LAPPEENRANNAN-LAHDEN TEKNILLINEN YLIOPISTO
School of Engineering Science
Laskennallisen tekniikan koulutusohjelma
Kandidaatintyö

Joonas Maljanen

Agenttipohjainen ruuhkan mallinnus sisätiloissa

Ohjaaja: Matylda Jablonska-Sabuka

TIIVISTELMÄ

Lappeenrannan-Lahden teknillinen yliopisto

School of Engineering Science

Laskennallisen tekniikan koulutusohjelma

Joonas Maljanen

Agenttipohjainen ruuhkan mallinnus sisätiloissa

Kandidaatintyö

2020

17 sivua, 9 kuvaa

Ohjaaja: Matylda Jablonska-Sabuka

Avainsanat: Agent-based Model; Real-time Systems; Crowd Simulation;

Agenttipohjainen liikehdinnän mallinnus on kasvava tieteenala, joka tietokoneiden kehittyessä on muodostunut tärkeäksi osaksi yksilöiden mallinnusta. Ruuhkan mallintaminen on yksi mallinnuksen sovelluskohteista, jota hyödynnetään palo- ja tartuntaturvallisuutta suunniteltaessa. Työn tarkoituksena on hyödyntää agenttipohjaista simulaatiota mallintamaan ja kartoittamaan ihmisten muodostamia ruuhkia annetuissa ympäristöissä. Työssä rajoitutaan tutkimaan ruuhkien muodostumista rajatuissa tiloissa ja käsittelemään yksilöitä yksinkertaistettuina malleina. Työ on toteutettu selainpohjaisella ohjelmistolla, joka sisältää kaiken mallinnuksesta tulosten visualisointiin. Ohjelmisto sisältää useita eri komponentteja reitinetsinnästä yksilöiden väistöreittien suunnitteluun. Tulokset osoittavan mallin toimivan odotetulla tavalla, ja tuottavan järkeviä tuloksia. Tuloksista on todettavissa rakennuksen puolittavan sisäänkäynnille keskittyvän ruuhkan, kun sisäänkäyntejä on kaksi yhden sijaan. Ohjelmisto ja malli toimivat odotetusti, ja kykenevät mallintamaan yksinkertaisia ympäristöjä. Ohjelmisto on valmis suuntaa antavaksi työkaluksi kiinteistöjen suunnittelussa.

Sisällys

1 JOHDANTO

Viime vuosikymmenenä väkijoukon mallinnuksesta on tullut tärkeä tutkimuksen ala infrastruktuuri-, kaupunki- ja turvallisuussuunnittelussa, sekä viihdeteollisuudessa (Almeida et al. 2013). Aiheen tutkimus ja kehitys ovat kiihtyvässä tahdissa, ja agenttipohjainen mallinnus on johtava keino luoda tarkkoja ja vakaita malleja yksilöiden liikehdinnästä ja käyttäytymisestä (Livio 2020). Agenttipohjaisen joukkoliikehdinnän aikaisimpia teoksia on Reynolds 1987, jossa lähestytään agenttipohjaista liikehdintää tutkimalla lintu- ja kalaparvien, sekä muiden eläinlaumojen laumaliikehdintää. Tietokoneiden kehittyessä on pystytty agenttipohjaista mallinnusta hyödyntämään ruuhkan analysoinnissa, jota hyödynnetään hätäpoistumisreittien suunnittelussa ja infektiotautien leviämisen mallintamisessa (Ahmad et al. 2018 Livio 2020). Työn tarkoituksena on tutkia miten agenttipohjainen malli implementoidaan, jonka pohjalta on toteutettu toimiva kokonaisuus, jota voi käyttää ruuhkien mallintamiseen. Työ koostuu teoriasta, mallin rakenteesta ja tuloksien käsittelystä.

1.1 Tutkimusongelma, tavoitteet ja rajaus

Simulaation tavoitteena on simuloida ihmisyksilöiden luonnollista ja päämäärätietoista liikehdintää erilaisissa tiloissa, ja tehdä analyysiä agenttien välisistä vuorovaikutuksista. Luonnollinen liikehdintä on rajattu huomioimaan tilojen rakenteet ja muiden agenttien liikeradat, jättäen huomiotta sosiaaliset kanssakäymiset, ryhmäliikehdinnän ja muut haasteellisemmat konseptit.

Tutkimuksen taustajatuksena on hyödyntää analysoitua ruuhkaa infektiotautien leviämisen tutkimukseen. Etäisyys on yksi merkittävimmistä influenssatautien leviämiseen vaikuttavista seikoista (WHO 2019), joten sen tutkimiseen keskittymisellä saa osviittaa todellisen taudin leviämisestä.

1.2 Tutkimusmetodologia

Tutkimuksen tarkoituksena on hyödyntää agenttipohjaista simulaatiota analysoimaan tilaratkaisujen vaikutusta ruuhkien muodostumiseen. Malli on agenttipohjainen simulaatio, jossa agentit kulkevat deterministisen reitin väistellen vastaantulevaa ja risteävää liikennettä. Agentit pyrkivät noudattamaan luonnollista liikehdintää, huomioimatta ryhmäkäyttäytymistä tai monitavoitteisia reittejä. Simulaatio on selainpohjainen ohjelmisto, joka on ajettavissa

nykyaikaisella selaimella. Simulaatioon haluaman pohjapiirrustuksen voi tuoda python aputyökalulla.

2 LIIKEHDINNÄN MALLINNUS

Simulaatio on selaimella ajettava JavaScript-ohjelma, jossa agentit liikkuvat päämääriään kohti väistellen toisia yksilöitä ja noudattaen annettuja rajoitteita ja rajoituksia. Simulaatio koostuu monesta osasta jotka ovat esitelty kuvassa 1.

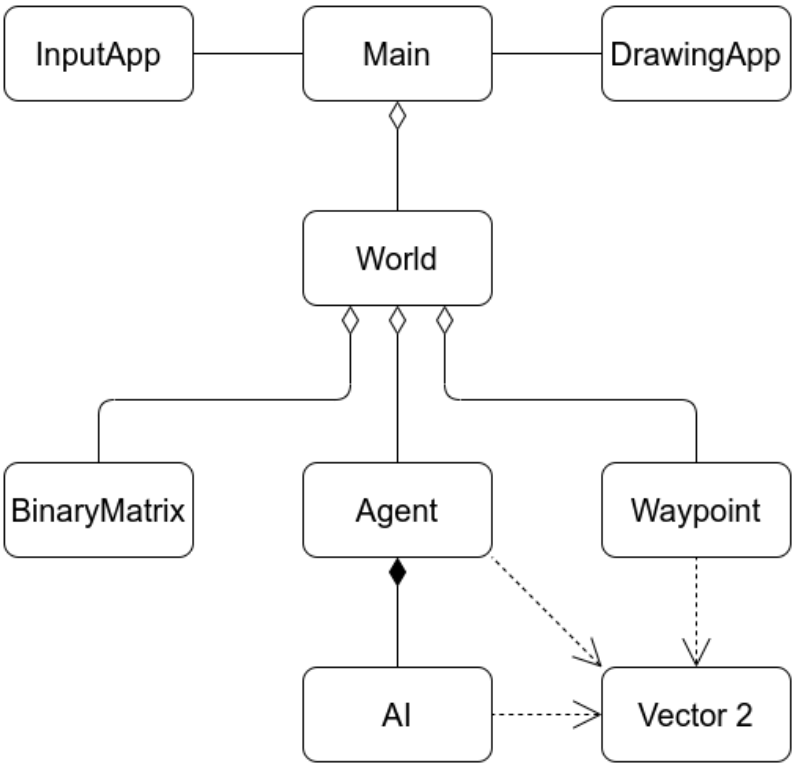
Simulaation hierarkiassa ylimmällä tasolla ovat pääohjelma Main, grafiikkaliitymä DrawingApp ja käyttäjän syötteistä vastaava InputApp. Main vastaa simulaation päivityssykleistä, ja toimii rajapintana käyttäjän ja simulaation välillä. DrawingApp vastaa simulaation graafisesta päivittämisestä, ja siitä miten jokainen simulaation elementti piirretään. InputApp vastaa käyttäjän syötteiden hallinnoinnista ja kutsuu pääohjelma Mainia annettujen syötteiden mukaisesti.

Pääohjelma Main alapuolella on instanssi varsinaisesta simuloidusta maailmasta World, joka sisältää tiedon kaikesta mitä simuloitu maailma sisältää ja hallinnoi sitä tietoa. Hallittaviin asioihin kuuluvat agentit, esteet ja agenttien reittipisteet. Agentit hallitsevat omaa älykkyyttään AI ja ovat varsinaiset simuloitavat yksilöt, jotka AI:n ohjaamana kulkevat simuloidussa maailmassa. Esteet ovat matriisi, jonka solujen arvot ilmaisevat vastaavan simuloidun maailman ruudun rajoituksia. Reittipisteet ovat pisteitä joiden välillä agentit kulkevat, jotka kuvastavat yksilöille tärkeitä pisteitä kuten myymälät, ruokalat, WC-tilat ja luokkahuoneet.

Monen luokan toiminnallisuus vaatii tiedon sijainnista ja suunnasta, jotka ovat Vector2 instansseja. Vector2 on rakenne kaksiulotteiselle vektorille jolla on vektorin yleisimmät ominaisuudet, kuten: sijainti, magnitudi ja suunta. Vector2 sisältää myös useita funktioita joilla voi toteuttaa yleisimpiä laskutoimituksia, kuten: vektorien kertominen skalaarilla, kahden vektorin summaus ja kertominen ja yksikkövektorin muodostaminen.

2.1 Graafiikan toteutus

Simuloitava maailma piirtyy DrawingApp kautta HTML canvas- elementille, joka päivittyy käyttäjän valitsemalla päivitystaajuudella. Simulaatio tukee reaaliaikaista visualisointia, jossa piirretty maailma päivittyy käyttäjän antaman kuvaa sekunnissa -arvon mukaisesti. Toinen tuettu muoto on intervallikuvaus (eng. time-lapse), jossa käyttäjän antaman ajan välein simuloidun maailman tilanne päivittyy.



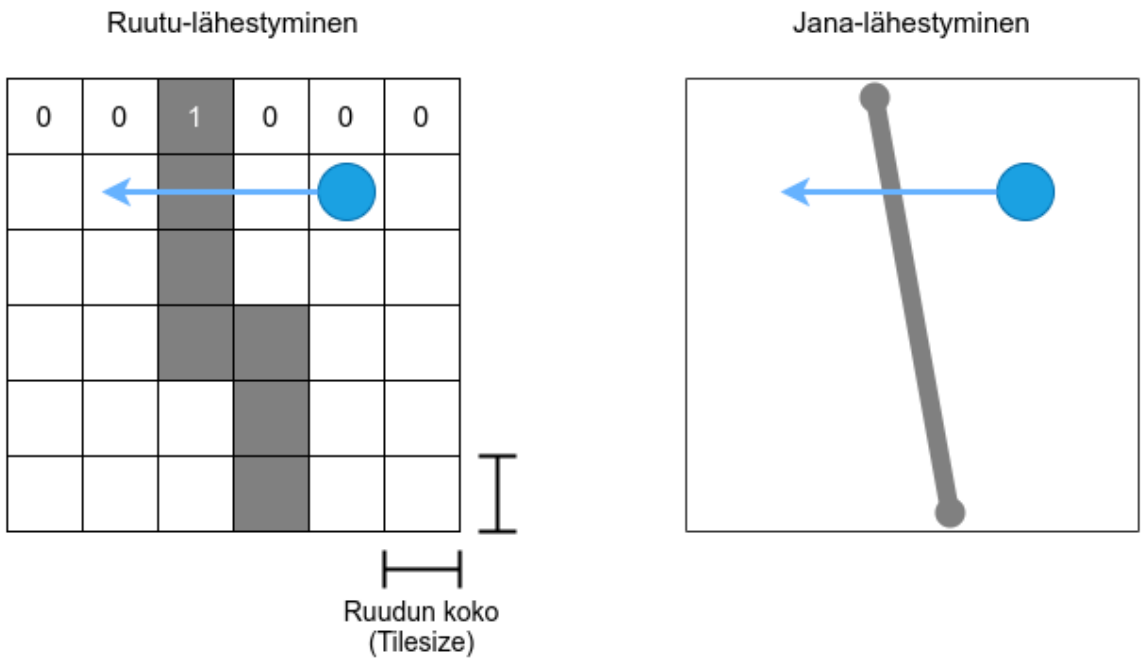
Kuva 1. Ohjelman luokkakaavio

2.2 Agenttien liikehdinnän rajoittaminen

Agenttipohjaisissa malleissa on keskeistä rajoittaa esteillä agenttien liikehdintää, jotta rakennusten ja tilojen mallintaminen olisi mahdollista. Liikehdinnän rajoittamisessa on käsitelty kahta yksinkertaista lähestymistapaa: ruutupohjainen ja janapohjainen lähestyminen, joilla on omat hyvät ja huonot puolensa, joiden toimintaperiaatteet on esitelty kuvassa 2.

Ruutupohjaisessa lähestymisessä simuloitava pinta-ala jaetaan vakiokokoisiin ruutuihin, jotka tallennetaan matriisiksi. Matriisi on aluksi vapaasti liikuttava, eli nollamatriisi, mutta luodessa uusia seiniä ja esteitä merkitään kyseiset kielletyt ruudut ykkösiksi. Sallitut liikkeet ja vapaat ruudut löytyvät muuntamalla halutut koordinaatit ruudukkomuotoon, jonka antamalla rivinumerolla ja sarakenumerolla saadaan matriisista haluttu alkio. Alkion arvon ollessa 0, on kyseinen ruutu vapaa ja arvon ollessa 1.

Ruutupohjaisen systeemin vahvuutena on matala laskennallinen aikakompleksisuus, joka mahdollistaa nopean ja tehokkaan simuloinnin. Ruutupohjaisessa systeemissä esteruutujen määrä ei vaikuta aikakompleksisuuteen *O(1)*, eli on aikakompleksisesti vakio. Ainoa merkitsevä tekijä on matriisin koko, joka tulee vastaan etsiessä haluttua solua. Prosessi jossa et-



Kuva 2. Esimerkkitilanne esteestä ja agentin liikkumisesta ruutu- ja janalähestymisellä

sitään solu matriisista on kompleksisuudeltaan $O(log\ n^2)$ binäärihakupuulla, jossa n on matriisin rivit kertaa sarakkeet, ja jos matriisi koostuu lineearisista listoista on hakukompleksisuus $O(n^2)$.

Ruutupohjaisen systeemin heikkoutena on ruudun koko, johon nähden simulaation mikään agenteista ei saa ottaa askelta joka olisi isompi kuin ruudun koko, muuten agentti astuisi ruudun yli. Ruudun koko myös tulee ongelmaksi jos halutaan teräviä muotoja, kuten kaarevia seiniä jolloin on käytettävä pientä ruutukokoa, joka puolestaan kasvattaa hakukompleksisuutta ja pienentää turvallisen askeleen pituutta.

Janapohjaisessa lähestymisessä simulaatio käsittelee listaa estejanoja, jotka esittävät esteitä liikkeelle eli seiniä. Estejanat sisältävät tiedon janan alkupisteen ja loppupisteen koordinaateista joiden välille estejana pingotetaan. Sallitut liikkeet ja suunnat lasketaan pingottamalla uusi jana liikkeen alkupisteen ja loppupisteen välille, ja jos tämä jana leikkaa jotain estejanaa on liike kielletty.

Janapohjaisen systeemin vahvuutena on luotavien esteiden vähäiset rajoitteet. Estejanan suunta ja pituus ovat täysin rajoittamattomat, eikä niillä tule vastaan ongelmia vaikka agenttien askeleenpituus kasvaisi. Esteanoilla voi luoda äärettömän ohuita ja kaltevia esteitä ilman

suurempia rajoitteita, toisin kuin ruutupohjaisella lähestymisellä.

Janapohjaisen systeemin heikkoudet ovat suoritustehollisia heikkouksia. Agenttien liikkuessa täytyy jokaisen agentin käydä läpi kaikki simulaatiossa esiintyvät estejanat, ja tutkia jokaiselle erikseen estejanan ja liikkeen leikkaavuutta. Estejanojen läpikäymisen kompleksisuus on *O(n)*, jossa *n* on estejanojen määrä simulaatiossa. Suoritustehollisesti suurin ongelma on kahden janan leikkaavuuden laskeminen, joka käyttää kahta determinanttia jotka yhdistettynä sisältävät 12 vähennyslaskua ja 4 kertolaskua. Vaikka kyseiset operaatiot eivät ole suoranaisesti raskaita, ovat ne enemmän kuin ruudukkopohjaisessa.

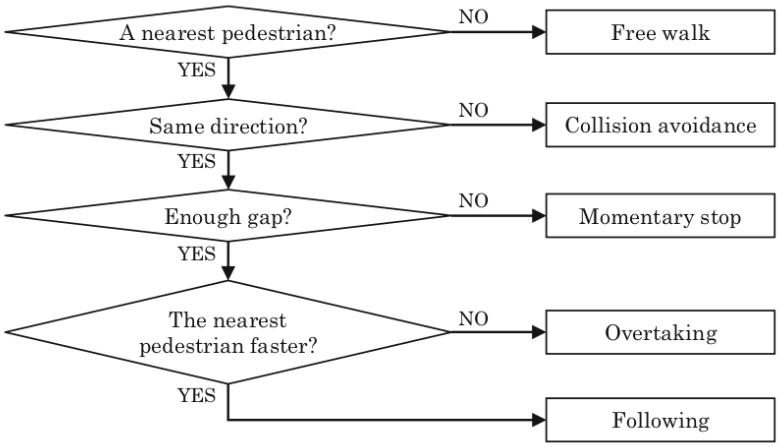
Tämän työn toteutuksessa on hyödynnetty ruutulähestymistä, sillä se takaa esteiden lukumäärästä riippumattoman suoritusajan, ja on laskennallisesti kevyempi kuin janapohjainen. Vaikka ruutupohjainen lähestyminen rajoittaa agenttien askeleenpituutta, ei se ole tule ongelmaksi, sillä pienemmät askelkoot simulaatioissa parantavat simulaation tarkkuutta, ja siten ovat edullisia simulaatiolle (Kuo et al. 2012).

2.3 Agenttien ohitustilanteet

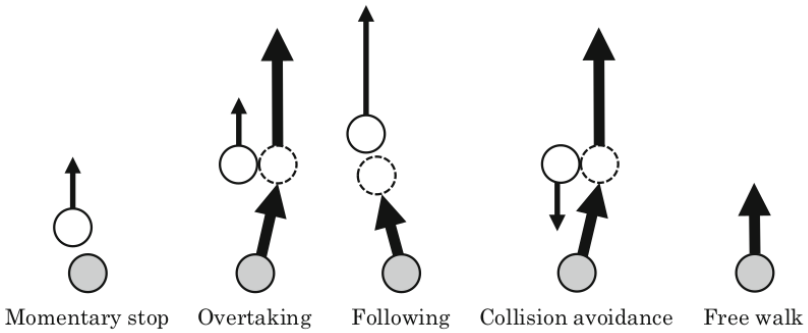
Ohitustilanteet ja niiden mallintaminen ovat keskeinen osa agenttipohjaisia simulaatioita, sillä ne mahdollistavat luontevan joukkokäyttäytymisen (Livio 2020). Ohitustilanteiden perustana toimii Crowd Dynamics, vol.2 esittämä tapa, jossa agentit tunnistavat kulkusuuntaansa lähinnä olevan toisen agentin, ja käyvät läpi päätöspuun määrittäen parhaan tavan edetä. Etenemistapoja on yhteensä 5 kappaletta, joista jokaiselle on oma toteutumisehto, jotka ovat visualisoitu kuvissa 3 ja 4.

- Vapaaliike, *Free Walk*, on liike jossa agenttin kulkusuunnassa ei ole muita agentteja, joten agentin liikettä ei rajoiteta.
- Törmäyksen välttely, *Collision avoidance*, on liike jossa pyritään väistämään edessä oleva mutta vastakkaisuuntainen agentti väistämällä oikealta.
- Hetkellinen pysähdys, *Momentary stop*, on liike jossa kulkusuunnassa lähimpänä olevaan agenttiin ei ole tarpeeksi välimatkaa, jolloin täytyy pysähtyä odottamaan.
- Ohitus, *Overtaking*, on liike jossa kulkusuunnassa samaan suuntaan kulkeva agentti on hitaampi, jolloin agentti ohitetaan oikealta.

- Seuraaminen, *Following*, on liike jossa kulkusuunnassa samaan suuntaan kulkeva agentti on nopeampi, jolloin seurataan agenttia.

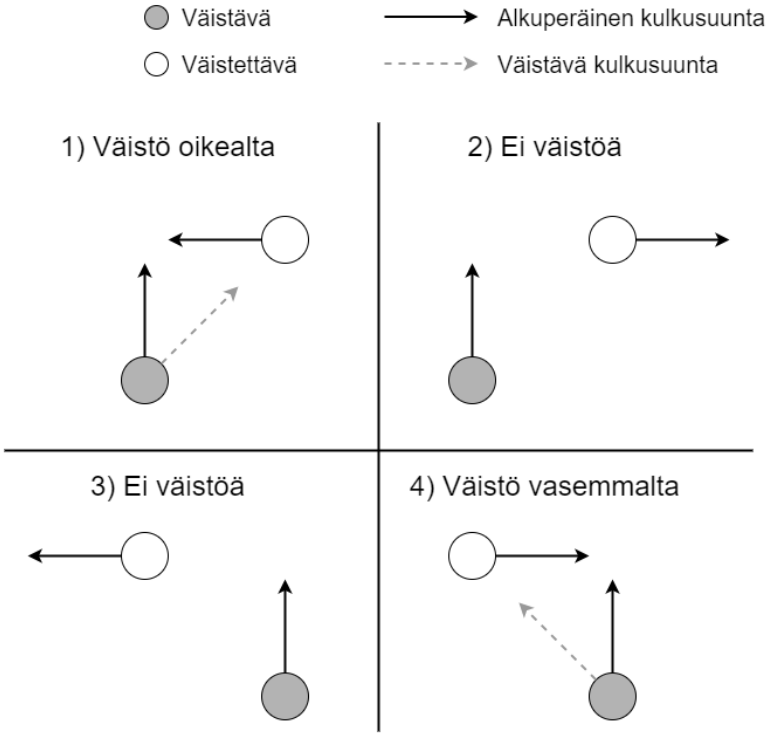


Kuva 3. Ohitustilanteen päätöspuu (Livio 2020)



Kuva 4. Ohitustilanteen ohitustyylit (Livio 2020)

Simulaation toteuttama ohitus käyttää muunneltua versiota edellä esitellystä Gibelli Livio:n versiosta, jossa otetaan huomioon sivusuunnasta tapahtuvat väistötilanteet. Sivusuunnasta tapahtuvissa väistötilanteissa väistösuuntaan vaikuttavat väistettävän sijainti ja suunta väistävän menosuuntaan nähden. Pistetuloa hyödyntäen määritetään onko väistettävä väistäjän oikealla vai vasemmalla puolella, sekä onko väistettävän liikesuunta väistävästä oikealle vai vaisemmalle. Näillä arvoilla lasketaan XOR-ehto (Eksklusiivinen disjunktio / ”joko tai”), ja ehdon toteututessa väistetään suuntaan josta väistävä tulee. Eri kombinaatiot ovat esitelty kuvassa 5.



Kuva 5. Ohitus sivusuunnasta

2.4 Agenttien reitinetsintä

Reitinetsintä on oleellinen osa agenttipohjaista simulaatiota, jonka avulla agentit löytävät niille osoitettuihin maalipisteisiin. Simulaation reitinetsintä-algoritmina toimii A*, joka on yleisesti käytetty ja tehokas graafin läpikäyvä algoritmi (Thalmann et al. 2013). A* reitinetsinnän pohjana on tavoitehakuinen heurestiikka, jossa graafin solmuja painotettaan perustuen solmun etäisyyteen maalista H, ja solmun saavuttamiseen kumuloituneiden matkakustannuksien G avulla, joiden summa antaa solmun painoarvon $G+H=F$ (Delling et al. 2009).

Simulaatiossa graafina toimii matriisi, joka edustaa simuloitavaa maailmaa. Graafissa voi liikkua ruudusta jokaiseen kahdeksasta naapriruudusta, jotka ovat vapaina, eli eivät ole läpipääsemättömiä Matriisin ruutujen arvot kuvaavat ruutujen painoarvoja, jossa ruudun arvo 0 kuvastaa läpipääsemätöntä estettä ja 1 tai sitä suuremmat arvot ovat vapaasti kuljettavia, siten että ruudun arvon kasvaessa ruudun haastavuus kasvaa, ja reitinetsintä-algoritmi pyrkii välttämään niitä.

A* algoritmin implementoinnissa on käytetty matriisia joka ylläpitää tietoa käydyistä ruuduista. Ruudun tieto sisältää tiedon ruudusta jonka kautta kyseiseen ruutuun on päästy ja tiedon onko ruutu avoin vai suljettu. Implementaatio sisältää myös itseorganisoituvan listan, jossa alkiot ovat suuruusjärjestyksessä painoarvon mukaan järjestettyinä. Reitin löydyttyä

algoritmi palauttaa takaperin jäljitetyn reitin, joka muodostuu lähtemällä maalin saavuttaneesta ruudusta aina sen ruudun edelliseen ruutuun, kunnes lähtöpiste on saavutettu.

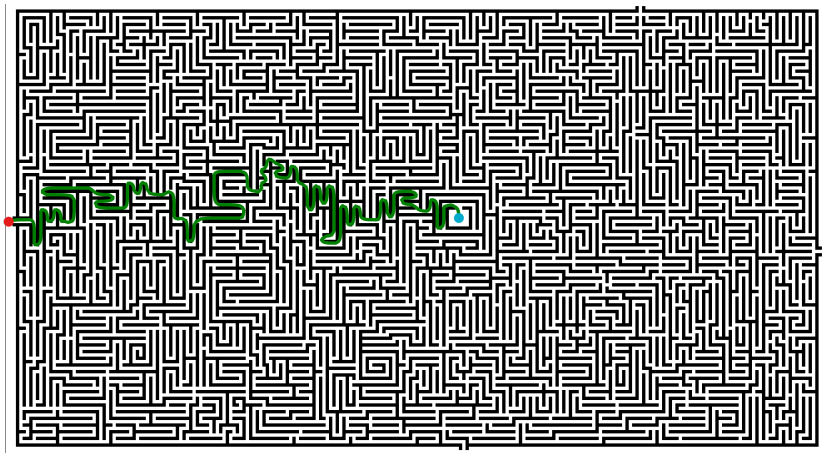
Implementoitu algoritmi avattuna:

1. Aloittava ruutu merkitään matriisissa avonaiseksi ja lisätään listaan
2. Toistetaan kunnes avonaisia ruutuja ei enää ole
    (a) Valitaan pienimmän painoarvon omaava ruutu listasta pivot ruuduksi
    (b) Merkitään ruutu suljetuksi
    (c) Tarkistetaan onko nykyinen ruutu maaliruutu, jos on jäljitetään reitti ja poistutaan algoritmista.
    (d) Etsitään kaikki naapuriruudut jotka eivät ole läpipääsemättömillä ruuduilla
    (e) Jokaiselle naapuriruudulle:
        i. Lasketaan naapuriruudun painoarvo ja merkitään sitä edeltävä ruutu sen vanhemmaksi
        ii. Jos naapuriruudussa ei ole käyty, merkitään se matriisiin avonaiseksi ja lisätään listaan
        iii. Jos naapuriruutu on suljettu siirrytään seuraavaan naapuriin
        iv. Jos naapuriruutu on avoin, ja jo olemassa olevan painoarvo on suurempi kuin nykyisen korvataan se uudella

Algoritmin tehokkuutta on testattu tietokoneen luomalla labyrintilla (Ilkant et al. 2006). Luodun implementaation reitinetsintä nopeus on 4-6ms, ja tiheissä sokkeloissa 20-40ms. Testaukset tuottivat nopeudellisesti samanlaisia tuloksia kuin A* -kirjaston JavaScriptiin luonut Brian Grinstead (Grinstead 2012). Käyttäen Grinsteadin A* -kirjastoa vertailukohtana voi todeta simulaation implementoinnin olevan tehokas ja nopea (kuva 6).

Algoritmin tehokkuutta voi analysoida käytettyjen tietorakenteiden ja algoritmien aikakompleksisuuksilla, joista yleisimmät ovat valmiiksi analysoitu kuten verkkosivulla (Rowell 2016). Implementoinnissa merkkittävä tehokkuus on saavutettu käsittelemällä avoimia ja suljettuja ruutuja matriisina, jolloin alkion etsintä matriisista on vakio aikakompleksisuudelta *O(1)*. Kompleksisuus on huomattavasti matalampi kuin tilanteessa jossa alkiot ovat epäjärjestetyssä listassa, jolloin pahimmassa tilanteessa täytyy lista käydä kokonaan läpi löytääkseen haetun alkion *O(n)*, kuten on toteutettu naivissa A* -implementaatiosa (Delling et al. 2009). Toinen tietorakenteen optimointi on avoimien ruutujen ylläpitäminen matriisin lisäksi itseorganisoituvassa listassa. Lista organisoi itseään alkioden syöttövaiheessa, käymällä listaa läpi

kunnes löytyy sijainti jossa syötetty alkio on suuruusjärjestyksellisesti kohdallaan, tämän jälkeen lista on aina suuruusjärjestyksessä. Optimointi on tärkeää, koska algoritmin vaiheessa 1.(a) tarvitaan pienin avoin ruutu, ja mitä tehokkaammin tämä ruutu löytyy sen nopeampi on algoritmi.



Kuva 6. A* ratkaisu 24ms suoritusajalla labyrintissä (labyrintti otettu Ilkant et al. 2006)

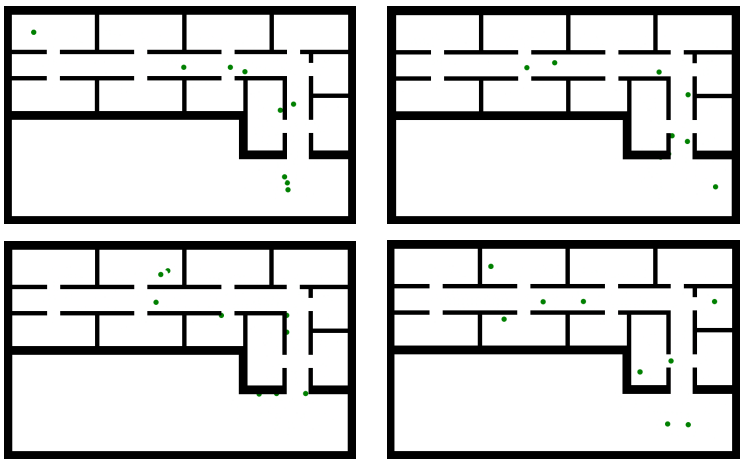
2.5 Ohjelmistot

Simulaation ohjelmisto on tuotettu Visual Studio Code -ohjelmointiympäristöllä. VS Code on kevyt ohjelmisto joka sisältää monipuolisen valikoiman lisäosia, kuten oikeinkirjoituksen tarkistus ja debuggeri. Tuotettu lähdekoodi on viety GitHub versionhallintaan GitHub Desktop työpöytäohjelmistolla, joka samalla pitää huolen lähdekoodin varmuuskopioinnista.

Lähdekoodi käännetään selaimen ymmärtämäksi JavaScriptiksi TypeScript kääntäjän avulla. Nykyaikaisiin selaimiin sisäänrakennettu JavaScript engine ajaa luodun JavaScriptin ja siten ajaa simulaatiota. Pohjapiirrustuksen tuonti ohjelmistoon on toteutettu python ohjelmalla, joka muuntaa kuvatiedoston JavaScriptille sopivaan muotoon.

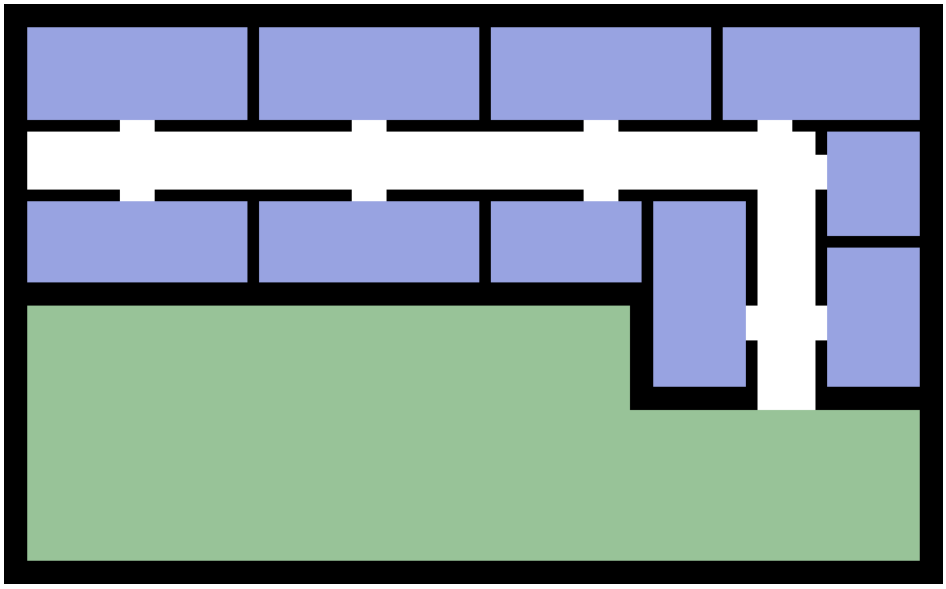
3 TULOKSET

Tutkimuksessa käytetty rakennus on yksinkertainen kiinteistö, jossa on piha-alue ja varsinainen rakennus (kuva 8). Agentit aloittavat pihalta käyden yhdessä sattumanvaraisessa huoneessa, josta palaavat pihalle ja poistuvat simulaatiosta. Simulaatiota ajettiin tuhansia iteraatioita, jotta lopullisen ruuhkakartan data tasaantuisi, ja kohina poistuisi. Kuvassa 7 on otettu neljä kuvaa, joissa on nähtävissä miten agentit liikkuvat simulaatiossa.

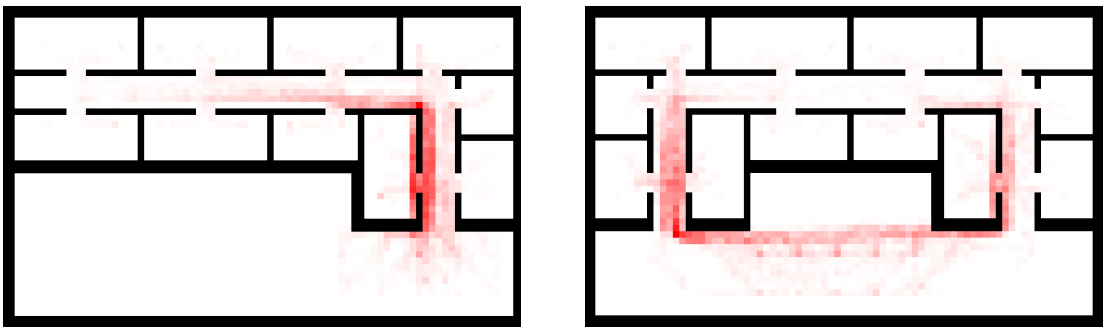


Kuva 7. Otoksia simulaation aja-aikana

Ruuhkakartoista (kuva 9) on havaittavissa ruuhkan keskittyvän kiinteistön sisäänkäynneille, josta vähitellen laskee syvemmällä kiinteistössä. Tulos on järkevä huomioiden jokaiseen huoneeseen kohdistuvan agenttien osalta yhtäläisen verran kulkua, ja mentäessä syvemälle kiinteistöön, täytyy kulkea reittiä jota myös muihin huoneisiin kulkevien täytyy kulkea. Saaduilla tuloksilla rakennus kahdella sisäänkäynnillä vähentäisi ruuhkaa huomattavasti enemmän, kuin rakennus yhdellä sisäänkäynnillä.



Kuva 8. Rakennuksen pohjapiirrustus. Siniset ovat huoneita ja vihreä piha-alue



(a) Yksi sisäänkäynti

(b) Kaksi sisäänkäyntiä

Kuva 9. Ruuhkakartat

4 KESKUSTELU JA JOHTOPÄÄTÖKSET

Työssä tutkittiin kiinteistöjen rakenteellisten ratkaisujen vaikutusta ruuhkautumiseen. Työn pohjana on selainpohjainen simulaatio, joka mallintaa agenteilla yksilöiden luonnollista liikehdintää annetuissa tiloissa. Agentit kykenevät yleisimpiin väistöliikkeisiin ja navigoivat käyttäen A* reitinetsintäalgoritmia, mutta ylläpitääkseen liikkehdinnän sulavuutta eivät pidä seiniä läpipääsemättöminä esteinä. Simulaation tuottamat ruuhkakartat ovat järkeviä ja johdonmukaisia, joista käy selkästi ilmi useamman sisäänkäyntien määrän vaikutus ruuhkautumiseen.

Työssä keskityttiin kuvaamaan luonnollista liikehdintää, joten monimutkaisemmat ryhmäliikehdinnät ja -käyttäytymiset jätettiin huomiotta. Yksiöiden ryhmäkäyttäytyminen vaikuttaa suuresti yksilöiden luonnoliseen mallintamiseen, mutta sen toteutus on laaja ja monimutkainen ongelma. Oletuksena oli myös agenttien käyvän vain yhdessä huoneessa, jonka jälkeen poistuvat rakennuksesta samaa reittiä kuin mistä tulivat, jonka vuoksi monimutkaisempia liikeratoja ei pääse muodostumaan. Työssä ei myöskään toteutettu käyttöliittymää simulaation konfigurointiin, tai tulosten yksinkertaiseen taltioimiseen.

Tuotettu simulaatio on toimiva työkalu apuvälineeksi kiinteistöjen analysoinnissa, jonka avulla ruuhkaisat alueet ovat helppoja kartoittaa jo kiinteistön suunnitteluvaiheessa. Nykyisessä muodossaan simulaatio ei kykene luotettavasti kuvaamaan monimutkaisia tiloja tai kompeksista käyttäytymistä. Jatkokehityksen kohteita olisi tuoda malliin ryhmäkäyttäytymista, agenttien rutiineiden syventämstä ja monipuolisempia tavoitteita agenteille ja ryhmille. Tekniikan puolelta jatkokehitystä olisi siirtää varsinaisen simulaation ajaminen esimerkiksi python-ympäristöön, jolla saisi lisää laskennallista tehokkuutta malliin. Jatkokehityksillä parannettaisiin mallin tarkkuutta ja käytettävyyttä.

Lähteet

Ahmad, I. S., Sun, S. ja Boufama, B. (2018). "Agent-based Crowd Simulation Modeling in a Gaming Environment". Teoksessa: *2018 6th International Conference on Multimedia Computing and Systems (ICMCS)*, s. 1–6. DOI: 10.1109/ICMCS.2018.8525969.

Almeida, João, Rossetti, Rosaldo ja Coelho, António (maaliskuu 2013). "Crowd Simulation Modeling Applied to Emergency and Evacuation Simulations using Multi-Agent Systems".

Delling, Daniel, Sanders, Peter, Schultes, Dominik ja Wagner, Dorothea (2009). "Engineering Route Planning Algorithms". Teoksessa: *Algorithmics of Large and Complex Networks: Design, Analysis, and Simulation*. Toim. Jürgen Lerner, Dorothea Wagner ja Katharina A. Zweig. Berlin, Heidelberg: Springer Berlin Heidelberg, s. 117–139. ISBN: 978-3-642-02094-0. DOI: 10.1007/978-3-642-02094-0_7. URL: https://doi.org/10.1007/978-3-642-02094-0_7.

Grinstead, Brian (2012). *A* Search Algorithm in JavaScript*. URL: https://briangrinstead.com/blog/astar-search-algorithm-in-javascript/.

Ilkant ja Mysid (2006). *Computer generated maze*. URL: https://commons.wikimedia.org/wiki/File:Cg_pp_maze.png.

Kuo, Chia-Tung, Wang, Da-Wei ja Hsu, Tsan sheng (2012). "A Simple Efficient Technique to Adjust Time Step Size in a Stochastic Discrete Time Agent-based Simulation." Teoksessa: *SIMULTECH*. Toim. Nuno Pina, Janusz Kacprzyk ja Mohammad S. Obaidat. SciTePress, s. 42–48. ISBN: 978-989-8565-20-4. URL: https://www.iis.sinica.edu.tw/papers/wdw/13816-F.pdf.

Livio, Gibelli (2020). *Crowd Dynamics, Volume 2 Theory, Models, and Applications*. 1. painos. Vol. 2. An optional note. Web: Cham : Springer International Publishing. ISBN: 3-030-50450-6. URL: https://link-springer-com.ezproxy.cc.lut.fi/book/10.1007%2F978-3-030-50450-2.

Reynolds, Craig W. (elokuu 1987). "Flocks, Herds and Schools: A Distributed Behavioral Model". *SIGGRAPH Comput. Graph.* 21.4, s. 25–34. ISSN: 0097-8930. DOI: 10.1145/37402.37406. URL: https://doi-org.ezproxy.cc.lut.fi/10.1145/37402.37406.

Rowell, Eric (2016). *Big-O Algorithm Complexity Cheat Sheet*. URL: https://www.bigocheatsheet.com/.

Thalmann, Daniel ja Musse, Soraia Raupp (2013). *Crowd Simulation*. eng. 2. Aufl. London: Springer Verlag London Limited. ISBN: 9781447144496.

WHO (2019). *Advice for the public on COVID-19*. Accessed: 2021-05-18. URL: https://www.who.int/emergencies/diseases/novel-coronavirus-2019/advice-for-public.