SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 7016

PRIMJENA FIZIKALNE SIMULACIJE U ANIMACIJI KRETANJA VIRTUALNOG LIKA U 3D VIDEOIGRI

Šimun Šprem

Zagreb, lipanj 2020.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 7016

PRIMJENA FIZIKALNE SIMULACIJE U ANIMACIJI KRETANJA VIRTUALNOG LIKA U 3D VIDEOIGRI

Šimun Šprem

Zagreb, lipanj 2020.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 13. ožujka 2020.

ZAVRŠNI ZADATAK br. 7016

Pristupnik:	**Šimun Šprem (0036507152)**
Studij:	Računarstvo
Modul:	Računarska znanost
Mentor:	prof. dr. sc. Maja Matijašević

Zadatak: **Primjena fizikalne simulacije u animaciji kretanja virtualnog lika u 3D videoigri**

Opis zadatka:

Od brojnih izazova u oblikovanju virtualnih okruženja za videoigre, tema ovog završnog rada uključuje primjenu znanja iz područja primjene fizikalno zasnovane simulacije u računalnoj grafici te razvoja programske podrške. Vaš zadatak je osmisliti i prema vlastitoj ideji programski izvesti jednostavnu 3D računalnu igru primjenom programskog okvira Unity. U skladu s ranije dogovorenom osnovnom idejom, koju trebate razraditi, igra treba uključivati fizikalnu simulaciju različitih oblika kretanja (hod, trčanje, skokove, penjanje) virtualnog lika, uz razne prepreke u virtualnom svijetu. Posebnu pažnju obratite na modeliranje realistične animacije kretnji virtualnog lika. Svu potrebnu literaturu i uvjete za rad osigurat će Vam Zavod za telekomunikacije.

Rok za predaju rada: 12. lipnja 2020.

Sadržaj

Uvod

Izrada videoigara izazovan je zadatak koji se sastoji od nekoliko slojeva. Kako bi učinili videoigru u isto vrijeme zabavnom, privlačnom i uvjerljivom, programeri moraju biti dobro upoznati s načinom izrade virtualnog okruženja te moraju znati kako implementirati zanimljive funkcionalnosti. Poneke videoigre se temelje na fizikalno zasnovanoj simulaciji pa je bitno da ta simulacija bude uvjerljiva. Jedna od takvih videoigara je „Floyd“, čiji je proces izrade koristeći razvojnu okolinu Unity opisan u ovom završnom radu. Cilj igre je doći s početne platforme na ciljnu uz pomoć pomičnih zidova, pritom izbjegavajući neprijatelje. U završnom radu se objašnjava dodavanje objekata u 3D prostor, prikazuje se uporaba skripti i opisuje se kako animirati pokrete lika preko Unityjevog sustava animacija.

Završni rad je podijeljen na 5 poglavlja:

1. Programski alati – navode se alati i tehnologije korištene za ostvarenje zadatka
2. Modeliranje 3D okoline – opisuje se postupak dodavanja objekata u 3D okolinu, dodavanje svojstava tim objektima i njihovo kombiniranje u uvjerljivu cjelinu
3. Skripte – koristeći programski jezik C#, dodana je mogućnost interakcije igrača s modelom igrača, kao i modela igrača s virtualnim svijetom u kojem se nalazi
4. Animiranje modela igrača – detaljno se objašnjava kako animirati pokrete igrača kako bi se postigla uvjerljivost pokreta
5. Rezultati – prikazani su okviri iz videoigre

1. Programski alati

Razvojna okolina u kojoj je modelirano 3D okruženje ovoga završnog rada je Unity. Sve skripte napisane su koristeći Microsoft Visual Studio jer Unity vrlo dobro povezuje objekte iz scene s Visual Studio skriptama. Programski jezik kojim su skripte pisane je C#.

1.1. Unity

Unity (https://en.wikipedia.org/wiki/Unity_(game_engine)) je razvojna okolina koja omogućava izradu višeplatformskih videoigara i aplikacija na čak 27 različitih platformi (Windows, macOS, Linux, PlayStation 4, Xbox One i brojne druge). To znači da kada završimo s izradom igre/aplikacije, potrebno je samo odabrati na kojem operacijskom sustavu želimo pokretati završni program, a Unity će dati direktorij s .exe datotekom. Unity nudi i izradu videoigara za virtualnu stvarnost (*engl. virtual reality*) koje se mogu pokretati koristeći Oculus Rift, Steam VR, PlayStation VR i ostale uređaje za virtualnu stvarnost.

Ova razvojna okolina omogućava izradu videoigara u 2D i 3D. Ova postavka može se promijeniti klikom gumba u alatnoj traci.

Slika 1.1 Gumb za promjenu iz 3D u 2D

Pisanje skripti u Unityju radi se primarno jezikom C#, ali se nudi i opcija pisanja koda JavaScriptom. Jedna od najzanimljivijih funkcionalnosti je povuci i ispusti (*engl. drag and drop*) koja omogućava dodavanje skripti, materijala, animacija i mnogih drugih svojstava objektima u sceni klikom miša.

Videoigre u razvojnoj okolini Unity pokreću se klikom na gumb „Play“ koji se nalazi iznad prozora scene. Jednom kada je igra pokrenuta, programer može mijenjati javne, ne statičke varijable pojedinih objekata, njihove veličine i položaje i drugo. Sve promjene napravljene za vrijeme testiranja neće se zadržati. Kako bi se spriječilo da programer zaboravi da je gumb „Play“ pritisnut te da ne napravi veliki napredak koji se neće spremiti, svi prozori osim prozora igre bit će zatamnjeni (ili obojani u boju po izboru programera).

Zbog jednostavnosti korištenja, povezanosti s Visual Studiom i portabilnosti, Unity je uz Godot, Unreal Engine i GameMaker jedna od najpopularnijih javnih razvojnih okolina [1].

Verzija Unityja kojom je zadatak obavljen je 2019.2.10f1.

1.2. Microsoft Visual Studio

Integrirano razvojno okruženje (kraće IDE, *engl. integrated development environment*) korišteno za ostvarenje programskog zadatka je Microsoft Visual Studio. Zbog brojnih funkcionalnosti, upravo je ovaj IDE zadan za pisanje skripti. Glavno svojstvo ovog IDE-a je to što se sve promjene napravljene u kodu gotovo trenutno vide u razvojnoj okolini Unity. Primjerice, ako se u skripti nadoda javna, ne statička varijabla, Unity će nakon svega nekoliko sekundi od spremanja te datoteke registrirati promjenu i dodati tu varijablu u Unityjev inspektor.

Slika 1.2 „Speed“ je jedna od javnih, ne statičkih varijabli skripte „Player Movement“, može se mijenjati i za vrijeme testiranja

Verzija Microsoft Visual Studia korištenog za pisanje skripti je 15.9.18.

1.3. Programski jezik C#

Jedan od najzastupljenijih programskih jezika korištenih za pisanje Unity skripti je C#. Ovaj programski jezik je općenito najuobičajeniji jezik za pisanje skripti Unity projekata. Razlog tomu je što je objektno orijentiran te je brži od ostalih objektno orijentiranih programskih jezika.

Verzija korištena u projektu je 7.0.

2. Modeliranje 3D okoline

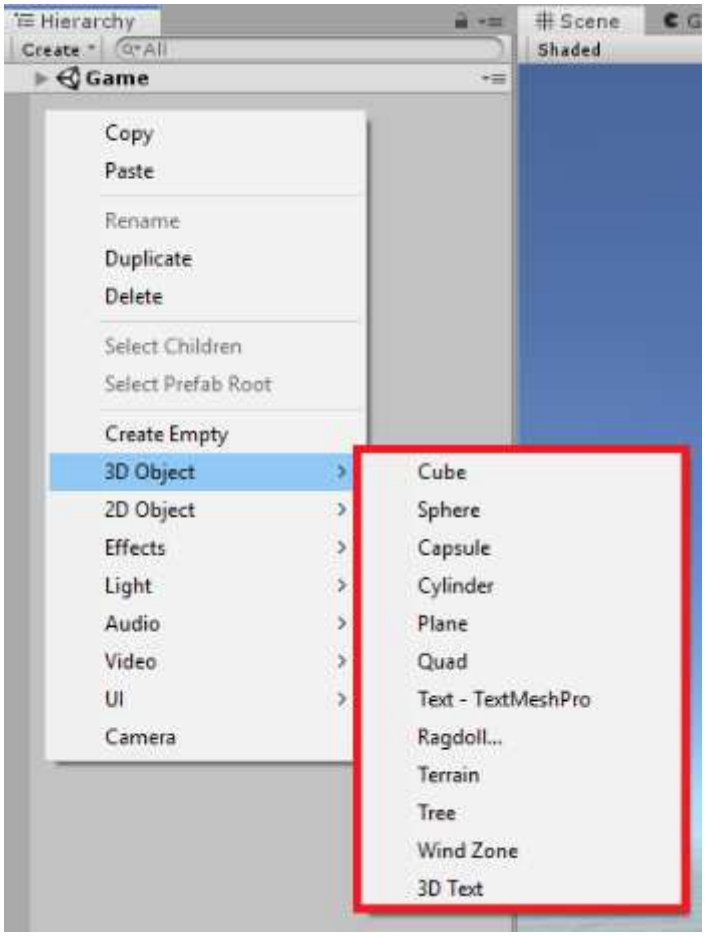
Izgradnja 3D okoline u Unityju sastoji se od izgradnje pojedinih scena, primjerice scena za početak igre, scena za pojedinu razinu, scena kada igrač izgubi, itd. Nova scena može se napraviti u projektu klikom na File > New Scene (ili kombinacijom tipki Ctrl + N) u alatnoj traci. Unutar scene se modeliraju 3D objekti, a ponašanja im se pridodaju skriptama.

Igra je zamišljena kao 3D platformer, a cilj je da igrač dovede model igrača do ciljne platforme skakući po zidovima i platformama [2]. Igrač pritom mora izbjegavati neprijatelje i padanje u lavu.

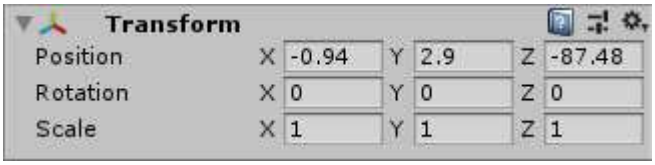
2.1. Izrada objekata

U svaku scenu se može dodati nekoliko osnovnih oblika, npr. kugla, kocka, valjak, kapsula itd. Objekte je moguće dodati desnim klikom miša u prozor „Hierarchy“ te se klikom na gumb 3D Object odabire željeni objekt, kako je prikazano na slici (Slika 2.2) Svaki objekt ima svoje ime, specifičnog je tipa (*engl. Tag*) i nalazi se u određenom sloju. Isto tako, klikom na kvadratni gumb moguće je ukloniti/vratiti objekt. Stvorenom objektu moguće je mijenjati položaj, rotaciju i veličinu u prostoru (Slika 2.3). Moguće je i stvoriti objekt unutar objekta, što kao rezultat daje da je jedan objekt roditelj drugome. Kod objekata koji

Slika 2.1 Ime, tip i sloj objekta



Slika 2.2 Odabir ponuđenih 3D objekata

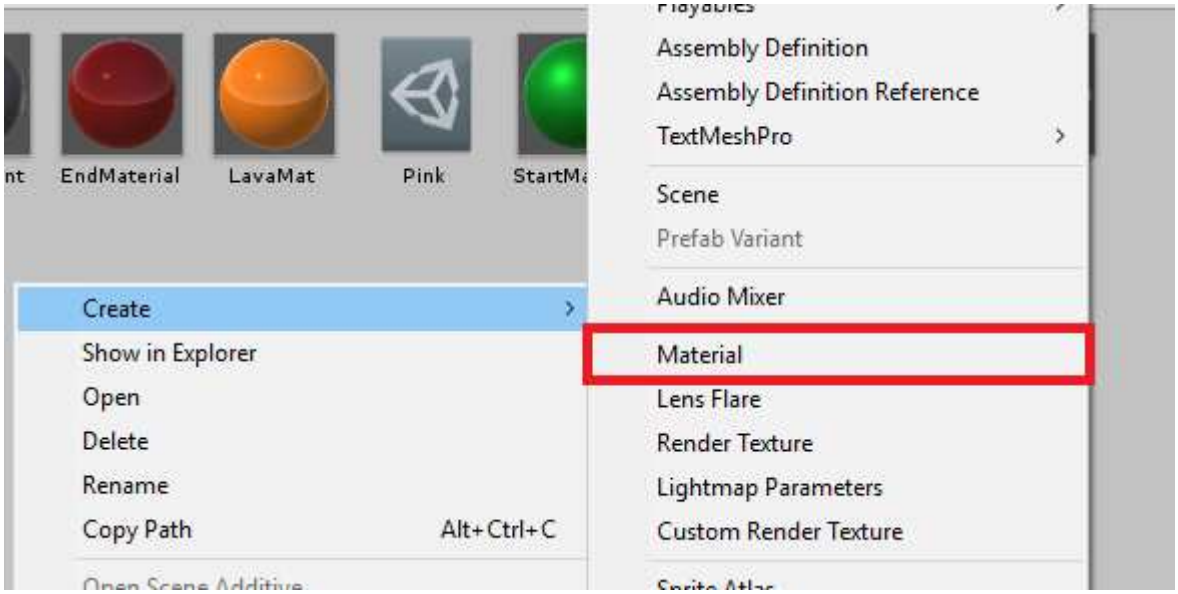


Slika 2.3 „Transform“ komponenta objekta u kojoj je moguće mijenjati položaj, rotaciju i veličinu objekta

2.2. Materijali

Dodani objekt zadano je bijele boje te ako se želi promijeniti, to je moguće u nekoliko koraka. Prvo je potrebno u prozor „Project“ desnim klikom miša kliknuti Create > Material, kako je prikazano na slici (Slika 2.4). U prozoru „Inspector“ novonastalog objekta moguće je mijenjati boju materijala. Kako bi objekt poprimio željenu boju, potrebno je odvući materijal na željeni objekt i ispustiti (metoda povuci i ispusti). Osim

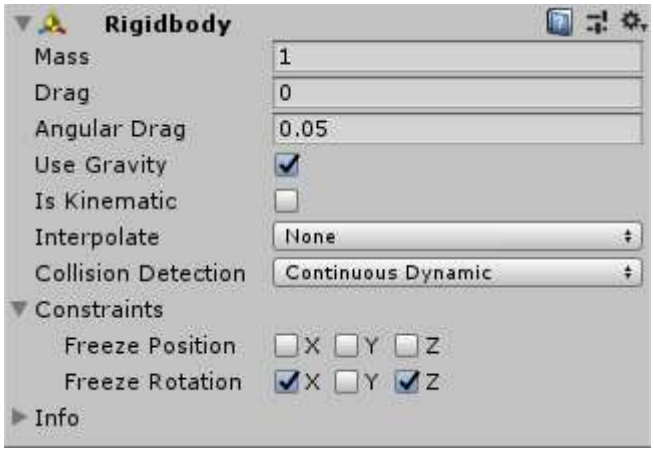
boje, objekt može poprimiti i fizički materijal, kojim je moguće modificirati trenje i faktor odskakivanja objekta koji sadrži taj fizički materijal [3].



Slika 2.4 Izrada novog materijala

2.3. Kruto tijelo

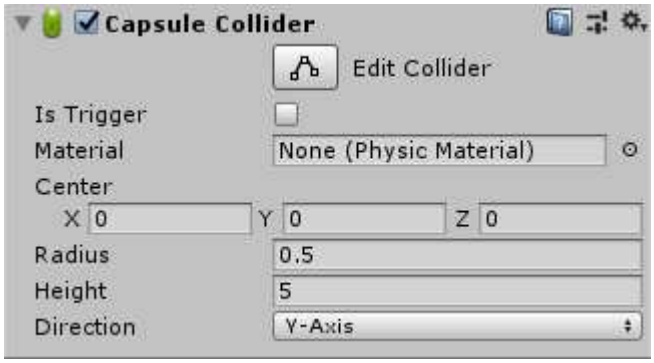
Novostvoreni objekt nema nikakva fizička svojstva, na njega ne djeluju nikakve sile. Dodavanjem komponente Kruto tijelo (*engl. Rigidbody*) na objekt počinju djelovati sile poput sile teže. Kroz ovu komponentu moguće je promijeniti masu objekta, opterećenje, uključiti/isključiti gravitaciju na objekt, zamrznuti mu položaj/rotaciju po određenim osima i ostalo [4].



Slika 2.5 Komponenta Rigidbody sa svojim opcijama

2.4. Detektor sudara

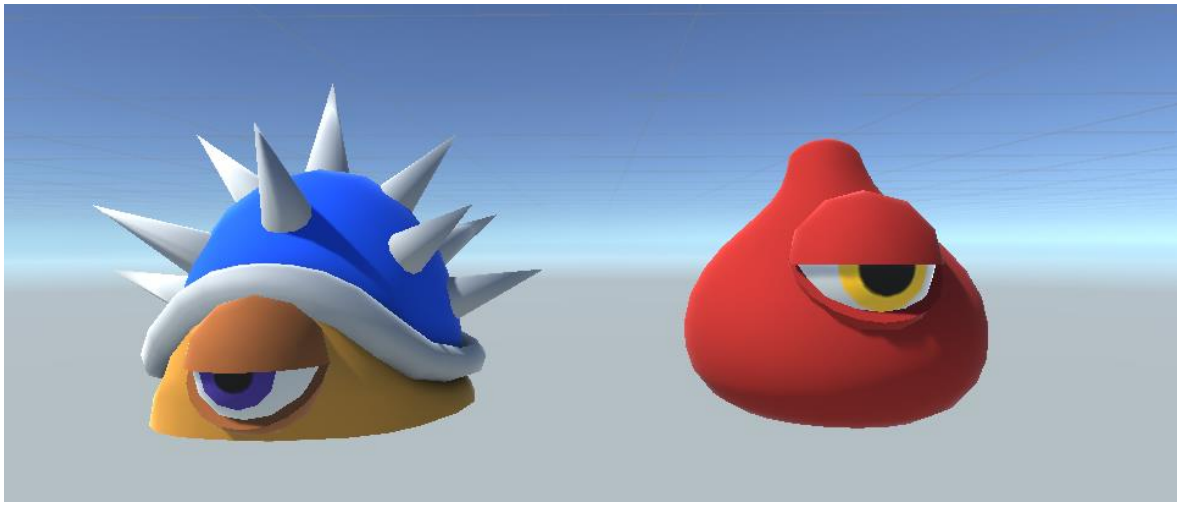
Kako bi Unity mogao registrirati sudare dvaju ili više objekata, potrebno im je dodati komponentu Detektor sudara (*engl. Collider*). Za svaki osnovni objekt (kugla, kocka, valjak itd.) postoji detektor sudara. Tako primjerice za kocku postoji komponenta Box Collider, za kuglu Sphere Collider i sl. Svim detektorima sudara moguće je promijeniti visinu i širinu, centar te im je moguće dodati fizički materijal (Slika 2.6).



Slika 2.6 Komponenta Capsule Collider sa svojim opcijama

2.5. Trgovina sredstvima

Vrlo bitna komponenta Unityja je Trgovina sredstvima (*engl. Asset Store*) u kojoj je moguće kupiti razne modele, teksture, zvukove i mnoge druge komponente potrebne za izradu igre. Besplatna sredstva korištena za ovaj rad su „RPG Monster Duo PBR Polyart“ korisnika Dungeon Mason (Slika 2.7), „Lava Flowing Shader“ korisnika Moonflower Carnivore te „18 High Resolution Wall Textures“ korisnika A dog's life software.



Slika 2.7 „RPG Monster Duo PBR Polyart“ korisnika Dungeon Mason

2.6. Scene

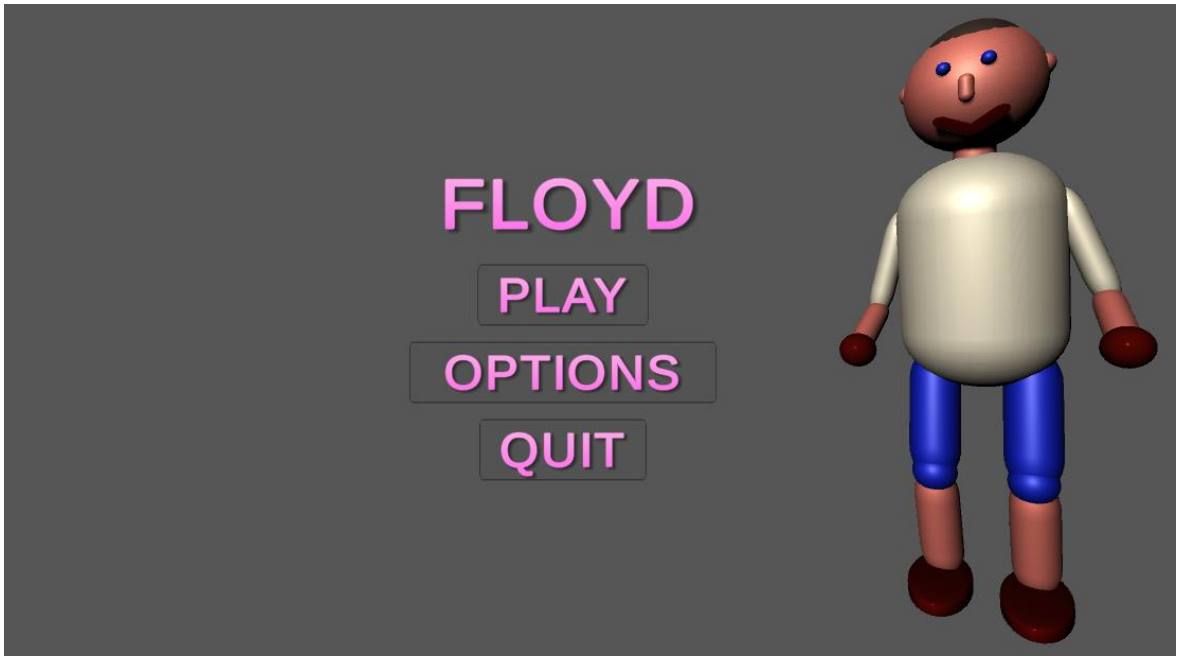
Svaka videoigra sastoji se od više različitih scena. Ovaj rad sastoji se od 3 scene: „Menu“, „Game“ i „Game Over“.

2.6.1. Scena „Menu“

Kada igrač pokrene igru, prva scena koju vidi je „Menu“ (Slika 2.8). Ova scena sastoji se od tamno sive pozadine, tri gumba i modela igrača.

Gumbi koji se nalaze na sceni „Menu“ su redom: „Play“, „Options“ i „Quit“. Pritiskom na gumb „Play“, igra se prebacuje na scenu „Game“ i korisnik može micati igrača. Gumb „Options“ otvara novi izbornik sa dva gumba: „Music“ i „Back“. Gumb „Music“ uključuje/isključuje glazbu, dok gumb „Back“ vraća korisnika na prijašnji izbornik. Gumb „Quit“ gasi aplikaciju.

Osim gumba, na sceni se nalazi i model igrača. Razlog tomu je da upozna igrača s modelom igrača te da poboljša estetiku scene [5].

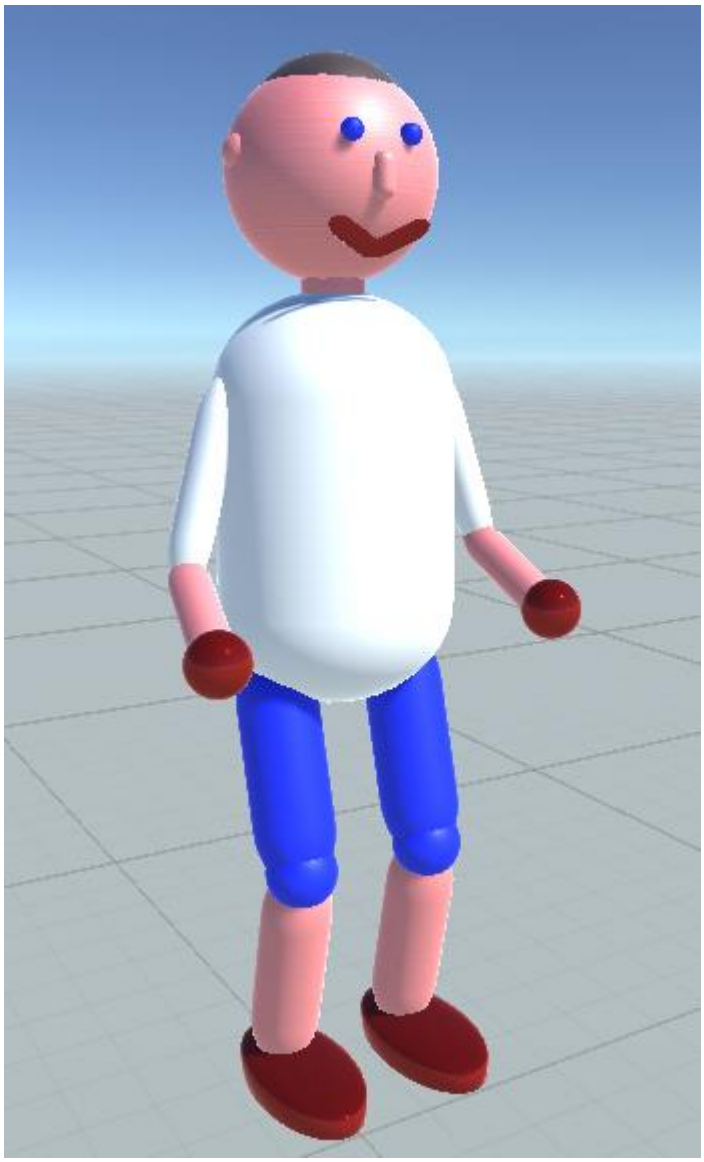


Slika 2.8 Scena „Menu“

2.6.2. Scena „Game“

Scena „Game“ predstavlja jedinu razinu rada. Sastoji se od nekoliko ključnih komponenata: igrač (Slika 2.9), neprijatelji (Slika 2.7) te zidovi (Slika 2.10 i Slika 2.11) za koje se igrač može primiti.

Model igrača sastoji se od približno 20 osnovnih elemenata koji se mogu kreirati unutar razvojnog okruženja Unity. Od komponenti igrač sadržava komponentu Rigidbody kako bi sile mogle utjecati na njega te komponentu Capsule Collider da ne prolazi kroz druge objekte. Kretanje igrača ostvareno je kroz tri skripte: „Player Movement“, „Jump Reseter“ i „Rotate Character“. Skripta „Player Movement“ zadužena je za mijenjanje pozicije igrača. Kroz igračevo pomicanje kroz igru, potrebno ga je i rotirati, za što je zadužena skripta „Rotate Character“. Na kraju, kako bi se spriječilo da igrač može neograničeno skakati, skripta „Jump Reseter“ provjerava je li igrač uzemljen te ako je, smije se odraziti od tla. Osim provjere smije li igrač skočiti, skripta provjerava sudare igarča s objektima te u slučaju sudara s neprijateljem, igra se prebacuje na scenu „Game Over“. Kako bi kamera pratila igrača, postoji još i skripta „Camera Movement“ unutar objekta „Camera“. Ona se pomicanjem i rotiranjem igrača pomiče i rotira u istom smjeru.



Slika 2.9 Model igrača

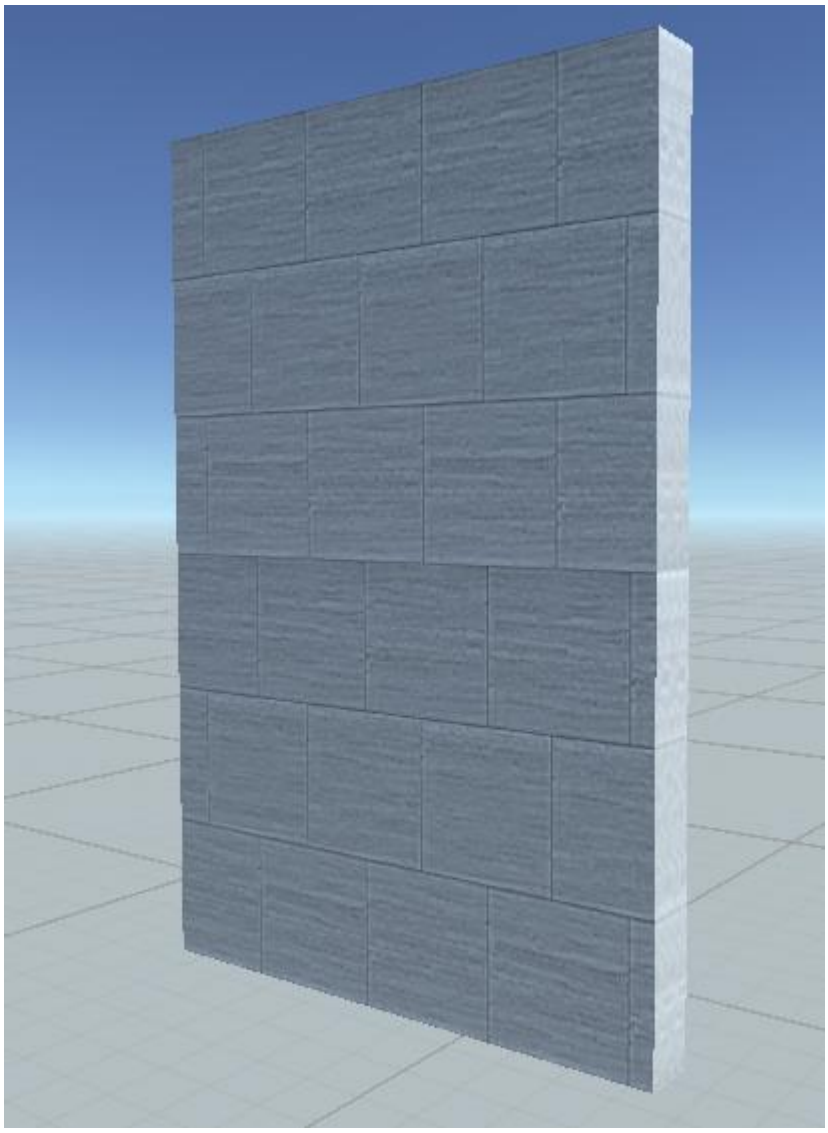
Model neprijatelja preuzet je iz Trgovine sredstvima od korisnika Dungeon Mason (Slika 2.7). Neprijatelji su objekti stvoreni kako bi otežali igraču završetak igre. Postoje dvije vrste neprijatelja: crveni i plavi.

Crveni neprijatelji imaju kružne kretnje i nastoje otežati prvi dio razine. Njihove kretnje opisane su skriptom „Enemy Movement Circular“, kreću se po kružnici.

Plavi neprijatelji ima vertikalnu kretnju i kreirani su s ciljem da otežaju drugi dio razine. Kretnja im je opisana skriptom „Enemy Movement Vertical“.

U igri postoje dvije vrste zidova, podijeljeni po boji i smjeru kretanja, a to su bijeli (tekstura bijelog kamena) (Slika 2.10) i crni (tekstura crnog kamena) (Slika 2.11). Zidovi služe kako bi se igrač mogao primiti za zid te tako napredovati kroz igru.

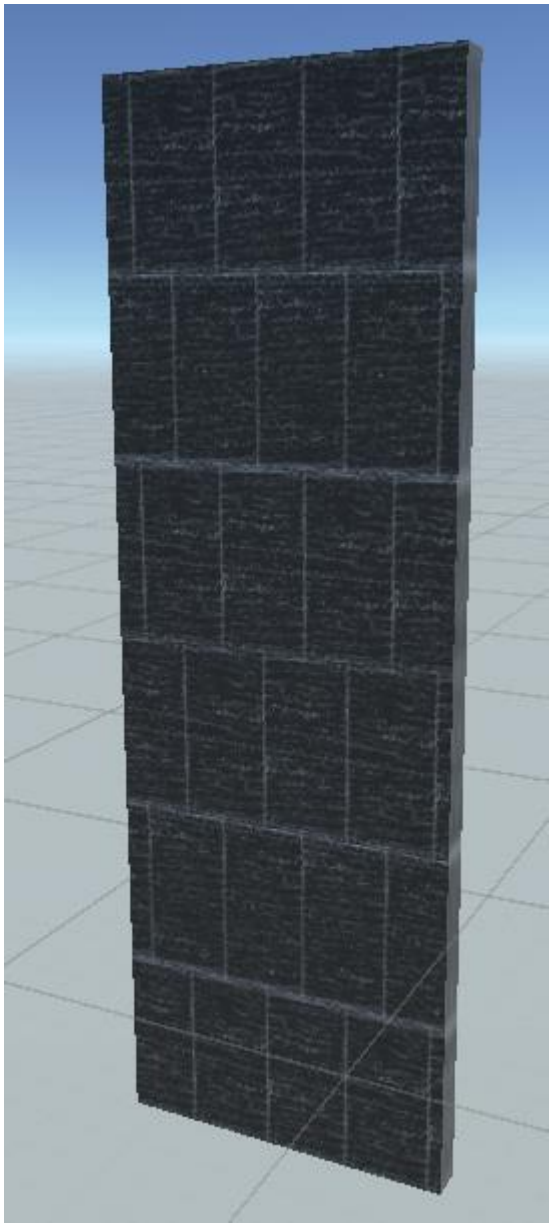
Bijeli zidovi nalaze se u prvom dijelu igre. Ova vrsta zidova pomiče se horizontalno, u smjeru užih strana. Igrač koristeći ove zidove može doći do središnje platforme preko koje može preći u drugi dio igre. Kretnja zidova postignuta je skriptom „WallsMovement“.



Slika 2.10 Bijeli zid

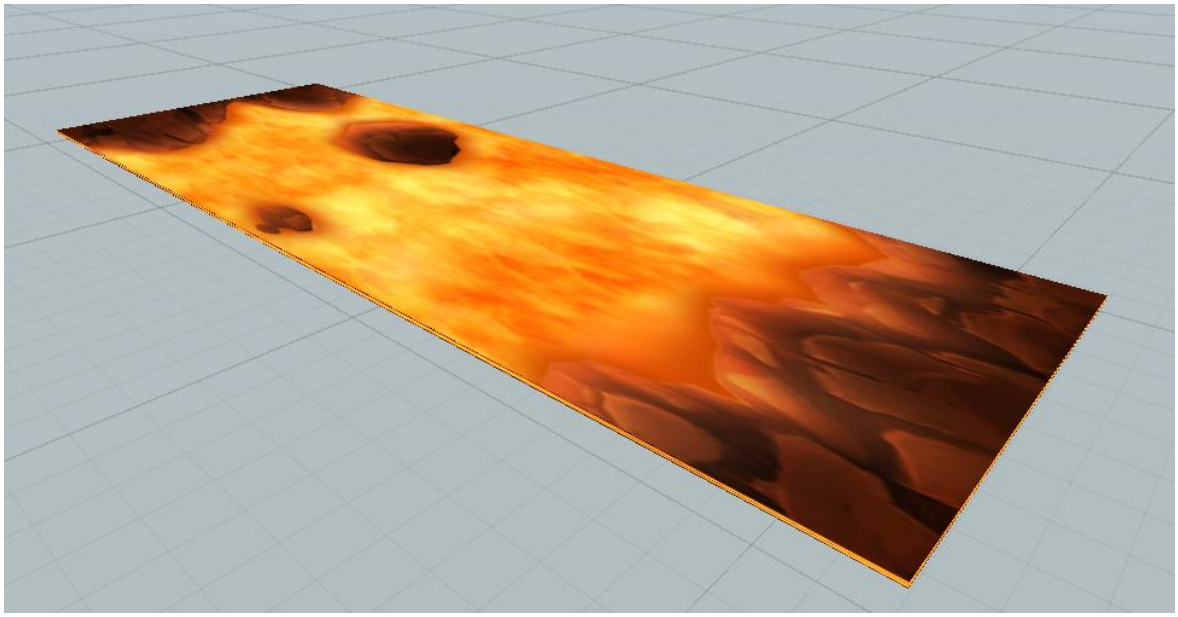
Crni zidovi nalaze se u drugom dijelu igre. Viši su od bijelih zidova, a igrač ih treba iskoristiti kako bi došao do kraja igre, odnosno ciljne platforme koja se nalazi iznad igrača. Crni zidovi se također pomiču horizontalno, ali se od bijelih razlikuju jer se kreću u smjeru širih strana. Njihova kretnja ostvarena je animacijama, ali se krajnji rezultat ne razlikuje da je ostvarena skriptom (kao što je to obavljeno za bijele zidove).

Oba zida koriste teksture preuzete iz trgovine sredstvima „18 High Resolution Wall Textures“.



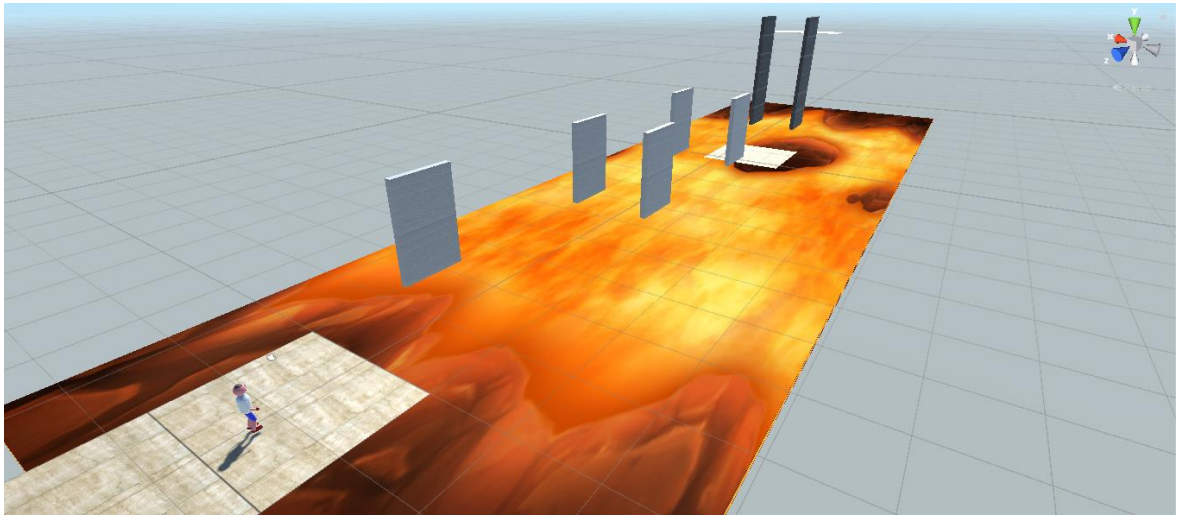
Slika 2.11 Crni zid

Kako bi igraču igra bila izazovnija, ispod zidova dodana je lava (Slika 2.12) u koju igrač ne smije upasti. Kako bi se postigao bolji izgled lave, iz trgovine sredstvima preuzete su i korištene teksture „Lava Flow Shader“ koje su primijenjene nad oblikom kvadra. Ako igrač padne u lavu (ako dođe do sudara modela igrača i kvadra s teksturom lave), igra završava i prikaz se prebacuje na scenu „Game Over“.



Slika 2.12 Lava

Rezultat kombiniranja navedenih elemenata u sceni prikazan je na slici (Slika 2.13) (neprijatelji se na slici ne vide jer su ispod lave).



Slika 2.13 Scena „Game“

2.6.3. Scena „Game Over“

U slučaju da igrač padne u lavu ili ako dotakne neprijatelja, igra završava i prikaz će biti prebačen na scenu „Game Over“ (Slika 2.14). Ova scena sastoji se od sive pozadine te dva gumba: „Restart“ i „Quit“. Odabirom gumba „Restart“, igrač se postavlja na početnu poziciju i ponovno može igrati. Klikom na gumb „Quit“ igrač izlazi iz igre.

Slika 2.14 Scena „Game Over“

3. Skripte

Skripte u Unityju se koriste za upravljanje događajima u 3D okolini, primjerice pomicanje igrača u virtualnom svijetu, njegove interakcije s objektima, itd. C# skripta u Unityju izrađuje se desnim klikom miša u prozor „Project“ te klikom na Create > C# Script. Skriptu je moguće otvoriti dvostrukim klikom na skriptu u prozoru „Project“. Skripta se otvara u Microsoft Visual Studiu i sadrži dvije vrlo bitne metode: `Start` i `Update`. Metoda `Start` pozivat će se prije no što se prvi okvir (*engl. frame*) ažurira. Metoda `Update` pozivat će se svaki okvir. Kada je skripta napisana, dodaje se objektu povlačenjem i ispuštanjem.

U nastavku će biti opisane one skripte koje su razvijene za potrebe ovog završnog rada.

3.1. Skripta „PlayerMovement“

Skripta „PlayerMovement“ zadužena je za upravljanje modelom igrača. Koristeći klasu UnityEngine.Input moguće je dobiti podatke o tipkama koje je korisnik pritisnuo. Metoda GetAxisRaw navedene klase kao argument prima ime osi („Horizontal“ ili „Vertical“) te vraća broj koji predstavlja koeficijent u smjeru te osi.

```
1 float h = Input.GetAxisRaw("Horizontal");
2 float v = Input.GetAxisRaw("Vertical");
3 moveDirection.Set(h, 0, v);
4 moveDirection = -moveDirection.normalized * speed * Time.deltaTime;
5 transform.position += moveDirection;
```

Kôd 3.1 – Program za određivanje smjera kretanja

Primjerice, kod 3.1 koristi dvije varijable, `h` i `v`, u koje sprema koeficijente smjerova [6]. Zatim, postavlja x i z komponente vektora `moveDirection` u `h` i `v`. U 4. redu `moveDirection` se normalizira i množi s varijablom speed, koja služi za skaliranje normaliziranog vektora. Još je potrebno pomnožiti vektor s varijablom `Time.deltaTime`. Kada se u Unityju radi s brzinama, gotovo je uvijek potrebno množiti s ovom varijablom kako bi se postigli zadovoljavajući rezultati. Razlog tomu je što se bez varijable `Time.deltaTime` položaj objekta koji se pomiče mijenja svaki put kada se ažurira okvir. Primjerice, kod vrlo brzog računala koje ažurira okvir 100 puta u sekundi,

brzina objekta biti će 2 puta veća nego kod sporijeg računala koje ažurira okvir 50 puta u sekundi. Ovo je neželjena posljedica koja se rješava množenjem pomaka varijablom `Time.deltaTime`. Ova varijabla vratit će vrijeme u sekundama od zadnjeg ažuriranja okvira i time se neutralizira brzina ažuriranja okvira te se dobiva jednaka brzina na svim računalima. U zadnjem koraku na trenutni položaj objekta pribraja se vektor `moveDirection,` čime se objekt pomiče u željenom smjeru. Ovaj pomak dogodit će se svaki puta kada se pozove metoda `Update`, odnosno svaki puta kada se ažurira okvir te se vrlo brzim ažuriranjima dobiva osjećaj kretanja objekta.

Osim pomake po x i z osi, ova skripta ostvaruje i pomak po y osi, odnosno skakanje.

```
1 if (Input.GetKeyDown(KeyCode.Space))
2        {
3                if (onGround)
4                    {
5                        onGround = false;
9                        rb.velocity = jump * jumpSpeed;
8                    }
10        }
```

Kôd 3.2 – Dio koda za skakanje

Skakanje u igri biti će ostvareno pritiskom tipke Space. Kada se tipka Space pritisne, provjerava se je li model igrača na tlu. Ako je to slučaj, brzina tijela postavlja se na vrijednost vektora `jump` pomnoženog s varijablom `jumpSpeed`. Vektor `jump` definiran je prije u kodu te njegova deklaracija glasi: `jump = new Vector3(0, 2, 0);`. Varijabla `jumpSpeed` javna je varijabla ove skripte te joj je pretpostavljena vrijednost 3. Nakon množenja, ove dvije varijable daju brzinu u pozitivnom smjeru y osi koja se pridaje objektu.

Kako bi se ova skripta uključila u videoigru, potrebno je odabrati objekt (u ovom slučaju model igrača) kojeg želimo pomicati. Ovo se može postići tako da se u prozoru „Project“ povuče i spusti skripta na željeni objekt (Slika 1.2 prikazuje uspješno nadodanu skriptu objektu).

3.2. Skripta „CameraMovement“

Osim pomicanja modela igrača, potrebno je s njime pomicati i kameru. Ova funkcionalnost postiže se u skripti „CameraMovement“. U metodi `Start` definirana je varijabla `offset`: `offset = new Vector3(0, 6, 20);`. Ova varijabla označava na kojoj udaljenosti od igrača će kamera biti postavljena.

```
1 offset = Quaternion.AngleAxis(Input.GetAxis("Mouse X") * rotationSpeed, Vector3.up) * offset;
2 transform.position = player.position + offset;
3 transform.LookAt(player.position);
```

Kôd 3.3 – Dio koda koji pomiče kameru oko igrača

Za rotaciju objekta potrebno je koristiti metodu AngleAxis razreda Quaternion. Ova metoda prima dva argumenta: kut za koji želimo rotirati objekt i os oko koje ga želimo rotirati. U kodu 3.3, uzima se kut koji za koji se pomiče miš te se kameru rotira oko osi koja gleda prema gore, osi y te ga se množi javnom varijablom `rotationSpeed` koja služi za skaliranje brzine rotacije kamere. Varijabla `offset` postavlja se na vrijednost starog `offseta` pomnoženu s kutom za koji želimo rotirati kameru. U drugoj liniji postavlja se položaj kamere na poziciju igrača pomaknutu za `offset`. Linija 3 rotira kameru tako da je ona uvijek usmjerena prema igraču [7].

3.3. Skripta „RotateCharacter“

Skripta „PlayerMovement“ pomiče igrača kroz igru, ali ga ne rotira. Ova funkcionalnost dodaje se kroz skriptu „RotateCharacter“. Rotacija se obavlja tako da se uzme rotacija kamere te se model igrača rotira za isti kut na koji se još zbroji 180 kako bi model igrača gledao u smjeru kamere, a ne u kameru. Linija koja mijenja rotaciju modela igrača je `transform.rotation = Quaternion.Euler(0, cameraRotation.eulerAngles.y + 180, 0);`.

3.4. Skripta „JumpReseter“

Osnovni zadatak ove skripte je registriranje sudara igrača s objektima u njegovoj okolini. Ovi sudari mogu prebaciti prikaz na scenu „GameOver“ (sudar s lavom ili neprijateljem, za igrača su nepoželjni) ili pomoći igraču da dođe do ciljne platforme (sudar sa zidovima).

Temeljna metoda ove skripte je OnCollisionEnter, koja prima parametar tipa Collision. Ova metoda poziva se svaki put kada se objekt sudari s nekim drugim objektom, a parametar `collision` sadrži podatke o sudaru.

```
1 if (collisionTag == "Lava" || collisionTag == "Enemy")
2   SceneManager.LoadScene("GameOver");
```

Kôd 3.4 – Registriranje sudara s lavom ili neprijateljem

U slučaju da dođe do sudara modela igrača i lave ili neprijatelja, potrebno je promijeniti scenu u „Game Over“.

```
1 if (timer == 0 & collisionTag != "Enemy" & collisionTag != "Lava")
2     PlayerMovement.onGround = true;
```

Kôd 3.5 – Registriranje da je korisnik na tlu/zidu

Ako se model igrača sudari s bilo čime što nije lava ili neprijatelj, igraču se daje mogućnost skakanja. Također, kako bi se igraču omogućilo da skoči, varijabla `timer` mora biti 0. Ova varijabla bitna je jer Unityjeva fizikalna simulacija radi tako da određenom frekvencijom provjerava je li između bilo koja dva objekta došlo do sudara. Problem je što je ta frekvencija ponekad premalena te se jedan objekt može naći unutar drugoga. U ovom slučaju bi to značilo da se dio modela igrača našao u zidu. Kada bi igrač pokušao skočiti na drugi zid, sustav detekcije sudara ponovno bi registrirao sudar te bi igrača ponovno „zalijepio“ za isti zid, uz minimalni pomak. Budući da to nije željeni efekt unutar igre, za očekivati je da bi uzrokovao frustracije kod igrača te ga stoga treba izbjeći. Varijabla `timer` ne sprečava da se igrač nađe u zidu, ali ga sprečava da se unutar određenog vremenskog intervala (u ovom slučaju 0.2 sekunde) ponovno primi za zid.

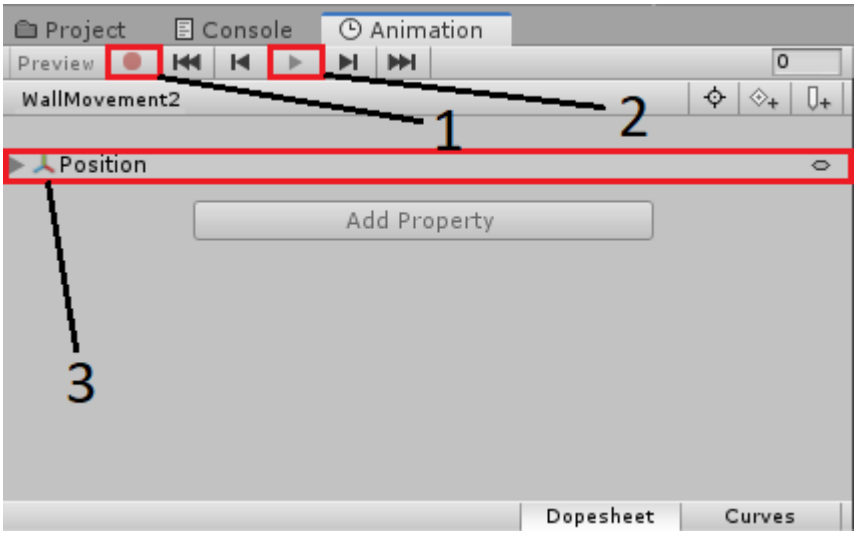
4. Animiranje modela igrača

Osim pomicanja modela igrača u prostoru, potrebno je i animirati kretnje virtualnog lika kako bi on izgledao uvjerljivo. Animacije omogućuju manipulaciju objektima kakvu je teško postići skriptama [8].

4.1. Izrada animacija

Animacije se kreiraju desnim klikom miša u prozor „Project“ te klikom na Create > Animation. Tada će se stvoriti nova animacija (file tipa .anim) koju je moguće otvoriti dvostrukim klikom miša. Ovime će se pokraj kartice za prozor „Project“ i „Console“ pojaviti i kartica „Animation“. Ova kartica sastoji se od dva povezana dijela, dio s obilježjima i polje vremena. Prvi dio sastoji se od nekoliko bitnih gumbi te sadrži sve komponente objekata koji sudjeluju u animaciji. Na slici (Slika 4.1) brojevima su označene najbitnije komponente prvoga dijela:

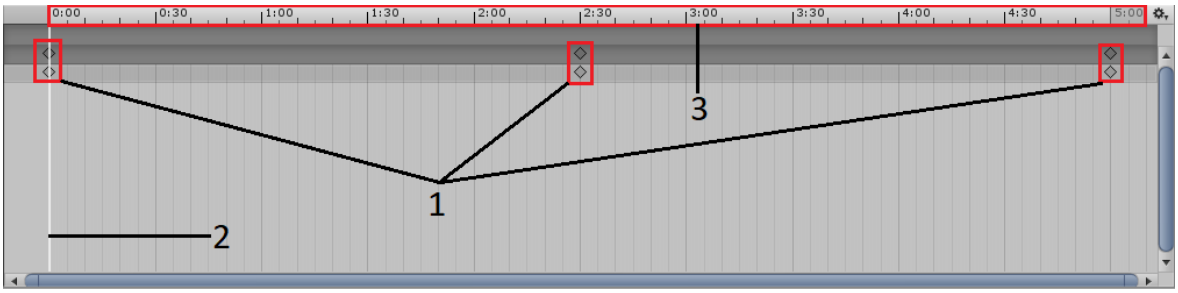
1. Gumb za snimanje – kada je pritisnut, animacija se snima te se promijene položaja zapisuju u trenutak naznačen na polju vremena
2. Gumb za reproduciranje – kada je pritisnut, trenutno napravljena animacija kreće s izvođenjem te se ponavlja dok se gumb ne pritisne ponovno
3. Obilježje čiji se položaj pamti



Slika 4.1 Dio s obilježjima

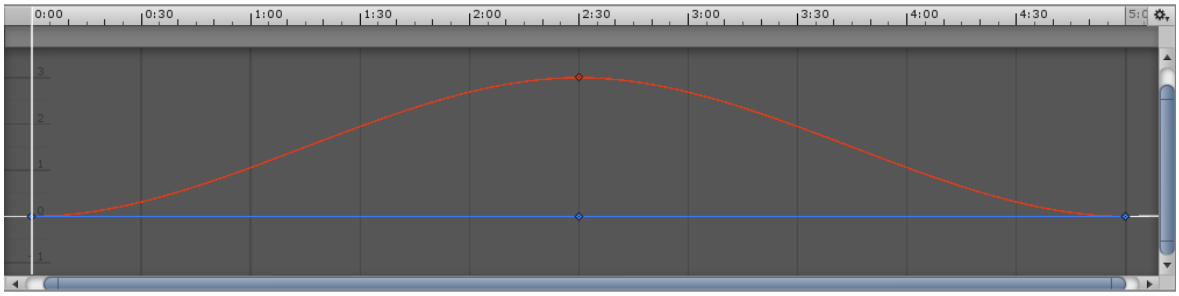
Drugi dio služi kako bi se precizno odredilo koji će dio u kojem trenutku biti na kojem mjestu. Na slici (Slika 4.2) su označene ključne komponente polja vremena:

1. Stanje komponenata objekta u određenom trenutku (primjerice, u trenutku 0:00 položaj objekta je 0, 0, 0, u trenutku 2:30 je 0, 5, 0, a u trenutku 5:00 je opet 0, 0, 0; primjer prikazuje animaciju koja pomiče i spušta objekt kroz 5 sekundi.)
2. Bijela crta – predstavlja trenutno vrijeme; stanje komponenata objekta u tom trenutku može se mijenjati u dijelu s obilježjima
3. Lenta vremena – prikazuje za koje sve trenutke postoje stanja komponenata objekta, pritiskom na određeno mjesto na lenti vremena prebacuje bijelu crtu na pritisnuto mjesto



Slika 4.2 Polje vremena

Kada su ključne točke odabrane, Unity ih sam povezuje i uglađuje. Učinak toga vidljiv je pokretanjem animacije. Ako postoji potreba za ručnim popravljanjem, to je moguće klikom na gumb „Curves“ u donjem desnom kutu dijela s obilježjima. Sada se umjesto polja vremena nalaze krivulje (Slika 4.3) koje predstavljaju promjene iz jednog stanja komponenata u drugo [9].



Slika 4.3 Krivulje stanja komponenata

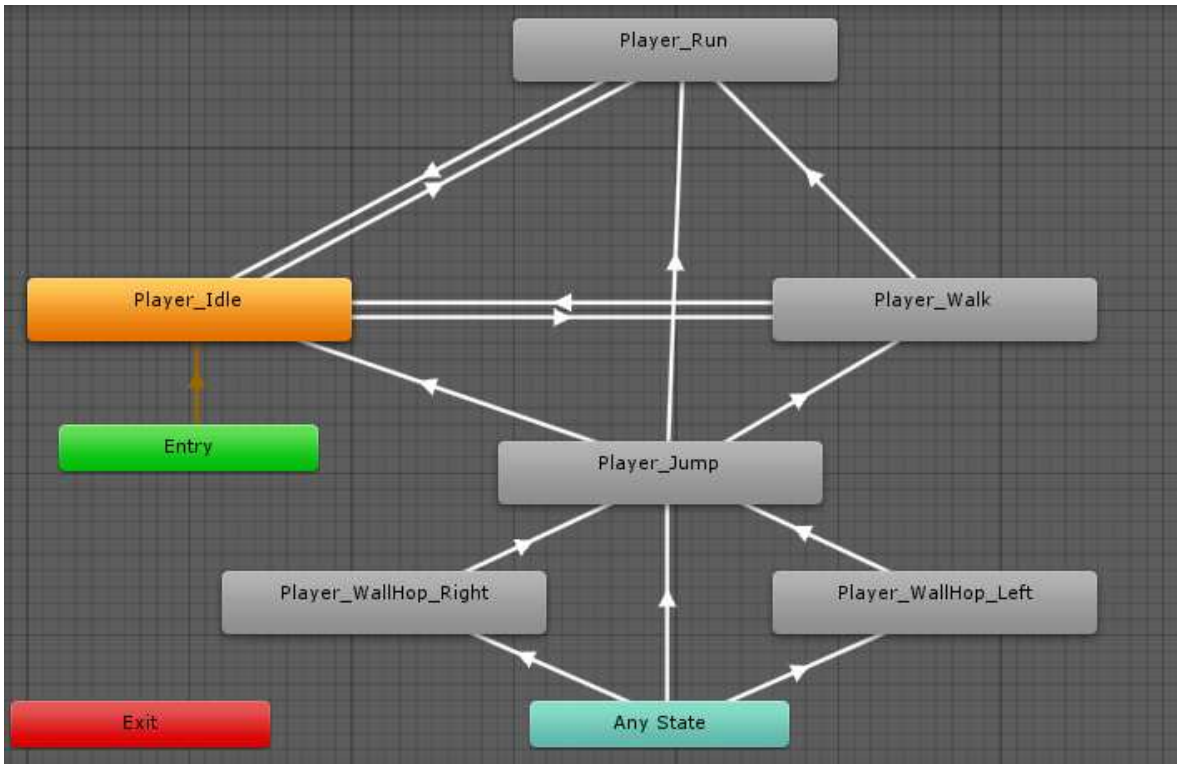
Na prethodnoj slici (Slika 4.3) crvena krivulja predstavlja promjenu x koordinate objekta kroz 5 sekundi, a plava krivulja (u ovom slučaju linija) z koordinatu. Krivulja za y koordinatu se ne vidi jer se podudara s plavom krivuljom. Iz ovoga se vidi da se objekt koji sadrži ovu animaciju pomiče samo po x osi kroz 5 sekundi za 3 jedinice. Također, vidi se da je u trenutku 2:30 sekundi objekt najdalje od početnog položaja (3 jedinice). S obzirom da ovo nije linija već krivulja, objekt će se pomicati s ubrzanjima te će izgledati prirodno.

Pritiskom na točke na krivulji moguće je odvući krivulju u bilo kojem smjeru i tako je promijeniti. Tako primjerice ako se želi postići naglije animacije, potrebno je povući točku u lijevo ili u desno .Isto tako, krivulja se može podizati i spuštati čime će se točke između kojih se objekt kreće pomaknuti te se može sužavati i širiti, čime će se period animacije smanjiti, odnosno povećati.

4.2. Animator

Stvaranjem animacije, odnosno datoteke s ekstenzijom .anim, paralelno se stvori i animator, datoteka s ekstenzijom .controller. Dvostrukim klikom na ovu datoteku otvorit će se novi prozor, Animator. U ovom prozoru postoje dva bitna dijela: automat stanja i parametri. Na slici (Slika 4.4) vidi se automat stanja koji se sastoji od 5 vrsta stanja:

- Stanje „Entry“ – stanje u kojem je objekt nad kojim se animacija vrši prije no što se igra pokrene
- Stanje „Any State“ – ako je bilo koji od uvjeta za tranziciju ostvaren, prelazi se u stanje za koje je prijelaz istinita
- Stanje „Exit“ – vraćanje u stanje „Entry“ direktno nije moguće, ali je moguće otići u stanje „Exit“ čime se kao trenutno stanje postavlja „Entry“
- Zadano stanje – stanje u koje se uvijek ulazi iz stanja „Entry“
- Obična stanja – sva stanja koja nisu niti jedno od gore navedenih; za svaku animaciju koja je napravljena postoji jedno stanje

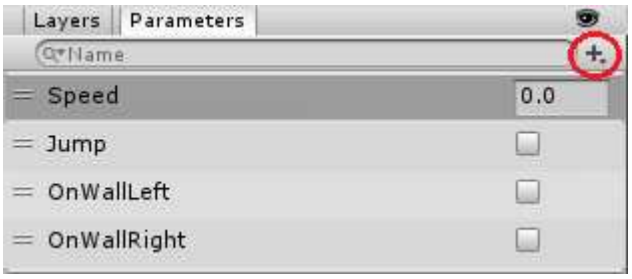


Slika 4.4 Automat stanja

Osim stanja, na slici se vide i prijelazi. Prijelaz se može napraviti desnim klikom na jedno stanje te odabirom „Make Transition“. Nakon toga treba odabrati drugo stanje. U napravljenom prijelazu se prelazi iz prvog u drugo stanje. Potrebno je još dodati uvjete kada treba napraviti prijelaz. Prije dodavanja uvjeta, potrebno je stvoriti parametre u dijelu s parametrima klikom na gumb „+“. Za izradu parametara odabire se tip parametra i njegovo ime. Od tipova moguće je odabrati:

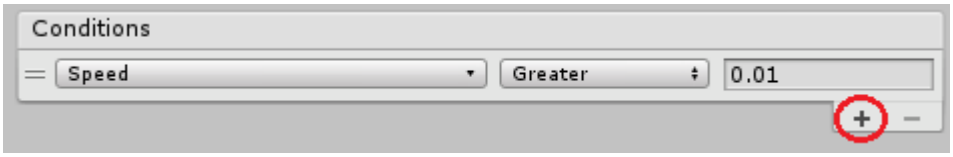
- Float
- Int
- Bool
- Trigger

Na slici (Slika 4.5) prikazan je prozor s već kreiranim parametrima te njihovim inicijalnim vrijednostima.



Slika 4.5 Parametri

Uvjeti za prijelaz postavljaju se tako da se u prozoru „Inspector“ pritisne gumb „+“, kako je prikazano na slici (Slika 4.6). Nakon toga se odabire koji se parametar želi kontrolirati te vrijednost za koju se ispituje taj parametar. U primjeru na slici (Slika 4.6), prijelaz će se obaviti ako je parametar „Speed“ veći od 0.01.



Slika 4.6 Dodavanje uvjeta za tranziciju

S obzirom da je predodređena vrijednost parametra „Speed“ 0, a prijelaz sa Slika 4.6 će se dogoditi kada je parametar „Speed“ veći od 0.01 znači da se taj prijelaz nikad neće dogoditi

Budući da ova varijabla pomicanjem igrača ostaje 0 jer, nije specificirano kada se mijenja, potrebno je u kodu dodati linije koje je mijenjaju. Parametar „Speed“ bi se trebao povećati kada se igrač pomiče. S obzirom da je parametar „Speed“ tipa float, u skriptu „PlayerMovement“ je dodana linija: `animator.SetFloat(„Speed“, Mathf.Abs(h) + Mathf.Abs(v));`, gdje su varijable `h` i `v` realan broj između -1 i 1 koji opisuju smjer kretanja (`h` predstavlja pomicanje po x osi, a `v` po z osi). Apsolutna vrijednost koristi se jer postoji slučaj u kojem su `h` i `v` iste vrijednosti obrnutog predznaka te bi u zbroju dale 0. Kada bi parametar bio tipa bool, funkcija bi bila `animator.SetBool`[10].

U postavkama prijelaza moguće je promijeniti njegovo trajanje. U ovoj videoigri on je jednak 0 jer je tada prijelaz iz animacija trenutan. Izrada animacija preuzeta je s YouTube videa [11].

4.3. Animacije modela igrača

Igrač je u stanju mirovanja kada se ne miče, kada ne skače i kada nije na zidu. U animatoru je ovo stanje označeno narančastom bojom, što znači da je ovo zadano (*engl. default*) stanje. Ovo stanje prikazano je ranije u radu (Slika 2.9).

Animacije korištene u videoigri su:

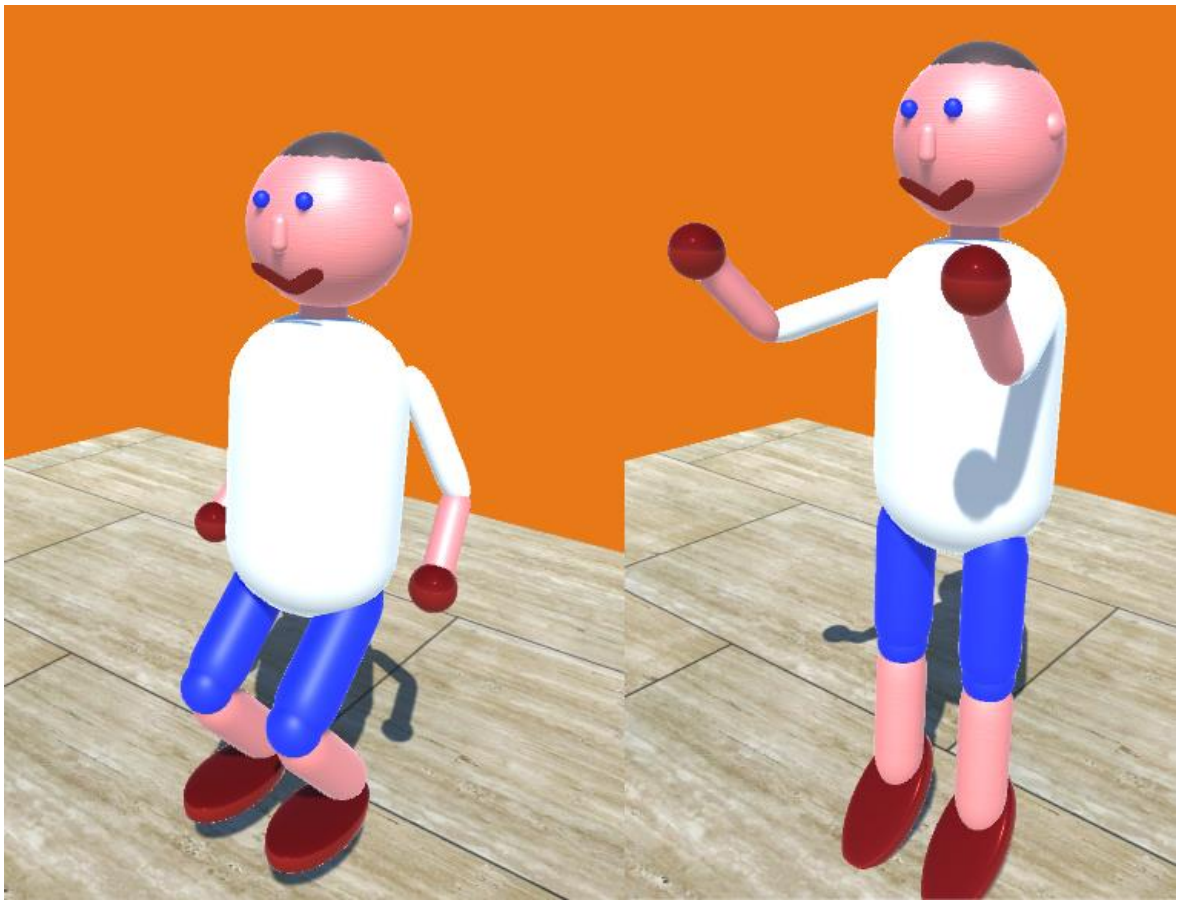
- Hodanje
- Trčanje
- Skakanje
- Primanje za zid

Hodanje je animacija koja se aktivira kada se iz stanja mirovanja počne kretati (tipke w-a-s-d ili strelice), a da se pritom ne drži tipka Shift. Brzina hodanja je 15, a animaciju čine blagi pokreti rukama i nogama naprijed i natrag.

Kada se korisnik kreće uz pritisnutu tipku Shift te ako nije na zidu, počet će se prikazivati animacija trčanja. Ova animacija slična je animaciji hodanja, ali su pokreti brži i intenzivniji (ruke i noge se pomiču pod većim kutom nego kod hodanja). Također, kada je pritisnuta tipka Shift, igrač se kreće dvostruko brže. Na Slika 4.7 prikazan je jedan okvir animacije „Trčanje“.

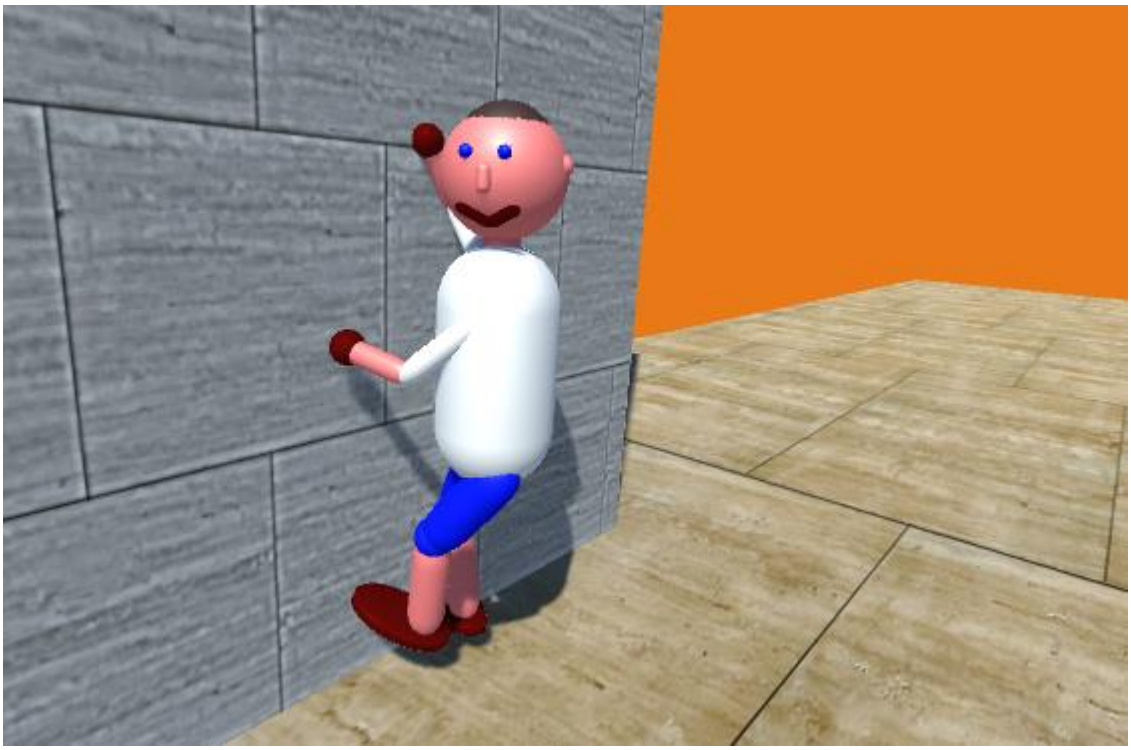
Slika 4.7 Trčanje

Kada je igrač u skoku (kada je pritisnuta tipka Space), pokrenut će se animacija skakanja. Ova animacija može se podijeliti na dva dijela: dio u kojem su modelu igrača koljena savijena i dio kada se odrazi od tla (prikazani na Slika 4.8).



Slika 4.8 Skakanje

Zadnja animacija, odnosno stanje, je kada se igrač primi za zid (Slika 4.9). Prilikom implementacije ove animacije, bilo je potrebno napraviti dvije .anim datoteke, jednu kada se igrač primi s lijeve strane te jednu s desne. Kako bi se primijenila točna animacija, napisana je skripta „FistCollision“ koja provjerava koji se dio modela igrača sudario sa zidom [12]. U slučaju da su to desna šaka ili desno stopalo, potrebno je primijeniti animaciju za desnu stranu te je identično za lijevu stranu. Ova skripta također koristi varijablu `timer` kako bi se izbjegli neželjene situacije (uloga varijable `timer` objašnjena je u poglavlju 3.4).

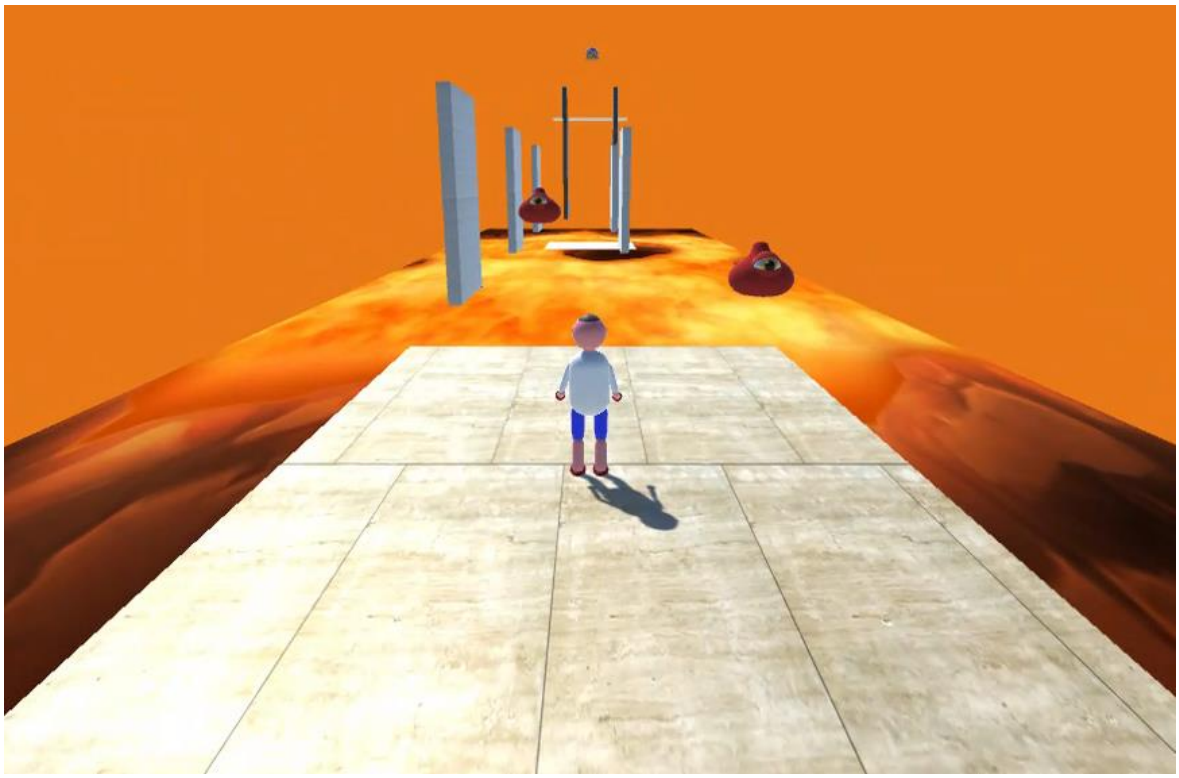


Slika 4.9 Primanje za zid

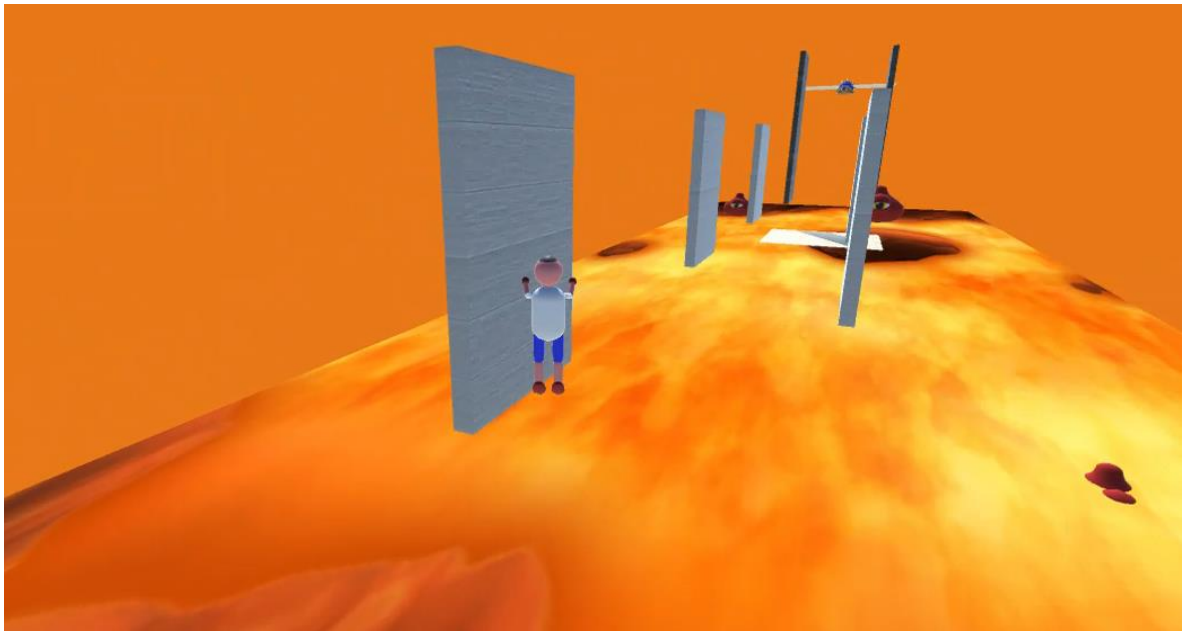
Sve navedene animacije rađene su po uzoru na pokrete iz stvarnog svijeta kako bi akcije igrača bile uvjerljivije.

5. Rezultati

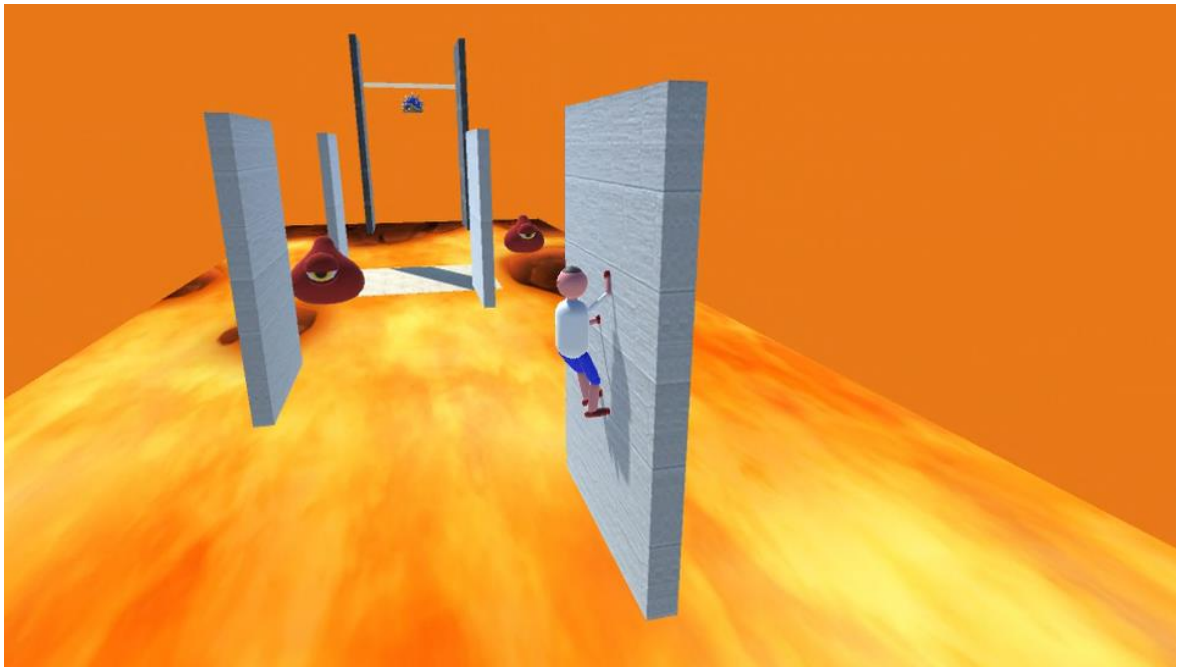
Ovaj završni rad rezultirao je videoigrom „Floyd“. Igrač može pomicati model igrača po platformi i skakati po zidovima, a radnje koje izvodi su popraćene prikladnim animacijama. U nastavku su prikazane slike iz videoigre. Na slikama se vidi model igrača u raznim animacijama, neprijatelji, zidovi, lava i platforme.



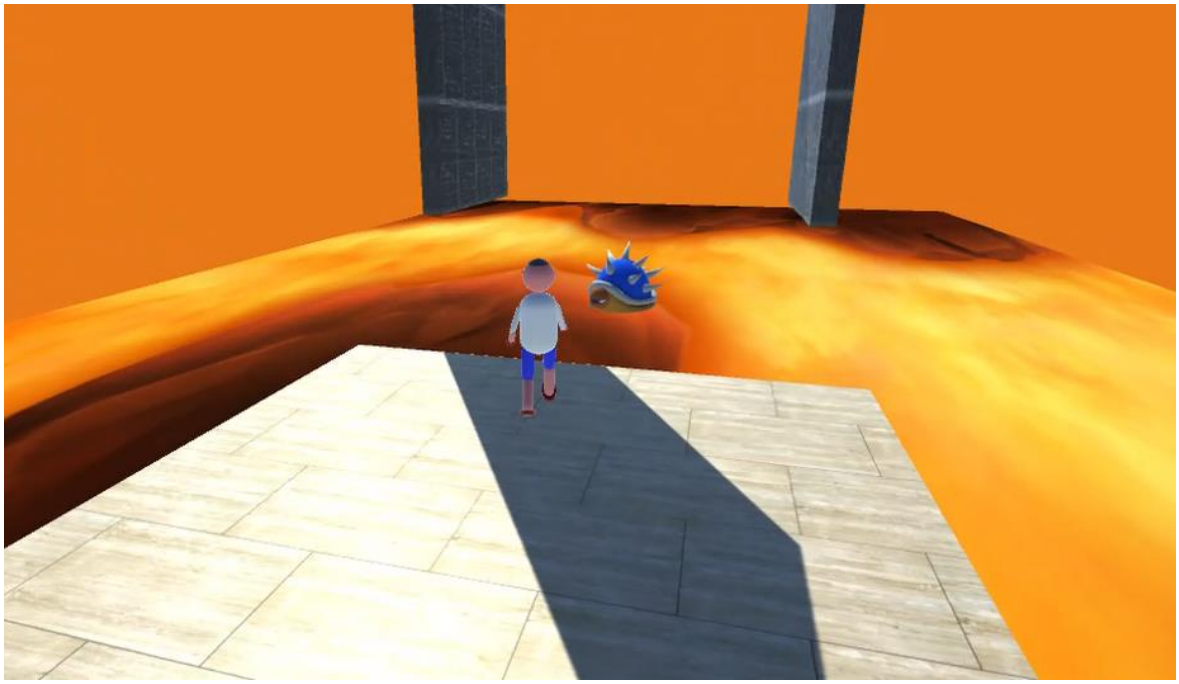
Slika 5.1 Model igrača na početnoj platformi



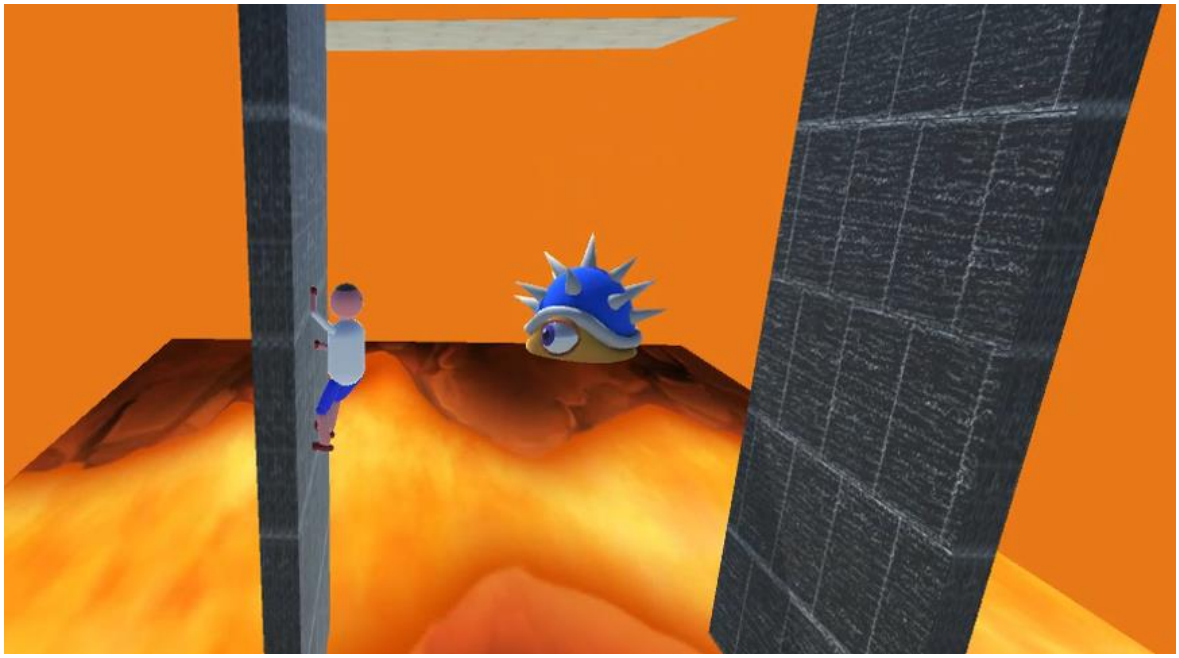
Slika 5.2 Model igrača skače na bijeli zid



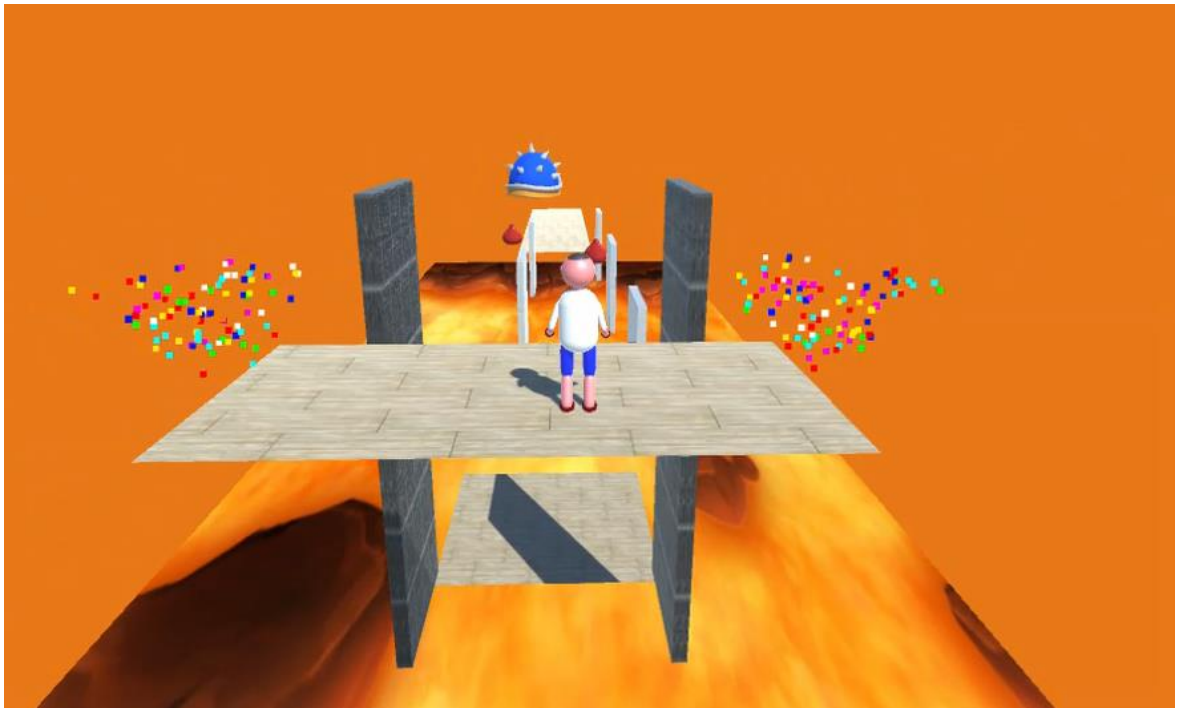
Slika 5.3 Držanje za zid i crveni neprijatelji



Slika 5.4 Model igrača na platformi prije crnih zidova i plavi neprijatelj



Slika 5.5 Model igrača na crnom zidu izbjegava plavog neprijatelja



Slika 5.6 Model igrača na ciljnoj platformi

Zaključak

Za izradu ovog završnog rada, napravljena je videoigra „Floyd“ koja se klasificira kao 3D platformer. Izgrađen je virtualni svijet koristeći osnovne oblike razvojne okoline Unity. Elementi ovog virtualnog svijeta su platforme i zidovi koje igrač koristi kako bi došao do ciljne platforme, a pritom izbjegava neprijatelje i lavu.

Napisane su skripte u programskom jeziku C# koje omogućavaju igraču hodanje, trčanje, skakanje i penjanje po zidovima. Preko skripti se također uvodi interakcija modela igrača s objektima virtualnog svijeta, kao što su platforme, zidovi, neprijatelji i lava.

Koristeći Unityjev sustav animacija, napravljene su pojedine animacije koje predstavljaju svaku od navedenih akcija koje igrač može izvoditi u svijetu. Animacije su napravljene kopirajući pokrete iz stvarnoga svijeta kako bi se podigla kvaliteta simulacije.

Literatura

[1] GameDesigning, *The Top 10 Video Game Engines*, GameDesigning, (2020, lipanj). Poveznica: https://www.gamedesigning.org/career/video-game-engines/; pristupljeno 11. lipnja 2020.

[2] Wikipedia, *Platform Game*, Wikipedia, (2020, svibanj). Poveznica: https://en.wikipedia.org/wiki/Platform_game#The_third_dimension; pristupljeno 11. lipnja 2020.

[3] Unity, *Materials - Unity Official Tutorials*, YouTube, (2013, travanj). Poveznica: https://www.youtube.com/watch?v=IFlXvDZezBQ; pristupljeno 10. svibnja 2020.

[4] Unity Technologies, *Rigidbody*, Unity Technologies (2020). Poveznica: https://docs.unity3d.com/ScriptReference/Rigidbody.html; pristupljeno 11. svibnja 2020.

[5] Brackeys, *START MENU in Unity*, YouTube, (2017, studeni). Poveznica: https://www.youtube.com/watch?v=zc8ac_qUXQY; pristupljeno 20. svibnja 2020.

[6] Unity Technologies, *Input.GetAxisRaw*, Unity Technologies, (2020). Poveznica: https://docs.unity3d.com/ScriptReference/Input.GetAxisRaw.html; pristupljeno 12. svibnja 2020.

[7] Unity Technologies, *Transform.LookAt*, Unity Technologies, (2020). Poveznica: https://docs.unity3d.com/ScriptReference/Transform.LookAt.html; pristupljeno 12. svibnja 2020.

[8] Brackeys, *2D Animation in Unity (Tutorial)*, YouTube, (2018, kolovoz). Poveznica: https://www.youtube.com/watch?v=hkaysu1Z-N8; pristupljeno 20. svibnja. 2020.

[9] Unity Technologies, *Working with Animations and Animation Curves*, Unity Technologies (2020). Poveznica: https://learn.unity.com/tutorial/working-with-animations-and-animation-curves; pristupljeno 21. svibnja 2020.

[10] Learn Everything Fast, *15 - Introduction to Animation Parameters in Unity*, YouTube, (2017, veljača). Poveznica: https://www.youtube.com/watch?v=xiL_7on-NP4; pristupljeno 26. svibnja 2020.

[11] Jonas Tyroller, *How To Animate In Unity 3D*, YouTube, (2019, kolovoz). Poveznica: https://www.youtube.com/watch?v=sgHicuJAu3g; pristupljeno 20.svibnja 2020.

[12] Unity Technologies, *Colliders*, Unity Technologies (2020). Poveznica: https://docs.unity3d.com/Manual/CollidersOverview.html; pristupljeno 26. svibnja 2020.

Sažetak

U radu „Primjena fizikalne simulacije u animaciji kretanja virtualnog lika u 3D igri“ opisuje se postupak izrade videoigre u razvojnoj okolini Unity. Napravljena videoigra uključuje fizikalnu simulaciju hodanja, trčanja, skakanja i penjanja po zidu. Detaljno se opisuje modeliranje 3D okruženja koristeći osnovne oblike u kombinaciji s materijalima i teksturama. Objašnjene su skripte i funkcionalnosti koje se postižu istima. Objašnjava se postupak izrade animacije koristeći polje vremena i krivulje, povezivanje napravljenih animacija preko prozora „Animator“, izrada prijelaza te parametri i na koji način oni utječu na prijelaze između animacija. Na kraju su prikazane slike iz videoigre gdje se jasno vide elementi virtualnog svijeta i animacije na modelu igrača.

Summary

The paper "Application of physical simulation in the animation of the movement of a virtual character in a 3D game" describes the process of making a video game in the Game Engine Unity. The created video game includes a physical simulation of walking, running, jumping and climbing a wall. Modeling the 3D environment using basic shapes combined with materials and textures is described in detail. The paper explains scripts and functionalities that are achieved by them. It also explains the process of creating an animation using Unity Dopesheet and using Curves, connecting created animations through the "Animator", creating parameters and how they affect transitions between animations.