TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Britta Pung 155203IAPB

KUUELAMU IGLUNA 3D SIMULATSIOON

Bakalaureusetöö

Juhendaja: Jüri Vain
Professor

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: Britta Pung

21.05.2019

Annotatsioon

Antud lõputöö eesmärgiks on luua kuuelamu 3D simulatsioon, mis oskab suhelda kuuelamu sisekliima juhtimistarkvaraga ning reageerida saadud sisendile, et demonstreerida ning verifitseerida juhtimistarkvara tööd.

Töö toimub IGLUNA projekti raames. IGLUNA on rahvusvaheline tudengite demoprojekt, kus meeskonnad 14-st Euroopa ülikoolist teevad koostööd, et ehitada võimaliku kuuelamu ehk "iglu" prototüüp. Lõputöö autori meeskonna eesmärk on luua õppimisvõimeline juhtimistarkvara, mis reguleerib iglu valgustust ning temperatuuri. Tarkvara eesmärk on optimeerida energiakasutust ning teha iglu elanike elu mugavamaks.

Lõputöö autori ülesanne on luua 3D simulatsioon, mille abil on võimalik juhtimistarkvara testida. Simulatsiooni loomiseks kasutatakse mängumootorit Unreal Engine 4. Simulatsioonis on 3D mudel kuuelamust ning inimtegelased, kes teevad erinevaid tegevusi, näiteks magamine, söömine või töötamine. Simulatsioon teavitab juhtimistarkvara kõikidest iglu oleku muutustest ning reageerib juhtimistarkvara poolt saadetud käsklustele. Simulatsioon kuvab kasutajale informatsiooni iglu sensorite hetkeseisu ja toimunud sündmuste kohta ning võimaldab kasutajal sisestada väärtusi, mida juhtimistarkvara võtab otsuste tegemisel arvesse.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 29 leheküljel, 5 peatükki, 18 joonist, 0 tabelit.

Abstract

3D Simulation of the IGLUNA Moon Habitat

The main objective of this thesis is to create a 3D simulation of a moon habitat which is capable of communicating with habitat climate control software and reacting to its commands. The practical goal of the simulation is to demonstrate and verify the operation of the climate control software in realistic scenarios.

This thesis project is part of the IGLUNA moon habitat project, where teams from 14 different European universities are collaborating to create a prototype of a possible moon habitat or "igloo". The main goal of the author's team is to create control software that controls the igloo's lighting and heating systems with the aim of optimizing energy usage and meeting the inhabitants' comfort preferences.

The author's task is to create a 3D simulation which can be used for testing and demonstrating the use of the rule-based control software. The simulation is made using the game engine Unreal Engine 4. It contains a 3D model of the moon habitat and characters controlled by AI controllers. The characters perform various activities like sleeping, eating, working, etc. The AI controllers use a behaviour tree which picks a random activity and moves the character to the correct location. After a random period of time this action is repeated. This allows creating realistic situations where climate control should decide on control set points.

The simulation displays information about the current state of all sensors and any events that have occurred, as well as allowing the user to input values which the control software will take into consideration. Communication between the simulation and the control software is implemented through text files. The simulation keeps an updated list of every sensor value and every character's current activity and also informs the control software of any changes the user has made. The control software can send the simulation task inputs such as changing the light value of a certain room or displaying a warning message, and the simulation will act accordingly.

The thesis is in Estonian and contains 29 pages of text, 5 chapters, 18 figures, 0 tables.

Lühendite ja mõistete sõnastik

Actor	*Blueprintide* kogumik, mis võimaldab objekti paigutada mängumaailma
AI controller	*Blueprintide* kogumik, mis juhib *Pawn* objekti
API	*Application Programming Interface*, programmiliides, API-liides. Liides, mis defineerib, kuidas toimib programmidevaheline suhtlus.
Blueprint	Üks klass Unreal Engine 4 visuaalses skriptimiskeeles
Blueprint node	*Blueprintis* kasutatav kastike, mis esindab ühte funktsiooni ning millele saab määrata sisendeid ning väljundeid
Blueprints Visual Scripting	Unreal Engine 4 mängumootoris kasutatud visuaalne skriptimiskeel
Collider	Kujund, mis määrab ära objekti kokkupõrkeala
Complex collision	*Collider* objekt, milleks on objekti kuju täpselt jälgiv kujund
Controller	*Pawn* objekti juhtiv kood
Event Graph	Osa *blueprint* objektist, kus on võimalik kodeerida ajast sõltuvaid ülesandeid
EXE	Täitmisfaili formaat. Faililaiend on ".exe".
FBX	*Filmbox*, failiformaat 3D mudelite salvestamiseks. Faililaiend on ".fbx".
GameInstance	Globaalse ligipääsuga objekt, mis kestab läbi terve mängu
Iglu	IGLUNA kuuelamu mudel
Level	Mängutase
Lightmass Importance Volume	*Actor*, mis defineerib mängutaseme alad, kus peab olema detailne ja realistlik valgus
Mängumootor	Arvutimängude loomiseks mõeldud tarkvararaamistik
Mängutegelane	*Pawn* objekt
Nav Mesh Bounds Volume	*Actor*, mis defineerib mängutaseme alad, kus tegelane saab liikuda
Pawn	*Actor*, mida saab mängija või *AI controller* juhtida
Skeletal mesh	3D tegelase mudel, millel on ära määratud sisemise skeleti kuju ning mida on võimalik animeerida

Simple collision	*Collider* objekt, milleks on primitiivne kujund, mis objekti ümbritseb
TXT	Tekstifail, ehk fail mis sisaldab ainult lihtteksti. Faililaiend on ".txt".
Valguskaart	2D kujutus "lahtipakitud" 3D objektist, mis võimaldab UE4-l objekti korrektselt valgustada
Vidin	*Blueprint*, mis saab kasutada kasutajaliidese tegemisel
VR	*Virtual reality*, virtuaalreaalsus
UE4	Unreal Engine 4
UV kaart	2D kujutus "lahtipakitud" 3D objektist, mida UE4 kasutab valgusekaartide genereermiseks ning mille peale on võimalik tekstuure teha

Sisukord

Jooniste loetelu

1 Sissejuhatus

1.1 Probleemi taustast ja motivatsioonist

Teel interplanetaarse liigini loovad õpilased tehnoloogia telliskive, millest ehitada üles elukeskkond, kus elu on elamist väärt. [1]

IGLUNA [1] on 2018. aastal Šveitsi Kosmosekeskuse poolt algatatud rahvusvaheline tudengite demoprojekt, mille käigus meeskonnad 14-st Euroopa ülikoolist teevad koostööd, et ehitada valmis kuupealse elamu ehk „iglu“ prototüüp. Projektide arendamine toimub terve aasta ning kulmineerub 2019. aasta suvel demoüritusega, mis toimub Šveitsis Matterhorni jääkoopas ja kus eksponeeritakse kõiki valminud projekte.

Lõputöö autor osaleb Tallinna Tehnikaülikooli meeskonnas. Meeskonna eesmärk on luua õppimisvõimeline juhtimistarkvara, mis optimeerib iglu energiakasutust ning arvestab maja elanike vajaduste ja soovidega. Hea lahendus teeks maja elanike elu palju lihtsamaks ning vähendaks ebavajalikke energiakulusid. Lahenduse arendamisel vaadeldakse energiajuhtimise probleemi kui spetsiifiliste nõuetega targa maja reeglipõhise juhtimise probleemi. Energiajuhtimise lahendamisel keskendutakse temperatuuri ning valguse juhtimisele ning samuti ohuolukordade äratundmisele ning nendest teavitamisele. Reeglipõhine juhtimistarkvara luuakse keeles SWI-Prolog [2].

Lõputöö autori panuseks ülesande lahendamisel on luua 3D simulatsioon, mis kujutab võimalikku kuuelamut ning selle elanike igapäevaseid tegevusi ja võimaldab tiimikaaslaste poolt loodud energiajuhtimise tarkvara töötamist valideerida ning visualiseerida. Simulatsiooni abil on võimalik näha, kuidas maja reageerib keskkonna tingimuste muutustele ning kohandub vastavalt elanike vajadustele, ja selle läbi teha kindlaks, et tarkvara töötab ootuspäraselt. Samuti võimaldab simulatsioon tarkvara efektselt demonstreerida demoüritusel, kuna on võimalik tarkvara tööd reaalajas jälgida.

1.2 Ülesandepüstitus

Simulatsiooni ülesanne on visualiseerida kuuelamu sisekliima juhtimissüsteemi käitumist, et valideerida reeglipõhise juhtimissüsteemi tarkvara jõudlust ning funktsionaalset korrektsust.

Simulatsioonis peab olema 3D mudel kuuelamust, mis on jagatud erinevate otstarvetega tubadeks – magamistoad, töötoad, puhkeala, treeningsaal jne. Peab olema võimalik hoida mälus muutujaid, mis esindaksid erinevate sensorite väärtuseid, ehk iga ruumi puhul peab olema teada selle hetke temperatuur ning valguse intensiivsus ehk valgustugevus. Samuti peab olema võimalik kuvada erineva tugevusega valgust ning selle muutumist juhtida läbi koodi.

Simulatsioonis peab olema võimalik näidata iglu elanikke, kes liiguvad majas ringi ning teevad erinevaid tegevusi, näiteks magamine, söömine, töötamine. Tegevused on seotud kindla asukohaga, nii et inimesed peavad liikuma tegevuse sooritamiseks selleks ettenähtud ruumi.

Kasutajal peab olema võimalik muuta valguse- ning temperatuurieelistusi. See võimaldab kasutajal otseselt mõjutada simulatsiooni seisu ning imiteerida iglu elaniku otsustusi ise valgust või temperatuuri muuta. Eelistus võetakse juhtimistarkvara poolt arvesse ning seda kasutatakse edasiste juhtimisotsuste tegemisel.

Töös peab olema defineeritud API, mille läbi simulatsioon ja juhtprogramm omavahel suhtlevad. Simulatsioon peab tegema juhtprogrammile nähtavaks sensorite väärtused ning simulatsiooni oleku muutused ja peab oskama reageerida juhtprogrammilt saadud käsklustele.

1.3 Ülevaade sarnastest lahendustest

Simulatsioonid leiavad laialdast kasutamist mitmetes erinevates valdkondades. Eriti kasulikud on need siis, kui reaalsete eksperimentide läbiviimine on äärmiselt kulukas või ajamahukas või on veel hoopis võimatu.

Järgnevalt on välja toodud mõned näited erinevatest simulatsioonidest:

- Mars-sim [3] on avatud lähtekoodiga Java projekt, kus simuleeritakse inimeste elu Marsil, võttes fookusesse inimestevahelise suhtluse, tööülesannete jagamise ning meeskonnatöö. Simulatsioonis on 2D mudel elamukompleksist, kus inimesed on kujutatud väikeste ringidena. Iga inimese kohta on teada erinevaid andmeid: asukoht, tegevus, töökoht, iseloomutüüp, stressitase jne. Inimestele määratakse tööülesandeid, mida nad võivad teha üksi või mitmekesi. Simulatsioon on sarnane antud lõputöö projektile, aga mars-simis on elamu juhtimise asemel fokuseeritud inimestevahelisele suhtlusele ning inimeste elu organiseerimisele võõras keskkonnas.

- HI-SEAS [4] on füüsiline simulatsioon, mille puhul on Hawaii Mauna Loa vulkaani külje alla ehitatud elamu, kus viiakse läbi Marsi või Kuu elukeskkonda simuleerivaid missioone. Missioonid kestavad enamasti mitu kuud. Meeskond ei tohi simulatsiooni alalt lahkuda, nad tohivad süüa ainult kaasas olevat toitu või laboris kasvatatud juurvilju ning suhtlus välismaailmaga toimub 20-minutilise ooteajaga. Projekti põhiline eesmärk on jälgida, kuidas meeskond suudab pikaajalistel missioonidel väga piiratud elukeskkonnas hakkama saada ning koostööd teha. See sarnaneb lõputöö projektiga, kuna simuleeritakse maavälist elukeskkonda, kuid kuna HI-SEAS on füüsiline simulatsioon on selle ehitamine ning missioonide läbiviimine palju kulukam ja ajamahukam. Sellist projekti on mõistlik eelnevalt katsetada arvutisimulatsiooniga.

- CIROS [5] on simulatsiooniprogramm, kus saab testida tööstusrobotite jaoks kirjutatud koodi. Programmis on olemas 3D mudelid mitmetest erinevatest robotitest ning on võimalik jooksutada koodi ning näha roboti toimimist reaalajas. See on analoogne antud lõputöö projektile, kuna võimaldab testida tarkvara tööd ning näha selle tulemusi 3D-s.

- The Sims [6] on arvutimängude sari, mis simuleerib päris elu. Mängija saab ehitada maju ning kontrollida nende elanike igapäevaseid tegevusi. Inimesed, keda mängija ei kontrolli on juhitud agendipõhise tehisintellekti poolt ning oskavad iseseisvalt oma elu elada. Lõputöö autor võttis simulatsiooni ehitamisel The Simsist inspiratsiooni, kuna nii inimeste kui maja töö juhtimise loogika on mõlemas projektis väga sarnane.

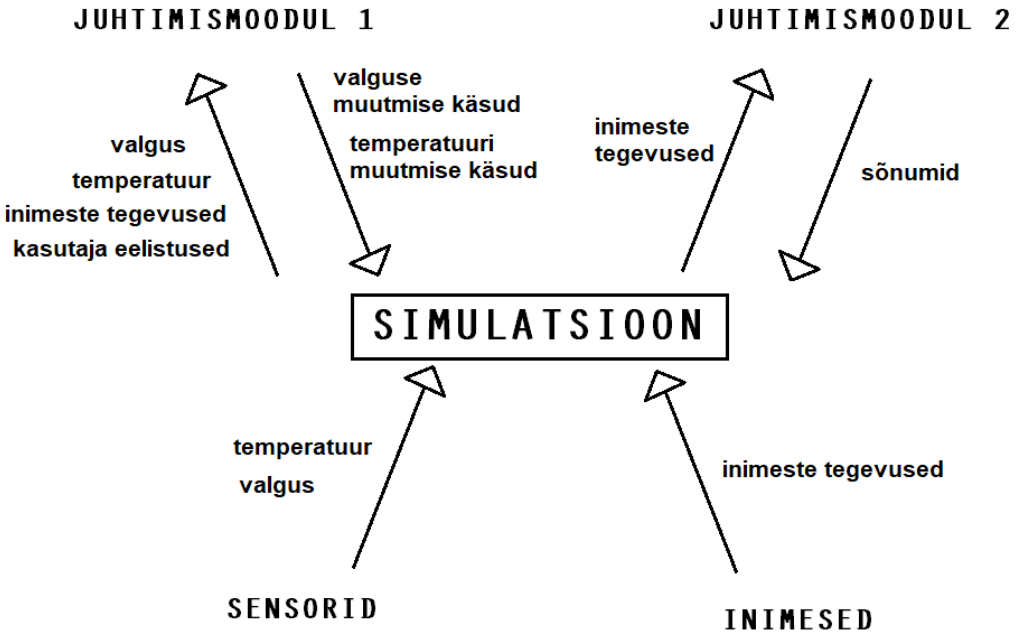
1.4 Lahendusmetoodika

Lõputöö käigus luuakse 3D mudel kuuelamust, kasutades nii lõputöö autori tehtud kui ka internetis tasuta saada olevaid mudeleid. Simulatsiooni jaoks luuakse iga ruumi kohta temperatuuri ning valguse muutujad, mille väärtusi on võimalik koodis seadistada. Samuti lisatakse ruumidesse valgusobjektid, mille intensiivsuse väärtused on seotud antud ruumi üldise valguse seadistusega. Simulatsiooni lisatakse tegelased, kelle igapäevaseid tegevusstsenaariume juhib reeglipõhine otsustusmootor.

Lisaks simulatsioonile luuakse kasutajaliides, mille abil kasutaja saab tagasisidet programmi töö kohta ning kus on võimalik seadistada eelistusi. Kasutajaliideses on näha reaalajas muutuvate sensorite väärtuseid ning sõnumeid, mis kirjeldavad simulatsiooni tööd. Samuti on olemas sisendväljad, mille abil on võimalik sisestada valguse, temperatuuri ning energia eelistusi, mida juhtprogramm võtab edaspidi otsuseid tehes arvesse.

Lepitakse kokku API struktuur, mille abil simulatsioon ja juhtimistarkvara suhtlevad, ning luuakse funktsioonid, mille abil suhtlus toimub. Simulatsioon peab oskama antud käsklusele vastavalt käituda, näiteks hakata temperatuuri väärtust järkjärgult muutma või ohu korral väljastama hoiatussõnumi.

Projekti komponente ning nende vahelist suhtlust illustreerib Joonis 1.



Joonis 1. Andmevoo diagramm.

Simulatsiooni ehitamiseks kasutatakse mängumootorit Unreal Engine 4 ning modelleerimistarkvara Blender. Programmidevaheline suhtlus toimub läbi TXT-failide.

2 Töös kasutatavate tehnoloogiate kirjeldus

2.1 Simuleerimise põhimõtted

„Simulatsiooni loomine on protsess, mille käigus luuakse mudel reaalsest või väljamõeldud süsteemist ning tehakse selle mudeliga eksperimente.“ [7] Simulatsioon on lihtsustatud mudel mingist süsteemist, mille eesmärk on süsteemi tööd visualiseerida ja võimaldada seda paremini mõista. Simulatsiooni abil saab läbi proovida süsteemi erinevaid lahendusi, et oleks võimalik teha läbimõelduid ja otstarbekaid disainiotsuseid ning ära hoida kulukaid vigu.

Päriselu süsteemid on enamasti komplekssed, seetõttu on simulatsiooni tegemisel vaja otsustada mis osa simulatsioonist peab olema täpne ning millise osa puhul on võimalik teha lihtsustusi, mõjutamata simulatsiooni kasulikkust. Valiku tegemine oleneb eelkõige simulatsiooni eesmärgist.

Järgnevalt on välja toodud mõned simulatsiooni tegemise põhimõtted:

- Simulatsioon koosneb olekutest, sündmustest ning objektidest. Olekud kirjeldavad süsteemi seisundit, ehk antud projektis on olekut kirjeldavateks muutujateks temperatuur, valguse intensiivsus, inimese asukoht. Sündmused väljenduvad süsteemi oleku muutuses. Sündmuste näiteks on juhtimisotsuste väljastamine, kasutajapoolse sisendi andmine simulatsioonile jms. Objektid on olemid, mida olekumuutujad kirjeldavad ning mille oleku muutusi sisendsündmused mõjutavad ja väljundsündmused kirjeldavad. [7]

- Aega käsitletakse muutujana, mille muutumiskiirus ei pea üldse sõltuma reaalajast. Näiteks võib simulatsioon hüpata otse ühe sündmuse juurest teiseni või siis hüpata üle kindlatest ajavahemikest. Tehnika valik sõltub sellest, mida on antud projektis oluline esitada ja jälgida. [7]

- Statistilise variatsiooni simuleerimiseks kasutavad paljud mudelid juhuslike arvude genereerimist, kuna päriselus ei saa oodata, et sündmused kulgeksid täpselt deterministliku olekumasina otsuste kohaselt. [7]

- Inimeste käitumise simuleerimiseks kasutatakse paljudes simulatsioonimudelites tehisintellekti. Mudelid võivad sisaldada lõplikke olekumasinaid, närvivõrke, juhtumipõhiseid arutluskäike. Tehisintellekti teostus oleneb sellest, kui realistlikult ning inimesesarnaselt see peab käituma ja mis võimalusi mudeli loomiseks kasutatakse – süvaõppe algoritme, mudeli ekstraheerimist koodist või käsitsi konstrueerimist. [7]

2.2 Simulatsiooniprogrammi valik

Lõputöö autor kaalus simulatsiooni tegemiseks mitmeid erinevaid programme. Kõigepealt uuriti programme, mis on mõeldud spetsiifiliselt simuleerimiseks, näiteks Gazebo [8], FlexSim [9] ja Webots [10]. Autor leidis, et enamus tuntumatest simulatsiooniprogammidest on loodud kindla eesmärgiga, näiteks tööstusrobotite simuleerimine, vabrikukeskondade simuleerimine jne, ning nende puhul ei saa kindel olla, et on olemas kõik antud projekti tegemiseks vajalikud vahendid. Samuti ei ole paljud programmid laialt levinud ning on seega ka vähem dokumenteeritud.

Seetõttu otsustas autor valida simulatsiooni tegemiseks mängumootori. Mängumootorid on väga versatiilsed ning nende populaarsuse tõttu on nende kasutamise kohta internetis arvukalt õpetusi ning artikleid. Mängumootori põhilisteks valikuteks otsustusid Unreal Engine 4 [11] ning Unity [12]. Mõlemad mootorid on piisavalt võimekad lõputöös nõutud simulatsiooni loomiseks, seetõttu oli lõplik otsus pigem subjektiivne. Autor oli enne Unity’t kasutanud ning eelistas õppida midagi uut. Unreal Engine'is on olemas *blueprintide* funktsionaalsus, ehk programmi õppimine ning töötava prototüübi tegemine on mõnevõrra lihtsam. Seetõttu saab autor keskenduda simulatsiooni ehitamisele. Lõplik otsus langes Unreal Engine 4 kasuks.

2.3 Unreal Engine 4

Unreal Engine 4 (UE4) on Epic Games Inc. poolt arendatud mängumootor, mis on loodud selleks, et lihtsustada arvutimängude arendamise protsessi. Unreal Engine 4 on

üks laialdasemalt kasutatavaid mängumootoreid ning on väga võimekas ja mitmekülgne, nii et seda on võimalik kasutada ka muuks kui ainult mängude arendamiseks.

UE4-s on olemas kõik vajalikud tööriistad tervikliku projekti tegemiseks. Üks väheseid põhjuseid, milleks võib mängu tegemise käigus mõnda teist programmi vaja minna, on see, kui on vaja teha mängu jaoks mudeleid või tekstuure. UE4-s saab luua mängutasemeid ning siduda neid kokku ühtseks terviklikuks mänguks. Mängumaailma saab luua kasutaja soovi järgi, lisada sellele kasutajaliidese elemente ning menüüsid ja luua reeglistik, mille põhjal mänguloogika töötab. Samuti on võimalik teha enda materjale, animatsioone, kontrollida tegelasi läbi tehisintellekti jne.

Koodi kirjutamiseks on võimalik kasutada nii programmeerimiskeelt C++ [13] kui ka UE4 enda *blueprintide* funktsionaalsust. *Blueprintid* on osa UE4 jaoks loodud visuaalsest skriptimiskeelest, kus uute funktsioonide tegemiseks järjestatakse funktsioonikastikesi (*blueprint node*'id). UE4 kompileerib kõik *blueprintid* C++-i koodiks. Autor otsustas kasutada just *blueprinte,* kuna nendega on efektiivne prototüüpida ning see on mõnevõrra algajasõbralikum variant mängumootori kasutama õppimiseks.

Antud lõputöös kasutati Unreal Engine 4 versiooni 4.21.

2.4 Blenderi tutvustus

Blender on tasuta avatud lähtekoodiga 3D graafikaprogramm, mis võimaldab luua 3D mudeleid ning neid eksportida sobivasse failiformaati. Mudeli tegemiseks on olemas palju erinevaid baasobjekte, nagu kerad, silindrid, kuubid, mida saab muuta vastavalt soovile. Mudelitel on tipud ning servad, mida saab lisada, eemaldada ning ümber paigutada. Samuti on olemas palju erinevaid tööriistu, mis võimaldavad kiiremini ja efektiivsemalt modelleerida, näiteks peegeldamine, lõikamine, silumine, trianguleerimine jm.

Selleks, et loodud mudelit UE4-s kasutada, on mõistlik teha mudelile ka UV kaart, et UE4 teaks, kuidas objekti korrektselt valgustada. UV kaardi tegemiseks peab 3D mudeli niimoodi lahti pakkima, et kõik selle pinnad saab mahutada 2D tasapinnale. Niimoodi on lihtne teha ka objektile tekstuure, mida saab hiljem UE4-s materjalidena kasutada.

Mudeli importimisel on mõistlik lasta UE4-l ka valguskaardid genereerida, kuna siis on kõige tõenäolisem, et objekti valgustatakse korrektselt. Kui mudel on imporditud, saab sellele määrata materjale ning kasutada seda edaspidi oma programmis.

2.5 Suhtlus juhtimistarkvaraga

Suhtlus simulatsiooni ning juhtimistarkvara vahel toimub läbi TXT-failide. Simulatsiooni töötamise ajal kontrollitakse pidevalt sisendfaile ning uuendatakse andmete seisu väljundfailides. Sensorite andmed ning info simulatsiooni tegelaste tegevuste kohta hoitakse failides pidevalt värskena, nii et alati on võimalik näha iga muutuja hetkeseisu.

Kui kasutaja muudab oma eelistusi või kui juhtprogrammil on simulatsioonile käsk, siis saadetakse see sõnumina läbi faili. Üks programm kirjutab faili käsu, teine loeb selle ning kustutab failist loetud käsu. Uut sõnumit ei tohi saata enne kui fail on jälle tühi, et vältida faili ülekirjutamist ja kirjutamise konflikte.

3 Simulatsiooni loomine

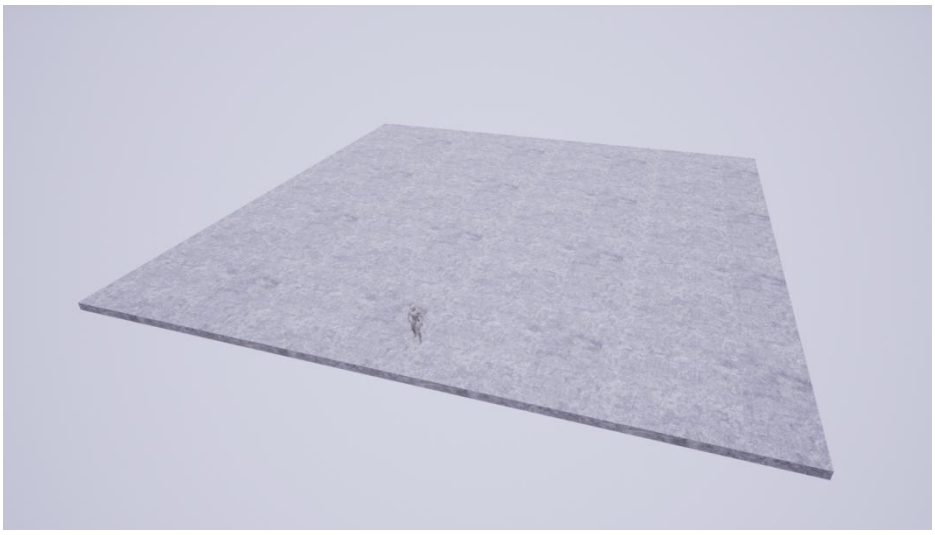
Simulatsiooni loomiseks pidi autor ära õppima kahe keerulise ning järsu õppekõveraga programmi kasutamise. Selle saavutamiseks kasutas autor erinevaid juhendeid ning artikleid ning katsetas läbi programmide võimalusi, enne kui sai hakata simulatsiooni ennast looma. Suureks abiks oli Tommy Trani juhendite sari algajatele [14], mis seletas põhjalikult ära Unreal Engine 4 põhilised funktsionaalsused ning kuidas neid rakendada, et luua terviklik projekt.

3.1 Mängutaseme loomine

Projekti luues pakub UE4 kasutamiseks erinevaid malle, millega tulevad kaasa projektitüübile vastavad funktsionaalsused ning seadistused. Näiteks on *Side Scrolleris* kaamera ühel küljel ja simulatsiooni tegelased liiguvad kahes dimensioonis ning VR mallid võimaldavad luua rakendusi, mida saab juhtida virtuaalreaalsusseadmetega.

Simulatsiooni projekti tehes valis autor mallitüübiks *Top Down*, kuna eesmärgiks oli luua pealtvaates rakendus, kus kasutajal on alati üldine ülevaade kogu majast ning selles toimuvast. Malliga tuli kaasa esialgne projekti struktuur ning juhitav tegelane. Tegelase külge oli ühendatud pealtvaates kaamera ning tema liikumist oli võimalik hiirega juhtida, mis oli väga kasulik selleks, et katsetada kuidas tehisintellekti poolt juhitud tegelased hakkavad iglus ringi liikuma.

Iglu ehitamise jaoks loodi uus *level* ehk mängutase. Mängutaseme algses versioonis on olemas hõljuv tükk maapinda ning objektid, mis seadistavad maailma valgustuse ning taeva väljanägemise. Kuuelamu mahutamiseks laiendas autor maapinda. Maapinna ümber lisati *Lightmass Importance Volume*, mille abil mängumootor teab, millistes alades peab valgus välja nägema detailne ning realistlik. Samuti lisati *Nav Mesh Bounds Volume*, mis defineerib ala, kus tegelastel on võimalik liikuda. Viimasena lisati maailmale inimese kuju, mis määrab iglu simulatsiooni ehitamisel ruumide mõõtkava. Selleks kasutati UE4 valmis objekti „mannekeen“. Tühja mängutased on näha Joonisel 2.

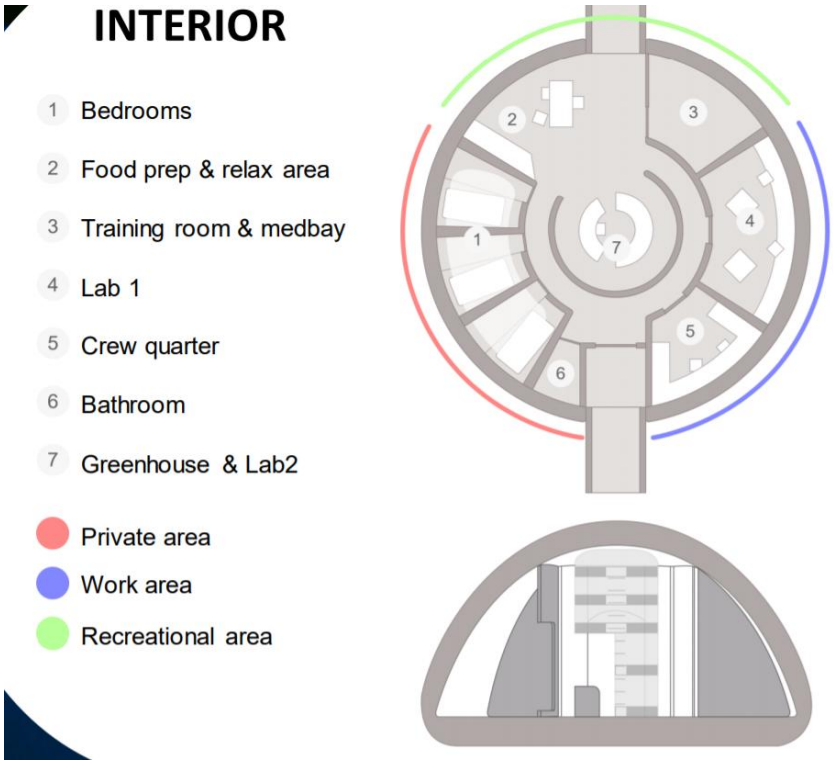


Joonis 2. Tühi mängutase.

Iglu valgustussüsteemi tegemise ajal eemaldati maailmast päike ning lisati ümbeerringi tähistaevas. Iglu ruumidesse lisati üldist valgust, et ka pimedas oleks võimalik objekte ning tegelasi eristada.

3.2 Iglu ehitamine

Iglu sisearhitektuur on inspireeritud ühe teise IGLUNA projekti tiimi poolt loodud skeemist (Joonis 3), mis määrab maja kuju ning ruumide jaotuse.

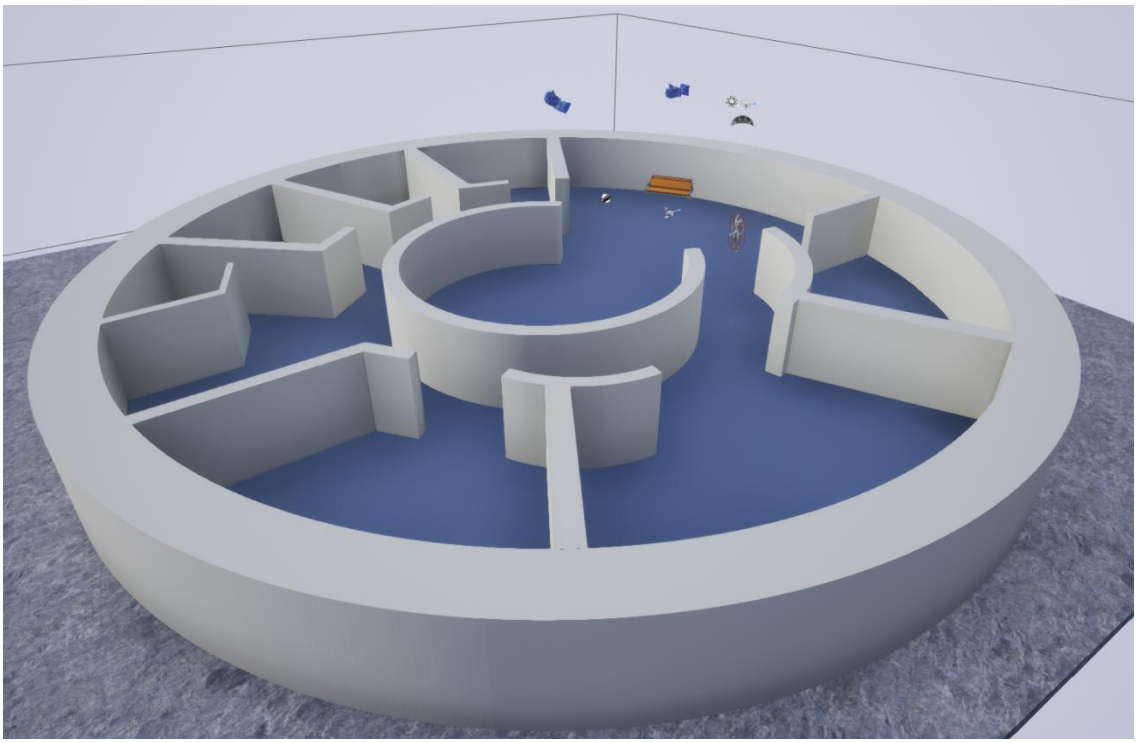


Joonis 3. Iglu struktuuri skeem. (Moony Schematic by Irene Zaccara 2018)

Maja üldise struktuuri ehitamisel kasutas autor modelleerimistarkvara Blender. Ümarate seinade tegemiseks tehti Blenderis lai ning seest tühi silinder, mida kasutati ümmarguse välisseina jaoks. Sealjuures oli oluline teha mudelile korrektne UV kaart, mille abil UE4 oskab objektile paistvat valgust arvutada. Mudel salvestati FBX formaadis ning imporditi, lastes UE4-l importimisel genereerida mudelile valguskaardid.

Iglu sisemiste seinade jaoks kasutati välisseina mudelist lõigatud fragmente ning UE4-s juba olemasolevaid sirgete seinade mudeleid. Põrandad tehti samuti laiadest ning lapikutest silindritest. Mudelite mängutasemesse lisamisel tuli kontrollida, et oleks kasutusel loogilise kujuga *collider*, ehk kujund, mis määrab ära objekti kokkupõrkeala. Näiteks kui ümmarguse välisseina puhul kasutada *simple collisionit*, siis ümbritseb lihtne *collider* tervet objekti ning tegelased ei saa selle sees liikuda. Kui kasutada *complex collisionit*, siis võetakse arvesse objekti õõnes kuju ning kokkupõrkealaks määratakse vaid sein ise.

Iglu struktuuri on näha Joonisel 4.

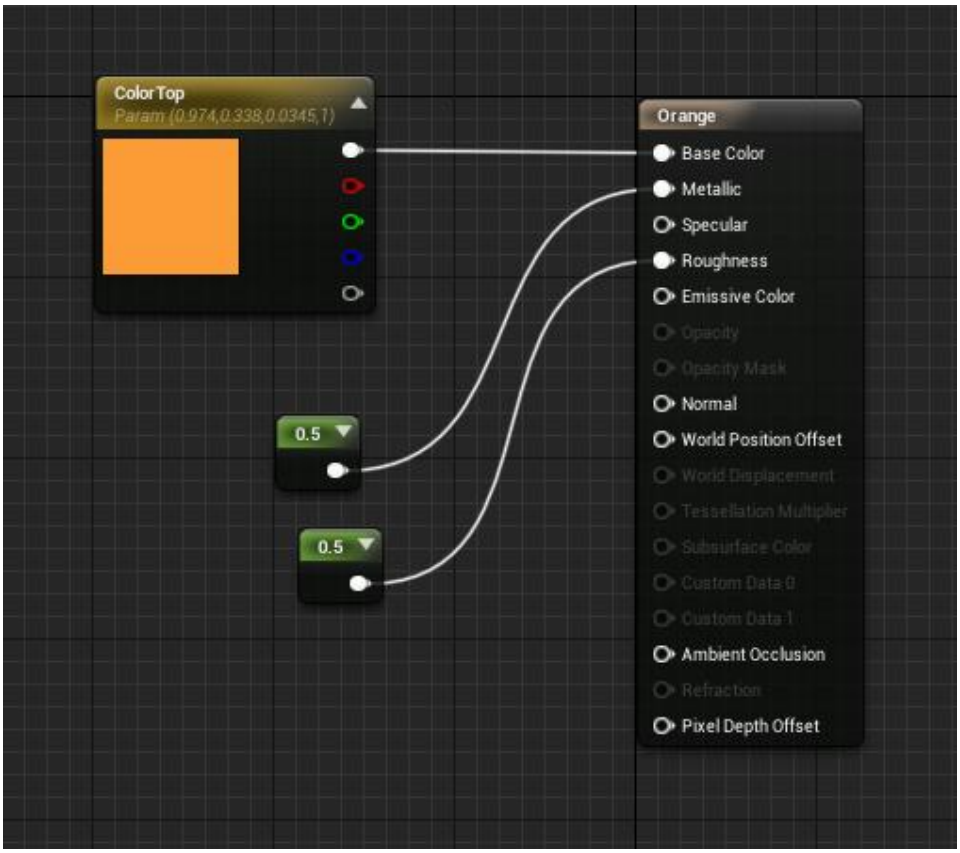


Joonis 4. Iglu struktuur.

3.3 Ruumide sisustamine

Ruumide sisustamisel leidis autor palju abi erinevatelt internetilehekülgedelt, kus inimesed pakkuvad enda tehtud tasulisi ja tasuta 3D mudeleid. Kaheks põhiliseks allikaks olid veebilehed CGTrader [15] ning TurboSquid [16].

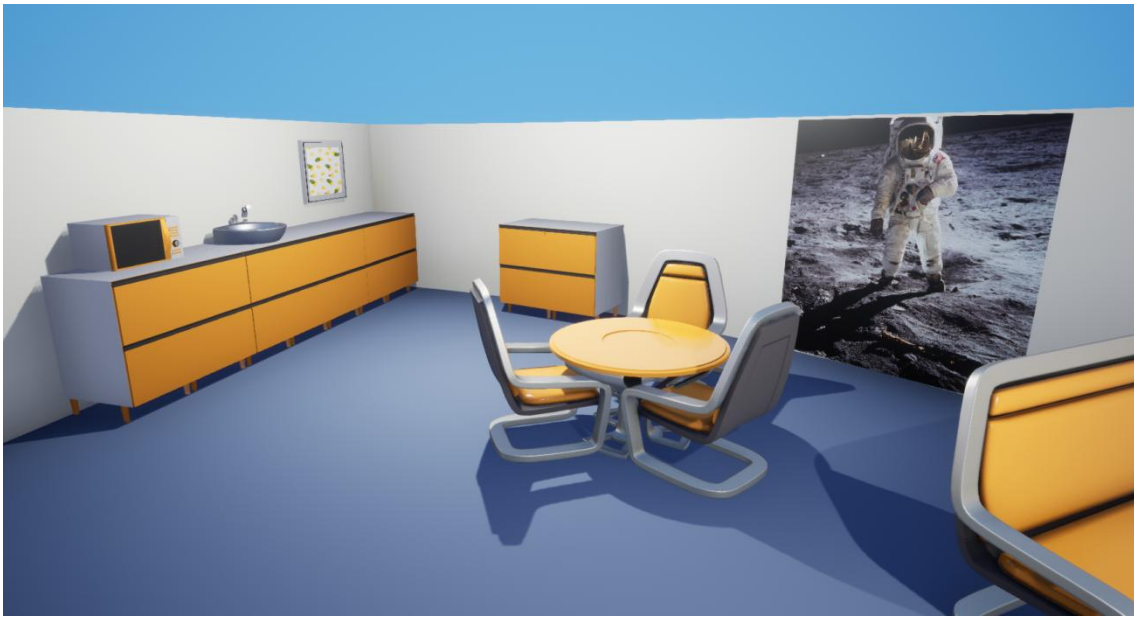
Sisustuse põhilisteks värvideks valis autor valge, oranži ja musta ning metalsete pindade jaoks enamasti kroomi. Valge ning kroomi jaoks kasutati UE4-s olemasolevaid materjale. Oranži ning musta jaoks tegi autor ise materjalid, kus seadistas ära materjali värvi ning *metallic* (materjali metalsus) ja *roughness* (materjali läikivus) väärtused. Oranži värvi materjali *blueprint* on näha Joonisel 5.



Joonis 5. Oranži materjali *blueprint.*

Sisustus on pigem minimalistlik, kuid põhiline eluks vajalik on olemas, ning on üritatud lisada ka natuke nüansse, näiteks erinevaid seinakaunistusi. Joonistel 6-12 on näha iglu erinevaid ruume.

Joonis 6. Magamistuba.



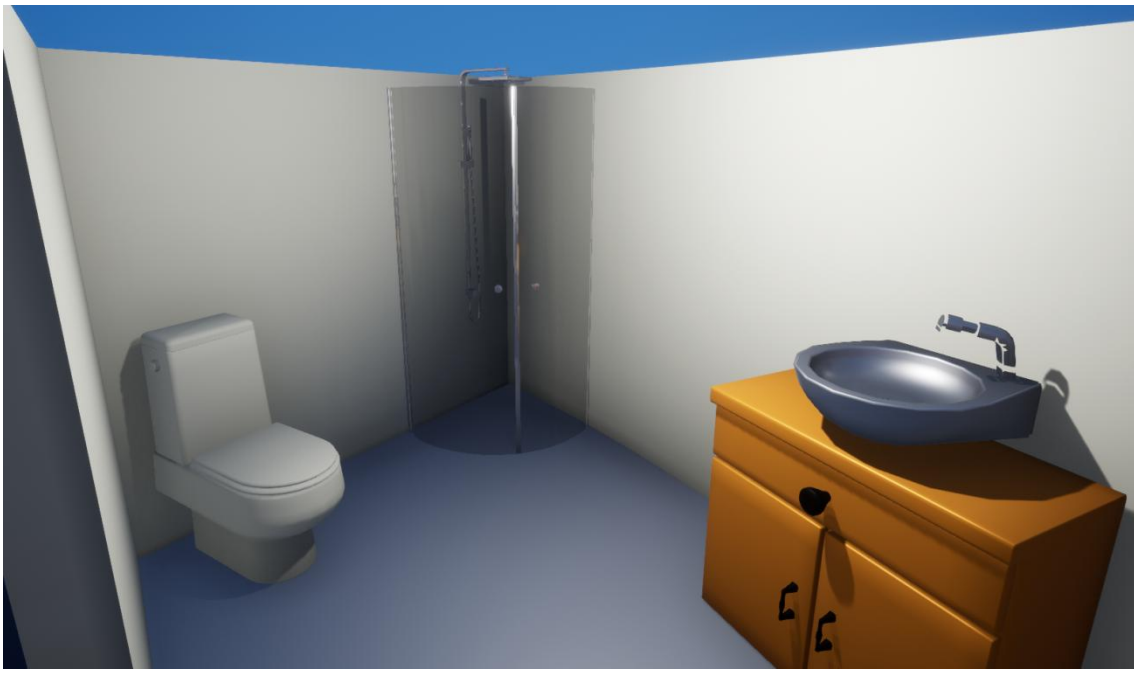
Joonis 7. Köögiala.

Joonis 8. Puhkeala.

Joonis 9. Laboratoorium-kasvuhoone iglu keskel.

Joonis 10. Tööruum.

Joonis 11. Treeningsaal.

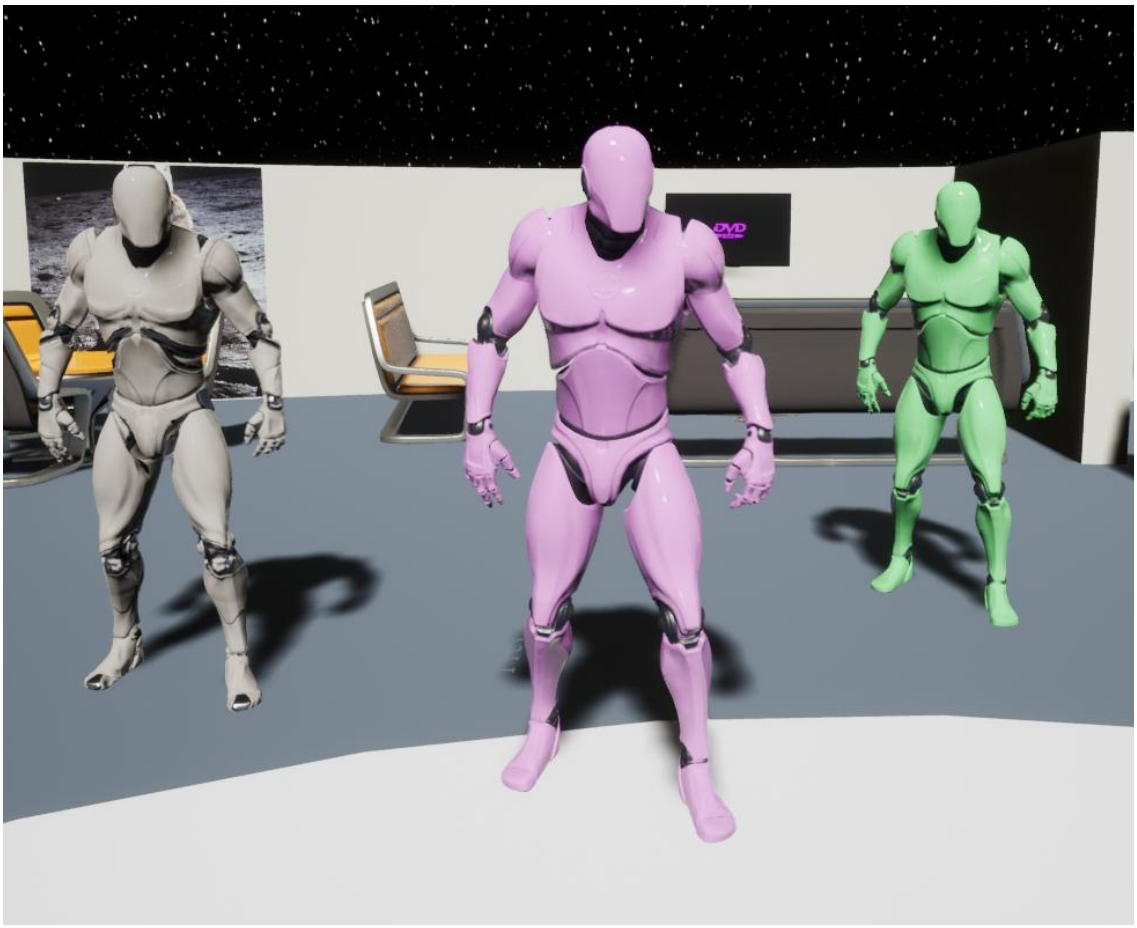


Joonis 12. Vannituba.

3.4 Inimese mudel

Iglu elanike kujutatamiseks kasutatakse UE4 mannekeene, kes on simulatsioonis värvitud erinevat värvi. Mannekeene otsustati kasutada sellepärast, et neil on olemas töötav *skeletal mesh*, ehk skelett, mida saab kasutada animatsioonide tegemiseks, nii et tegelastele saab lihtsasti erinevate tegevuste illustreerimiseks animatsioone juurde lisada.

Tegelaste eristamiseks on mannekeenide värvid hall, roosa ja roheline. Tegelased (Joonis 13) said tiimi liikmete järgi nimedeks Britta, Merlin ja Anna.

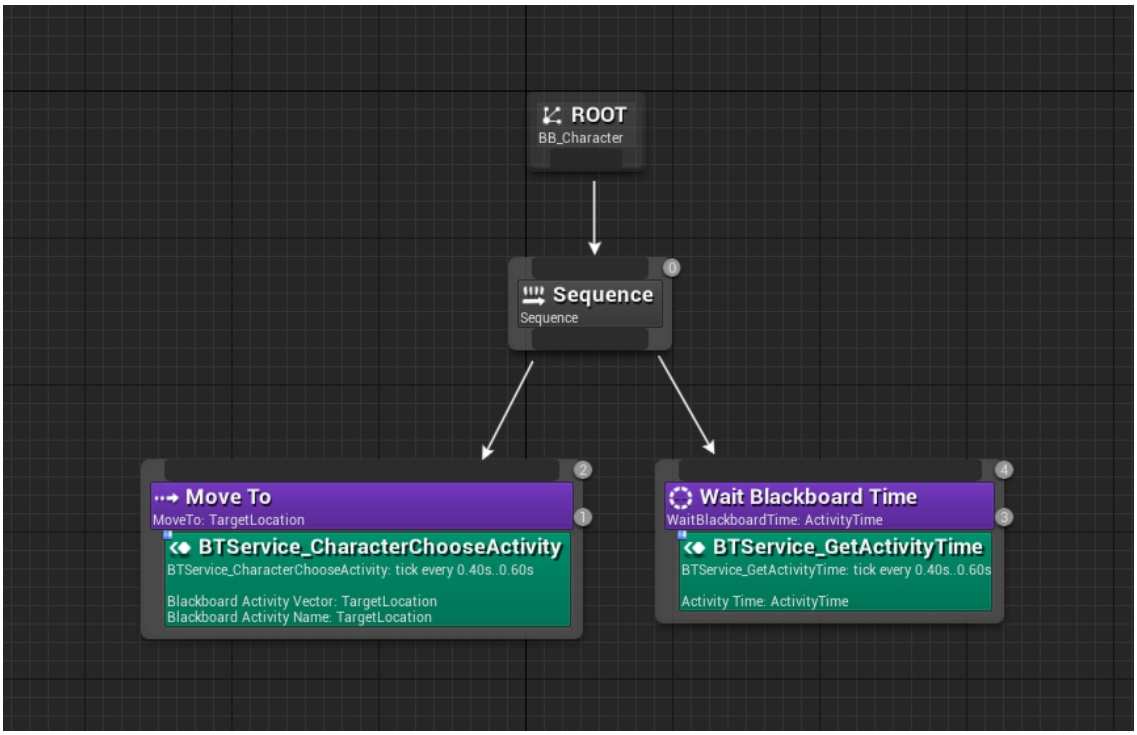


Joonis 13. Iglu elanikud.

3.5 Inimeste tegevused

Iglu elanike puhul on võetud kasutusele UE4 tehisintellekti programmeerimise võimalused. Tegelasi juhivad *AI controllerid,* mis kasutavad otsuste tegemiseks käitumispuud, mis on näha Joonisel 14. Antud hetkel on käitumispuu väga lihtne. Võimalikud tegevused on magamine, söömine, puhkamine, treenimine ning töötamine. Igal tegelasel on iga tegevuse jaoks oma eelistatud koht. Tegelane valib oma tegevuse viie põhitegevuse hulgast ning kõnnib selle alustamiseks eelmääratud kohta. Seejärel ühe kuni viie tunni pärast võtab vastu otsuse uue tegevuse kohta.

Käitumispuud on võimalik keerukamaks teha, näiteks kasutada kindlat päevaplaani või võtta arvesse kellaaega, teiste inimeste asukohta, jne. Tehisintellekti täpne implementatsioon sõltub projekti vajadustest ning seda on võimalik täiendavalt muuta. Võimalik edasiarendus oleks ka kasutada sisendina ühe teise IGLUNA projekti poolt loodud päevakava haldamise süsteemi, millega saaks määrata igale tegelasele tema igapäevased tegevused.



Joonis 14. Tegelase käitumispuu.

UE4 mannekeenidega on automaatselt kaasas ning seadistatud kõndimise, jooksmise ning paigal seismise animatsioonid. Tegelase liikumisel kasutatakse liikumiskiirusele vastavat animatsiooni. Lõputöö autoril on plaanis tulevikus projekti lisada ka iglu elanike tegevustele vastavad animatsioonid. Vajalikud animatsioonid on näiteks istumine, pikali heitmine, söömine ja töötamine.

3.6 API programmidevaheliseks suhtluseks

Suhtlus juhtprogrammidega käib läbi TXT-failide. Simulatsioon kirjutab pidevalt hetkeseisu ning kasutajate eelistuste muutusi väljundfailidesse ning loeb uusi käske sisendfailidest. Andmevahetust illustreeriv skeem on näha Joonisel 1.

Failide lugemine ja kirjutamine on implementeeritud C++ koodis. Loodud funktsioonid kasutavad UE4 enda FFileHelper moodulit, et faile lugeda ning neisse kirjutada. Funktsioon LoadTxt võtab sisendina faili nime ja tagastab selle faili sisu, ning funktsioon SaveTxt võtab sisenditena faili nime ning salvestatava teksti ning kirjutab teksti antud faili. C++ koodi kompileerides tekivad nendele funktsioonidele samuti *blueprint node*'id, mida on edaspidi võimalik *blueprintides* kasutada.

Simulatsioon kasutab failidesse kirjutamisel Prologi süntaksit, et juhtimistarkvara saaks informatsiooni Horni reeglitena kasutada. Juhtimistarkvara poolt saadetud käsud on üherealised komaga eraldatud väärtuste jadad. Igale sõnumile lisatakse sõnumi kuupäev ning kellaaeg.

Kasutusel on kuus väljundfaili. Kolm nendest kajastavad hetkeseisu ning neis on iga toa või inimese jaoks eraldi rida, mida pidevalt uuendatakse värske infoga. Failinimed algavad prefiksiga „current_“ Nendes on kirjas vastavalt kas kindla toa valguse intensiivsus või temperatuur või kindla inimese hetke asukoht ning tegevus.

Näited:

- current_lights – current_light(bed1, 50, 2019, 05, 15, 23, 42).
- current_temperatures – current_temperature(storage, 23.1, 2019, 05, 15, 23, 42).
- current_activities – current_activity(lab1, merlin, working, 2019, 05, 15, 23, 42).

Ülejäänud kolme väljundfaili eesmärk on anda juhtimistarkvarale teada kasutaja eelistuste muutustest (eelistuste muutmine on täpsemalt kirjeldatud alapeatükis 3.8). Failinimed algavad prefiksiga „user_“. Kui kasutaja eelistust muudetakse, siis simulatsioon avab vastava faili ning kui see on tühi, kirjutab sinna uue teate. Kui see ei ole tühi, siis simulatsioon ootab, kuni juhtimistarkvara loeb failist sõnumi või kuni määratud ooteaeg saab otsa.

Näited:

- user_light – user_light(bed2, 100, 2019, 5, 15, 6, 32).
- user_temperature – user_temperature(relax, 24, 2019, 5, 15, 6, 32).
- user_energy – available_energy(5, 2019, 5, 15, 5, 0).

Hetkel on kasutusel neli sisendfaili simulatsiooni töö juhtimiseks. Kaks neist on temperatuuri ning valguse muutmiseks ning nende failide nimed algavad prefiksiga „tasks_“. Simulatsioon kontrollib neid faile pidevalt ning kui failis on uus ülesanne, siis käitub programm vastavalt käsule ning seejärel teeb faili tühjaks. Sisend koosneb komadega eraldatud väärtustest.

Näited:

- tasks_light – bed1,20,2019.5.19-11.49.44.395102024.
- tasks_power – training,5,2019.5.19-11.49.44.395102024.

Teised kaks sisendfaili on sõnumite kuvamiseks. Failide nimed algavad prefiksiga „log_“. Faili *log_message.txt* kirjutatud sõnumid lisab simulaator ekraanile sõnumite logisse. Faili *log_warning.txt* sõnumid lähevad samuti sõnumite logisse, aga kuna ohu korral on vaja sellest teavitada kõiki majaelanikke kuvatakse nei ka viieks sekundiks punaselt ekraani keskel. Pärast faili sisu lugemist kustutatakse faili sisu.

Näited:

- log_message – Anna: Your blood sugar is low, please eat something sweet.
- log_warning – Merlin’s body temperature is dangerously high, she needs help!

3.7 Mänguloogika

Simulatsiooni südameks on *Game Instance* tüüpi klass *IgluGameInstance*. *Game Instance* klass luuakse mängu alguses, see kestab läbi erinevate mängutasemete ning on globaalse ligipääsuga. *IgluGameInstance*’is hoitakse simulatsioonis vajaminevaid muutujaid ning funktsioone, mis ei ole seotud mingi kindla *Actor* objektiga mängutasemes.

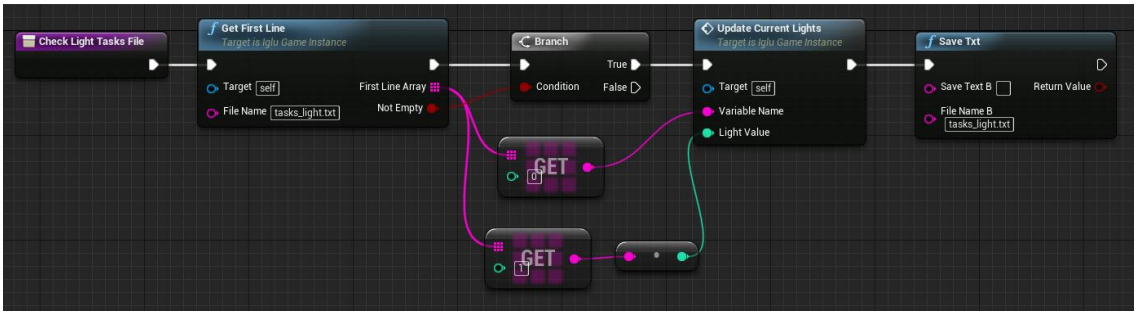
Iglus on 9 erinevat ruumi ning neile on muutujates kasutamiseks määratud lühikesed inglisekeelsed koodnimed. Magamistoad on *bed1*, *bed2* ja *bed3*, puhkeala on *relax*, treeningsaal on *training*, töötoad on *lab1* ja *lab2*, laoruum on *storage* ja vannituba on *bathroom*.

Muutujad: aeg, energia, iga ruumi temperatuur, iga ruumi *power* ehk kütmisvõimsus, iga ruumi valguse intensiivsus. Valguse ja temperatuuri kasutajaeelistused, mida on vaja eraldi salvestada selleks, et neid kasutajaliideses kuvada. Samuti on muutujatena salvestatud iga toa ning inimese reaindeks, mida kasutatakse *current* failide uuendamiseks, kuna failide läbi suhtlus on kokku lepitud ning igale toale/inimesele

vastab alati sama rida. See välistab vajaduse faili iga kord uuesti parsida, et õige rida üles leida.

Funktsioonid: valguse väärtuse uuendamine, energia väärtuse uuendamine, sisendfailide kontrollimine, failide initsialiseerimine ning uuendamine, abifunktsioonid failidesse kirjutamise ning kasutajaliideses asuvasse sõnumilogisse kirjutamise jaoks.

Joonisel 15 on funktsiooni näitena toodud funktsioon *CheckLightTasks*. Funktsioon loeb faili *tasks_light.txt* ning kutsub välja funktsiooni *GetFirstLine*, mis võtab failit esimese tekstirea ja teeb selle järjendiks, kasutades eraldajana koma. Kui rida ei olnud tühi, siis uuendatakse valguste väärtusi kasutades funktsiooni *UpdateCurrentLights*. Lõpuks salvestatakse fail *tasks_light.txt* üle, andes sisendiks tühja stringi.



Joonis 15. Funktsioon *CheckLightTasks*.

IgluGameInstance'is on olemas ka *Event Graph*, kus on võimalik kodeerida ajast sõltuvaid ülesandeid. Graafis on sündmus *Init*, mis aktiveeritakse programmi käivitudes ning mis toimib simulatsiooni sisendpunktina. Programmi käivitudes initsialiseeritakse *datetime* tüüpi *time* muutuja, kus hoitakse jooksvat kellaaega ning kuupäeva. Kuupäevaks valitakse jooksva päeva kuupäev ning kellaajaks kell 05:00. Siis käivitub taimer, mis iga sekund suurendab aja väärtust, võttes arvesse muutuja *TimeModifier* väärtust. Simulatsiooni aeg on seadistatud nii, et üks päriselu sekund on simulatsioonis võrdeline 4 minutiga, ehk ööpäeva pikkuseks on 6 minutit.

Pärast seda initsialiseeritakse tegevuste, temperatuuride, valgusintensiivsuste ning energia väärtused ning väljundfailid, arvestades muutujate vaikimisi väärtusi. Tulede objektid on programmi käivitumisel kõik intensiivsusega 0 ning neile määratakse siis intensiivsus, mis võtab arvesse vaikimisi intensiivsuse väärtuse ning igale ruumile seadistatud intensiivsuse maksimumväärtuse.

Lõpuks käivitatakse mitu taimerit, millest igaüks vastutab mingi mänguloogika osa eest. Kaks taimerit kontrollivad igas sekundis *tasks_power.txt* ning *tasks_light.txt* faile, et lugeda sealt juhtprogrammi sisendit. Kaks taimerit kontrollivad igas sekundis *log_message.txt* ning *log_warning.txt* faile. Viimane taimer uuendab igas sekundis temperatuuride väärtusi, olenevalt vastava ruumi atribuudi *power* ehk kütmisvõimsuse väärtusest. Kui võimsus on 0, hakkab temperatuur langema. Reaalselt sõltub maja jahtumine selle pindalast, emissiivsusest ning välistest soojusallikatest, kuid lihtsutamise eesmärgil vähendatakse simulatsioonis temperatuuri konstantselt 0.1°C võrra ajaühikus. Kui võimsus on suurem kui 0, siis kasutatakse temperatuuri muudu arvutamiseks järgnevat valemit

$$\Delta T = \frac{P\ x\ t}{C}$$

kus ΔT on temperatuuri muut (°C), P on võimsus (W), t on ajakoefitsient (s) ja C on õhu soojusmahtuvus, milleks on võetud konstantne väärtus 1006 (J/kgC).

Event Graphis on veel kaks sündmust, mis käivituvad siis, kui kasutaja muudab valguse või temperatuuri eelistuse väärtust. Sündmus uuendab vastava muutuja väärtust ning siis käivitab taimeri. Taimeri jooksmise ajal kontrollib simulatsioon iga sekund väljundfaili (*user_lights.txt* või *user_temperatures.txt*). Kui fail on tühi siis kirjutab simulatsioon sõnumi faili ning lõpetab taimeri töö. Taimer töötab maksimaalselt 20 sekundit, et vältida programmi lõpmatusse tsüklisse sattumist.

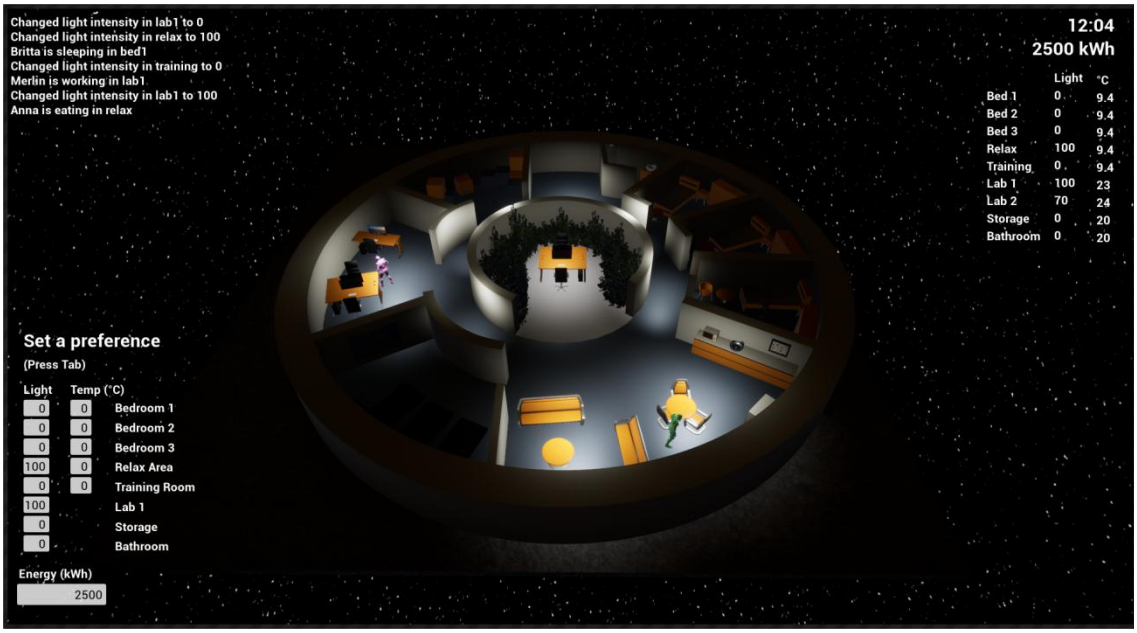
Peale *IgluGameInstance*'i on simulatsioonis iglu töö juhtimiseks veel eraldi *blueprint* nimega *LightingSystem*, mis loodi kõikide iglu valgusobjektide liitmisel üheks *blueprindiks*. Igal valgusobjektil on toanimega silt. *LightingSystemis* on funktsioon nimega *ChangeLightIntensity*, mis saab sisendiks toa nime ning intensiivuse. Funktsioon käib kõik tuleobjektid läbi ning muudab intensiivsust iga tule puhul, millel on vastava toa nimega silt.

3.8 Kasutaja sisend ja tagasiside

Simulatsiooni kasutajaliidest on näha Joonisel 15. Kasutajaliidese tegemiseks on UE4-s olemas vidinad, mille abil on võimalik erinevaid kasutajaliidese elemente disainida. Neid saab läbi koodi vaateaknasse lisada ning sealt eemaldada. Simulatsiooni

kasutajaliidese jaoks on loodud vidin *Overlay*, mis on terve ekraani suurun ja mis tehakse nähtavaks simulatsiooni pildi peal. *Overlay* ülemises parempoolses nurgas on näha simulatsiooni jooksvat väärtuste seisu, milleks on aeg, energia ning iga ruumi temperatuur ning valgus. Üleval vasakpoolses nurgas on iglu seisundi visualiseerimiseks sõnumite logi, kuhu kirjutatakse temperatuuri ja valguse muutused, kasutaja eelistuste muutused, kasutajate tegevused ning sõnumite sisendfailidest loetud sõnumid.

All vasakpoolses nurgas on kasutajaeelistuste muutmise ala. Eelistuste muutmiseks on mitu tekstiakent, mille sisu muutmine muudab vastava eelistuse väärtust. Võimalik on muuta energialimiidi väärtust ning osade ruumide valgust ja temperatuuri. Valgust on võimalik muuta igas ruumis peale keskmise laboratoorium-kasvuhoone, kuna seal hoitakse pidevalt valgust, mis on taimede kasvuks sobilik. Kui kasutaja muudab valguse väärtust, siis on seda kohe simulatsioonis näha. Samuti kirjutatakse vastav sõnum väljundfaili *user_light.txt*. Temperatuuri on võimalik muuta viies ruumis, konstantset temperatuuri hoitakse laboratooriumites, laoruumis ning vannitoas. Temperatuuri väärtuse muutmise puhul kirjutatakse muutus väljundfaili *user_temperature.txt*. Juhtimistarkvara arvestab seda ning annab teada, kuidas vastava ruumi kütmisvõimsust muuta.



Joonis 16. Simulatsiooni kasutajaliides.

3.9 Kasutaja tegelase modifitseerimine

Simulatsiooni tegemise alguses valiti projekti tüübiks *Top Down*, mille puhul tegelast sai juhtida klõpsates hiirega kohale, kuhu taheti tegelast suunata. Projekti lõpuosas eemaldati kasutaja poolt juhitud mannekeeniga tegelane ning liikumiseks valiti teine loogika, et anda kasutajale rohkem vabadust ning lubada tal maja ning inimesi lähemalt näha ja jälgida. Selle saavutamiseks kasutati UE4 *SpectatorPawn* objekti, mis on põhimõtteliselt nn "ringi lendav" kaamera, mida saab igas suunas liigutada kasutades hiirt ja W, A, S ja D klahve.

SpectatorPawni objektist tehti koopia, mis sai nime *TopDownCamera*. Sellele lisati loogika, mis laseb Tab klahvi vajutades kasutajal kasutajaliidesega suhelda, tehes nähtavaks hiirekursori ning seisatades liikumise. Uuesti Tabile vajutades taastub tavaline liikumine. *TopDownCamera* määrati vaikimisi mängija *controlleriks*.

4 Simulatsiooni valideerimine

Simulatsiooni ülesanne on valideerida ja visualiseerida iglu sisekliima juhtimistarkvara töötamist. Selle jaoks peab see simuleerima elukeskkonda koos selles elavate inimestega. Simulatsioon peab muutuste korral uuendama väljundfaile, et teha simulatsiooni olek juhtimistarkvarale nähtavaks, ning peab kontrollima pidevalt sisendfaile ja reageerima saadud käsklustele. Samuti peab simulatsioonis toimuv olema kasutajale arusaadav. Kui midagi muudetakse peab sellest andma kasutajale märku, kas visuaalselt (tulede muutmine) või kirjutades sõnumeid sõnumite logisse. See võimaldab kasutajal simulatsiooni tööprotsessi mõista ning osata juhtimistarkvara töötamist jälgida ning valideerida.

Simulatsiooni töö puhul on kõige olulisem teha kindlaks see, et toimiks õigesti tarkvarakomponentide suhtlus läbi jagatud failide ning et simulatsiooniprogramm reageeriks korrektselt sisendist saadud andmetele.

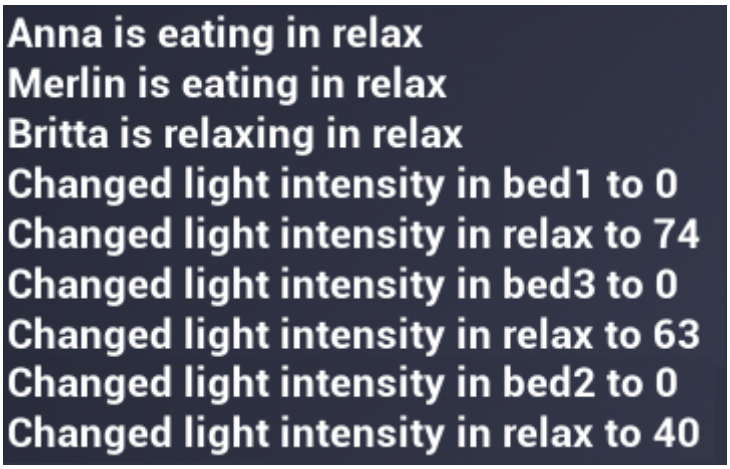
Programmi töö jooksvaks testimiseks sisestas autor sisendfailidesse käsklusi kogu andmete määramispiirkonna ulatuses. Samuti oli võimalik testida kasutajapoolsete eelistuste muutmist, kontrollides et eelistuse muutmise puhul kirjutatakse selle kohta sõnum õigesse väljundfaili. Hiljem, kui simulatsioon oli täielikult töökorras, sai katsetada simulatsiooni tööd koos valgustuse muutmise juhtprogrammiga.

UE4-s on võimalik projekti kompileerida paljudele erinevatele platvormidele. Demo tegemiseks kasutati võimalust teha projektist 64-bitise Windowsi peal töötav projekt, mis koosnes EXE rakendusfailist ning mängumootori poolt loodud projekti tööks vajalikest failidest.

Juhtimistarkvara sisendpunktiks on fail main.pl. Selle kasutamiseks seadistati kõigepealt failis vajaminevate failide rajad.

Simulatsiooni töö testimiseks pandi kõigepealt tööle simulatsiooni rakendus ning seejärel juhtimistarkvara. Demo ajal oli võimalik näha, kuidas inimeste tegevuste ning liikumise tagajärjel saatis juhtimistarkvara simulatsioonile käske tulede intensiivsuse

muutmiseks. Simulatsioon kontrollis pidevalt sisendfaile ning muutis valgust vastavalt seadesuurustele. Joonistel 16 ja 17 on näha mõningaid simulatsiooni ning juhtimistarkvara poolt demo jooksul logitud sõnumeid.



Joonis 17. Simulatsiooni sõnumite logi.

```
?- light.
---------------------------------
merlins last location was bed1 with light 100
WROTE: bed1,0,2019.5.20-19.6.24.701400995
Turned off light in bed1
New light at bed1 is 0
relax light is 0
merlins new location is relax
WROTE: relax,74,2019.5.20-19.6.24.726398944
New light at relax is 74
---------------------------------
---------------------------------
brittas last location was bed3 with light 100
WROTE: bed3,0,2019.5.20-19.6.25.734394073
Turned off light in bed3
New light at bed3 is 0
relax light is 74
brittas new location is relax
WROTE: relax,63,2019.5.20-19.6.26.74338293
New light at relax is 63
---------------------------------
---------------------------------
annas last location was bed2 with light 100
WROTE: bed2,0,2019.5.20-19.6.27.750401973
Turned off light in bed2
New light at bed2 is 0
relax light is 63
annas new location is relax
WROTE: relax,40,2019.5.20-19.6.28.76642704
New light at relax is 40
---------------------------------
---------------------------------
```

Joonis 18. Juhtimistarkvara logi.

5 Kokkuvõte

Lõputöö tulemusena valmis 3D simulatsioon kuupealsest elamust „iglu“, mida on võimalik kasutada selleks, et testida ja visualiseerida energiasäästliku reeglipõhise juhtimistarkvara tööd. Simulatsioonis on olemas kuuelamu mudel koos valgustussüsteemiga ning muutujatega, mis esindavad iglu erinevate ruumide temperatuuri ning valgusintensiivsuse väärtusi. Neid väärtusi on võimalik läbi koodi muuta. Samuti on olemas kolm tehisintellekti poolt juhitud inimtegelast ehk "mannekeeni", kes valivad oma tegevusi juhuslikus järjekorras. Tegevuste alustamiseks liiguvad mannekeenid tegevuseks ettenähtud ruumi. Iglu seisund ja mannekeenide tegevused omakorda mõjutavad sisekliima juhtimistarkvara otsuseid.

Töös on loodud funktsioonid, mille abil saab failidesse kirjutada ja neist lugeda ning on kokku lepitud API struktuur, läbi mille toimub simulatsiooni ning juhtprogrammide vaheline suhtlus.

5.1 Edasine töö

Valminud simulatsioon on mõeldud baasprojektina, mis saavutas töös püstitatud eemärgid, kuid mida on võimalik veel edasi arendada. Näiteks saaks simulatsiooni lisada veel jälgitavaid andmeid, et võimaldada juhtimistarkvaral arvestada lisafaktoreid ning teha informeeritumaid otsuseid. Samuti saaks muuta inimeste käitumispuud ja teha seda komplekssemaks või integreerida see hoopis IGLUNA teise tiimi poolt loodud päevakava pidamise tarkvaraga. Samuti võiks lisada võimaluse mängida simulatsioonis audioklippe.

Visuaalse poole pealt saaks simulatsiooni välimust veel täiendada, tehes juurde mudeleid ning täiendades sisustust. Kindlasti võiks lisada animatsioone, mis demonstreeriksid inimeste tegevusi ning teeksid simulatsiooni realistlikumaks. Samuti saaks lisada uusi tegevusi ning ka varieeruvust tegevuste sees.

Projekti autori esmane prioriteet edaspidiseks on simulatsiooni testida integratsioonis projekti teiste komponentidega. Koostöö meeskonnakaaslastega on vajalik selleks, et tagada simulatsiooni ootuspärane ja vigadeta töö IGLUNA projekti saabuvaks demoürituseks.

Kasutatud kirjandus

[1] IGLUNA - a habitat in space. [WWW] https://www.spacecenter.ch/igluna/ (20.05.2019)

[2] SWI-Prolog. [WWW]. http://www.swi-prolog.org/ (21.05.2019)

[3] Mars Simulation Project. [WWW] https://mars-sim.github.io (20.05.2019)

[4] HI-SEAS. [WWW] https://hi-seas.org/ (20.05.2019)

[5] CIROS. [WWW] https://www.festo-didactic.com/int-en/services/software/software-licences-trial-version/ciros.htm?fbid=aW50LmVuLjU1Ny4xNy4zMi44MjUuNzgyOQ (20.05.2019)

[6] The Sims Video Games - Official EA Site. [WWW] https://www.ea.com/games/the-sims (20.05.2019)

[7] R. D. Smith, Simulation: The Engine Behind the Virtual World, 1999. [WWW] http://www.simulationfirst.com/papers/sim2000/SimulationEngine.PDF (20.05.2019)

[8] Gazebo. [WWW] http://gazebosim.org/ (20.05.2019)

[9] Simulation software for manufacturing, material handling, healthcare, etc. - FlexSim Simulation Software. [WWW] https://www.flexsim.com (20.05.2019)

[10] Webots: robot simulator. [WWW] https://cyberbotics.com (20.05.2019)

[11] What is Unreal Engine 4. [WWW] https://www.unrealengine.com (20.05.2019)

[12] Unity. [WWW] https://unity.com (20.05.2019)

[13] cplusplus.com - The C++ Resources Network. [WWW] http://www.cplusplus.com/ (21.05.2019)

[14] T. Tran, Unreal Engine 4 Tutorial for Beginners: Getting Started, 2017. [WWW] https://www.raywenderlich.com/771-unreal-engine-4-tutorial-for-beginners-getting-started (20.05.2019)

[15] CGTrader - 3D Models for VR / AR and CG projects. [WWW] https://www.cgtrader.com (20.05.2019)

[16] 3D Models for Professionals :: TurboSquid. [WWW] https://www.turbosquid.com (20.05.2019)