Sveučilište u Rijeci – Odjel za informatiku

Preddiplomski jednopredmetni studij informatike

Lorena Mršić

Animacija sportskih akcija u računalnoj igri

Završni rad

Mentor: izv.prof.dr.sc. Marina Ivašić-Kos

Rijeka, Lipanj 2019.

Sadržaj:

1. Zadatak završnog rada

2. Sažetak

Tema ovog rada bila je proučiti postupke sinteze likova i koristeći Unity Engine napraviti sintezu akcija u košarci kao što je kretanje po terenu, uzimanje lopte te pravilan košarkaški šut.

Na početku je objašnjena sinteza likova: tko se njome bavi, koja je njena svrha te zašto je bitan njen razvoj u budućnosti.

Zatim se opisuje aplikacija Unity Engine u kojoj se radi, te tijek izrade simulacija sportskih akcija u košarci. Postupak simulacije, odnosno animacije lika je detaljno opisan zajedno s parametrima i skriptama koje su korištene prilikom simulacije.

Pri izradi animacije koristio se C# programski jezik i znanje iz objektno orijentiranog programiranja.

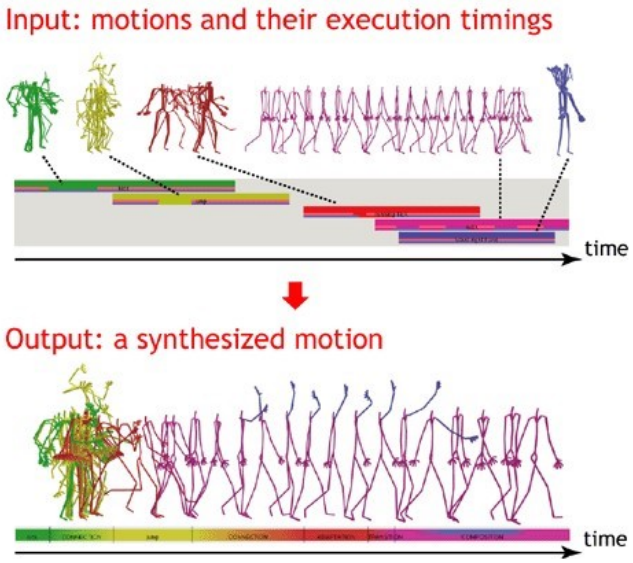
Ključne riječi: sinteze likova, animacije, Unity Engine, C#

3. Uvod

Likovi imaju važnu ulogu u gotovo svim vrstama računalnih igara ili interaktivnih simulacija. Ono što igrača privuče u igranje je dobra priča, a kroz tu priču ih vodi glavni lik. Upravljajući njime čovjek osjeća kontrolu i povezanost s virtualnom realnošću igrice, postaje dio igrice i proživljava sve uspone i padove [1]. U sjećanju nam ostaju likovi po kojima se pamte razne igrice, kao što su Super Mario i Luigi, Lara Croft, Pacman i mnogi drugi [2]. Tijek simulacije i likovi su najčešće ispričani na dva načina: kroz video zapise ili kroz animacije. Animacija je slika u pokretu. U prošlosti animacije su bile ručno nacrtane, sličica po sličica, a danas to sve rade računala [4]. Simulacija je približna imitacija nekih stvarnih situacija ili procesa. Ona prikazuje glavne karakteristike, ponašanja i funkcije modela objekta. Simulacija se može koristiti u raznim područjima kao što su testiranje, trening, edukacija i video igrice [5]. Kretanja u simulacijama su najčešće izrađena na način da se prethodno snimljeni klipovi kretnji(ili uz snimanje kretnji ili uz keyframing) spoje po potrebi. Takvu sintezu likova nazivamo Synthesis-By-Example – SBE [3].

Stvaranje kretnji odnosno animacija za likove u igricama je iznimno teško. Animiranje ljudskih likova je teško samo po sebi zato što su ljudski pokretni iznimno kompleksni i različiti, npr. pravilan košarkaški šut zahtijeva kretanje svakog dijela tijela pod različitim kutovima. Ljudi mogu napraviti puno stvari na puno načina, pa tako čak i najjednostavnije dnevne aktivnosti poput hodanja zahtijevaju kompleksnu koordinaciju pokreta pod određenim kutovima. Interakcija sa drugim predmetima je još kompliciranija zbog povezivanja skripti i fizike tijela sa predmetima. Kretnje likova moraju reagirati na situacije u igri, na dodire s predmetima iz okoline, bio lik u kontroli igrača ili ne. Zato je animiranje likova dug i težak proces [3].

Sinteza likova je proces spajanja više kretnji u jednu i pritom zadržavajući realan izgled. To najbolje možemo slikovno prikazati (Slika 1) : dodajemo različite kretnje na vremensku liniju i poredamo ih određenim redoslijedom. Bez sinteze likova svaki taj pokret bi se odvijao jedan po jedan, nepovezano. Ono što sinteza radi je spaja kretnje minimizirajući vidljivost prijelaza [26].



Slika 1 Sinteza

Animacije hodanja, skoka, mahanja itd., što je prikazano blokovima na slici 1 (Input) su animacije jednog lika, no one se mogu primijeniti na različitim likovima. Za rezultat na slici 1 (Output) je zaslužan algoritam koji nažalost nije dostupan, no ono što taj algoritam radi je dinamičkim programiranjem iz baze podataka kretnji pronalazi blokove pokreta koji se mogu smjestiti zajedno u nizu pokreta [27]. Radi se o blokovima pokreta čiji su počeci i krajevi slični te se zbog toga mogu smjestiti jedno pored drugoga da bi se kasnije spojili.

Dvije su glavne strategije za izradu lika, i to sinteza likova uz pomoć algoritama, te sinteza temeljena na primjerima, tj SBE. Sinteza likova uz pomoć algoritama se bavi izradom specijaliziranih tehnika baziranih na razumijevanju kretnji, dok SBE koriste opće algoritme koji zaobilaze to razumijevanje kretnji, bazirajući se na općim podacima prikupljenim jednostavnim procedurama[3].

U ovom radu primjenjuje se sinteza temeljena na primjerima tj. SBE.

4. Povijest SBE-a

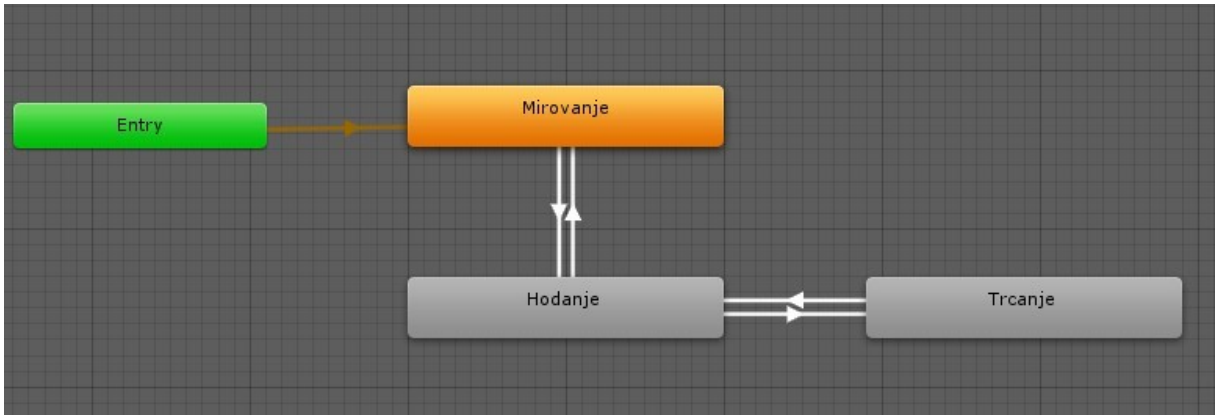
Malo je podataka o tome kada se točno razvila SBE, no već sredinom 1990-ih SBE postaje vidljiv u likovima tadašnjih igrica. Grafovi tranzicija likova su bili planirani unaprijed, i ručno stvoreni. Blendanje je tada korišteno da se stvore tranzicije, ali i da se ostvari veća kontrola. Tako možemo navesti prvu verziju igrice Mortal Kombat proizvedene 1992. godine. To je borilačka igra sa likovima koji svaki posebno imaju specifične napadačke pokrete. Prilikom izrade animacije lika snimao se određeni pokret kojeg je izvodila stvarna osoba (Slika 2), izrezala se figura osobe (Slika 3), te se potom ubacilo u igru (Slika 4), različite pokrete koje je osoba izvodila.

Slika 2 Snimanje lika

Slika 3 Izrezivanje i ubacivanje u igricu

Slika 4 Finalni izgled

Današnji likovi u igricama su puno napredniji, no oni i dalje koriste iste osnovne elemente izrade. Obično imaju određeni set akcija, gdje svaka akcija ima svoje parametre. Na primjer, lik može udarati rukama i nogama na raznim lokacijama te hodati različitim nagibom i brzinom. Takvi likovi i dalje koriste graf tranzicije (Slika 5) da bi specificirali koja akcija može slijediti nakon koje, ali ti grafovi također i opisuju granične parametre kretnji i ne izrađuje se ručno već koristeći računalne alate. Na slici 5 dan je primjer takvog grafa u Unity-u za animaciju hodanja i trčanja. Iz grafa se može vidjeti da igrač iz mirovanja može preći u hodanje i obrnuto, te da iz hodanja može preći u trčanje, međutim igrač ne može iz mirovanja preći u akciju trčanja jer ne postoji tranzicija između mirovanja i trčanja.



Slika 5 Graf tranzicije u Unity-u

Kretnje za parametre akcija mogu biti stvorene na različite načine, kao što je blendanje primjera ili stvaranje promjena na slojevima, kao na primjer za igru prikazanu na slici 4: igrač ima dva grafa tranzicije, jedan se odnosi na hodanje i trčanje, a drugi na vađenje i spremanje noža. Mi želimo da igrač nakon što izvadi nož i dalje može trčati i hodati, zato se uređuje sloj vezan za nož tako da se animacija odvija samo za gornji dio tijela, dok donji i dalje koristi animacije hodanja i trčanja. Također, sve više se koristi motion capture [3].

Motion capture je proces snimanja pokreta. Najčešće se koristi u vojsci, robotici, filmskoj industriji te sportu. Osoba na sebi nosi odijelo koje se sastoji od markera na zglobovima (Slika 6) kako bi se odredila udaljenost i kut između svakog markera i stvorio realističan prikaz ljudskog tijela. Snimi se određena kretnja s kamerom, sa što više okvira u sekundi. Zatim se odstranjuje pozadina i ostaju samo markeri koji se povezuju i stvaraju kostur čovjeka. Ti prikupljeni podaci se povezuju sa 3D modelom tako da bi on mogao izvoditi iste kretnje kao i čovjek koji ih je izveo [6]. Razlika između snimanja pokreta danas i nekad je ta što se danas snima pokret tijela i taj isti pokret se može izmijeniti i izvesti na bilo kojem modelu, a nekad se slikao i izrezivao, bez mogućnosti za izmjenu pozicije niti rotacije.

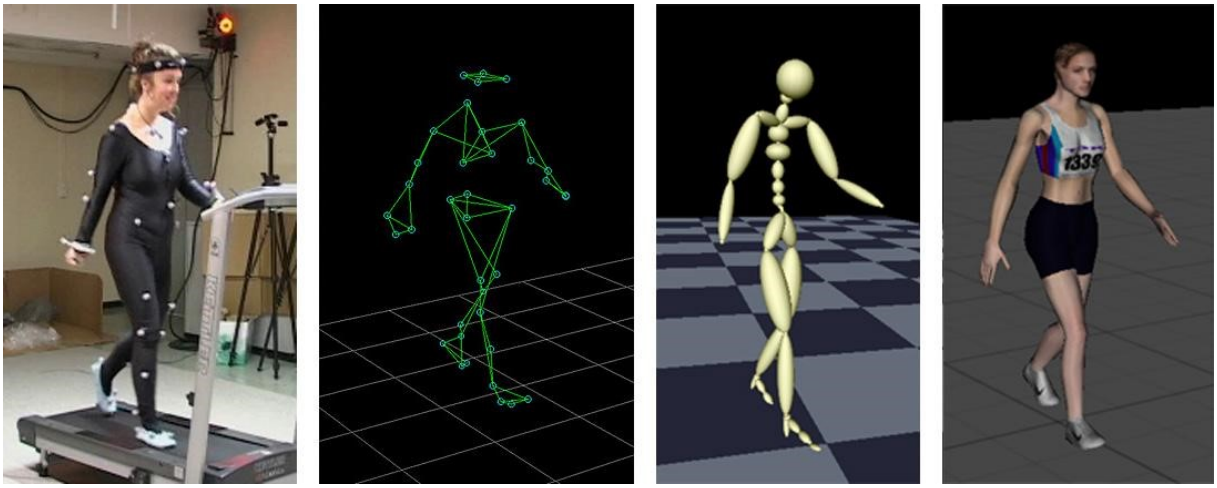
Slika 6 Nogometaš Messi prilikom snimanja pokreta za igricu PES

Ključ praktične primjene SBE pristupa je u pažljivom planiranju i ručnoj pripremi podataka kako bi primjeri kretnji mogli funkcionirati zajedno. Prvo se biraju kretnje koje će se koristiti, planiraju se pokreti kako bi se na njima moglo koristiti blendiranje i tranzicija ukoliko je potrebno, te se biraju tranzicije i blendiranje kako bi se olakšala kontrola nad likom. Blendiranje je spajanje dvije ili više animacija u jednu sa što manje vidljivim prijelazom, kao na primjer trčanje u raznim smjerovima. Tranzicija je prijelaz iz jedne animacije na drugu kao što je tranzicija iz stanja mirovanja u hodanje (Slika 5) [28]. Na kraju se izvode ti pokreti kako bi se stvorili primjeri koji se mogu povezati [3].

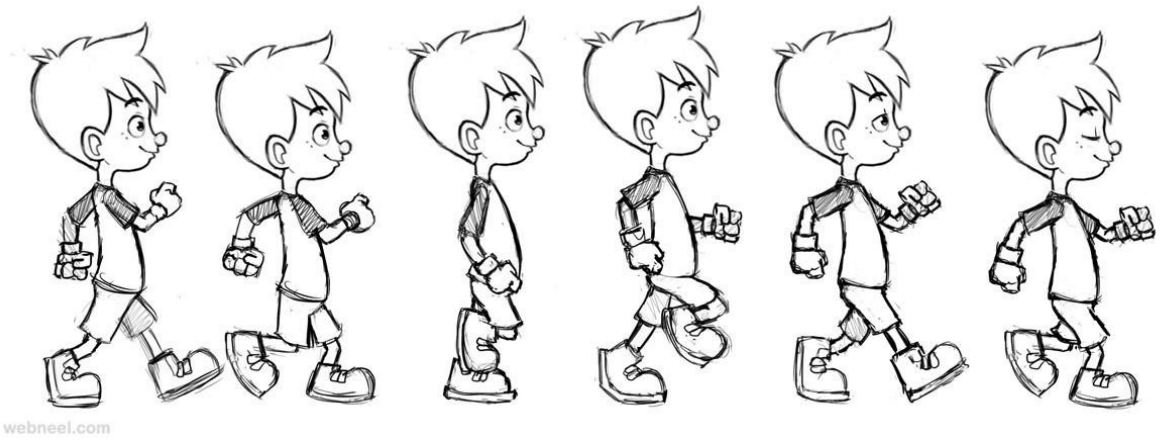
5. Zašto SBE?

Razlozi za korištenje SBE pristupa su pristupačnost, jednostavnost i brzina izrade kretnje.

Glavna ideja Synthesis-By-Example pristupa je da su kretnje prikupljene od nekog vanjskog izvora, kao na primjer snimanje hodanja (motion capture) neke osobe (Slika 7) ili stvaranje istog hodanja u animatoru koristeći keyframe-ova (Slika 8).



Slika 7 Motion capture



Slika 8 Keyframe

Prikupljene kretnje su podaci koji se ako je potrebno mogu ponavljati. Lik može izvršavati puno različitih kretnji i akcija, npr. košarkaš može hodati, trčati, skakati, driblati, ukrasti loptu itd. Raznovrsnost košarkaških kretnji i animacija je dobro prikazana u košarkaškoj igrici NBA 2K[1].

Prilikom izrade animacije kreator može izraziti svoju kreativnost, ne mora napraviti animaciju košarkaškog šuta onako kako to nalažu pravila šuta, već kako on želi. Također

[1] NBA 2K je video igrica simulacije košarke, te jednom godišnje izlazi nova verzija igrice započevši od 1999.

moguće je uzeti postojeću animaciju te ju urediti po vlastitom ukusu, pomaknuti ruku malo više u desno, nogu u lijevo itd. Sve je u rukama osobe koja animira.

SBE pristup spaja više primjera kretnji uz pomoć blendiranja. Blendiranje se bavi stvaranjem tranzicije s jedne kretnje na drugu ili spajanje dvoje kretnji u jednu [6]. Traženje kretnje hodanja u čučnju ili tranzicije između hodanja i stoja na rukama može biti jednako teška kao i snimanje tih kretnji. No, ukoliko pronađemo kretnju koja je dovoljno slična kretnji koju želimo stvoriti, uz malu promjenu na postojećoj možemo stvoriti novu animaciju i time izbjeći snimanje pokreta. Npr. ako smo pronašli animaciju košarkaškog šuta gdje igrač zavrti loptu, lupi par puta pod, podigne iznad čela i šutne, a mi želimo da naš igrač podigne loptu iznad čela i šutne, onda možemo editirati i preraditi postojeću animaciju da se ponaša onako kako želimo. Potrebno je pronaći trenutak u kojem je lopta ispred igrača u rukama prije nego ju igrač zavrti i spojiti ga s keyframeovima u trenutku kada završava driblanje lopte i kreće podizanje lopte iznad čela. Tim postupkom stvorili smo novu animaciju bez snimanja kretnje od nule.

Gotovo svi SBE se temelje na istim elementima izrade, kao i sama sinteza likova:

- kretnje se nižu – prikupljene kretnje stavljaju se jedna pored druge na vremensku crtu određenim redoslijedom
- spajaju se – pronalazi se određeni moment u kojem su dvije kretnje u najsličnijoj poziciji tijela i tamo se stvara prijelaz iz jedne kretnje u drugu
- stvaraju se slojevi – u određenom trenutku želimo da gornji dio tijela radi jedno, a donji drugo, stvaramo dva sloja koji kontroliraju svaki svoj dio.
- transformiraju – na kraju potrebno je kretnju pozicionirati prostorno i vremenski, npr. ukoliko noge tijekom kretnje 'propadaju' u zemlju, potrebno je malo podignuti animaciju, isto tako je i sa vremenom, ukoliko je nešto prebrzo, usporiti i obrnuto. [3].

6. Unity Engine

U početku tvrtke koje su stvarale igrice morale su razviti i vlastiti game engine za razvoj igre što je bilo skupo, i uzimalo je puno vremena.

Nakon nekoliko desetaka godina Unity Technologies 2005. godine razvija jedan od najpopularnijih game engine-a današnjice: Unity game engine. Objavio ga je prilikom svjetske konferencije za developere Apple-a kao službeni game engine Mac OS X-a, no uskoro je postao dostupan i za Windows. Unity game engine danas podržava više od 25 platformi (neke od njih su PlayStation VR, Xbox One, Linux, Android...) .

Cilj Unity-a je bio pojednostaviti razvoj digitalnih igara, skratiti vrijeme razvoja, smanjiti trošak, ali i omogućiti besplatno korištenje enginea za razvoj igara. Unity game engine je baziran na C# programskom jeziku.

Iako se Unity najviše koristi za izradu video igara, to nisu jedine industrije koje su ga prihvatile. Unity se koristi i u izradi filmova, arhitekturi, inženjerstvu, građevini te automobilskoj industriji jer se može koristiti za stvaranje 2D i 3D simulacija, te simulacija virtualne stvarnosti i proširene stvarnosti [10,11].

7. Izrada simulacije – Košarka

7.1. Ideja

Ideja je bila napraviti simulaciju košarkaških akcija koristeći ekspertno znanje prikupljeno višegodišnjim treniranjem košarke i Unity alat za izradu igara.

Sljedeće je trebalo smisliti što će sve biti potrebno za izradu te simulacije, točnije od kojih će se dijelova ona sastojati.

Tabela 1 Dijelovi simulacije

OBJEKTI	FUNKCIJE	OPIS RADNJI
Igrač	• Pokretanje igrača • Animacija igrača • Podizanje lopte • Šut	Igrač treba hodati u raznim pravcima, dok ne dođe do stalka sa loptama gdje podiže loptu i šutira. Čim se podigne jedna lopta, druga se stvori.
Lopta	• Nalazi se u rukama igrača • Odvoji se • Leti • Prolazi kroz mrežicu • Odbijanje od poda	Lopta je na stalku. Prilikom uzimanja ona mora biti u rukama, podignuti se iznad čela te odletjeti prema košu kada igrač odluči šutnuti. Tada prolazi kroz mrežicu i cilj je postignut.
Teren	• Izgled terena • Ograde • Koš • Obruč • Stalak	Teren se sastoji od polovice košarkaškog igrališta, koša, stalka za loptu te ograde kako lopta i igrač nebi mogli napustiti igralište.
Meni	• GUI • UI	Izrada start meni-a pretežito zbog vizualnog izgleda. Korištenje UI-a za poruke tijekom simulacije.

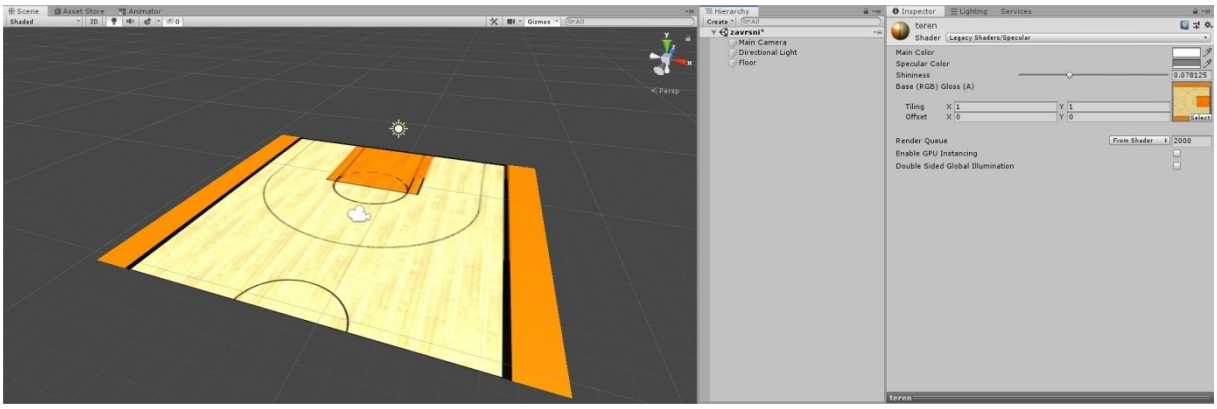
Nakon što je plan izrade napravljen, može se krenuti s izradom simulacije.

7.2. Izrada objekata

Redoslijed izrade simulacije uključuje izradu terena po kojem se igrač kreće, zatim izrada statičnih objekata (koš, obruč, ograde), te izrada igrača i lopte.

7.2.1. Izrada terena

Dodajemo 3D objekt i nazivamo ga Floor. Po default-u svaki objekt koji dodamo je bijele boje, no objekt treba prikazivati košarkašku polovicu terena, prema tome stvaramo materijal kojem dodajemo sliku polovice terena, te taj materijal [24] dodajemo objektu Floor (Slika 9). Kako objekti ne bi prolazili kroz teren potrebno je na Floor dodati Box Collider koji će objektu dodati fizička svojstva i onemogućiti da objekti prolaze kroz njega.



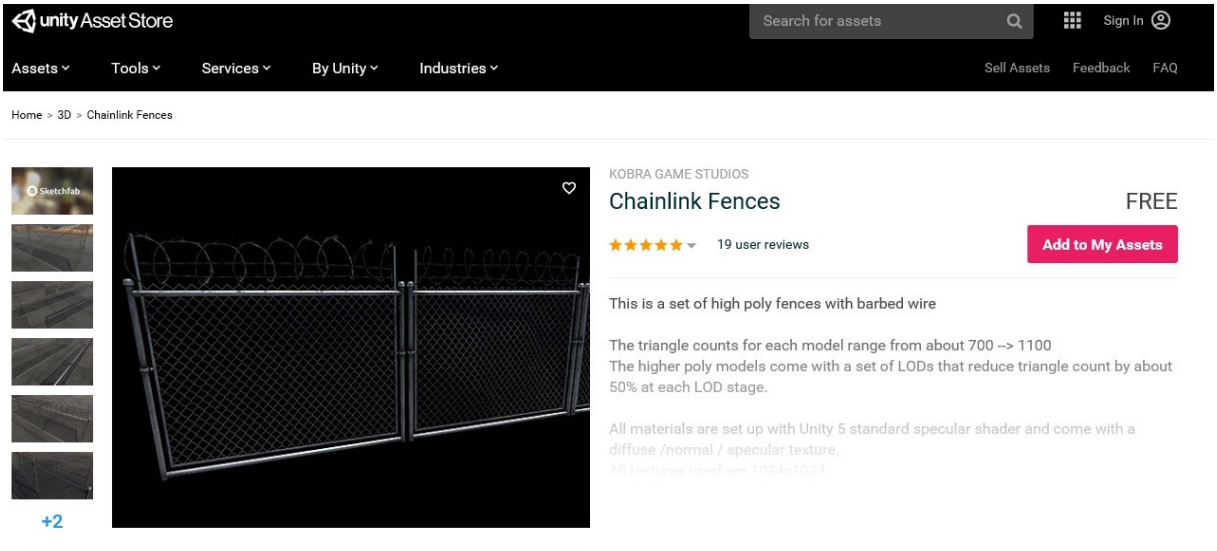
Slika 9 Prikaz plane-a u sceni i Floor materijala

Kako lopta i igrač ne bi padali s ruba terena potrebno je napraviti prepreke, u ovom slučaju dodane su ograde.

Za generiranje više objekata istog tipa koristimo Prefabse i modele. Prefabe koristimo i za NPC (non-player-character) GameObject-e. NPC su likovi unutar igrice koje ne možemo kontrolirati [29]. Bilo koju promjenu koju napravimo na Prefab-u automatski se primjenjuje i na sve ostale Prefab-e tog objekta unutar projekta, što štedi vrijeme, jer se ne mora uređivati svaki objekt posebno. Modeli su jako slični Prefab-ima, te se čak i nazivaju Model Prefabs. Oni najčešće sadrže neki 3D model kao što su likovi, zgrade ili dio kućanstva. Modeli također mogu sadržavati i Animation data, što znači da se modeli mogu animirati [15,22].

7.2.2. Izrada ograda

Objekti poput ograda, stolova, stolica, likova, auta i slično mogu se preuzeti iz Asset on-line trgovine Unity-a [12] ili se mogu modelirati uz pomoć programa kao što je Blender [8]. Unity Asset Store sadrži 3D i 2D modele, audio isječke (zvukovi auta, glasanje životinja, zvukovi pušaka…), predloške za kontroliranje igrača, alate za animacije, modeliranje i slično. Za potrebe ovog rada koristili su se modeli ograde iz Unity Asset Store-a (Slika 10).

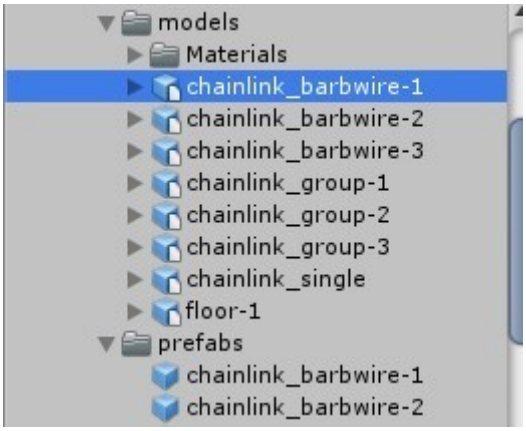


Slika 10 Unity Asset Store i ograde

Nakon što se u trgovini Asset Store pronađe objekt koji se želi uključiti u projekt, doda ga se u projekt s "Add to My Assets'.

Asset Store je službeni store Unity-a, no to nije jedina stranica na kojoj se mogu preuzeti objekti i modeli. U ovom projektu se obruč preuzeo sa stranice Free3D [13] i igračica sa stranice RenderHub [14].

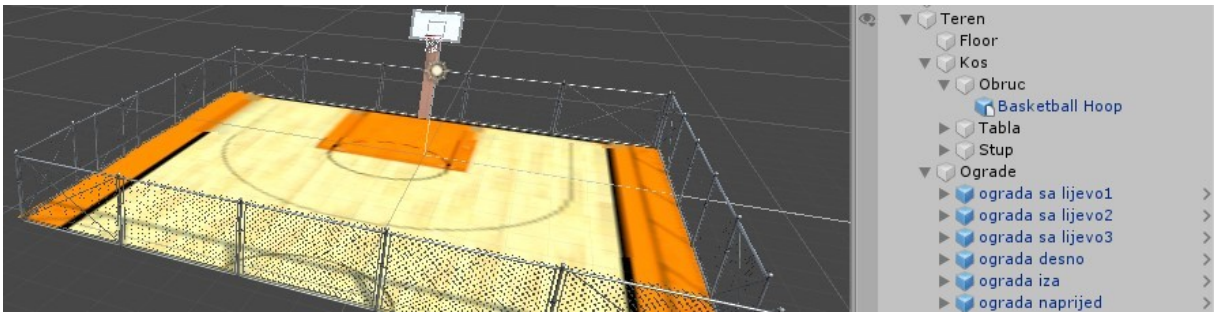
Preuzeti modeli dolaze u obliku Prefab-a ili Model-a, ili oboje kao u slučaju ograda(Slika 11). Nama će ograda služiti kao statičan objekt i zbog toga ćemo se koristiti prefabom ograde a ne modelom, dok kod dodavanja igrača se koristimo modelom zbog mogućnosti animiranja.



Slika 11 Modeli i prefabi ograda

Na isti način kao i ogradu dodali smo koš, tablu, obruč i sve postavljamo tako da vizualno što više podsjeća na realni košarkaški teren.

Kod projekata je bitna dobra organizacija kako se ne bi pogubili među objektima koje smo stvorili. Pa tako u projektu stvaramo Empty objekt (gledajte na to kao na prazan folder) i nazivamo ga Teren, i taj Teren postaje roditelj svim dijelovima terena. Također stvaramo i Empty pod nazivom Kos koji će sadržavati obruc, tablu i stup. Tako se stvara hijerarhija objekata (Slika 12).



Slika 12 Hijerarhija Teren-a i Kos-a skupa sa primjerom modela i prefaba

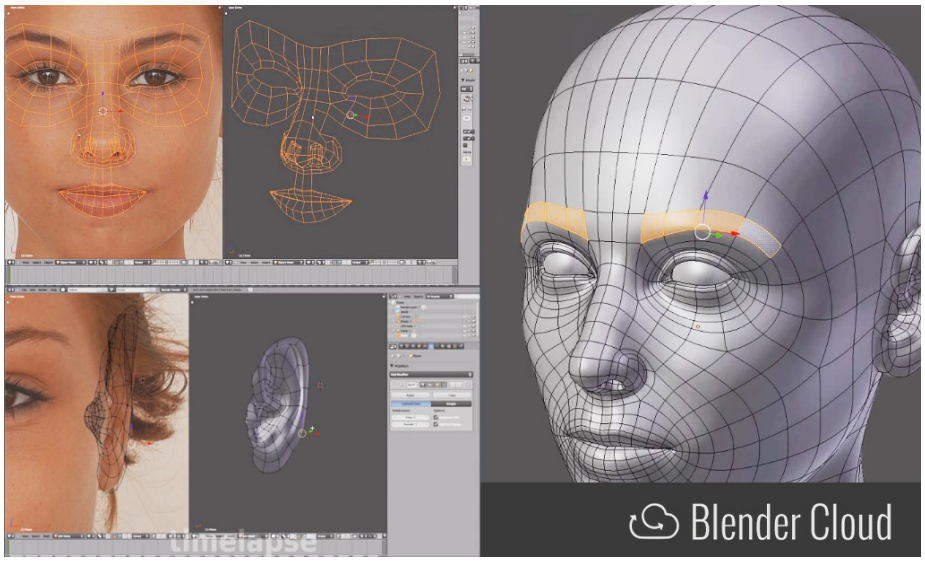
Veza između roditelja i djeteta je takva da dijete postaje dio roditelja, pa tako Obruc postaje dijete Kos-a. Pozicija Obruc-a se sada mijenja ovisno o poziciji Kos-a. Ukoliko pomaknemo Kos, sva njegova djeca se pomiču skupa s njime.

7.2.3. Igrač

Modeli su najčešće likovi (eng. Characteri), pa je tako i naš igrač model prefab.

Pošto želimo da se igrač kreće po terenu, hoda u svim pravcima, podiže loptu i šutira, potrebno je imati model spreman za sve te animacije, odnosno u potpunosti namješten model. Model koji ima izrađenu i povezanu mrežu tijela i kostur nazivamo potpuno namješteni model. Svaka izrada igrača spremnog za animacije sastoji se od tri djela: modelling, rigging and skinning.

- Modelling – modeliranje je zapravo stvaranje 'mreže' tijela igrača. Raznim trokutićima, kvadratima i drugim geometrijskim likovima cilj je se što bolje prikazati ljudsko tijelo, ili bilo koji predmet koji se modelira. Najčešće se modelira u T pozi (Slika 15), jer ona omogućava bolje modeliranje nekih dijelova ruke koji se inače ne vide, ali i također olakšava sljedeći korak. Modeli se izrađuju u programima Blender (Slika 13), Maya, 3DSMax…

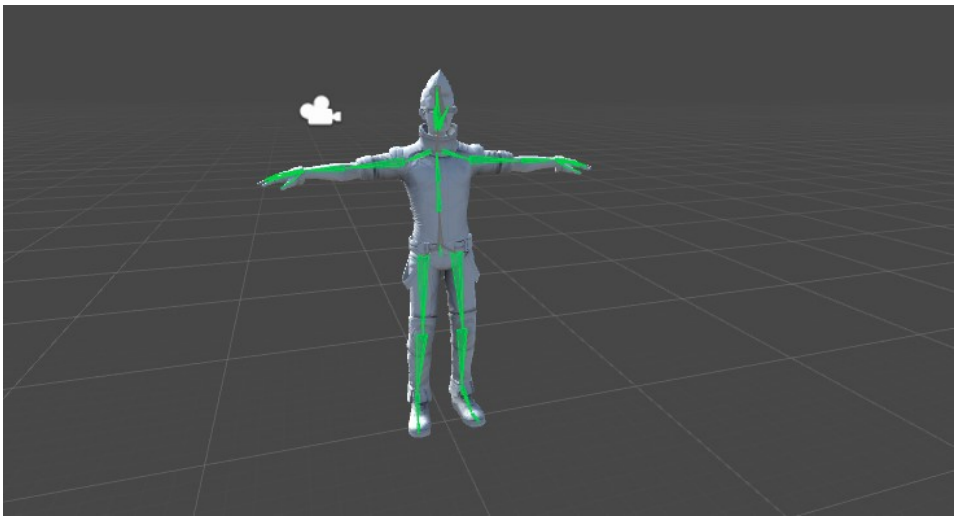


Slika 13 Modelling

- Rigging (Slika 14) – to je izrada kostura napravljenog od zglobova koji omogućavaju kretanje u igrici. Postoje različite hijerarhije povezivanja kostura a neke od najčešćih su:

  HIPS – spine – chest – shoulders – arm – forearm – hand
  HIPS – spine – chest – neck – head
  HIPS – UpLeg – Leg – foot – toe – toe_end

  U svim slučajevima pojavljuje se HIPS kao prvi korijenski čvor hijerarhije kostura jer pravilo izrade lika spremnog za animaciju nalaže da korijenski element kostura budu kukovi (HIPS).



Slika 14 Rigging

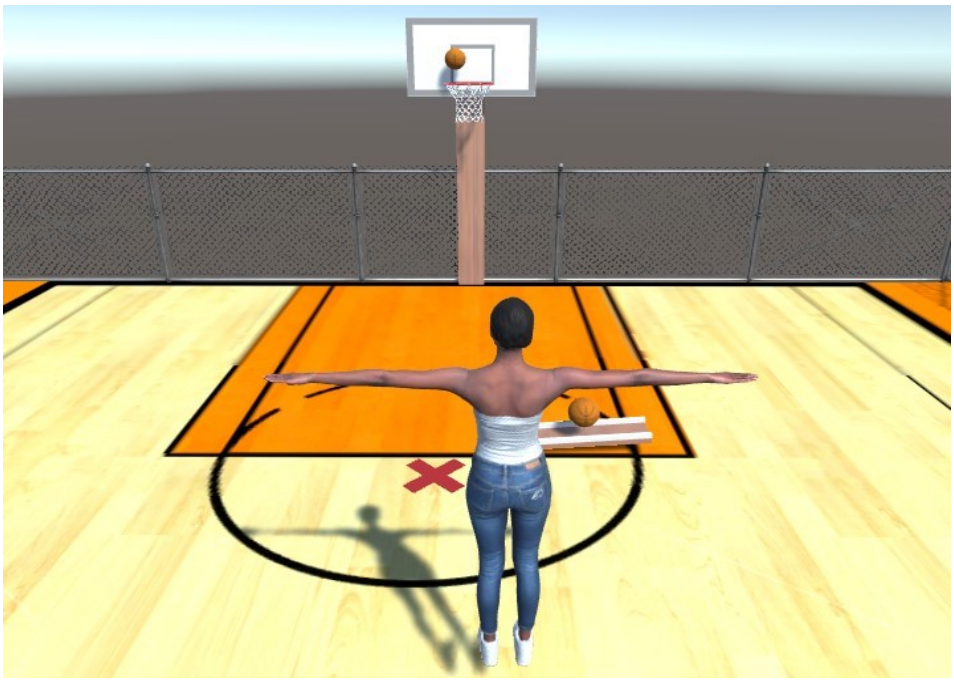
- Skinning – je proces povezivanja mreže i kostura. [16]

Kada smo pronašli model koji odgovara našim potrebama, dodajemo ga u projekt, U ovom slučaju u T pozi (Slika 15).

Slika 15 T poza lika

Bitno je da liku pridružimo Rigidbody komponentu Unity-a koja omogućava da objekt ima svoju masu, trenje i brzinu te da na njega utječe fizika, poput gravitacije.

Lopta je drugi važan objekt kojeg dodajemo na način da na meniju odaberemo novi 3D objekt → Sphere, i na nju dodajemo materijal lopte, te također Rigidbody. Nakon izrade stalka za loptu i oznake X na podu gdje želimo da košarkašica podigne loptu, dolazimo do finalnog izgleda scene (Slika 16).

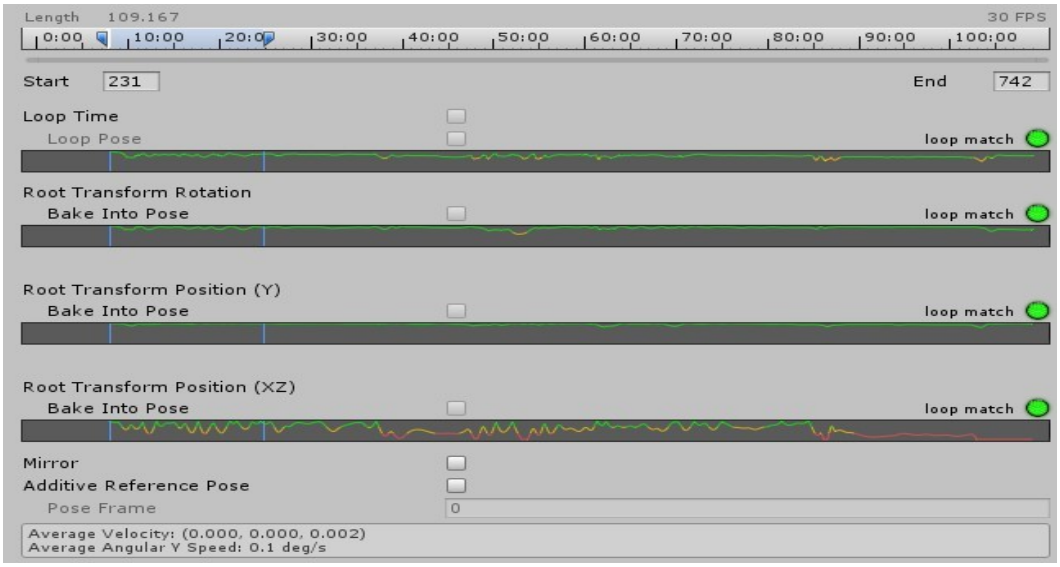


Slika 16 Finalni izgled scene

7.3. Animacije i Animator

Kako bi se omogućio brži razvoj igara, Unity omogućuje preuzimanje likova ali i gotovih animacija poput trčanja, hodanja. Glavni autori animacija su i Roidz i Deozaan koji na Unity Asset Store-u postoje pod nazivom cMonkeys [30]. Oni su za korištenje u Unity-u objavili ogromnu biblioteku motion capture animacija, sa preko 2000 animacija, od dnevnih aktivnosti do sporta, razgovora, gesti…

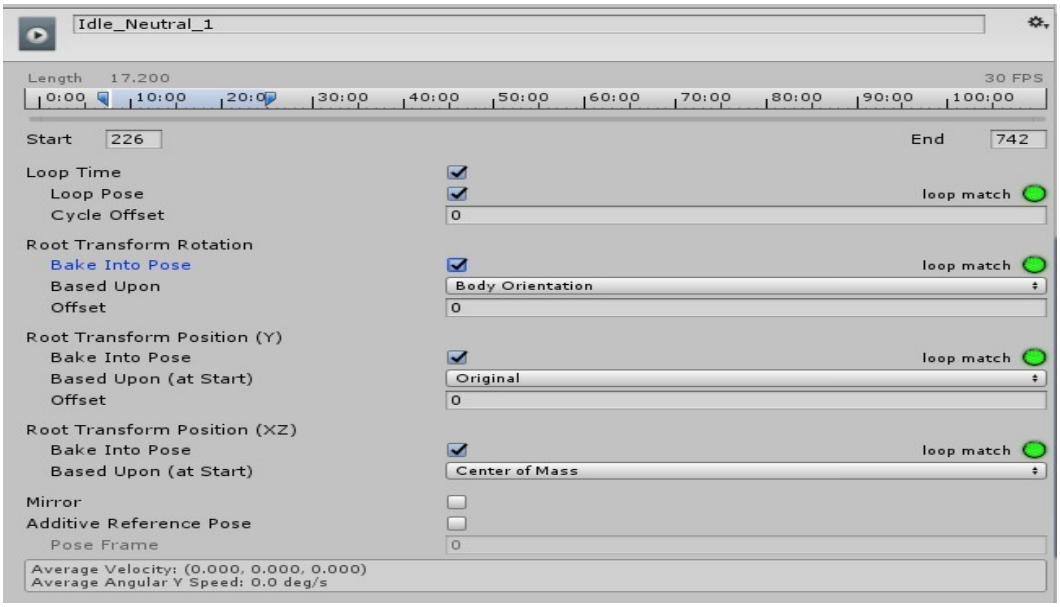
Nakon pronalaska animacija koje želimo koristiti, potrebno ih je urediti da izvršavaju akcije koje smo planirali. Neka animacija npr. animacija mirovanja u mjestu može trajati jako dugo. Pošto će se naša animacija mirovanja ponavljati, najvažnije je bilo pronaći duljinu u kojoj je ponavljanje optimalno. Kada želimo promijeniti početak/završetak animacije prikazuju nam se grafovi čije su linije obojane zelenom, narančastom i crvenom bojom (Slika 17).



Slika 17 Graf animacije

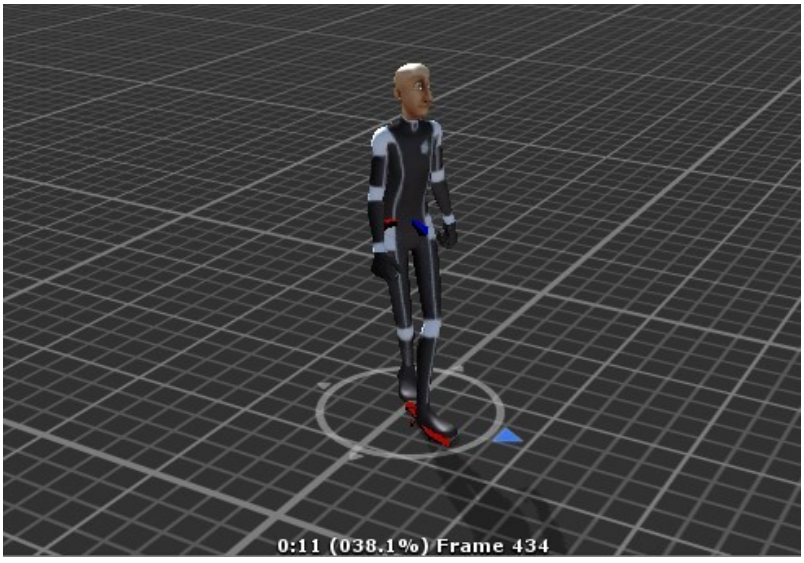
U dijelovima u kojima je linija zelene boje, ponavljanje je optimalno, gdje je narančasto je srednje optimalno dok linija crvene boje prikazuje dio animacije gdje ponavljanje nije optimalno. Ukoliko odaberemo neku duljinu u kojoj nam nisu svi kružići zelene boje najvjerojatnije ponavljanje animacije neće izgledati najbolje, neće biti najbolje povezano. Primjer animacije koju želimo ponavljati je stanje mirovanja ili hodanje.

Nakon toga da bi se animacija ponavljala potrebno je označiti opciju Loop Time i Loop Pose. Sljedeća bitna stvar je Bake Into Pose i Offset (Slika 18).

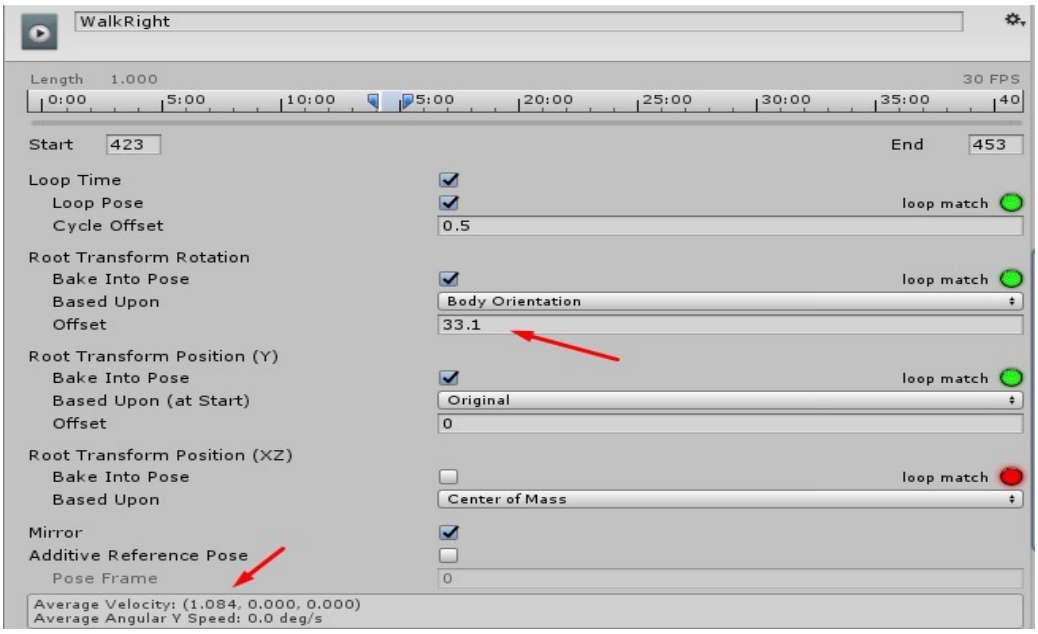


Slika 18 Animacija mirovanja

Opcija Bake Into Pose nam omogućava da animacija ne utječe na rotaciju i poziciju objekta. No ponekad sama ta opcija nije dovoljna da bi postigli ono što želimo, pa tako koristimo i Offset. Offset omogućava da manualno podesimo rotaciju i poziciju animacije, kao npr. kod skretanja u desno (Slika 19) [17]. Rezultate offseta vidimo na dnu pod 'Average Velocity' (Slika 20).



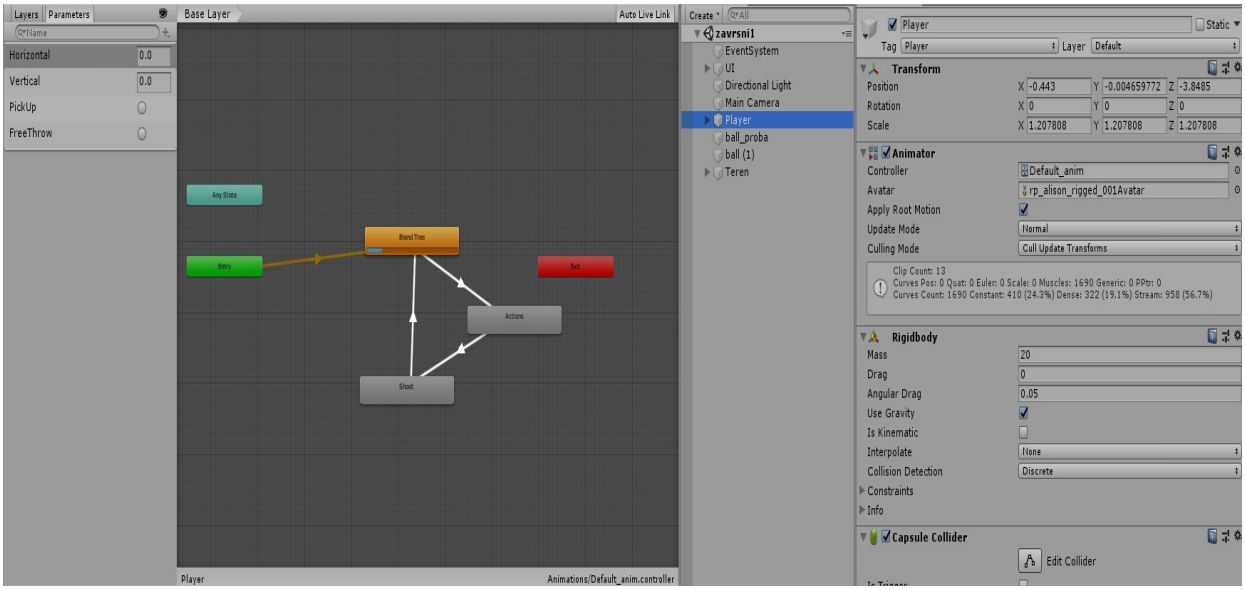
Slika 19 Prikaz animacije skretanja u desno



Slika 20 Animacija skretanja u desno i offset

Nakon što uredimo sve animacije onako kako želimo, potrebno ih je spojiti s igračem i posložiti ih na način na koji to ima smisla.

Kako bi to mogli na objekt igrača dodajemo komponentu pod nazivom Animator. Zatim stvorimo i Animator Controller (Slika 21). Animator Controller je jako sličan dijagramu stanja i grafu tranzicija, on koristi jedan ili više stanja koji pokazuju koja se animacija trenutno pokreće. Stanja se sastoje od animacija i povezane su tranzicijama. Stanja su kontrolirana parametrima unutar tranzicija koji dobivaju svoje vrijednosti preko skripti [18]. Narančastom bojom je označeno početno stanje, u našem slučaju to je stanje mirovanja.



Slika 21 Animator Controller

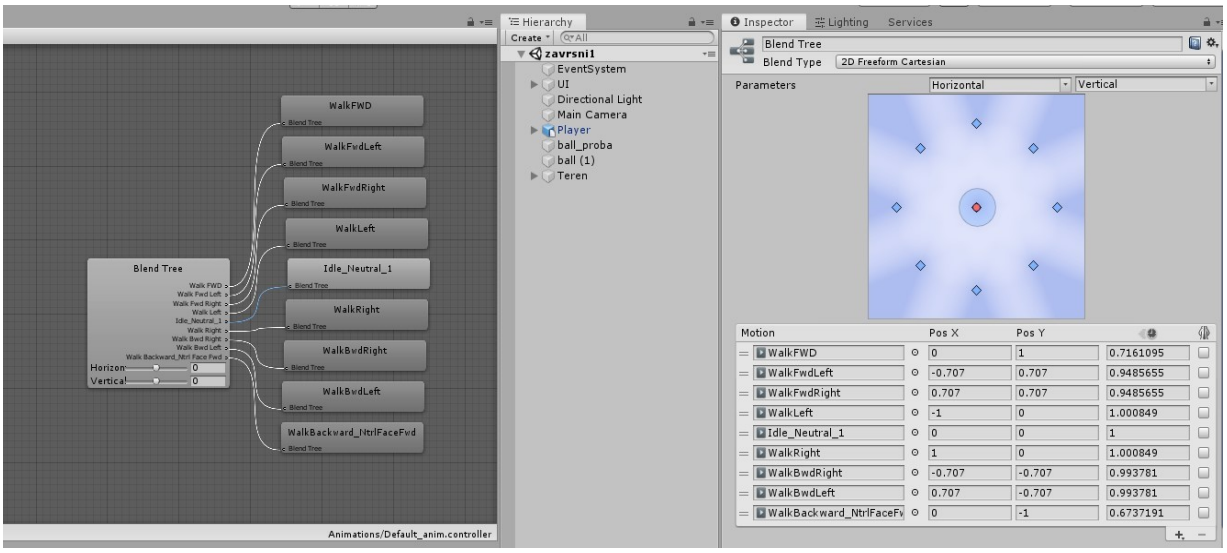
7.3.1. Primjer SBE sinteze

Napravivši izmjene na postojećoj animaciji i stvorivši Animator Controller prikazali smo jedan proces sinteze likova koristeći algoritam ugrađen u Unity. Prikupili smo animacije mirovanja, hodanja, podizanja lopte i šuta. Skratili smo ih, uredili offset, poziciju i rotaciju tijela, te odabrali keyframe u koji ćemo postaviti prijelaz. U Animator Controller-u smo ih poredali određenim redoslijedom. Povezali smo ih tranzicijama s određenim uvjetima i ostvarili cilj sinteze.

Kao primjer sinteze likova unutar Unity-a možemo navesti i ponavljanje animacije. Imamo animaciju mirovanja, koju želimo ponavljati dok god se likom ne upravlja. Upravo graf animacije (Slika 17) prikazuje u kojem trenutku možemo započeti i završiti animaciju tako da bi se ona ponavljala realistično, odnosno da bi prijelazi između kraja i ponovnog početka bili oku nevidljivi.

7.3.2. Blend Tree - blendanje

Ponekad kad animiramo želimo spojiti dvije ili više animacija, npr. imamo animacije za trčanje naprijed, desno i lijevo, a želimo animaciju kojom idemo lagano u lijevo ili lagano u desno. Opcija Blend Tree se koristi da bi se to postiglo. Dok smo u Animator Controlleru desnim klikom dodamo novo stanje → From New Blend Tree i kliknemo dvaput na njega. Blend Type određuje vrstu Blend Tree-a, parametri određuju kretanje po X i Y osi, a nova stanja dodajemo pod Motion. Za svaku animaciju posebno određujemo njezinu poziciju na X i Y osi. Sve to je prikazano na dijagramu iznad Motion-a (Slika 22).



Slika 22 Blend Tree

7.4. Skripte

Skripte su nužan dio Unity projekta. One upravljaju sa gotovo svim segmentima naše simulacije i u ovom projektu ih koristimo za upravljanje radom kamere, kretanje i animiranje pokreta igrača.

7.4.1. Kamera

Kamera prikazuje kako će igra izgledati tijekom igranja. Kako bi kamera pratila igrača potrebno je stvoriti skriptu sa varijablom igrača. U funkciji Start() provjeravamo udaljenost kamere i igrača i tu vrijednost spremamo u Vector3 varijablu offset. Zatim u LateUpdate() funkciji pomičemo kameru skupa s igračem (Slika 23). Zašto nismo koristili uobičajenu Update() funkciju iako se poziva jednom tijekom svakog frame-a kao i LateUpdate()? Za kamere koje prate igrača i prikupljaju zadnja poznata stanja najbolje je koristiti LateUpdate(), s njime smo sigurni da će se, nakon što se svi objekti procesuiraju u update-u, kamera pokrenuti zajedno s igračem.

```
public class CameraScript : MonoBehaviour
{
    public GameObject player;
    private Vector3 offset;

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

    void LateUpdate()
    {
        transform.position = player.transform.position + offset;
    }
}
```

Slika 23 Skripta kamere

7.4.2. Blend Tree

Kako bi Blend Tree animacije povezali sa tipkama kojima se kreće igrač potrebno je na Blend Tree dodati skriptu. U toj skripti stvaramo varijablu m_Damping, ona služi da bi se zagladio prijenos iz jedne animacije u drugu prilikom promjene tipki. Zatim dohvaćamo parametre 'Horizontal' i 'Vertical' iz Animatora.
U funkciji OnStateUpdate dohvaćamo animator, njegovo stanje i indeks sloja. U toj funkciji dohvaćamo unos tipki i spajamo u objekt input klase Vector2, te zatim te vrijednosti šaljemo u animator controller (Slika 24).

```
public class TestSMB : StateMachineBehaviour
{
    public float m_Damping = 0.15f;

    private readonly int m_HashHorizontalPara = Animator.StringToHash("Horizontal");
    private readonly int m_HashVerticalPara = Animator.StringToHash("Vertical");

    override public void OnStateMove(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
    {
        float horizontal = Input.GetAxis("Horizontal");
        float vertical = Input.GetAxis("Vertical");

        Vector2 input = new Vector2(horizontal, vertical).normalized;

        animator.SetFloat(m_HashHorizontalPara, input.x, m_Damping, Time.deltaTime);
        animator.SetFloat(m_HashVerticalPara, input.y, m_Damping, Time.deltaTime);
    }
```

Slika 24 Blend Tree skripta

7.4.3. Kretanje igrača

Skriptu za kretanje igrača nazvali smo RigidbodyController [23]. Prvo što trebamo napraviti je stvoriti varijable Rigidbody-a igrača, Vector3 inputs u koji ćemo spremati unose sa tipkovnice, brzinu, udaljenost od tla te varijable za provjeru prizemljenosti našeg igrača. Također treba nam i varijabla za Animator.

U funkciji Start() dohvaćamo igrača i animator (Slika 25).

```
_body = GetComponent<Rigidbody>();
anim = GetComponent<Animator>();
```

Slika 25 Dohvaćanje komponenti

U funkciji Update() prvo provjeravamo da li je igrač prizemljen. Zatim dohvaćamo unose s tipkovnice te ih spremamo u Vector3. Te unose prosljeđujemo u funkciju Move() (Slika 26) kojom povezujemo animator s kretanjem (Slika 27).

```
void Update()
{
    _isGrounded = Physics.CheckSphere(_groundChecker.position, GroundDistance, Ground, QueryTriggerInteraction.Ignore);


    _inputs = Vector3.zero;
    _inputs.x = Input.GetAxis("Horizontal");
    _inputs.z = Input.GetAxis("Vertical");
    Move(_inputs.x, _inputs.z);
```

Slika 26 RigidbodyController Update funkcija

```
private void Move(float x, float z)
{
    anim.SetFloat("Horizontal", x);
    anim.SetFloat("Vertical", z);
}
```

Slika 27 Move funkcija

Samo kretanje igrača se zapravo odvija u funkciji FixedUpdate() koristeći funkciju MovePosition (Slika 28). Razlog iz kojeg smo kretanje unijeli u FixedUpdate() a ne u Update() je taj što je Unity odlučio odvojiti Physic update od klasičnog update-a, pa tako sistem unosa Unity-a nije sikroniziran sa FixedUpdate(). Zbog toga smo prikupili unose u Update() funkciji i onda ih iskoristili u FixedUpdate() funkciji.

```
void FixedUpdate()
{
    _body.MovePosition(_body.position + _inputs * Speed * Time.fixedDeltaTime);
}
```

Slika 28 FixedUpdate funkcija

7.4.4. Podizanje i šut - animacije

Kao što smo već spomenuli ova simulacija će se sastojati i od podizanja lopte i šuta. Tako moramo pokrenuti animaciju tih kretnji putem skripte, što je prilično jednostavno. Sve što moramo napraviti je stvoriti dvije varijable tipa KeyCode, PickUp i FreeThrow, čije ćemo vrijednosti odnosno tipke odrediti naknadno. Za korištenje animacije koristit ćemo se

funkcijom GetKeyDown, što znači da se pritiskom tipke nešto događa, pa tako u našem slučaju pritiskom tipki se pokreću animacije (Slika 29).

```
if (Input.GetKeyDown(PickUp))
{
    anim.SetTrigger("PickUp");
}

if (Input.GetKeyDown(FreeThrow))
{
    anim.SetTrigger("FreeThrow");
}
```

Slika 29 Pokretanje animacija PickUp i FreeThrow

7.4.5. UI

Kada se igrač približi oznaci X želimo da se na ekranu prikaže poruka u kojoj igraču dajemo informaciju sa kojom tipkom da pokupi loptu, a nakon toga i sa kojom tipkom da šutne. Također želimo i u trenutku kada se tipka stisne da ta informacija nestane sa ekrana. Da bi sve to napravili potrebne su nam dvije skripte, jedna koju ćemo nazvati UI i dodat ju na objekt X, i potrebno je izmijeniti postojeću skriptu RigidbodyController.

Na oznaku X smo stavili Box Collider i označili Is Trigger. To znači da u skripti možemo pozvati funkciju OnTriggerEnter koja kada X dođe u dodir s nekim objektom, izvršava neku radnju. Koristimo i funkciju OnTriggerExit koja radi suprotno (Slika 30). U trenu kada igrač uđe u prostor Collider-a, aktiviramo poruku, kada izađe deaktiviramo ju.

```
void OnTriggerEnter(Collider other)
{
    if(other.tag== "Player")
    {
            MessagePanel.SetActive(true);
    }
}

void OnTriggerExit(Collider other)
{
    if (other.tag == "Player")
    {
        MessagePanel.SetActive(false);
    }
}
```

Slika 30 UI skripta

U trenutku kada se tipka stisne želimo da poruka nestane sa ekrana, zbog toga radimo izmjene u skripti RigidbodyController (Slika 31).

```
if (Input.GetKeyDown(PickUp))
{
    anim.SetTrigger("PickUp");
    Ui.MessagePanel.SetActive(false);
}

if (Input.GetKeyDown(FreeThrow))
{
    anim.SetTrigger("FreeThrow");
    Ui.MessagePanel1.SetActive(false);
}
```

Slika 31 Deaktivacija UI panela

Ne želimo da igrač može pokretati animaciju kad god želi, već samo ako je igrač unutar collider-a objekta X, pa za svaki slučaj i u skriptu RigidbodyController dodajemo novu bool varijablu pod nazivom canPickup, koja provjerava ako je igrač u unutar collider-a, te ako je, omogućuje da se stisnu tipke.

Postoje još i male izmijene na zadnjoj skripti pod nazivom Ball_equip, no njezin sadržaj i funkciju ćemo objasniti u sljedećem poglavlju.

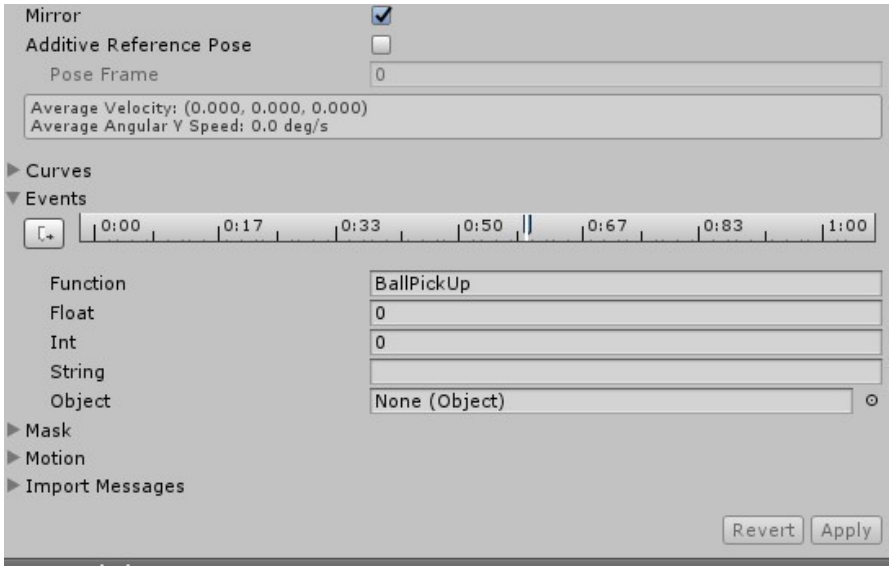
7.5. Interakcija između objekata

Stvorili smo skripte za pokretanje animacija podizanja lopte i šuta, no još uvijek nismo povezali loptu sa svime time. To povezivanje je ujedno bio i najveći izazov ovog rada.

7.5.1. Podizanje lopte

Ono što želimo je da lopta tijekom animacije ostane u rukama igrača u trenutku kada je ruke igrača dotaknu. Spajamo objekt lopte i objekt igrača. Samim time objekt lopte u određenom trenutku postaje dijete ruke igrača.

Kako bi odredili taj trenutak moramo napraviti Event u animaciji (Slika 32). Eventi nam omogućuju da zovemo funkcije u skripti objekta u točno određeno vrijeme animacije. Kod animacije podizanja lopte, odaberemo frame koji nam odgovara te dodamo Event. Naziv funkcije eventa određuje nam naziv funkcije u skripti.



Slika 32 Events

Za gotovo sve što se događa vezano za loptu stvorili smo skriptu Ball_equip. Da bi uopće mogli dohvatiti loptu ona mora biti u blizini igrača, pa tako u funckiji Update() prvo provjeravamo to. Sljedeće stvaramo funkciju BallPickUp() i bool varijablu isHolding (Slika 33). Kada frame animacije dostigne frame na koji je stavljen event, isHolding poprima vrijednost true. Ovom skriptom kontroliramo i prethodno objašnjene UI objekte.

```
//Eventi u animaciji
public void BallPickUp()
{
    isHolding = true;
    Ui.MessagePanel1.SetActive(true);
}
```

Slika 33 BallPickUp funkcija

Nakon toga u Update() funkciji definiramo loptu kao dio ruke ako je lopta blizu i ako isHolding ima vrijednost true i stvaramo funkciju ChangeParent() u kojoj postavljamo da je ball.transform.parent = hand.transform [25].

Lopta ima svoj Rigidbody, collider, masu, poziciju i rotaciju. No, da bi se lopta prirodno kretala trebalo je kontrolirati i bacanje lopte, a u tom trenutku ona se više ne treba ponašati kao dijete igrača. Naime, prilikom podizanja lopte, u trenutku kada igrač podigne ruku i lopta bi se lagano podigla, Rigidbody igrača i Rigidbody lopte se spajaju, ali nakon toga bi lopta ispala iz ruke jer se odbila i oponašala pokrete igrača, što znači da se ponašala kao dijete igrača i kretala skupa s njim.

Da bi podizanje lopte bilo vjerodostojno trebalo je zamrznuti poziciju i rotaciju te isključiti collider lopte (Slika 34).

```
void Update()
{
    float distance = Vector3.Distance(ball.transform.position, hand.transform.position);
    if (distance <= 2f)
    {
        isNear = true;
    }
    else
    {
        isNear = false;

    }

   if (isNear)
    {
        if (isNear && isHolding)
        {
            ChangeParent();
            ball_rb.constraints = RigidbodyConstraints.FreezeAll;
            ball_co.enabled = false;
            ball.transform.localPosition = new Vector3(0, 0, 0);
        }
```

Slika 34 Podizanje lopte

Napravili smo i manje promjene na skripti RigidbodyController gdje smo onemogućili kretanje igrača nakon što podigne loptu i tijekom šuta koristeći RigidbodyConstraints.FreezePosition.

7.5.2. Izbačaj lopte

Za izbačaj lopte također je potrebno stvoriti Event u čijem trenutku će lopta prestati biti dijete ruke. Funkciju eventa nazivamo BallThrow() i dodajemo funkciju BallThrow() u skriptu Ball_equip (Slika 35).

U funckiji BallThrow() vraćamo roditelja lopte na null, sva ograničenja poništavamo i uključujemo collider lopte. Kako smo prije onemogućili daljnje kretanje igrača sada ga moramo ponovo omogućiti nakon što on izbaci loptu iz ruke, i vratiti ograničenja na onakva kakva su bila na početku.

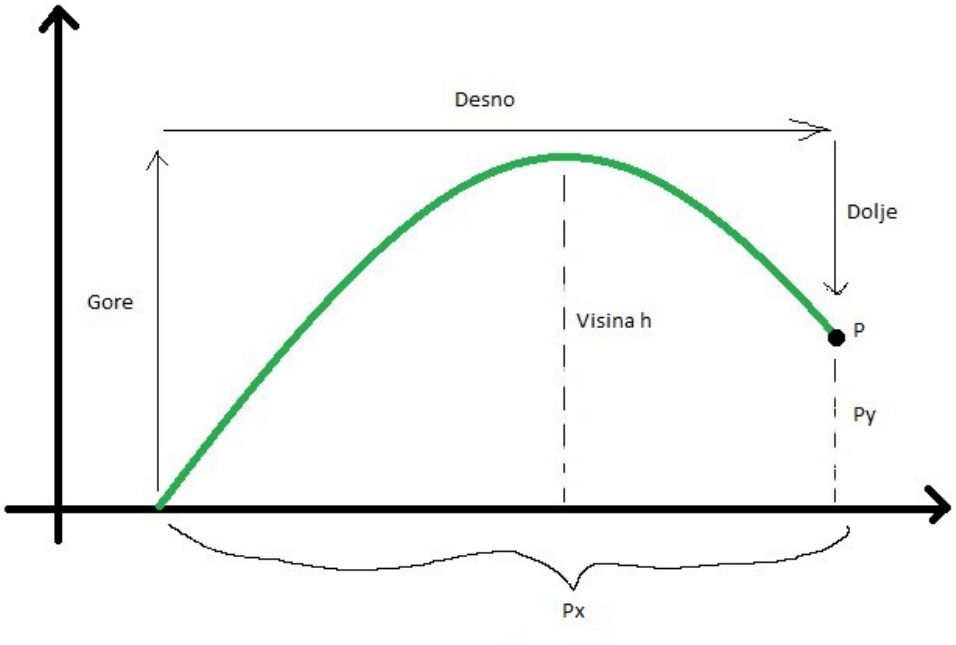
```
public void BallThrow()
{
    isHolding = false;
    ball.transform.parent = null;
    ball_rb.constraints = RigidbodyConstraints.None;
    ball_co.enabled = true;
    body_rb.constraints = RigidbodyConstraints.FreezeRotation;
```

Slika 35 BallThrow funckija

Kada pokrenemo simulaciju lopta nakon Eventa BallThrow pada na pod.

Želimo da u trenutku kada lopta prestane biti dijete ruke ne padne pa pod već odleti u koš. Za to je potrebno uključiti zakone fizike [20,21] koji se odnose na kosi hitac kod kojeg izbačeno tijelo ide po putanji koja ima oblik parabole s tjemenom na vrhu (Slika 36).

Prvo trebamo izračunati kretanje lopte prema gore, lopta se kreće do visine h, ubrzanje je jednako gravitaciji ali u suprotnom smjeru a završna brzina je jednaka nuli. Dok se lopta kreće prema dolje ubrzanje je jednako gravitaciji i početna brzina je 0, dok je kretnja jednaka dužini Py – h. Kretanje u desno izračunavamo uz pomoć horizontalne dužine Px, ubrzanje je jednako 0. Vrijeme kretanja u desno će biti jednako zbroju vremena kretanja prema gore i kretanja prema dole.



Slika 36 Graf parabole

Da bi izračunali kretanje prema gore potrebno je koristiti jednadžbu u kojoj je početna brzina poznata, kao i ubrzanje i put: v^2 je finalno ubrzanje, ubrzanje postaje g, a put postane visina.

$$v^2 = v_0^2 + 2as$$
$$0 = v_{up}^2 + 2gh$$
$$v_{up} = \sqrt{-2gh}$$

Da bi izračunali vrijeme puta prema gore potrebna je sljedeća jednadžba :

$$s = vt - \frac{at^2}{2}$$
$$h = 0 * t_{up} - \frac{g * t_{up}^2}{2}$$
$$t_{up}^2 = -\frac{2h}{g}$$
$$t_{up} = \sqrt{-\frac{2h}{g}}$$

Zatim je potrebno vrijeme kretanja prema dolje. Ovaj put ćemo koristiti drugu jednadžbu pošto znamo vrijednost početnog ubrzanja, a ne znamo finalno ubrzanje.

$$s = v_0 t + \frac{at^2}{2}$$
$$Py - h = 0 * t_{down} + \frac{g * t_{down}^2}{2}$$
$$t_{down}^2 = \frac{2(Py - h)}{g}$$
$$t_{down} = \sqrt{\frac{2(Py - h)}{g}}$$

Sada kada imamo jednadžbe za vrijeme kretanja prema gore i prema dolje možemo izračunati početno ubrzanje za kretanje u desno odnosno horizontalno kretanje. Ponovno ćemo koristiti jednadžbu za put iz prethodnog izračuna i unijeti poznato.

$$Px = v_{right} * (t_{up} + t_{down}) + \frac{0 * (t_{up} + t_{down})^2}{2}$$
$$v_{right} = \frac{Px}{(t_{up} + t_{down})}$$
$$v_{right} = \frac{Px}{\sqrt{-\frac{2h}{g}} + \sqrt{\frac{2(Py - h)}{g}}}$$

Sada kada imamo sve što nam treba unesimo to u novu skriptu ShootBall. Trebamo naravno dohvatiti Rigidbody lopte i Transform poziciju cilja, odnosno koša. Zatim nam treba i maksimalna točka koju će parabola doseći odnosno visina i nju ćemo označiti sa slovom h, te na kraju oznaka za gravitaciju.

Stvaramo strukturu LaunchData kako bi mogli dohvaćati početno ubrzanje i vrijeme potrebno da se stigne do cilja (Slika 37).

```
struct LaunchData
{
    public readonly Vector3 initialVelocity;
    public readonly float timeToTarget;

    public LaunchData(Vector3 initialVelocity, float timeToTarget)
    {
        this.initialVelocity = initialVelocity;
        this.timeToTarget = timeToTarget;
    }
```

Slika 37 LaunchData struktura

Stvaramo metodu tipa LaunchData i nazivamo ju CalculateLaunchData. U njoj smještamo sve jednadžbe potrebne za izračun ubrzanja i vremena koje smo prethodno prikupili (Slika 38).

```
LaunchData CalculateLaunchData()
{
    float displacementY = target.position.y - ball.position.y; //Py
    Vector3 displacementXZ = new Vector3(target.position.x - ball.position.x, 0, target.position.z - ball.position.z); //Px u formuli

    float time = Mathf.Sqrt(-2 * h / gravity) + Mathf.Sqrt(2 * (displacementY - h) / gravity); //vrijeme kretanja u desno

    Vector3 velocityY = Vector3.up * Mathf.Sqrt(-2 * gravity * h);  // ubrzanje prema gore
    Vector3 velocityXZ = displacementXZ / (Mathf.Sqrt(-2 * h / gravity) + Mathf.Sqrt(2 * (displacementY - h) / gravity)); //ubrzanje prema desno

    return new LaunchData(velocityXZ + velocityY * -Mathf.Sign(gravity), time);
}
```

Slika 38 CalculateLaunchData metoda

Stvaramo i metodu Launch() koju ćemo pozivati kada želimo da igrač šutne loptu na koš. U njoj postavljamo vrijednost ubrzanja lopte na vrijednost koju smo izračunali uz pomoć CalculateLaunchData metode. Također želimo da Rigidbody u tom trenutku koristi vrijednost gravitacije naznačenu u skripti (Slika 39).

```
public void Launch()
{
    Physics.gravity = Vector3.up * gravity;
    ball.velocity = CalculateLaunchData().initialVelocity;
}
```

Slika 39 Launch metoda

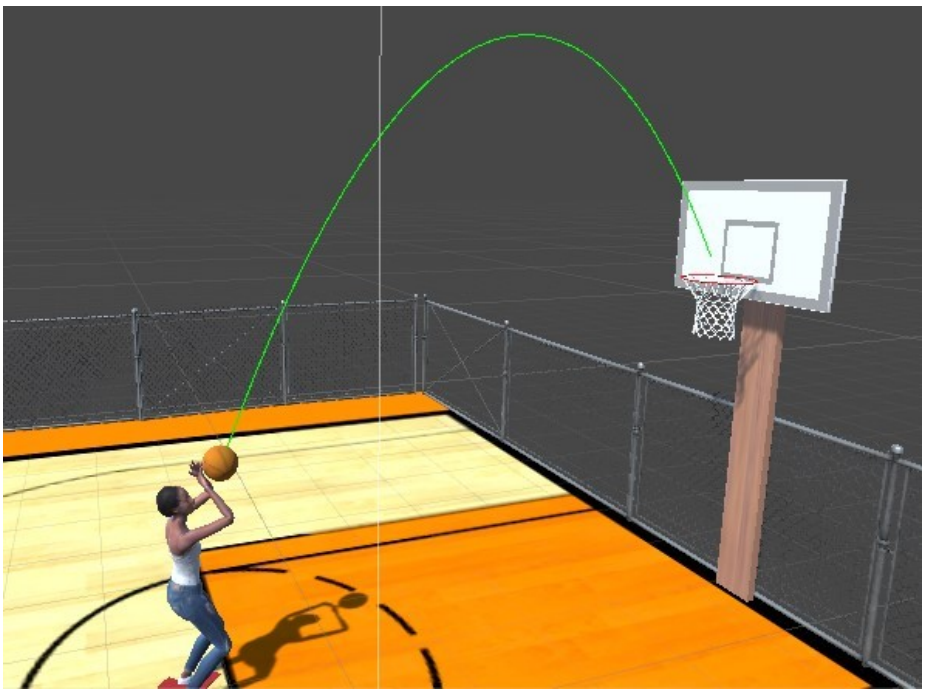
Kako bismo lakše mogli vidjeti i namjestiti putanju lopte onako kako želimo stvorit ćemo liniju u sceni koja ocrtava kretanje lopte (Slika 40 i 41).

```
void DrawPath()
{
    LaunchData launchData = CalculateLaunchData();
    Vector3 previousDrawPoint = ball.position;

    int resolution = 30;
    for (int i = 1; i <= resolution; i++)
    {
        float simulationTime = i / (float)resolution * launchData.timeToTarget;
        Vector3 displacement = launchData.initialVelocity * simulationTime + Vector3.up * gravity * simulationTime * simulationTime / 2f;
        Vector3 drawPoint = ball.position + displacement;
        Debug.DrawLine(previousDrawPoint, drawPoint, Color.green);
        previousDrawPoint = drawPoint;
    }
```

Slika 40 DrawPath metoda

S time je izbačaj lopte u potpunosti gotov.



Slika 41 Parabola u sceni

Ovaj način izračuna parabole omogućava da gdje god se lopta nalazila, tj. s koje god pozicije je šutnuli, ona će uvijek pogoditi svoj cilj.

7.6. Start Meni

To je prvi meni kojeg igrač ugleda kada pokrene igru. Scena početnog meni-a mora ostaviti dobar prvi dojam na igrača i dati mu upute o samoj igrici, načinu korištenja, o postavkama i mogućnostima prilagođavanja igraču.

Napravljen je jednostavan meni s tri funkcije: Start, About i Exit. Prva i zadnja funkcija služe za pokretanje, odnosno zatvaranje igre, dok About daje informacije o igri. Dodali smo i jednostavnu animaciju povećanja slova prilikom prelaska preko gumba kako bi napravili meni malo zanimljivijim (Slika 42).

Slika 42 Povećanje riječi "About" u početnom meniju

Prilikom igranja dodana je mogućnost pauziranja igre i povratka na početni meni. Sve to povezano je skriptom [19].

Simulacija košarkaškog šuta je gotova (Slika 43-47).

Slika 43 Prilaženje lopti

Slika 44 Podizanje lopte

Slika 45 Šutiranje

Slika 46 Let lopte

Slika 47 Postignuti koš

8. Zaključak

Tijekom slušanja predmeta Objektno orijentirano programiranje i rada na projektu izrade edukativne igre jako me zaintrigirao rad u Unity-u te sam odlučila proširiti svoje znanje i za završni rad izraditi simulaciju košarkaškog šuta. Tijekom izrade pokazalo se da je za simulaciju šuta potrebno i znanje iz fizike.

Ono što me najviše oduševilo prilikom izrade simulacije je što se jedna stvar može napraviti na više načina, točnije programer bira da li će se više koristiti skriptama ili Unity komponentama. Unity nudi bogat Asset store u kojem se mogu odabrati likovi i materijali, ima odličan priručnik i tutorijale pomoću kojih se brzo može napredovati i stvoriti ne samo jednostavne projekte i mini igre nego i kompleksnije open world simulacije.

Simulacija košarkaškog šuta je simulacija koja ima puno prostora za nadogradnju, npr. trenutno se simulacija bazira na šutu slobodnog bacanja, no moguće je implementirati i šut s bilo kojeg djela terena.

Nadam se da je ovo tek moj početak bavljenja sa simulacijama i da ću je nastaviti nadograđivati i proširivati svoje znanje.

9. Literatura

[1] https://wibbu.com/importance-narrative-video-games/ Pronađeno 23.6.2019

[2] https://www.rollingstone.com/culture/culture-lists/50-most-iconic-video-game-characters-of-the-21st-century-193057/44-freddy-fazbear-104318/ Pronađeno 23.6.2019

[3] Motion in Games: First International Workshop, MIG 2008, Utrecht, The Netherlands, June 14-17, 2008, Revised Papers. Pronađeno 5.6.2019

[4] https://en.wikipedia.org/wiki/Animation Pronađeno 23.6.2019

[5] https://en.wikipedia.org/wiki/Simulation#Computer_simulation Pronađeno 5.6.2019

[6] Motion Blending, Kristine Slot, February 2007. Pronađeno 23.6.2019

[7] https://en.wikipedia.org/wiki/Motion_capture Pronađeno 23.6.2019

[8] https://www.blender.org/ Pronađeno 23.6.2019

[10] https://en.wikipedia.org/wiki/Unity_(game_engine) Pronađeno 11.6.2019

[11] http://gamedev.machina.hr/unity-3d-game-engine-tvornica-igara-indie-studija/ Pronađeno 11.6.2019

[12] https://assetstore.unity.com/ Pronađeno 23.6.2019.

[13] https://free3d.com/ Pronađeno 23.6.2019

[14] https://www.renderhub.com/ Pronađeno 23.6.2019

[15] https://docs.unity3d.com/Manual/Prefabs.html Pronađeno 14.6.2019

[16]https://docs.unity3d.com/560/Documentation/Manual/Preparingacharacterfromscratch.html Pronađeno 17.6.2019

[17] https://docs.unity3d.com/Manual/AnimationClips.html Pronađeno 18.6.2019

[18] https://docs.unity3d.com/Manual/class-AnimatorController.html Pronađeno 18.6.2019

[19] https://docs.unity3d.com/ScriptReference/SceneManagement.SceneManager.html Pronađeno 18.6.2019

[20] https://hr.wikipedia.org/wiki/Kinematika Pronađeno 21.6.2019.

[21] https://bs.wikipedia.org/wiki/Ubrzanje Pronađeno 21.6.2019.

[22] https://docs.unity3d.com/530/Documentation/Manual/FBXImporter-Model.html Pronađeno 14.6.2019

[23] https://medium.com/ironequal/unity-character-controller-vs-rigidbody-a1e243591483 Pronađeno 7.6.2019

[24] https://docs.unity3d.com/ScriptReference/Material.html Pronađeno 12.6.2019

[25] https://docs.unity3d.com/ScriptReference/Transform.html Pronađeno 12.6.2019

[26] http://www.oshita-lab.org/research/synthesis/index.html Pronađeno 23.6.2019

[27] Motion Synthesis from Annotations, Arikan, A.Forsyth, F. O'Brien, ACM SIGGRAPH 2003 conference proceedings. Pronađeno 23.6.2019

[28] https://www.studica.com/blog/game-design-tutorial-blend-trees-unity Pronađeno 3.7.2019

[29] https://en.wikipedia.org/wiki/Non-player_character Pronađeno 3.7.2019

[30] https://cmonkeys.com/about/ Pronađeno 3.7.2019

10. Popis slika

11. Prilozi:

Uz pismeni dio rada priložen je CD projekta simulacije košarkaškog šuta.