



УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У
НОВОМ САДУ



Страхиња Ераковић

Примена инверзне кинематике у креирању анимације кретања 3D модела

ДИПЛОМСКИ РАД
- Основне академске студије -


Нови Сад, 2023

КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

Редни број, РБР:	
Идентификациони број, ИБР:	
Тип документације, ТД:	Монографска публикација
Тип записа, ТЗ:	Текстуални штампани документ/ЦД
Врста рада, ВР:	Завршни-bachelor рад
Аутор, АУ:	Страхиња Ераковић
Ментор, МН:	Проф. др Драган Иветић
Наслов рада, НР:	Примена инверзне кинематике у креирању анимације кретања 3D модела
Језик публикације, ЈП:	Српски(ћирилица)/Српски (ћирилица)
Језик извода, ЈИ:	Српски/Енглески
Земља публиковања, ЗП:	Србија
Уже географско подручје, УГП:	Војводина
Година, ГО:	2023
Издавач, ИЗ:	Ауторски репринт
Место и адреса, МА:	Факултет Техничких Наука (ФТН), Д. Обрадовића 6, 21000 Нови Сад
Физички опис рада, ФО: (поглавља/страна/ цитата/табела/слика/графика/прилога)	8/34/1/0/18/2/0
Научна област, НО:	Електротехничко и рачунарско инжењерство
Научна дисциплина, НД:	Примењене рачунарске науке и информатика
Предметна одредница/Кључне речи, ПО:	Рачунарска графика, Анимација, Инверзна кинематика, Јунити
УДК	
Чува се, ЧУ:	Библиотека ФТН, Д. Обрадовића 6, 21000 Нови Сад
Важна напомена, ВН:	
Извод, ИЗ:	Овај рад се фокусира на стварање апликације која ће олакшати процес креирања анимације ходања за 3D модел и истовремено демонстрирати важност инверзне кинематике у креирању овакве анимације. Апликација је развијена користећи Јунити погон. Скрипте су написане у програмском језику C#. Као резултат, створена је интерактивна апликација која омогућава корисницима да на једноставан и интерактиван начин креирају анимације ходања за 3D модел.
Датум прихватања теме, ДП:	
Датум одбране, ДО:	
Чланови комисије, КО:	Председник: Др Александар Купусинац, ред. проф
	Члан: Др Дуња Врбашки, доцент
	Члан, ментор: Др Драган Иветић, ред. проф.
	Потпис ментора

KEY WORDS DOCUMENTATION

Accession number, ANO :			
Identification number, INO :			
Document type, DT :	Monographic publication		
Type of record, TR :	Textual material, printed/CD		
Contents code, CC :	Bachelor thesis		
Author, AU :	Strahinja Eraković		
Mentor, MN :	Dragan Ivetić, PhD, full professor		
Title, TI :	Application of Inverse Kinematics in Creating Animation for 3D Models		
Language of text, LT :	Serbian (cyrillic script)/Serbian (latin script)		
Language of abstract, LA :	Serbian/English		
Country of publication, CP :	Serbia		
Locality of publication, LP :	Vojvodina		
Publication year, PY :	2023		
Publisher, PB :	Author reprint		
Publication place, PP :	Faculty of Technical Sciences, D. Obradovića 6, 21000 Novi Sad		
Physical description, PD : (chapters/pages/ref./tables/pictures/graphs/appendixes)	8/34/1/0/18/2/0		
Scientific field, SF :	Electrical and computer engineering		
Scientific discipline, SD :	Applied computer science and informatics		
Subject/Key words, S/KW :	Computer Graphics, Animation, Inverse Kinematics, Unity		
UC			
Holding data, HD :	Library of the Faculty of Technical Sciences, D. Obradovića 6, 21000 Novi Sad		
Note, N :			
Abstract, AB :	<p>This paper focuses on the development of an application designed to simplify the process of creating walking animations for 3D models while simultaneously demonstrating the importance of inverse kinematics in such animations. The application was developed using the Unity engine, and scripts were written in the C# programming language. As a result, an interactive application was created, allowing users to easily and interactively generate walking animations for 3D models.</p>		
Accepted by the Scientific Board on, ASB :			
Defended on, DE :			
Defended Board, DB :	President:	Aleksandar Kupusinac, PhD, full professor	
	Member:	Dunja Vrbaški, PhD, assist. prof.	Menthor's sign
	Member, Mentor:	Dragan Ivetić, PhD, full professor	

	УНИВЕРЗИТЕТ У НОВОМ САДУ • ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА 21000 НОВИ САД, Трг Доситеја Обрадовића 6	Број:
	ЗАДАТАК ЗА ИЗРАДУ ДИПЛОМСКОГ (BACHELOR) РАДА	Датум:

(Податке уноси предметни наставник - ментор)

Врста студија:	<input type="checkbox"/> Основне академске студије
Студијски програм:	РАЧУНАРСТВО И АУТОМАТИКА
Руководилац студијског програма:	Проф. др Милан Рапаић

Студент:	Страхиња Ераковић	Број индекса:	RA 152/2019
Област:	Рачунарска графика		
Ментор:	Проф. др Драган Иветић		

НА ОСНОВУ ПОДНЕТЕ ПРИЈАВЕ, ПРИЛОЖЕНЕ ДОКУМЕНТАЦИЈЕ И ОДРЕДБИ СТАТУТА ФАКУЛТЕТА ИЗДАЈЕ СЕ ЗАДАТАК ЗА ЗАВРШНИ (Bachelor) РАД, СА СЛЕДЕЋИМ ЕЛЕМЕНТИМА:

- проблем – тема рада;
- начин решавања проблема и начин практичне провере резултата рада, ако је таква провера неопходна;
- литература

НАСЛОВ ДИПЛОМСКОГ (BACHELOR) РАДА:

Примена инверзне кинематике у креирању анимације кретања 3D модела

ТЕКСТ ЗАДАТКА:

Проучити алгоритме инверзне кинематике при анимирању модела и на основу стечених сазнања развити Јунити решење за анимирање артикулатног модела дефинисањем терминалних положаја у простору.

Руководилац студијског програма:	Ментор рада:

Примерак за: ☐ - Студента; ☐ - Ментора

Spisak korišćenih skraćenica

3D	Trodimenzijski
IK	Inverse kinematics
FPS	Frames per second

Sadržaj

1. Uvod.....	1
2. 3D Model	2
2.1 Skeletni sistem.....	2
3. Tipovi animacije.....	4
3.1 Keyframe animacija	4
3.2 Proceduralna animacija	4
4. Kinematika	5
4.1 Direktna kinematika	5
4.2 Inverzna kinematika	5
5. Implementacija	7
5.1 Tehnologije	7
5.1.1 Juniti pogon	7
5.1.2 C#.....	7
5.1.3 Animation rigging	8
5.2 Konfiguracije Animation Rigging paketa.....	8
5.3 Kamera.....	10
5.3.1 Kamera u prvom licu.....	10
5.3.2 Zadavanje koordinata krajnjih pozicija stopala	11
5.4 Kretanje noge	11
5.5 Rotacija stopala	15
5.6 Pomeranje kukova	16
5.7 Pomeranje ruku	17
5.8 Pomeranje ramena	19
5.8 Pomeranje torzoa	21
6. Korišćenje aplikacije	24
6.1 Komande i korisnički interfejs	24
6.2 Modovi.....	24
6.3 Primer kreiranja animacije	25
7. Ograničenja i unapređenja	26
7.1 Realističnije kretanje	26
7.2 Izbor modela.....	26
7.3 Izbor drugačije animacije.....	26
7.4 Mogućnost izvoza animacije.....	26
8. Zaključak	27
Literatura	28
Podaci o kandidatu	29

1. Uvod

Animacija je umetnost stvaranja iluzije pokreta putem niza slika. Osnovna ideja animacije je da se statične slike ili objekti redovno menjaju kako bi stvorili dojam kontinuiranog kretanja. Animacija se koristi u različitim medijima, uključujući filmove, televiziju, video igre, reklame, veb stranice i još mnogo toga. Postoje različite tehnike animacije, uključujući:

- Ručno crtana animacija: U ovoj tehnici, svaka slika se ručno crta, obično na papiru, a zatim se sličice brzo prikazuju jedna za drugom kako bi stvorile iluziju pokreta.
- Stop-motion animacija: Ovde se fizički objekti pomeraju u malim koracima i fotografišu, a zatim se fotografije spajaju kako bi stvorile kretanje.
- Računarska animacija: Računarska grafika omogućava animatorima da stvaraju animaciju koristeći računarski softver. To uključuje 2D animaciju i 3D animaciju.

Animacija se koristi za različite svrhe, od zabave i edukacije do komercijalnih i umetničkih izraza. S obzirom na napredak tehnologije, animacija je postala neizmerno važan aspekt zabavne industrije i modernih medija.

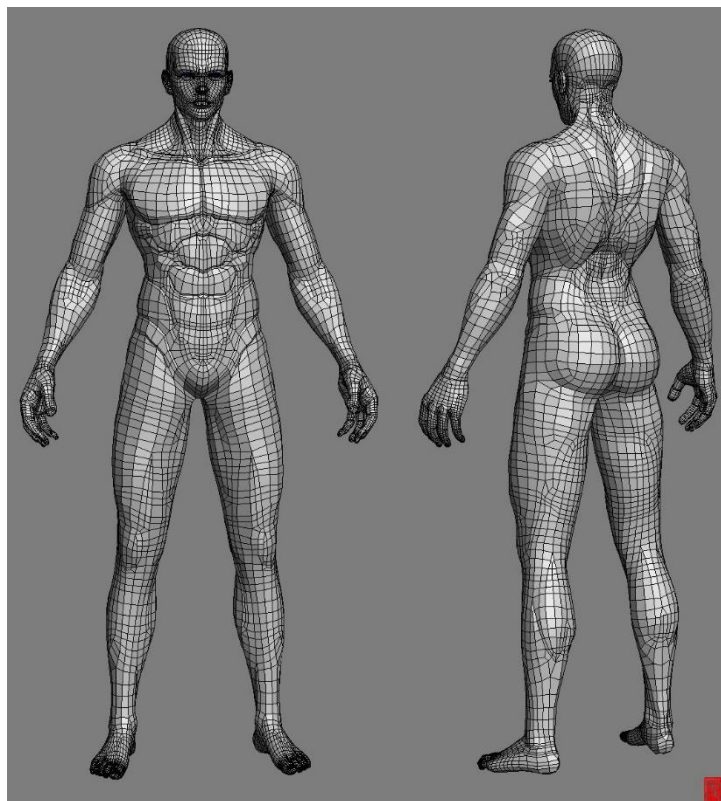
Proces kreiranja animacija predstavlja izuzetno izazovan i zahteva naporan trud. Postoje specifične tehnike koje značajno olakšavaju ovaj proces, posebno kada je reč o animiranju 3D modela. U tu svrhu, softverski alati za animaciju nude napredne mogućnosti koje pomažu animatorima u njihovom kreativnom radu. Među poznatim softverima za animaciju ističu se Blender, Maya, 3ds Max i Cinema4D.

Primer takvog alata je Character Studio u okviru softvera 3ds Max. Ovaj alat omogućava animatorima da definišu samo krajnje položaje stopala 3D modela, nakon čega program samostalno generiše ostatak animacije. Ovakav pristup omogućava ubrzanje procesa animiranja i smanjenje potrebne ručne intervencije.

Cilj je razviti aplikaciju koja će pružiti slično rešenje kao 3ds Max-ov Character Studio, ali će biti integrisana u Juniti pogon. Ovo će omogućiti spajanje sveta video igara i 3D animacija u jedinstvenom okruženju. Aplikacija će biti koncipirana kao mini video igra, u kojoj će korisnik imati aktivnu ulogu u oblikovanju animacije. Korisnički interaktivni pristup omogućiće kreiranje animacija sa većim stepenom kontrole i prilagođavanja, čime će se olakšati proces animacije i omogućiti kreativno izražavanje.

2. 3D Model

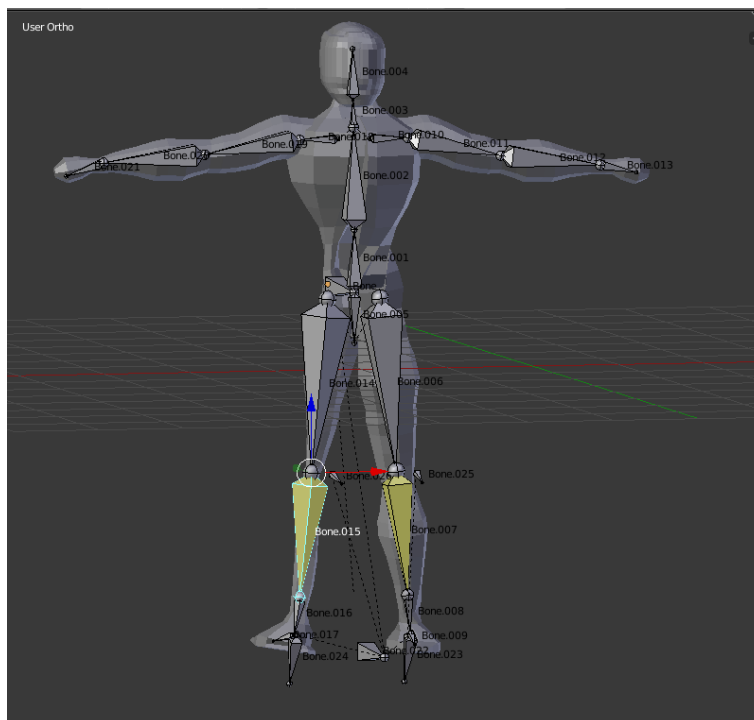
3D model je digitalna reprezentacija trodimenzionalnog objekta ili scene u računarskoj grafici. Ovaj model je sastavljen od različitih komponenti koje zajedno opisuju objekat ili scenu na trodimenzionalan način. Na slici 1 je prikazan model čoveka. Osnovne komponente koje čine 3D model jesu njegova geometrija, tekstura, materijal i skeletni sistem. Geometrijski elementi čine osnovnu strukturu modela. To uključuje tačke (vertices), linije (edges) i površine (faces) koje definišu oblik objekta. Ova geometrija se koristi za reprezentaciju kontura i struktura 3D objekta. Tekstura se primenjuje na površinu modela kako bi se dodale boje i detalji. Ona je dvodimenzionalna slika koja se projektuje na površinu 3D modela kako bi se postigao realističan izgled. Materijala definiše kako svetlost interaguje sa površinom modela. To uključuje svojstva kao što su sjajnost, refleksija, prozirnost i transparentnost, što doprinosi realizmu i estetici objekta. U slučaju animacije, 3D modeli mogu sadržavati skeletni sistem koji omogućava animatorima da postavljaju i kontrolišu pokrete modela. Ovaj sistem se sastoji od kostiju, zglobova i ograničenja kako bi se simuliralo pokretanje objekta.



Slika 1 Model čoveka

2.1 Skeletni sistem

Skeletni sistem ili rig u je složena hijerarhijska struktura kostiju, zglobova i kontrolera koja se koristi za animaciju 3D modela. Ovaj sistem omogućava animatorima da daju život objektima simulirajući prirodne pokrete i deformacije. Na slici 2 prikazan je izgled modela i njegovog skeletnog sistema.



Slika 2 Skeletni sistem modela

Kosti su osnovne komponente skeletnog sistema. Svaka kost predstavlja deo modela koji se može pomerati ili rotirati. Kosti su obično povezane zajedno kako bi stvorile hijerarhijsku strukturu. Na primer, model ruke može imati kosti za nadlakticu, podlakticu i šaku. Zglobovi su tačke na kojima se kosti spajaju i omogućavaju im da se rotiraju oko određene ose. Zglobovi određuju kako se kosti međusobno povezuju i kako će se kretati tokom animacije. Proces kreiranja skeletnog sistema naziva se rigging.

3. Tipovi animacije

3.1 Keyframe animacija

Keyframe animacija je pristup animaciji u računarskoj grafici koji se oslanja na postavljanje ključnih trenutaka ili stanja objekata u animaciji u određenim vremenskim trenucima. Animator postavlja ove ključne trenutke koji definišu početno i krajnje stanje animiranog objekta ili stanja između njih. Ovaj pristup je često korišćen u animaciji likova i objekata u filmovima, video igrima i drugim medijskim sadržajima.

Ključna stvar u kontekstu keyframe animacije je da se naglasak stavlja na unapred pripremljenu animaciju koja se izrađuje i fiksira pre upotrebe. Ova unapred definisana animacija služi kao gotov proizvod i ostaje nepromenjena tokom upotrebe, bez mogućnosti dinamičkih izmena. Na primer, u slučaju animacije hodanja lika u video igri, karakter će uvek izvoditi pokrete na način koji je unapred kreiran i postavljen, nepodložno promenama ili prilagođavanjima tokom igranja.

3.2 Proceduralna animacija

Proceduralna animacija, s druge strane, koristi algoritme i matematičke funkcije kako bi generisala animaciju na osnovu unapred definisanih pravila i parametara, umesto postavljanja ključnih trenutaka ručno. Razlika između ova dva pristupa je u načinu generisanja animacije: keyframe animacija se oslanja na ručno postavljanje trenutka, dok proceduralna animacija koristi matematičke algoritme za generisanje pokreta ili promena.

Centralna karakteristika proceduralne animacije ističe se kroz mogućnost stvaranja animacije u realnom vremenu, što predstavlja značajnu razliku u odnosu na keyframe animaciju. Na primer, u slučaju hodanja karaktera unutar video igre, proceduralna animacija omogućava dinamičko prilagođavanje animacije hoda samom terenu kojim karakter prolazi. Ova adaptacija učiniće da koraci lika budu prirodni i usklađeni sa specifičnostima terena, što doprinosi većem realizmu i interaktivnosti animacije u igri.

4. Kinematika

Kinematika u 3D animaciji je grana koja se bavi proučavanjem i modeliranjem pokreta objekata ili karaktera. Postoje dve osnovne vrste kinematike: direktna kinematika i inverzna kinematika.

4.1 Direktna kinematika

Direktna kinematika ili forward kinematics se koristi za definisanje kretanja objekta ili karaktera od početne tačke do krajnje tačke. Animatori precizno kontrolišu svaki segment pokreta, kao što su rotacije zglobova, kako bi postigli željeni rezultat. Na primer, za animaciju hodanja, animatori će postavljati svaki korak i rotaciju zglobova nogu.

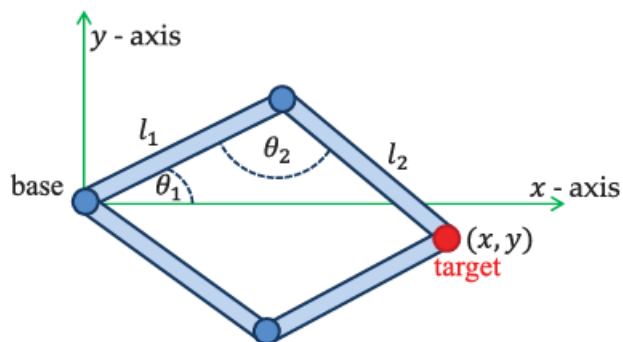
Animacija uz pomoć direktne kinematike zahteva znatno više napora, jer za čak i relativno jednostavan pokret zahteva precizno pomeranje i rotiranje više segmenata skeleta modela. Na primer, da bi se animirao pokret noge, mora se pažljivo pomerati butna kost, lisna kost i rotirati samo stopalo kako bi se postigao željeni rezultat. Ovaj pristup zahteva detaljno ručno upravljanje svakim elementom skeleta, što može biti vremenski i umno zahtevan proces u animaciji.

4.2 Inverzna kinematika

Kod inverzne kinematike ili inverse kinematics se pristup menja. Umesto da se ručno postavlja svaki segment pokreta, ovaj pristup omogućava da se definiše krajnji cilj ili poziciju objekta, a računar automatski izračunava kako se zglobovi i kosti trebaju pomeriti ili rotirati da bi se postigla ta pozicija.

Prilikom animiranja pokreta noge, inverzna kinematika omogućava da se samo definiše konačna pozicija stopala. Algoritam će potom automatizovano izračunati potrebne rotacije butne i lisne kosti kako bi se postigla željena pozicija. Ovaj elegantan pristup omogućava znatno pojednostavljenje procesa animacije, gde računar preuzima složene izračune rotacija i doprinosi većoj efikasnosti u postizanju željenih pokreta.

Inverzna kinematika se suštinski zasniva na matematici, preciznije rečeno, trigonometrijskom proračunu, koji se koristi za određivanje položaja i rotacije kostiju unutar kinematičkog lanca na osnovu unapred definisane ciljane tačke. Ovaj proces može predstavljati izazovan problem koji zahteva pažljivo rešavanje. Radi ilustracije, može se razmotriti jednostavan primer. Na slici 3 prikazane su dve kosti čija su dužine označena l_1 i l_2 , zajedno sa koordinatama ciljane pozicije. Ovaj koncept se može zamisliti kao model ruke, gde l_1 označava dužinu nadlaktice, a l_2 dužinu podlaktice. Tačka označena crvenom bojom sa koordinatama x i y predstavlja željenu poziciju šake. Ključno je izračunati uglove θ_1 i θ_2 , koji precizno određuju pod kojim uglom treba rotirati nadlakticu i podlakticu respektivno kako bi se postigla željena pozicija.



Slika 3 Jednostavan primer inverzne kinematike

Ova dva ugla se mogu izračunati primenom sledećih trigonometrijskih formula [1].

$$\theta_1 = \cos^{-1} \left(\frac{l_1^2 + x^2 + y^2 - l_2^2}{2l_1\sqrt{x^2 + y^2}} \right)$$

$$\theta_2 = \cos^{-1} \left(\frac{l_1^2 + l_2^2 - (x^2 + y^2)}{2l_1l_2} \right)$$

Naravno, kako se kinematički lanac produžava ili kada se cela analiza prenese iz dvodimenzionalnog u trodimenzionalni prostor, jednačbe za izračunavanje ovih uglova postaju znatno složenije. Inverzna kinematika, osim u animaciji, takođe pronalazi primenu u robotici, gde se koristi kako bi se precizno izračunalo koliko je potrebno rotirati robotsku ruku kako bi se postigla specifična pozicija. Na slici 4 je prikazana robotska ruka. Za postizanje željenog položaja hvataljke koriste se jednačine inverzne kinematike. Ovaj pristup omogućava izračunavanje potrebne rotacije motora koji pokreću ruku kako bi se postigao ciljani položaj. Inverzna kinematika je ključna za precizno upravljanje robotima čija struktura nalikuje skeletu.



Slika 4 Robotska ruka

5. Implementacija

Za uspešnu realizaciju ovog projekta neophodno je odabrati adekvatan softverski alat. U nastavku biće detaljno predstavljeni softverski alati i dodaci koji su odabrani i primenjeni kako bi se efikasno rešio postavljeni problem. Takođe će biti pruženo detaljno objašnjenje konkretnog procesa implementacije unutar izabranog softverskog alata.

5.1 Tehnologije

Kao što je pomenuto efikasno rešenje svakog izazova zahteva neophodan odabir pravilnog alata. U skladu s tim, za realizaciju rešenja ovog problema pažljivo su selektovani i primenjeni sledeći softverski alati.

5.1.1 Juniti pogon

Juniti pogon ili Unity game engine je izuzetno svestrana razvojna platforma koja omogućava kreiranje video igara i interaktivnih aplikacija za različite platforme. Ovo uključuje računare, mobilne uređaje, konzole, web i druge uređaje. Zahvaljujući ovoj univerzalnosti, Juniti je postao prvi izbor za mnoge razvojne timove širom sveta.

Jedna od ključnih prednosti Juniti platforme su njeni moćni vizuelni alati. Oni omogućavaju kreiranje, uređivanje i animaciju objekata, scena i elemenata igre. Ovo čini razvoj igara i aplikacija pristupačnim i intuitivnim, čak i za one bez dubokog programerskog iskustva.

Što se programiranja tiče, Juniti podržava C# kao glavni programski jezik za razvoj. To omogućava developerima da pišu skripte i upravljaju igrom uz pomoć ovog popularnog jezika.

Nadalje, Unity Asset Store predstavlja dragocen izvor dodataka. On omogućava programerima da pronađu i koriste resurse, modele, skripte i dodatke koji su drugi developeri kreirali. Ovi dodaci znatno olakšavaju razvoj igara i aplikacija, štedeći vreme i resurse.

Sve ove karakteristike čine Juniti izvanrednim alatom za razvoj video igara i aplikacija, privlačeći razvojne timove i pojedince širom sveta.

Za implementaciju rešenja Juniti je pažljivo izabran iz niza razloga, pri čemu ključni faktor predstavlja njegova sposobnost manipulacije 3D modelima. Dodatno, Unity Package Manager nam pruža širok spektar dodataka koji značajno olakšavaju proces animacije modela. Važno je naglasiti da se velika prednost ovog alata ogleda u njegovoj jednostavnosti upotrebe. Takođe, Juniti se ističe izvanrednom dokumentacijom, čime omogućava korisnicima da se brzo i efikasno upuste u proces razvoja. Osim toga, obilje dostupnih materijala i resursa na internetu predstavlja dragocen izvor podrške i znanja koji znatno doprinose uspešnoj realizaciji rešenja.

5.1.2 C#

C# je izuzetno moćan programski jezik koji se često koristi za razvoj različitih vrsta aplikacija, uključujući desktop aplikacije, web aplikacije i video igre. On je deo Microsoft-ove .NET platforme i nudi mnoge prednosti. Jedna od ključnih karakteristika C# je njegova objektno orijentisana priroda, što omogućava programerima da organizuju svoj kod u logičke celine, olakšavajući održavanje i proširivanje aplikacija.

C# takođe pruža moćan sistem za upravljanje memorijom, što znači da programeri ne moraju ručno upravljati memorijom kao u nekim drugim jezicima, poput C++. Ovo smanjuje rizik od curenja memorije i olakšava razvoj stabilnih aplikacija. Osim toga, C# je dobro podržan od strane različitih integrisanih razvojnih okvira (IDE), kao što su Visual Studio i Visual Studio Code, što čini proces programiranja u C# još produktivnijim.

Juniti pogon koristi programski jezik C# za izradu skripti [2], koje se primenjuju na različite objekte unutar scene kako bi se definisalo njihovo ponašanje. Juniti proširuje C# pomoću svojih posebnih biblioteka i komponenti. Ove biblioteke su dizajnirane tako da olakšaju integraciju jezika i samog Junitija.

S obzirom na sve ove prednosti, C# ostaje popularan izbor za mnoge razvojne projekte i pruža programerima snažan alat za kreiranje raznovrsnih i funkcionalnih aplikacija.

5.1.3 Animation rigging

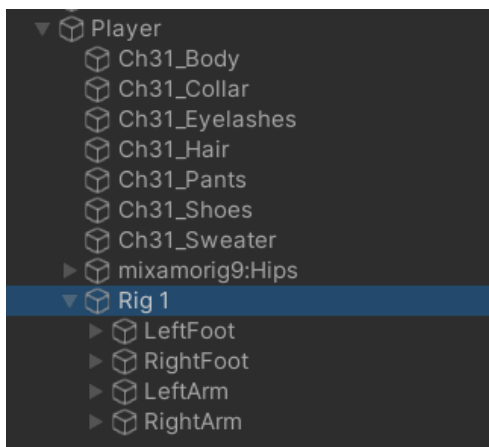
Animation Rigging [3] je paket dostupan putem Unity Package Manager-a, koji pruža efikasno rešenje za animaciju 3D modela putem direktnog, a posebno inverznog kinematičkog pristupa. Ovaj paket preuzima na sebe sve složene matematičke račune potrebne za animaciju modela putem inverzne kinematike, oslobađajući korisnike od ovih tehničkih detalja.

Jedna od značajnih prednosti ovog paketa je njegova jednostavnost upotrebe, što omogućava korisnicima da brzo i lako prilagode svoje 3D modele za animaciju putem inverzne kinematike. Za transformaciju bilo kog modela sa skeletnim sistemom u model koji je podložan manipulaciji putem inverzne kinematike, potrebno je svega nekoliko minuta.

Razlog zbog kojeg se opredeljuje za ovaj paket leži u njegovoj sposobnosti da omogući animaciju ekstremiteta, konkretno ruku i nogu, putem inverzne kinematike na veoma jednostan način.

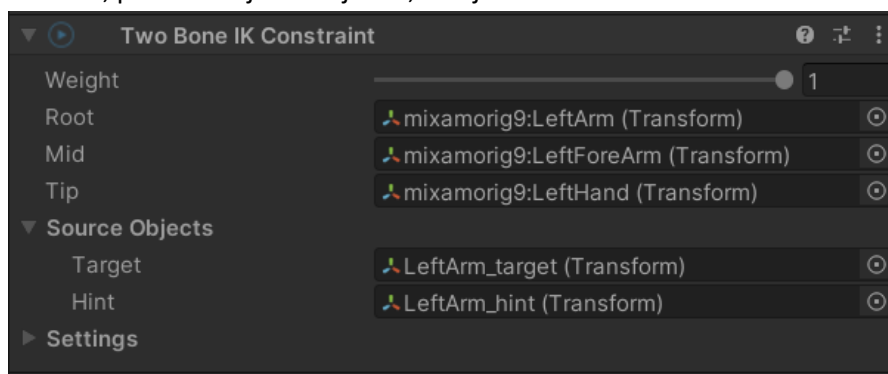
5.2 Konfiguracije Animation Rigging paketa

Proces konfiguracije Animation Rigging paketa je stvar koja je prvi urađen. On se može izvesti u nekoliko jednostavnih koraka. Prvo, potrebno je dodati Rig komponentu na ciljni objekat, što je prikazano na slici 5. Ova komponenta obuhvata sve kinematičke lance modela koje želimo da koristimo, u ovom slučaju, lance za levu i desnu nogu, kao i lance za levu i desnu ruku.



Slika 5 Rig komponenta na objektu

Nakon toga, neophodno je kreirati komponentu koja predstavlja kinematički lanac i odabrati njegov tip, kao što je prikazano na slici 6. Svi kinematički lanci na modelu su tipa Two Bone IK, što znači da se sastoje od dve kosti za koje je potrebno izračunati poziciju, u ovom slučaju to su nadlaktica i podlaktica. Dalje, treba precizirati koje kosti iz skeleta modela čine lanac i definisati njihov redosled, na primer, nadlaktica se smatra bazom lanca, podlaktica je srednji deo, dok je kost šake vrh lanca.

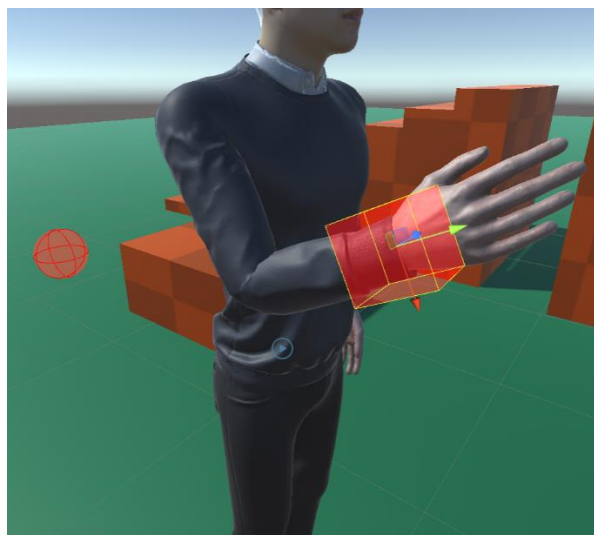


Slika 6 Kreiranje kinematičkog lanca

Nakon uspešno izvedenih ovih koraka, paket će generisati potrebne komponente. Na slici 7 je prikazano šta je paket izgenerisao, uključujući dva nova objekta. Metu (Target), označenu crvenom kockom, koja definiše krajnju poziciju poslednje kosti u lancu, na primer željena pozicija šake, i nagoveštaj (Hint), označenu crvenom sferom, koja služi kao nagoveštava kako treba orijentisati lakat. Pomeranjem crvene kocke, šaka prati njen položaj, dok se ostatak kostiju u lancu automatski pravilno pozicionira. Pomeranje se može videti na slici 8. Šaka je pomerena u odnosu na sliku 7.



Slika 7 Izgenerisana meta i nagoveštaj



Slika 8 Pomeranjem šake

5.3 Kamera

Kamera je implementirana iz perspektive prvog lica i simbolizuje virtuelnog posmatrača koji preuzima ulogu zadavanja komandi modelu. U nastavku će biti detaljno obrađeno kretanje kamere iz prvog lica, kao i procedura zadavanja komandi koje definišu kretanje modela.

5.3.1 Kamera u prvom licu

Na listingu je priložen kod koji omogućava rotaciju kamere u skladu sa promenama u poziciji miša. Preciznije, rotacija kamere se odvija horizontalno ili vertikalno, u zavisnosti od smera pomaka miša duž x ili y ose.

```
void Update()
{
    float mouseX = Input.GetAxisRaw("Mouse X") * Time.deltaTime * sensX;
    float mouseY = Input.GetAxisRaw("Mouse Y") * Time.deltaTime * sensY;

    xRotation -= mouseY;
    yRotation += mouseX;
    xRotation = Mathf.Clamp(xRotation, -90f, 90f);

    transform.rotation = Quaternion.Euler(xRotation, yRotation, 0);
    playerBody.rotation = Quaternion.Euler(0, yRotation, 0);
}
```

Listing 1 Rotacija kamere

Kretanje kamere napred-nazad to jest levo-desno realizovano je putem analize korisničkih komandi, tačnije, detektovanjem da li je korisnik aktivirao komandu za kretanje napred, nazad, levo ili desno. Na osnovu ovih detekcija generiše se vektor čiji smer odgovara smeru u kojem kamera treba da se pomeri. Smer kreiranog vektora zavisi od smera trenutnog pogleda kamere i komande korisnika. Na listingu 2 prikazana je implementacija.


```

void Update()
{
    float moveX = Input.GetAxis("Horizontal");
    float moveZ = Input.GetAxis("Vertical");

    Vector3 move = transform.right * moveX + transform.forward * moveZ;
    move.y = 0f;

    characterController.Move(move * Time.deltaTime * moveSpeed);

    bool spacePressed = Input.GetKey(KeyCode.Space);
    bool shiftPressed = Input.GetKey(KeyCode.LeftShift);

    if(spacePressed)
    {
        characterController.Move(Vector3.up * Time.deltaTime * flySpeed);
    }
    else if(shiftPressed)
    {
        characterController.Move(-Vector3.up * Time.deltaTime * flySpeed);
    }
}

```

Listing 2 Kretanje kamere

5.3.2 Zadavanje koordinata krajnjih pozicija stopala

Cilj je da kreirati niz koordinata koje će reprezentovati uzastopne krajnje pozicije stopala modela. Ovaj proces se postiže putem izvođenja sledećih koraka. Prvo, iz kamere se emituje nevidljivi zrak koji je sposoban da detektuje trenutak kada se sudari sa nekom površinom. Ovaj nevidljivi snop takođe obezbeđuje podatke o koordinatama tačke kontakta sa površinom. Te koordinate se ubacuju u niz željenih pozicija stopala. Važno je istaći da ovaj zrak takođe pruža informacije o normali na površine sa kojom se sudario, što će biti korisno u narednim kasnije. Na listingu 3 prikazan je deo koda koji prikazuje način korišćenja zraka u Junitiju tj. Raycast-a.

```

void Update()
{
    Ray ray = new Ray(transform.position, transform.forward);

    if(Physics.Raycast(ray, out hitInfo, 30f, mask))
    {
        if(mode == Mode.StepGiver)
        {
            if(Input.GetKeyDown(KeyCode.E))
            {
                Vector3 stepGoal = hitInfo.point;
                stepGoal += hitInfo.normal * 0.1154f;

                //Dodaj koordiante u niz
                goalsArray.AddLast(stepGoal);
                footAngleArray.AddLast(hitInfo.normal);
            }
        }
    }
}

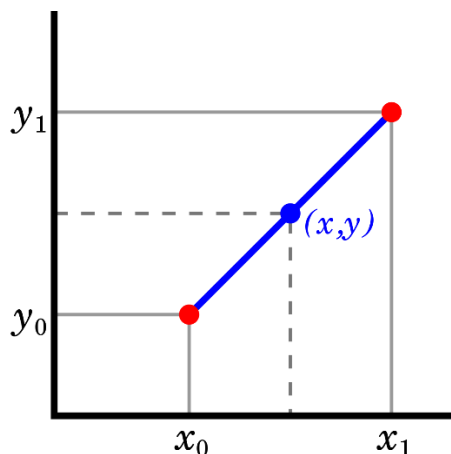
```

Listing 3 Emitovanje zraka

5.4 Kretanje noge

Sada kada smo uspostavili početnu i krajnju poziciju stopala, neophodno je da se stopalo pomakne na odgovarajući način između ove dve tačke. Jedno od prikladnih rešenja koje se može primeniti u ovom kontekstu jeste upotreba linearne interpolacije.

Linearna interpolacija je matematički postupak koji se koristi za procenu vrednosti između dve poznate tačke na osnovu linearnog povećanja ili smanjenja između tih tačaka. Ova tehnika se često koristi u različitim oblastima kao način za aproksimaciju vrednosti između poznatih tačaka ili za pronalaženje vrednosti izvan opsega tačaka. Na primer, ako postoje dve poznate tačke (x_1, y_1) i (x_2, y_2) , u ovim primeru koordinate stopala, linearna interpolacija omogućava procenu vrednost y za neku vrednost x između x_1 i x_2 . Osnovna ideja linearnog interpoliranja je da se pretpostavi da između ove dve tačke postoji prava linija, a zatim se koristi jednačina prave ($y = k \cdot x + n$) da se izračuna vrednost y za datu vrednost x . Na slici 9 je prikazan grafik ovog postupka.

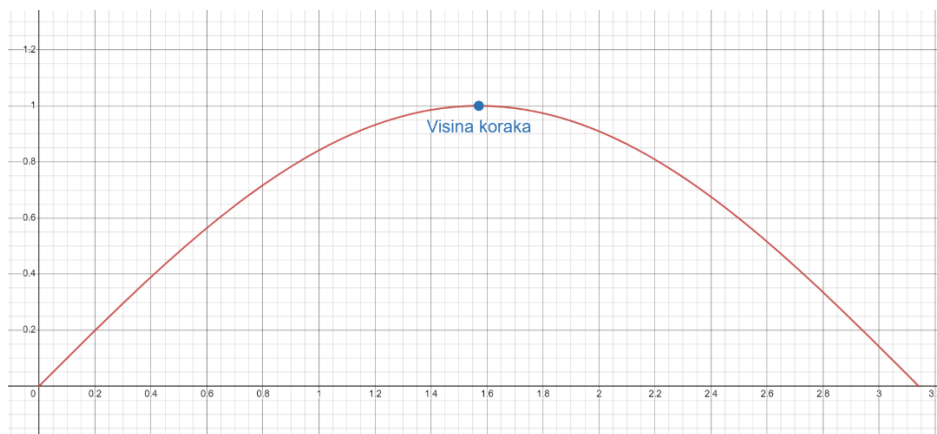


Slika 9 Linearna interpolacija

Još jedna istaknuta prednost ovog pristupa je prisutnost ugrađene funkcionalnosti za linearnu interpolaciju unutar klase Vector3 u okviru Junitija. Metoda Lerp izvršava linearnu interpolaciju između dve tačke. Njeni parametri obuhvataju dve tačke, između kojih se želi izvršiti interpolacija, kao i vrednost koja se kreće u opsegu od 0 do 1. Ova vrednost precizira koliko je potrebno da se pomakne od početne tačke ka ciljnoj tački. Na primer, ako se prosledi vrednost 0.5, metoda vraća tačku koja se nalazi na sredini između dve zadate tačke.

Linearna interpolacija, iako korisna, nosi sa sobom određenu manu koja može uticati na percepciju kretanja nogu. U slučaju primene samo linearne interpolacije, rezultat će biti linearno pomeranje stopala, što neće stvoriti dojam kao da model podiže nogu i pravi korak. Ovaj nedostatak može se prevazići putem modifikovanja y koordinate stopala tokom kretanja po pravi izgenerisane Lerp metodom. Ovaj pristup će stvoriti iluziju da se stopalo podiže i kreće u luku.

Da bi se postigao ovaj cilj, neophodno je odabrati funkciju koja ima putanju luka. Prvi pokušaj u rešavanju ovog problema uključuje upotrebu sinusne funkcije konkretno u domenu od 0 do π . Na slici 10 prikazana je sinusna funkcija. Iako se ovo čini kao prihvatljiv izbor, postoje određeni problemi. Ovo rešenje uzima u obzir samo visinu koraka kao parametar, što je optimalno kada model hoda po ravnom terenu. Međutim kada se model suoči s hodaњem po stepenicama ili sličnim neravnim terenima, ovaj pristup ne omogućava adekvatno podizanje stopala.



Slika 10 Sinusna funkcija [4]

Kako bi se ovaj problem rešio, neophodno je razmotriti još dva dodatna parametra, visinu stopala pre koraka i visinu stopala u krajnjoj poziciji. Za ovu svrhu, biramo kvadratni polinom. Potrebno je pronaći kvadratnu krivu koja će se poklapati s ovim tri parametra. Na slici 11 prikazan je izgled ove funkcije prilagođen navedenim parametrima. Razlog zašto nismo koristili sinusnu funkciju i prilagođavali je parametrima je znatno veća kompleksnost tog postupka, dok je prilagođavanje kvadratnog polinoma ovim parametrima mnogo jednostavnije i efikasnije.



Slika 10 Kvadratni polinom prilagođen parametrima [4]

Kao što je već objašnjeno, metoda Lerp prima parametar koji varira u opsegu od 0 do 1. Ova metoda generiše pravu koju možemo posmatrati kao duž ograničenu tačkama $(0, y_1)$ i $(1, y_2)$, pri čemu y_1 predstavlja visinu stopala u početnoj poziciji, a y_2 visinu stopala u krajnjoj poziciji. Uvođenjem dodatne treće tačke $(0.5, y_3)$, gde y_3 predstavlja visinu koraka, dobijamo tri tačke koje definišu parametre spomenute ranije. Kroz ove tri tačke se provlači kvadratna kriva. Ovaj postupak je detaljno prikazan na listingu 4. Na Slici 12 možemo videti poređenje između duži dobijene primenom Lerp metode i dobijene krive.

```
private Vector3 FitCurve(Vector3 startPosition, Vector3 goalPosition)
{
    float y1 = startPosition.y;
    float y2 = goalPosition.y;
    float y3 = calculatedStepHeight

    float c = y1;
    float a = -4 * y3 + 2 * y2 + 2 * c;
    float b = y2 - c - a;

    return new Vector3(a, b, c);
}
```

Listing 4 Računanje kvadratne jednačine na osnovu parametara

Nakon dobijanja formule za kvadratnu jednačinu može se primeniti Lerp metoda kako bi se stopalo pomerilo na željeni način. Ovaj postupak je prikazan na listingu 5.

```
private void DoLeftSideLerp()
{
    //Lerp stopala
    Vector3 currentFootPos = Vector3.Lerp(leftFootStartPosition, leftFootGoal, lerp);
    currentFootPos.y = Quadratic(coef, lerp);
    leftFoot.position = currentFootPos;
}
```

Listing 5 Primena Lerp metode za pomeranje stopala

Metoda Quadratic je jednostavna i samo računa y za prosleđene koeficijente i željeno x po sledećoj formuli.

$$y = ax^2 + bx + c$$

Na listingu 6 je prikazan kod ove metode.

```
private float Quadratic(Vector3 coef, float X)
{
    return (coef.x * X * X) + (coef.y * X) + coef.z;
}
```

Listing 6 Metoda Quadratic

Metoda za izračunavanje visine koraka predstavlja jednostavan proces. Ona uzima u obzir trenutnu visinu stopala i visinu stopala u krajnjoj poziciji, te od ova dva parametara bira veći. Na ovu vrednost se dodaje konstanta kako bi se dobila željena visina koraka. Kod ove metode prikazan je na listingu 7.

```
private float CalculateStepHeight(Vector3 goalPosition, Vector3 currentPosition)
{
    return Mathf.Max(goalPosition.y, currentPosition.y) + stepHeight;
}
```

Listing 7 Računanje visine koraka

Treći parametar Lerp metode, u ovom slučaju promenljiva pod nazivom lerp, se određuje putem sledećeg postupka. Počinje sa vrednošću 0 i svakim frejmom se inkrementuje za unapred definisanu malu vrednost. Kada dostigne vrednost 1, ponovno se resetuje na početnu vrednost 0. Ovaj proces je prikazan na Listingu 8.

```
lerp += stepSpeed * Time.deltaTime;
```

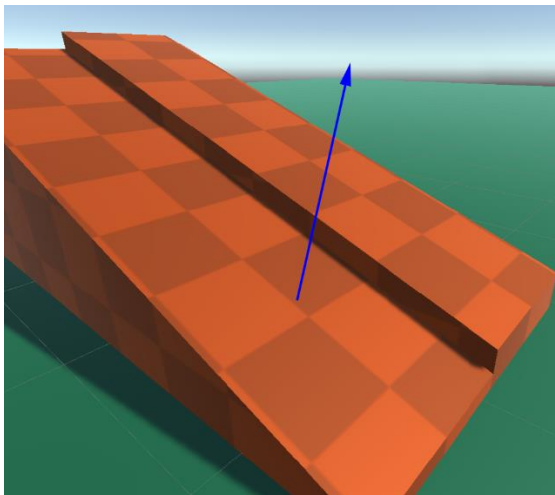
Listing 8 Promena promenljive lerp

Razlog množenja sa Time.deltaTime je da bi ova promena bila nezavisna od broja frejmova u sekundi tj. FPS-a.

5.5 Rotacija stopala

Potrebno je stopalo rotirati tako da naleže na podlogu. Da bi se postigla odgovarajuća orijentacija stopala u skladu sa površinom prepreke, potrebno je rotirati stopalo kako bi se uskladilo sa nagibom te površine. Kada model korača po ravnoj površini, stopalo ostaje u svojoj početnoj orijentaciji, dok će se u slučaju zakoračenja na nagibu od, na primer 30 stepeni, stopalo rotirati za isti taj ugao.

Ključni korak u rešavanju ovog problema je korišćenje normale površine prepreke. Na slici 12 prikazana je vizualizacija normale nagiba. Normala je vektor koji izlazi iz površine prepreke pod pravim uglom u odnosu na ravan te površine. Ovaj vektor daje informaciju pod kolikim uglom je površina prepreke.



Slika 12 Normala površine nagiba

Kao što je već napomenuto, emitovanje zraka iz pravca kamere pruža nam korisnu informaciju o normali površine s kojom se zrak sudara. S obzirom na to da je vektor normale dobro poznat, proces njegove primene u izračunavanju potrebne rotacije stopala je veoma jednostavan jer u okviru Junitija već postoji ugrađena metoda za ovu svrhu.

Da bismo ostvarili ovu rotaciju, koristimo pomenutu metodu, gde je potrebno proslediti dva parametra, početni vektor, koji u ovom slučaju predstavlja y-osu, i krajnji vektor, koji predstavlja normalu površine prepreke. Nakon primene metode njena povratna vrednost se množi sa trenutnom rotacijom stopala kako bi se dobila željena krajnja rotacija stopala. Upotreba ove metode je prikazana na listingu 9.

```
private void CalculateFootRotationLeft()
{
    //Racunanje goal rotacije za stopalo
    leftFootGoalRotation = Quaternion.FromToRotation(
        Vector3.up,
        laserPointerScript.footAngleArray.First.Value) * leftFootStartRot;

    laserPointerScript.footAngleArray.RemoveFirst();
}
```

Listing 9 Računanje krajne rotacije stopala

Preostaje još postepeno izvršiti rotaciju stopala u svakom frejmu, slično kao što je slučaj sa pomeranjem noge. Za ovu svrhu koristimo sferičnu linearnu interpolaciju, koja je posebno prilagođena za rad s rotacijama. Ova metoda je slična standardnoj linearnoj interpolaciji, ali se primenjuje kada je potrebno manipulirati rotacijama umesto pozicijama.

U okviru Juniti biblioteke, postoji ugrađena metoda Slerp (spherical linear interpolation) u okviru klase Quaternion, koja takođe zahteva tri parametra, baš kao i metoda Lerp. Ovi parametri uključuju početnu rotaciju stopala, krajnju rotaciju i promenljivu lerp koja varira u opsegu od 0 do 1. Na Listingu 10 možemo videti kako je ova metoda primenjena u projektu.

```
private void DoLeftSideLerp()
{
    //...

    //Slerp stopala
    leftFoot.rotation =
        Quaternion.Slerp(leftFootStartRotLerp, leftFootGoalRotation, lerp);

    //...
}
```

Listing 10 Primena Slerp metode za rotiranje stopala

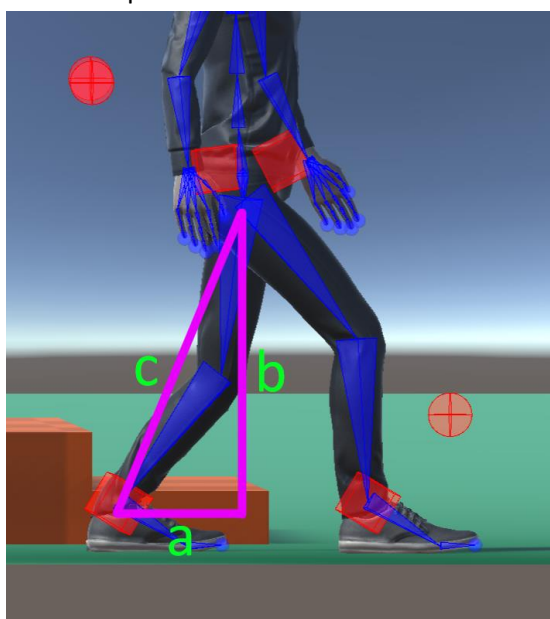
5.6 Pomeranje kukova

Pod pomeranjem kukova podrazumevamo promenu pozicije kosti kukova modela. Budući da su kukovi prva kost u hijerarhiji, njihovim pomeranjem utičemo i na poziciju svih ostalih kostiju. Ovim se postiže pomak celog modela prema napredu ili unazadu. Prilikom pomeranja kukova, bitno je uskladiti ih sa kretanjem nogu. Na primer, ako model korača unapred, potrebno je pomeriti kukove unapred, a ako se model penje uz stepenice potrebno je pomeriti kukove ka gore.

Jednostavan način za rešavanje ovog problema podrazumeva uočavanje trougla koji formiraju noge i stranica ograničena stopalima. Ovaj trougao se može podeliti na pola i ovim dobijamo novi pravougli trougao. Ovaj trougao je prikazan na slici 13. Dužina hipotenuze ovog trougla predstavlja dužinu noge, jedna kateta je polovina razdaljine između stopala, a druga kateta je udaljenost kukova od poda. Nepoznata kateta predstavlja udaljenost kukova od poda. Primenom Pitagorine teoreme, može se izračunati visinu kukova u krajnjem položaju, budući da su dužina hipotenuze i druge katete poznate. Pitagorina teorema glasi:

$$a^2 + b^2 = c^2$$

Gde su a i b dužine kateta, a c dužina hipotenuze.



Slika 13 Uočeni pravougli trougao

Potrebno je takođe izračunati x i z koordinate pozicije kukova. To se postiže primenom Lerp metode, kojoj se kao parametri prosleđuju koordinate stopala i vrednost 0.5. Ovim dobijamo tačku koja se nalazi između stopala. Na Listingu 11 možete videti kod koji izračunava poziciju kukova u krajnjem položaju.

```
private Vector3 CalculateHipsPosition(Vector3 leftFootPosition, Vector3
rightFootPosition)
{
    float feetY = 0f;
    if(leftFootPosition.y < rightFootPosition.y)
    {
        rightFootPosition.y = leftFootPosition.y;
        feetY = leftFootPosition.y;
    } else
    {
        leftFootPosition.y = rightFootPosition.y;
        feetY = rightFootPosition.y;
    }

    float feetDist = Vector3.Distance(rightFootPosition, leftFootPosition)/2;
    float hipHeight = legLength * legLength - feetDist * feetDist;

    Vector3 middlePoint = Vector3.Lerp(leftFootPosition, rightFootPosition, 0.5f);

    return new Vector3(
        middlePoint.x,
        hipHeight - modellegHeight + (feetY - 0.1154f),
        middlePoint.z);
}
```

Listing 11 Računanje krajnje pozicije kukova

Kako bismo postigli željeno pomeranje kosti kukova od početne do izračunate krajnje pozicije, primenjujemo linearnu interpolaciju. Ovaj proces je detaljno prikazan na listingu 12, gde se primenjuje Lerp metoda kako bi se postiglo glatko i precizno pomeranje kosti kukova od početne do ciljane pozicije.

```
private void DoLeftSideLerp()
{
    //...

    //Hips lerp
    hips.position = Vector3.Lerp(hipsStartPosition, calculatedHipsPos, lerp);

    //...
}
```

Listing 12 Primena Lerp metoda za pomeranje kosti kukova

Važno je napomenuti da ovaj metod za izračunavanje visine kukova ima jedan nedostatak. ne uzima u obzir savijenost noge, već pretpostavlja konstantnu razdaljinu od kuka do stopala. Iako ovaj nedostatak ne stvara značajan problem u konkretnoj implementaciji, važno je imati na umu da postoji, kako bismo bili svesni njegovih ograničenja.

5.7 Pomeranje ruku

Kada se analizira ljudsko kretanje, preciznije hodanje, uočavamo da se ruke pomeraju na određen način. Ovaj prirodan hod podrazumeva da se ruke pomeraju u skladu sa suprotnom nogom prilikom koraka. Na primer, ako osoba zakorači levom nogom, desna ruka se pomera unapred, dok se leva ruka pomera unazad. Ovaj način hodanja je prirodan za ljude i doprinosi stabilnosti i efikasnosti hoda.

Implementacija ovog obrasca kretanja u aplikaciji je relativno jednostavna. Svaki put kada model napravi korak, suprotna šaka se pomera unapred, dok se druga šaka pomera unazad. Ovo pomeranje se ostvaruje primenom inverzne kinematike, pri čemu se samo pozicija šake menja, dok ostale kosti ruke prate ovaj pokret. Na Listingu 13 možemo videti način izračunavanja krajnje pozicije šake.

```
private void CalculateArmsLeft()
{
    //Racunanje goal-a za ruke
    rightArmGoal = new Vector3(
        calculatedHipsPos.x + handHipDistance,
        calculatedHipsPos.y + handOffsetForward,
        calculatedHipsPos.z + 0.1f);

    leftArmGoal = new Vector3(
        calculatedHipsPos.x - handHipDistance,
        calculatedHipsPos.y + handOffsetBack,
        calculatedHipsPos.z - 0.07f);
}
```

Listing 13 Računanje krajnje pozicije šake

Na Listingu 13 je prikazan konkretan primer kada model pravi iskorak levom nogom. Pozicija šake može se razložiti na tri komponente. Svaka od ovih komponentata se izračunava na osnovu krajnje pozicije kukova modela. Prva komponenta predstavlja x-koordinatu, koja može tumačiti kao udaljenost između šake i kuka. Ovu vrednost dobijamo dodavanjem konstante na x-koordinatu izračunate pozicije kuka. Druga komponenta odnosi se na y-koordinatu, koja opisuje visinu šake. Da bi se dobila tačna visinu šake, dodaje se mala vrednost na izračunatu visinu kukova. Ova vrednost zavisi od dužine ruku modela i stoga je definisana kao parametar. Treća komponenta je z-koordinata, koja u suštini određuje da li će se ruka pomeriti unazad ili unapred. Kako je u pitanju iskorak levom nogom, na z-koordinatu desne ruke dodajemo određenu konstantu, dok od z-koordinate leve ruke oduzimamo konstantu. Ovim postupkom postićemo sinhronizaciju pokreta ruku sa iskorakom leve noge.

Kako bi se postiglo željeno pomeranje šake od početne do krajnje pozicije, primenjujemo linearnu interpolaciju. Ovaj postupak je detaljno prikazan na listingu 14.

```
private void DoLeftSideLerp()
{
    //...

    //Pomeranje desne ruke unapred dok se leva noga pomera
    rightArmTarget.position =
        Vector3.Lerp(rightArmStartPosition, rightArmGoal, lerp);

    //Pomeranje leve ruke unazad dok se leva noga pomera
    leftArmTarget.position =
        Vector3.Lerp(leftArmStartPosition, leftArmGoal, lerp);

    //...
}
```

Listing 14 Primena Lerp metode za pomeranje šake

Na slici 14 je prikazan izgled ruku kada model iskorači levom nogom.



Slika 14 Izgled ruku nakon levog iskoraka

5.8 Pomeranje ramena

Slično kao i kod pomeranja ruku prilikom hodanja, ljudi takođe pomeraju i ramena, posebno muškarci. Pomeranje ramena se odvija na sličan način kao i pomeranje ruku. Kada leva noga pravi iskorak unapred, desno rame se pomera unapred, dok se levo rame pomera unazad.

Da bi se postiglo ovo pomeranje ramena, primenjena je rotacija na ključne kosti modela. Kada model napravi iskorak levom nogom, desna ključna kost se rotira prema napred, dok se leva ključna kost rotira prema nazad. Proces izračunavanja krajnjih rotacija ramena je detaljno prikazan na listingu 15.

```
private void CalculateShouldersLeft()
{
    //Pomeranje ramena
    rightShoulderGoalRotation = Quaternion.Euler(
        rightShoulderStartRotation.eulerAngles.x,
        rightShoulderStartRotation.eulerAngles.y,
        rightShoulderStartRotation.eulerAngles.z + shoulderAngle);

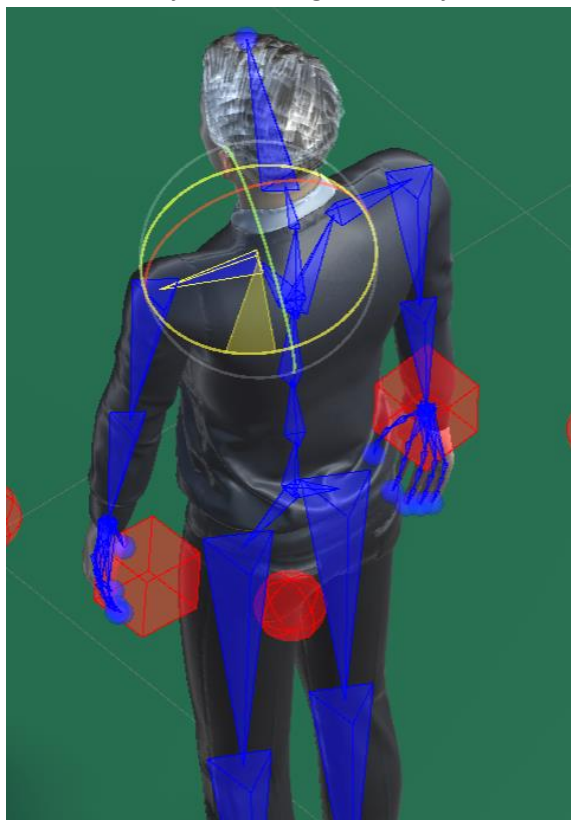
    leftShoulderGoalRotation = Quaternion.Euler(
        leftShoulderStartRotation.eulerAngles.x,
        leftShoulderStartRotation.eulerAngles.y,
        leftShoulderStartRotation.eulerAngles.z + shoulderAngle);

    currentRightShoulderRotation = rightShoulder.localRotation;
    currentLeftShoulderRotation = leftShoulder.localRotation;
}
```

Listing 15 Računanje krajnje rotacije ramena

Rotaciju ramena možemo analizirati kroz tri osnovne komponente: x , y i z . Međutim, u kontekstu promene rotacije ramena tokom hodanja, fokusiramo se samo na komponentu z , budući da ona igra ključnu ulogu u određivanju rotacije ramena unapred ili unazad. Ostale dve komponente, x i y , ostaju konstante i ne menjaju se tokom procesa hodanja.

Povećavanjem vrednosti z komponente, ramena se rotiraju u smeru kazaljke na satu ka napred, što je prikazano na slici 15. Kada je potrebno da se levo rame zarotira unapred, dok desno rame ide unazad, potrebno je povećati z komponentu za određenu vrednost. Suprotno tome, u slučaju obrnutog pokreta, kada je potrebno da desno rame napravi pokret unapred, dok levo ide unazad, obe z komponente se umanjuju za neku vrednost. Ova vrednost je u stvari ugao rotiranja ramena.



Slika 15 Rotacija ramena

Da bismo postigli fluidnu rotaciju ramena od početne do krajnje pozicije, koristimo sferičnu linearnu interpolaciju, odnosno metodu Slerp. Na listingu 16 je demonstrirana primena ove metode.

```
private void DoLeftSideLerp()
{
    //...

    //Slerp ramena
    leftShoulder.localRotation =
        Quaternion.Slerp(currentLeftShoulderRotation, leftShoulderGoalRotation, lerp);
    rightShoulder.localRotation =
        Quaternion.Slerp(currentRightShoulderRotation, rightShoulderGoalRotation, lerp);

    //...
}
```

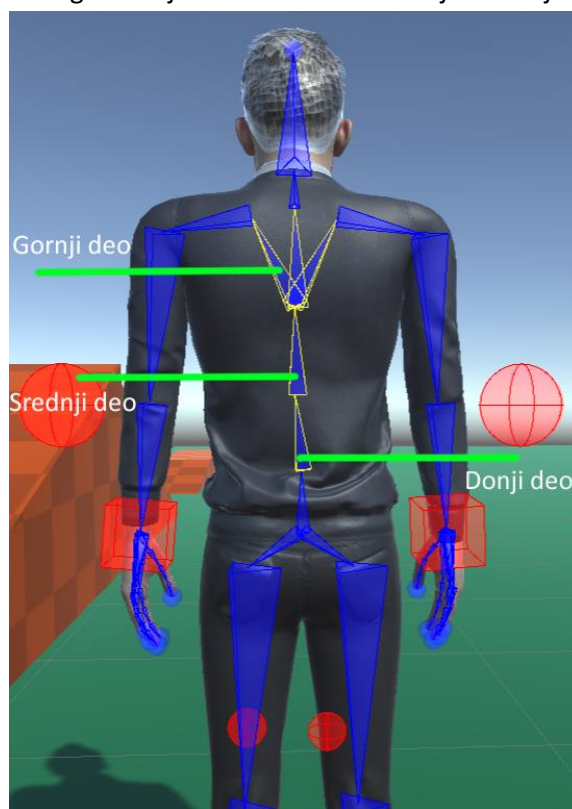
Listing 16 Primena Slerp metode za rotaciju ramena

5.8 Pomeranje torzoa

Da bi animacija hodanja delovala realističnije, važno je uključiti i kretanje torza ili kičmenog stuba. Realistično kretanje torza, odnosno kičme, može biti izazovno za implementaciju, zbog čega su razmotreni samo neki jednostavniji scenariji.

Kada model pravi korak naviše, kao što je slučaj kada se penje uz stepenice, torzo modela se nagnje unapred. Ako model hoda po ravnoj površini ili pravi niži korak, kao što je silazak niz nagib, torzo modela ostaje uspravno. Kada model izvodi veći korak unapred, torzo se nagne unazad. Osim toga, kako bismo efekat rotacije ramena učinili lepšim, potrebno je da torzo modela rotiramo levo i desno.

Kako bismo pojednostavili proces implementacije, kičmeni stub možemo podeliti na tri dela: gornji deo, srednji deo i donji deo, kao što je prikazano na Slici 16. Kada se model nagnje napred ili nazad, pomeramo srednji i donji deo kičme. Za rotaciju torza levo i desno, rotiramo samo gornji deo kičme. Ovom podelom olakšavamo simulaciju realističnog kretanja torza tokom animacije hodanja.



Slika 16 Podela kičmenog stuba

Na Listing 17 možemo videti kod koji se koristi za izračunavanje potrebnih rotacija za svaki deo kičmenog stuba. Srednji i donji deo kičme se rotiraju duž x-ose, što znači napred ili unazad, dok se gornji deo kičme rotira samo duž y-ose, odnosno levo ili desno.

Određivanje tipa koraka se zasniva na razlici u visini između početnog položaja stopala i krajnjeg položaja, kao i razlici u visini između krajnjeg položaja stopala koje čini iskorak i drugog stopala koje ostaje stacionarno. Kada se tip koraka identifikuje, postavljamo rotaciju srednjeg i donjeg dela kičme na odgovarajuće vrednosti. Što se tiče rotacije gornjeg dela kičme, ona ostaje konstantna, ali se mora

pomnožiti sa -1 ako se radi o levom koraku. Ovo određuje da li će se gornji deo rotirati ka levoj ili desnoj strani, pridodavši realističan efekat rotacije ramena tokom hodanja.

```
private void CalculateSpineRotation(Vector3 stepGoal, Vector3 currentStep, Vector3
otherFootPos, bool isLeftLeg)
{
    float zeroHeight = 0.05f;
    float bigStep = 0.9f;
    float xRotationLower = 0f;
    float xRotationMiddle = 0f;
    float yRotationUpper = 10f;
    float footHeightDiff = stepGoal.y - currentStep.y;
    float otherFootHeightDiff = stepGoal.y - otherFootPos.y;

    if (otherFootHeightDiff > zeroHeight)
    {
        //Korak navise
        xRotationLower = lowerSpineForwardAngle;
        xRotationMiddle = middleSpineForwardAngle;

    } else if (otherFootHeightDiff < -zeroHeight)
    {
        //Korak nanize
        xRotationLower = 0f;
        xRotationMiddle = 0f;

    } else if (footHeightDiff <= zeroHeight && footHeightDiff >= -zeroHeight)
    {
        if (Vector3.Distance(stepGoal, otherFootPos) >= bigStep)
        {
            //Digacak iskorak
            xRotationLower = -lowerSpineForwardAngle*1.2f;
            xRotationMiddle = -middleSpineForwardAngle;
        } else
        {
            //Normalan korak
            xRotationLower = 0f;
            xRotationMiddle = 0f;
        }
    }

    if (isLeftLeg)
    {
        yRotationUpper *= -1;
    }

    lowerSpineGoalRot = Quaternion.Euler(
        lowerSpineDefaultRot.x + xRotationLower,
        lowerSpineDefaultRot.y,
        lowerSpineDefaultRot.z);

    middleSpineGoalRot = Quaternion.Euler(
        middleSpineDefaultRot.x + xRotationMiddle,
        middleSpineDefaultRot.y,
        middleSpineDefaultRot.z);

    upperSpineGoalRot = Quaternion.Euler(
        upperSpineDefaultRot.x,
        upperSpineDefaultRot.y + yRotationUpper,
        upperSpineDefaultRot.z);
}
```

Listing 17 Računanje krajnje rotacije torzoa

Kao i u prethodnim slučajevima, koristi se sferična linearna interpolacija kako bi se postigla glatka tranziciju između početne i krajnje rotacije kičmenog stuba. Na listingu 18 je prikazan kod koji primenjuje Slerp metodu kako bi se postigao željeni efekat postepene rotacije kičme tokom animacije hodanja.

```
private void DoLeftSideLerp()
{
    //...

    //Slerp kicme
    lowerSpine.rotation =
        Quaternion.Slerp(lowerSpineCurrentRot, lowerSpineGoalRot, lerp);
    middleSpine.rotation =
        Quaternion.Slerp(middleSpineCurrentRot, middleSpineGoalRot, lerp);

    Vector3 rot = upperSpine.eulerAngles;
    upperSpine.rotation =
        Quaternion.Slerp(upperSpineCurrentRot, upperSpineGoalRot, lerp);
    upperSpine.rotation =
        Quaternion.Euler(rot.x, upperSpine.eulerAngles.y, rot.z);

    //...
}
```

Listing 18 Primena Slerp metode za rotaciju torzoa

6. Korišćenje aplikacije

Kada se aplikacija pokrene, korisnika će dočekati 3D model čoveka [5] i teren sa preprekama. Odmah će biti u mogućnosti da prilagode kameru i zadati komande modelu o tome kuda da se kreće. Na samom korisničkom interfejsu će takođe biti prikazane relevantne informacije za korisnika. U ovom poglavlju biće dat primer kreiranja animacije.

6.1 Komande i korisnički interfejs

Komande koje korisnik može koristiti su jasno prikazane na korisničkom interfejsu, a izgled interfejsa može se videti na slici 17. Komande su sledeće:

- Za rotaciju kamere koristi se pomeranje miša.
- Za translaciju kamere koriste se tasteri W (napred), S (nazad), A (levo) i D (desno).
- Za podizanje kamere koristi se taster Space, dok za spuštanje kamere služi leva Shift tipka.
- Za brisanje definisanih koraka koristi se taster C.
- Za promenu moda koristi se taster M.
- Potvrda komande se izvršava pritiskom tastera E.
- Promena stanja aplikacije iz pauziranog u aktivno, i obrnuto, postiže se pritiskom tastera P.



Slika 17 Izgled korisničkog interfejsa

Korisnički interfejs pruža informacije o statusu aplikacije, precizirajući da li je aplikacija trenutno u pauziranom ili aktivnom režimu. Takođe, pruža informaciju o tome koji korak se trenutno definiše - da li je to iskorak levom ili desnom nogom. Pored toga, korisnički interfejs omogućava praćenje trenutnog moda u kojem se korisnik nalazi.

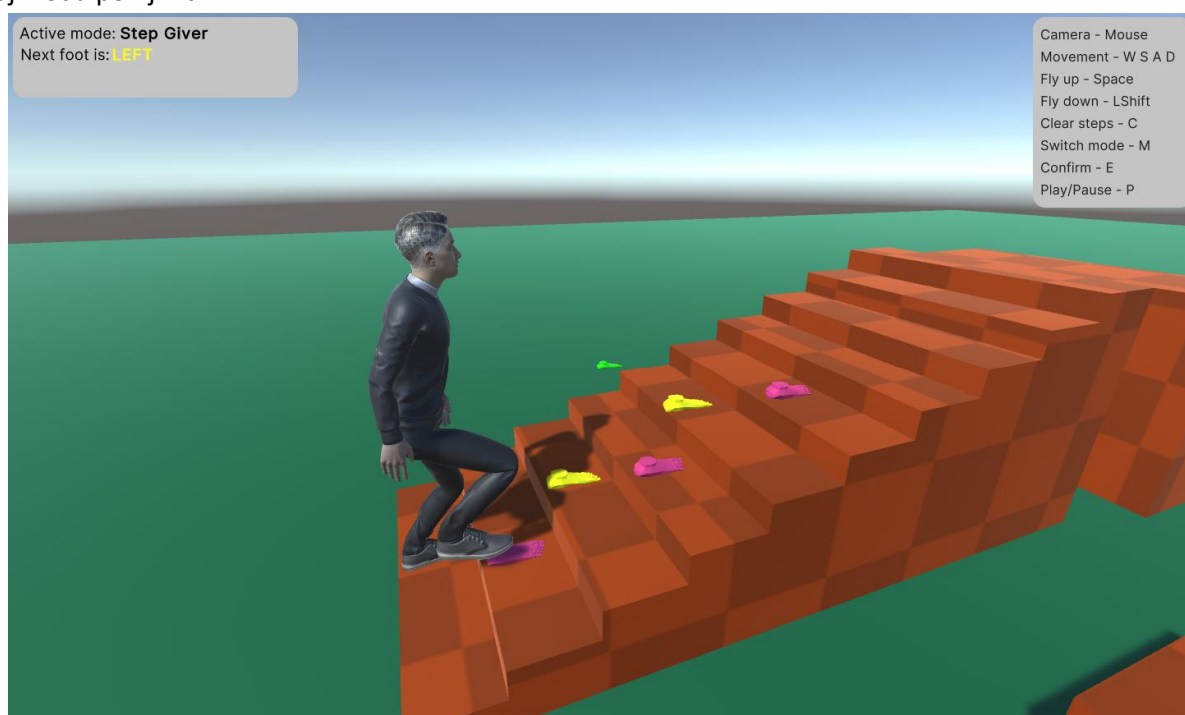
6.2 Modovi

Aplikacija korisniku nudi dva moda tj. režima rada. Prvi od tih režima služi za zadavanje koraka modelu, dok je drugi namenjen za pomeranje samog modela i njegovo postavljanje u drugu početnu poziciju. Prilikom pokretanja aplikacije ona se nalazi u režim za zadavanje koraka. Važno je napomenuti da prelaskom iz režima za zadavanje koraka u režim za pomeranje modela, svi prethodno uneseni koraci će biti obrisani, a aplikacija će preći u pauzirano stanje.

6.3 Primer kreiranja animacije

Da bi se kreirala animacija hoda, prvo je neophodno postaviti kameru tako da bude usmerena prema mestu gde korisnik želi da model napravi prvi korak. Na toj poziciji će se pojaviti marker u obliku zelene stope. Kada korisnik želi da potvrdi mesto prvog iskoraka, označeno markerom, potrebno je pritisnuti taster za potvrdu. Nakon toga, na tom mestu će se pojaviti marker roze boje ukoliko je u pitanju iskorak desnom nogom, ili žute boje ako je u pitanju iskorak levom nogom. Važno je napomenuti da prvi iskorak koje se definiše je desnom nogom, a zatim sledi definicija iskoraka levom nogom.

Kada korisnik definiše sve korake, potrebno je aplikaciju prebaciti iz pauziranog u aktivno stanje. Tada će model početi koračati po definisanim stopama. Na slici 18 je prikazan izgled definisanih stopa i modela koji hoda po njima.



Slika 18 Kretanje modela po definisanim stopama

7. Ograničenja i unapređenja

Kao i svaki softverski proizvod, i ova aplikacija poseduje svoje nedostatke i potencijal za unapređenje. Ovi aspekti zahtevaju pažnju i analizu. U narednom odeljku će biti istaknuta ograničenja ove aplikacije, zajedno s predlozima za potencijalna poboljšanja.

7.1 Realističnije kretanje

Potrebno je poraditi na unapređenju načina na koji ostatak tela reaguje na korake modela. To uključuje postizanje veće realističnosti u pokretima torza, kako bi se bolje simulirao prirodan hod. Takođe je neophodno poboljšati pokrete stopala tokom hoda. Na primer, prilikom hodanja, ljudi podižu stopalo tako da se prvo odvajaju peta, a zatim prsti, i isto tako prilikom spuštavanja stopala, prsti dodiruju podlogu pre pete. Dodatno, trebalo bi dodati mogućnost modelu da skreće levo i desno, jer trenutno ima sposobnost da hoda samo u jednom smeru. Ove promene bi doprinele znatno većem nivou realizma i prirodnosti u animaciji.

7.2 Izbor modela

Korisnicima bi značilo da ima mogućnost izbora vlastitog modela koji žele animirati, budući da trenutno imaju mogućnost animacije samo jednog modela. Osim toga, bilo bi korisno uvesti prilagođenja parametara u zavisnosti od odabranog modela, uključujući brzinu hodanja, dužinu koraka pa čak i stil hoda.

7.3 Izbor drugačije animacije

Aplikacija je prvobitno koncipirana kako bi omogućila kreiranje animaciju hoda, no postoji potencijal za proširenje njenih mogućnosti kako bi se omogućilo kreiranje različitih vrsta animacija. Na primer, mogla bi se dodati funkcionalnosti za stvaranje animacija u kojima model trči ili skače prema unapred definisanim stopama. Takođe, razmotrila bi se opcija za kombinovanje ovih različitih vrsta animacija, omogućavajući korisnicima da stvore kompleksne sekvence, na primer, da model najpre hoda, a zatim pređe u trčanje.

7.4 Mogućnost izvoza animacije

Dodatna funkcionalnost za čuvanje kreirane animacije bi bila izuzetno korisna i poželjna. Ova opcija bi omogućila korisnicima da arhiviraju svoje kreacije i da ih izvoze tj. eksportuju u druge softverske aplikacije radi daljeg uređivanja ili upotrebe.

8. Zaključak

U ovom istraživačkom radu pružen je uvid u implementaciju aplikacije koja primenjuje koncepte inverzne i direktne kinematike za animiranje trodimenzionalnog modela.

Najpre je predstavljen detaljan opis 3D modela i njegovih komponenti, istaknut je značaj skeletnog sistema modela i njegova ključna uloga u procesu animacije.

Rad je dalje produbio razumevanje ključnih koncepata keyframe i proceduralne animacije, istaknuvši njihove primene i razlike između njih. Poseban naglasak stavljen je na karakteristike proceduralne animacije u kreiranju dinamičkih animacija u realnom vremenu.

U narednom segmentu rada je detaljno objašnjena kinematika, pri čemu je istaknuta njena ključna uloga u procesu animacije, kao i različiti tipovi kinematike. Direktna kinematika je ukratko objašnjena, dok je inverzna kinematika detaljnije razmatrana, s obzirom na njenu dominantnu primenu u kontekstu implementacije. Rad je naglasio kompleksnost inverzne kinematike i njenu širu primenu u oblastima izvan animacije, posebno u robotici.

Navedene su korištene tehnologije i obrazloženi razlozi za njihov izbor. Sama implementacija u Juniti pogonu je detaljno prikazana putem priloženih listinga i slika. Svaka komponenta koja čini animaciju hoda je analizirana u dubini, zajedno sa metodama njihove implementacije. Matematički aspekti su prezentovani na pristupačan način, uz vizualne prikaze, slike i formule radi boljeg razumevanja.

U narednom odeljku opisan je način korišćenja aplikacije. Detaljno su opisane komande koje su na raspolaganju korisnicima, zajedno s preciznim instrukcijama o njihovom pravilnom korišćenju. Priložen je i konkretni primer kako zadati korake modelu i generisati animaciju.

Na kraju, obavljena je identifikacija ograničenja aplikacije i istaknute su moguće putanje unapređenja. Naglašava se važnost pažljivog razmatranja ovih aspekata i razmišljanja o budućem razvoju aplikacije. Osim navedenih, postoje i druge potencijalne ideje za unapređenje aplikacije. Na primer, moguće je razmotriti transformaciju aplikacije u video igru, gde bi korisnik preuzeo ulogu modela umesto da upravlja kamerom. Ove dodatne mogućnosti otvaraju vrata za širenje funkcionalnosti i poboljšanja u budućnosti.

Početni problem, iako izazovan na prvi pogled, uspešno je rešen uz upotrebu pravilno odabranih alata i temeljnog istraživanja. Ovaj rad predstavlja solidnu osnovu za sve koji žele da se bave sličnim projektima u oblasti proceduralne animacije uz primenu inverzne kinematike, posebno u okviru Juniti pogona.

Literatura

- [1] A.Aristidou,J.Lasenby,Y.Chrysanthou and A.Shamir, "Inverse Kinematics Techniquesin Computer Graphics: A Survey", http://www.andreasaristidou.com/publications/papers/IK_survey.pdf
- [2] Juniti dokumentacija, <https://docs.unity.com/>
- [3] Animation rigging dokumentacija, <https://docs.unity3d.com/Packages/com.unity.animation.rigging@1.0/manual/index.html>
- [4] Desmos grafički kalkulator za crtanje grafika, <https://www.desmos.com/calculator>
- [5] Mixamo, sajt odakle je preuzet 3D model čoveka, <https://www.mixamo.com/>

Podaci o kandidatu

Strahinja Eraković je rođen 2001. godine u Kikindi. Osnovnu školu “Žarko Zrenjanin” završio je 2015. godine u Kikindi. Nakon toga, završio je Srednju Tehničku školu u Kikindi 2019. godine. Fakultet Tehničkih Nauka u Novom Sadu upisao je iste godine. Ispunio je sve obaveze i položio je sve ispite predviđene studijskim programom sa prosečnom ocenom od 8.97.