TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Kaspar Lippmaa 144317

SIMULATSIOONIKESKKOND TARGA AUTOTEKI PROJEKTI JAOKS

Magistritöö

Juhendaja: Priit Ruberg
PhD

Tallinn 2020

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: Kaspar Lippmaa

04.05.2020

Annotatsioon

Töö eesmärk on luua simulatsioonikeskkond Targa Autoteki süsteemi jaoks. Loodav simulatsioonikeskkond võimaldab Targa Autoteki süsteemi testida ilma vajaduseta reaalsel laeval eksperiment korraldada.

Töö käigus arutletakse põgusalt modelleerimise ja simulatsioonide eesmärkidest üldisemalt, seejärel tehakse sissejuhatus Targa Autoteki projekti ja selle eesmärkidesse. Seejärel defineeritakse simulatsioonikeskkonna piirid Targa Autoteki projekti kontekstis. Simulatsiooni keskkonna ehitamiseks kasutati Unity platvormi. Töö käigus tutvustatakse nii Unity’t kui ka teisi valitud tööriistu ja kirjeldatakse tähtsamaid otsuseid simulatsioonikeskkonna ehitamisel. Lisaks võrreldakse simulatsiooni poolt genereeritud andmeid reaalsete andmete vastu. Viimaks testitakse ühte Targa Autoteki komponenti kasutades loodud simulatsioonikeskkonda.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 34 leheküljel, 8 peatükki, 20 joonist, 10 tabelit.

Abstract

Simulation environment for Smart Car Deck project

The goal of this work is to create a simulation environment for the Smart Deck project. The created simulation environment enables testing of the Smart Deck project without the need of conducting real experiments on the ship.

In this work the goals of modelling and simulation are discussed briefly, followed by an introduction into the Smart Car Deck project. The scope of the simulation environment in the context of the Smart Car Deck project is defined. The simulation environment was built using Unity. An introduction to Unity will be given along with other used tools and the most important decisions in the building of the simulation environment will be discussed. In addition, data generated by the simulation environment will be compared to real-world measurements. Lastly, a component of the Smart Car Deck will be tested using the created simulation environment.

The thesis is in Estonian and contains 34 pages of text, 8 chapters, 20 figures, 10 tables.

Lühendite ja mõistete sõnastik

Ahter	Laeva päraosa
API	*Application Programming Interface*, rakendusliides
ATI	TTÜ Arvutitehnika instituut
IOT	*Internet Of Things*, asjade internet
JSON	*JavaScript Object Notation*
LED	*Light Emitting Diode*, valgusdiood
LIDAR	*Light Detection And Ranging*,
Pakpoord	Laeva vasakpoolne parras
React	Javascriptil põhinev kasutajaliideste ehitamise raamistik
REST	*Representational State Transfer*
Tüürpoord	Laeva parempoolne parras
UI	*User Interface*, kasutajaliides
Vöör	Laeva esiots
XML	*Extensible Markup Language*

Sisukord

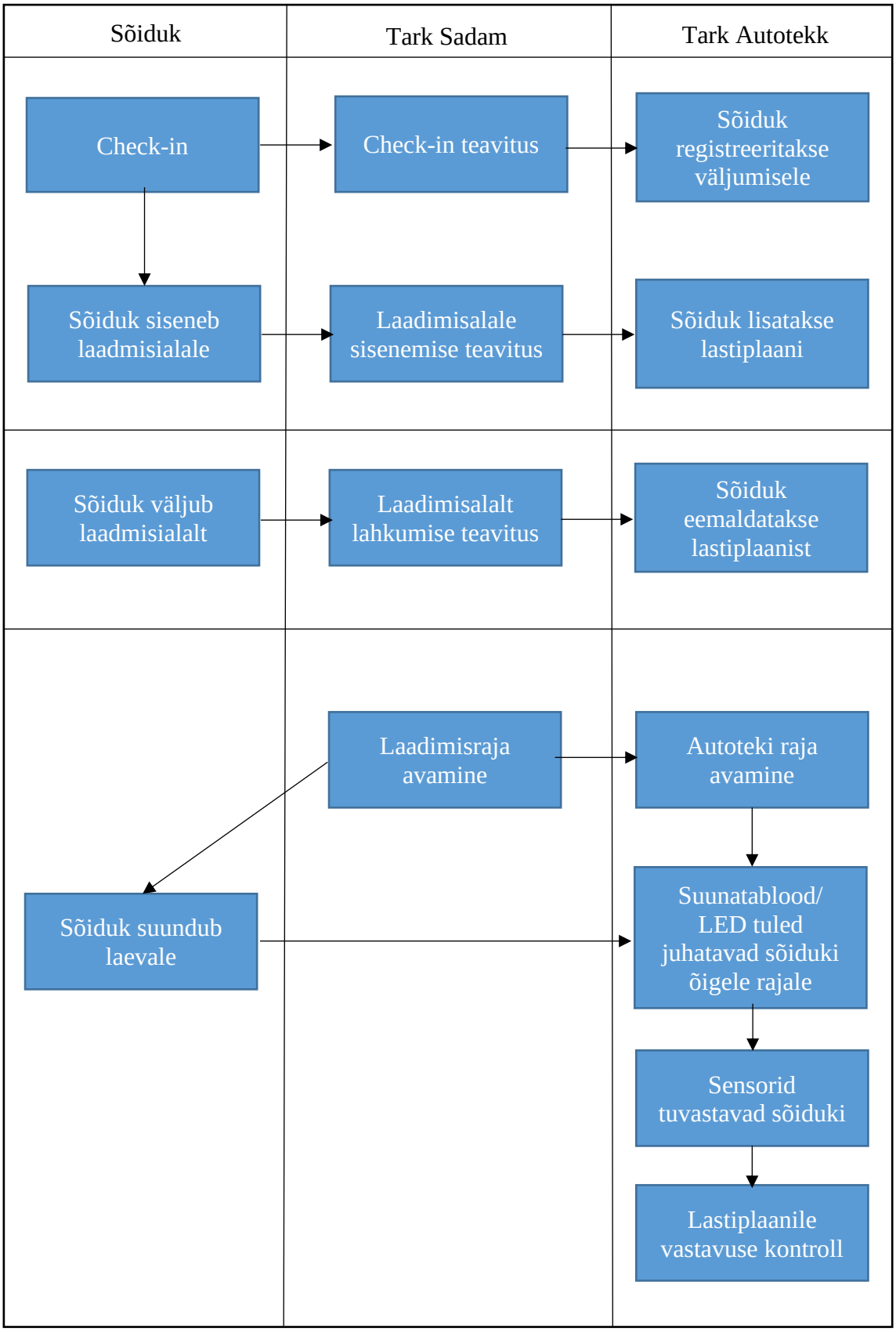
Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Tallink ja TalTech arendavad koostöös „Targa Autoteki“ projekti mille eesmärk on luua süsteem, mis optimeerib ja automatiseerib sõidukite laadimist ja lossimist Tallinki laeva [1]. Tallinna Sadamas on juba kasutusel Targa Sadama [2]. „Targa Autoteki“ projekt üritab sarnaseid eesmärke täita laevade tekkidel. Loetletud eesmärkide saavutamiseks uuritakse projekti raames erinevaid IoT *(Internet of Things*), sensorite ja muid tark- ja riistvaralisi lahendusi, et leida parim lahendus.

„Targa Autoteki“ süsteem koostab Tallinna Sadamalt saadud *check-in* andmete põhjal lastiplaani ja vastavalt sellele juhatab sõidukid laadimisel õigele kohale. Selleks juhib süsteem laevale paigaldatud suunatabloosid ja LED (*Light Emitting Diode*)-tulesid, mille järgi autod enda koha laevatekil leiavad. Lisaks on laevatekkidele lakke paigaldatud ultrahelisensorid, mis suudavad mõõta enda all olevate objektide kaugust seeläbi tuvastades, kas sensori all on auto ja mis on selle kõrgus. Selliseid mõõtmisi tehes aitavad sensorid süsteemil jälgida olukorda laeva autotekkidel, tuvastada erinevaid veaolukordasid (näiteks sõiduki puudumine lastiplaanis ette nähtud asukohas) ja valideerida reaalse olukorra vastavust laeval lastiplaanile. Lisaks on süsteemil brauseris kasutatav kasutajaliides, mille kaudu on võimalik juhtida süsteemi LED-tulesid ja suunatabloosid ja saada infot sensorite kohta. Projekti raames uuritakse ka võimalust muuta autotekkide radade juhtimise täielikult automaatseks sedasi, et süsteem juhib ise automaatselt sõidukid lastiplaanis ette nähtud kohale. Joonis 1 selgitab lihtsustatult Targa Autoteki ülesandeid.



Joonis 1. Lihtsustatud visualisatsioon Targa Autoteki põhiprotsessidest.

Süsteemi testimist raskendavad erinevad asjaolud. Kuigi prototüüpide testimiseks on projekti meeskonnale korraldatud ligipääs Tallinki Megastar laevale, on see võimalik ainult laeva töövälisel ajal. Süsteemi täielikuks testimiseks peaks laeva tekkidele paigaldama sadu sensoreid ja LED-tulesid koos vajalike võrguseadmetega ning korraldama reaalseid eksperimente sadade erinevat tüüpi sõidukitega. Mõnede testide korraldamine võib olla ka laevale ohtlik, näiteks on oluline, et laev ei kalduks laadimisel, lossimisel ega sõidu ajal liigselt ühele küljele. Kui testimise käigus peaks ilmnema, et sensorite paigutust oleks vaja muuta, tähendaks see veel lisatööd. Selliseid probleeme aitaks ennetada süsteemi osade tarkvaraline simulatsioon.

Antud töö eesmärk on luua Targa Autoteki süsteemi testimiseks virtuaalne simulatsioonikeskkond, mis võimaldaks süsteemi testida ilma vajaduseta reaalsel laeval eksperimente korraldada, säästes seeläbi raha ja aega. Vajadusel on lihtne simuleerida erinevaid olukordi erinevate sõiduki tüüpidega, lihtne on teha katsetusi erinevate sensorite seadistustega. Sensorite tihedust, positsiooni saaks muuta lihtsa vaevaga, võib katsetada ka erinevate sensorite tüüpidega.

Järgnevas töös kirjeldame milliseid tehnoloogiaid simulatsiooni ehitamiseks kasutati, kuidas neid tehnoloogiaid kombineeriti, millised on loodud simulatsiooni funktsionaalsused ja kuidas võiks simulatsiooni tulevikus edasi arendada.

2 Modelleerimine ja simulatsioon

Süsteemi modelleerimiseks nimetame reaalset süsteemi kirjeldava mudeli koostamist. Mudeliks nimetame mingisugust lihtsustatud kirjeldust mingisugusest süsteemist. On võimalik eristada staatilisi ja dünaamilisi mudeleid, staatiline mudel kirjeldab süsteemi kindlal ajahetkel, dünaamiline mudel kirjeldab ka süsteemi muudatusi aja jooksul. Mudel võib olla esitatud erinevatel kujudel, graafiliselt, matemaatiliselt, kirjalikult, verbaalselt, füüsiliselt jne. Modelleerida võib kõike, füüsilisi objekte, käitumismustreid, ilmastiku nähtusi jne. Mudeleid saab jaotada kahte gruppi: füüsilised ja matemaatilised mudelid. Füüsilised mudelid on meile kõigile elust ühel või teisel kujul tuttavad, näiteks mudelautod, hoonete maketid, mudellennukid jne. Matemaatilised mudelid jagunevad omakorda analüütilisteks mudeliteks ja simulatsioonideks. Analüütiliste mudelite puhul väljendatakse mudelit läbi matemaatiliselt võrrandite. Simulatsioonide puhul ei ole läbi võrrandite väljendamine niivõrd oluline. Sobivat analüütilist mudelit mingisuguse nähtuse kirjeldamiseks ei ole ka alati lihtne leida. Mudeli eesmärk ei ole modelleeritava objekti või süsteemi võimalikult täpne jäljendamine, mudel võib olla täpselt nii täpne või ebatäpne kui on vajalik modelleerimise eesmärgi saavutamiseks. Süsteemi simuleerimiseks nimetame reaalse süsteemi käitumise uurimist mudeli abil. Simuleerimisel võib olla erinevaid eesmärke, näiteks lennusimulaatoreid ehitatakse õpetamiseks ja treenimiseks, liiklussimulatsioone kasutatakse mingi liikluslahenduse läbilaskevõime analüüsimiseks jne. Diskreetseks süsteemiks loetakse süsteemi, mille olek muutub diskreetsetel ajahetkedel, pidevate süsteemide puhul muutub süsteemi olek pidevalt üle aja. Paljudel süsteemidel võivad olla üheaegselt nii diskreetse süsteemi tunnused kui ka pideva süsteemi tunnused [3].

Agendipõhine modelleerimise puhul modelleeritakse nähtusi maailmas kasutades niinimetatud agente, keskkonda ja kirjeldatakse kuidas agendid omavahel reageerivad ja kuidas agendid keskkonnale reageerivad. Agendipõhise mudeli eelis analüütilise mudeli ees, mis kirjeldaks kogu süsteemi on asjaolu, et maailma on võimalik kirjeldada lihtsate käitumisreeglite abil, mida agendid järgivad. Sellist simulatsiooni on tihti lihtsam arendada ja inimestel on sellest intuitiivselt lihtsam aru saada [4]

Järjest enam ettevõtteid on võtmas kasutusele ka nn digitaalsete kaksikute (inglise keeles *digital twin*) tehnoloogiaid. Iga füüsilise objekti jaoks luuakse tema digitaalne kaksik, mille seisu uuendatakse reaalsele objektile paigaldatud sensoritelt tulnud infoga. Sellise infoga on võimalik täiendada objekti mudeleid, mille abil tulevikus erinevaid simulatsioone käitada või reageerida ennetavalt mingisugusele veaolukorrale. Näiteks Tesla loob digitaalse kaksiku kõikidele oma autodele ja kogutud andmete põhjal disainib autodele omakorda tarkvarauuendusi [5]. Digitaalne kaksik eksisteerib objekti terve elutsükli vältel.

Valdkondade nagu elektroonika, keemia, lennundus või kosmosetehnika jpm jaoks on arendatud välja spetsiifilised tarkvaralahendused simulatsioonide käitamiseks. Need lahendused on optimeeritud vastavalt nende valdkondade probleemide modelleerimiseks ja simuleerimiseks. Viimase kümnendi jooksul on aga järjest rohkem uuritud ka mängumootorite kasutamist erinevate simulatsioonide käitamiseks [6]. Mängumootorites on enamasti sisseehitatud 3D füüsikamootor, mis suudab veenvalt modelleerida lihtsamaid jäikade kehade liikumist. Kui simulatsioon vajab täpsemaid arvutusi või funktsionaalsust mida füüsikamootor ei võimalda, võimaldab mängumootor vastava loogika ise kirjutada. Mõned mängumootorid võimaldavad ka mitme füüsikamootori vahel valida. Kuna mängumootori põhieesmärk on videomängude loomine, kus pannakse suur rõhk visuaalsele esitlusele, siis ka simulatsioonide ehitamise puhul on lihtne simuleeritavaid protsesse atraktiivselt visualiseerida.

3 Targa Autoteki süsteemi komponendid

Järgnevalt kirjeldame Targa Autoteki süsteemi komponente nii nagu nad töö kirjutamise hetkel planeeritud on. Süsteemi disain ei ole täielik kuna arendus ei ole lõpetatud vaid on töö kirjutamise ajaks COVID-19 pandeemia tõttu pausile pandud. Targa Autoteki süsteemi disain ja teostus ei kuulu antud töö hulka, siin kirjeldame neid ainult lugejale taustainfo andmiseks.

3.1 Targa Autoteki teenused

Smart Port – Veebiserver, mis võtab vastu erinevaid sõnumeid Tallinna Sadamalt. Tallinna Sadam saadab välja XML(*Extensible Markup Language*) sõnumeid erinevate sõidukitega seotud sündmuste kohta sadama laadimisalal. Järgnevalt kirjeldame Targa Autoteki jaoks olulisemad sõnumid:

- registerPass – Sõiduk registreeritakse väljumisele või uuendatakse juba väljumisele registreeritud sõiduki infot
- waitingAreaInNotification – Sõiduk on sisenenud laadimisalale
- waitingAreaOutNotification – Sõiduk on väljunud laadimisalalt

Cargo Web UI (*User Interface*) backend – Hoiab endas veebiliideses kuvatavat infot, veebiliidese kasutajakontosid, nende privileege, edastab veebiliidesest tulnud käsud laadimisradade avamise ja sulgemise kohta vastavatele teenustele. Teenusega on võimalik suhelda saates REST (*Representational State Transfer*) liidesele päringuid. Päringute sisu peab olema JSON (*Javascript Object Notation*) vormingus.

Cargo Web UI frontend – React projekt, mille ülesandeks on kuvada kasutajaliides. Suhtleb Cargo Web UI backend'iga üle selle poolt pakutud REST API (*Application Programming Interface*). Kasutajaliidese funktsionaalsusest räägib lähemalt peatükk 3.4.

AmpronProxy – Proksiteenus läbi mille juhitakse laeva tekkidel olevaid suunatabloosid. Suunatabloodest räägib peatükk 3.3.

NodeCoreService - Organiseerib laekunud info põhjal sõidukid väljumistesse ja genereerib sõidukite info põhjal lastiplaane.

3.1.1 Docker

Eelnevalt kirjeldatud Targa Autoteki teenuseid käitatakse Docker konteinerites. Docker on tarkvara, mis võimaldab tarkvara käitada niinimetatud konteinerites [7]. Dockeri konteiner on standardne ühik tarkvara, mis sisaldab kõiki enda sõltuvusi ja mida on võimalik käitada kõikidel platvormidel mis toetavad Docker'it. Docker'i konteinerid isoleerivad enda sisu keskkonnast, milles nad jooksevad. Seega vähendab Docker oluliselt nõudmisi keskkonnale milles tarkvara käitada [8]. Ainuke sõltuvus keskkonnale on, et seal peab olema võimalik Docker'it käitada. Dockeri konteinerites on võimalik käitada niinimetatud Dockeri tõmmiseid (inglise keeles *image*). Tõmmiste sisu kirjeldatatakse spetsiaalses failis nimega Dockerfile.

Dockerfile on spetsiaalne tekstifail, mis kirjeldab Dockeri tõmmise sisu ja sisaldab endas samme kuidas Docker tõmmist luua.

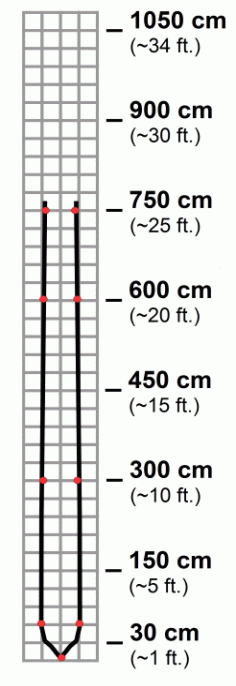
Docker-compose võimaldab kerge vaevaga mitmeid Docker konteinereid samaaegselt käitada. Selleks tuleb defineerida fail docker-compose.yaml, mis kirjeldab milliseid konteinereid käitada ja kuidas need omavahel suhelda saavad.

3.2 Sensorid

Igale tekile paigaldatakse lakke ultrahelisensorid, mis suudavad tuvastada kui nende alt auto läbi sõidab. Sensorid paigutatakse lakke iga 320cm tagant ja suunatakse otse alla. Joonis 2 on foto sõidukitekile paigaldatud prototüüp sensorist.

Joonis 2. Laevatekile paigaldatud sensori prototüüp [9].

Sensorid teevad mõõtmisi iga 100 millisekundi tagant ja saadavad mõõtmistulemused serverisse. Süsteem kasutab seda infot, et võrrelda eelnevalt koostatud lastiplaani reaalse olukorraga sõidukitekkidel. Sensorina on kasutusel MB7060-500 XL-MaxSonar WRC, mille kiire laius on 60cm ja ulatus 765cm. Joonis 3 esitab sensori kiire kuju.



Joonis 3. Sensori kiire kuju [10]. Ühe halli ruudu külg on 30cm.

Sensorid teostavad mõõtmisi 1cm täpsusega, kuid edastavad tulemusi 10cm täpsusega. Kuna sõidukid erinevad oma kujudelt ja suurustelt ei ole sõidukite olemasolu tuvastamiseks täpsemaid mõõtmisi vaja ja sellisel viisil on võimalik mõõtmistulemuse saatmisel kasutada ainult 6 bitti. Tabel 1 selgitab mõõtmistulemuste konverteerimist.

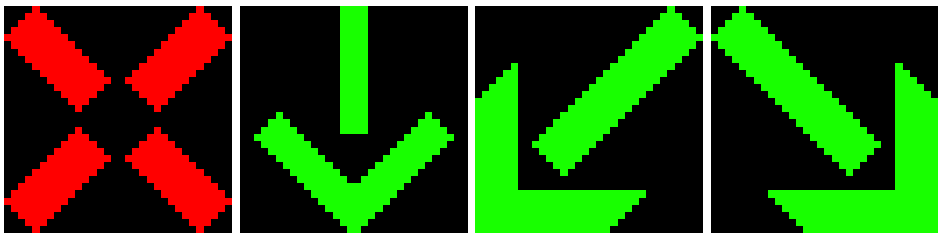
Tabel 1. Mõõtmistulemuste konverteerimine.

Mõõtetulemus	**Edastatav väärtus**
0-20 cm	0
21-30 cm	1
30-40 cm	2
…	…
>640 cm	63

Iga sensori küljes on ka LED-tuli mis aitab autosid oma kohale juhatada. LED suudab näidata kas punast või rohelist valgust. Sõiduki peaksid vältima punase LED-tule alt läbi sõitmist ja oma koha leidmisel autotekil järgima rohelisi LED-tulesid. Kui süsteem tuvastab, et sensorite all on auto seisma jäänud, muudab see vastava LED-tuled punaseks ning annab teistele sõidukitele märku, et antud asukoht on hõivatud.

3.3 Suunatablood

Lisaks LED-tuledele kasutab Targu Autoteki süsteem autode juhtimiseks suunatabloosid. Suunatablood on lihtsad LED-ekraanid mis paigaldatakse laevatekkide sissesõitudele ja väljapääsudele ning mis kuvavad kas juhtivaid või keelavaid märguandeid autojuhtidele. Sunnatabloodena on kasutusel Ampron DS-320x320 [11]. Joonis 4 esitab suunatabloode poolt kuvataid märke. Suunatabloosid on võimalik juhtida läbi Targa Autoteki kasutajaliidese.



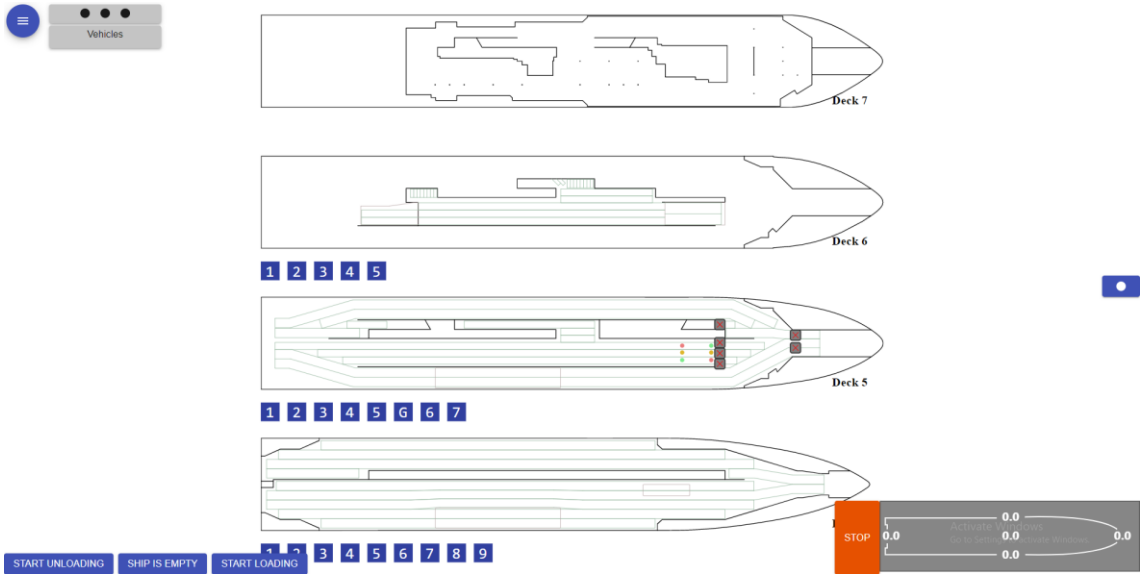
(a) (b) (c) (d)

Joonis 4. Suunatabloode märgid Punane rist (a) - sõiduk peaks vältima suunatabloo alt läbi sõitmist. Roheline nool alla (b) - sõiduk võib sõita suunatabloo alt läbi ja peaks hoidma otse suunda. Roheline nool alla ja vasakule (c) – sõiduk võib sõita märgi alt läbi.

3.4 Kasutajaliides

Süsteemil on ka interaktiivne veebiliides, mille kaudu kasutajatel on võimalik laadimise seisu jälgida. Lisaks on kasutajatel võimalik laadimise kulgu juhtida läbi radade avamise või sulgemise. Radade avamine või sulgemine muudab vastavalt laeval olevate suunatabloode kuvatavaid juhtmärke ja sensorite LED-tulede värvi. Suunatabloosid on võimalik ka iseseisvalt juhtida sõltumata sellest kas rada on avatud või mitte.

Kasutajaliides annab ülevaate laadimise seisust, tuues eraldi välja laevale jõudnud sõidukite arvu, laadimisalale jõudnud sõidukite arvu ja väljumisele registreerunud sõidukite arvu. Lisaks annab kasutajaliides infot laeva kreeni kohta. Kasutajaliides on ehitatud kasutades React raamistikku. Joonis 5 esitab kasutajaliidese kuvatõmmist.

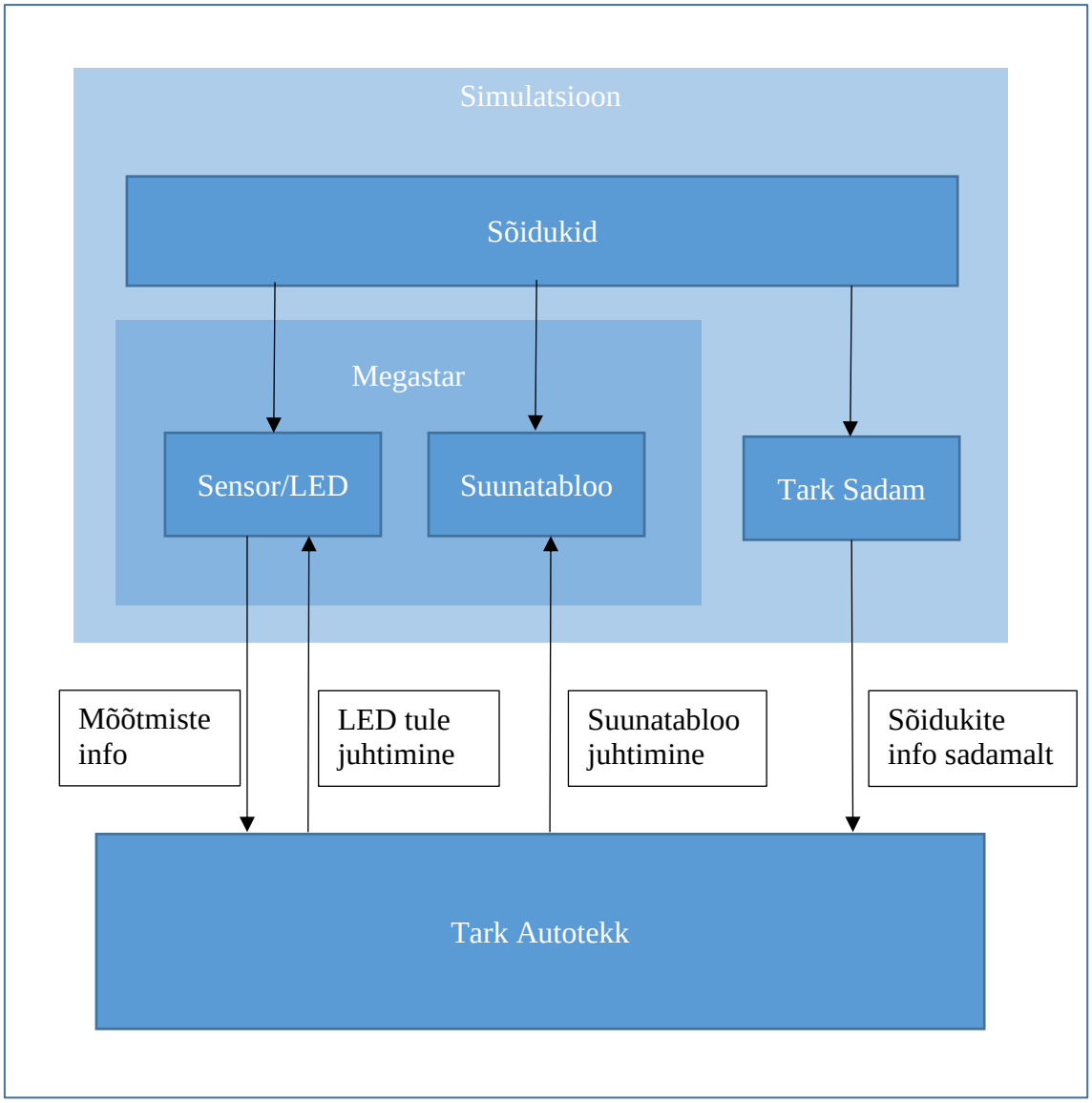


Joonis 5. Kasutajaliidese kuvatõmmis.

4 Targa Autoteki simulatsioonikeskkond

Loodav simulatsioonikeskkond peab suutma süsteemile genereerida realistlikku sisendinfot ja suutma mõõta põhilisi näitajaid süsteemi toimimise kohta (laadimise aeg). Simulatsioonikeskkond peab suutma tekitada erinevaid liiklusolukordi nii laevatekkidel kui ka laadimisalal, näiteks aeglaselt liikuvad sõidukid, valele rajale sõitvad sõidukid jne. Lisaks peab läbi simulaatori olema võimalik testida, kuidas süsteem reageerib erinevatele vigadele, mis võivad esineda süsteemi enda komponentides. Näiteks kui sensor saadab valesid andmeid või ei saada üldse andmeid. Antud juhul on simulaatori ülesanne modelleerida autode liikumist laeva tekkidel ja saata süsteemi sensori mõõtmisandmeid, mida autode liikumine tekitab.

Teisest küljest saab simulatsioonikeskkond süsteemilt sõnumeid selle kohta, milliseid märke peavad suunatablood näitama ja millised LED tuled peaksid rohelised või punased olema, et autod õigetele radadele juhatada. Seega peab simulatsioon dünaamiliselt reageerima ka süsteemi poolt antud sisenditele. Joonis 6 kirjeldab simulatsioonikeskkonda kuuluvaid Targa Autoteki komponente ja nende suhtlemist ülejäänud Targa Autoteki süsteemiga.



Joonis 6. Simulatsiooni kirjeldus.

Simulatsioonide käitamisest ilmnevad loodava süsteemi kitsaskohad varem, enne laevale täieliku süsteemi paigaldamise kulutuste tegemist. Võib osutuda, et sensoreid on võimalik paigutada hõredamalt, tihedamalt, või et ultrahelisensorite asemel on tarvis kasutada LIDAR (*Light Detection And Ranging*) sensoreid või videokaameraid, et tõsta sõidukite tuvastamise töökindlust.

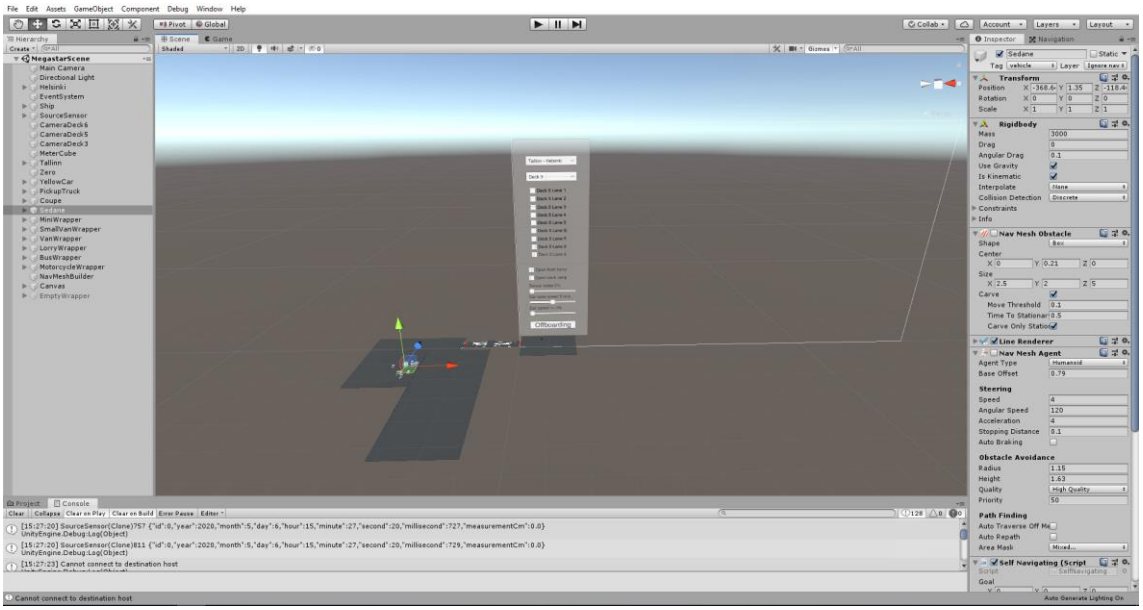
5 Unity

Unity on videomängude ja muude 3D, liitreaalsuse või virtuaalreaalsuse keskkondade loomise tarkvara [12]. Unity sai alguse mänguarenduse platvormina, aga tänapäeval kasutatakse seda ka paljudes teistes valdkondades peale mängude arendamise. Unity on kasutusel näiteks autotööstuses isejuhtivate autode treenimiseks simulatsioonikeskkondade ehitamiseks [13]. Lisaks filminduses [14], arhitektuuris [15], ehituses ja paljudes muudes insenerivaldkondades. Samuti leiab viiteid Unity kasutamisest akadeemilisest kirjandusest. Unity't on kasutatud näiteks tööstusprotsesside simuleerimiseks ja visualiseerimiseks [16], lisaks on leitud, et Unity't on võimalik kasutada inimeste hoonetest evakueerimise simuleerimiseks [17]. Kuna Unity'r rakendatakse järjest rohkem simulatsioonide käitamiseks, siis on käivitatud ka teenus Unity Simulation mis võimaldab Unity's ehitatud simulatsioone pilves käitada [18]. Antud hetkel on Unity Simulation teenus beta faasis.

Unity toetab Windows ja MacOS operatsioonisüsteeme ja projekte on võimalik eksportida paljudele erinevatele platvormidele nagu WebGL, Windows, MacOS, Linux, iOS, Android ja paljud muud. Unity kasutab sisemiselt füüsika simulatsioonideks Nvidia PhysX füüsikamootorit [19].

5.1 Unity Editor

Unity organiseerib erinevad ressursid nn stseenidesse (inglise keeles *Scene*). Antud simulatsiooni ressursid on paigutatud stseeni MegastarScene. Kõik objektid Unity stseenides on nn mänguobjektid (inglise keeles *GameObject*). Mänguobjektidest saab hierarhiaid luua, märkides ühe mänguobjekti teise alamaks. Mänguobjektidel endil ei ole peale suuruse, nurga ja asukoha maailma koordinaatsüsteemi suhtes ühtegi teist omadust. Mänguobjektidele saab omadusi ja käitumisi anda lisades neile nn komponente(inglise keeles *Component*). Mänguobjekte võibki pidada pelgalt komponentide hoidjateks. Joonis 7 näitab kuidas Unity kasutajaliides mänguobjekte ja komponente organiseerib.



Joonis 7. Unity kasutajaliidese kuvatõmmis. Vasakul on loetletud stseeni kuuluvad mänguobjektid. Keskel on stseeni 3D maailm. Paremal on valitud mänguobjekti komponendid.

5.2 Unity Component

Unity komponendid lisavad mänguobjektidele omadusi. Komponentide abil saab mänguobjektidele lisada kuju, massi, materjali jne. Igal mänguobjektil on automaatselt seadistatud Transform komponent, mille abil määratakse tema asukoht, suurus, orientatsioon maailmas. Transform komponenti ei ole võimalik mänguobjektidele eemaldada. Komponendid võimaldavad mänguobjektidele lisada ka skripte (inglise keeles *Script*), mis võimaldavad programmeerida mänguobjektidele automaatset käitumist. Unity toetab skriptide kirjutamiseks C# programmeerimiskeelt.

5.3 MonoBehaviour

Unity poolt on pakutud skriptimiseks MonoBehaviour baasklass, mille peale on võimalik oma skripte arendada. MonoBehaviour defineerib hulga baasmeetodeid, mida kasutaja Unity komponendi elutsükli erinevatel hetkedel kutsub. Nende meetodite kirjeldamisega ongi võimalik oma komponentidele omadusi või käitumist lisada. Olulisemad meetodid mida MonoBehaviour pakub on:

Start - meetodit kutsutakse komponendi elutsükli jooksul ühe korra, esimesel korral kus komponent on aktiivses olekus enne esimest OnUpdate või OnFixedUpdate meetodi käitamist.

OnUpdate - meetodit kutsutakse iga kaadri joonistamisel seni kuni komponent on aktiivses olekus. On oluline meeles pidada, et kaadrisagedus võib varieeruda

OnFixedUpdate - sarnane OnUpdate meetodile, kuid kutsutakse fikseeritud intervalli, vaikimisi iga 20 millisekundi tagant, sagedust on võimalik projekti tasemel muuta.

5.4 NavMesh

Sõiduki juhtimise aluseks kasutati Unity3d NavMesh süsteemi. NavMesh süsteem koosneb järgmistest komponentidest [20]:

NavMesh – andmestruktuur, mis kirjeldab mudeli liigeldavaid pindu ja võimaldab leida otseima tee ühest punktist teise, ehitatakse automaatselt mudeli geomeetria põhjal.

NavMesh Agent - võimaldab objektidel mööda maailma navigeerida, samal ajal vältides teiste agentide ja takistustega kokkupõrkeid.

NavMesh Obstacle - võimaldab tähistada objekte mida agendid peaksid navigeerimisel vältima.

Off-Mesh link - võimaldab tekitada otseteid, mida pole võimalik tavalise pinnana kujutada, näiteks uksed või kohad, mis vajad navigeerimiseks hüppamist.

NavMesh Agent’id kasutavad sihtmärgini lühima tee leidmiseks A* rajaleidmise algoritmi [21], mida kirjeldame lühidalt peatükis 5.5.

Kuna laeval on ka liikuvad rambid, mille kaudu saab tekkide vahel liikuda ei ole antud juhul võimalik kasutada tavalist NavMesh’i kuna seda on võimalik genereerida vaid staatiliste pindade peal enne simulatsiooni käivitamist. NavMesh asemel tuleb kasutada komponenti **NavMeshSurface** mis võimaldab NavMesh’i genereerida ja uuendada ka programmi jooksmise ajal [22]. See võimaldab NavMesh’i genereerida iga kord peale mõne rambi avamist või sulgemist.

5.5 A* rajaleidmise algoritm

Unity jagab navigeeritavad pinnad polügonideks ja seejärel rakendab A* rajaleidmise algoritmi, et leida kiireim tee sihtmärgini. Alates alguspunktist hindab algoritm iga sammu tegemise jaoks valemit (1).

$$f(n) = g(n) + h(n) \quad (1)$$

Kus g(n) on alguspunktist polügonini n liikumise hind ja h(n) on heuristik, mis hindab polügonist n sihtpunktini liikumise hinda. Igal iteratsioonil valitakse edasi liikumiseks polügon, millel on madalaim f(n) ja peale liikumist tehakse arvutus uuesti [23].

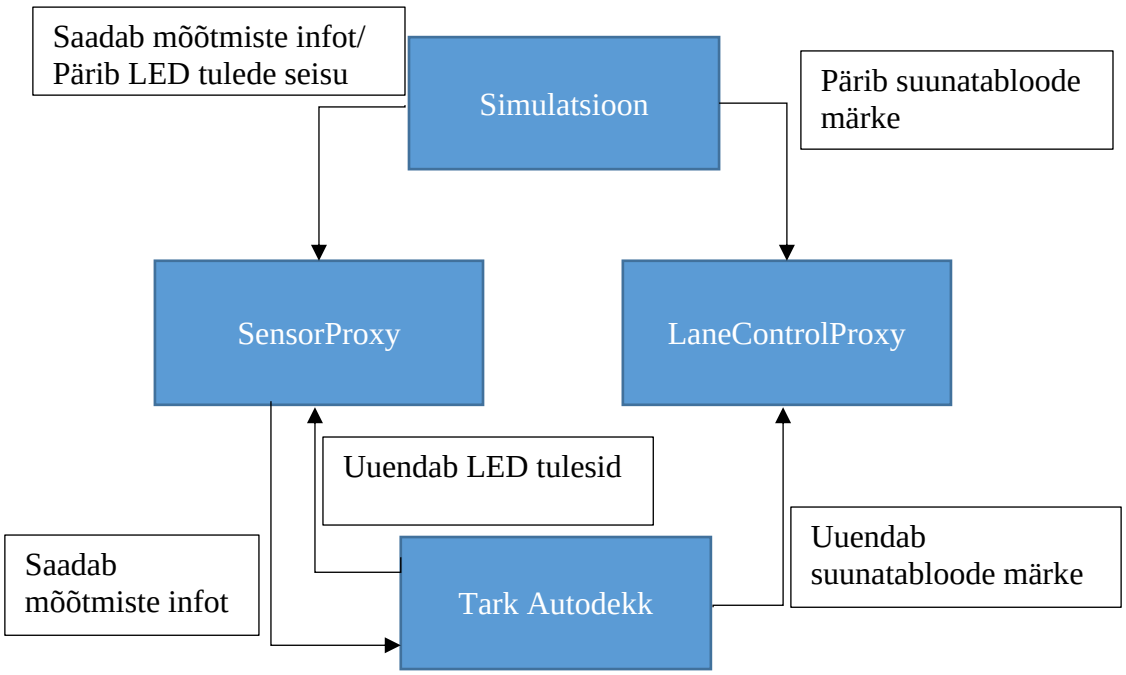
6 Mudeli kirjeldus

Simulatsioonide käitamiseks tuli Unity3D keskkonnas luua Tallink Megastar autotekkidest 3D mudel. Mudelisse on kaasatud 3., 5. ja 6. autotekk. 7. autotekil on autode paigutamisel teistest tekkidest erinev loogika, tekk on radadeks jaotatud ülejäänud tekkidest erineva loogika alusel, samuti pole ka selle jaoks olemas Targa Autoteki lahenduse kontseptsiooni. Seega tuleb 7. autoteki mudel simulatsiooni lisada peale sellise kontseptsiooni arendamist.

Kasutajaliidesest sai üle võetud koordinaatsüsteem, mille suhtes paigutatakse sensorid ja sõidukid, et lihtsustada andmete võrdlemist ja soodustada sensorite kaardistamise taaskasutust simulatsiooni ja kasutajaliidese vahel. Kasutajaliides ja simulatsioon paigutavad mõlemad koordinaatsüsteemi nullpunkti Megastar'i ahtri ja pakpoordi nurgale. Kasutajaliidese x-telg on nullpunktist suunaga vööri poole ja y-telg on nullpunktist suunaga tüürpoordi poole. Simulatsiooni teljed on sarnase asetusega, kuid kuna tegemist on 3D simulatsiooniga siis y-telge kasutatakse vertikaalse kõrguse märkimiseks. Kasutajaliidese y-telje väärtused on üle kantavad simulatsiooni z-teljele. Simulatsiooni 3D maailma 1 ruumiühik on määratud võrduma 1m füüsilises maailmas. Viide simulatsioonikeskkonna lähtekoodile on välja toodud Lisas 1.

6.1 Simulatsiooni arhitektuur

Simulatsioonide käitamisel ilmnes, et sensorite poolt nõutav suur päringute arv avaldab mõju simulatsiooni kaadrisagedusele, muutes simulatsiooni aeglasemaks. Lisaks sellele puudub Unity's sisse ehitatud viis kuidas lihtsalt serveerida REST liidest. Seetõttu sai vastu võetud otsus viia 100ms tagant mõõtmise uuendamine simulatsioonist välja ja luua 2 uut vaheteenust mis aitavad simulatsioonil suhelda Targa Autoteki teenustega. Joonis 8 näitab kuidas abiteenused simulatsioonikeskkonna arhitektuuri sobituvad.



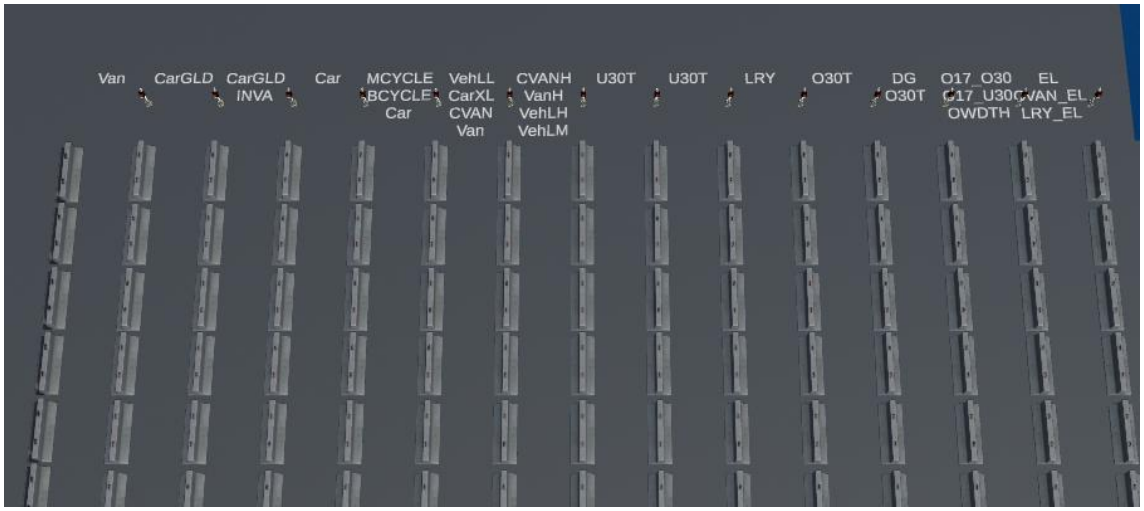
Joonis 8. Abiteenuste roll simulatsiooni arhitektuuris.

SensorProxy ja LaneControlProxy on lihtsad serverid, mis serveerivad REST liideseid, ühelt poolt võimaldades Targa Autoteki teenustelt vastu võtta käske LED-tulede või suunatabloode uuendamiseks, teiselt poolt võimaldades Unity simulatsioonil neid uuendusi pärida. SensorProxy'l on lisaülesanne iga 100ms tagant edastada Targale Autotekile sensorite viimased mõõtetulemused. SensorProxy ja LaneControlProxy kirjutamiseks on kasutatud Go keelt [24]. Go programmid on väga madala ressursinõudlusega [25] ja SensorProxy puhul võimaldab *goroutine*'de abil sensorite mõõtmistulemuste saatmist lihtsalt parallelliseerida. Mõlema teenuse Docker tõmmise aluseks on Golang'i ametlik tõmmis, kus on kõik Golang programmide käitamiseks vajalik juba olemas.

6.2 Laadimisrajad

Laadimisalal jaotatakse autod laadimisradadesse. Igale laadimisrajale on lubatud sõita ainult kindlat tüüpi sõidukitel. Laadimisradade arvu ja nendele lubatud sõidukite tüübid on seadistatud failis Resources/conf/loadingLanes.json. Autode arv tüüpide kaupa on seadistatav failis Resources/conf/cars.json. Simulatsiooniprogrammi käivitamisel loetakse seadistused mõlemast failist, mudelisse luuakse laadimisalale vastavalt

seadistatud laadimisrajad ja seadistatud sõidukid jaotatakse vastavalt sõiduki tüübile laadimisradade vahel ära. Näiteks on välja toodud kergesõidukite klassid Lisas 2. Kui rohkem kui üks laadimisrada on sama tüübiga siis jaotab süsteem sõiduki laadimisradade vahel võrdselt. Laadimisraja avamiseks tuleb teha hiirega vasakklikk vastava raja valgusfooril. Joonis 9 kujutab kuvatõmmist genereeritud laadimisradadest sadamaalal.

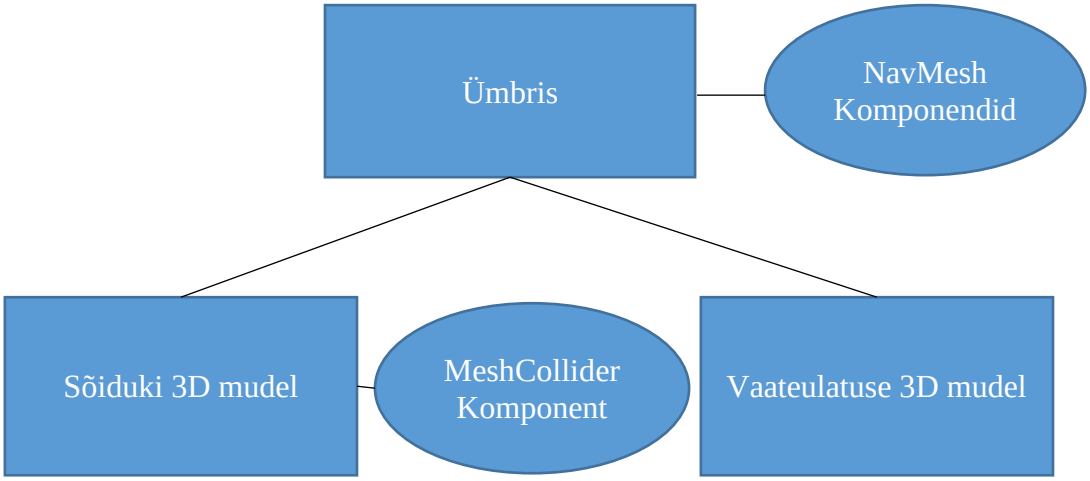


Joonis 9. Kuvatõmmis näitab genereeritud laadimisradu ja nendele lubatud sõidukite tüüpe.

6.3 Sõidukite mudelid

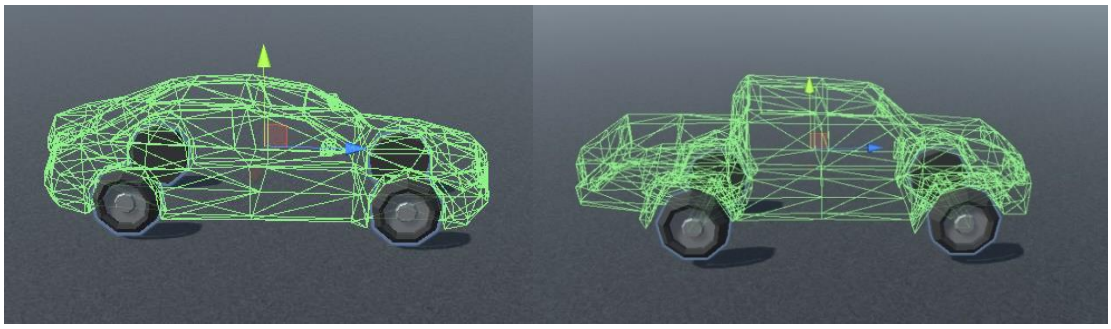
Erinevate sõidukite 3D mudelid on pärit Unity AssetStore'st. Unity AssetStore on veebikeskkond kus on võimalik osta, müüa ja jagada erinevaid ressursse mida on võimalik Unity's kasutada. Kasutatud mudelite paketid on „Low Poly Cars" [26], „Low Poly Streetpack" [27], „Yughues Free Concrete Barriers" [28], „TGU Skybox Pack" [29] ja „UAA - City Props – Vehicles" [30]. Kõik kasutatud mudelite paketid on tasuta saadaval ja litsents lubab neid kasutada kõikidel eesmärkidel.

Kõik sõidukid simulatsioonis on organiseeritud ümbris-mänguobjektidesse, mis hoiavad endas 3D mudelit, vaatevälja mudelit ja vajalikke komponente. Joonis 10 esitab sõidukite mänguobjektide ja komponentide hierarhiat visuaalselt. Järgnevad lõigud seletavad nendest olulisemad lahti.



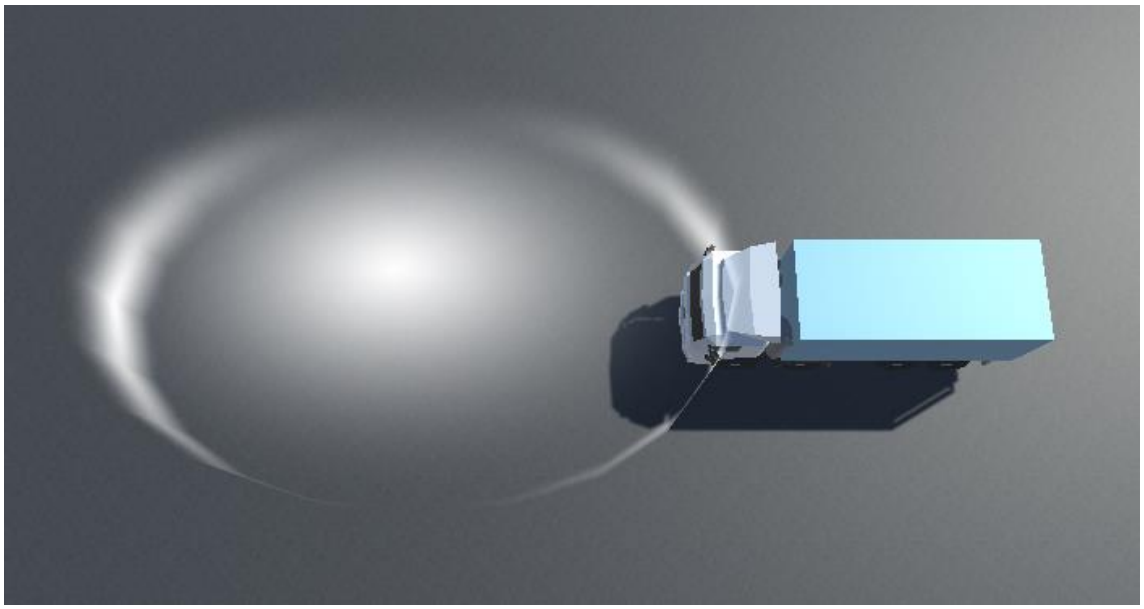
Joonis 10. Sõidukite mudelite omavaheline paigutus simulatsioonis.

Sõidukite 3D mudelitele on lisatud MeshCollider komponent. Collider komponendid defineerivad, milline on mänguobjekti füüsiline kuju. MeshCollider komponent järgib füüsilise kuju lisamisel objekti enda kuju. Joonis 11 esitab näited kuidas MeshCollider komponent järgib erinevate sõidukite kuju. Peamine põhjus, miks kasutada MeshCollider'it mõne lihtsama kuju – näiteks BoxCollider'i, mis on lihtne ristkülik – asemel, on sensoritele realistliku kuju tekitamiseks.



Joonis 11. Näited sellest kuidas MeshCollider järgib erinevate sõidukite kuju.

Sõiduki vaatevälja modelleerimiseks kasutame lihtsat silindri mudelit mis on läbi z-telje venitamise elliptiliseks muudetud. Joonis 12 näitab sõiduki vaatevälja visuaalselt. Kaugeima rohelise LED-tule või eesoleva sõiduki otsimisel arvestavad sõidukid ainult selliseid objekte jäävad nende vaatevälja sisse.



Joonis 12. Visualiseeritud on sõiduki vaateulatus mida kasutatakse sensorite ja teiste sõidukite otsimiseks.

6.4 Sõidukite navigeerimine tekkidel

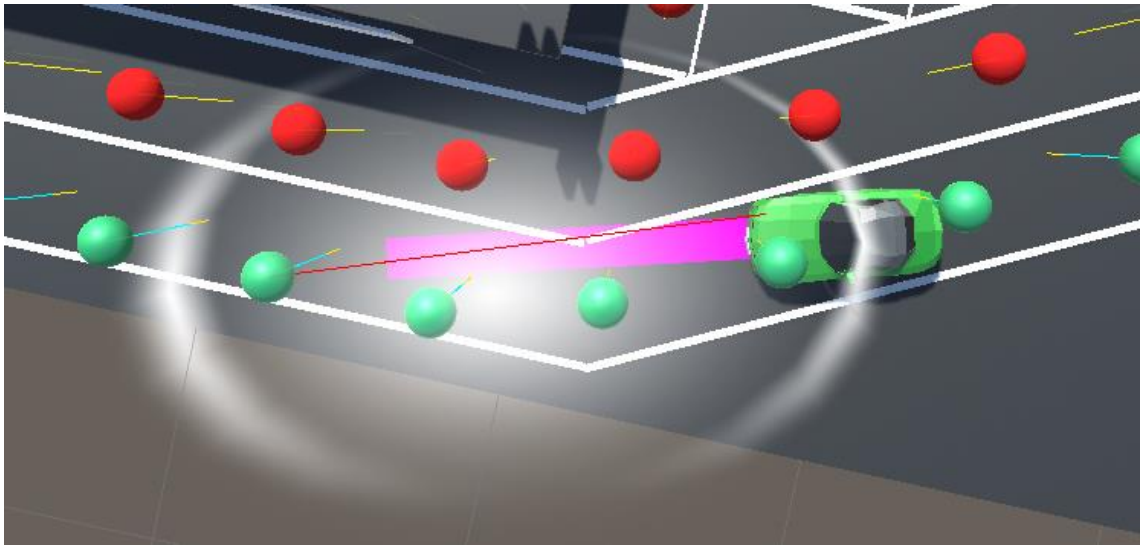
Kuna laadimise alguseks on genereeritud täpne lastiplaan kõikide sõidukite asukohaga võiks arvata, et kõikidele sõidukitele saab lihtsalt vastavalt lastiplaanile sihtkohad määrata ja lasta neil iseseisvalt sinna navigeerida, kasutades NavMeshAgent funktsionaalsust. Kuid reaalses laadimisolukorras ükski autojuht ei tea laevale sõitu alustades, mis rajale ta suunatakse. Selle asemel järgivad autojuhid täna laevameeskonna juhiseid. Targa Autoteki süsteemis juhivad autosid õigele asukohale LED-signaalid ja suunatablood, seega peaksid ka simulatsiooni sõidukid suutma laeval navigeerida ainult neid signaale järgides. Teisiti talitades tekib oht, et simulatsiooni käigus liiguvad kõik autod alati korrektselt oma sihtkohale olenemata sellest, millised rajad on süsteem avanud või milliseid juhiseid edastavad juhtidele liiklusmärgid.

Laeva sõidukitekid on simulatsioonis märgitud NavMesh navigeeritavaks alaks. Erinevad seinad ja muud takistused märgiti NavMesh Obstacle'ks, et sõidukid neid väldiksid. Kõik sõidukid märgiti NavMesh Agent'ideks ja jagati laadimisradadele. Kui laadimisrada avatakse hakkavad antud raja sõidukid 5 sekundiliste intervallidega laevale liikuma.

Esimeseks sihtmärgiks on NavMesh Agent'ile seatud vastavavalt sõiduki tüübile laevateki sisenemisala. Kui sõiduk on jõudnud laeva tekini hakkab ta otsima oma vaatevälja ulatuses rohelisi LED-tulesid. Lisaks kasutavad sõidukid Unity Linecast funktsionaalsust veendumaks, et LED-tuli ei oleks seina, lae või mõne teise objekti taga.

Linecast funktsioon võtab argumentideks algpunkti ja lõpp-punkti ja tuvastab, kas neid kahte punkti läbiva sirge vahel on objekt.

Leitud LED-tulede hulgast valitakse sihtmärgiks LED-tuli, millel on antud hetkel suurim kaugus autost. Sõidukid liiguvad sellisel viisil edasi kuni enam pole leida kaugemat LED-tuld, mis seada uueks sihtmärgiks, sellisel juhul jääb sõiduk peatuma viimasena sihtmärgiks valitud LED-tule all. Joonis 13 on kuvatõmmis simulatsioonis autotekil liikuvast sõidukist.



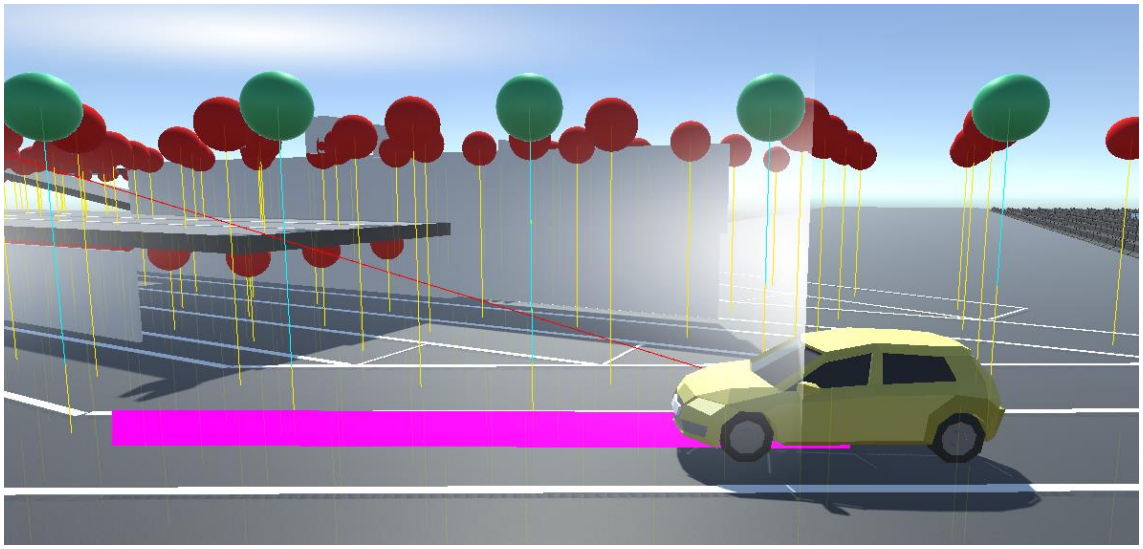
Joonis 13. Näide laeval navigeerivast sõidukist. Sõiduk võtab sihtmärgiks vaateulatuses oleva kaugeima rohelise LED-tule. Punane joon viitab LED-tulele mille sõiduk on endale sihtmärgiks valinud.

Kui sõiduki teele satub mõni teine sõiduk oleneb edasine käitumine sellest, kas eesolev sõiduk on juba peatunud või alles liigub. Liikuva sõiduki puhul muudame oma liikumiskiirust vastavalt eesolevale, et vältida kokkupõrkeohtu. Kui ees on peatunud sõiduk ja sõiduki vaatevälja ulatuses puuduvad kaugemad rohelised LED'id, mille poole liikuda, siis peatub sõiduk eesoleva sõiduki järel. Kui sõiduk on jõudnud oma sihtpunkti, märgitakse ka sõiduk ise automaatselt *NavMesh Obstacle*'ks, et järgmised sõidukid väldiksid neid.

Laeva on võimalik laadida läbi ahtri kui ka vööri. Simulaatoris sisenevad sõidukid vaikimisi laeva läbi vööri. Läbi ahtri laadimise testimiseks on võimalik kasutada suuna muutmise *dropdown* menüüd, mis pöörab laeva ümber y-telje 180 kraadi. Laadimissuuna muutmine muudab ka tekisensorite radade kaardistust – läbi vööri laadimisel peavad sõidukid mõnel juhul läbima teisi radu kui läbi ahtri laadimisel.

6.5 Simuleeritud sensorid

Sensoritel sõidukite tuvastamise võime andmiseks kasutame Unity CapsuleCast funktsionaalsust. CapsuleCast suunab algpunktist valitud suunda nn kapsli diameetriga 60cm ja tagastab kauguse lähimast objektist, millel on Collider komponent, mis jääb kapsli teele. Kõik sensorid suunavad CapsuleCast sirge otse alla. Kui sensori all pole mõnda teist objekti siis tagastab CapsuleCast kauguse teki põrandast, kui sensori all on mõni sõiduk siis tagastab CapsuleCast sensori kauguse autost. Joonis 14 on kuvatõmmis simulatsioonis sensorite all liikuvast sõidukist.



Joonis 14. Näide sensorite all liikuvast sõidukist.

Simulaatoris on võimalik sensorite LED-tulede valgust muuta sensoritel klikkides. Testimise hõlbustamiseks on võimalik simulaatoris radasid avada ja sulgeda ka raja kaupa. Selle saavutamiseks on kõik sensorid kaardistatud radadesse. Iga sensori juures on defineeritud, millistele radadele ta kuulub. Sensor võib kuuluda rohkem kui ühele rajale kuna osadele radadele pääseb ainult läbides laevateki teisi radu. Sellisel juhul peaksid ka laevateki teiste radade sensorid osaliselt rohelist tuld näitama, et sõidukeid õigele kohale juhatada. Lisaks oleneb sensorite kaardistus laadimissuunast.

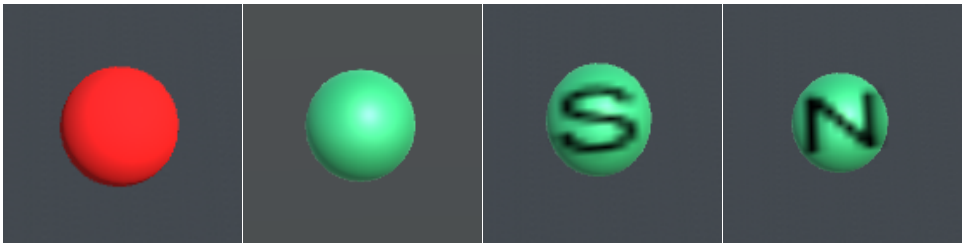
Laeva autotekki võib ultrahelisensorite jaoks pidada üsna mürarikkaks keskkonnaks. Autode pinnad, nagu tuuleklaasid on sensorite suhtes nurga all ja võivad tekitada valesid mõõtmistulemusi. Tekkidel liiguvad peale sõidukite ka reisijad ja laevatöölised, lisaks mõjutab ultrahelisensorite tööd temperatuuri kõikumine, teatud oludes võivad sensorid ka üksteise tööd segama hakata. Reaalsete sensorite genereeritud andmetest on näha, et teatud tingimustel genereerivad ultrahelisensorid isegi mõõtmisi, mis on suuremad kui

sensori kaugus põrandast. Ka simulatsioonikeskkonnas on võimalik sensorite mõõtetulemustesse müra lisada, et võimaldada sensorite mõõtmiste põhjal töötava autode tuvastamise algoritmi testimist mürarikkas keskkonnas. Selleks on simulaatorisse lisatud funktsionaalsus, mis võimaldab kasutajal määrata 0 kuni 10 protsendi vahel kui suur hulk sensorite mõõtmistest peaks müra olema. Iga mõõtmise ajal genereerib sensor juhuarvu vahemikus 0 kuni 100 ja kui juhuarv on väiksem kui kasutaja poolt määratud vea tekkimise protsent, siis genereerib sensor uue juhuarvu vahemikus 0 kuni 650, mis loetakse mõõtmise tulemuseks.

Peale üldise müra oleks vajalik simuleerida ka üksikute sensorite rikkeid. Selleks on simulatsioonikeskkonna sensoritel 3 töörežiimi, mille vahel on võimalik lülitada tehes sensoril paremklikk:

- *Normal* – Sensor töötab ootuspäraselt ja mõõdab kaugust autost.
- *Stuck* – Sensor jääb viimase mõõtmistulemuse peale kinni. Sensoril kuvatakse täht „S“ et visualiseerida hetkel aktiivne töörežiim.
- *Noise* – Sensor edastab ainult juhuslikke väärtusi. Sensoril kuvatakse täht „N“ et visualiseerida hetkel aktiivne töörežiim. Juhuslik mõõtetulemus genereeritakse sarnaselt eelnevalt kirjeldatule.

Sensoreid simulatsioonis eraldi ei visualiseerita, nii sensori kui LED-tule kujutamiseks kasutatakse ühte kera, mis võib olla kas punast või rohelist värvi, vastavalt sellele, kas rada on avatud või kas sensori all on juba mõni sõiduk parkinud. Selliselt käitumine vähendab visuaalset müra ja võimaldab kasutajal paremini LED-tulede seisu näha. Joonis 15 kujutab LED-tulede visualiseerimist simulatsioonis erinevate töörežiimides.



(a) (b) (c) (d)

Joonis 15. Simulatsioonis on LED-tuli ja sensor kujutatud ühe sfäärina.

Kujutatud on keelav LED-tuli (a), lubav LED-tuli (b), *Stuck* töörežiimis sensoriga lubav LED-tuli (c) ja *Noise* töörežiimis sensoriga lubav LED-tuli (d).

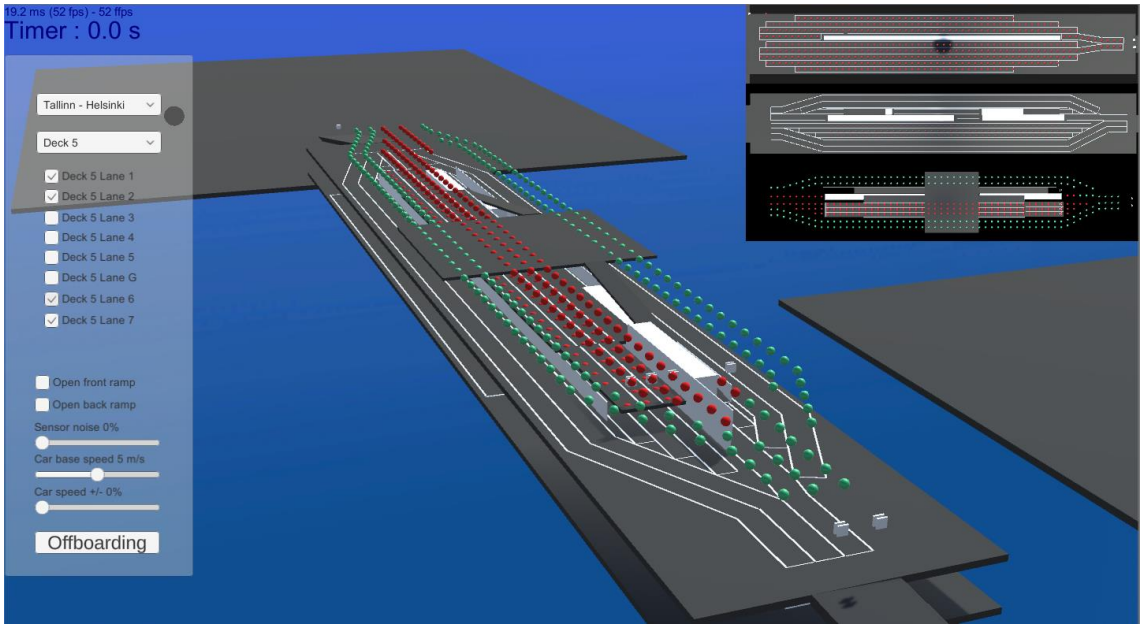
6.6 Simuleeritud suunatablood

Suunatablood simulatsioonikeskkonnas on lihtsad kuubikujulised mänguobjektid, mis on Transform komponendi abil risttahukateks muudetud. Suunatabloode mänguobjekti külgedele kuvatakse vastava juhise pilt. Suunatabloode uuendamiseks teeb simulatsioon kord sekundis päringu LaneControlProxy teenusele ja uuendab vastavalt kõik suunatablood. Joonis 16 kujutab suunatablood simulatsioonis.

Joonis 16. Näide suunatabloost simulatsioonis.

6.7 Simulatsiooni kasutajaliides

Unity’s simulatsiooni ehitamisel on eelis, et tulemuseks on intuitiivselt arusaadav 3D maailm. Joonis 17 on simulatsioonikeskkonna kuvatõmmis. Kasutaja saab hõlpsalt maailmas ringi liikuda ja interaktiivselt simulatsiooni käiku mõjutada.



Joonis 17. Simulatsioonikeskkonna kuvatõmmis.

Kasutajaliideses üleval paremas nurgas on eraldi kaameravaated, et paremini näha 3., 5. ja 6. autotekil toimuvat. Kasutajaliidese vasakus ääres on juhtpaneel, kust kasutajal on võimalik erinevaid seadistusi muuta.

- **Laadimissuuna valik** – kasutaja valib kas laadimine toimub ahter-vöör suunal või vöör-ahter suunal.

- **Tekiradade avamine ja sulgemine** – rippmenüüst on võimalik valida, millise teki radu kasutaja soovib juhtida. Seejärel saab kasutaja radu avada ja sulgeda kasutades vastava raja kõrval olevaid märkeruute.

- **6. tekile pääsemiseks kaldteede avamine ja sulgemine** – kasutaja saab kasutada vastavaid märkeruute.

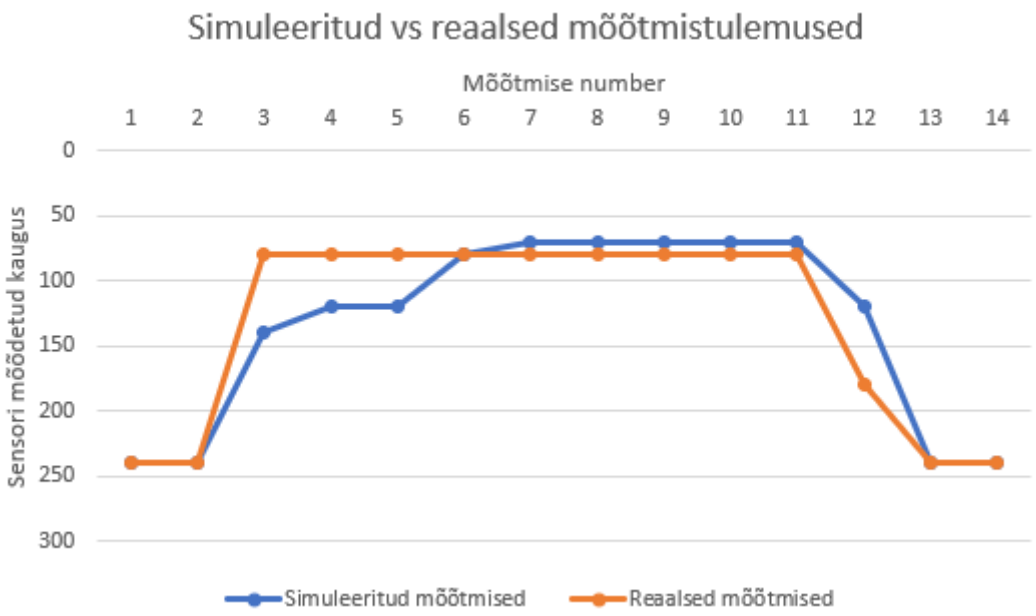
- **Sensorite müirataseme kontroll** – vaikimisi on valitud 0%.

- **Sõidukite baaskiiruse valik** – vaikimis on valitud 5m/s.

- **Sõidukite kiiruse juhukomponendi valik** – vaikimisi on valitud 0%.

6.8 Simulatsiooni ja reaalsete mõõtmistulemuste võrdlus

Megastar'i 5. autotekile on testimiseks paigaldatud 6 sensorit, mis kirjutavad oma mõõtmisi logifailidesse, seega on võimalik võrrelda simulatsiooni tekitatud andmeid reaalsete andmetega, et valideerida mudelite vastavus reaalsusega. Simulatsiooni poolt genereeritud mõõtmistulemuste uurimisel on näha, et kui sensori all pole autot liikumas töötavad nii simuleeritud sensor kui reaalne sensor identselt ja annavad mõõtmistulemuseks oma kõrguse laest, milleks on 240cm. Simuleeritud andmete puhul on selgelt näha kuidas auto möödumisel mõõdetud väärtused vastavad vastu kujule. Reaalsed andmed aga sellist täpsust ei paista omavat. Tulemuste võrdlemise teeb raskemaks asjaolu, et Targa Autoteki meeskonnal pole võimalik laeva laadimise ajal ise mõõtmistulemusi valideerida. Joonis 18 esitab simuleeritud ja reaalseid mõõtmistulemusi ühe sõiduki näitel.



Joonis 18. Näide simuleeritud ja reaalsetest mõõtmistulemustest ühe sõiduki möödumisel sensorist.

Reaalsed andmed sisaldavad ka oluliselt rohkem müra. Edaspidi tuleks simulatsioonide käitamisel sellega arvestada ja kasutada simulatsioonikeskkonnaga müra tekitamise funktsionaalsust. Teisest küljest tuleks uurida kas mõned autotekil müra tekitavad põhjused on eemaldatavad. Näiteks ultrahelisensorite üksteisele liiga lähedale paigutamisel võivad nad üksteise tööd segama hakata.

7 Sõidukite tuvastamise algoritmi testimine

Targa Autoteki projekti raames on valminud esimene iteratsioon algoritmist, mille eesmärk on tuvastada sõidukeid autotekil sensorite mõõtmiste põhjal. Algoritm loodi Targa Autoteki raames ja ei kuulu käesoleva töö hulka. See annab võimaluse demonstreerida simulatsiooni kasutamist süsteemi osade valideerimiseks ilma vajaduseta füüsilisi eksperimente läbi viia.

Sõidukite tuvastamise algoritm otsib mustreid sensorite mõõtmiste infost ja kirjeldab väljundis mitu mustrile vastavat jada mõõtmistest leiti. Joonis 19 on näide algoritmi väljundist.

```
Sensor:S1    S2    S3    S4    S5    S6    Description
-----------------------------------------------------------
Algo1: 6     1     6     8     2     7     (length)
Algo2: 6     1     6     8     1     6     (shape)
Algo3: 2     1     2     1     1     1     (stopped)
Algo4: 1     1     3     2     1     1     (slow)
```

Joonis 19. Näide sõidukite tuvastamise algoritmi väljundist. Veergudes on loetletud sensorid, ridades mustrid, mida algoritm otsib andmete seast. Kui vastava sensori andmetest leitakse vaste, suureneb tulemus 1 võrra.

Väärib märkimist, et töö kirjutamise hetkel on algoritmi väljundi trükkimisel viga, mis põhjustab autode loendamist 1st, selle asemel, et loendamist alustada 0st. Seega tuleb tulemuste tõlgendamisel tuvastatud sõidukite arvust lahutada 1. Selguse huvides teeme selle paranduse edaspidi kõikide väljundite kuvamisel ja kuvame tulemused tabelites. Tabel 2 on näide parandatud väljundist.

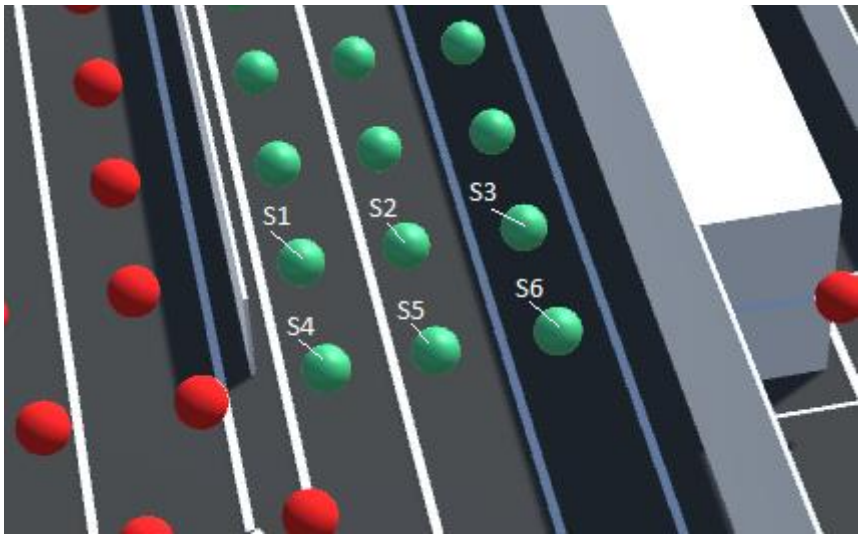
Tabel 2. Näide parandatud väljundist

	S1	**S2**	**S3**	**S4**	**S5**	**S6**	**Korrektne**
Algo1	5	0	5	7	1	6	Jah
Algo2	5	0	5	7	0	5	Jah
Algo3	1	0	1	0	0	0	Jah
Algo4	0	0	2	1	0	0	Jah

Järgnevalt selgitame erinevaid mustreid mida algoritmid andmetest otsivad.

- Algo1 otsib andmete hulgast sobiva pikkusega deviatsioone, ehk erinevusi sensori baaskõrgusest, milleks on sensori enda kõrgus teki põranda suhtes. Antud tekil on sensorid paigutatud 240cm kõrgusele.
- Algo2 üritab sõidukeid tuvastada sõiduki kuju järgi.
- Algo3 kontrollib kas sensori alla liikunud auto on sensori alla seisma jäänud
- Algo4 tuvastab aeglaselt liikuvaid sõidukeid.

Algoritmi jaoks testandmete genereerimisel kaardistati 6 sensorit simulatsioonis sarnaselt Megastar'ile paigaldatud sensoritele. Sensorid valiti 5. teki 3. 4. ja 5. rajalt, igal rajal 2 sensorit üksteise järel. Joonis 20 näitab visuaalselt kuidas sensorid simulatsioonikeskkonnas kaardistati.



Joonis 20. Sensorite kaardistamine eksperimentide jaoks.

Kõikide testide puhul liikus sensorite S3 ja S6 alt läbi 6 sõidukit. Genereeriti 7 erinevat jada testandmeid, mille jaoks kasutatud seadistused on välja toodud tabelis Tabel 3. Iga testi andmetega käitati sõidukite tuvastamise algoritmi ja märgiti tulemused üles. Järgnevalt arutame testimise tulemuste üle.

Tabel 3. Testitud stsenaariumid.

Testi number	**Sõidukite baaskiirus**	**Sensorite veaprotsent**	**Tulemus**
1	2 m/s	0%	Positiivne
2	5 m/s	0%	Positiivne
3	5 m/s (peatumisega sensori all)	0%	Negatiivne
4	8 m/s	0%	Positiivne
5	5 m/s	1%	Positiivne
6	5 m/s	3%	Positiivne
7	5 m/s	5%	Positiivne

Test 1 –Kõik sõidukid tuvastati korrektselt. Tasub märkida, et S3 on märkinud 2 sõidukit aeglaselt liikuvaks ja S6 on märkinud 1 sõiduki aeglaselt liikuvaks. Nagu eelnevalt kirjeldatud, aeglustavad sõidukid kui nad tuvastavad, et nende ees on teine liikuv sõiduk, see seletab miks mõned sõidukid on aeglaseks märgitud, kuid teised mitte. Tabel 4 kirjeldab teststsenaariumi 1 tulemusi.

Tabel 4. Teststsenaariumi 1 tulemused.

	S1	**S2**	**S3**	**S4**	**S5**	**S6**	**Korrektne**
Algo1	0	0	6	0	0	6	Jah
Algo2	0	0	6	0	0	6	Jah
Algo3	0	0	0	0	0	0	Jah
Algo4	0	0	2	0	0	1	Jah

Test 2 – kõik sõidukid tuvastati korrektselt, enam pole ükski sõiduk määratud aeglaselt liikuvaks. Tabel 5 kirjeldab teststsenaariumi 2 tulemusi.

Tabel 5. Teststsenaariumi 2 tulemused.

	S1	**S2**	**S3**	**S4**	**S5**	**S6**	**Korrektne**
Algo1	0	0	6	0	0	6	Jah
Algo2	0	0	6	0	0	6	Jah
Algo3	0	0	0	0	0	0	Jah
Algo4	0	0	0	0	0	0	Jah

Test 3 – antud tulemustest on näha, et sensorid loendavad sõidukeid korrektselt, kuid peatuvaid sõidukeid ei ole tulemuste märgitud. See vajab edasist uurimist. Tulevikus peab peatunud sõiduki kohal olev LED-tuli punaseks muutuma, seega on oluline, et sõidukite peatumine tuvastatakse korrektselt. Tabel 6 kirjeldab teststsenaariumi 3 tulemusi.

Tabel 6. Teststsenaariumi 3 tulemused.

	S1	**S2**	**S3**	**S4**	**S5**	**S6**	**Korrektne**
Algo1	0	0	6	0	0	5	Jah
Algo2	0	0	6	0	0	5	Jah
Algo3	0	0	0	0	0	0	Ei
Algo4	0	0	1	0	0	2	Jah

Test 4 - Sõidukite kiirem liikumine autotekil ei paista algoritmile probleeme tekitavat, kõik sõidukid on korrektselt tuvastatud. Tabel 7 kirjeldab teststsenaariumi 4 tulemusi.

Tabel 7. Teststsenaariumi 4 tulemused.

	S1	**S2**	**S3**	**S4**	**S5**	**S6**	**Korrektne**
Algo1	0	0	6	0	0	6	Jah
Algo2	0	0	6	0	0	6	Jah
Algo3	0	0	0	0	0	0	Jah
Algo4	0	0	0	0	0	0	Jah

Test 5 - Antud test lisas 1% müra kõikide sensorite mõõtmistele, see tähendab, et umbkaudselt 1% sensorite poolt saadetud väärtustest oli juhuslik väärtus. Nagu tulemustest näha siis selline müratase algoritmi tööd ei seganud ja kõik sõidukid said tuvastatud. Tabel 8 kirjeldab teststsenaariumi 5 tulemusi.

Tabel 8. Teststsenaariumi 5 tulemused.

	S1	**S2**	**S3**	**S4**	**S5**	**S6**	**Korrektne**
Algo1	0	0	6	0	0	6	Jah
Algo2	0	0	6	0	0	6	Jah
Algo3	0	0	0	0	0	0	Jah

Algo4	0	0	0	0	0	0	Jah

Test 6 - Antud testi puhul tõstsime müraprotsendi 3% peale kasutades sama meetodi nagu eelmises testis, nagu tulemustest näha ka see ei mõjutanud algoritmi tööd kuidagi. Kõik sõidukid tuvastati. Tabel 9 kirjeldab teststsenaariumi 6 tulemusi.

Tabel 9. Teststsenaariumi 6 tulemused.

	S1	**S2**	**S3**	**S4**	**S5**	**S6**	**Korrektne**
Algo1	0	0	6	0	0	6	Jah
Algo2	0	0	6	0	0	6	Jah
Algo3	0	0	0	0	0	0	Jah
Algo4	0	0	0	0	0	0	Jah

Test 7 - Antud testi puhul tõstsime müraprotsendi 5% peale kasutades sama meetodi nagu eelmises testis. Algoritm tuvastas endiselt kõik sõidukid korrektselt. See näitab, et algoritm töötab ka üsna mürarikkas keskkonnas. Tabel 10 kirjeldab teststsenaariumi 7 tulemusi.

Tabel 10. Teststsenaariumi 7 tulemused.

	S1	**S2**	**S3**	**S4**	**S5**	**S6**	**Korrektne**
Algo1	0	0	6	0	0	6	Jah
Algo2	0	0	6	0	0	6	Jah
Algo3	0	0	0	0	0	0	Jah
Algo4	0	0	0	0	0	0	Jah

Sõidukite tuvastamise algoritmi testimine kindlasti jätkub tulevikus kuna Targa Autoteki süsteem peab suutma täpselt kõikide autotekkide ulatuses laadimise edenemist jälgida. Selleks on oluline, et erinevate sensorite andmete komplekteerimisel süsteem loendaks kõik sõidukid korrektselt ja ei loendaks ühtegi sõidukit topelt. Eelnevate teststsenaariumite näitel - sensorid S3 ja S6 tuvastavad kumbki, et 6 sõidukit on nende alt läbi sõitnud. Sellises olukorras võib see tähendada, et autotekil viibib kokku 6 sõidukit. Kuid kui sensorid paigutada teisiti võib see ka tähendada, et autotekil viibib kokku 12 sõidukit.

8 Kokkuvõte

Töö raames loodi mudel Targa Autoteki süsteemi osade simuleerimise jaoks. Mudel koosneb Megastar laeva autotekkidest, nendele paigutatud sensoritest, suunatabloodest, laadimisalast ja sõidukitest, mis navigeerivad loodud aladel. Sensorite arv ja asukohad, sõidukite arv ja tüübid on lihtsalt konfigureeritavad, lihtsustades seeläbi erinevate stsenaariumite testimist. Töö võimaldab simuleerida erinevate sõidukite liikumist erinevatel kiirustel ja sensorite mõõtmist erinevatel laevatekkidel. Sellest genereeritud testandmeid on võimalik kasutada sõidukite tuvastamise algoritmi edasiarendamiseks, täiendamiseks ja testimiseks.

Unity on osutunud üsna heaks tööriistaks. Sisse ehitatud 3D füüsikamootor ja lihtne võimalus objekte programmeerida annavad Unity'le, kuid ka teistele mängumootoritele, palju kasutusvõimalusi ka teistes valdkondades peale mängude arenduse. Autori jaoks oli antud töö esimene kokkupuude 3D-mängumootoritega ja kuigi teatav õppekõver on olemas, on Unity kohta rohkelt dokumentatsiooni. Tänu suurele turuosale mängude arenduses on paljud probleemid ka teiste kasutajate poolt juba lahendatud, pakkudes nö käidud rada. Kõige rohkem probleeme tekitas sõidukite navigeerimisloogika ehitamine Unity poolt pakutud NavMesh tööriistu kasutades. Tegemist võib olla autori kogenematusega Unity platvormil töötamises, kuid kui see peaks ka tuleviku arendustes probleemiks osutuma on võimalik ka NavMesh asendada mõne kolmanda osapoole pakutud lahendusega.

Kuna Targa Autoteki projekt ise on alles arendamisfaasis ja lahtiseid küsimusi on mitmeid, siis vajab ka simulatsioonikeskkond kindlasti tulevikus muudatusi ja lisaarendusi. Tulevikus, peale sõiduki tekkide radade juhtimise automatiseerimist Targa Autoteki süsteemi poolt, on ka seda funktsionaalsust võimalik simulatsioone kasutades testida.

Kasutatud kirjandus

[1] “Tallink ja TalTech hakkavad ühiselt arendama tarka laeva,” 16 Jaanuar 2019. [Online]. Available: https://www.ttu.ee/tallink-ja-taltech-hakkavad-uhiselt-arendama-tarka-laeva.

[2] “D-terminali sõidukite check-in'i ala asukoht muutub,” 09 05 2018. [Online]. Available: https://www.tallink.ee/et/blogi/-/blogs/d-terminali-soiduautode-check-in-ala-asukoht-muutub. [Accessed 29 04 2020].

[3] S. Bandyopadhyay and R. Bhattacharya, “Introduction to Simulation,” in *Discrete and Continuous Simulation Theory and Practice,* Boca Raton, CRC Press, 2014, pp. 1-16.

[4] U. Wilensky and W. Rand, “What Is Agent-Based Modeling?,” in *An Introduction to Agent-Based Modeling: Modeling Natural, Social, and Engineered Complex Systems with NetLogo,* Cambridge, The MIT Press, 2015, pp. 21-43.

[5] “Modern manufacturing’s triple play: Digital twins, analytics and the internet of things,” *The Economist,* pp. https://expectexceptional.economist.com/digital-twins-analytics-internet-of-things.html.

[6] J. R. Juang, W. H. Hung and S. C. Kang, “Using game engines for physics-based simulations – A forklift,” *Journal of Information Technology in Construction,* vol. 16, pp. 3-22, 2011.

[7] “Docker,” [Online]. Available: https://www.docker.com/.

[8] “Docker concepts,” [Online]. Available: https://docs.docker.com/get-started/#docker-concepts. [Accessed 29 04 2020].

[9] Tark Autodekk, *Laevatekile paigaldatud sensori prototüüp,* Tallinn, 2020.

[10] MaxBotix Inc., “MB7060 XL-MaxSonar-WR,” [Online]. Available: https://www.maxbotix.com/Ultrasonic_Sensors/MB7060.htm. [Accessed 29 04 2020].

[11] Ampron, “Ampron - Model: DS-320x320,” Ampron, [Online]. Available: https://www.ampron.eu/product/model-type-ds-320x320-p8/. [Accessed 05 05 2020].

[12] “Unity,” [Online]. Available: https://unity.com/.

[13] “Efficient development of simulated environments for autonomous vehicle training,” [Online]. Available: https://unity3d.com/simulated-environments-for-autonomous-vehicle-training. [Accessed 29 04 2020].

[14] “Unity helps with crowd control in The Relative Worlds,” [Online]. Available: https://unity.com/case-study/craftar. [Accessed 05 05 2020].

[15] “SHoP Architects: A Unity Reflect case study,” [Online]. Available: https://unity.com/case-study/shop-architects. [Accessed 04 05 2020].

[16] J. Wang, L. Phillips, J. Moreland, B. Wu and C. Zhou, “Simulation and visualization of industrial processes in unity,” in *SummerSim '15: Proceedings of the Conference on Summer Computer Simulation,* Chicago Illinois, 2015.

[17] Y.-P. Chiu and Y.-C. Shiau, “Study on the application of unity software in emergency evacuation simulation for elder,” *Artificial Life and Robotics,* vol. 21, no. 2, p. 232–238, 2016.

[18] “Unity Simulation,” [Online]. Available: https://unity.com/products/simulation. [Accessed 29 04 2020].

[19] Unity, “High-performance physics in Unity 5,” [Online]. Available: https://blogs.unity3d.com/2014/07/08/high-performance-physics-in-unity-5/. [Accessed 29 04 2020].

[20] “Navigation System in Unity,” [Online]. Available: https://docs.unity3d.com/Manual/nav-NavigationSystem.html. [Accessed 29 04 2020].

[21] “Inner Workings of the Navigation System,” [Online]. Available: https://docs.unity3d.com/Manual/nav-InnerWorkings.html. [Accessed 29 04 2020].

[22] “NavMesh building components,” [Online]. Available: https://docs.unity3d.com/Manual/NavMesh-BuildingComponents.html. [Accessed 29 04 2020].

[23] D. M. Bourg and G. Seemann, “A* Pathfinding,” in *AI for Game Developers*, Sebastopol, O'Reilly Media, Inc., 2004, pp. 126-148.

[24] “Golang,” [Online]. Available: https://golang.org/.

[25] P. Costanza, C. Herzeel and W. Verachtert, “A comparison of three programming languages for a full-fledged next-generation sequencing tool,” *BMC Bioinformatics*, vol. 20, no. 301, 2019.

[26] “Low Poly Cars,” [Online]. Available: https://assetstore.unity.com/packages/3d/vehicles/land/low-poly-cars-101798.

[27] “Low Poly Street Pack,” [Online]. Available: https://assetstore.unity.com/packages/3d/environments/urban/low-poly-street-pack-67475.

[28] “Yughues Free Concrete Barriers,” [Online]. Available: https://assetstore.unity.com/packages/3d/props/exterior/yughues-free-concrete-barriers-13248.

[29] “TGU Skybox Pack,” [Online]. Available: https://assetstore.unity.com/packages/2d/textures-materials/sky/tgu-skybox-pack-96433.

[30] “UAA - City Props - Vehicles,” [Online]. Available: https://assetstore.unity.com/packages/3d/vehicles/land/uaa-city-props-vehicles-120339.

Lisa 1 – Targa Autoteki simulatsioonikeskkonna lähtekood

Simulatsioonikeskkonna lähtekood on saadaval aadressil https://bitbucket.org/kasparius/3dsmartdeck. Ligipääsu saamiseks võtke ühendust kaspar.lippmaa@gmail.com.

Lisa 2 – Kergesõidukite klassid

Laadimisraja klassi kood	**Kirjeldus**
Bus	Bussid
VanH	Kõrge kaubik
Van	Kaubik
VehLH	Pikk ja kõrge sõiduk
VehLM	Pikk, keskmise kõrgusega sõiduk
VehLL	Pikk, madal sõiduk
Car	Sõiduauto
CarXL	Katuseboksi või jalgrattaraamiga sõiduauto
CarGLD	Club one gold kaardiga sõiduauto
MCYCLE	Mootorratas(2x1,5x1,5) või külgkorviga mootorratas(4x1,9x2,5)
BCYCLE	Jalgratas
INVA	Erivajadustega sõiduauto
ALAND	Kõik sõidukid mis suunduvad Ålandile