# Projektityö dokumentti

Okko Katajamäki
100875986
Tietotekniikka
2. vuosikurssi
2.2.2024

## Yleiskuvaus

Ohjelmassa simuloidaan tilannetta, jossa joukko ihmisiä pyrkii liikkumaan huoneesta ulos mahdollisimman nopeasti. Käyttäjä voi räätälöidä simuloinnin määrittelemällä huoneen koon, yksilöiden lukumäärän ja oven koon. Kun nämä parametrit on asetettu, simulaatio voidaan käynnistää, jolloin yksilöt sijoitetaan huoneeseen satunnaisiin paikkoihin ja ne lähtevät liikkumaan kohti ovea. Huoneessa on ruuhkaa, joten he joutuvat välttämään osumista toisiinsa ja seiniin. Kun yksilö saavuttaa oven se katoaa simulaatiosta. Ohjelman käyttäjä voi myös lisätä huoneeseen hahmoja, sekä esteitä reaaliaikaisesti. Ohjelmassa on graafinen käyttöliittymä, eli simulaatiota voi seurata reaaliaikaisesti.

## Käyttöohje

Ennen simulaation käynnistystä simulaation parametrit on mahdollista määritellä data.txt tiedostoon juurikansiossa. Datan formaatti on pilkuilla eroitetut numeroarvot. Data tulee antaa seuraavassa järjestyksessä:

Huoneen leveys, Huoneen korkeus, oven koko, hahmojen lukumäärä, simulaation nopeus

Arvoille on seuraavia rajoituksia:

-Huoneen leveys ja korkeus: min 100

-oven koko: min 60, max huoneen korkeus

-hahmojen lukumäärä: ohjelman vakauden kannalta max 20. Toimii myös selvästi suuremmilla määrillä, mutta tällöin ohjelma kaatuu tai jäätyy ajoittain. Jos annettu lukumäärä ei mahdu huoneeseen, hahmoja asetellaan niin monta kuin mahdollista.

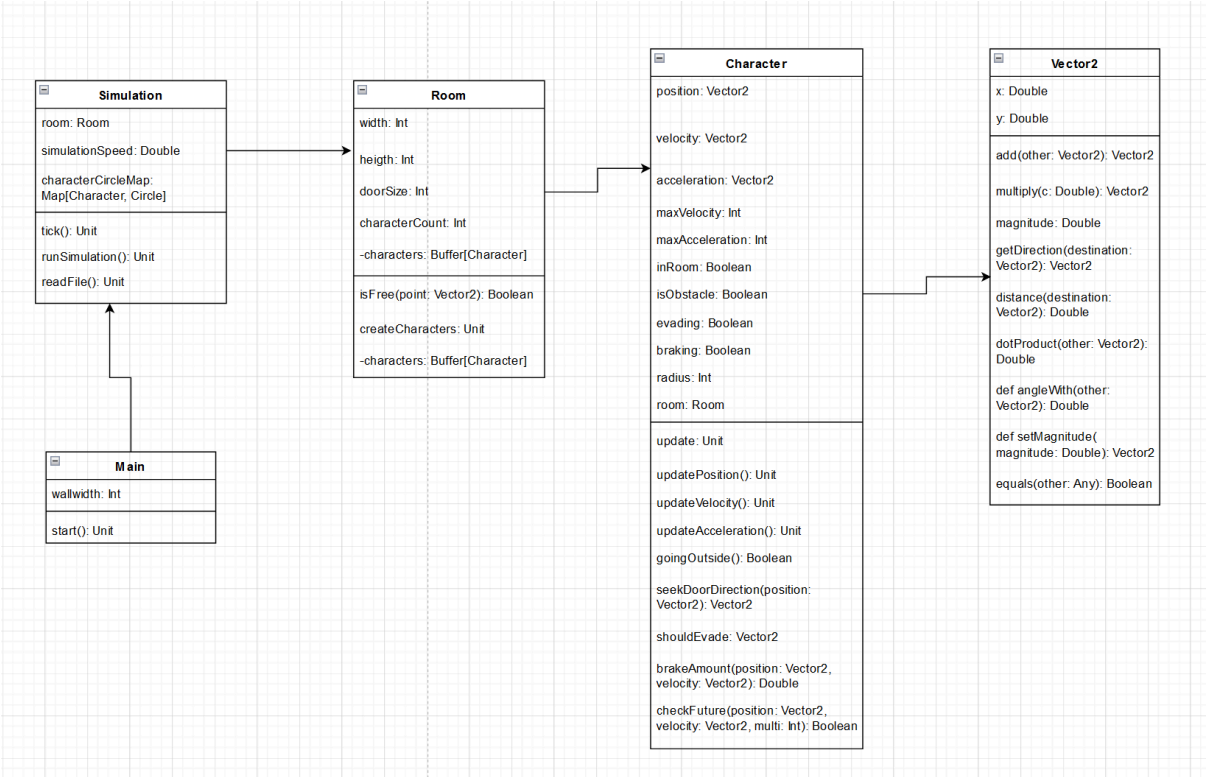
-simulaation nopeus: ohjelman vakauden kannalta max 30, toimii myös suuremmilla arvoilla.

Kun arvot on annettu simulaation voi laittaa pyörimään Main.scala tiedostosta. Kun ohjelma ajetaan hahmot asetellaan paikoilleen satunnaisesti ja simulaatio alkaa pyörimään. Simulaation pyörimisen aikana huoneeseen voi lisätä esteitä hiiren oikealla klikkauksella. Uusia hahmoja voi lisätä hiiren vasemmalla klikkauksella.

Ohjelman pyörimisen aikana hahmojen värit osoittaa niiden liikkeen tilan seuraavasti:

Vihreä: hahmo liikkuu suorinta tietä kohti ovea
Punainen: hahmo jarruttaa
Keltainen: hahmo väistää tai kiertää toisia hahmoja tai esteitä.

## Ohjelman rakenne




Ohjelma käynnistetään main luokasta. Main luokka toimii graafisena käyttöliittymänä. Main luokka kuuntelee käyttäjän komentoja. Jos käyttäjä painaa ohjelmaa hiirellä huoneeseen luodaan uusi hahmo.

Simulation luokan tehtävä on päivittää huoneen tilaa ja käyttöliittymää tasaisesti simulaation nopeuden mukaisesti. Lisäksi luokka lukee simulaation tiedot tekstitiedostosta.

Room luokka sisältää huoneen mitat, sekä huoneen hahmot. Luokan createCharacters metodi sijoittaa hahmot paikoilleen satunnaisesti.

Character luokka kuvaa yhtä simulaation yksilöä. Luokka sisältää esimerkiksi tiedon yksilön sijainnista, nopeudesta ja kiihtyvyydestä. Luokalla on myös metodit näiden muuttujien päivittämiseen. Hahmoa ohjataan päivittämällä kiihtyvyyttä. updateAcceleration metodilla on muutama apumetodia joiden avulla voidaan päättää miten kannattaa liikkua missäkin tilanteessa.

Vector2 luokkaa käytetään sijaintien, nopeuksien ja kiihtyvyyksien säilömiseen. Luokassa on useita metodeita, jotka auttavat vektoriin liittyvissä operaatioissa.

## Algoritmit

**createCharacters**

Metodi asettelee hahmot huoneeseen satunnaisesti. Algoritmi asettelee hahmoja huoneeseen niin kauan, kunnes hahmoja on käyttäjän toivoma määrä tai huoneessa ei ole enää tilaa uusille hahmoille. Asettelu tehdään niin, että huoneesta arvotaan satunnainen paikka, ja jos paikalla on tilaa se valitaan ja jos ei kokeillaan toista satunnaista paikkaa uudestaan. Jos vapaata paikkaa ei löydy 10000 iteraation aikana asettelu lopetetaan, koska huoneessa ei ole tilaa.

**goingOutside**

Metodin tarkoitus on estää hahmon joutuminen huoneen ulkopuolelle. Algoritmi lisää hahmon nykyiseen sijaintiin 10 nopeus vektoria ja jos huoneen rajat ylittyvät tuloksena saadussa vektorissa, hahmo kiihdyttää poispäin seinästä.

**shouldEvade**

Metodin tarkoitus on laskea pitäisikö hahmon väistellä toisia hahmoja tai esteitä ja jos pitäisi mihin suuntaan. Metodi lisää hahmon sijaintiin mahdollisia suuntia liikkua ja käyttää brakeamount metodia tarkistamaan onko jossain suunnassa enemmän tilaa liikkumaan kohti ovea kuin nykyisessä sijainnissa. Jos vapaata tilaa löytyy hahmo kiihdyttää kyseiseen suuntaan.

**brakeamount ja checkFuture**

brakeamount metodin tarkoitus on laskea pitäisikö jarruttaa ja jos pitäisi miten paljon. Metodi lisää hahmon nykyiseen sijaintiin eri määriä sen nykyistä nopeusvektoria ja katsoo käyttäen checkFuture metodia tuleeko hahmo törmäämään näissä mahdollisissa tulevissa hetkissä. Checkfuture käy läpi huoneen jokaisen hahmon tulevan sijainnin ja katsoo olisivatko hahmot päällekkäin tulevaisuudessa. Metodi myös katsoo kumman velvollisuus on väistää sen perusteella, että kumpi on lähempänä ovea. Brakeamount metodi antaa

jarrutuskertoimen, joka on sitä suurempi mitä lähempänä hetkenä törmäys on tapahtumassa.

## Tietorakenteet

Huoneen hahmot on säilötty muuttuvatilaiseen Buffer kokoelmaan. Valinta on mielestäni hyvä, koska se tekee hahmojen ajonaikaisen lisäämisen helpoksi.. Kokoelman yli on myös helppo iteroida, joka mahdollistaa hahmojen sijaintien noutamisen, sekä niiden tilan päivittämisen.

Ohjelmassa on myös Hakurakenne, joka mahdollistaa hahmoja vastaavan ympyrän noutamisen. Ympyrän olisi myös luultavasti voinut laittaa hahmon muuttujaksi tai yhdyksi luontiparametriksi.

Itse tehdyn Vektoriluokan käyttö sijainnin, nopeuden ja kiihtyvyyden kuvaamiseksi oli onnistunut valinta. Se, että kaikki kolme yksilön liikettä kuvaavaa asiaa oli samassa yksikössä helpotti koodausta huomattavasti. Oli myös kätevä, että nopeuden ja kiihtyvyyden suuruus ja suunta oli sidottu yhteen yksikköön. Itse tehty vektori luokka mahdollisti vektori operaatioihin liittyvien metodien luomisen, mikä auttoi siistimään koodia ja nopeutti ohjelman tekemistä.

## Tiedostot

Simulaation parametrit annetaan tekstitiedostossa. Datan formaatti on pilkuilla eroitetut numeroarvot. Käyttäjä saa ilmoituksen konsoliin jos data on virheellisessä muodossa. Tarkemmat ohjeet datan syöttämisestä löytyvät käyttöohjeessa.

## Testaus

Tein suurimman osan testauksesta käyttöliittymän kautta. Testaus toimi käytännössä niin, että laitoin simulaation pyörimään ja katsoin tapahtuuko jotain outoa tai epätoivottavaa ja jos tapahtui pyrin miettimään mikä tämän aiheutti. Testauksessa auttoi paljon, se että hahmon väri muuttuu ohjelmassa sen tilan mukaan, joka mahdollista ongelmatilanteen aiheuttavan koodin löytämisen nopeammin. Pyrin myös kokeilemaan simulaation ajamista monenlaisilla parametreilla, joka auttoi ongelmien löytämisen monipuolisemmin.

Käytin ohjelman teon alkuvaiheessa myös yksikkötestausta. Pyrin testaamaan, että ohjelman keskeisimmät metodin toimivat toivotulla tavalla. Yksikkötestit menivät läpi

aikaisemmin, mutta myöhäisemmässä vaiheessa tehdyt muutokset ohjelman eri kohtiin aiheuttivat, että yksikkötestejä ei voi enää ajaa ilman virheitä.

## Tunnetut puutteet ja viat

Ohjelma voi kaatua tai jäätyä tietyissä tilanteissa. Ongelma vaikuttaa johtuvan jollain tavalla suorituskykyyn liittyvistä asiasta, koska ongelma näkyy lähinnä vaativammissa simulaatiossa ja ongelma on tuntunut pahemmalta epätehokkaalla läppärillä, kuin pöytäkoneella. Ongelman voisi selvittää näkemällä vaivaa debukkauksessa. Ohjelman monet vaiheet voisi myös optimoida huomattavasti paremmin. Esimerkiksi hahmot seuraavat jatkuvasti jokaisen huoneen hahmon sijainteja, kun tämän voisi tehdä myös niin, että seuraaminen tapahtuu vain kun hahmot ovat samalla alueella tai lähellä toisiaan. Ongelma voisi toimia myös monisäikeisesti.

Ohjelman toinen ongelma on se, että hahmot voivat peruuttaa toistensa päälle välttäessä toista törmäystä, jos hahmot ovat ahtaassa tilassa lähellä seinää. Ongelma saattaisi olla ratkaistavissa säätämällä eri liikettä ohjaavia vakiota. Vaikuttaa kuitenkin siltä, että liikkumisen perusperiaatetta tulisi jollain tavalla parannella.

## 3 parasta ja 3 heikointa kohtaa

Yleisesti ohjelmassa on hyvää, että siinä voi simuloida monenlaisia tilanteita ja asiat toimivat useimmassa tapauksessa suunnilleen niin kuin pitäisi. Esimerkiksi Huoneen mittoja voi muuttaa ilman rajoituksia ja kartalla voi lisätä esteitä ja simuloitavien yksilöiden lukumäärän voi päättää.

Ohjelman heikoimmat kohdat mainittiin jo tunnetuissa puutteissa ja vioissa. Näiden lisäksi ohjelmassa on ongelmana, se että simulaatio on säädetty toimimaan odotetulla vaan tietyllä maksiminopeudella ja maksimikiihtyvyydellä. Ohjelman eri numeroarvot olisi voinut suhteuttaa maksimiarvoihin, jotta vakiot skaalautuisivat sopivasti ja liike pysyisi järkevänä.

Hahmojen liike voisi myös toimia paremmin. Hahmo voi esimerkiksi jäädä jumiin jos karttaan laittaa paljon esteitä. Hahmot voisivat olla “kohteliaampia” ja välttää paremmin väistöliikkeitä muiden päälle. Lisäksi hahmot valitsee välillä epäoptimaalisen reitin ovelle. Koodissa olisi myös huomattavasti varaa optimaatiolle.

## Poikkeamat suunnitelmasta, toteutunut työjärjestys ja aikataulu

Toteutin työt suunnilleen projektisuunnitelman mukaisessa järjestyksessä. Projektin aikataulu ei taas pätenyt ollenkaan. Sain ensimmäisellä viikolla huomattavan paljon aikaan, kun taas työnteon tahti hidastui projektin loppua kohden. Tein arvioit eri vaiheiden ajan käytöstä joka viikolle jonkin verran yläkanttiin. Toteutusjärjestys oli taas melko onnistuneesti suunnitelma ja seurasin sitä melko tarkasti. Prosessi ainakin opetti, että kun miettii etukäteen tekemistä, työnteko selkeytyy ja esimerkiksi optimaalisella toteutus järjestyksellä voi säästää hyvin aikaa.

## Kokonaisarvio lopputuloksesta

Olen kokonaisuudessaan tyytyväinen lopputulokseen.  Mielestäni ohjelman luokkarakenne on järkevä, sekä koodi on kohtalaisen laadukasta. Esimerkiksi koodi välttelee turhaa toisteisuttaa ja se on yleisesti järkevästi jäsenneltyä. Ohjelma voisi taas olla paremmin optimoitu ja hahmojen liikkeessä olisi vielä hiomisen varaa. Ohjelmaa voisi parantaa selvittämällä kyseiset ongelmat tai lisäämällä uusia ominaisuuksia. Ohjelmassa voi esimerkiksi olla mahdollisuus muuttaa simulaation nopeutta kesken simulaation, mahdollisuus muokata maksimi nopeuksia ja kiihtyvyyksiä, sekä ohjelma voisi tukea monenlaisia huoneita. Ohjelmassa on jonkin verran ongelmia laajennettavuuden kannalta. Ohjelman koodiin on laitettu useita vakiota ja kertoimia numeroarvoina, kun arvot olisi voinut suhteuttaa esimerkiksi maksiminopeuteen. Toisaalta luokkarakenne on melko järkevä ja sen pitäisi mahdollistaa monenlaisia lisäyksiä. Esimerkiksi eri tyylisiä yksilöitä olisi huomattavan helppo lisätä. Kokonaisuudessaan ratkaisumenetelmä, luokkarakenne ja tietorakenteet ovat mielestäni onnistuneesti valittuja. Jos aloittaisin projektin uudestaan miettisin alussa enemmän miten ratkaisu voisi tukea paremmin laajennettavuutta ja miten sen voisi optimoida nykyistä paremmin. Olisin myös voinut miettiä yksilöille älykkäämpää tapaa liikkua, huomioon ottaen sen, että lisäsin huoneeseen myöhemmin mahdollisuuden laittaa esteitä.

## Viitteet


Steering Behaviors For Autonomous Characters (Craig W. Reynolds)
http://www.red3d.com/cwr/steer/gdc99/

https://www.scalafx.org/

**https://medium.com/@niiicolai/an-introduction-to-movement-in-2d-games-281ff3b5853**

**https://stackoverflow.com/questions/4373741/how-can-i-randomly-place-several-non-colliding-rects**