Sami Huoppila, Sampo Pekkanen, Elmo Vahvaselkä

Ruokakaupan jonossa

Simulaatioprojekti

Metropolia Ammattikorkeakoulu
Tieto- ja viestintätekniikka
Ohjelmistotuotanto
Ohjelmointiprojekti
24.10.2021

Sisällys

Liitteet

Liite 1: Javadoc-dokumentaatio

Liite 2: Linkki SVN-repositorioon

1 Johdanto

Tämä dokumentti esittelee Java-kielellä kehitettyä ohjelmistoprojektia, joka on toteutettu Metropolia Ammattikorkeakoulun ”Ohjelmointiprojekti TX00CD79-3013” -kurssia varten. Projektin aiheena oli simulaatio, joka mallintaa kaupan asiakasvirtoja painopisteen ollessa kassoihin liittyvissä suorituskykysuureissa. Tehtävänanto edellytti, että ohjelma toteutetaan kolmivaihesimulointina. Myös tämä dokumentti on osa arviointia.

Kappaleessa kaksi esitellään tarkemmin projektin visio ja tavoite, jonka jälkeen kappale kolme tarjoaa perehdytyksen dokumentissa käytettävään terminologiaan.

Neljännessä kappaleessa havainnollistetaan toteutusta käsitteellisesti sekä esitellään millaisia tietoja käyttäjä voi syöttää ohjelmaan ja millaista dataa ohjelmasta saadaan ulos. Lisäksi kerrotaan millaisia oletuksia ja yksinkertaistuksia mallia luodessa on käytetty sekä havainnollistetaan visuaalisesti mallin toimintaa prosessikaavion avulla.

Ohjelmointiteknistä toteutusta esitellään kappaleessa viisi. Käytettyjen kielien ja kirjastojen kautta siirrytään ohjelman arkkitehtuuriin, jonka jälkeen kuvaillaan käyttöliittymän ja sisäisen logiikan toteutusta sekä esitellään tietovaraston toiminta. Kappaleen päätteeksi kuvaillaan, miten testaus toteutettiin.

Simulaattorin käyttöohje löytyy kappaleesta kuusi. Kappaleessa seitsemän esitellään, millaisia tuloksia ja johtopäätöksiä simulaation ajoilla on saavutettu. Lopuksi kappale kahdeksan viimeistelee dokumentin yhteenvedolla, jossa esitellään myös potentiaalisia jatkokehitysideoita.

2 Visio

Tavoitteena oli kehittää yksinkertaistettu malli ruokakaupasta ja simuloida sen asiakasvirtoja. Erityisenä kiinnostuksen kohteena ovat asiakkaiden kesimääräiset jonotusajat ja läpimenoajat palvelupisteillä sekä näiden palvelupisteiden käyttöaste. Tarkoituksena on myös havainnoida, miten asiakastiheys ja käytössä olevien kassojen määrä vaikuttaa näihin suureisiin.

Käyttäjälle on tarkoitus tarjota selkeästi luettavaa dataa, jota voidaan hyödyntää aukiolevien kassojen määrän mitoituksessa ja käyttää erilaisten kassatyyppien hyödyllisyyden arvioinnissa. Simulaation ajosta saadut tulokset tallennetaan tietokantaan.

Käyttäjä pystyy syöttämään simulaatioon erilaisia parametreja kuten simuloinnin kokonaisajan, simulointinopeuden, käytössä olevien kassojen määrän ja ruuhkan määrän. Kun simulaatio on käynnistetty, voi simuloinnin edistymistä seurata visuaalisesti. Simulointinopeutta on mahdollista muuttaa kesken simulaation.

Simulaatio tulee sisältämään paljon yksinkertaistuksia niin asiakkaiden ja kuin palvelupisteiden suhteen. Palvelupisteitä ovat kassat ja erilaiset palvelustikit kuten esimerkiksi lihatiski. Kassojen tyypit ovat tavallinen, itsepalvelu ja pika. Asiakkaiden saapumisten väliajat ja palveluajat generoidaan hyödyntäen erilaisia jakaumia ja käyttäjän syötteitä.

Jokaisella asiakkaalla on ostoslista, joka vaikuttaa kaupassa asioimisen pituuteen ja mahdollisiin asiointeihin palvelutiskeillä. Hyllyiltä tapahtuvaa yksittäisten tuotteiden noutamista ei simuloida. Isompi ostosmäärä kasvattaa myös palveluun kuluvaa aikaa kassalla. Mikäli asiakas ei voi maksaa kortilla, hän ei kykene vierailemaan itsepalvelukassalla.

Simulaation päätteeksi tulokset tallennetaan tietokantaan ja halutessaan käyttäjä pystyy tarkastelemaan aikaisempien ajojen tuloksia sekä poistamaan yksittäisen ajon tiedot tietokannasta.

3 Käsitteet, määritelmät

Asiakkaalla tarkoitetaan kaupassa asioivaa henkilöä.

Palvelupisteellä tarkoitetaan kassoja tai kaupan palvelutiskejä, jotka toimivat FIFO-jonotusperiaatteella (First in First Out) eli asiakkaat palvellaan saapumisjärjestyksessä.

Haahuilu (H) tarkoittaa asiakkaan ostosten keräilyä kuvaavaa aikaa. Haahuilu on toteutettu palvelupisteenä. Muista palvelupisteistä poiketen, asiakas viipyy palvelupisteellä ennalta määrätyn ajan. Aika määräytyy ostoslistan mukaan ja on muista asiakkaista riippumaton. Eli Haahuilu ei noudata FIFO-periaatetta.

Tavallinen kassa (K) on kassa, jossa henkilökunta hoitaa tuotteiden skannaamisen ja rahastamisen. Kaikki maksutavat käyvät.

Itsepalvelukassa (IP) tarkoitetaan kassaa, jossa asiakas itse hoitaa tuotteiden skannaamisen ja maksamisen. Palveluaika on hieman muita kassoja hitaampi. Asiakkaan on maksettava kortilla.

Pikakassa (PK) eroaa tavallisesta kassasta siten, että siihen voi mennä korkeintaan viiden ostoksen kanssa. Kaikki maksutavat käyvät.

Palvelutiski on palvelupiste, josta asiakas voi hakea erilaisia tuotteita. Näitä ovat liha-, kala- ja juustotiski.

Ostoslista kertoo, montako tuotetta asiakas ostaa kaupasta ja millä palvelutiskeillä hän vierailee. Ostoslistan pituus vaikuttaa kaupassa vietettyyn aikaan.

Palveluajalla tarkoitetaan aikaa, jonka asiakas viettää jollain tietyllä palvelupisteellä palveltavana.

MVC-malli (Model View Controller). Ohjelman arkkitehtuurityyppi, jossa simulaation malli, käyttöliittymä ja nämä toisiinsa liittävä kontrolleri on eritelty toisistaan.

DAO-malli (Data Access Object) on malli, jossa tietokantaratkaisu on ohjelmoitu omaksi kerrokseksi ja se peittää alleen kaikki tietovaraston yksityiskohdat.

4 Käsitteellinen malli

4.1 Tavoite

Tavoitteena on havainnoida asiakasvirtoja ja miten mahdolliset muuttujat, kuten asiakkaiden saapumistiheys tai auki olevien kassojen määrä vaikuttaa ruuhkautumiseen.

Käyttäjä pystyy antamaan erilaisia lähtöarvoja ja seuraamaan simulaation edistymistä. Simulaation päätteeksi käyttäjälle tarjotaan tietoa simulaation tuloksista ja nämä tulokset tallennetaan tietokantaan.

4.2 Syötteet

Simulointiaika määrittää simulaation kokonaisajan sekunneissa. Arvon tulee olla väliltä 1000-50000s.

Viive määrittää simulointinopeuden, kun simulaatio käynnistetään. Arvon tulee olla väliltä 0-3000ms.

Normikassojen määrä asettaa asiakkaiden käytössä olevien normikassojen määrän. Arvon tulee olla väliltä 1–5.

Pikakassojen määrä asettaa käytössä olevien pikakassojen määrän. Arvon tulee olla joko 1 tai 2.

Itsepalvelukassojen määrä asettaa käytössä olevien itsepalvelukassojen määrän. Arvon tulee olla väliltä 1–4.

Korttimaksun todennäköisyys määrittää millä prosenttitodennäköisyydellä yksittäisellä asiakkaalla on maksukortti mukana. Arvon tulee olla väliltä 0–100.

Ruuhkanmäärä kaupassa määrittää asiakkaiden saapumistiheyden. Käyttäjä voi valita kolmesta arvosta: hiljaista, normaalia ja ruuhkaa. Nämä sanalliset arvot antavat eksponentiaaliselle jakaumalle arvon 100 (hiljaista), 50 (normaalia) tai 20 (ruuhkaa).

4.3 Tulosteet

4.3.1 Simulaation yleiset tiedot

Simulaation ID on yksilöllinen tunnus, jolla eri simulointikertojen tulokset voi erottaa toisistansa.

Simulaation PVM kertoo päivämäärän ja ajan, jolloin simulaatio on suoritettu.

Asiakaslukumäärä kertoo, kuinka monta asiakasta simulaation aikana on saapunut kauppaan.

Kokonaisaika ilmoittaa simuloidun ajan pituuden.

Ruuhkan määrä kertoo sanallisesti, kuinka tiheään asiakkaita on saapunut kauppaan.

4.3.2 Palvelupisteen tiedot

Tyyppi kertoo, onko palvelupisteen tyyppi kassa tai liha-, kala- tai juustotiski.

Kassatyyppi kertoo, onko kassan tyyppi normaali, pika tai itsepalvelu. Palvelutiskien kohdalla arvo on tyhjä.

Keskimääräinen palveluaika kertoo kuinka pitkään yhden asiakkaan palveleminen keskimäärin kesti.

Keskimääräinen jononpituus kertoo, montako asiakasta jonossa on ollut keskimäärin simulaation aikana.

Keskimääräinen läpimenoaika kertoo kuinka pitkään yksi asiakas vietti keskimäärin palvelupisteellä mukaan lukien jonotusaika.

Käyttöaste havainnoi käytön suhdetta kapasiteettiin.

Palvellut asiakkaat kertoo, kuinka monta asiakasta palvelupiste on palvellut.

4.4 Mallin toiminta

Malli jäljittelee yksinkertaistetusti reaalimaailmasta kelloa, asiakkaita, kassoja ja palvelutiskejä. Kelloa hyödynnetään tapahtumien ajoittamiseen ja suorituskykysuureiden mittaamiseen.

Asiakkaiden saapumisajat lasketaan eksponentiaalisen jakauman avulla, jonka arvo riippuu käyttäjän syötteistä. Jokaisella asiakkaalla on ostoislista, jonka koko määrittää asiakkaan kaupassa viettämän ajan. Lisäksi ostoslista vaikuttaa siihen, millä palvelutiskeillä asiakas vierailee.

Palvelutiskeillä kuluva aika generoidaan hyödyntämällä normaalijakaumaa, jolle on keskiarvoksi annettu arvo 60 ja varianssiksi arvo 200. Negatiivisia arvoja ei kuitenkaan hyväksytä. Arvo generoidaan tarvittaessa niin monta kertaa uudestaan, että luku on positiivinen.

Palveluaika kassoilla laskentaan normaalijakauman ja ostosten määrän avulla. Ostosten määrä kerrotaan satunnaisgeneraattorin antamalla arvolla. Tähän arvoon lisätään vielä

15 sekuntia, joka kuvaa maksuun kuluvaa aikaa. Normi- ja pikakassoilla normaalijakaumalle annetaan keskiarvoksi 5 ja varianssiksi 2. Pikakassalla keskiarvo on 7 ja varianssi 3. Pikakassalla voi asioida vain, mikäli tuotteitta on viisi tai vähemmän. Mikäli asiakas maksaa käteisellä, ei hän voi käyttää itsepalvelukassaa.

Korttimaksun todennäköisyyttä säätämällä voidaan mallintaa tilanteita, joissa korttimaksu ei esimerkiksi teknisen vian tai yksittäisen pankin ongelmien takia toimi lainkaan tai toimii vain osittain.

4.5 Oletukset ja yksinkertaistukset

Simulaatio toimii monilta osin yksinkertaistetusti. Asiakkaiden käytöksestä oletetaan, että he tulevat kauppaan tekemään vain ostoksia. Kaupassa oletetaan olevan aina riittävä määrä tuotteita. Saapuessaan kauppaan asiakkaat keräävät ensin kaikki ostokset ja vasta sitten vierailevat mahdollisilla palvelutiskeillä. Palvelutiskeillä asiakkaan oletetaan vierailevan tietyssä järjestyksessä. Ensimmäinen prioriteetti on lihatiskillä, toinen kalatiskillä ja viimeinen juustotiskillä.

Asiakas valitsee kassoista sen, jossa on lyhyin jono ja johon hänen on mahdollista mennä, kun huomioidaan ostosten määrän ja maksutavan tuomat rajoitukset. Asiakas ei koskaan yritä mennä vääräntyyppiselle kassalle.

Aukiolevien kassojen määrä pysyy samana koko simulaation ajan. Myöskään ruuhka-aikoja ei ole huomioitu. Lisäksi simulaatio pysähtyy käytännössä heti kun simuloinnille asetettu aikaraja on saavutettu. Toisin sanoen jonossa olevia asiakkaita ei palvella, kun aikaraja on saavutettu.

Huomionarvoista on myös, että simulaatiossa käytettävien jakaumien arvot eivät tarkasti mallinna todellisuutta. Tämän projektin pääpainona oli ohjelmistotekninen toteutus eikä matemaattinen mallintaminen.

4.6 Mallin kuvaus

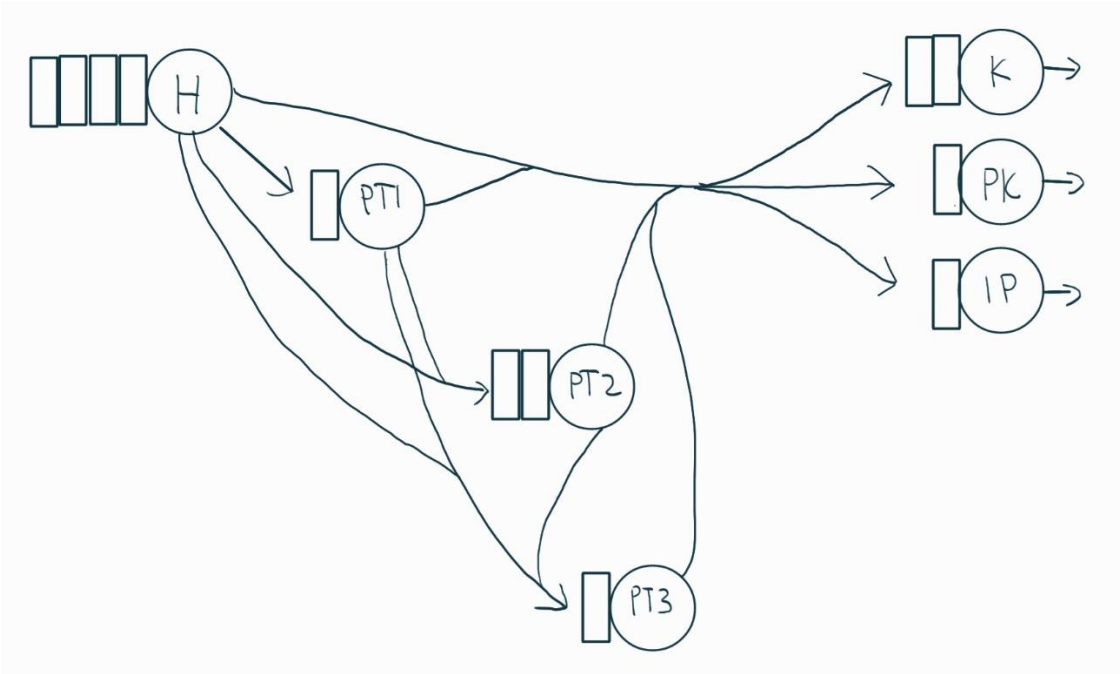
4.6.1 Komponenttilista

Komponentti	**Yksityiskohtia**
Asiakas	Saapumisaikojen tiheys noudattaa eksponentiaalista jakaumaa.
Palvelupisteiden jonot	Palvelupisteiden jonot ovat rajattomat.
Palvelupisteet	Palvelupisteet ovat joko kassoja tai kaupan palvelutiskejä. Palvelupisteille arvotaan palveluajat eksponenttijakauman mukaisesti.

4.6.2 Prosessikaavio

Kuviossa 1 havainnollistetaan kaupan simuloinnin toimintaa prosessikaaviolla. Jokainen kassatyyppi eli tavallinen kassa, pikakassa ja itsepalvelukassa on kuvattu vain kerran. Käyttäjä voi kuitenkin asettaa itse käytössä olevien kassojen määrän. Palvelutiskien määrään käyttäjä ei pysty vaikuttamaan.

Kauppaan saapuvat asiakkaat aloittavat vapaasti keräämällä tuotteita. Seuravaksi asiakas siirtyy palvelutiskeille, mikäli ostoslistassa on tuotteita, jotka edellyttävät tiskillä vierailemista. Kun asiakas on kerännyt kaikki tuotteet ja suorittanut mahdolliset palvelutiskivierailut, siirtyy hän jollekin kolmesta kassatyypistä ja lopuksi poistuu kaupasta.



Kuvio 1. Prosessikaavio kaupan simulaatiosta

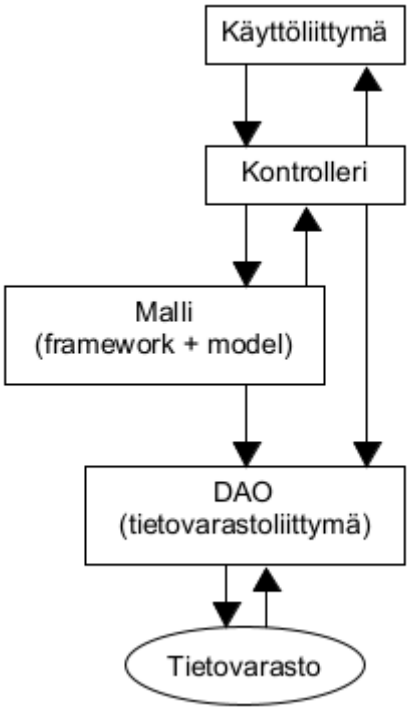
5 Mallin ohjelmointitekninen toteutus

5.1 Käytetyt ohjelmointikielet ja kirjastot

Ohjelma on kirjoitettu käyttäen Java-kieltä ja graafisen käyttöliittymän toteutukseen on käytetty JavaFX-kirjastoa. Tietokanta on toteutettu käyttämällä Hibernate-kirjastoa, jota hyödynnetään datan tallentamisessa SQL-pohjaiseen tietokantaan. Yksikkötestaus toteutettiin käyttämällä JUnit 5-kirjastoa.

5.2 Arkkitehtuuri

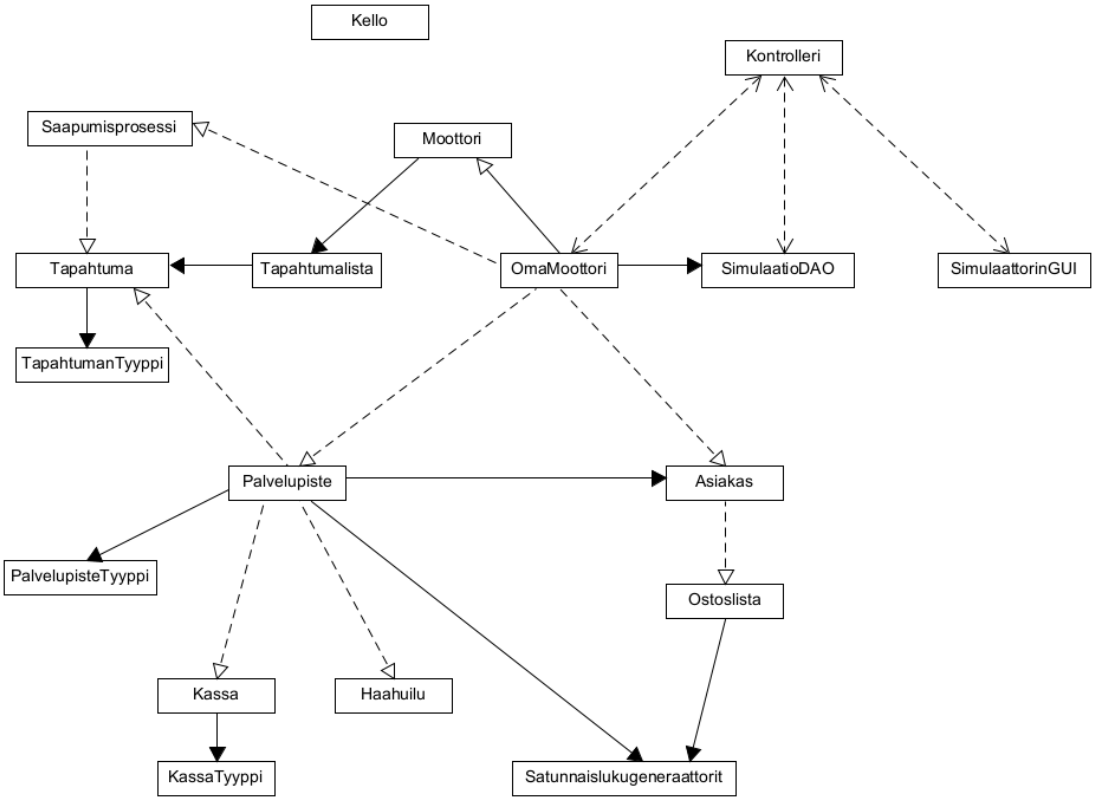
Kuviossa 2 esitellään simulaattorin arkkitehtuuria. Ohjelman toiminta on jaettu useampaan kerrokseen MVC-mallin perusteella (Model View Controller), johon on yhdistetty DAO-mallia (Data Access Object) hyödyntävä tietovarastoliittymä, joka tallentaa tietoa MySQL-tietokantaan. Täten ohjelmasta tulee modulaarinen ja yksittäistä osa-aluetta voidaan muokata koskematta muiden osa-alueiden koodiin.



Kuvio 2. Simulaation kerrosarkkitehtuuri

Käyttöliittymä (view) on yhteydessä malliin (model) ja tietovarastoliittymään kontrollerin (controller) kautta. Simulaation päätteeksi malli kuitenkin automaattisesti tallentaa simulaation tiedot ja on suoraan yhteydessä tietovarastoliittymään. Luokkien väliset tarkemmat suhteet esitellään kuviossa 3. Simulaation mallin kehys (framework) on saatu valmiina kurssin opettajalta ja siihen on tehty vain pieniä muutoksia.

(21)



Kuvio 3. Luokkakaavio mallin rakenteesta

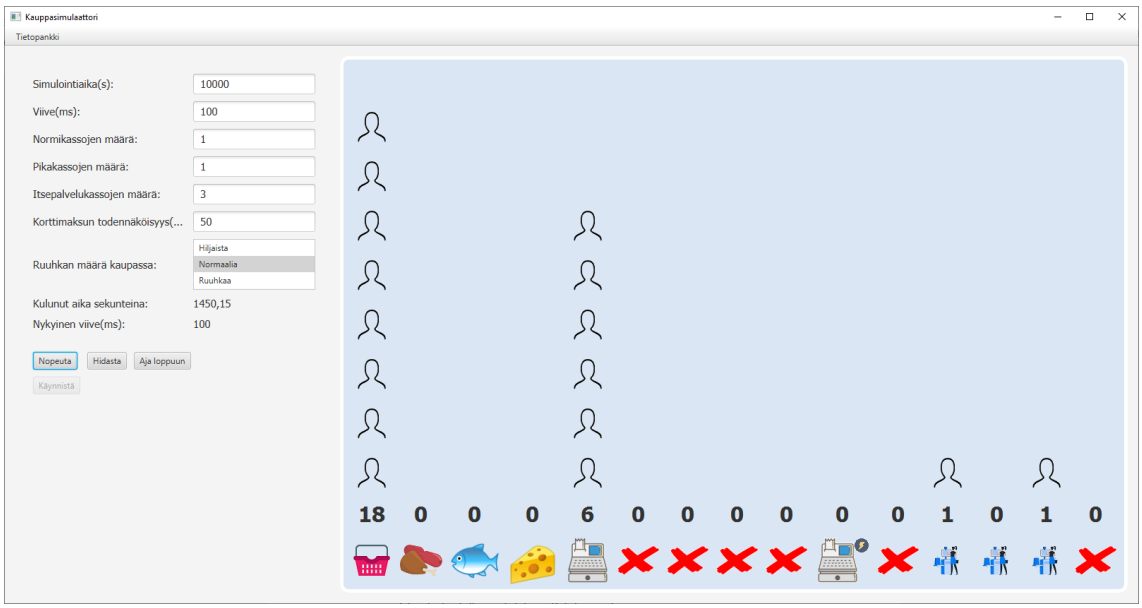
5.3 Käyttöliittymän kuvaus

Ennen simulaation käynnistämistä käyttäjän tulee antaa arvot simulaation parametreille. Mikäli käyttäjän syöttämät parametrit eivät ole annetuissa rajoissa, käynnistä-painiketta painaessa käyttäjä saa virheilmoituksen, joka ilmoittaa, mikä arvo on virheellinen ja kehottaa muuttamaan kyseisen arvon.

Mikäli arvot ovat annetuissa rajoissa, käyttäjä voi käynnistää simulaation painamalla käynnistä-painiketta. Käynnistyksen yhteydessä käyttöliittymä kutsuu kontrolleria, joka luo OmaMoottori-olion annetuilla parametreilla ja täten käynnistää simulaation.

Simulaation ollessa käynnissä, asiakkaat visualisoidaan palvelupisteille jonoon kuvakkeina. Aina kun asiakas saapuu tai poistuu jonosta, OmaMoottori välittää tiedon muutoksista kontrollerin kautta käyttöliittymälle. Visualisoidun jonon maksimipituus on kah-

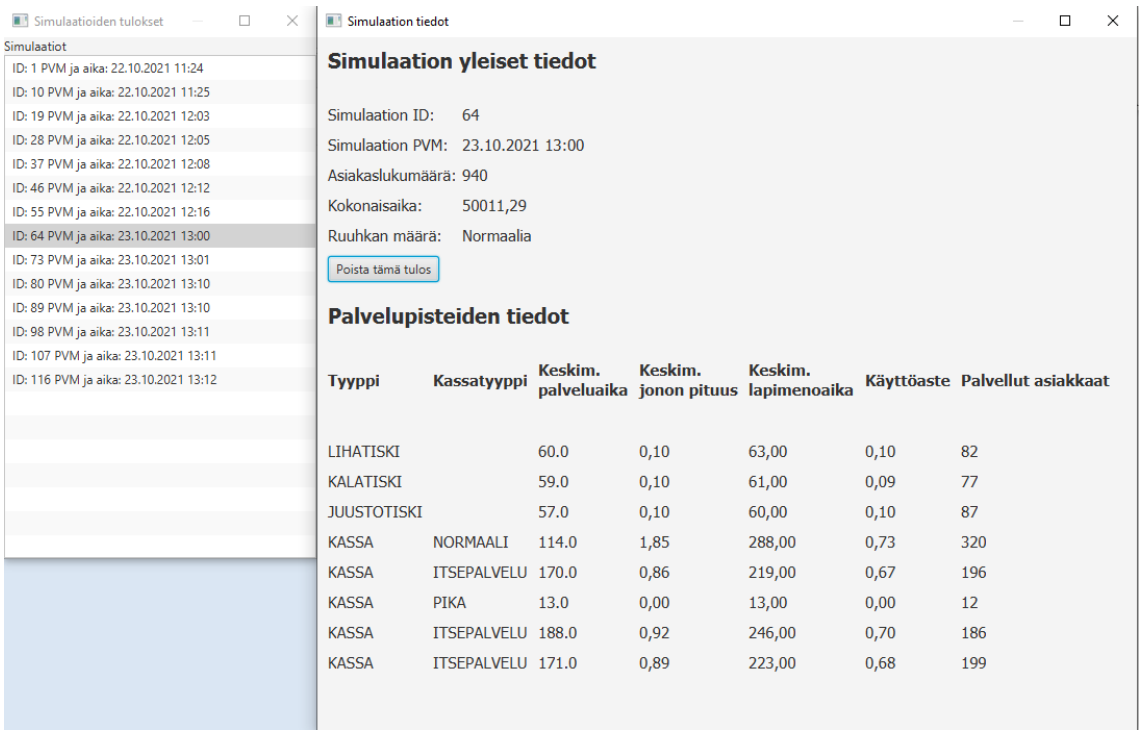
deksan kuvaketta. Lisäksi jonon alla on numero, joka ilmoittaa asiakasmäärän myös lukuna. Numeron alla on kuvake, joka havainnollistaa mikä palvelupiste on kyseessä. Kuviossa 4 näkyy simulaattorin käyttöliittymä ajon ollessa käynnissä.



Kuvio 4. Simulaattorin käyttöliittymän päänäkymä simulaation ajon aikana

Simulaatiota ajetaan aluksi käyttäjän syöttämällä viiveellä. Tätä viivettä voi kuitenkin muuttaa kesken ajon nopeuta- ja hidasta-painikkeilla. Käyttöliittymä välittää tiedon muuttuneesta viiveestä kontrollerin kautta Moottori-luokalle. Aja loppuun -nappi asettaa viiveen nollaan ja ajaa simulaation lähes välittömästi loppuun.

Ikkunan yläreunasta löytyy tietopankki-pudotusvalikko, josta löytyy kohdat tietoja ja tulokset. Tiedoista löytyy ohjeita simulaation käyttöön ja ikonien selitykset. Tuloksista taas pystyy tarkastelemaan jo ajettujen simulaatioiden tuloksia. Käyttöliittymä hakee kontrollerin kautta tietokannasta listan ajetuista simulaatioista. Kuviossa 5 näkyy, kuinka valitsemalla jonkin tietyn simulaation listasta, haetaan tietokannasta tarkemmat tiedot valitun simulaation ID:en perusteella. Halutessaan käyttäjä voi poistaa tarkasteltavan tuloksen tietokannasta.



Kuvio 5. Näkymä graafisessa käyttöliittymässä simulaation tuloksista ja yhden simulaation tarkemmat tiedot

5.4 Sisäisen logiikan kuvaus

Simulaation logiikka perustuu kolmivaihesimuloinnin toteutustapaan, eli simulaatiota suoritetaan kolmivaiheisessa (A, B ja C) syklissä. A-vaihe kuvaa kellon siirtämistä vastaamaan seuraavan toteutuvan tapahtuman aikaa, eli siis ajan etenemistä. Ajan muutosten tarkastelu tapahtuu diskreetisti, eli aika siirretään seuraavaan tulevaan tapahtumaan ja tyhjät ajanhetket jätetään huomiotta.

Ajan mallintamiseen käytetään singletonina tuotettua luokkaa Kello, jonka ajan asettamisessa ja tarkistamisessa käytetään kaikkialle näkyviä staattisia metodeja. Singletonina toteutetusta luokasta voidaan luoda vain yksi instanssi. Toisin sanoen, ohjelmaan luotava kello on uniikki.

B-vaiheessa suoritetaan seuraava ajastettu tapahtuma, eli tietylle ajanhetkelle skeduloitu tapahtuma. Näitä ovat simulaatiossa uusien asiakkaiden saapumiset ja palvelupisteiltä poistumiset.

(21)

C-vaiheessa käydään läpi kaikki palvelupisteet ja tarkistetaan, onko palvelupisteellä jonoa sekä onko palvelupisteellä asiakas palveltavana. Jos pisteellä on jonoa ja palveltavana ei ole ketään, siirretään jonosta ensimmäinen palveltavaksi. C-tapahtumien suorittamisen jälkeen palataan takaisin A-vaiheeseen ja vaiheiden toistaminen jatkuu simulaation käyttäjän asettamaan lopetushetkeen asti.

Mallin käynnistyessä OmaMoottori-luokka luo kaikki palvelupisteet, joiden määrät on annettu konstruktorin argumenttina. Alustuksien yhteydessä OmaMoottori luo myös Saapumisprosessi-olion, joka puolestaan luo ensimmäisen asiakkaan ja lisää tämän tapahtumalistaan, mikä käynnistää uusien tapahtumien tuottamisen.

Kun asiakas luodaan, lisätään samalla tapahtumalistaan uusi saapumistapahtuma jakauman mukaan generoidulle ajankohdalle. Näin saadaan luotua jatkuva asiakkaiden virta kauppaan.

Tapahtumalistassa ovat kaikki tulevat B-tapahtumat ja ne ovat järjestetty tapahtuma-ajan mukaisesti. Listassa ensimmäisenä oleva tapahtuma on siis seuraavaksi toteutuva tapahtuma. Edellä mainitut tapahtumat ovat hetkiä ajassa, joille kuuluu tapahtuman tyyppi, jonka mukaan suoritetaan tiettyä toiminnallisuutta tyypistä riippuen, kun tapahtuma käsitellään.

Tapahtuman tyyppejä tässä mallissa ovat kauppaan saapuminen, kaupassa haahuilun päättyminen, lihatiskiltä poistuminen, juustotiskiltä poistuminen, kalatiskiltä poistuminen ja kaupasta poistuminen.

Asiakkaan saapuessa kauppaan tälle arvotaan maksutapa (kortti tai käteinen) ja ostoslista, jonka perusteelta hänen reittinsä sekä kaupassa viettämänsä aika määrittyy.

Ostoslistan satunnaisgeneraattorilla arvottu pituus vaikuttaa suoraan itsenäiseen ostosten keräilyyn kuluvaan aikaan ja kassalla asioinnissa kuluvaan aikaan. Ostoslistalla mahdollisesti olevat liha, kala ja juusto vaikuttavat siihen, mille palvelupisteille asiakas päätyy.

(21)

Kauppaan saapumisen jälkeen asiakas siirtyy ensimmäisen palvelupisteen jonoon, joka on kaupassa haahuilu. Tällä palvelupisteellä voi olla rajaton määrä asiakkaita palveltavana samanaikaisesti ja asiakkaat tällä pisteellä ovat järjestetty jäljellä olevan haahuiluajan mukaisesti.

Kuten kaikki muutkin palvelupisteet, haahuilu-palvelupiste asettaa palvelun pituudelle tietyn ajan jokaiselle palveltavalle asiakkaalle ja luo tälle myös tapahtumalistaan poistumistapahtuman.

Ensimmäiseltä palvelupisteeltä poistuessa asiakkaat haarautuvat neljälle eri polulle. Osa siirtyy suoraan kassoille, kun taas ne asiakkaat, joilla on ostettavana juustoa, lihaa tai kalaa, kiertävät vastaavien palvelutiskien kautta. Näillä palvelupisteillä voi olla ainoastaan yksi asiakas kerrallaan palveltavana. Liha-, kala- ja juustotiskit ovat toiminnallisuuksiltaan identtisiä. Ainoa eroavaisuus on saapumiskriteeri kullekin tiskille.

Kun asiakkaan on seuraavaksi siirryttävä kassalle, tehdään kassan valinta tämän tapahtuman yhteydessä OmaMoottori -luokassa. Taulukosta 1 ilmenee se, miten kassa valitaan sen perusteella, kuinka monta tuotetta asiakas on ostanut ja onko hän kykeneväinen maksamaan kortilla. Jos asiakas ei kykene maksamaan kortilla, täytyy hänen valita joko normaalin kassan ja pikakassan välillä. Pikakassalle voi siirtyä ainoastaan, jos on ostanut enintään viisi tuotetta. Lopullinen kassan valinta tehdään jonojen pituuksien perusteella.

	Normaali	*Pika*	*Itsepalvelu*
Kortti, ostoksia <= 5	✓	✓	✓
Kortti, ostoksia > 5	✓	X	✓
Käteinen, ostoksia <= 5	✓	✓	X
Käteinen, ostoksia > 5	✓	X	X

Taulukko 1. Kassatyypit ja niille päätymisen ehdot

5.5 Ulkoisten tietovarastojen (tiedostot, tietokannat) kuvaukset

Ohjelma tallentaa tietoa MariaDB MySQL-tietokantaan hyödyntäen DAO-mallia ja hibernate-kirjastoa. Kaikki tallennukseen liittyvä toiminnallisuus löytyy dao-pakkauksesta, joka sisältää ISimulaationTuloksetDAO-rajapinnan sekä SimulaationTulokset, PalvelupisteenTulos ja SimulaationTuloksetAccessObject luokat.

Kun OmaMoottori on saanut ajettua simulaation loppuun, se luo SimulaationTulokset-olion, johon tallennetaan simulaation ajon parametrit, PalvelupisteenTulos-oliot sekä päivämäärän ja kellonajan, jolloin simulaatio on ajettu. Tämän jälkeen OmaMoottori luo jokaisesta palvelupisteestä PalvelupisteenTulos-olion ja tallentaa ne SimulaationTulokset-oliossa olevaan settiin. Lopuksi OmaMoottori kutsuu SimulaationTuloksetAccessObject-oliota, joka tallentaa olioiden tiedot tietokantaan hyödyntäen Hibernate ORMia.

Hibernate huolehtii automatisoidusti olio-relaatiomuunnoksista ja luo SimulaationTulokset- ja PalvelupisteenTulos-olioissa olevien annotatioiden pohjalta tietokantataulut ja niiden sarakkeet sekä taulujen väliset suhteet.

Kun käyttöliittymään halutaan hakea tietoa tietokannasta, kutsuu käyttöliittymä kontrolleria, joka puolestaan kutsuu SimulaationTuloksetAccessObject-oliota. Tietokannasta voi hakea taulukon kaikista SimulaationTulokset-olioista tai taulukon kaikista tiettyyn simulaatiokertaan liittyvistä PalvelupisteenTulos-olioista. Hibernate hakee datan tietokannasta ja muuttaa sen Java-kielelle sopiviksi olioiksi.

5.6 Testaus

5.6.1 Yleisesti

Simulaatiota on ajettu yli sata kertaa, testaten peräkkäisiä ajoja, parametrien raja-arvoja käyttäen ja myös syöttämällä epäkelvollisia arvoja alustuksiin. Simulaattorin kaikki osat ovat toimineet valtaosin odotetusti ja havaitut puutteet on korjattu.

5.6.2 JUnit-testit

Missään luokassa ei testattu settereitä tai gettereitä niiden yksinkertaisuuden vuoksi.

Asiakas-luokasta testattiin konstruktorin toimintaa, asiakkaiden ID:n inkrementointia ja korttimaksun asettamista todennäköisyyksien 0 % ja 100 % perusteella. Todennäköisyyden satunnaisuudesta johtuen näiden todennäköisyyksien väliltä testaaminen osoittautui erittäin haastavaksi.

Ostoslista-luokan osalta testattiin olion luomista ja ostoslistaan koon ylä- ja alarajaa. Ylärajaa testattiin jakauman kannalta liian isolla numerolla, mutta kuitenkin sellaisella arvolla, että ostoslistan koko ei mahdollisen virheen sattuessa ole absurdin suuri.

Palvelupiste-luokasta testattiin palvelupisteen luomista, jonon pituuden tarkistamista, jonoon lisäämistä ja siitä poistamista sekä palvelun aikana palvelupisteen varaamisen toimintaa.

Kassa-luokan toiminnasta testattiin ainoastaan kassojen luominen ja kassan valinta lyhimmän jonon perusteella, sillä Kassa on Palvelupisteen aliluokka ja toiminnallisuudet ovat pitkälti samoja.

PalvelupisteenTulos-luokkaan luotiin testit sen jokaiselle konstruktorille.

SimulaationTulokset-luokasta testattiin konsturktoreiden toiminta. Parametrillisen konstruktorin sisältä löytyvä päivämäärän asettaminen testattiin myös. Lisäksi luotiin testit ruuhkaStringiksi- ja lisaaPalvelupisteenTulos-metodeilla.

6 Simulaattorin käyttöohje

Simulaattorin käynnistyessä käyttäjä syöttää simulaatioon vaadittavat tiedot: simulaatioajan sekunteina, viiveen millisekunteina, kassojen määrän, korttimaksun todennäköisyyden ja ruuhkan asteen. Tietojen syötön jälkeen käyttäjä voi aloittaa simulaation painamalla käynnistyspainiketta.

Simulaation ajon aikana voi nopeuta- ja hidasta-painikkeilla muuttaa simulaation etenemisnopeutta. Aja loppuun -painike asettaa viiveen nollaksi ja simulaatio ajaa itsensä loppuun mahdollisimman nopeasti.

Simulaation päättyessä ajetun simulaation tiedot tallentuvat tietokantaan ja ovat tarkasteltavissa vasemman yläkulman tietopankkivalikon kohdasta tulokset. Tietokantaan tallentuu simulaation yleiset tiedot, eli simulaation ID, päivämäärä ja aika, jolloin simulaatio on suoritettu, kokonaisasiakasmäärä, kokonaisaika ja ruuhkan aste.

Palvelupistekohtaisia tallennettavia tietoja ovat palvelupisteen tyyppi, keskimääräinen palveluaika, keskimääräinen jonon pituus, käyttöaste, palvellut asiakkaat, ja jos palvelupiste on kassa, niin tallennetaan myös kassan tyyppi. Simulaatiolle annettuja syötteitä voi muokata ajojen välissä ja simulaatioita voi suorittaa useampia kertoja peräkkäin.

7 Tehdyt simulointikokeet

7.1 Ensimmäinen koe, korttimaksu ei toimi

Kattavan maksulaitehäiriön jäljittelemiseksi simuloitiin kaksi ajoa, joissa lähtöarvot erosivat toisistaan vain korttimaksun todennäköisyyden osalta. Tilanteessa oletetaan kaupan mitoittaneen henkilökunnan määrän minimiin ja luottavan itsepalvelukassoihin. Simulaattorin asettamien rajoituksien takia oletetaan, että jokaisella asiakkaalla on aina riittävä määrä käteistä mukana ja he eivät poistu kaupasta, vaikka jonotusaika nousisi huomattavasti.

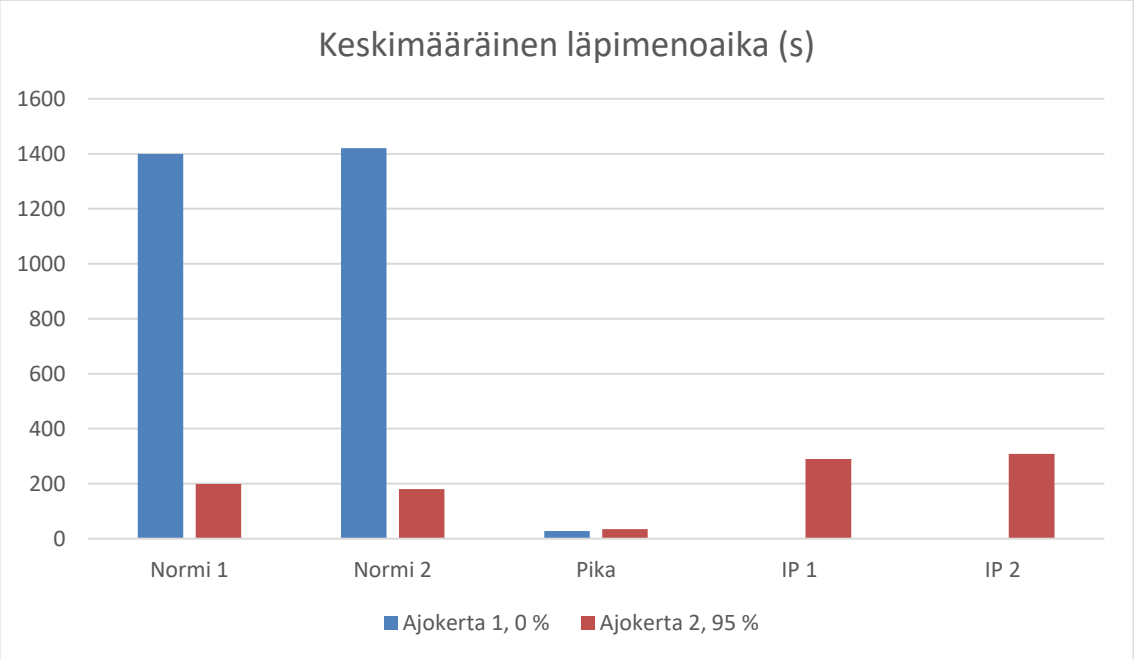
Kokeiden yhteneväiset lähtöarvot on ilmoitettu taulukossa 1. Ensimmäisessä ajossa korttimaksun todennäkyisyys oli 0% ja toisessa ajossa todennäköisyys oli 95%.

(21)

Simulointiaika	12000s
Normikassojen määrä	2
Pikakassojen määrä	1
Itsepalvelukassojen määrä	2
Ruuhkan määrä	normaali

Taulukko 2. Ensimmäisen simulaatiokokeen vakioarvot

Kun itsepalvelukassat eivät olleet käytössä, normikassat ruuhkaantuivat selvästi. Normikasoilla läpimenoajat nousivat vajaasta 200 sekunnista noin 1400 sekuntiin. Simulaation pidentyessä ongelma olisi vain pahentunut. Pikakassan läpimenoaika pysyi lähes muuttumattomana. Kuvio 6 havainnollistaa kassojen keskimääräisten läpimenoaikojen muutosta korttimaksun todennäköisyyteen sidottuna.



Kuvio 6. Ensimmäisen simulaatiokokeen läpimenoajat kassoittain

(21)

Tuloksista voidaan pääteellä, että kauppa, joka luottaa paljon itsepalvelukassoihin, voi merkittävän tietoliikennehäiriön takia kokea merkittäviä taloudellisia menetyksiä, mikäli kassahenkilökuntaa ei ole saatavilla. On kuitenkin hyvä pitää mielessä simulaation rajoitteet ja kokeen kärjistyneisyys.

7.2 Toinen koe, pikakassojen hyödyllisyys

Pikakassojen hyödyllisyyden arvioimiseksi suoritettiin kaksi ajoa (A1 ja A2). Ajojen parametrit on listattu taulukkoon 2. Itsepalvelukassojen määrä, aika ja ruuhkan pysyivät molemmissa ajoissa samoina. Ensimmäisessä ajossa normi- ja itsepalvelukassojen määrä oli isoin mahdollinen ja toisessa ajossa määrä asetettiin alhaisimmaksi mahdolliseksi.

	A1	A2
Simulointiaika	50000s	50000s
Normikassojen määrä	5	1
Pikakassojen määrä	1	1
Itsepalvelukassojen määrä	4	1
Ruuhkan määrä	ruuhkaa	ruuhkaa

Taulukko 3. Toisen simulaatiokokeen lähtöarvot

Ajossa yksi (A1) pikakassan käyttöaste oli 0,03 ja ajossa kaksi (A2) käyttöaste oli 0,04. Ajossa kaksi muiden kassojen käyttöasteet olivat 0,99 ja ajossa yksi vastaava arvo oli välillä 0,78–0,94.

Simulointien perusteella pikakassat eivät ole kannattavia, mikäli ne ovat jatkuvasti auki. Simulaatiossa asiakkaiden ostosmäärät rajoittavat pikakassojen käyttöä, mikä selittää hyvin matalan käyttöasteen. Simulaattorin rajoitteista johtuen, ostosmääriin ei voi vaikuttaa. Voidaan todeta, että kaupassa, jossa asiakkaat todennäköisemmin ostavat paljon tuotteita, kannattaa panostaa muun tyyppisiin kassoihin.

8 Yhteenveto

8.1 Yleisesti

Vaikka simulaatio onkin yksinkertainen, on sen toteuttanut ryhmä tyytyväinen suoritukseensa ja kokee sen vastaavan tehtävänannon vaatimuksia. Kyseessä oli selkeästi isoin projekti, johon yksikään projektin jäsen on tähän mennessä osallistunut.

Simulaatiolla pystyy karkeasti havainnoimaan, miten yksinkertaistetuissa ongelmatilanteessa jonojen pituudet ja asiakkaiden jonotusajat karkaavat käsistä. Toki simulaattori, on vielä kaukana siitä, että sitä voisi hyödyntää aidosti todellisessa maailmassa.

8.2 Jatkokehitysideat

Jotta simulaatio vastaisi enemmän todellisuutta tulisi kassojen määrän valitsemisen tulisi olla vapaampaa. Projektin loppuvaiheilla ryhmä totesi, että tietyn kassatyypin määrän asettaminen nollaan tulisi olla vaihtoehto. Projektin palauttamisen määräaika oli kuitenkin lähellä ja ajatus päätettiin jättää potentiaaliseksi jatkokehitysmahdollisuudeksi. Nykyiset rajoitukset ovat jäänteitä projektin alkuvaiheista.

Lisäksi kassojen määrää pitäisi pystyä muuttamaan myös simulaation ajon aika. Ei ole realistista, että kaupan henkilökunta ei reagoi mitenkään jatkuvasti kasvaviin jonoihin. Simulaatio ei pystynyt osoittamaan pikakassoja hyödyllisiksi niiden ollessa jatkuvasti auki, mutta ehkä ne voisivat toimia strategisesti jonojen purkamiseen hetkellisillä aukioloilla.

Nykyisessä versiossa simulaation ajo lopetetaan, kun käyttäjän antama aikaraja on saavutettu. Todellisuudessa kauppa kuitenkin palvelee kaikki sisään päästetyt asiakkaat. Olisi hyödyllistä nähdä, miten pitkään jonojen purkaminen kestää kaupan sulkemisen jälkeen.

Satunnaisgeneraattoreiden jakaumien arvot on valittu ilman syvällistä perehtymistä, joten niiden optimoinnista löytyy oletettavasti paljon parannettavaa. Simulaation haluttaisiin tulevaisuudessa mallintavan myös ruuhka-aikoja. Ryhmä myös mietti ominaisuutta, joka mahdollistaisi, että käyttäjä voi nappia painamalla tuoda kauppaan haluamansa määrä asiakkaita kerralla kesken ajon.

9 Liitteet

Liite 1: https://users.metropolia.fi/~samihuo/Ohjelmointiprojektin%20javadoc/doc/

Liite2:

https://innoscm-new.metropolia.fi/scm/repo/samihuo/Kauppasimulaattori/code/sources/