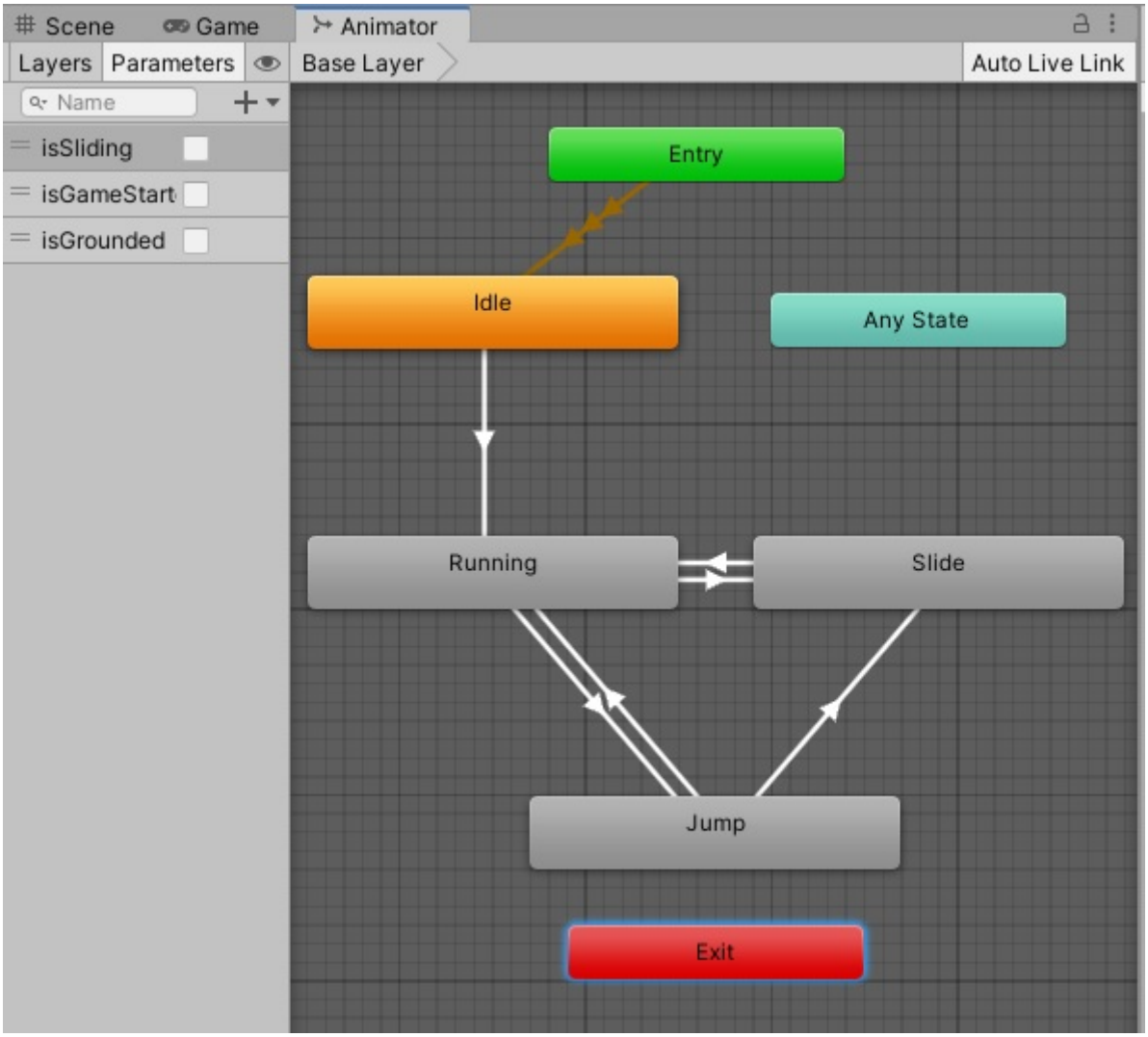
4.2 Modeli, teksture i animacije

Za potrebe ove video-igre većinom su korišćeni gotovi modeli koje nudi prodavnica Unity Asset Store. Jedino je model za novčić napravljen ručno u Unity okruženju. Isti model se koristi za već nabrojane vrste pomoći. Za efekat usporavanja narandžasti, za efekat magneta crveni i za efekat štita plavi novčić.

Kada je reč o korišćenim teksturama, neke od njih su uvezene u sklopu paketa "Meshtint Free Santa Claus", dok su ostale ručno pravljene u alatu Paint.

Takođe su korišćene i gotove animacije dostupne na sajtu https://www.mixamo.com/#/. Korišćene su animacije za trčanje, skok, slide i idle. Sve se odnose na animiranje Deda Mraza. U igri postoji još jedna animacija koja je ručno pravljena u Unity okruženju i odnosi se na tekst "PRESS SPACE TO START".

Na slici 4.2.1 prikazan je izgled animatora za Deda Mraza koji služi za kontrolu animacija. Iz animatora se može videti da je na početku igrice Deda Mraz u neaktivnom stanju (animacija Idle). Tranzicije predstavljaju usmerene veze između animacija. Ukoliko postoji tranzicija iz jedne animacije u drugu to znači da je moguće iz jedne animacije preći u drugu. Da li će se ti prelazi desiti zavisi od zadatih parametara za tranzicije. Recimo, iz stanja Idle moguće je preći u stanje Running kada je vrednost parametra isGameStarted = true što se i dešava kada korisnik pritisne taster space na tastaturi i tako pokrene igru. Slično, iz stanja Running se može preći u stanje Slide ukoliko je vrednost parametra isSliding = true, a u stanje Jump ukoliko je vrednost parametra isGrounded = false. Iz stanja Slide se Deda Mraz vraća u stanje Running ukoliko je vrednost parametra isSliding = false. Isto važi i za stanje Jump kada je isGrounded = true. Iz stanja Jump se takođe može preći u stanje Slide ukoliko je isSliding = true.



Slika 4.2.1 - Animator

4.3 Zvukovi

U ovoj video-igri korišćena su četiri zvuka koji su preuzeti sa sajta https://freesound.org/. MainTheme je glavni zvuk koji je aktivan dok je igrica u toku. PickUpCoin se aktivira svaki put kada Deda Mraz pokupi novčić. GameOver zvuk se javlja kada Deda Mraz udari u neku od prepreka i ButtonClick kada se klikne na neki od dugmića “Play”, “Replay” ili “Quit”.

U Project View panelu najpre se kreira novi folder Sounds, i ovde se uvoze preuzeti mp3 fajlovi. U hijerarhijskom panelu kreira se novi prazan objekat (Empty Object) koji se naziva AudioManager, koji neće biti vizuelno predstavljen na sceni. Zatim se u folderu Script kreiraju skripte Sound.cs i AudioManager.cs. Skripta Sound.cs je prikazana na slici 4.3.1 i služi za podešavanje određenih parametara iz Inspector panela u Unity okruženju, poput naziva i jačine zvuka, zatim postoji i parametar loop tipa bool od kog zavisi da li će se zvuk ponavljati ili ne, dok će se u polje clip prevući odabrani zvuk iz foldera Sounds. Peta linija koda u ovoj skripti “**[System.Serializable]**” će omogućiti da ovi parametri budu vidljivi u Inspector panelu.

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

[System.Serializable]
public class Sound
{
    public string name;
    public AudioClip clip;

    public float volume;

    public bool loop;

    public AudioSource source;
}
```

Slika 4.3.1 – Skripta Sound.cs

Skripta AudioManager.cs prikazana je na slici 4.3.2. Ova skripta će biti zakačena za AudioManager prazan objekat. U klasi AudioManager je prvobitno definisan niz koji će sadržati sve zvukove koji se koriste u igri. U Start metodi se kroz foreach petlju svakom zvuku pridružuje AudioSource komponenta. Ova skripta sadrži i metodu PlaySound koja služi za puštanje zvuka. Ova metoda kao argument uzima naziv zvuka. Implementirana je tako da se na osnovu naziva zvuka pronađe taj zvuk u nizu zvukova i nad njim pozove ugrađeni metod Play(). PlaySound metod se u okviru ove skripte poziva u Start metodi i time se pokreće zvuk MainTheme koji je aktivan od početka igre do momenta kada se desi GameOver. Ovaj metod se može pozivati i iz drugih skripti. Recimo GameOver zvuk se aktivira kada dođe do kolizije između Deda Mraza i prepreke. To je regulisano tako što se u okviru metode OnTriggerEnter, koja se nalazi u skripti PlayerController.cs, poziva ovaj metod. Pristupa mu se na ovaj način: **FindObjectOfType<AudioManager>().PlaySound("GameOver");**. Slično, PickUpCoin zvuk će se aktivirati iz skripte CoinEffectConfig.cs, dok će se ButtonClick pozivati na drugačiji način. Naime, Button komponenta u Inspector panelu sadrži metod OnClick u kom se može definisati između ostalog i željeni zvuk.

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

public class AudioManager : MonoBehaviour
{
    public Sound[] sounds;
    // Start is called before the first frame update
    void Start()
    {
        foreach (Sound s in sounds)
        {
            s.source = gameObject.AddComponent<AudioSource>();
            s.source.clip = s.clip;
            s.source.loop = s.loop;
        }
        PlaySound("MainTheme");
    }

    public void PlaySound(string name)
    {
        foreach (Sound s in sounds)
        {
            if (s.name == name)
                s.source.Play();
        }
    }
}
```

Slika 4.3.2 – Skripta AudioManager.cs

4.4 Kontrola kretanja igrača

Kretanje Deda Mraza je implementirano u skripti PlayerController.cs. Ova skripta je zakačena kao komponenta na objekat Player u Unity okruženju. Objekat Player je ustvari Capsule materijal koji će služiti i kao kolajder, dok je model Deda Mraza zakačen na njega i oni su u odnosu roditelj-dete. Da bi se moglo kontrolisati kretanje igrača u Inspector panelu se objektu Player dodaje komponenta Character Controller. Referenca na tu komponentu postoji u skripti pod nazivom controller i pristupiće joj se uz pomoć ugrađenog metoda GetComponent.

4.4.1 Kretanje igrača unapred

Pre svega je potrebno definisati određene parametre koji će pomoći u implementaciji kretanja igrača unapred. To su: direction (smer u kom će se igrač kretati), forwardSpeed (brzina kojom će se kretati) i maxSpeed (maksimalna brzina). Ovo će se regulisati u Update metodi. Potrebno je pomerati igraca po Z osi što će biti obezbeđeno komandom **direction.z = forwardSpeed;**. Parametar forwardSpeed je u Inspector panelu postavljen na početnu vrednost 10, dok je parametru maxSpeed dodeljena vrednost 40. Povećavanje brzine će se obezbediti takođe u Update metodi, a na koji način prikazano je na slici 4.4.1.1.

```
//Increase speed
if (forwardSpeed < maxSpeed)
    forwardSpeed += 0.1f * Time.deltaTime;
```

Slika 4.4.1.1 – Povećavanje brzine

Dakle, sve dok je trenutna brzina manja od maksimalne, trenutna brzina će se povećavati.

4.4.2 Kretanje igrača u levo i u desno

Staza niz koju se spušta Deda Mraz ima tri trake. Parametar desiredLane koristi se za označavanje u kojoj traci se Deda Mraz nalazi. Default vrednost tog parametra je 1, odnosno traka u sredini, vrednost 0 označava levu traku, dok vrednost 2 označava desnu. Parametar laneDistance označava razdaljinu između dve trake i postavljen je na vrednost 2.5. Potrebno je obezbediti i to da Deda Mraz ne može skliznuti van staze. Na slici 4.4.2.1 može se videti programski kod kojim je ovo obezbeđeno. Dakle, ukoliko se pritisne desna strelica na tastaturi, vrednost parametra desiredLane se povećava za 1. Međutim, ako se već igrač nalazi u traci 2, vrednost parametra desiredLane biće vraćena na vrednost 2. Time se obezbeđuje da ukoliko se igrač nalazi u desnoj traci i ukoliko se pritisne strelica na desno, igrač će ostati u desnoj traci. Slično važi i za levu traku.

```
// Gather the inputs on which lane we shoud be
if (Input.GetKeyDown(KeyCode.RightArrow))
{
    desiredLane++;
    if (desiredLane == 3)
        desiredLane = 2;
}

if (Input.GetKeyDown(KeyCode.LeftArrow))
{
    desiredLane--;
    if (desiredLane == -1)
        desiredLane = 0;
}
```

Slika 4.4.2.1 – Kontrola u kojoj traci će se Deda Mraz naći

Sada je potrebno izračunati na kojoj poziciji će se naći igrač pri kretanju levo i desno kao i kako će izgledati pomeranje igrača. Dakle, potrebno je obezbediti prirodno kretanje, da ne bi došlo do efekta teleportovanja igrača sa jedne pozicije na drugu. Implementacija tog dela je prikazana na slici 4.4.2.2. Naime, prvo je potrebno inicijalizovati trenutnu poziciju koja će se čuvati u promenljivoj targetPosition koja je tipa Vector3. Zatim se u if uslovu proverava da li je vrednost parametra desiredLane == 0. Ukoliko jeste, pozicija igrača će se promeniti za vrednost 2.5 (laneDistance) u levo. Zatim se u else if uslovu proverava da li je desiredLane == 2. Ukoliko jeste pozicija igrača će se promeniti za vrednosti 2.5 (laneDistance) u desno. Prirodno kretanje iz jedne trake u drugu obezbeđeno je u drugom delu koda. Dakle, ukoliko igrač još uvek nije na ciljanoj poziciji, računa se za koliko je potrebno da se pomeri (targetDir) i koliki će biti korak pri svakom pomeranju (moveDir) dok ne dostigne ciljanu poziciju (targetPosition). Ideja je da se Deda Mraz ne premesti odjednom u željenu traku već malo po malo da bi se dobio prirodan efekat pomeranja. Sve dok je taj mali korak (moveDir) manji od targetDir, igrač će se pomerati u vrednosti parametra moveDir. Parametar

targetDir se u svakom tom pomeraju smanjuje za po jednu vrednost moveDir. Kada moveDir više nije manja vrednost od targetDir, igrač će dostići ciljanu poziciju pomeranjem u vrednosti od targetDir.

```
// Calculate where we should be in the future

// Initialize to current location
Vector3 targetPosition = transform.position.z * transform.forward + transform.position.y * transform.up;

if (desiredLane == 0)
{
    targetPosition += Vector3.left * laneDistance;
}
else if (desiredLane == 2)
{
    targetPosition += Vector3.right * laneDistance;
}

// Time to swtich lanes
if (transform.position != targetPosition)
{
    Vector3 targetDir = targetPosition - transform.position;
    Vector3 moveDir = targetDir.normalized * laneSwitchSpeed * Time.deltaTime;

    if (moveDir.sqrMagnitude < targetDir.sqrMagnitude)
        controller.Move(moveDir);
    else
        controller.Move(targetDir);
}

controller.Move(direction * Time.deltaTime);
```

Slika 4.4.2.2 – Efekat prirodnog kretanja u levo i u desno

4.4.3 Skok i gravitacija

Na slici 4.4.3.1 prikazan je programski kod kojim se obezbeđuje da Deda Mraz može imati i opciju skakanja. Skok se može realizovati samo ukoliko je igrač već na zemlji što će se proveravati uz pomoć parametra isGrounded koji je tipa bool. Dakle, ukoliko se na tastaturi pritisne strelica na gore proverava se da li je igrač na zemlji. Ukoliko jeste, igrač će skočiti. Skok je implementiran uz pomoć metode Jump. Ideja je pomerati igrača duž Y ose. Pomeraj će biti u vrednosti promenljive jumpForce čija je vrednost postavljena na 10.

```
    if (Input.GetKeyDown(KeyCode.UpArrow))
    {
        if (controller.isGrounded)
        {
            Jump();
        }
    }
}

private void Jump()
{
    direction.y = jumpForce;
}
```

Slika 4.4.3.1 – Jump metoda

Dalje je potrebno obezbediti i da se igrač pri skoku vrati na zemlju. To će se realizovati uz pomoć gravitacije. Uvedena je promenljiva Gravity čija je vrednost postavljena na -20. Implementacija ovog dela se može videti na slici 4.4.3.2. Naime, ukoliko je vrednost parametra isGrounded = false, vrednost Y koordinate za vektor direction će se postepeno smanjivati sve dok se igrač ne nađe na zemlji.

```
if (!controller.isGrounded)
{
    direction.y += Gravity * Time.deltaTime;
}
```

Slika 4.4.3.2 - Gravitacija

4.4.4 Slide (podvlačenje ispod prepreke)

Za dodavanje mogućnosti podvlačenja ispod prepreke koristiće se metoda Slide čija se implementacija može videti na slici 4.4.4.1. Na početku se vrednost parametra isSliding postavlja na vrednost true. Zatim se nad animatorom poziva metoda SetBool kojom se parametru isSliding dodeljuje vrednost bool. Time će se pokrenuti željena animacija za slide. Nije dovoljno samo pokrenuti animaciju slide za Deda Mraza. Potrebno je i promeniti određene parametre za Character controller kako ne bi doslo do kolizije između objekata. Vrednost Y koordinate u vektoru center biće promenjena na vrednost -0.5f, dok će visina kolajdera biti postavljena na vrednost 1. Ovim se postiže smanjenje kolajdera dok je aktivna metoda slide kako bi se uspešno izbegla prepreka. Zatim je potrebno obezbediti koliko vremena će trajati slide. To će se postići korišćenjem metode WaitForSeconds gde je naglašeno da će slide trajati 1.1 sekundi. Nakon isteka tog vremena veličina kolajdera se vraća na početne vrednosti, gasi se animacija za slide i promenljiva isSliding se postavlja na vrednost false.

```
private IEnumerator Slide()
{
    isSliding = true;

    animator.SetBool("isSliding", true);
    controller.center = new Vector3(0, -0.5f, 0);
    controller.height = 1;

    yield return new WaitForSeconds(1.1f);

    controller.center = new Vector3(0, 0, 0);
    controller.height = 2;

    animator.SetBool("isSliding", false);

    isSliding = false;
}
```

Slika 4.4.4.1 – Slide metoda

```
if (Input.GetKeyDown(KeyCode.DownArrow) && !isSliding)
{
    StartCoroutine(Slide());
}
```

Slika 4.4.4.2 – Poziv Slide metode

Poziv metode Slide vrši se u Update metodi što je prikazano na slici 4.4.4.2. Dakle ukoliko korisnik na tastaturi pritisne strelicu na dole i ukoliko Deda Mraz već nije u stanju slide pokrenuće se metod Slide.

4.5. Podešavanja kamere

Pre svega je potrebno napraviti skriptu CameraController.cs (Slika 4.5.1) i zakačiti je na objekat Main Camera. Potrebno je uvesti promenljivu target koja je tipa Transform i odnosi se na igrača, kao i promenljivu offset koja je tipa Vector3 i njome se definiše razdaljina između kamere i igrača. Offset predstavlja razliku između pozicije kamere i pozicije igrača. Potrebno je obezbediti da kamera sve vreme prati igrača. Iz tog razloga će se u Update metodi računati nova pozicija kamere (newPosition). X i Y koordinata će ostati nepromenjene iz razloga što kamera neće pratiti igrača kada se kreće u levo i desno već samo napred, dok će se Z koordinata menjati u zavisnosti od vrednosti promenljive offset.

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

public class CameraController : MonoBehaviour
{
    public Transform target;
    private Vector3 offset;

    void Start()
    {
        offset = transform.position - target.position;
    }

    // Update is called once per frame
    void Update()
    {
        Vector3 newPosition = new Vector3(transform.position.x, transform.position.y, offset.z + target.position.z);
        transform.position = newPosition;
    }
}
```

Slika 4.5.1 – Skripta CameraController.cs

4.6. Prefabrikovani modeli i efekat beskonačne staze

Kreiranje prefabrikovanih modela (Prefabs) u Unity okruženju je prilično jednostavno. Prvo je potrebno napraviti prazan objekat (Empty Object) na koji je potrebno prevući željene modele čija svojstva se mogu podešavati. Na kraju je potrebno željeni model sačuvati u folderu Prefabs. Ovakvi modeli se kao gotovi objekti dalje mogu koristiti. U ovoj video-igri staza je napravljena od 6 prefabrikovanih modela koji predstavljaju delove staze i nastavljaju se jedan na drugi nasumično. Implementacija ovog dela rađena je u skripti TileManager.cs koja je zakačena kao komponenta na objekat Tile Manager. Pre svega je potrebno uvesti potrebne promenljive (Slika 4.6.1).

```
public GameObject[] tilePrefabs;
public float zSpawn = 0;
public float tileLength = 30;
public int numberOfTiles = 5;
private List<GameObject> activeTiles = new List<GameObject>();
public Transform playerTranfsorm;
```

Slika 4.6.1 – Promenljive u skripti TileManager.cs

Promenljiva tilePrefabs predstavlja niz u kom će biti smešteni delovi staza. Pošto je potrebno nadovezivati delove staza samo po Z osi, za pamćenje te vrednosti korišćena je promenljiva zSpawn. Promenljivom tileLength se definiše dužina jednog dela staze. Promenljiva numberOfTiles se koristi za određivanje koliko delova staza će se naći na sceni u svakom trenutku. activeTiles predstavlja listu aktivnih delova staza i playerTransform je referenca na igrača.

```
public void SpawnTile(int tileIndex)
{
    GameObject go = Instantiate(tilePrefabs[tileIndex], transform.forward * zSpawn, transform.rotation);
    activeTiles.Add(go);
    zSpawn += tileLength;
}

private void DeleteTile()
{
    Destroy(activeTiles[0]);
    activeTiles.RemoveAt(0);
}
```

Slika 4.6.2 – Metode SpawnTile i DeleteTile

Na slici 4.6.2 prikazana su dva metoda koja će pomoći u realizaciji beskonačne staze. To su SpawnTile kojim je implementirano nadovezivanje jednog dela staze na drugi i DeleteTile kojim će se omogućiti brisanje dela staze preko kog je već Deda Mraz prešao. Dakle, pošto je nemoguće zaista napraviti beskonačnu stazu, ideja je u zavisnosti od pozicije igrača dodavati po jedan deo staze napred, a istovremeno i brisati već pređeni deo staze tako da u svakom trenutku bude ukupno 5 delova staze na sceni. Na slici 4.6.3 prikazane su Start i Update metode skripte TileManager.cs.

```
void Start()
{
    for (int i = 0; i < numberOfTiles; i++)
    {
        if (i == 0)
            SpawnTile(0);
        else
            SpawnTile(Random.Range(0, tilePrefabs.Length));
    }
}

// Update is called once per frame
void Update()
{
    if (playerTranfsorm.position.z -35 > zSpawn - (numberOfTiles * tileLength))
    {
        SpawnTile(Random.Range(0, tilePrefabs.Length));
        DeleteTile();
    }
}
```

Slika 4.6.3 – Start i Update u skripti TileManager.cs

U okviru Start metode se proverava da li ima već delova staze na sceni. Ukoliko nema unapred odabran deo staze se uvek postavlja kao prvi i na njemu nema prepreka da bi se izbegla kolizija već na samom startu igre. Inače se uzima nasumičan deo i nadovezuje na već postojeći. U Update metodi se proverava pozicija igraca u odnosu na vrednosti promenljivih: zSpawn, numberOfTiles i tileLength. Zatim se poziva metod SpawnTile kojim će se odabrati nasumičan deo staze i nadovezati na poslednji deo staze na sceni. Nakon toga se poziva metod DeleteTile kojim se briše prvi element iz liste activeTiles.

4.7. Detekcija kolizije i “GameOver” ekran

Da bi se napravio “GameOver” ekran pre svega je potrebno napraviti novi objekat Panel u okviru Hierarchy View panela koji će se zvati GameOverPanel. U okviru ovog panela potrebno je napraviti dva dugmeta (“Replay” i “Quit”) kao i Text čiji će sadržaj biti “Game Over”. GameOverPanel će biti neaktivan sve dok se ne desi kolizija između igrača i prepreke kada će se prikazati na ekranu. Ovaj deo je implementiran u skripti PlayerManager.cs (Slika 4.7.1) koja je zakačena za objekat PlayerManager.

```
void Start()
{
    gameOver = false;
    Time.timeScale = 1;
    isGameStarted = false;
    numberOfCoins = 0;
}

// Update is called once per frame
void Update()
{
    if (gameOver)
    {
        Time.timeScale = 0;
        gameOverPanel.SetActive(true);
    }

    coinsText.text = "Coins: " + numberOfCoins;

    if (Input.GetKeyDown(KeyCode.Space))
    {
        isGameStarted = true;
        Destroy(startingText);
    }
}
```

Slika 4.7.1 – Skripta PlayerManager.cs

Dakle, na početku je u Start metodi vrednost promenljive gameOver postavljena na false. U Update metodi se proverava da li je gameOver == true. Ukoliko jeste igra će se zaustaviti i prikazaće se GameOverPanel.

Da bi se označile prepreke u igri napravljen je tag “Obstacle”. U skripti PlayerController.cs je implementirana metoda OnTrigerEnter (Slika 4.7.2). U ovoj metodi se proverava da li je došlo do kolizije između igrača i prepreke (“Obstacle”). Ukoliko jeste promenljiva gameOver se postavlja na vrednost true čime će se aktivirati GameOver ekran. U tom trenutku će se aktivirati i zvuk za GameOver.

```
private void OnTriggerEnter(Collider other)
{
    if (other.transform.tag == "Obstacle")
    {
        PlayerManager.gameOver = true;
        FindObjectOfType<AudioManager>().PlaySound("GameOver");
    }
```

Slika 4.7.2 – Metoda OnTrigerEnter

Sada je potrebno osposobiti dugmiće “Replay” i “Quit”. Kada korisnik klikne na dugme “Replay” igra se započinje ispočetka, a kada klikne na “Quit” igra se gasi. Ovo će biti implementirano u skripti Events.cs (Slika 4.7.3). Metod ReplayGame služi za učitavanje scene “Level”, a metod QuitGame za gašenje igrice.

```
public class Events : MonoBehaviour
{
    public void ReplayGame()
    {
        SceneManager.LoadScene("Level");
    }

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

Slika 4.7.3 – Skripta Events.cs

Potrebno je napraviti novi prazan objekat Events i u njega prevući skriptu Events.cs. Da bi se koristile metode iz ove skripte potrebno je u okviru dugmića ReplayButton i QuitButton dodati ovu skriptu u metod On Click. Zatim za ReplayButton odabrati metod ReplayGame, a za QuitButton metod QuitGame.

4.8. Početak igre

U ovom delu je potrebno obezbediti da kada se pokrene igrica, igrač je neaktivan sve dok korisnik ne pritisne taster space na tastaturi. Pre svega se na scenu dodaje novi tekst sadržine “PRESS SPACE TO START”. Zatim se pravi animacija koja će se odnositi na ovaj tekst. Logika za ovaj deo je implementirana u skripti PlayerManager.cs (Slika 4.7.1). Na početku se promenljiva isGameStarted postavlja na false vrednost. U Update metodi se proverava da li je korisnik pritisnuo taster space. Ukoliko jeste, vrednost promenljive isGameStarted se postavlja na true i uništava se početni tekst. Dodatno je potrebno u PlayerController.cs skripti u okviru metode Update obezbediti da se igrač ne može kretati dok je vrednost promenljive isGameStarted = false (Slika 4.8.1).

```
void Update()
{
    // Check if game should play
    if (!PlayerManager.isGameStarted)
        return;
```

Slika 4.8.1 – Kontrola početka igre

4.9 Kreiranje glavnog menija

Pre svega je potrebno kreirati novu scenu pod nazivom “Menu” u okviru foldera Scenes. Zatim je potrebno podesiti izgled glavnog menija prevlačenjem željenih objekata na scenu kao i kreiranjem dugmića

“Play” i “Quit”. Izgled glavnog menija se može videti na slici 4.1.1. Ponašanje dugmića je određeno uvezenim sprajtovima (sprite) koji se nalaze u folderu GUI. Prirodno su dugmići crvene boje. Ukoliko se pređe mišem preko dugmića oni postaju žuti, a ukoliko se klikne na neki od dugmića postaće zelene boje. Ova podešavanja se mogu obaviti u okviru Inspector panela. Logika koja se krije iza dugmića je implementirana u skripti MainMenu.cs (Slika 4.9.1). Objašnjenje je slično kao za dugmiće na GameOver ekranu na koje će se takođe primeniti pomenuti sprajtovi.

```
public class MainMenu : MonoBehaviour
{
    public void PlayGame()
    {
        SceneManager.LoadScene("Level");
    }

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

Slika 4.9.1 – Skripta MainMenu.cs

4.10 Unapređivanje grafike i Curved Shader

Za unapređivanje grafike korišćen je Universal RP paket. Takođe su dodati i neki od efekata za Post Processing kao što su Bloom, Color Adjustments i Vignette.

Curved Shader je korišćen za efekat iskrivljene staze i već gotov je uvezen u ovaj projekat. Da bi se primenio ovaj efekat potrebno je podesiti ovaj shader za sve materijale, modele i prefabrikovane modele u igri kao i podesiti teksture.

4.11 Efekti

U ovoj igri postojaće četiri vrste efekata. Efekat koji se javlja pri prikupljanju žutih novčića, zatim efekat magneta (crveni novčić), efekat štita (plavi novčić) i efekat usporavanja (narandžasti novčić). U implementaciji efekata će pomoći dve apstraktne klase: AEffectConfig (Slika 4.11.1) i AbstractEffect (Slika 4.11.2). Klasa AEffectConfig nasleđuje ScriptableObject. ScriptableObject služi za skladištenje podataka i uglavnom se koristi radi uštede memorije izbegavanjem kopija vrednosti. Ovo je korisno ukoliko projekat sadrži prefabrikovane modele. Svaki put kada se koristi određeni prefabrikovani model, on će dobiti svoju kopiju tih podataka. Umesto korišćenja metoda i skladištenja dupliranih podataka, može se koristiti ScriptableObject za skladištenje podataka, a zatim mu se može pristupiti na osnovu reference iz svih prefabrikovanih modela. To znači da postoji jedna kopija podataka u memoriji. Klasa AEffectConfig sadrži apstraktnu metodu ApplyEffect koja služi za primenjivanje efekta. Ova metoda će se implementirati u svim klasama koje nasleđuju klasu AEffectConfig i služe za konfiguraciju pojedinačnih efekata.

```
public abstract class AEffectConfig : ScriptableObject
{
    public abstract void ApplyEffect(Player player);
}
```

Slika 4.11.1 – Apstraktna klasa AEffectConfig

Klasa AbstractEffect sadrži apstraktnu metodu Tick koja će se koristiti za implementaciju efekata u klasama koje će naslediti ovu klasu.

```
public abstract class AbstractEffect
{
    public abstract void Tick(Player player, float deltaTime);
}
```

Slika 4.11.2 – Apstraktna klasa AbstractEffect

4.11.1 Skripta Player.cs

U skripti Player.cs postoje tri liste koje će sadržati efekte i služe za kontrolu nad njima. Lista effectsToRemove služi za skladištenje efekata koje je potrebno ukloniti. U listi effectsToAdd se čuvaju efekti koje je potrebno dodati, dok lista effects služi za ažuriranje efekata. Ova skripta takođe sadrži i metode za dodavanje efekata u pomenute liste. Te metode su: AddEffect, RemoveEffect i AddEffectM. Ovaj deo je prikazan na slici 4.11.1.1.

```
private List<AbstractEffect> effects = new List<AbstractEffect>();
private List<AbstractEffect> effectsToRemove = new List<AbstractEffect>();
private List<AbstractEffect> effectsToAdd = new List<AbstractEffect>();

public void AddEffect(AbstractEffect effect)
{
    effects.Add(effect);
}

public void RemoveEffect(AbstractEffect effect)
{
    effectsToRemove.Add(effect);
}

public void AddEffectM(AbstractEffect effect)
{
    effectsToAdd.Add(effect);
}
```

Slika 4.11.1.1 – Skripta Player.cs (prvi deo)

U drugom delu (Slika 4.11.1.2) implementiran je metod Tick. Ovaj metod služi za ažuriranje listi efekata uz pomoć tri foreach petlje.

```
public void Tick(float deltaTime)
{
    // Remove effects
    foreach (var toRemove in effectsToRemove)
    {
        effects.Remove(toRemove);
    }

    effectsToRemove.Clear();

    // Add effects
    foreach (var toAdd in effectsToAdd)
    {
        effects.Add(toAdd);
    }

    effectsToAdd.Clear();

    // Update all effects
    foreach (var effect in effects)
    {
        effect.Tick(this, deltaTime);
    }
}
```

Slika 4.11.1.2 - Skripta Player.cs (drugi deo)

4.11.2 Efekat prikupljanja novčića

Kada Deda Mraz dođe u dodir sa žutim novčićima potrebno je impementirati efekat koji će obezbediti prikupljanje novčića i računanje skora na osnovu njih. Vrednost svakog novčića iznosi 1, pa će postignuti rezultat biti jednak broju pokupljenih novčića.

Na slici 4.11.2.1 prikazana je klasa CoinEffectConfig koja nasleđuje klasu AEffectConfig. U ovoj klasi je implementiran metod ApplyEffect. U ovom metodu će se pokrenuti zvuk PickUpCoin svaki put kada igrač pokupi novčić i pozvaće se metod za dodavanje novog efekta u listu aktivnih efekata.

```
[CreateAssetMenu(fileName = "CoinEffect", menuName = "Effects/Coin", order = 1)]
public class CoinEffectConfig : AEffectConfig
{
    public override void ApplyEffect(Player player)
    {
        FindObjectOfType<AudioManager>().PlaySound("PickUpCoin");
        player.AddEffect(new CoinEffect());
    }
}
```

Slika 4.11.2.1 – Klasa CoinEffectConfig

Na slici 4.11.2.2 prikazana je klasa CoinEffect koja nasleđuje klasu AbstractEffect. U ovoj klasi je implementiran metod Tick za efekat prikupljanja novčića. Naime, ažuriraće se broj prikupljenih novčića, a zatim taj efekat dodati u listu effectsToRemove.

```
public class CoinEffect : AbstractEffect
{
    [SerializeField]
    private int points = 1;

    public override void Tick(Player player, float deltaTime)
    {
        player.NumberOfCoins += points;
        //FindObjectOfType<AudioManager>().PlaySound("PickUpCoin");
        player.RemoveEffect(this);
    }
}
```

Slika 4.11.2.2 – Klasa CoinEffect

4.11.3 Efekat Slow

Kada Deda Mraz dođe u dodir sa narandžastim novčićem potrebno je impementirati efekat koji će obezbediti smanjenje brzine kretanja Deda Mraza. Trajanje ovog efekta je postavljeno na četiri sekunde.

Na slici 4.11.3.1 prikazana je klasa SlowEffectConfig koja nasleđuje klasu AEffectConfig. U njoj je implementiran metod ApplyEffect u kom se dodaje novi efekat usporavanja u listu aktivnih efekata.

```
[CreateAssetMenu(fileName = "SlowEffect", menuName = "Effects/Slow", order = 1)]
public class SlowEffectConfig : AEffectConfig
{
    [SerializeField]
    private float duration = 1.5f;
    [SerializeField]
    private float speedModifier = 0.5f;

    public override void ApplyEffect(Player player)
    {
        player.AddEffect(new SlowEffect(duration, speedModifier));
    }
}
```

Slika 4.11.3.1 – Klasa SlowEffectConfig

Na slikama 4.11.3.2 i 4.11.3.3 prikazana je klasa SlowEffect koja nasleđuje klasu AbstractEffect. Sadrži konstruktor za SlowEffect kao i implementaciju metode Tick. Ideja je da se pamti brzina igrača u trenutku u kom je pokupio narandžasti novčić u promenljivoj oldPlayerSpeed. Zatim se brzina modifikuje uz pomoć promenljive speedModifier. Kada istekne vreme predviđeno za trajanje ovog efekta, brzina se vraća na vrednost oldPlayerSpeed i poziva se metoda RemoveEffect.

```
public sealed class SlowEffect : AbstractEffect
{
    private float currentDuration = 0f;
    private float oldPlayerSpeed = 0f;
    private bool initialized = false;

    private float duration;
    private float speedModifier;

    public SlowEffect(float duration, float speedModifier)
    {
        this.duration = duration;
        this.speedModifier = speedModifier;
    }
```

Slika 4.11.3.2 – Klasa SlowEffect (prvi deo)

```
public override void Tick(Player player, float deltaTime)
{
    if (!initialized)
    {
        currentDuration = 0f;

        oldPlayerSpeed = player.PlayerSpeed;
        player.PlayerSpeed *= speedModifier;

        initialized = true;
    }

    currentDuration += deltaTime;

    if (currentDuration >= duration)
    {
        // Ovde je kraj
        initialized = false;

        player.PlayerSpeed = oldPlayerSpeed;
        player.RemoveEffect(this);
        return;
    }
}
```

Slika 4.11.3.3 - Klasa SlowEffect (drugi deo)

4.11.4 Efekat Shield

Na slici 4.11.4.1 prikazana je klasa ShieldEffectConfig koja nasleđuje klasu AEffectConfig. U njoj je implementiran metod ApplyEffect u kom se dodaje novi efekat štita u listu aktivnih efekata.

```
[CreateAssetMenu(fileName = "ShieldEffect", menuName = "Effects/Shield", order = 1)]
public class ShieldEffectConfig : AEffectConfig
{
    [SerializeField]
    private float duration = 1.0f;

    public override void ApplyEffect(Player player)
    {
        player.AddEffect(new ShieldEffect(duration));
    }
}
```

Slika 4.11.4.1 – klasa ShieldEffectConfig

Na slici 4.11.4.2 prikazana je klasa ShieldEffect koja nasleđuje klasu AbstractEffect. Sadrži kostruktor za ShieldEffect kao i implementaciju Tick metode. Ideja je da kada Deda Mraz pokupi plavi novčić u narednih 5 sekundi može prolaziti kroz prepreke bez kolizije sa njima. Ovaj efekat će biti i vizuelno prikazan u vidu plave prozirne sfere postavljene oko Deda Mraza. Da bi se ignorisala kolizija između željenih objekata korišćen je ugrađeni metod IgnoreLayerCollision. Ovaj metod kao argumente prima Layer-e između kojih je potrebno ignorisati koliziju. U ovom slučaju su to Layer “Obstacles” kojim se označavaju prepreke i Layer “Player” koji predstavlja igrača. Treći argument u ovom metodu je tipa bool. Ukoliko ima vrednost true, ignorisaće se kolizija, a ukoliko je vrednost false efekat ovog metoda se poništava.

```
public class ShieldEffect : AbstractEffect
{
    private float duration = 0;
    public ShieldEffect(float duration)
    {
        this.duration = duration;
    }

    private float currentDuration = 0;
    private bool initialized = false;

    public override void Tick(Player player, float deltaTime)
    {
        if (!initialized)
        {
            player.IsShielded = true;
            Physics.IgnoreLayerCollision(LayerMask.NameToLayer("Obstacles"), LayerMask.NameToLayer("Player"), true);
        }

        currentDuration += Time.deltaTime;

        if (currentDuration > duration)
        {
            Physics.IgnoreLayerCollision(LayerMask.NameToLayer("Obstacles"), LayerMask.NameToLayer("Player"), false);
            player.RemoveEffect(this);
            player.IsShielded = false;
            return;
        }
    }
}
```

Slika 4.11.4.2 – Klasa ShieldEffect

4.11.5 Efekat Magnet

Na slici 4.11.5.1 prikazana je klasa MagnetEffectConfig koja nasleđuje klasu AEffectConfig. U njoj je implementiran metod ApplyEffect u kom se dodaje novi efekat magneta u listu aktivnih efekata.

```
[CreateAssetMenu(fileName = "MagnetEffect", menuName = "Effects/Magnet", order = 1)]
public class MagnetEffectConfig : AEffectConfig
{
    [SerializeField]
    private float duration = 1.0f;

    public override void ApplyEffect(Player player)
    {
        player.AddEffect(new MagnetEffect(duration));
    }
}
```

Slika 4.11.5.1 – Klasa MagnetEffectConfig

Na slici 4.11.5.2 prikazana je klasa MagnetEffect koja nasleđuje klasu AbstractEffect. Ona sadrži konstruktor za MagnetEffect kao i imlementaciju metoda Tick. Kada igrač pokupi crveni novčić u narednih 5 sekundi imaće efekat magneta. Ideja za implementaciju efekta magneta je sledeća. Korišćen je metod OverlapSphere uz pomoć kog se postavlja kolajder u obliku sfere oko igrača i služi za detektovanje novčića koji se nađu u sferi. Nakon toga je potrebno obezbediti da se ti novčići približavaju igraču kako bi ih pokupio. Za to je korišćen metod MagnetToPlayer (Slika 4.11.5.3) koji je implementiran u skripti Pickup.cs. U Update metodi skripte Pickup.cs korišćen je ugrađeni metod Lerp koji predstavlja ustvari linearnu interpolaciju između dve tačke. U ovom slučaju te dve tačke su pozicija igrača i pozicija novčića. Nakon isteka predviđenog vremena za trajanje efekta, efekat se uklanja pozivanjem metode RemoveEffect.

```
public class MagnetEffect : AbstractEffect
{
    private float duration = 0;
    public MagnetEffect(float duration)
    {
        this.duration = duration;
    }

    private float currentDuration = 0;

    public override void Tick(Player player, float deltaTime)
    {
        Collider[] coinPickUps = Physics.OverlapSphere(player.pController.transform.position, 10f, 1 << LayerMask.NameToLayer("Coins"));
        foreach (var coin in coinPickUps)
        {
            Pickup pickup = coin.gameObject.GetComponent<Pickup>();
            pickup.MagnetToPlayer(player.pController);
        }

        currentDuration += Time.deltaTime;

        if (currentDuration > duration)
        {
            player.RemoveEffect(this);
            return;
        }
    }
}
```

Slika 4.11.5.2 – Klasa MagnetEffect

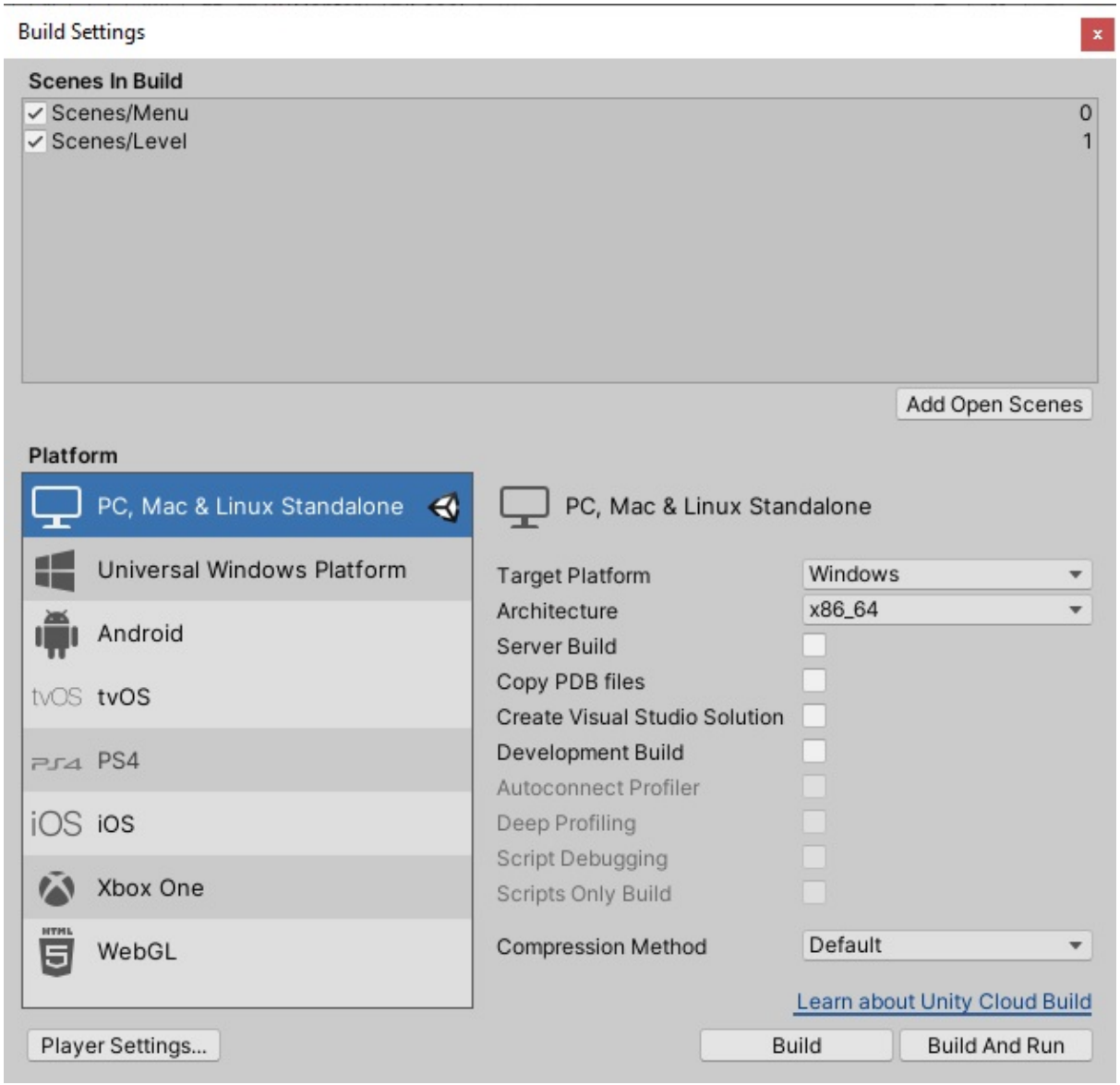
Pickup.cs skripta je zakačena za svaki od novčića, s tim što se u okviru nje može odabrati koji efekat će se primeniti na koji novčić. Pa su na taj način žuti novčići vezani za CoinEffect, narandžasti za SlowEffect, plavi za ShieldEffect i crveni za MagnetEffect.

```
public void MagnetToPlayer(PlayerController playerCon)
{
    if (isMoving) return;

    startPos = transform.position;
    timeStarted = Time.time;
    this.playerCon = playerCon;
    isMoving = true;
}

void Update()
{
    if (isMoving)
    {
        float moveDist = (Time.time - timeStarted) * magnetForce;
        transform.position = Vector3.Lerp(startPos, playerCon.transform.position, moveDist);
    }

    transform.Rotate(50 * Time.deltaTime, 0, 0);
}
```

Slika 4.11.5.3 – Metod MagnetToPlayer

4.12 Bildovanje video-igre za Windows platformu

Kao što je već rečeno, ova video-igra je predviđena za računare. Da bi se dobila završna verzija igrice i kako bi igrica mogla biti pokrenuta sa računara potrebno je izvršiti bildovanje. U File meniju je potrebno odabrati opciju Build settings nakon čega će se otvoriti prozor na slici 4.12.1. U odeljak Scenes in Build je potrebno dodati scene onim redom kojim želimo da se pojavljuju u igrici. U odeljku Platform odabrana je opcija PC, Mac & Linux Standalone, dok je za parametar Target platform odabran Windows. Klikom na dugme Build izvršiće se bildovanje igrice.



Slika 4.12.1 – Build podešavanja

5. Zaključak

U ovom radu su obrađeni osnovni koncepti Unity tehnologije. Dat je opis interfejsa i razvojnog okruženja kroz upoznavanje sa osnovnim elementima, komponentama i alatima. Predstavljen je GameObject, kao osnovna jedinica građe Unity aplikacija, i njemu odgovarajuća Game Objects - Components - Variables struktura, na kojoj se zasniva čitav razvoj složenih objekata i njihovih osobina. Paralelno sa uvođenjem osnovnih koncepata, u radu je dat opis razvoja složene 3D igre. Proces razvoja započet je idejama, i preko opisa funkcionalnosti, uvođenja novih pojmova, upoznavanja načina rada i dostupnih alata, dolazi se do implementacije rešenja i konačne realizacije. Logičan nastavak razvoja video-igre bio bi proširenje funkcionalnosti, rad na novim elementima igre, kreiranje novih nivoa, unapređenje grafike i vizuelnih efekata.

6. Literatura

[1] Unity Manual

https://docs.unity3d.com/Manual/

[2] Unity Global Game Industry

https://unity.com/our-company

[3] Unity

https://unity.com/

[4] Unity – wikipedia

https://en.wikipedia.org/wiki/Unity_(game_engine)

[5] Master rad, *Razvoj 3d igara za sistem Android pomoću okruženja Unity*, Nikola Milojević

http://www.racunarstvo.matf.bg.ac.rs/MasterRadovi/2015_05_27_Nikola_Milojevic/rad.pdf

[6] Master rad, *Unity: Osnovni koncepti i razvoj 3D igre*, Miljan Mijić

https://www.pmf.ni.ac.rs/download/master/master_radovi_računarske_nauke/računarske_nauke_master_radovi/2016/2016-12-14-mm.pdf