TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Kalev Kuusk 163967IASB

INTERAKTIIVSE VIRTUAALSE PUU KONTSEPTSIOON JA MÄNGU PROTOTÜÜP

Bakalaureusetöö

Juhendaja: Eduard Petlenkov
PhD

Tallinn 2019

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Kalev Kuusk

20.05.2019

Annotatsioon

Antud lõputöö eesmärgiks oli luua liitreaalsusel põhinev interaktiivne mängu prototüüp ja virtuaalse puu kontseptsioon. Töö eesmärkideks olid samuti õppida tundma mängude loomise protsessi Unity mängumootoris ja mudelite ning animeerimise protsessi Blender tarkvaras.

Mängu loomisega tegeles 2 tudengit. Lõputöö autori ülesanneteks oli luua mängu funktsionaalsus, mänguloogika, mängukeskkond ja mängus kasutatavad mudelid. Lõputöö sisu selgitab nende ülesannete lahendamise töökäiku, esinenud probleeme ja saadud tulemusi.

Lõputöö tulemuseks saadi töötav mängu prototüüp, mis andis parema arusaama liitreaalsuse mängu loomise protsessidest ja valitud vahendite puudustest ning eelistest. Valminud lahenduses reageerib virtuaalses maailmas olev ahv reaalses maailmas olevate mängijate liigutustele ja asukohale. Lisaks on mängijatel võimalik võtta kätte virtuaalses maailmas asetsev toit, mis muudab ahvi käitumist. Mäng sisaldab ka loodud puu kontseptsiooni ja animeeritud mudeleid.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 22 leheküljel, 6 peatükki, 21 joonist.

Abstract

Interactive Virtual Tree Concept and Game Prototype

The purpose of this thesis project was to create a game prototype based on augmented reality and a virtual tree concept. Additionally, the aim of this project was to get a better understanding of the game making process in the Unity game engine along with modelling and animating process in the Blender software.

The game making process involved 2 students of Tallinn University of Technology. The author's tasks in this thesis project were to develop the game functionality, game logic and to create the models used in the game. The content of the thesis explains the process of creating the game and models, describes the problems that are present in the final game prototype along with a full view of the finished game.

The result of this thesis work was a fully functional augmented reality game prototype using the Microsoft Kinect as an input device. The game also includes a virtual tree concept and a fully animated animal model. The process of creating this game gave a better understanding of augmented reality game development and showed the benefits and flaws of the chosen hardware and software. It also showed the process of modelling, texturing and animating.

The finished game includes a virtual animal which reacts to the movement and positions of the players in the real world. Additionally, the players have the option to pick up food which changes the way the virtual animal reacts. By default the animal is afraid of the player and tries to escape. If the player picks up the food and stays calm, then the animal would get friendlier towards the player and would start approaching the player. Any sudden movement would scare the animal.

The thesis is in Estonian and contains 22 pages of text, 6 chapters, 21 figures.

Lühendite ja mõistete sõnastik

Script	Unity keskkonnas kasutatav komponent, mis sisaldab C# programmeerimiskeeles kirjutatud koodi [2].
Tag	Unity keskkonnas kasutatav viit, mida on võimalik määrata mänguobjektidele [3].
UV mapping	2D pildi 3D kujule projekteerimine tekstuuri määramise eesmärgil [4].
Hitbox	Nähtamatu kujund mudeli ümber kokkupõrgete tuvastamise eesmärgil [5].
Frame	Kaader, üks paljudest liikumatutest piltidest. Mitu kaadrit koos moodustavad animatsiooni [6].
Keyframe	Kaader, milles on salvestatud mingid väärtused [7].
2D	Kahemõõtmeline
3D	Kolmemõõtmeline

Sisukord

Jooniste loetelu

1 Sissejuhatus

Tallinna Tehnikaülikooli Mektory soovis luua innovaatilist lahendust, mis rakendaks keskse elemendina elus puud. Meie meeskonna poolt välja pakutud lahendus kasutaks seda puud loodava liitreaalsuse mängu keskse objektina. Sama puu oleks kujutatud ka virtuaalses maailmas, kus puu küljes oleks loom, kes reageeriks reaalses maailmas olevate inimeste tegevustele ja asukohale puu suhtes. Inimestel oleks looma tegevust võimalik jälgida läbi ruumis asuvate ekraanide. Järgnev lõputöö tutvustab selle mängu prototüübi valmimise protsessi mängu loomise vaatepunktist kasutades Unity mängumootorit. Töö eesmärgiks on saada töötav mängu prototüüp koos vajaliku mängu funktsionaalsuse, mänguloogikaga ja visuaalidega.

Lõputöö teine peatükk tutvustab mängu ideed täpsemini ja selgitab mängu loomiseks valitud vahendite põhjuseid. Kolmandas peatükkis antakse ülevaade mängu loomise protsessist Unity keskkonnas ja mudelite loomise protsessist Blender keskkonnas. Lõputöö neljas peatükk annab ülevaate loodud mängu prototüübist ning viies peatükk arutleb prototüübi probleemidest ja edasiarendamise võimalustest.

2 Projekti idee

2.1 Projekti algne idee

Projekti ideeks oli luua interaktiivne liitreaalsuse mäng. Idee sai alguse Mektorys asuvast taimetoast, kuhu soovitakse luua lahendust, mis näitaks Taltech'i innovatiivsust hoone külastajatele. Selle toa keskseks elemendiks on puu, mida sooviti kasutada peamise elemendina loodavas lahenduses. Mängu idee oli kujutada see sama puu ja puu küljes olev loom virtuaalses maailmas. Loom liiguks vastavalt inimese asukohale ja liigutustele. Kui inimene läheneks asukohale, kus loom on kujutatud siis üritaks loom inimese eest põgeneda. Mängus kuvatakse ka mingi toit või ese, mida on inimesel võimalik kätte haarata, mille peale loom läheneb inimesele ja üritab seda ära võtta. Mängus toimuv kuvataks ruumis asuvale ekraanile, mille järgi oleks inimesel võimalik orienteeruda.

Tegemist on meekonna tööga, mis jagunes kaheks. Osa projektist on kirjeldatud Renee Gustasson „Microsoft Kinecti rakendamine liigutuste tuvastamiseks interaktiivse virtuaalse puu jaoks" [1] lõputöös. Autori roll projektis oli mängu loomine Unity keskkonnas. Renee roll oli Microsoft Kinect sensorit rakendades realiseerida inimese ja liigutuste tuvastus.

2.2 Vahendite ja keskkonna valik

Loodava mängu kontrollimiseks oli vaja valida andur, mis oleks võimeline tuvastama inimese ja tema liigeste asukohta kolmemõõtmelises ruumis ning erinevaid liigutusi nagu näiteks käe haardesse minek. Samuti oli vaja ruumis toimuvast pilti, mida kuvada mängu taustale. Anduriks valisime Microsoft Kinect versioon 1, mis on valmis lahendus täites kõiki eelnimetatud vajadusi. Kinectist ja sellega töötamisest räägib täpsemini tiimikaaslase lõputöö [1].

Keskkonna valikul oli oluline, et see võimaldaks sellise liitreaalsuse mängu loomist. See pidi võimaldama Microsoft Kinecti abil mängu kontrollimist ja kaamera pildi taustale

kuvamist. Sellest tulenevalt valisime keskkonnaks Unity mängu mootori kuna see täitis meie nõuded, oli lihtne kasutada koos põhjalikku dokumentatsiooni ja õpetustega ning on tasuta kasutamisõigusega. Unity keskkonnast räägib täpsemini alapeatükk 2.2.1.

Mudelite ja animatsioonide valmistamiseks pidime alguses kasutama Unity keskkonda sisseehitatud võimalusi, kuid töö käigus selgus, et Unity ei olnud piisavalt võimas ja sobilik meie lõputöö otstarbeks. Seega oli tarvis spetsiifilisemat modelleerimise ja animeerimise tarkvara, milleks valisime Blender keskkonna. See täidab kõik meie nõuded ja on vabatarkvara. Blender keskkonnast räägib täpsemini alapeatükk 2.2.2.

2.2.1 Unity keskkonna tutvustus

Unity mängumootor avalikustati maailmale esmalt 2005 aastal. Aastal 2018 toetab Unity üle 25 erineva platvormi ja mängumootorit saab kasutada 3D, 2D, virtuaalreaalsus kui ka liitreaalsus mängude loomiseks. Unity mängumootorit kasutab üle poolte mobiili mängudest ja virtuaal- ning liitreaalsuse aplikatsioonidest moodustab Unity üle 60% [8]. Unity kasutamiseks on olemas kaks versiooni: tasuline ja personaalne tasuta versioon. Personaalset versiooni võib kasutada seni kuni sellega valmistatava toote tulu ega toetused ei ületa 100 000$ [9].

Unity peamiseks programmeerimiskeeleks on C#. Mängude disainimiseks on olemas ka 3D disainer koos *drag and drop* (tõmba ja lase lahti) funktsionaalsusega [8].

Valmis mängumootorite nagu Unity kasutamine annab arendajale lihtsa ja funktsionaalse keskkonna, kus mänge toota. Tänu sellele ei pea arendaja enam tegelema näiteks füüsika, graafika, valgustuse ja muu sellise programmeerimisega.

2.2.2 Blender keskkonna tutvustus

Blender on vabatarkvaraline 3D graafika programm [10]. Seda programmi kasutatakse näiteks animeeritud filmide, visuaalsete efektide, kunsti, 3D mudelite modelleerimise ja videomängude loomise eesmärkidel. Lisaks võimaldab see 3D objektide projekteerimist 2D pinnale ehk *UV mapping*, tekstuuride valmistamist ja värvimist, suitsu, vedeliku ja pehmekehade füüsika simuleerimist. Valminud töödest saab programmi abil ka toota videoid ning kõrge kvaliteediga pilte [11, 12].

3 Mängu loomise töö käik

3.1 Mängu disaini valikud ja nõuded

Mängu põhitegelaseks valisime meeskonnaga ahvi, looma meelitavaks toiduks banaani ja mängu keskseks puuks palmipuu. Mängu välimuse valikuks osutus minimalistlik disain. Puu on disainitud roheliseks või pruuniks värvitud sfääridest, ahv ja banaan aga kandilistest kujunditest.

Mäng funktsionaalsusele seatud nõuded:

- Ahv saab liikuda ainult mööda puulehti ja peab kogu aeg vähemalt ühe lehe küljes rippuma.
- Ahv põgeneb inimese eest kui inimene ei hoia banaani käes.
- Ahv põgeneb inimese eest kui inimene hoiab banaani aga teeb järske ootamatuid liigutusi.
- Ahv läheneb inimesele kui inimene hoiab banaani ja inimene ei liiguta või liigutab rahulikult.
- Kui ahvi puudutada põgenemise hetkel ilmub mängu ebaõnnestumise lõputekst. Lähenemise hetkel puudutamine toob nähtavale mängu õnnitlemise lõputeksti.
- Banaan asetatakse mängu mistahes leheobjekti külge, mis ei ole tüve lähedal.
- Kui mängija laseb banaanil käest kukkuda tekitab mäng puu külge uue banaani.
- Banaani kätte võtmiseks peab mängija oma käe banaani asukohta viima ja käe haardesse panema.

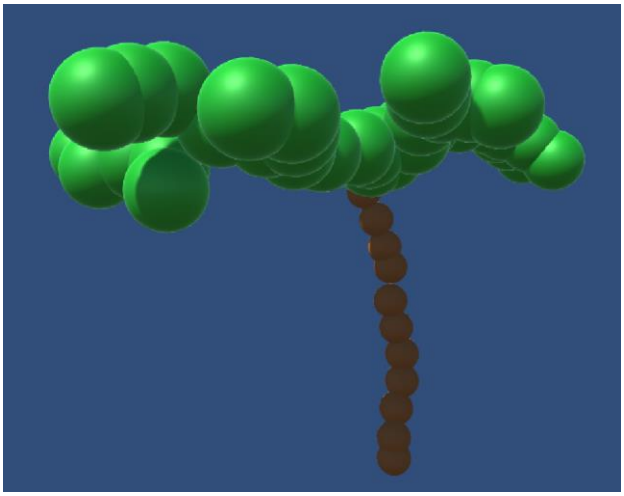
3.2 Mängu ja selle loogika loomine Unity keskkonnas

Mängu loomine algas Unity keskkonnas mängu esimese katselise versiooni loomisega. See sisaldas kahte käeobjekti, mida kontrollis C# programmeerimiskeeles loodud *script*. Käed liikusid kindlate ettemääratud punktide vahel püsides igas punktis teatud aja. Lisaks sisaldas mäng looma kujutavat objekti, mis üritas käte eest põgeneda vabalt 3D ruumis liikudes.

Selle versiooni eesmärk oli tutvuda Unity võimaluste ja kasutamisega. See õpetas tundma objektide liigutamist ruumis ja nende vaheliste seoste loomist. Lisaks sisaldas

see versioon funktsionaalsust, mis tuvastas objektide omavahelisi kokkupõrkeid. Sellest versioonist jõudis lõpliku mängu vaid lihtne loogika ja üksikud funktsioonid.

Teise mängu versiooni loomise eesmärgiks oli keskenduda virtuaalse puu loomisele ja looma liikumisele puu suhtes. Puu loomiseks olid kasutusel tavalised roheliseks ja pruuniks värvitud sfäärid (Joonis 1). Puu lehe ja tüve eristamiseks on igal objektil määratud *tag* vastavalt *Leaf* või *Trunk*. Valmis ka esimene *script*, mis oli võimeline mängus oleva puu infot faili salvestada või failist loetud info abil mängu puud tekitada. See *script* on osa ka lõplikust mängust, kus seda kasutatakse mängu alguses puu tekitamiseks. Selle *scripti* olemasolu võimaldab salvestada mitut erinevat puu disaini eraldi tekstifailides (Joonis 2). Lisaks võimaldab see funktsionaalsuse, kus mängust välised programmid või isikud saaksid mängu tekkivat puud ise disainida või manipuleerida. (Lisa 5)



Joonis 1. Esialgne puu disain.

```
Trunk
(8.8, 3.2, 8.4)
Trunk
(8.8, 2.9, 8.4)
Leaf
(11.3, 7.9, 7.3)
Leaf
(11.1, 7.7, 8.5)
```

Joonis 2. Puu infot hoidva tekstifaili sisu näide.

Järgmisena valmis uus käte kontroller, mis võimaldas käte juhtimist Xbox One kontrolleri abil. Käsi sai ruumis vabalt liigutada X ja Z koordinaatidel kasutades kontrolleri paremat ja vasakut *joystick'*i ja Y koordinaatidel liigutamiseks kasutati

kontrolleri nelja nuppu. See *script* valmis mängu testimise eesmärgil ja lõplikus mängus kontrolleriga enam käsi kontrollida ei saa. See *script* vastutab ka kokkupõrgete tuvastamise eest, et kontrollida banaani haaramist ja loomaga kokkupõrkeid. Kui tuvastatakse käe *hitbox* sisenemist näiteks ahvi või banaani *hitbox*'i saadetakse selle kohta teade vastavasse funktsiooni. Banaanist kinni hoidmine on realiseeritud käte kontrolleris ja ahviga kokkupõrgete info edastatakse looma kontrollerisse. (Lisa 2 ja 3)

Edasi toimus looma kontrolleri *script*'i arendus. *Script* otsib esmalt mängust üles parema ja vasaku käeobjektid, et edaspidi teada nende asukohta maailmas. Samuti otsib *script* üles kõik puulehe objektid kasutades varem määratud *tag*'i *Leaf* ja asetab kõik leitud objektid massiivi. See *script* vastutab ka looma põgenemise eest kontrollides iga Unity *Update* tsükli jooksul distantsi käte ja looma vahel. Kui vahemik on väiksem, kui määratud miinimum üritab loom leida uut leheobjekti kuhu liikuda. Leheobjekti leidmiseks käiakse läbi varem määratud leheobjektide massiiv üritades leida leheobjekti, millele liikudes suureneks distants käte ja looma vahel. Otsingule on lisatud ka maksimum leheobjekti kaugus, et vältida liiga suuri hüppeid leheobjektide vahel. (Lisa 1)

Antud põgenemise loogika töötas, kuid tulemus oli hüplik ja ettearvatav, kuna leheobjektid olid jadas samas järjestuses ja seega kordusid välja valitud leheobjektid tihti. Selle vea parandamiseks sai lisatud enne otsingut leheobjektide massiivi sorteerimine. Sorteerimise tulemuseks oli looma ja leheobjektide vahelise kauguse poolest kasvav jada, kust valiti esimene lähim sobiv leheobjekt. Tehtud muudatus parandas põgenemise ettearvatuse probleemi. Hüplikuse probleemi õnnestus paremaks muuta dünaamilise kiiruse arvutuse lisamisega. Loom üritab põgeneda kiiremini, mida lähemal on käed tema asukohale.

Järgmisena valmis mängu kontroller, mis vastutab uue mängu alustamise ja lõpetamise eest ning puuvilja mängu paigutamise eest. Puuviljale sobiliku koha leidmiseks otsib *script* üles kõik *Leaf tag*'iga objektid ja valib mistahes leheobjekti, mille külge puuvili kinnitada tingimusel et see on tüvest teatud miinimum kaugus eemal. Samuti jälgib see *script* banaani asukohta ja kui tuvastatakse banaani mänguväljast välja kukkumine siis hävitatakse banaaniobjekt ja paigutatakse uus mängu. (Lisa 4)

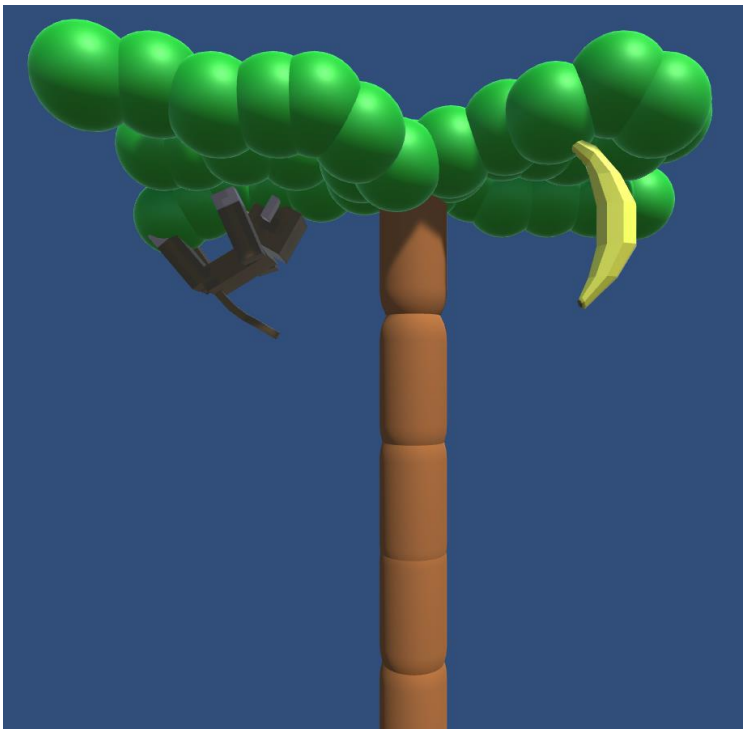
Mängu lisandusid nüüd ka mängu lõppu näitavad ekraanid (Joonised 3 ja 4). Ekraanidel kuvatav tekst ja värv annab mängijale teada tema tulemustest. Ekraanide huvitavamaks muutmiseks on neile ka lisatud animatsioonid, kus esimesena ilmub nähtavale taustvärv, siis tekst ning pärast teatud ajavahemikku algab uus mäng. Uue mängu alustamiseks kutsutakse mängu kontrolleri *script*'ist välja vastav funktsioon.

Joonis 3. Mängu positiivne lõpuekraan.

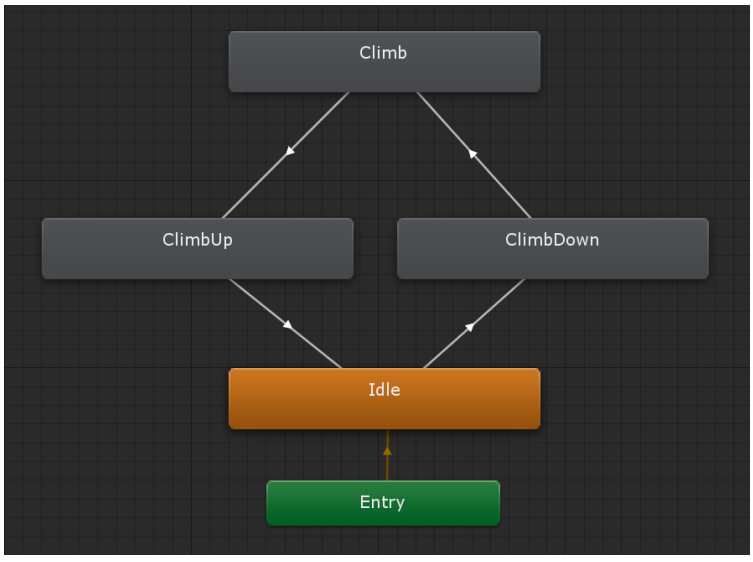
Joonis 4. Mängu negatiivne lõpuekraan.

Projektis oli selleks hetkeks realiseeritud kõik vajalik funktsionaalsus ja algas tiimikaaslase projektiga ühendamine. Kinectist tulev käte asukoha ja haardes oleku info on suunatud käte kontrolleritesse, kus see asendas esialgse kontrolleri sisendid. Ilmnes ka esimene suurem probleem, kus valminud mängu maailm oli tunduvalt suurem kui Kinecti poolt kontrollitavate käte liikumise ala suurus. Selle tulemusel tuli kogu mängumaailm väiksemaks skaleerida ja puu uuesti ehitada. Lisaks tuli arvestada puu disainimisel Kinecti vaateväljaga, et vältida olukordi, kus puu leheobjektid on kättesaamatud.

Edasine mängu arendus keskendus mängu välimusele. Puu leheobjektid jäid roheliste sfääridena ja puu tüveobjektid asendusid pruunide kapsli kujuliste elementidega (Joonis 5). Banaani ja ahvi pidid plaani järgi visualiseerima detailsemad mudelid. Sellest tulenes aga uus probleem, kuna selgus et Unity sisene mudelite loomise ja animeerimise võimalused ei olnud nii detailsed kui alguses lootsime. Seetõttu pidi mudelite loomiseks hakkama kasutama uut programmi, milleks osutus Blender. Mudeli loomisest ja animeerimisest on kirjeldatud alapeatükis 3.3. Pärast mudeli valmimist ja Unity keskkonda konverteerimist tuli seadistada ja määrata animatsioonid. Lisaks valmis animatsiooni kontroller, mis kontrollib animatsioonide järjekorda ja kiirust (Joonis 6). Vastavalt ahvi liikumisele seadistab looma kontroller, kas liikumine on aktiivne või mitte. Selle muutuja alusel kontrollib animatsiooni kontroller, mis animatsioon peab sel hetkel mängima ja kuidas erinevate animatsioonide vahel liikuda.



Joonis 5. Uus puu disain.



Joonis 6. Animatsiooni kontroller.

Järgnes mängu viimistlemine ja testimine, mille käigus lisandusid käte asukohta visualiseerivad läbipaistvad sinised sfäärid, et anda kasutajale parem arusaam tema käe asukoha kohta mängus.

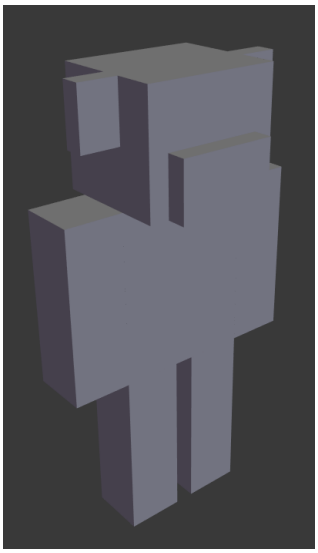
3.3 Mudelite loomine Blender keskkonnas

3.3.1 Ahvi mudeli loomine

Mudeli loomine ja animeerimine toimus Blender keskkonnas, mis osutus üheks raskemaks projekti osaks. Suurimaks probleemiks oli keskkonna kasutama õppimise keerukus, kuna suurem osa programmi funktsionaalsusest on peidetud menüüdesse või klaviatuuri otseteede taha.

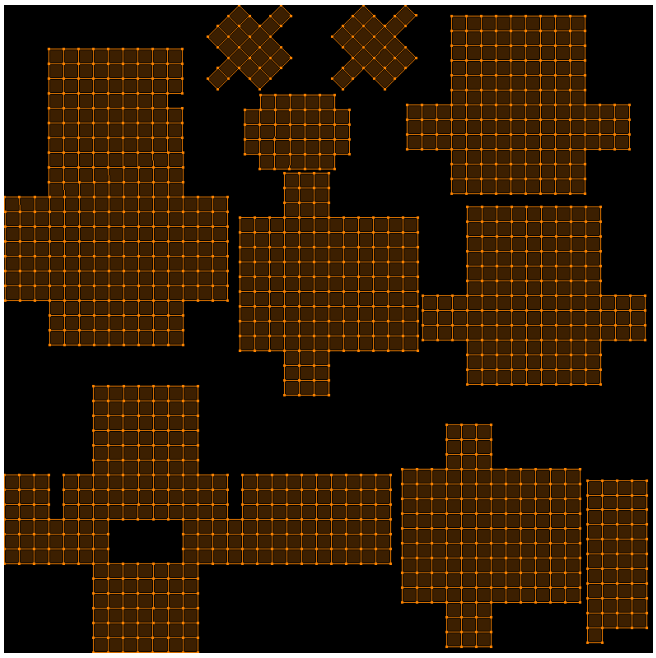
Ahvi mudeli ja disaini inspiratsiooniks oli internetist leitud ahvi kontseptsioon, mis oli esialgu loodud videomängu jaoks [14]. Ahvi modelleerimise ja hiljem animeerimise lihtsustamiseks sai valitud võimalikult lihtne keha mudel, mis koosnes 7 kehaosast: 2 jalga, 2 kätt, keha, pea ja 5 lüliga saba.

Ahvi mudeli loomine algas ahvi keha kujundamisega. Ahvi kõik kehaosad valmisid esialgu eraldiseisvate osadena, et tagada hilisem lihtsam animeerimise ja tekstuuri kujundamise protsess. (Joonis 7)



Joonis 7. Ahvi mudel.

Sellele järgnes *UV mapping* ehk 2D pildi 3D kujule projekteerimine tekstuuri värvimise eesmärgil [4]. Selle tegevuse käigus tuli märkida kujule tekstuuri liitekohad ja projekteerida antud 3D objekt 2D pildile. Seejärel tuli jaotada 2D pildil kehaosad nii, et need ei kataks üksteist. Samuti tuli keha jaotada sel hetkel väiksemateks osadeks ehk piksliteks, mida hiljem värvida saaks. (Joonis 8)



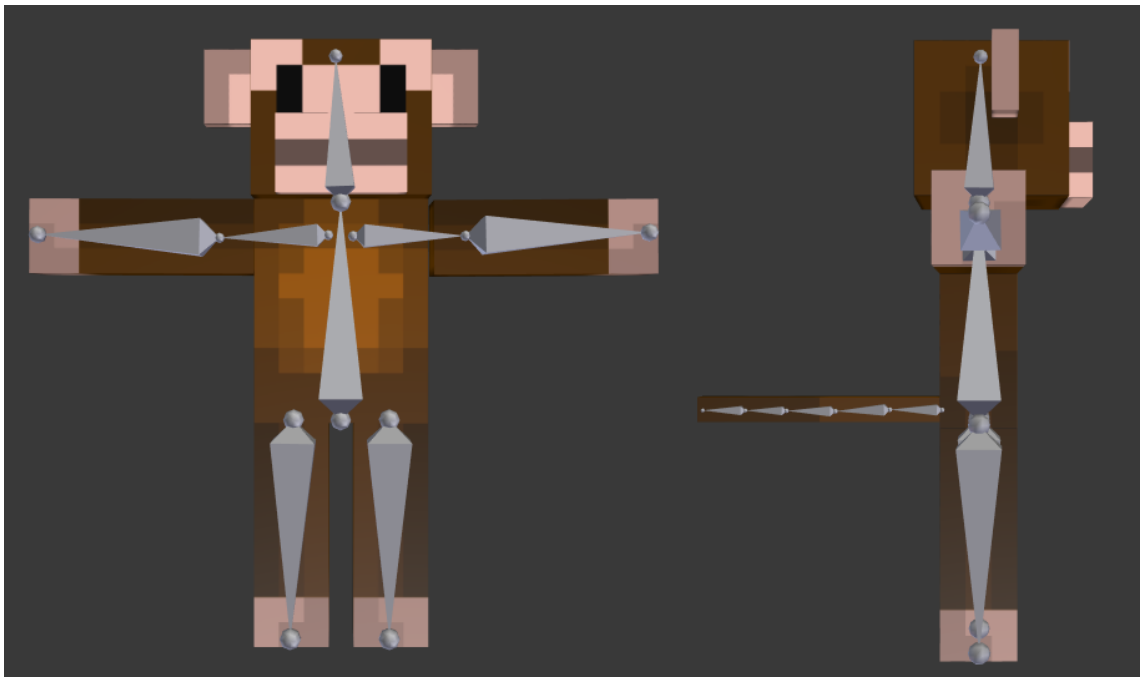
Joonis 8. Ahvi *UV map*.

Järgmiseks tuli keha tekstuur värvida. Seda protsessi lihtsustas varem lisatud pikslite ruudustik, kus iga piksel sai täidetud ühe kindla tooniga. Protsessi tulemuseks oli täielikult värvitud *UV map*, mis salvestati eraldi PNG failina ja hiljem mängu imporditi. (Joonis 9 ja 10)

Joonis 9. Värvitud *UV map*.

Joonis 10. Valmis ahvi mudel.

Ahvi mudel oli selleks hetkeks valmis (Joonis 10), edasine töö toimus animeerimise eesmärgil. Animeerimise esimeseks sammuks oli ahvile luukere ehitamine. Tänu lihtsale mudelile sai ahvi keha täielikult artikuleeritud kasutades 13 konti, millest 8 on keha ja 5 saba konti (Joonis 11). Lisaks tuli manuaalselt määrata ära millised kehaosad millistele kontidele kuulusid. See protsess oli tunduvalt lihtsustatud tänu modelleerimisel tehtud valikule valmistada kõik kehaosad eraldiseisvate kujunditena. Selle tulemuseks oli täielikult artikuleeritud jäsemed ja pea ilma tekstuure venitamata. Kahjuks saba puhul tekstuuri venimine esineb aga kuna sabal on kasutusel ainult 2 tooni on seda vähem märgata.



Joonis 11. Ahvi mudeli kondid.

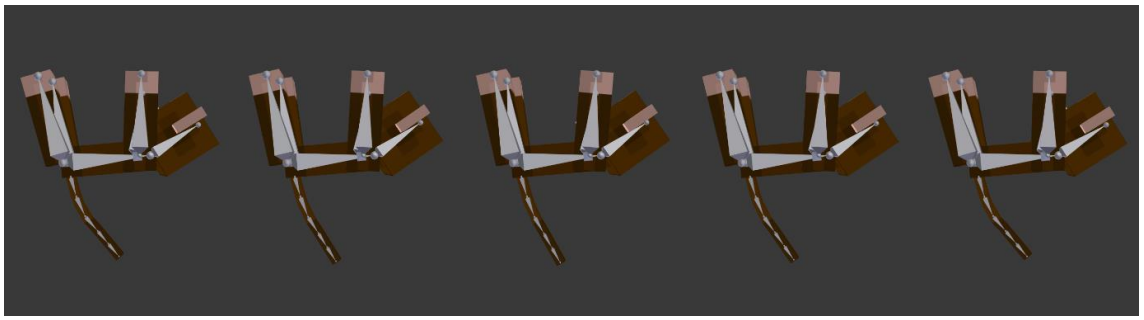
3.3.2 Ahvi mudeli animeerimine

Animeerimise protsessi eesmärgiks on panna mudel tegema liigutusi mingi aja vältel [13]. Mängu tarbeks oli vaja luua 4 erinevat animatsiooni: puhkepositsioon, ronimine, puhkepositsioonist ronimispositsiooni vahetumine ja ronimispositsioonist puhkepositsiooni vahetumine.

Animeerimise protsessi alustuseks tuleb valida animatsiooni kaadrisagedus, mis meie projektis on 24 kaadrit sekundis. Samuti tuleb valida animatsiooni pikkus kaadrites (*frame),* mis on märgitud järgnevates animatsioonide näidetes. Animeerimiseks tuleb määrata kondi asukohad manuaalselt iga või mõne kaardi puhul, mis muudab selle vastava kaadri *keyframe*'iks. *Keyframe* on kaader, milles on salvestatud mingi väärtus [7], antud töö puhul ahvi või kondi asukoht. Kui on kondile asukoht määratud kõigis kaadrites siis liiguvad kondid sujuvalt kõigi määratud asukohtade vahel. Kui määrata kontide asukoht ainult mõnes kaadris siis arvutab Blender neile sujuva liikumise tee ise ülejäänute kaadrite jaoks välja. Animatsioonide sujuvuse ja nende vahel vahetamise lihtsustamiseks on valitud 2 standard asendit, milleks on rippuv ja puu külge haakunud. Näiteks algab ja lõppeb ronimise animatsioon rippuva asendiga, puhkepositsiooni puhul puu külge haakunud asendiga. Vahetumise animatsioonide algus ja lõpp on samuti need kaks positsiooni.

Animatsioonide näitamiseks on välja valitud 5 *keyframe*'i, mis on kujutatud järgnevate piltide peal.

Puhkepositsioon ehk *idle* on kasutusel, kui ahv püsib paigal ja kasutab 32 kaadrit (Joonis 12).

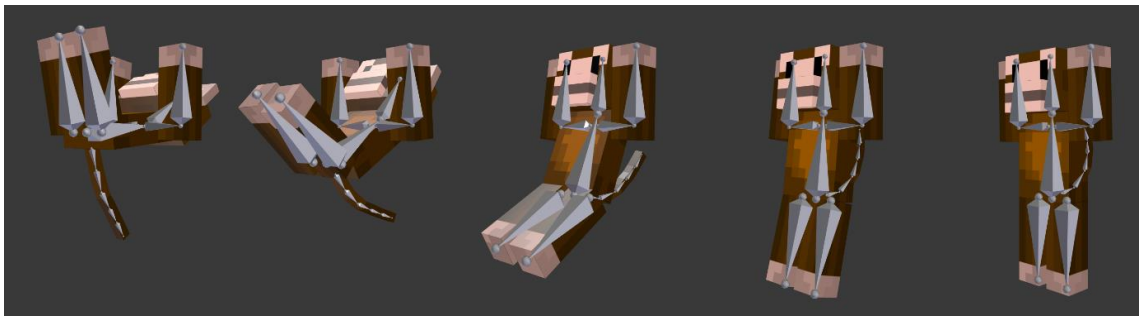


Joonis 12. Puhkepositsioon ehk *Idle*.

Ronimine ehk *Climb* on kasutusel ahvi liikumise ajal ja kasutab 32 kaadrit (Joonis 13).

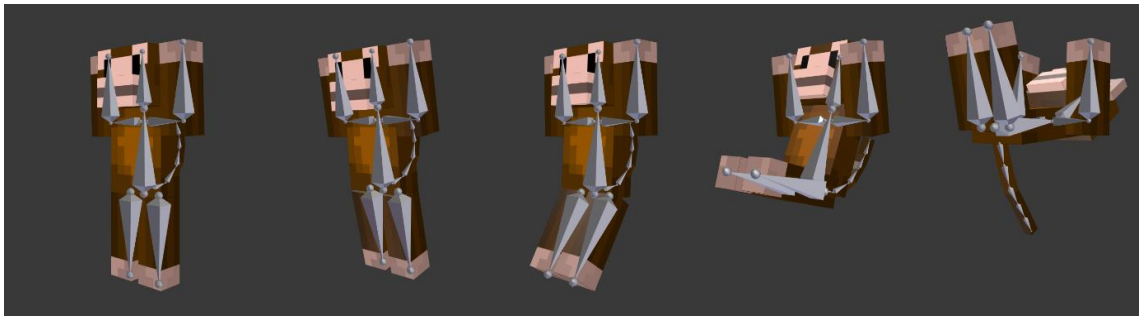
Joonis 13. Ronimine ehk *Climb*.

Puhkepositsioonist ronimispositsiooni vahetumine ehk *Climb down*, kasutab 24 kaadrit (Joonis 14).



Joonis 14. Puhkepositsioonist ronimispositsiooni vahetumine ehk *Climb down*.

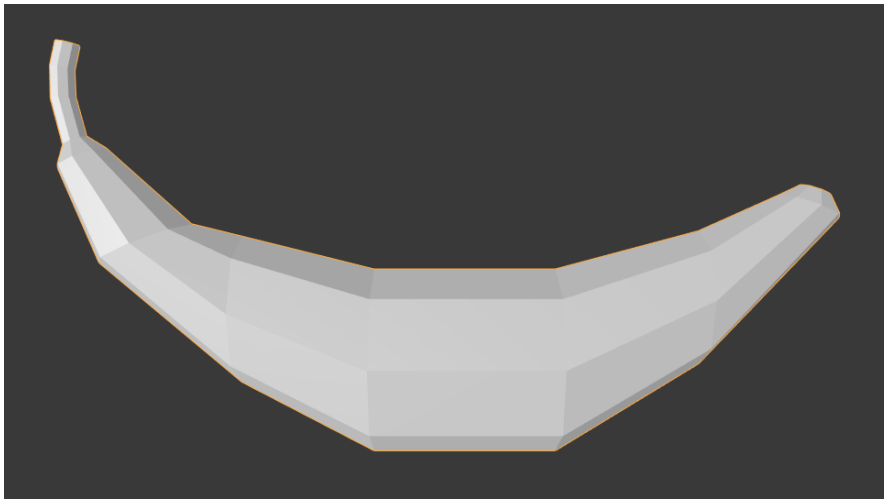
Ronimispositsioonist puhkepositsiooni vahetumine ehk *Climb up*, kasutab 24 kaadrit (Joonis 15).



Joonis 15. Ronimispositsioonist puhkepositsiooni vahetumine ehk *Climb up*.

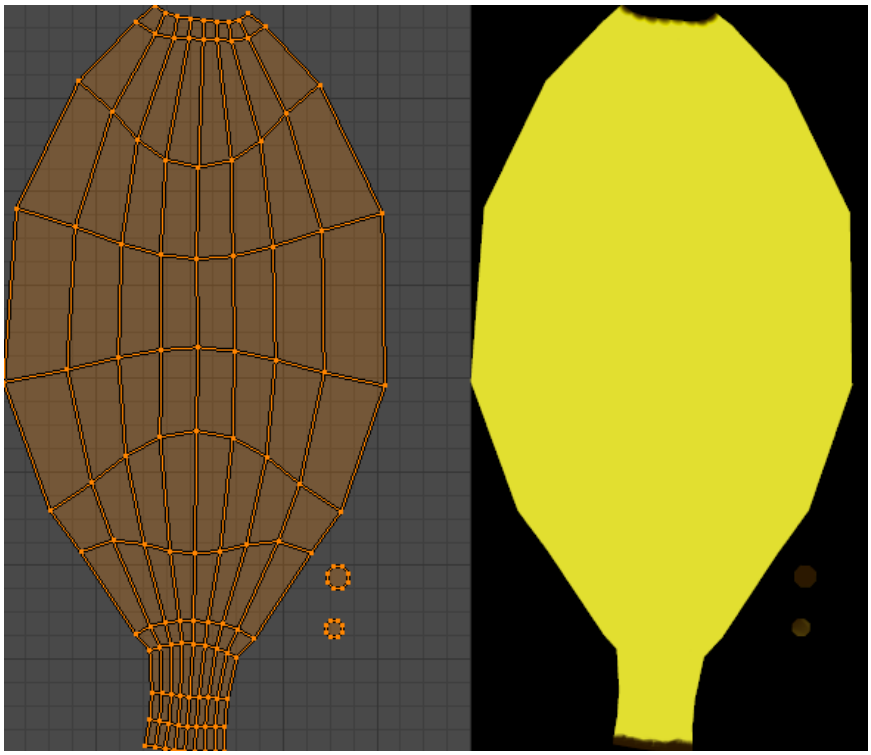
3.3.3 Banaani mudeli loomine

Banaani mudeli loomine algas 8 tahulise silindri modelleerimisest, kuni saavutati soovitud kuju ja on seega üks terviklik keha (Joonis 16).

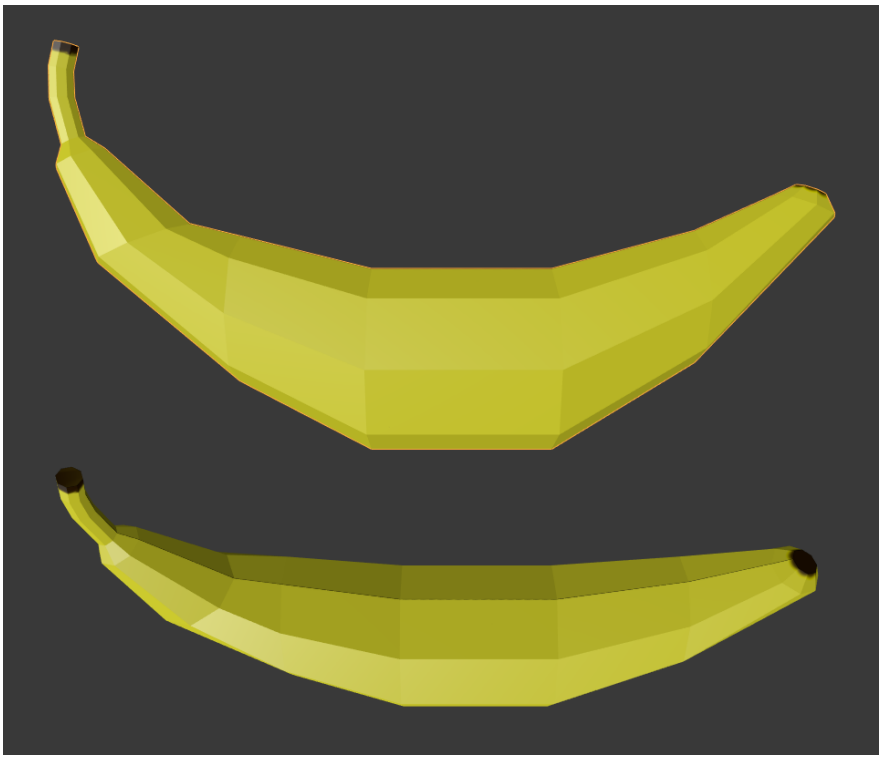


Joonis 16. Banaani mudel.

Järgnes *UV mapping*, mille käigus eraldati mõlemad banaani otsad ja banaani pind laotati laiali 2D tasandile. Banaani tekstuuri värvimise tulemusel on see üleni ühtlane kollane koos pruunide otsadega. (Joonis 17 ja 18)



Joonis 17. Banaani *UV map* ja värvitud *UV map*.



Joonis 18. Valmis banaani mudel.

4 Tulemused

Lõputöö tulemuseks valmis mängu prototüüp, mis täidab kõik mängule seatud funktsionaalsuse nõuded. Lahenduses esineb küll vigu, mis mõjutavad kasutajakogemust aga need vead ei muuda mängu mängimist võimatuks.

Mängu alguses tekitatakse virtuaalsesse maailma puu vastavalt tekstifailist saadud infole. Puu külge kinnitatakse ahvi mudel, kes hakkab jälgima käte asukohti ja nende järgi otsuseid langetama. Juhuslikult valitud leheobjekti külge kinnitatakse ka banaan.

Kui mäng on alanud, siis on mängijal võimalik hakata kontrollima mängus olevaid käeobjekte. Mäng on võimeline tuvastama mängijat ja asetab tema käsi visualiseerivad käeobjektid vastavatesse kohtadesse virtuaalses maailmas. Mängijal on võimalik puu küljes olevat banaani kätte võtta. Selleks peab mängija virtuaalses maailmas oma käeobjekti viima samasse asukohta, kus on banaaniobjekt mängus. Seejärel peab mängija panema oma käe haardesse ja ootama kuni banaan tema käeobjekti külge haakub. Banaanist lahti saamiseks võib kasutaja käega lehvitada.

Kui mängija liigutab käsi kiiresti, siis ahv kardab teda ja põgeneb. Kui mängija on rahulik ja hoiab käes banaani siis ahv hakkab temale lähenema. Kui ahv jõuab banaanini ja saab selle kätte on mäng edukalt lõppenud ja mängija näeb teda õnnitlevat ekraani. Kui mängija puudutab ahvi põgenemise hetkel lõppeb mäng ebaõnnestumisega ja mängijale näidatakse vastavat teadet.

Mängu mängimise ajal on ka näha ahvi animatsioonid. Rahulolu momendil on ahv 4 jäsemega puu külge kinnitatud ja õõtsub rahulikult. Kui ahv peab liikuma hakkama vahetab ta puhkeasendi rippuvasse asendisse, ja hakkab oma sihtmärgi suunas liikuma. Liikumise ajal on nähtav ronimise animatsioon. Kui ahv jõuab sihtkohta läheb ta rippuvast asendist tagasi puhkeasendisse.

Liitreaalsuse mängu loomise protsess oli väga õpetlik, andes parema arusaamise nii Unity mängumootoriga mängude loomisest kui ka Blenderi tarkvara abil mudelite loomise ja animeerimise protsessist. Mängu loomine tõi ka nähtavale probleeme nagu näiteks Kinecti käte tuvastuses esinevad ebatäpsused. Mängu prototüüpi võiks lugeda õnnestunuks, kuid see vajab kindlasti edasiarendamist. (Joonis 19)

Joonis 19. Valmis mängu prototüüp.

5 Analüüs

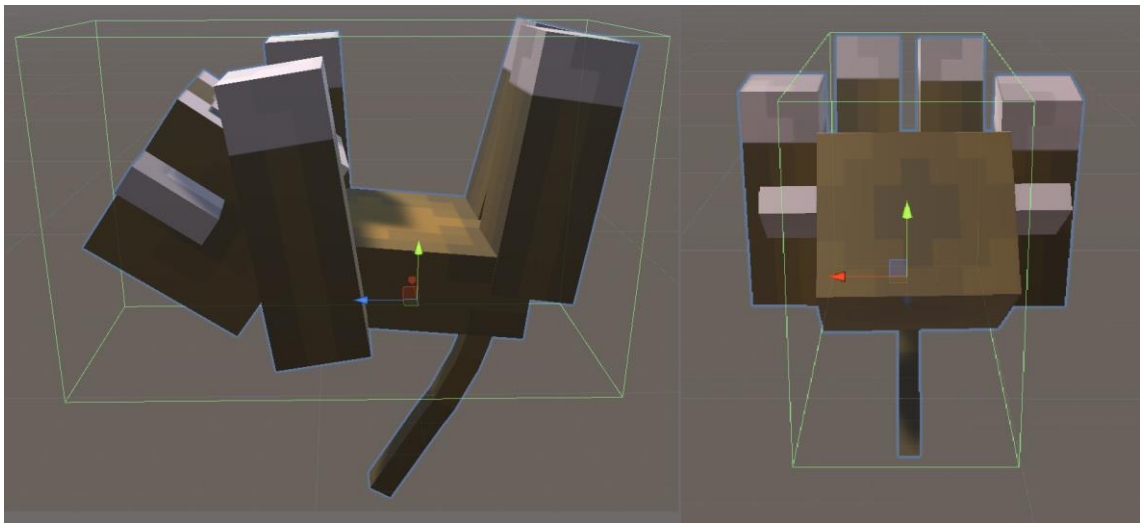
5.1 Probleemid

Projektis esineb mitmeid probleeme, mis ei takista mängu funktsioneerimist aga muudavad selle kasutaja kogemust halvemaks.

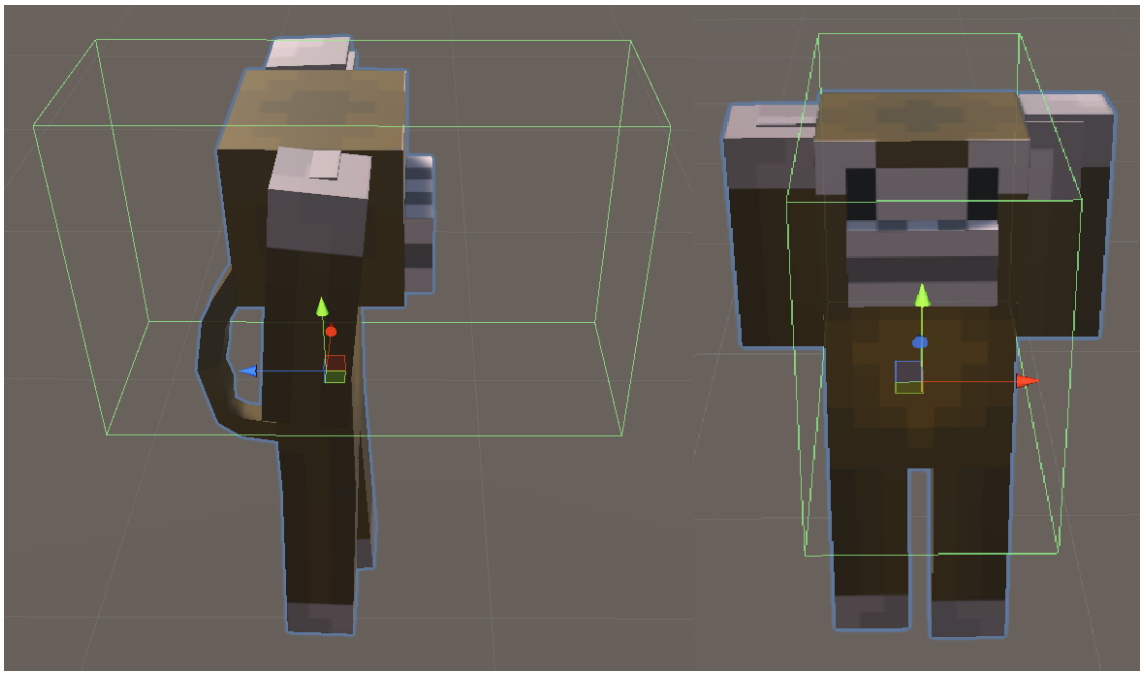
Esimeseks märgatavaks probleemiks on ahvi põgenemise loogika, mis on hüplik ja vigane. Hüplikus esineb ahvi järskudes ja suurtes asukoha vahetustes. Seda põhjustab põhiliselt puu ülesehitamise viis. Liikumise arvutamisel kasutatakse puu leheobjektide asukohti, mis aga asetsevad piisavalt suurte vahedega üksteisest, et iga asukoha vahetus näib kui järsk edasihüpe. Selle parandamiseks võiks olla näiteks leheobjektide vahel ühendavad vektorid, mida mööda saaks ahv sujuvamalt liikuda.

Põgenemise loogikal esineb ka viga kui esineb juhtum, kus ahv on jõudnud oksa lõppu ja pole sealt enam edasi liikumiseks võimalusi. Kui käed jõuavad sellisel hetkel piisavalt lähedale võib tekkida olukord kus leidub leheobjekt kätest teisel poolel, ahv valib selle objekti sihtkohaks ja liigub seejärel kätest läbi. Mäng tuvastab kokkupõrke ja mäng lõppeb. Selle situatsiooni parandamiseks oleks võimalik lisada erandjuhtum, kus mäng ei lõppeks sel hetkel või tuleks muuta ahvi liikumise loogikat märgatavalt.

Teine märgatav probleem on, et ahvi *hitbox* kuju ei vasta ahvi tegelikule mudelile, mille põhjustas Blender keskkonna vähene tundmine. Hetkel on ahvi ümber paigutatud tavaline risttahuka kujuline *hitbox*, mis ei jälgi ahvi tegelikku kehakuju ja on seega teiste mõõtmetega kui ahv visuaalselt näha on. Lisaks ei muutu see animatsiooniga kaasa, mistõttu rippuvas asendis ulatub ahvi alakeha *hitbox*'ist välja. (Joonised 20 ja 21) Selle parandamiseks tuleks ahvile uus *hitbox* modelleerida, mis järgiks ka ahvi animatsioone.



Joonis 20. *Hitbox* puhkeasendis olemise ajal.



Joonis 21. *Hitbox* rippuvas asendis olemise ajal.

Samuti häirib kasutajakogemust Kinecti kehatuvastusest tulenevad vead. Näiteks esineb Kinecti käte jälgimisel värelusi ja häireid, kus käe jälgimine jääb hetkeks seisma või Kinect tuvastab käena hetkeks mingi muu kehaosa või objekti ruumis. Selline ebakorrapärane käte liikumine põhjustab ka vigu mängu rahulikku liikumist tuvastavas *script*'is pannes näiteks looma vääralt mängijat kartma. Lisaks on Kinectil raskusi käe haardes oleku tuvastamisega, mis võtab keskmiselt 3 sekundit aega, halvimatel juhtudel üle 10 sekundi. Tänu sellele ei saa näiteks kasutaja banaani kätte haarata, mis tekitab mängijas segadust. Selle probleemi parandamiseks oleks näiteks võimalik uuendada Kinecti sensor versioon 2 peale.

5.2 Edasiarendamise võimalused

Antud mängu prototüüpi saaks edasi arendada näiteks parema liikumise loogika loomisega. Hetkelises lahenduses on nii põgenemine kui ka ahvi lähenemine hüplik, põgenemise puhul esineb ka vigu. Edasiarendustena võiks näiteks muuta kuidas ahv liigub. Leheobjektide vahel hüppamise asemel võiks leheobjektid olla ühendatud vektoritega, mida mööda saab ahv liikuda. Vektorid määraksid ka millised liikumisteed on üldse võimalikud. Samuti võiks ahvil lubada rohkem liikumis võimalusi, näiteks oksade peal.

Edasiarendusi saaks ka teha animatsioonides, kus võiks lisada rohkem erinevaid versioone juba olemasolevatest animatsioonidest, et eristada näiteks kui suurt vahemaad üritatakse ühe liigutusega läbida. Samuti saaks lisada uusi animatsioone nagu näiteks mängijalt puuvilja võtmine.

Edasiarendusena tuleks kindlasti parandada hetkeline viga ahvi *hitbox*'iga. Selleks tuleks luua ahviga sama kuju olev *hitbox*, mis järgiks ka ahvi liigutusi.

Edasiarendusena oleks ka võimalik lisada mängule juurde funktsionaalsust ja erinevaid mängu elemente. Näiteks veel esemeid, mida saab üles korjata. Samuti saaks mängu implementeerida näiteks punktikogumissüsteemi.

Mängule saaks lisada heliefekte ja loomale anda hääl. See aitaks mängu põnevamaks ja kaasahaaravamaks muuta. Looma häälitsuste järgi võib ka paremini aru saada, kas loom on hetkel kartlik, häiritud, sõbralik või uudishimulik.

6 Kokkuvõte

Töö eesmärgiks oli saada töötav mängu prototüüp koos vajaliku mängu funktsionaalsuse, mänguloogikaga ja visuaalidega.

Töö loomise eesmärgil sai uuritud Unity keskkonnas mängude loomise protsessi ning Blenderi keskkonnas mudelite modelleerimise ja animeerimise protsesse. Töö käigus oli mängu loomise eesmärgil loodud *script*'id, mis vastutavad puu mängu tekitamise, ahvi ja käte liikumise, mängu alustamise ja lõpetamise ning mängu funktsionaalsuse kontrollimise eest. Mängu visuaalse osa jaoks oli loodud lõpuekraanid, puu, ahvi ja banaani mudelid ning lisaks animatsioonid lõpuekraanidele ja ahvi mudelile.

Valminud mäng sisaldab kõike soovitud funktsionaalsust ja mänguloogikat, kuid parandamiseks ja edasiarendamiseks on veel ruumi. Valminud visualisatsioonid on kergesti arusaadavad, mis tagab mängu lihtsa mõistmise.

Mängu loomise protsess oli õpetlik andes parema arusaamise liitreaalsuse mängu loomisest, Unity mängumootorist, modelleerimisest ja animeerimisest.

Antud mängu prototüüpi võiks lugeda õnnestunuks, kuna tõestas, et antud vahendeid kasutades on võimalik luua sellist mängu, kuid edasiarenemiseks on ruumi.

Kasutatud kirjandus

[1] R. Gustasson, „Microsoft Kinecti rakendamine liigutuste tuvastamiseks interaktiivse virtuaalse puu jaoks,“ Tallinn: Tallinna Tehnikaülikool, 2019.

[2] „Creating and Using Scripts,“ Unity Technologies, 2019. [Online]. Available: https://docs.unity3d.com/Manual/CreatingAndUsingScripts.html. (Kasutatud 18.05.2019)

[3] „Tags,“ Unity Technologies, 2019. [Online]. Available: https://docs.unity3d.com/Manual/Tags.html. (Kasutatud 18.05.2019)

[4] „UV mapping,“ Wikipedia, 2018. [Online]. Available: https://en.wikipedia.org/wiki/UV_mapping. (Kasutatud 18.05.2019)

[5] „Hitbox,“ Wikipedia, 2019. [Online]. Available: https://en.wikipedia.org/wiki/Hitbox. (Kasutatud 18.05.2019)

[6] „Film frame,“ Wikipedia, 2019. [Online]. Available: https://en.wikipedia.org/wiki/Film_frame. (Kasutatud 18.05.2019)

[7] „Keyframes Introduction,“ Blender. [Online]. Available: https://docs.blender.org/manual/en/latest/animation/keyframes/introduction.html. (Kasutatud 18.05.2019)

[8] „Unity (game engine),“ Wikipedia, 2019. [Online]. Available: https://en.wikipedia.org/wiki/Unity_(game_engine). (Kasutatud 18.05.2019)

[9] „Unity,“ Unity Technologies, 2019. [Online]. Available: https://unity3d.com/unity. (Kasutatud 18.05.2019)

[10] „Blender (tarkvara),“ Vikipeedia, 2017. [Online]. Available: https://et.wikipedia.org/wiki/Blender_(tarkvara). (Kasutatud 19.05.2019)

[11] „Blender (software),“ Wikipedia, 2019. [Online]. Available: https://en.wikipedia.org/wiki/Blender_(software). (Kasutatud 19.05.2019)

[12] „About,“ Blender. [Online]. Available: https://www.blender.org/about/. (Kasutatud 19.05.2019)

[13] „Animation Introduction,“ Blender. [Online]. Available: https://docs.blender.org/manual/en/latest/animation/introduction.html. (Kasutatud 19.05.2019)

[14] „Monkeys,“ Planet Minecraft, 2012. [Online]. Available: https://www.planetminecraft.com/blog/monkeys/. (Kasutatud 18.05.2019)

Lisa 1 – AnimalController *script*

```
using System.Linq;
using UnityEngine;

public class AnimalController : MonoBehaviour
{
    private GameObject[] treeNodes;
    public GameObject EndScreenBad, EndScreenGood;
    private Animator MonkeyAnimator;
    private LeftHandController LeftHandScript;
    private RightHandController RightHandScript;
    private Transform RightHand, LeftHand, Fruit;
    private Vector3 AnimalLocation, NewLocation, AnimalPreviousLocation;
    private float DistanceToRH, DistanceToLH, speed, RunTime, TimeCounter;
    private int FriendlinessValue;
    private const float AnimalMovementDistance = 2.0f;
    private float minDistance = 1.5f;
    private float noMovementThreshold = 0.03f;
    Vector3[] previousLocationsRight = new Vector3[3];
    Vector3[] previousLocationsLeft = new Vector3[3];
    Vector3[] previousLocationsAnimal = new Vector3[5];

    void Start()
    {
        FriendlinessValue = -5;

        Invoke("AnimalInitialize", 0.5f); //Algatab looma 1 sekund pärast
mängu käima minemist

        //Seadistab käte ja looma liikumise koordinaatide listide algsed
väärtused 0 vektoriteks
        //List on kasutatud, et tuvastada järske kätte ja looma liigutusi
        for (int i = 0; i < previousLocationsRight.Length; i++)
        {
            previousLocationsRight[i] = Vector3.zero;
        }
        for (int i = 0; i < previousLocationsLeft.Length; i++)
        {
            previousLocationsLeft[i] = Vector3.zero;
        }
        for (int i = 0; i < previousLocationsAnimal.Length; i++)
        {
            previousLocationsAnimal[i] = Vector3.zero;
        }
```

```
        //Otsib mängust üles käte objektide scriptid, et hiljem sealt
funktsioone kutsuda
        RightHandScript =
GameObject.Find("RightHand").GetComponent<RightHandController>();
        LeftHandScript =
GameObject.Find("LeftHand").GetComponent<LeftHandController>();

        //Otsib üles ahvi animatsiooni kontrolleri, mille abil kontrollitakse
ahvi animatsioone
        MonkeyAnimator = gameObject.GetComponentInChildren<Animator>();
    }

    void Update()
    {
        //Loevad mängu jooksmise aega sekundites, runtime on kasutusel et
looma kontrollimine algaks 1.1 sekundit pärast mängu alustamist

        RunTime += Time.deltaTime;
        TimeCounter += Time.deltaTime;
        if (RunTime > 1.1f) //Ootab et Animal oleks initsialiseeritud
        {
            if(TimeCounter >= 1.0f && (RightHandScript.FruitTriggerRight ||
LeftHandScript.FruitTriggerLeft))//Iga 1 sekundi tagant suurendame
Friendliness väärtust ja nullime aja counteri
            {
                TimeCounter = 0.0f;
                FriendlinessValue++;
            }

            if (MovementDetection(previousLocationsAnimal, this.transform))
            {
                MonkeyAnimator.SetFloat("ClimbSpeed", 0.0f);
                MonkeyAnimator.SetBool("Climb", false);
            }


            //Käe liikumise kontrollija, reageerib järskudele ja suurtele
liigudustele
            if (MovementDetection(previousLocationsRight, RightHand))
            {
                if(FriendlinessValue - 2 > -5)//Kui tuvastati suur liikumine
vähendame Friendliness väärtust 2 võrra aga mitte -5 allapoole
                    FriendlinessValue -= 2;
            }
            if (MovementDetection(previousLocationsLeft, LeftHand))
            {
                if (FriendlinessValue - 2 > -5)
                    FriendlinessValue -= 2;
            }
```

```
            //Arvutame kauguse looma ja käte vahel
            DistanceToRH = Vector3.Distance(RightHand.position,
transform.position);
            DistanceToLH = Vector3.Distance(LeftHand.position,
transform.position);

            //Kiiruse arvutamine, käed lähedal põgeneb kiiremini
            speed = 0.05f/(DistanceToRH + DistanceToLH);

            //Kas mängija hoiab puuviljast kinni ja kas Friendliness väärtus
on positiivne
            if ((RightHandScript.FruitTriggerRight ||
LeftHandScript.FruitTriggerLeft) && FriendlinessValue > 0)
            {
                //Lähenemise script
                SortTreeNodes();//Sorteerime puu lehe pallid kauguse järgi
loomast kasvavasse listi
                Fruit = GameObject.Find("Banana(Clone)").transform;//Puuvilja
asukoht(ehk ka puuvilja hoidva käe asukoht)

                foreach (GameObject treeNode in treeNodes)//Puu lülide läbi
käimine kasvavas järjekorras
                {
                    //Kas puu lüli asub läheduses?
                    if (Vector3.Distance(treeNode.transform.position,
transform.position) < AnimalMovementDistance)
                    {
                        //Kas uus puu lüli on banaanile lähemal?
                        if (Vector3.Distance(transform.position,
Fruit.position) > Vector3.Distance(treeNode.transform.position + new
Vector3(0f, -1f, 0f), Fruit.position))
                        {
                            //Leitud puu lüli sobib ja me liigume uute
asukohta
                            transform.LookAt(treeNode.transform.position +
new Vector3(0f, -1f, 0f));
                            transform.rotation =
Quaternion.Inverse(transform.rotation);
                            transform.position =
Vector3.Lerp(transform.position, treeNode.transform.position + new
Vector3(0f, -1f, 0f), 0.001f);
                            MonkeyAnimator.SetBool("Climb", true);
                            MonkeyAnimator.SetFloat("ClimbSpeed", speed *
200.0f);
                        }
                    }
                }
            }
            else //Mängija ei hoia puuviljast kinni või on Friendliness
väärtus negatiivne
            {
                //Põgenemise script
                //Kas käed on liiga lähedale tulnud ja peaks liikuma?
```

```
                if (DistanceToLH < minDistance || DistanceToRH < minDistance)
                {
                    SortTreeNodes();//Sorteerime puu lehe pallid kauguse
järgi loomast kasvavasse listi

                    foreach (GameObject treeNode in treeNodes)//Puu lülide
läbi käimine kasvavas järjekorras
                    {
                        //Kas puu lüli asub läheduses?
                        if (Vector3.Distance(treeNode.transform.position,
transform.position) < AnimalMovementDistance)
                        {
                            //Kas uus puu lüli on kätest kaugemal?
                            if (Vector3.Distance(RightHand.position,
treeNode.transform.position) > minDistance &&
Vector3.Distance(LeftHand.position, treeNode.transform.position) >
minDistance)
                            {
                                //Leitud puu lüli sobib ja me liigume uute
asukohta
                                transform.LookAt(treeNode.transform.position
+ new Vector3(0f, -1f, 0f));
                                transform.rotation =
Quaternion.Inverse(transform.rotation);
                                transform.position =
Vector3.Lerp(transform.position, treeNode.transform.position + new
Vector3(0f, -1f, 0f), speed);
                                MonkeyAnimator.SetBool("Climb", true);
                                MonkeyAnimator.SetFloat("ClimbSpeed", speed *
200.0f);
                            }
                        }
                    }
                    //Ei leitud ühtegi sobivat puulüli
                }
            }
        }
    }

    //Looma algataja funktsioon
    void AnimalInitialize()
    {
        //Otsib kõik puu osad, millel on tag Leaf
        treeNodes = GameObject.FindGameObjectsWithTag("Leaf");

        //Kui puu osad leiti siis asetame looma kõige esimesele puu osale
        if (treeNodes != null)
        {
            transform.position = treeNodes[0].transform.position + new
Vector3(0f, -1f, 0f);
            transform.LookAt(treeNodes[1].transform.position + new
Vector3(0f, -1f, 0f));
        }
```

```
        //Kätte otsimine mängust ja määramine muutujale
        RightHand = GameObject.Find("RightHand").GetComponent<Transform>();
        LeftHand = GameObject.Find("LeftHand").GetComponent<Transform>();
    }

    //Sorteerib puu lülid kasvavasse järjekorda looma asukohast vaadates
    void SortTreeNodes()
    {
        treeNodes = treeNodes.OrderBy(treeNode =>
Vector3.Distance(transform.position, treeNode.transform.position)).ToArray();
    }

    //Tuvastab järsku käte liikumist
    bool MovementDetection(Vector3[] previousLocations, Transform Hand)
    {
        //Asukohtade liigutamine ja kõige uuema asukoha lõppu salvestamine
        for (int i = 0; i < previousLocations.Length - 1; i++)
        {
            previousLocations[i] = previousLocations[i + 1];
        }
        previousLocations[previousLocations.Length - 1] = Hand.position;

        //Kontrollib distantse viimaste asukohtade vahel, kui distantsid ei
ületa miinimumi siis võib arvata et ei ole järsku liikumist
        for (int i = 0; i < previousLocations.Length - 1; i++)
        {
            if (Vector3.Distance(previousLocations[i], previousLocations[i +
1]) >= noMovementThreshold)
            {
                return true;
            }
        }
        return false;
    }

    //Seab mängu tagasi algsetele väärtustele
    public void RestartGame()
    {
        FriendlinessValue = -5;
        transform.position = treeNodes[0].transform.position + new
Vector3(0f, -1f, 0f);
        transform.LookAt(treeNodes[1].transform.position + new Vector3(0f, -
1f, 0f));
    }

    //Mängu lõppu kontroll
    public void FinishGame()
    {
        //Mängija hoiab puuviljast kinni ja Friendliness on positiivne -
positiivne lõpp
```

```
        if((RightHandScript.FruitTriggerRight ||
LeftHandScript.FruitTriggerLeft) && FriendlinessValue > 0)
        {
            EndScreenGood.SetActive(true);//Mängu positiivse lõppu
väljakutsumine
        }
        //Mängija hoiab puuviljast kinni aga Friendliness on negatiivne -
negatiivne lõpp
        else if ((RightHandScript.FruitTriggerRight ||
LeftHandScript.FruitTriggerLeft) && FriendlinessValue < 0)
        {
            EndScreenBad.SetActive(true);//Mängu negatiivse lõppu
väljakutsumine
        }
        //Ei hoia puuviljast kinni ja Friendliness negatiivne - negatiivne
lõpp
        else
        {
            EndScreenBad.SetActive(true);//Mängu negatiivse lõppu
väljakutsumine
        }
    }
}
```

Lisa 2 – LeftHandController *script*

```
using UnityEngine;

public class LeftHandController : MonoBehaviour
{
    private Transform LeftHand;
    private GameObject fruit;
    public GameObject AnimalController;
    private Vector3 HandLocation;
    private KinectOverlayer KinectScript;
    float speed;
    public float smoothFactor = 5;
    private bool leftHand = false;
    private bool FruitTrigger = false;
    private bool HasBeenGrabbed = false;

    void Start()
    {
        //Määrab hetkelise objekti transformi LeftHand muutujasse
        LeftHand = gameObject.transform;
        //Otsib main camera scripti KinectOverlayer, et seal funktsioone
kasutada
        KinectScript = GameObject.Find("Main
Camera").GetComponent<KinectOverlayer>();

    }

    void Update()
    {
        //Käte liikumise kiirus, muudab ainult visuaalset poolt
        speed = 5f * Time.deltaTime;
        //Küsib Kinecti scriptilt käe asukohta
        HandLocation = KinectScript.LeftHand;
        leftHand = KinectScript.handIndex;
        transform.position = HandLocation;

        Vector3 vPosOverlay2 = Camera.main.ViewportToWorldPoint(new
Vector3(HandLocation.x, HandLocation.y, HandLocation.z));
        transform.position = Vector3.Lerp(transform.position, vPosOverlay2,
smoothFactor);

        //Kas mängija hoiab puuviljast kinni ja kas käsi on puuvilja sees
        if (KinectScript.isgrab && FruitTrigger)
        {
            if (!HasBeenGrabbed)//Kui puuvilja pole veel enne kätte võetud
siis muudame väärtuse true, kasutusel puuviljale gravitatsiooni määramisel
                HasBeenGrabbed = true;
            fruit = GameObject.Find("Banana(Clone)");//Banaani otsimine
mängust
```

```
            fruit.transform.position = LeftHand.position;//Banaani asukoha
seadmine võrdseks vasaku käe asukohaga
        }
        if(fruit == null)
            HasBeenGrabbed = false;
    }

    public void RestartGame()
    {
        HasBeenGrabbed = false;
        FruitTrigger = false;
    }

    //Tuvastab kas käsi on puuvilja sisse liikunud
    private void OnTriggerEnter(Collider collider)
    {
        if (collider.tag == "Fruit")//Kas sisenesime puuvilja objekti?
        {
            FruitTrigger = true;
            fruit = GameObject.Find("Banana(Clone)");//Puuvilja otsimine ja
muutujale määramine
        }
        else if (collider.tag == "Animal")
        {
            AnimalController.GetComponent<AnimalController>().FinishGame();
        }
    }

    //Tuvastab kas käsi on puuviljast välja liikunud
    private void OnTriggerExit(Collider collider)
    {
        if (collider.tag == "Fruit")//Kas tegemist on puuvilja objektiga?
        {
            FruitTrigger = false;
            if (HasBeenGrabbed)//Kui on üritatud juba krabada siis anname
puuviljale nüüd gravitatsiooni
            {
                fruit.GetComponent<Rigidbody>().useGravity = true;
                fruit.GetComponent<Rigidbody>().isKinematic = false;
            }
        }
    }

    //Annab muutuja väärtuse AnimalController scriptile
    public bool FruitTriggerLeft
    {
        get { return FruitTrigger && HasBeenGrabbed && KinectScript.isgrab; }
    }
}
```

Lisa 3 – RightHandController *script*

```
//Scripti sisemus sarnaneb LeftHandControlleriga seega pole kommenteeritud
using UnityEngine;

public class RightHandController : MonoBehaviour
{
    private Transform RightHand;
    private GameObject fruit;
    public GameObject AnimalController;
    private Vector3 HandLocation;
    private KinectOverlayer KinectScript;
    float speed;
    public float smoothFactor = 5;
    private bool rightHand = false;
    private bool FruitTrigger = false;
    private bool HasBeenGrabbed = false;

    void Start()
    {
        RightHand = gameObject.transform;

        KinectScript = GameObject.Find("Main
Camera").GetComponent<KinectOverlayer>();
    }

    void Update()
    {
        speed = 5f * Time.deltaTime;

        HandLocation = KinectScript.RightHand;
        rightHand = KinectScript.handIndex2;
        transform.position = HandLocation;//Käe asukoht == Kinecti käe
asukoht

        Vector3 vPosOverlay2 = Camera.main.ViewportToWorldPoint(new
Vector3(HandLocation.x, HandLocation.y, HandLocation.z));
        transform.position = Vector3.Lerp(transform.position, vPosOverlay2,
smoothFactor);

        if (KinectScript.isgrab && FruitTrigger)
        {
            if (!HasBeenGrabbed)
                HasBeenGrabbed = true;
            fruit = GameObject.Find("Banana(Clone)");
            fruit.transform.position = RightHand.position;
        }
        if (fruit == null)
            HasBeenGrabbed = false;
    }
```

```
    public void RestartGame()
    {
        HasBeenGrabbed = false;
        FruitTrigger = false;
    }

    private void OnTriggerEnter(Collider collider)
    {
        if (collider.tag == "Fruit")
        {
            FruitTrigger = true;
            fruit = GameObject.Find("Banana(Clone)");
        }
        else if (collider.tag == "Animal")
        {
            AnimalController.GetComponent<AnimalController>().FinishGame();
        }
    }

    private void OnTriggerExit(Collider collider)
    {
        if (collider.tag == "Fruit")
        {
            FruitTrigger = false;
            if (HasBeenGrabbed)
            {
                fruit.GetComponent<Rigidbody>().useGravity = true;
                fruit.GetComponent<Rigidbody>().isKinematic = false;
            }
        }
    }

    public bool FruitTriggerRight
    {
        get { return FruitTrigger && HasBeenGrabbed && KinectScript.isgrab; }
    }
}
```

Lisa 4 – GameController *script*

```
using UnityEngine;

public class gameController : MonoBehaviour
{
    public GameObject FruitPrefab, Animal, RHand, LHand;
    private GameObject Banana;

    void Start()
    {
        Invoke("SpawnFruit", 3.0f);//Algatab puuvilja maailma tekitamise 3
sekundit pärast mängu algust
    }

    void Update()
    {
        if (Banana != null)
        {
            if (Banana.transform.position.y < -20.0f)
            {
                Destroy(Banana);
                Invoke("SpawnFruit", 0.1f);
            }
        }
    }

    //Kutsub välja taaskäivitamise funktsioonid ja tekitab uue banaani
    public void FinishGame()
    {
        Animal.GetComponent<AnimalController>().RestartGame();
        RHand.GetComponent<RightHandController>().RestartGame();
        LHand.GetComponent<LeftHandController>().RestartGame();
        Destroy(Banana);
        Invoke("SpawnFruit", 0.1f);
    }

    //Puuviljale asukohta otsiv ja maailma asetav funktsioon
    void SpawnFruit()
    {
        GameObject[] leafNodes, trunkNodes;

        int leafIndex, loopCheck = 0;
        bool leafNodeFound = false;

        leafNodes = GameObject.FindGameObjectsWithTag("Leaf");
        trunkNodes = GameObject.FindGameObjectsWithTag("Trunk");

        //Leia leaf node, mis ei ole trunk node läheduses
        do
```

```
        {
            //Leaf node valimine
            leafIndex = Random.Range(0, leafNodes.Length);
            //Kõigi trunk node kontrollimine, et nad ei asuks liiga leaf node
läheduses
            leafNodeFound = true;
            foreach (GameObject trunkNode in trunkNodes)
            {
                if(Vector3.Distance(trunkNode.transform.position,
leafNodes[leafIndex].transform.position) < 1.0f)
                {
                    leafNodeFound = false;
                }
            }
            loopCheck++;
        } while (!leafNodeFound && loopCheck < 1000);//Jookseb seni kuni leaf
node on leitud või on jõutud 1000 tsüklini

        if (leafNodeFound)//Kui puu lüli leiti tekitatakse maailma puuvili
vastavale kohale
        {
            Banana = Instantiate(FruitPrefab,
leafNodes[leafIndex].transform.position + new Vector3(0f, -0.7f, 0f),
Quaternion.Euler(new Vector3(90, 0, 90)));
        } else
        {
            Debug.Log("Couldn't find a leaf node for the fruit " +
loopCheck);
        }
    }
}
```

Lisa 5 – TreeSpawner *script*

```
using UnityEngine;
using System.IO;

public class treeSpawner : MonoBehaviour
{

    private GameObject[] treeNodes;
    public GameObject LeafPrefab;
    public GameObject TrunkPrefab;

    void Start()
    {
        TreePlanter();//Puu maailma tekitamise funktsioon
        //TreeReader();//Puu maailmast lugemise funktsioon
    }

    //Puu maailma tekitamise funktsioon
    void TreePlanter()
    {
        string type;//Puulüli tüüpi hoidev muutuja
        string path = "Assets/treeData.txt";//Faili asukoht ja nimi

        StreamReader reader = new StreamReader(path);//Algatab failist
lugemise

        type = reader.ReadLine();//Loeb failist esimese rea ja salvestab
puulüli tüübi
        while (type != null)//Jookseb kuni fail on tühi ja ei leitud uut
püülüli tüüpi
        {
            //Puu lüli tüübi kontroll ja vastava mudeli maailma loomine
            if(type == "Leaf")
            {
                Instantiate(LeafPrefab, StringToVector3(reader.ReadLine()),
Quaternion.identity);
            } else if(type == "Trunk")
            {
                Instantiate(TrunkPrefab, StringToVector3(reader.ReadLine()),
Quaternion.identity);
            } else
            {
                Debug.Log("Something in file is broken!");
            }
            type = reader.ReadLine();
        }
        reader.Close();
    }
```

```
    //Puu maailmast lugemise funktsioon
    void TreeReader()
    {
        string path = "Assets/treeData.txt";//Faili asukoht ja nimi

        StreamWriter writer = new StreamWriter(path);//Algatab kirjutamise

        treeNodes = GameObject.FindGameObjectsWithTag("Trunk");//Otsib üles
kõik puulülid mille tag on Trunk

        foreach (GameObject treeNode in treeNodes)//Kirjutab kõik need
puulülid faili
        {
            writer.WriteLine("Trunk");
            writer.WriteLine(treeNode.transform.position);
        }

        treeNodes = null;//Nullib puulülide listi
        treeNodes = GameObject.FindGameObjectsWithTag("Leaf");//Otsib üles
kõik puulülid mille tag on Leaf

        foreach (GameObject treeNode in treeNodes)//Kirjutab kõik need
puulülid faili
        {
            writer.WriteLine("Leaf");
            writer.WriteLine(treeNode.transform.position);
        }

        writer.Close();
    }

    //Muudab failist saadud stringi vektoriks
    private static Vector3 StringToVector3(string sVector)
    {
        //Sulgude eemaldamine
        if (sVector.StartsWith("(") && sVector.EndsWith(")"))
        {
            sVector = sVector.Substring(1, sVector.Length - 2);
        }

        //Koma juurest eraldamine
        string[] sArray = sVector.Split(',');

        //Vector3 salvestamine
        Vector3 result = new Vector3(
            float.Parse(sArray[0]),
            float.Parse(sArray[1]),
            float.Parse(sArray[2]));
        return result;
    }
}
```

Lisa 6 – EndScreen *script*

```
using UnityEngine;

public class EndScreen : MonoBehaviour
{
    public GameObject GameManager;

    //Mängu lõpetamine
    public void EndGame()
    {
        gameObject.SetActive(false);//Eemaldab mängu lõppu ekraani
        GameManager.GetComponent<gameController>().FinishGame();//Kutsub
välja mängu lõppu funktsiooni
    }
}
```