TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Karl Lõoväli 142679 IAPB

DÜNAAMILISE KAAMERASÜSTEEMI JA LIHASÖÖJATAIME TEHISINTELLEKTI LOOMINE UNREAL ENGINE 4 MÄNGUMOOTORIL

Bakalaureusetöö

Juhendaja: Martin Rebane
MSc

Tallinn 2021

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: Karl Lõoväli

01.01.2021

Annotatsioon

Käesoleva bakalaureusetöö eesmärgiks on luua dünaamiline kaamerasüsteem side-scroller mitmikmängudele ja lihasööjataime tehisintellekt.

Töös võrreldakse erinevaid side-scroller mängude kaamerasüsteeme ja antakse ülevaade, kuidas luua tehisintellekti Unreal Engine 4 mängumootori abil.

Bakalaureusetöö praktilise osana valmis dünaamiline kaamerasüsteem mitmik side-scroller mängudele, mille abil on mängijatel võimalik mängu maailmas koos olla jagades ühte suurt ekraani või olla erinevates kohtades jagades ekraani dünaamiliselt pooleks vastavalt sellele, kus mängu tegelased üksteise suhtes asuvad.

Lisaks sellele valmis praktilise osana 3D lihasööjataime mudel, neli animatsiooni mudelile ning lihasööjataime tehisintellekt.

Lihasööjataime 3D mudel on loodud modelleerimise tarkvaraga Blender. Mõlemate mängu komponentide loogika on loodud Unreal Engine 4 mängumootoriga.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 41 leheküljel, 3 peatükki, 15 joonist.

Abstract

Development of Dynamical Camera System and Meat-Eating Plant Artificial Intelligence in Unreal Engine 4 Game Engine

The goal of this thesis is to create a dynamic camera system for multiplayer side-scroller games and a meat-eating plant artificial intelligence.

Different side-scroller camera systems are compared in this thesis and an overview of how to create artificial intelligence using the Unreal Engine 4 game engine is given.

As a practical part, a dynamic camera system for multiplayer side-scroller games was created. It makes it possible for game players to share one big screen when their game characters are close to each other or to have their characters be in different locations, resulting in the screen to be split dynamically depending on where the game characters are positioned to each other.

In addition to that, as a practical part a 3D meat-eating plant model, four model animations, and meat-eating plant artificial intelligence was created.

The meat-eating plant 3D model was created with Blender software. Both game components logic is created with Unreal Engine 4 game engine.

The thesis is in Estonian and contains 41 pages, 3 chapters, 15 figures.

Lühendite ja mõistete sõnastik

Blueprint	Visuaalne skriptimise keel Unreal Engine 4 mängumootoris. [14]
Game Mode	Vastutab mängu käitumise määramise ja reeglite jõustamise eest. [18]
Materjal	Abivahend, mis pannakse peale objektidele, et muuta objektide visuaalset väljanägemist. [1]
Node	Andmestruktuuri üksus.
Oleku masin	Objekt Unreal Engine 4, mis pakub graafilist viisi, kuidas jaotada animatsioonid olekuteks. [6]
Polygon	Kahemõõtmeline suletud kujund, mis koosneb sirgetest joontest.
Render target	*Render target* on tekstuur millele saab anda uut väärtust mängu töötamise ajal. [20]
Side-scroller	*Side-scroller* on videomängu tüüp, kus mängu tegelaste vaatamiseks kasutatakse külgvaate kaameranurka. [4]
UE4	Unreal Engine 4.
Widget	Unreal Engine 4 element, mida kasutatakse enamasti kasutajaliideste jaoks menüüde ja info näitamiseks mängu ajal. [12]

Sisukord

Jooniste loetelu

1 Sissejuhatus

Bakalaureusetöö eesmärgiks on luua dünaamiline kaamerasüsteem *side-scroller* mitmikmängudele ja lihasööjataime tehisintellekt.

Side-scroller mäng on mäng, kus mängu tegevust jälgitakse külje pealt. Mängu tegelased liiguvad kas vasakult paremale või ülevalt alla. Loodav kaamerasüsteem on unikaalne selle poolest, et ta lubab *side-scroller* mängus mängijatel jagada ühte ekraani ning loob ka vajadusel dünaamiliselt ekraani jagamise. Ekraani jagamine toimub Voronoi diagrammi alusel eukleidiliste kaugusmõõtude järgi. Töö põhiline keerukus seisneb kaamerasüsteemi loomises ning kaamera asetuses, et tegelane oleks nähtav ekraani jagamise ajal.

Töö esimeses osas räägitakse erinevatest *side-scroller* mängudes kasutuses olevatest kaamerasüsteemidest, analüüsitakse nende negatiivseid ja positiivseid külgi. Töö raames luuakse autori poolt Unreal Engine 4 mängumootori abil dünaamiline *side-scroller* mitmikmängu kaamerasüsteem.

Töö käigus luuakse ka lihasööjataime mängu tegelane, kellel on oma mõtlemine. Lihasööjataime tehisintellekti eesmärgiks on patrullida kahe punkti vahelist punkti ning rünnata objekte, mis talle lähenevad. Ründamise tulemusena mängu objekt kas hävitatakse või sülitatakse välja. Lihasööjataime tehisintellekti peamiseks keerukuseks on tema aju loomine kasutades Unreal Engine 4 mängumootorit. Lihasööjataime mudelile loodi skelett animatsioonide tegemise jaoks ning neli animatsiooni.

Töö on koostatud aastal 2020 Tallinna Tehnikaülikooli informaatika bakalaureuseõppe eriala lõputööna.

1.1 Unreal Engine 4

Unreal Engine 4 (UE4) on mängumootor, mis on bakalaureuse töö üks põhivahenditest. Töö käigus tehtud dünaamiline kaamerasüsteem ning lihasööjataime tehisintellekt on loodud Unreal Engine 4 mängumootori abil.

"Mängumootor on mängu arendajate platvorm mängu jooksutamiseks, maailma laadimiseks, sind mängu maailmasse asetamiseks ja sinu viibimise majutamiseks. On erinevaid mängumootoreid. Vastavalt mängu vajadustele, igal mängumootoril on erinev võimekus, kuidas nad renderdavad mängu maailma, arvutavad füüsikat, mängivad helisid ja palju muud", defineeris CD Projekt Red tiim mängumootori olemuse.[13]

UE4 mängumootor kasutab kahte peamist programmeerimise keelt, C++ ja Blueprint. Blueprint, lubab teha peaaegu kõike, mida C++ UE4 mängumootoris. [14]

Mõned populaarsed mängud tehtud UE4 mängumootori abil: Fortnite, Borderlands 3, Star Wars: Jedi Fallen Order, Sea of Thieves ja palju teisi. [14]

Antud lõputöö raames kasutati arenduseks peamiselt C++ programmeerimise keelt, kuid on kasutatud ka Blueprinte.

2 Side-scroller mitmikmängu dünaamiline kaamerasüsteem

Dünaamiline kaamerasüsteem on loodud *side-scroller* mitmikmängudele. Kaamerasüsteemi loomiseks kasutati mitmeid erinevaid UE4 komponente, milles implementeeriti trigonomeetrilisi ja vektorarvutustel põhinevaid algoritme.

2.1 Mis on mitmikmäng

Mitmikmäng on videomäng, mis võimaldab rohkem kui ühel mängijal korraga mängimist. Erinevad mängijad on samas serveris, levelis või mängu ruumis koos. [22]

Dünaamilise kaamerasüsteem on mõeldud lokaalsele *side-scroller* mitmikmängule.

2.2 Side-scroller mäng

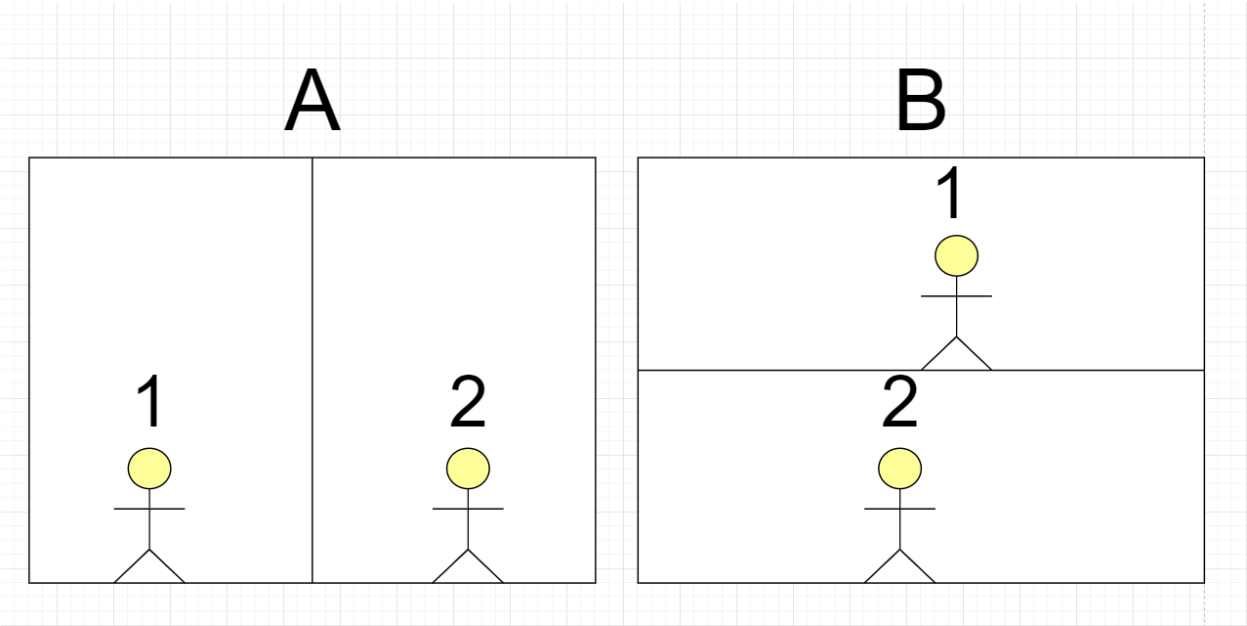
Side-scroller on videomängu tüüp, kus mängu tegelaste vaatamiseks kasutatakse külgvaate kaameranurka. *Side-scroller* mängud on enamasti 2D mängud, kus mängu tegelased liiguvad ühes suunas, üldiselt vasakult paremale. Keerulisemad *side-scroller* mängud lubavad lisaks horisontaalsele liikumisele ka vertikaalset liikumist. [4]

2.3 Erinevad kaamerasüsteemid side-scroller mitmikmängudes

Järgnevalt tutvustatakse levinud kaamerasüsteeme *side-scroller* mängudes: pooleks jagatud ekraaniga kaamerasüsteem (vt peatükk 2.3.1) ja ühise ekraaniga kaamerasüsteem (vt peatükk 2.3.2). Peale seda antakse ülevaade töö raames loodud dünaamilisest kaamerasüsteemist (vt peatükk 2.3.3).

2.3.1 Pooleks jagatud ekraaniga kaamerasüsteem

Pooleks jaotatud ekraaniga kaamerasüsteemid on populaarsed lokaalsetele mitmikmängijate mängudele, kus iga mängija saab omale osa kogu ekraanist. Iga mängija kontrollib oma tegelast enda ekraani poolel. [16] Pooleks jagatud ekraani sobib kasutada mängudes, kus mängu tegelased saavad olla üksteisest lahus. [17]



Joonis 1. Vertikaalne ja horisontaalne ekraani jagamise joonised.

Joonisel (Joonis 1) A graafik kujutab endast vertikaalset ekraani jagamist. Kui ekraan on vertikaalselt keskelt pooleks jagatud, asub mängija üks vasakul pool ekraani ja mängija kaks paremal pool ekraani.

Joonisel (Joonis 1) B graafik kujundab endast horisontaalset ekraani jagamist. Kui ekraan on horisontaalselt keskelt pooleks jagatud, asub mängija üks ülemisel ekraani poolel ja mängija kaks alumisel ekraani poolel.

Ekraani pooleks jagamise positiivne osa on selle äärmiselt lihtne implementeerimine. Unreal Engine 4 mängumootorisse on vertikaalne ja horisontaalne ekraani jagamine sisse ehitatud. Seda on võimalik sisse lülitada UE4 sätetest.

Ekraani pooleks jagamise negatiivseteks pooleks on, et mõlemad mängijad näevad terve mängu kestmisel väiksemat osa ekraanist.

Sellise kaamerasüsteemi jaoks on vaja kahte kaamerat, mõlema mängija kohta üks.

2.3.2 Ühise ekraaniga kaamerasüsteem

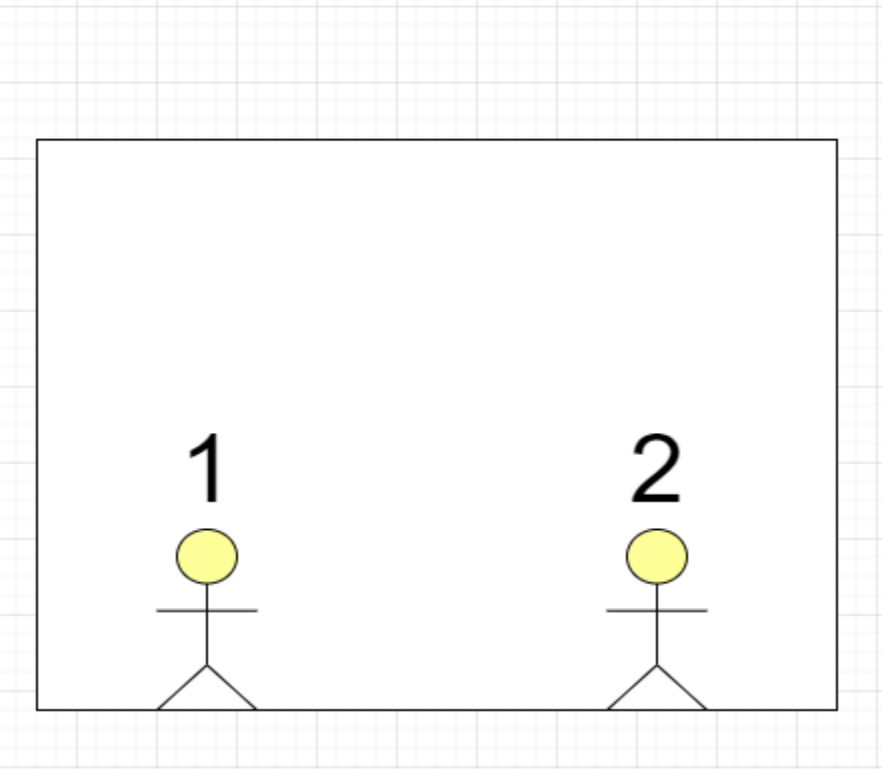
Ühise ekraaniga kaamerasüsteemi puhul jagavad mitu mängijat sama ekraani, kaamera pannakse kahe tegelase vahele. Mäng nimega Gauntlet[1] oli üks esimesi, kes sellist tehnikat rakendas, kui mängijad soovisid joosta vastassuunas ekraanist välja, blokeeris mäng mängu tegelaste liikumist. [21]

Ühise ekraani pluss on see, et mängijad näevad suuremat osa mängu maailmast, läbi selle on mängijatel üksteise tegevusi kergem jälgida.

Ühise ekraani kaamerasüsteemi negatiivseks osaks on, et mängijad peavad arvestama üksteise tasemega ning liikumise kiirusega. Alternatiivne lahendus Gauntlet lähenemisele on lubada ühel mängu tegelasel ekraani raamest välja liikuda ning seepeale hävitada ekraanist väljas olev tegelane ja ta taas elustada ekraani raames.

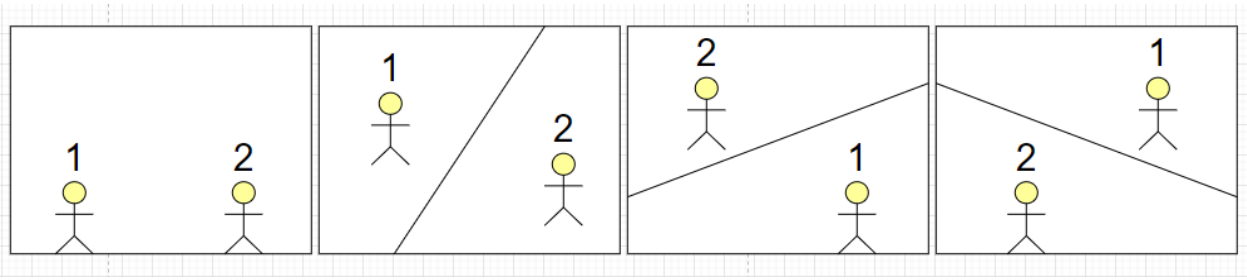
Ühise ekraaniga kaamerasüsteemi jaoks piisab ühest kaamerast.

[1] https://en.wikipedia.org/wiki/Gauntlet_(1985_video_game)



Joonis 2. Ühine ekraan.

2.3.3 Dünaamilise ekraani kaamerasüsteem



Joonis 3. Dünaamilise ekraani jagamise näited

Joonisel (Joonis 3) on näha, et dünaamiline ekraan on jagatud ekraani ning ühise ekraani kombinatsioon. Antud lahendus sobib mängule, millel on kaks mängu tegelast ja igal mängu tegelasel on oma personaalne kaamera. Mängu tegelaste kaamerad positsioneerivad ennast kahe mängu tegelase vahele, kui tegelased on üksteisele piisavalt lähedal, et mahtuda samale ekraanile. Kui tegelased liiguvad üksteisest kaugemale ja enam ei mahu samale ekraanile ära, siis hakkavad kaamerad kindlat mängu tegelast jälgima ning lisaks sellele toimub ekraani jagamine sellise nurga alt, mille tulemusena on aru saada millises suunas tegelased üksteisest asuvad, et nad saaksid alati

leida tee tagasi üksteise juurde. Selline lahendus lubab mängijatel nautida suuremat ekraanipilti, kui mängijad on koos ning lahendada ka erinevaid probleeme, olles erinevates kohtades mängu maailmas.

2.4 Dünaamilise ekraaniga kaamerasüsteemi komponendid

Dünaamilise ekraaniga kaamerasüsteem koosneb erinevatest komponentidest. Komponentide vahelist suhtlust kirjeldab diagramm Lisas 2.

2.4.1 Game Mode

Game Mode vastutab mängu käitumise määramise ja reeglite jõustamise eest. *Game Mode* hoiab endas informatsiooni mängu tegelaste kohta, kontrollib mis nendega juhtub, kui nad surevad, hoiab skoore ja palju muud. *Game Mode* on liim paljude erinevate süsteemide vahel mängus. [18]

Dünaamilise kaamerasüsteemi jaoks kasutatakse *Game Mode* kaamerate loomiseks ning nendele õigete sisendite andmiseks loomise käigus. Kaamerale antakse viide kahest mängu tegelasest, mängu tegelasest, keda ta peab jälgima, kui toimub ekraani jagamine ja tema *render target* (vt peatükk 2.4.2).

Kaamera loomine on programmeeritud C++ keeles.

2.4.2 Render target

Render target on tekstuur millele saab anda uut väärtust mängu töötamise ajal. Sobib kasutamiseks olukorras, kus on vaja kasutada lisa kaamerat. Näiteks, kui mängu ruumis on turvakaamera, *render target*'t kasutades saab simuleerida turvakaamera pilti mängijale. [20]

*Render targe*t objekti salvestub kaamera renderdatud pilt. Dünaamilise kaamerasüsteemi jaoks on kasutusel kaks *render target* objekti, iga kaamera kohta üks. *Render target'*d on vajalikud materjali (vt peatükk 2.4.6) jaoks, et oleks võimalik luua dünaamiline ekraani jagamine.

2.4.3 Kontrollitav mängu tegelane

Kontrollitav mängu tegelane on mängu tegelane, keda mängija kontrollib ja mida kaamera jälgib.

Dünaamilise kaamerasüsteemi jaoks on vaja kahte mängu tegelast. *Widget*'l (vt peatükk 2.4.4) ja Kaamera komponendil on vaja viiteid mõlemale mängu tegelasele.

Mängu tegelane liikumise loogika on tehtud C++ keeles.

2.4.4 Widget

Widget on UE4 element, mida kasutatakse enamasti kasutajaliideste jaoks. *Widget*'tega tehakse mängu menüüd ning elemendid, mida näidatakse mängijale mängu ajal, näiteks mängija elude arv või abistav jutt. [12]

Dünaamilise kaamera jaoks on loodud *widget*, mida kasutatakse mängijatele mängu pildi kuvamiseks. *Widget*'l on alamkomponendiks pilt, millele on peale pandud kaamera materjali eksemplari (vt peatükk 2.4.5). Lisaks sellele arvutab *widget* materjali jaoks mängu tegelaste kohad ekraani suhtes, et materjal saaks ekraani täpselt jagada.

Kui tegelaste vahemaa on väiksem kui ekraani jagamiseks vajalik, ei arvutata tegelaste kohtasid, vaid antakse materjali konstant väärtused, mille puhul ekraani jagamist ei toimu. Vastasel juhul tuleb arvutada materjali jaoks tegelaste asukohad ekraani suhtes.

Ekraan pikslid asuvad 0 ja 1 väärtuste vahel. Et saada mängu tegelaste asukohtasid ekraani suhtes peame võtma mängu tegelaste vahelise vektori mängu maailmas. Vektori koordinaadid on mängija üks positsioon ekraani suhtes. Vektori negatiivne tulemuse on mängija kaks positsioon ekraani suhtes.

Widget loogika on loodud visuaalse skriptimise keele Blueprint abil.

2.4.5 Materjali eksemplar

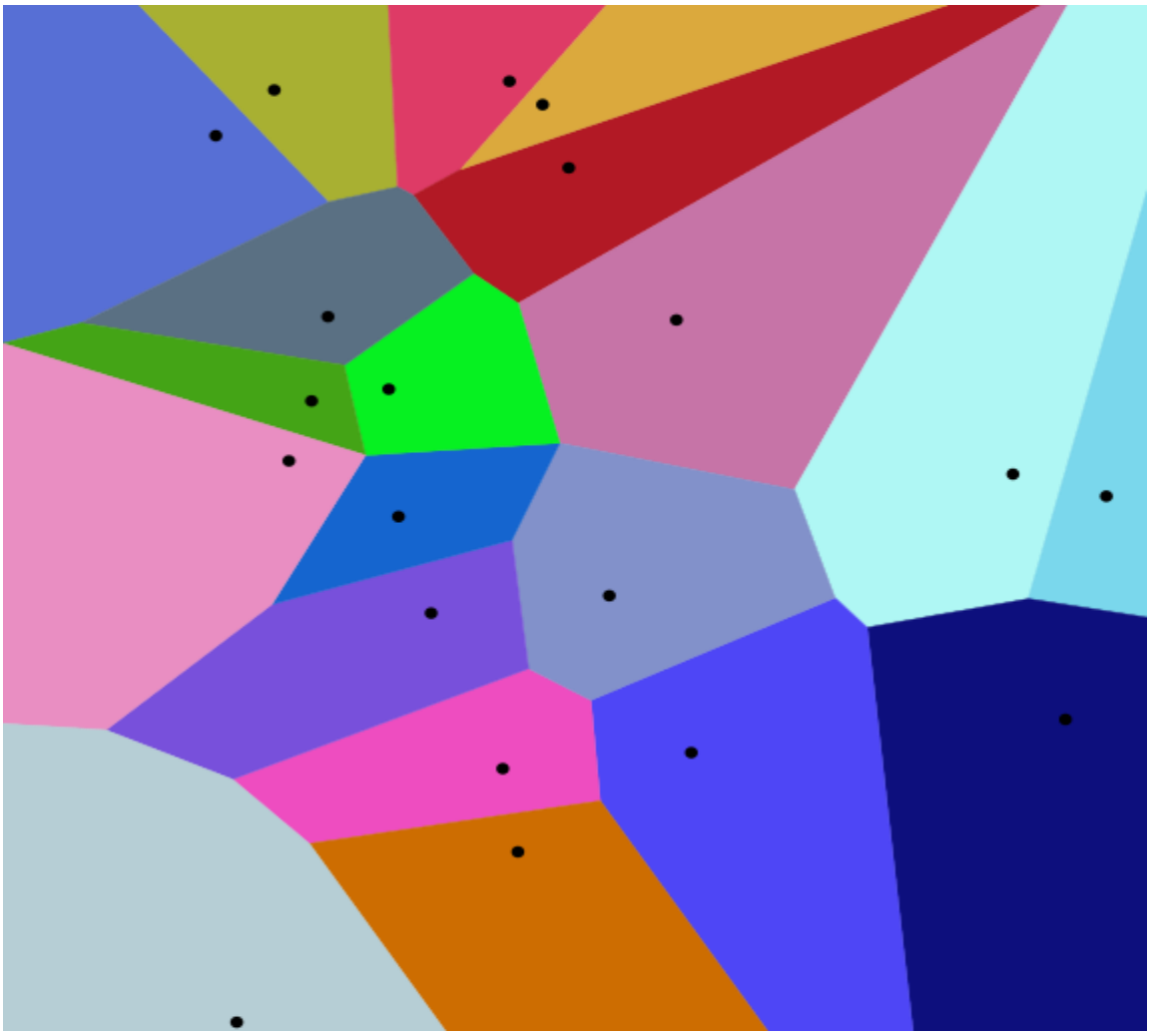
UE4 mängumootoris materjali eksemplar on objekt, mida kasutatakse, et muuta materjali välimust ilma, et teda peaks uuesti kompileerima. Tüüpilist materjali ei ole võimalik muutma ilma uuesti kompileerimata, materjali eksemplari on võimalik muuta ilma uuesti kompileerimata. See lubab muuta materjali väärtust mängu kestel. [19]

Antud projektis kasutatakse materjali eksemplari objekti materjali arvutuse tulemuse hoidmiseks. Tulemus pannakse *widget*'l olevale pildile, et näidata mänguekraani mängijatele.
Lisaks sellele, kasutatakse materjali eksemplari objekti, et anda materjalile edasi kahe mängu tegelase positsioonid. Neid väärtuseid kujutatakse joonisel (Joonis 5) x1, y1 ja x2, y2 väärtustena.

2.4.6 Materjal

Materjal on abivahend, mis pannakse peale objektidele, et muuta objektide visuaalset väljanägemist. Kõrgemal tasemel võib mõelda materjalist kui värvist, mis lisatakse objektile, materjal määrab objekti värvi, heleduse, kui läbipaistev objekt on ja võimeliseks veel paljuks muuks. [1]

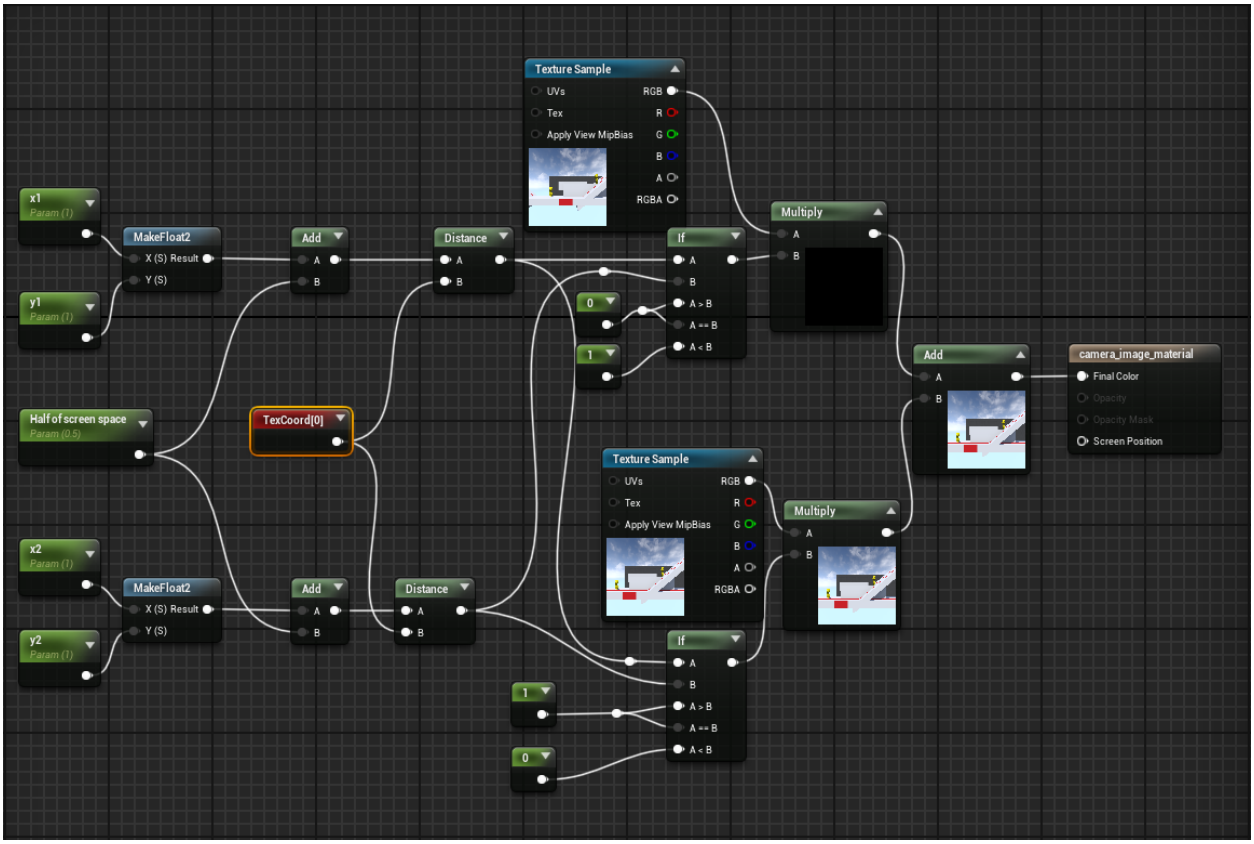
Dünaamilise ekraaniga kaamerasüsteemis haldab materjal ekraani jagamist Voronoi diagrammi Euclideani kauguse valemi järgi, mis arvutab kauguse kahe punkti vahel.
Euclidean kauguse valem [3] $l_2 = d[(a_1, a_2), (b_1, b_2)] = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2}$
a_1 ja a_2 on ühe punkti koordinaadid
b_1 ja b_2 on teise punkti koordinaadid



Joonis 4. Voronoi diagrammi tulemus 20 punkti alusel Euclidean kauguse valemi järgi. [3]

Joonisel (Joonis 4) on näide Euclidean kauguse valemi kasutamisest Voronoi diagrammi saamiseks. Joonisel (Joonis 4) on 20 musta punkti. Piksel saab endale värvi vastavalt sellele, milline must punkt on talle kõige lähemal.

Sarnast loogikat on vaja kasutada ka ekraani pildi saamiseks dünaamilise ekraaniga kaamerasüsteemis. Mustade punktide asemel on mängu tegelaste asukohad ekraani suhtes, kuid ekraani piksel saab endale väärtuse kaugemal oleva tegelase kaamera *render target*'st.



Joonis 5. Voronoi diagrammi arvutamiseks kasutatav loogika materjalis, [2] järgi kohendatud

Joonisel (Joonis 5) x1, y1 on esimese mängu tegelase positsioon mängu maailmas ja x2, y2 on teise mängu tegelase positsioon mängu maailmas.

Funktsioon *MakeFloat2* võtab sisse kaks numbrit ning loob nendest koordinaadi.

TexCoord[0] on (u,v) tekstuuri koordinaat, u ja v miinimum väärtus on 0 ja maksimum 1, mis tähendab, et kõik ekraani pikslid jäävad 0 ja 1 vahele.

Distance node arvutab Euclidean kauguse valemi järgi pikkuse vastavalt koordinaatidele, mis sisse antakse.

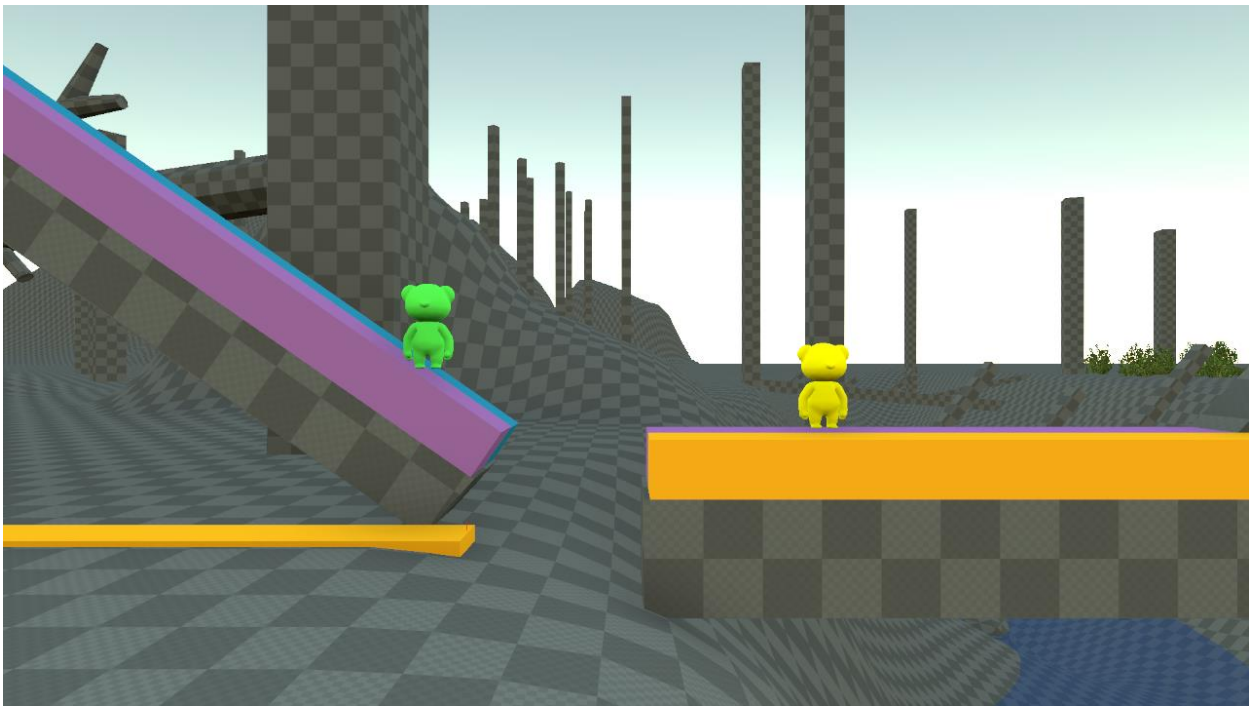
Texture Sample omab endas *render target* väärtust, alumise haru *Texture Sample* võtab sisendiks esimese mängu tegelase kaamera *render target*, ülemine haru *Texture Sample* võtab teise mängu tegelase *render target* sisendiks.

Half of screen space muutuja omab endas poolt maksimaalsest *TexCoord[0]* u ja v väärtusest. Lisada juurde pool maksimaalsest u ja v väärtusest on vajalik, kuna meil on arvutatud tegelaste positsioonid vastavalt ekraani (0,0) positsioonile, kuigi peaks olema arvutatud ekraani keskpunktist (0.5,0.5) *Half of screen space* on lisatud arvutusena, kuna see oleneb *TexCoord[0]* u ja v maksimaalsest väärtusest. Nagu mainitud ennem on maksimaalne väärtus mõlemale 1. Kui muuta u ja v maksimaalset väärtust tuleb muuta ka *Half of screen* väärtust.

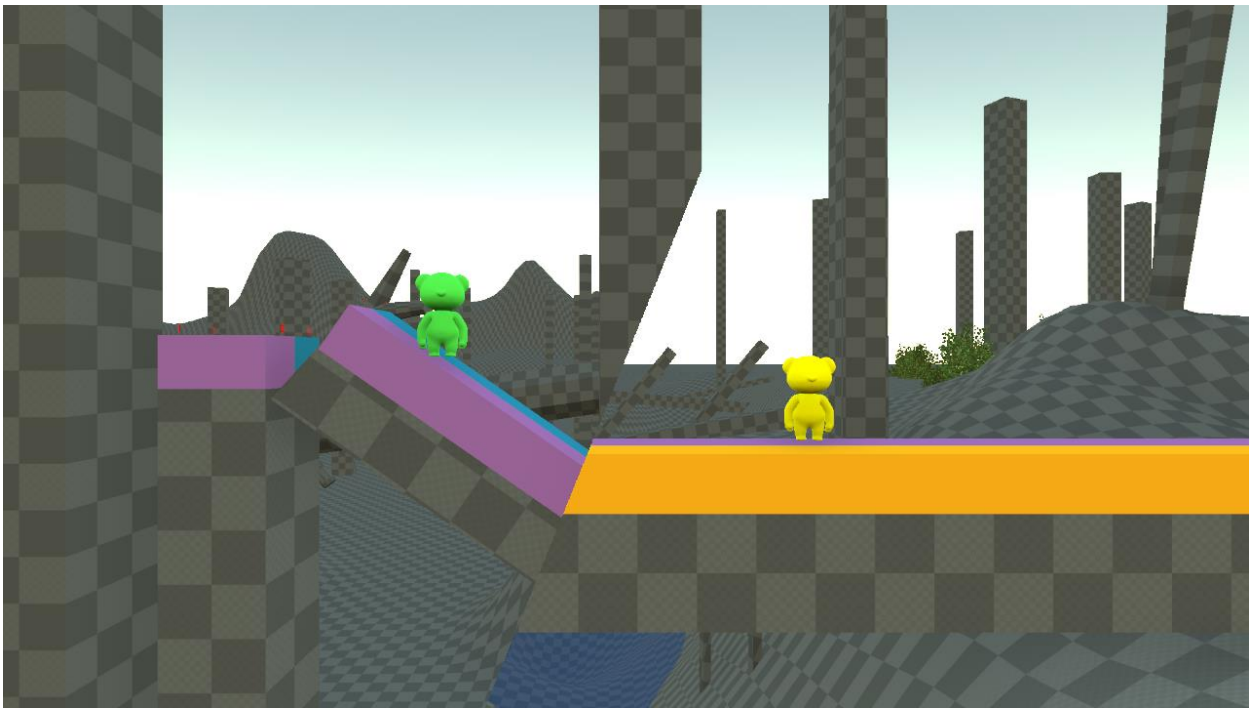
Multiply korrutab kaks väärtust omavahel, võetakse piksli väärtus *render targetist* ja see korrutatakse läbi kas 0 või 1, kui korrutatakse läbi 0, siis muutub piksli väärtus mustaks, kui korrutatakse läbi 1, siis jääb pikslile *render target* väärtus. Arvutus tehakse läbi iga piksli kohta *Texture Sample node*'s. Tulemuseks on uus *Texture Sample*, lihtsuse mõttes nimetame seda edaspidi pildiks. Kui mängu tegelased on üksteisele piisavalt lähedal, et mahtuda samale ekraanile on ühe *Multiply* tulemuseks must pilt, mida on näha ka joonisel (Joonis 5) ja teine pilt on *render target* väärtus. Juhul, kui tegelased on üksteisest kaugemal, et tekiks ekraani jagamine on ühe *Multiply* tulemuseks pilt, kus pool ekraani on must ja teine pool kaetud *render target* piksli väärtustega ning teine *Multiply* tulemus on vastand esimesele *Multiply* pildi väärtusele, ainult *render target* väärtused on võetud teisest *render target'*st.

Add lisab kokku kahe pildi väärtused. *Add* arvutus võtab sisse kaks pilti ning tulemuseks on ekraanipilt. Liites kokku must pilt ja *render target* pilt on tulemuseks *render target* pilt. Juhul, kui tegelased on üksteisest kaugemal, on tulemuseks pilt, mis jätab jagatud ekraani mulje, millest pool ekraani kuulub ühele mängu tegelasele ja teine pool teisele mängu tegelasele.

Näited tulemusest.



Joonis 6. Materjali tulemus, mõlemad mängu tegelased mahuvad ühele ekraanile.



Joonis 7. Materjali tulemus, ekraani jagamine Euclidean kauguse valemit kasutades, kui mängus toimub ekraani dünaamiline jagamine.

Materjali loogika on loodud visuaalse skriptimise keele Blueprint abil.

2.4.7 Kaamera

Kaamerat kasutatakse selleks, et näidata mängijale mängu maailma. See kuidas kaamerat kasutatakse otsustab, millise nurga alt ning kui suurt osa mängu maailmast mängija näeb. Kaamera võib liikuda, kuid see võib ka ühe koha peal olla. Viise, kuidas kaamerat saab kasutada on mitmeid. Mängudes, mis põhinevad teksti kasutamisel ei pruugi kaamerat vaja olla. [15, lk 94]

Antud mängus on kaks mängutegelast ning iga tegelase kohta on üks kaamera. Kaamera asukoht arvutatakse kahe erineva valemi järgi. Nendeks kaheks valemiks on järgnevad:

a) Kui tegelased on üksteisele piisavalt lähedal, et mahtuda samasse ekraani võetakse kahe tegelase keskpunkt kaamera asukohaks, sellisel juhul on mõlemad kaamerad samas kohas ning ainult ühe kaamera pilti näidatakse ekraanil.

$$x_{pos} = (t_{1pos} + t_{2pos}) / 2$$

x_pos on kaamera asukoht
t_1pos on tegelase 1 asukoht
t_2pos on tegelase 2 asukoht

b) Kui tegelased on üksteisest nii kaugel, et ei mahu enam koos ekraanile, siis lähevad kaamerad lahku, üks kaamera jälgib ühte tegelast ja teine teist.

Selleks, et ekraani jagamine toimuks võimalikult sujuvalt tuleb arvutada kaamera asukoht äärmiselt täpselt. Suurim keerukus tekib sellest, kuidas leida korrektne kaamera positsioon. Peale ekraani jagamist, peab kaamera olema tegelasest eemal vastavalt sellele, millise nurga alt on ekraani jagatud, kui kaamera on valel pool tegelast peale ekraani jagamise ei pruugi mängu tegelast enam ekraanil näha olla.

Kuna enne kaamera pildi jagamist on kaamera kahe tegelase keskel oleval positsioonil tuleb kahe tegelase vaheline punkt võtta aluseks ka peale ekraani jagamist.

Probleemi lahendamiseks kasutan järgmiseid valemeid:

$$d = 2\frac{arctan\left(\frac{g}{a}\right)}{\pi}$$ [2]

Antud valemi abil loon vektorid, mis liidetakse kahe tegelase vahelisele punktile.

g on kahe mängu tegelase vaheline vahemaa, millest on lahutatud kahe mängu tegelase vaheline vahemaa, millest alates hakatakse ekraani jagama.
a on vahemaa, millest alates hakatakse ekraani jagama.

d on väärtus suurus, mille abil saab arvutada kaamera positsiooni peale ekraani jagamist, selle väärtus on alati 0 ja 1 vahel, mida kaugemal on mängijad üksteisest seda suurem on *d*.

d tuleb läbi korrutada jälgitava mängu tegelase positsiooniga ja kahe tegelase vahelise punkti vektoriga, et saada vektor, mis tuleb juurde liita kahe tegelase vahelisele punktile. Selle tulemusena jääb mängu tegelane alati kaamera pildile õige nurga alt ning ekraani jagamine näeb sujuv välja.

Lähtudes [2] kirjeldatud teoreetiliselt lähtekoodist, koostasin kaamera asukoha määramiseks järgneva valemi. Kaamera asukoha *x_pos* kirjeldame valemiga.

$$x_{pos} = M + (t_{targetPos} - M) * d$$

$t_{targetPos}$ on tegelase positsioon, keda kaamera jälgib

M on kahe tegelase vaheline positsioon

Kaamera loogika on programmeeritud C++ keeles.

3 Lihasööjataime tehisintellekt

Tehisintellektid mängivad suurt rolli mängu kogemuse kujundamises tuues realismi ja lõbu virtuaalmaailma. Vaenlase tegelase loomine on kõige levinum tehisintellekti rakendamine viis. [5, lk 22]

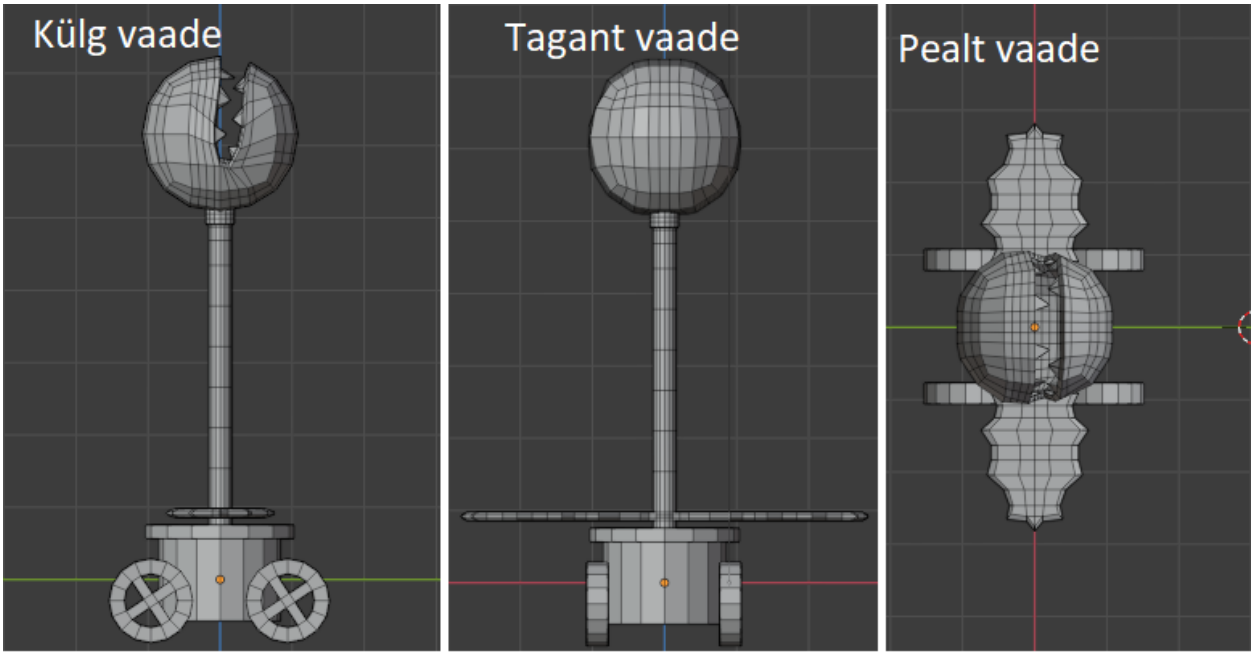
Lihasööjataim on oma mõtlemisega mängu tegelane, kes patrullib kahe punkti vahelist mängu ala. UE4 mängumootoris on taime külge lisatud kaks kokkupõrke ala: ohu kokkupõrke ala ja rünnaku kokkupõrke ala (vt joonist Lisa 3). Kui mängu objekt siseneb ohu kokkupõrke alasse muudab taim ohtlikuks. Kui mängu objekt siseneb rünnaku kokkupõrke alasse ründab taim mängu objekti, mille tulemusena taim kas hävitab mängu objekti või hammustab seda, võtab mängu objekti põske, ning pärast sülitab välja. Lihasööjataime tehisintellekt on loodud UE4 mängumootoris käitumise puu ja tahvli abil. Käitumise puu reguleerib lihasööjataime tegevust. Tegevused, mida lihasööjataim saab teha on loodud C++ programmeerimise keeles.

3.1 Mudel

3D mängude puhul on vaja enamus mängu objektidele luua 3D mudel, mis esindab mängu objekti mängu maailmas.

3D mudel koosneb erinevatest nelinurksetest *polygon*'dest, mis on omavahel seotud. Mudeli tegemiseks kasutas autor 3D modelleerimise programmi Blender.

Blender on tasuta avatud lähtekoodiga 3D objektide loomise komplekt. Blender toetab järgmisi 3D mudelite loomise elemente: modelleerimine, skelettide loomine, animeerimine, simulatsioonid, renderdamine, komponeerimine ja liikumise jälgimine, videotöötlus ja 2D animatsioone. [8]



Joonis 8. Autori poolt loodud lihasööjataime 3D mudel.

Joonisel (Joonis 8) olev mudel sai inspiratsiooni 1999. aastal välja tulnud mängust "Crash Bandicoot" lihasööjataimest[2], kes antud mängus ründas mängija poolt kontrollitud tegelast, kui talle läheneti. Peale mudeli valmimist näeb mudel hoopis rohkem välja Super Mario mängudes olevale "Piranha Plant"[3] tegelasele.

3.2 Animatsioonid

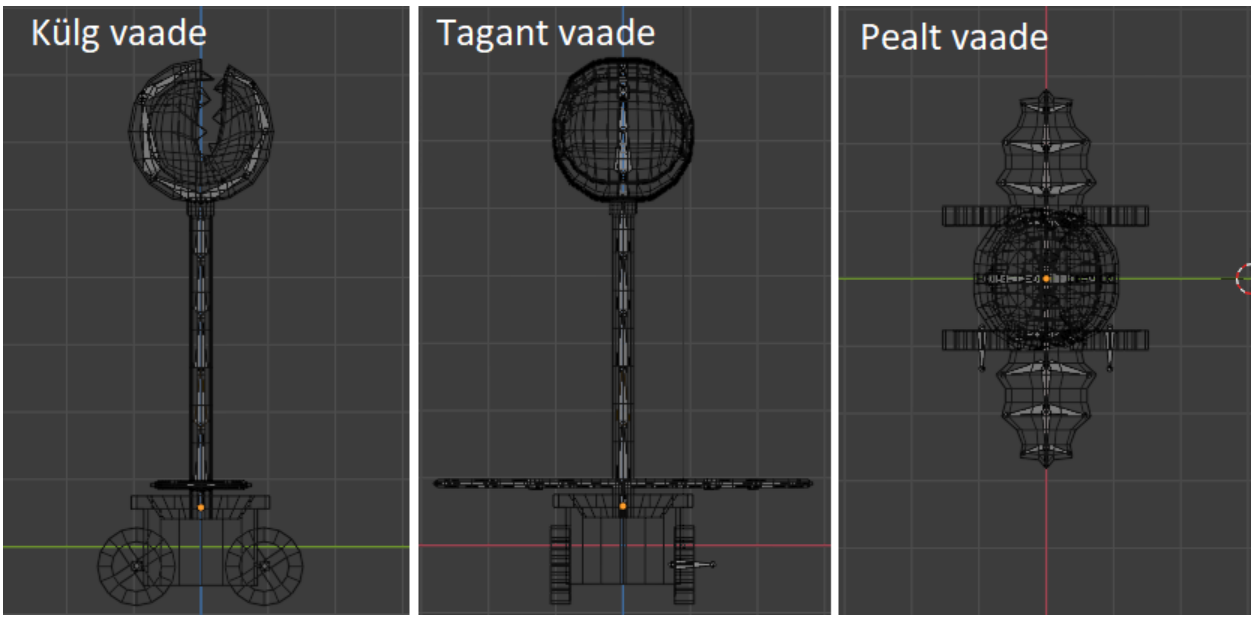
Animeerida tähendab elu anda. Animatsioonid on loodud luues järjestikuseid seisvaid pilte. Kui pilte kuvada kiiresti üksteise järel on võimalik silma petta, jättes mulje nagu piltidel olev tegevus on üks pidev liikumine.[10, lk 5]

3D mudeli puhul ei kasutata animeerimiseks pilte vaid 3D mudeli erinevaid positsioone, millesse mudel seatakse üksteise järel. 3D tegelase positsioonide kiire vahetus välja kui üks pidev liigutus.

[2] https://crashbandicoot.fandom.com/wiki/Man-Eating_Plant
[3] https://www.mariowiki.com/Piranha_Plant

3.2.1 Animatsioonide loomine

Animatsioonide loomiseks tuleb 3D mudelile luua skelett, skeletti loomist nimetatakse *riggimiseks*. *Riggimisega* luuakse 3D mudeline tema luustik. Luustiku kasutatakse, et manipuleerida 3D mudelit nagu nukku animatsioonide jaoks. [9]



Joonis 9. Autori poolt loodud lihasööjataime 3D skelett.

Iga luu kohta luustikus peab sätestama, milliseid *polygon*'e ta 3D mudelil liigutada tohib, seda protsessi nimetatakse raskuste värvimiseks. Blenderis lihtsamate mudelite puhul on võimalik, et selle jaoks ei pea tegema manuaalset tööd, kuid antud mudeli jaoks oli vaja teha manuaalne raskuste värvimine vähemalt pooltele luudele. Peamiseks probleemiks oli suu luude raskuste värvimine, kuna automaatne raskuse värvimine ei suutnud tuvastada, et mudeli peas on suu ava, ning kõik pea luud mõjutasid lisaks enda pea poolele ka teise pea poole *polygon*'e.

Nelja ratta kohta on loodud kaks luud, esimene luu esimeste rataste ning teine tagumiste jaoks.

Animatsioonid loodi kasutades 3D modelleerimise tarkvara Blender. Selleks, et animatsioone luua tuleb otsustada, mitu kaadrit pikk on animatsioon ning panna mudel manuaalselt asendisse, mis tal peaks olema teatud kaadri ajal. Blender on piisavalt tark tarkvara, et näiteks 30 kaadri pikkuse animatsiooni jaoks ei pea kasutaja käsitsi 30 positsiooni mudelile looma, vaid piisab minimaalselt

kahe positsiooni loomisest, üks esimese kaadri jaoks ja teine viimase kaadri jaoks ning nende vahel olevate kaadrite positsioonid interpoleeritakse.

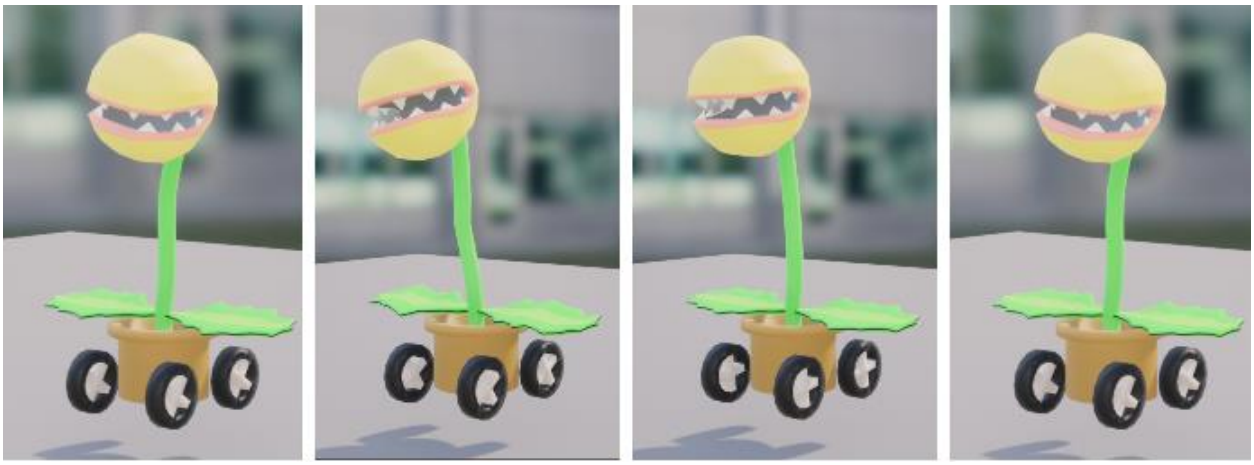
Interpoleerimine matemaatikas on tundmatute väärtuste *f(x)* või funktsiooni *x* määramiseks või ligikaudseks arvutamiseks teada olevate väärtuste abil.[11]

Blender kasutab interpoleerimist luude asukohtade arvutamiseks kaadritel, millele ei ole manuaalset positsiooni loodud.

Lihasööjataimele on loodud neli erinevat animatsiooni. Igale animatsioonile on manuaalselt loodud neli positsiooni ning nende vahed on interpoleeritud. Iga animatsioon on erineva kaadrite pikkusega. Selleks, et animatsioone saaks mängida UE4 mängumootoris, tuleb need importida koos mudeliga UE4 mängumootorisse.

3.2.2 Idle animatsioon

Idle animatsiooni käigus taim näeb välja rahulik ning mitte ohtlik. Patrullimise ajal kasutatakse *idle* animatsiooni.



Joonis 10. Lihasööjataime *idle* animatsiooni positsioonide kogum.

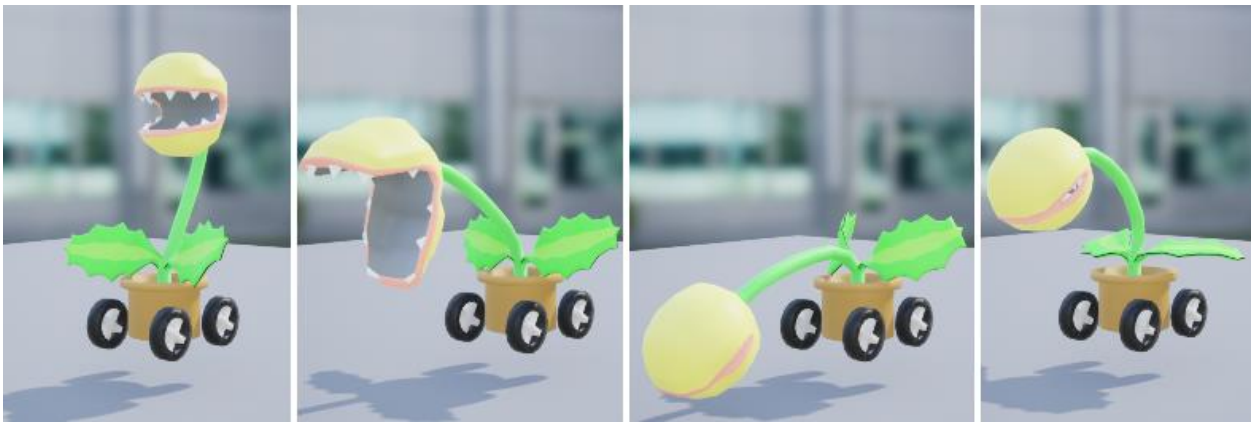
3.2.3 Ohtlik animatsioon

Ohu animatsiooni eesmärgiks on näidata, et ta on ohtlik. Lehtede keeramine vertikaalsesse asendisse jätab taimest suurema mulje ja nende edasi-tagasi liigutamine näitab välja ohtu. Lisaks sellele liigub suu rohkem, kui *idle* animatsiooni puhul.

Joonis 11. Lihasööjataime ohu animatsiooni positsioonide kogum.

3.2.4 Rünnaku animatsioon

Rünnaku animatsiooni käigus liigutab taim pea tahapoole, näitamaks, et ta on valmis ründama ning peale seda ründab. Peale ründamist tõstab taim oma pea tagasi kõrgemale.



Joonis 12. Lihasööjataime rünnaku animatsiooni positsioonide kogum.

3.2.5 Sülitamise animatsioon

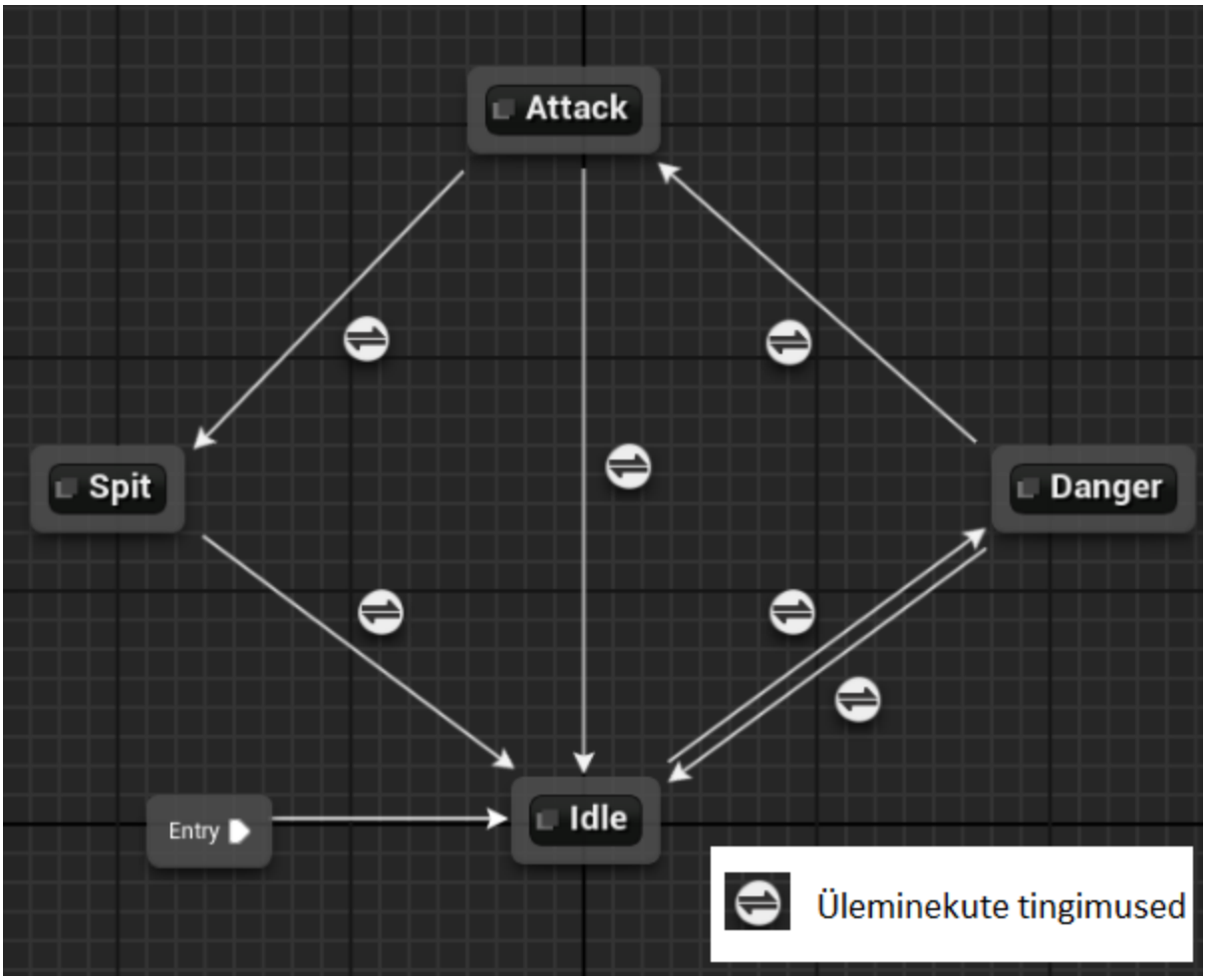
Sülitamise animatsiooni käigus hoiab taim suud kinni kuniks on valmis sülitama. Esialgu viib pea kaugemale, sarnaselt rünnaku animatsioonile, kuid erinevusega tehes lisa pea langetuse enne sirutamist ja välja sülitamist. Sülitamise ajal tehakse suu lahti. Peale sülitamist viib keha asendi *idle* animatsiooni asendi taolisesse asendisse, nagu kirjeldatud punktis 3.2.2.

Joonis 13. Lihasööjataime sülitamise animatsiooni positsioonide kogum.

3.3 Animatsioonide käivitamise loogika

Animatsioone saab UE4 mängumootoris kontrollida oleku masinaga.

Oleku masin pakub graafilist viisi, kuidas jaotada animatsioonid olekuteks. Animatsioonide vahetamiseks kasutatakse ülemineku reegleid, mis kontrollivad, millal ühest animatsioonist minnakse üle teise. [6]



Joonis 14. Lihasööjataime animatsioonide oleku masin.

Joonisel (Joonis 14) on loodud animatsioonide oleku masin, animatsioonid on loodud autori poolt, üleminekute tingimused (Joonisel 14 tähistatud vastava sümboliga) on loodud visuaalselt oleku masinasse, kuid tingimustes olevaid muutujaid kontrollitakse C++ programmeerimise keeles autori poolt.

Joonisel (Joonis 14) kujutatakse lihasööjataime oleku masinat, mille pealt näeb, et lihasööjataimel on vaikimisi *Idle* animatsioon. *Idle* animatsioonile saab järgneda vaid ohu (*Danger*) animatsioon. Ohu animatsiooni mängitakse, kui mõni mängu objekt satub ohu kokkupõrke alasse.

Ohu (*Danger*) animatsioonile saab järgneda kas *idle* animatsioon või rünnaku (*Attack*) animatsioon. *Idle* animatsioon järgneb ohu animatsioonile, kui ohu kokkupõrke alasse ei ole

jäänud mitte ühtegi objekti. Ohu animatsioonile järgneb rünnaku animatsioon, kui mängu objekt läheb rünnaku kokkupõrke alasse.

Rünnaku (*Attack*) animatsioonile saab järgneda kas *idle* või sülitus animatsioon.
Idle animatsioon järgneb rünnaku animatsioonile, kui taim hävitas mängu objekti, mida ta ründas. Sülitamise animatsioon järgneb rünnaku animatsioonile, kui taim hammustab mängu objekti, mille ta peab välja sülitama.

Sülitamise (*Spit*) animatsioonile saab järgneda vaid *idle* animatsioon, mida mängitakse peale seda, kui taim on mängu objekti välja sülitanud.

3.4 Tehisintellekt

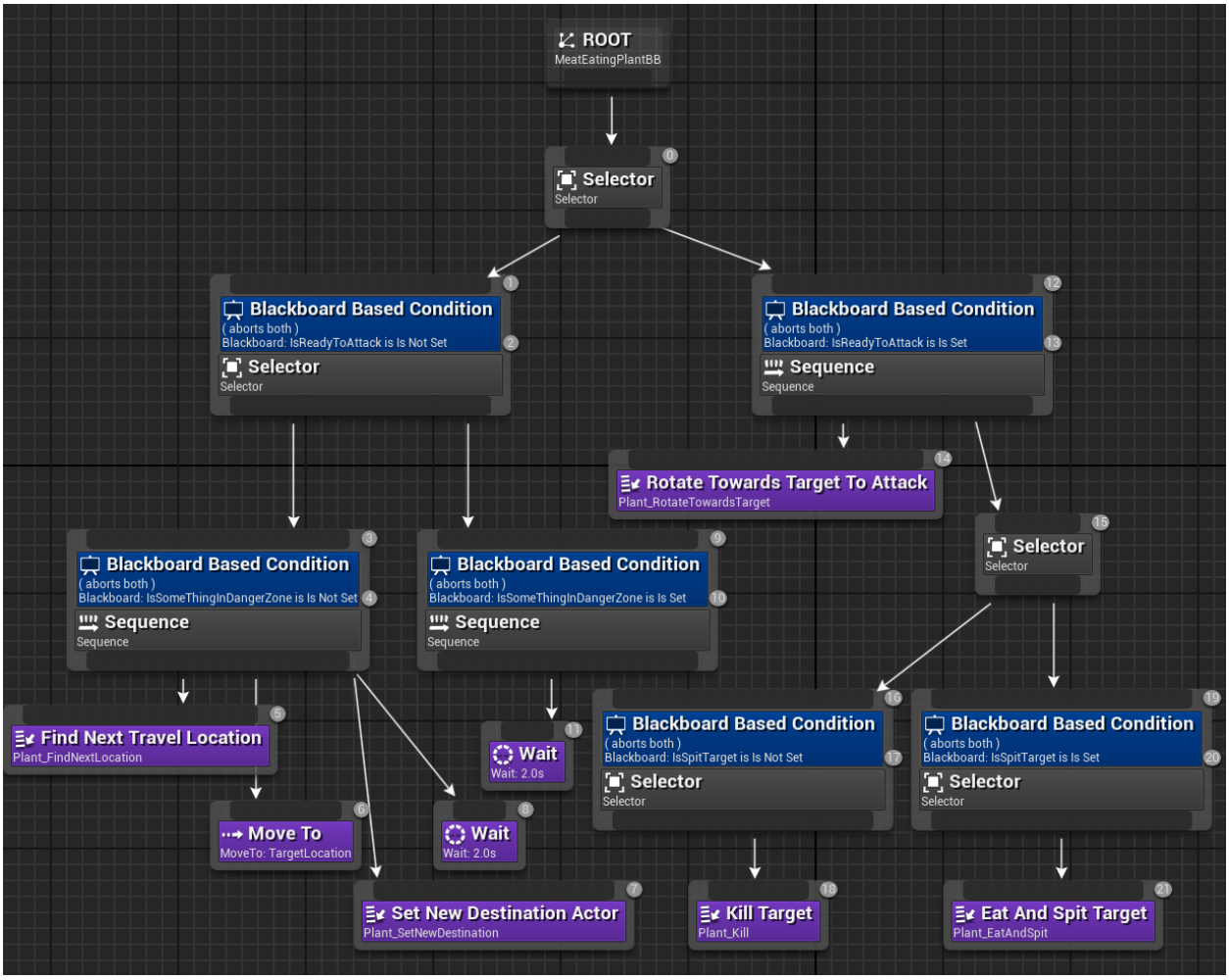
Lihasööjataim peab liikuma kahe erineva punkti vahel mängu maailmas. Kui talle jääb tee peale ette mõni mängu objekt, jääb ta seisma, näitab, et ta on ohtlik. Kui mängu objekt tuleb talle veel lähemale, siis ta ründab mängu objekti.

Mängu tegelasele tehisintellekti loomiseks on UE4 kasulik kasutada käitumise puud (ingl. *behavior tree*) ja tahvlit (ingl. *blackboard*).

Tehisintellekti käitumise puu ja oleku masin töötavad paralleelselt.

3.4.1 Käitumise puu

Kõige lihtsam on mõelda käitumise puu rollist tehisintellekti tegelase juures, kui tema ajust. Ta teeb otsuseid ning tagajärjena käitub nende põhjal. Käitumise puid loetakse ülevalt alla ja *node* käivitatakse vasakult paremale. [7, lk 33] Lisaks sellele on igal *node*’l number paremal ülemises nurgas, mis näitab *node*’de käivitamise järjekorda.



Joonis 15. Lihasööjataime tehisintellekti käitumise puu.

Käitumise puu koosneb viiest erinevast *node*'st: *task*, *decorator*, *service*, *composite* ja *root*. [7, lk 35]

Antud projektis kasutatakse *root* (vt peatükk 3.4.1.1), *decorator* (vt peatükk 3.4.1.4), *composite* (vt peatükk 3.4.1.3) ja *task* (vt peatükk 3.4.1.2) *node*'d.

3.4.1.1 Root

Root node on puu algus. *Root node*'l saab olla ainult 1 alamkomponent, ning see komponent peab olema *composite* tüüpi. [9]

3.4.1.2 Task

Task'd viivad ellu erinevaid tehisintellekti tegevusi, näiteks mängu tegelase liigutamine. *Task*'del ei ole väljundit, mis tähendab, et *task*'d ei mängi rolli otsustamisel, millal nad käivitatakse, vaid viib ellu ülesande, mis talle on antud. [7, lk 37-38]
Task lõpuks antakse tagasisidet, kas töö õnnestus või mitte.

Task'd on joonisel (Joonis 15) peal lilla taustavärviga.

Find Next Travel Location *task* otsib asukoha, mille poole taim peaks liikuma. Sätestab tahvlis olevale *TargetLocation* vektor tüüpi muutujale asukoha mängu maailma ruumis, kuhu taim peaks liikuma.

Move To *task* paneb taime liikuma asukoha poole mis *Find Next Travel Location* määras uueks asukohaks.

Set New Destination Actor *task* salvestab uue sihtkoha objekti. Sihtkoha objekt on objekt, mis asub mängu maailmas, temalt saab küsida viite järgmisele sihtkoha objektile. Nii saab luua kindla järjestuse asukohtadest, kus mängu tegelane peab liikuma.

Wait *task* peatab mängu tegelase, etteantud sekunditeks.

Rotate Towards Target To Attack *task* keerab lihasööjataime objekti suunas, mida ta hakkab ründama.

Kill Targe**t** *task* tapab objekti, kes sisenes rünnaku kokkupõrke alasse, kui rünnaku animatsioon on jõudnud nii kaugele, et hammustus toimub, siis objekt, mida rünnatakse, hävitatakse.

Eat And Spit Target *task* ründab objekti, kes sisenes rünnaku kokkupõrke alasse, kui rünnaku animatsioon on jõudnud nii kaugele, et hammustus toimub, siis objekt, mida rünnatakse, muudetakse ajutiselt nähtamatuks. Kui rünnaku animatsioon on lõppenud, alustatakse sülitamise animatsiooni, kui sülitamise animatsioon on nii kaugel, et taime suu on lahti, siis asetatakse

rünnatud objekti asukohaks taime avatud suu asukoht. Rünnatud objekt muutub uuesti nähtavaks, ning sellele lisatakse kiirendus, mille tulemus näeb välja nagu objekt oleks välja sülitatud.

Joonisel (Joonis 15) on kujundatud käitumise puu. Puu loogika on loodud autori poolt ning kõik *task*'d (välja arvatud *Move To* ja *Wait*) on autori poolt loodud C++ programmeerimiskeeles.

3.4.1.3 Composite

Composite node'd annavad käitumise puule otsuste langetamise võimekuse. On olemas kolme tüüpi *composite node*: *selector, sequence* ja *simple parallel*. [7, lk 40] Antud projektis kasutatakse *selector* ja *sequence composite node*.

***Selector** node* otsustab millise oma alam *node*'dest käivitada. Otsustus käib vasakult paremale, proovib käivitada järjest *node* kuniks leiab *node*, mille käivitamine õnnestus. [7, lk 41] Joonisel (Joonis 15) on *selector node*'d halli värvi ning nendel on kirjas "*Selector*".

Selector node kasutatakse käitumise puus koos *decorator node*'dega. *Selector node* käivitab oma puus alam objekti, kui *decorator node* väärtus on tema jaoks tõene.

***Sequence** node* töötab vastupidiselt *selector node*'le, *sequence node* käivitab kõik oma alam *node*'d vasakult paremale järjekorras, kui üks alam *node* õnnestub, siis käivitatakse järgmine, kui kõik alam *node*'d on edukad, siis antakse ülemisele *node*'le teada, et *sequence* oli edukas. Kui üks alam *node*'dest kukub läbi siis antakse ülemisele *node*'le teada, et *sequence* ebaõnnestus. [7, lk 41]
Joonisel (Joonis 15) on *sequence node*'d halli värvi, ning nendel on kirjas "*Sequence*".

Sequence node kasutatakse käitumise puus peamiselt patrullimiseks, patrullimisel on vaja korrata nelja tegevust. Esiteks taim peab küsima enda olemas olevaselt sihtkoha objektilt tema asukoha, kuhu liikuda. Teiseks hakata uue asukoha suunas liikuma. Kolmandaks küsima sihtkoha objektilt uus sihtkoha objekt ning viimaseks ootama kaks sekundit. Teine koht, kus kasutatakse *sequence node* on olukorras, kui mõni mängu objekt on ohutsoonis, selle peale taim lõpetab liikumise, mängides tsüklis kahe sekundilist ootamist, kuniks mängu objektid on lahkunud ohutsoonist.

3.4.1.4 Decorator

Decorator node'id ühendatakse kas *composite* või *task node*'dega. *Decorator node*'d võtavad vastu otsuseid, kas *node* millega ta on ühendatud käivitatakse või mitte. Lihtsamalt võttes on *decorator node*'d tingimused, millal ühendatud *node* võib käivitada. [7, lk 41]
Joonisel (Joonis 15) on *decorator node*'d nähtaval tumesinisel taustal.

Antud projektis kontrollivad *decorator node*'d, *boolean* väärtuseid: kas mõni mängu objekt on sisenenud ohu kokkupõrke alasse, kas mõni mängu objekt on sisenenud rünnaku kokkupõrke alasse ja kas mängu objekt, mis on sisenenud rünnaku kokkupõrke alasse tuleb hävitada või välja sülitada.
Decorator node'de antud projektis katkestavad alam objektide tegevuse koheselt, kui nende staatus muutub, et puu saaks koheselt korrektselt reageerida ümberringi toimuvale.

3.4.2 Tahvel

Käitumise puust võiks mõelda kui mängu tegelase ajust ja tahvlist kui tema mälust. Tahvel sai endale nime selle järgi, et klassiruumides kirjutatakse informatsioon tahvlile, et seda jagada. Igal käitumise puul on oma tahvel. Tahvlile lisatakse väärtuseid võtme ja väärtuse paaridena. [7, lk 49-50]

Antud projektis hoiab tahvel järgmisi väärtuseid:

1. asukoht, mille suunas liikuda;
2. kontrolli, kas midagi on ohutsoonis;
3. kas midagi peaks ründama;
4. kas objekt, mida rünnatakse peaks välja sülitama.

Kolm kontroll väärtust on kasutatud *decoratorite* poolt, et käivitada korrektne tegevus lihasööjataime poolt.

4 Kokkuvõte

Töö eesmärgiks oli luua dünaamiline kaamerasüsteem *side-scroller* mitmikmängudele ning tutvustada erinevaid kasutusel olevaid *side-scroller* mitmikmängude kaamerasüsteeme. Lisaks oli eesmärgiks luua lihasööjataime tehisintellekt.

Dünaamilise kaamerasüsteemi loogika kirjutati peamiselt C++ programmeerimise keeles, mõne komponendi jaoks kasutati Unreal Engine 4 visuaalset skriptimise keelt Blueprint.

Töö tulemusena valmis dünaamiline *side-scroller* mitmikmängu kaamerasüsteem, kus mängijate tegelased on ühisel ekraanil, kui nad on üksteisele piisavalt lähedal, et mahtuda samasse ekraani ruumi ning saavad ka üksteisest kaugemale minna, kui ekraani suurus. Sellisel juhul jagatakse ekraan pooleks sellise nurga alt, et mängijad saavad alati aru, millises suunas nad üksteise suhtes asuvad.

Lisaks valmis töö tulemusena lihasööjataime tehisintellekt, mis liigub ette antud mängu maailma asukohtade vahel, samal ajal patrullides enda ümbrust, rünnates objekte, mis talle lähenevad. Rünnatud objektid kas hävitatakse või sülitatakse välja.

Lihasööjataime jaoks loodi 3D mudel, mudeli skelett ja neli animatsiooni modelleerimise tarkvara Blenderi abil. Lihasööjataime loogika kirjutati peamiselt C++ keeles. Animatsioonide kontrollimise jaoks kasutati Unreal Engine 4 animatsioonide oleku masinat. Tehisintellekti otsuste tegemise jaoks kasutati Unreal Engine 4 käitumise puud.

Töö käigus lahendati erialase kirjanduse ja empiiriliste katsetuste abil mitmeid unikaalseid probleeme. Tulemuseni jõudmiseks kasutas autor trigonomeetrilisi ja vektorarvutustel põhinevaid algoritme ning käitumispuud. Kõik töö eesmärgid said täidetud.

Kasutatud kirjandus

[1] Epic Games, Inc “Materials - Unreal Engine 4 Documentation” [Võrgumaterjal] https://docs.unrealengine.com/en-US/Engine/Rendering/Materials/index.html [Kasutatud 20.10.2020]

[2] Matt Woelk “Voronoi split screen notes” [Võrgumaterjal] https://mattwoelk.github.io/voronoi_split_screen_notes/ [Kasutatud 20.10.2020]

[3] Wikipedia, “Voronoi diagramm” [Võrgumaterjal] https://en.wikipedia.org/wiki/Voronoi_diagram [Kasutatud 20.10.2020]

[4] Technopedia, “Side scroller definition” [Võrgumaterjal] https://www.techopedia.com/definition/27153/side-scroller [Kasutatud 20.10.2020]

[5] P. L. Newton ja J. Feng “Unreal Engine 4 AI Programming Essentials”, 2016

[6] Epic Games, Inc “State Machines - Unreal Engine 4 Documentation” [Võrgumaterjal] https://docs.unrealengine.com/en-US/AnimatingObjects/SkeletalMeshAnimation/StateMachines/index.html [Kasutatud 20.10.2020]

[7] F. Sapio “Hands-On Artificial Intelligence with Unreal Engine”, 2019

[8] Blender - Koduleht [Võrgumaterjal] https://www.blender.org/ [Kasutatud 21.12.2020]

[9] J. Petty “What is 3D Rigging For Animation & Character Design?” [Võrgumaterjal] https://conceptartempire.com/what-is-rigging/ [Kasutatud 21.12.2020]

[10] R. R. Renomeron, J.P. Lopez Jr., S. R. Morales “Competency Based Learning Module on Animation 211 2d Animation. Module 1 Produce Key Drawing Animation” [Võrgumaterjal] https://www.coursehero.com/file/41767685/Learning-Module-1-2D-Animationpdf/ [Kasutatud 28.12.2020]

[11] Britannica Entsüklopeedia toimetajad, “Interpolation” [Võrgumaterjal] https://www.britannica.com/science/interpolation [Kasutatud 21.12.2020]

[12] Epic Games, Inc “Widgets - Unreal Engine 4 Documentation” [Võrgumaterjal] https://docs.unrealengine.com/en-US/Engine/Rendering/Materials/index.html [Kasutatud 22.10.2020]

[13] OXM Staff, “The most crucial part of video-game development explained - and how it powered Fortnite's runaway success“ [Võrgumaterjal] https://www.gamesradar.com/what-is-a-game-engine-and-what-does-it-do/ [Kasutatud 20.10.2020]

[14] H. Fozi, G. Marques, D. Pereira, D. Sherry „Game Development Prjects with Unreal Engine“, 2020
[15] A. Anthropy, N. Clark, “A Game Design Vocabulary”, 2014
[16] M. MacDonald “Mastering C++ Game Development”, 2018
[17] J. Horton “Beginning C++ Game Programming”, 2019
[18] informIT, “Create an Arcade Shooter Game with Unreal Engine 4” [Võrgumaterjal] https://www.informit.com/articles/article.aspx?p=2521585&seqNum=3 [Kasutatud 13.12.2020]
[19] Epic Games, Inc “Instanced Materials - Unreal Engine 4 Documentation”[Võrgumaterjal] https://docs.unrealengine.com/en-US/RenderingAndGraphics/Materials/MaterialInstances/index.html [Kasutatud 13.12.2020]
[20] T. Tran “Unreal Engine 4 Tutorial: Painting With Render Targets” [Võrgumaterjal] https://www.raywenderlich.com/5246-unreal-engine-4-tutorial-painting-with-render-targets [Kasutatud 14.12.2020]
[21] Independent Games summit, “Scroll Back: The Theory and Practice of Cameras in Side-Scrollers”, 02.06.2015. [Võrgumaterjal] https://www.gdcvault.com/play/1022243/Scroll-Back-The-Theory-and [Kasutatud 14.12.2020]
[22] GameDesigning, “Make Your First Multiplayer Game (Start Here)” [Võrgumaterjal] https://www.gamedesigning.org/learn/multiplayer/ [Kasutatud 22.12.2020]

Lisa 1 – Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks[4]

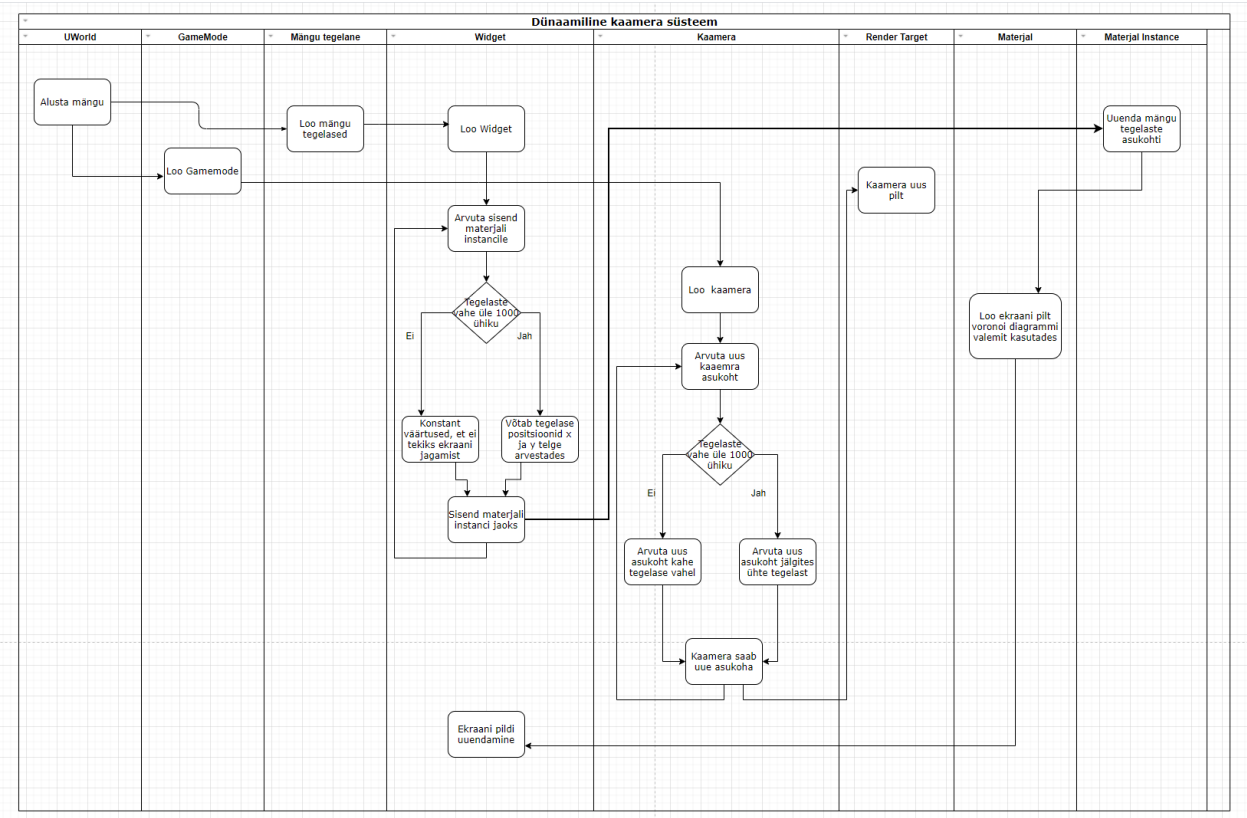
Mina, Karl Lõoväli

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „DÜNAAMILISE KAAMERASÜSTEEMI JA LIHASÖÖJATAIME TEHISINTELLEKTI LOOMINE UNREAL ENGINE 4 MÄNGUMOOTORIL“, mille juhendaja on Martin Rebane
   1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
   1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.
3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

01.01.2021

4 Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või enam isikut oma ühise loomingulise tegevusega ning lõputöö kaas- või ühisautor(id) ei ole andnud lõputööd kaitsvale üliõpilasele kindlaksmääratud tähtajaks nõusolekut lõputöö reprodutseerimiseks ja avalikustamiseks vastavalt lihtlitsentsi punktidele 1.1. ja 1.2, siis lihtlitsents nimetatud tähtaja jooksul ei kehti.

Lisa 2 - Kaamerasüsteemi jadadiagramm



Lisa 3 - Lihasööjataime kokkupõrke alad