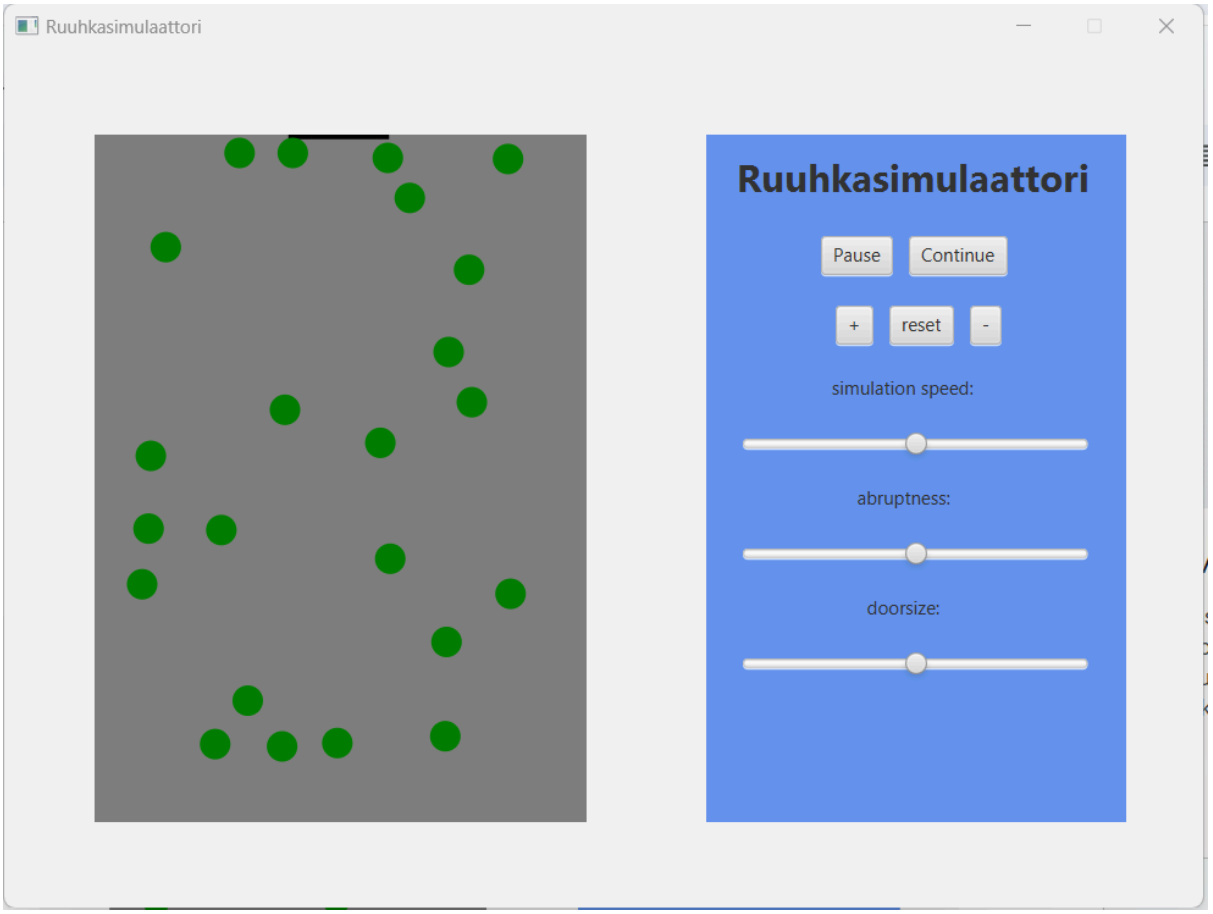
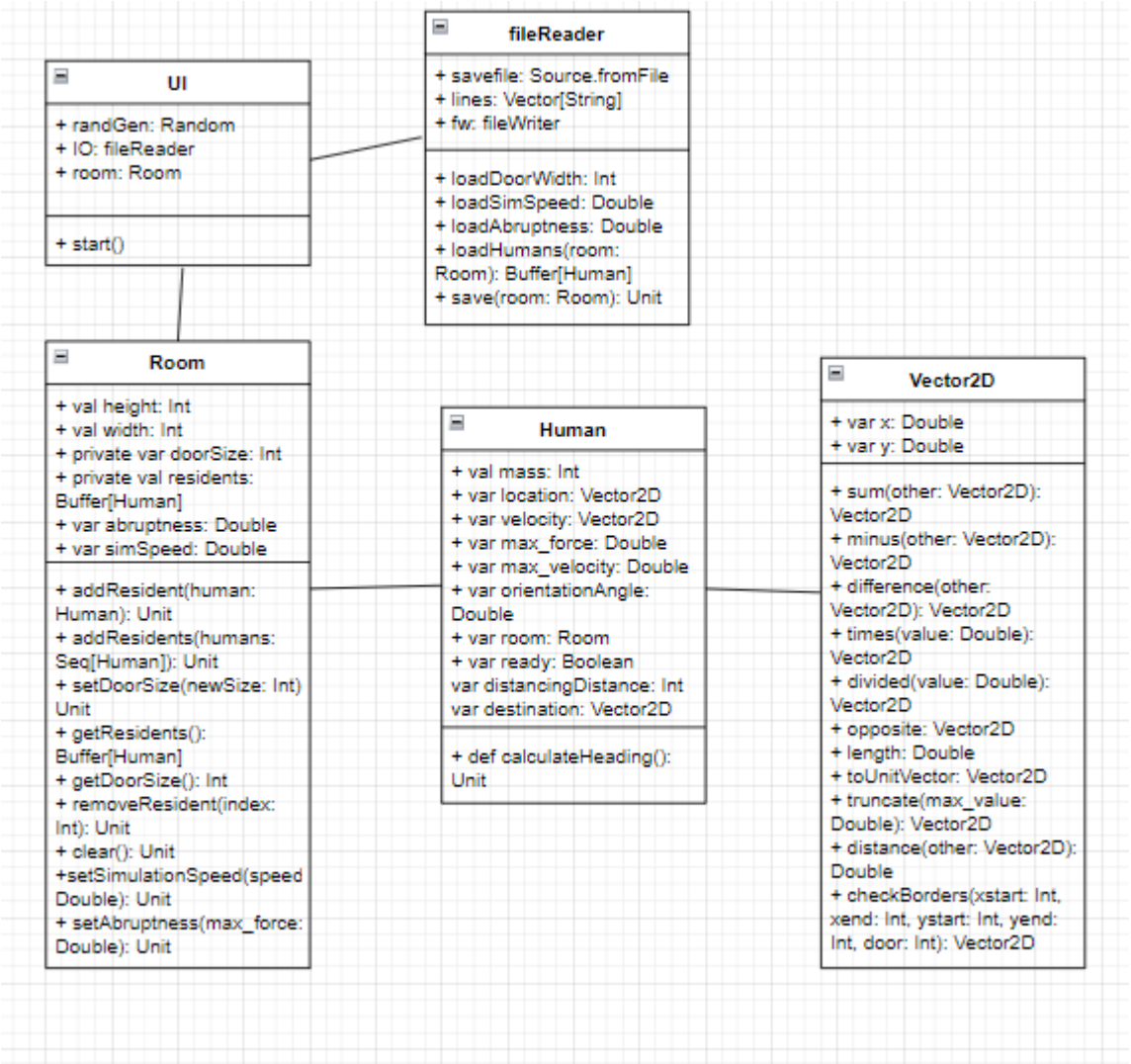
RUUHKASIMULAATTORIPROJEKTI

Juuso Toivanen, 101711117, Tietotekniikka, 1.vuosikurssi, 18.04.2024

Toteutin simple vehicle model -tekniikalla ja ScalaFX:llä ruuhkasimulaattorin jossa ihmisfiguurit pyrkivät ulos oviaukosta ja joiden toimintaan erilaiset käyttäytymiskuviot ja fyysiset rajoitteet vaikuttavat. Alla kuva käyttöliittymästä:



Kaikki käyttäjätoiminnallisuus on oikeanpuoleisessa laatikossa. Continue-nappi käynnistää simulaation ja Pause puolestaan keskeyttää. + - ja - -napeista saa joko lisättyä tai poistettua satunnaisen ihmisen. Lisätty ihminen ei voi ilmestyä kenenkään olemassa olevan päälle, eikä nappi tee mitään jos ihmisiä on jo vähintään 100. Kun simulaatio on käynnissä, aina kun ihminen pääsee ovelle, hän katoaa. Liukusäätimillä voi säätää erilaisia parametreja, järjestyksessä simulaation nopeutta, eli ihmisten maksiminopeutta, äkillisyyttä, eli maksimivoimaa joka ihmisiin voi vaikuttaa, sekä oven kokoa. Ovea kuvaava musta palkki muuttaa myös pituuttaan oven kokoa muutettaessa. Reset puolestaan palauttaa oletusparametrit, poistaa kaikki nykyiset ihmiset ja lisää satunnaisesti 3 - 50 uutta. *Laadullisesti parhaan simulaation saa yksinkertaisesti painelemalla resetiä, kunnes ihmisiä ilmestyy haluttu määrä (yleensä noin >25)*. Parametreilla leikkimällä saa myös varsin hauskoja ilmiöitä aikaan. Simulaation tila (ihmisten ja parametrien) tallennetaan tallennustiedostoon, kuitenkin niin, että jos ohjelmaa suljettaessa ei ole yhtään ihmistä jäljellä, niitä luodaan seuraavalla kerralla 3). Alla kuva rakenteesta UML-muodossa:



Rakenteeseen tein muutaman muutoksen alkuperäiseen suunnitelmaan verrattuna. Simulator-luokan poistin, sillä sain kaiken tarvittavan toiminnallisuuden liitettyä luontevasti Roomiin. Lisäksi Vector2D on uusi lisäys, joka helpottaa paikkaan liittyvien asioiden käsittelyä. Muuten projektissa on perinteinen UI-IO-logiikka -kolmijako, ja Human-päälogiikkaluokka on edelleen olemassa. Sillä on paljon muuttujia ja yksi ainoa metodi, joka laskee uuden sijainnin. Room keskittyy kaikkien Humanien kokoamiseen yhteen ja parametrien säilytykseen, ja pääohjelma on UI:ssa. Yksi huomioitava asia on se, miten Roomissa ei ole minkäänlaista “päivitä kaikki”-tyyppistä metodia. Tämä liittyy UI:n toteutukseen, eli siihen että visuaalisesti kaikki ihmiset ovat muuttuvia Circlejä, joihin ei pääsisi käsiksi Roomista. Niimpä UI päivittää kaikkia ihmisiä yksi kerrallaan (poikkeuksena Humanin muuttujiin liittyvät parametrit ja niiden muuttaminen).

Koko kokonaisuuden keskeisimmät algoritmit löytyvät Humanin calculateHeadingistä, ja ne perustuvat aiemmin mainittuun simple vehicle modeliin, joka taas perustuu yksinkertaisuudessaan siihen, että jokaisella ihmisellä on muuttujina skalaariset massa, maksimivoima, maksiminopeus ja suuntakulma, ja vektoriset sijainti ja nopeus. Erilaiset käyttäytymisestä johtuvat voimat muuttavat lopulta nopeusvektoria. Kiihtyvyys saadaan Newtonin 2. laista jakamalla voima massalla.

```
Simple Vehicle Model:
    mass          scalar
    position      vector
    velocity      vector
    max_force     scalar
    max_speed     scalar
    orientation   N basis vectors
```

```
steering_force = truncate (steering_direction, max_force)
acceleration = steering_force / mass
velocity = truncate (velocity + acceleration, max_speed)
position = position + velocity
```

Kuvassa olevista max_force ja max_velocity liittyvät abruptness- ja simulationspeed -parametreihin. Kääntyvyysvoimaan vaikuttavat erilaiset käyttäytymistavat. Esimerkiksi ihmiset yrittävät ohjautua ulos oviaukosta, (Seek):

```
desired_velocity = normalize (position - target) * max_speed
steering = desired_velocity - velocity
```

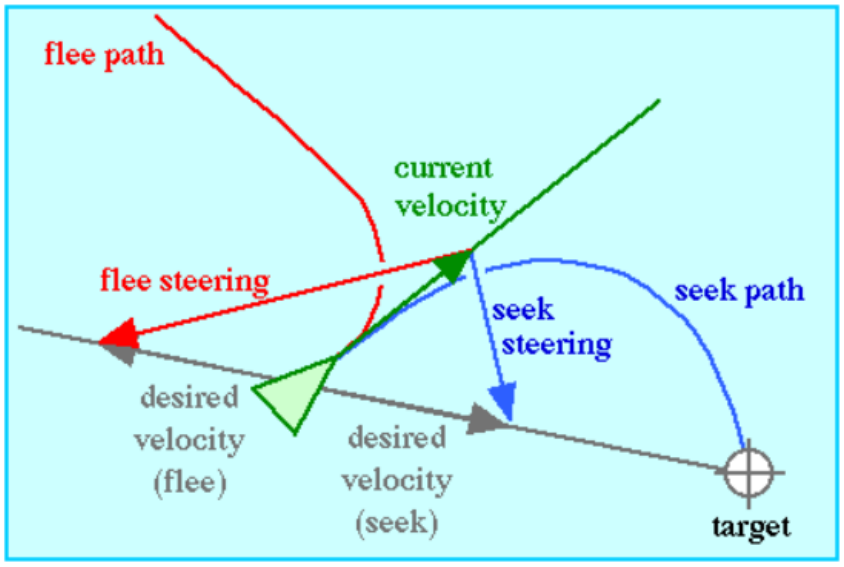


Figure 3: **seek** and **flee**

välttää seiniä, mikä on tulkittu vain fyysisiksi rajoitteiksi, mikä yleensä ryysistilanteessa päteekin ainakin oman kokemuksen mukaan, sekä pitää turvavälin muihin(Separation).

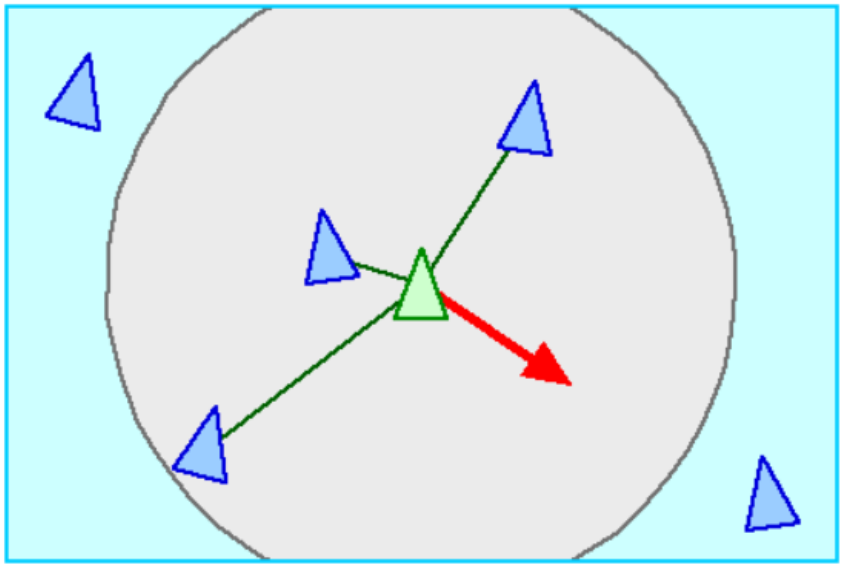


Figure 14: **separation**

Nämä kaikki yhdessä sopivin painokertoimin vaikuttavat siihen, miten ihmiset simulaatiossa liikkuvat.

Tietorakenteina on kaikista perinteisimmät scalan vakiovaihtoehdot Buffer ja harvakseltaan Vector. Myös Vector2D:n voisi lukea tietorakenteeksi joka yhdistää x- ja y -arvot, mutta se ei ehkä ole mielekästä.
Ulkoisia tiedostoja on käytetty simulaation tilanteen tallentamiseen formaatissa, jossa kolme ensimmäistä riviä sisältävät tiedot parametreista ja seuraavat mahdollisista jäljelle jääneistä ihmisistä, eikä tallennustiedostoa ole oikeastaan tarkoitus muokata missää vaiheessa, sillä simulaatiota voi muokata ajon aikana.
Testaukseen kehityksen aikana käytin vain UI:ta, sillä se tarjosi ylivoimaisesti parhaan palautteen projektin kontekstissa. Yritin lisäksi muuten vain lisätä yksikkötestin, joka olisi testannut päätyvätkö kaikki ihmiset ulos ovesta pitkähkön ajan jälkeen uuden simulaation alusta, mutta en saanut kirjastoriippuvuuksia toimimaan.

Ainoa isompi puute on logiikassa, sillä joskus (hyvin harvoin, mutta kuitenkin) ihmiset saattavat hetkellisesti mennä osittain päällekkäin. Hyväksyin tämän kuitenkin, sillä simulaatio toimii pääosin järkevästi nykyisillä parametreilla, ja päällekkäisyyden voi tulkita väistämättöminä törmäyksinä toisiin ryysiksen aikana. käyttäytyminen ei muutenkaan ole sataprosenttisen realistista, ja siinä onkin yksi jatkokehityksen mahdollinen aihe. Uuden sijainnin laskemisen olisi voinut pilkkoa useampaan metodiin, vaikka selkeyttä saikin hyvin myös riviväleillä ja muilla vastaavilla.
Hyviä puolia ovat kokonaisuuden sulavuus, ja kaikesta ylläolevasta huolimatta päätoimisesti onnistunut logiikka. Koodirivejä on myös määrällisesti suhteellisen vähän, ja rakenne on mielestäni selkeä.

Ajankäyttöön liittyen huomasin sen, että omaan taipumuksia pitkiin yksittäisiin rupeamiin. Tästä on sekä haittaa että hyötyä monella tavalla. Silloin kun aloitan tekemään saan myös paljon aikaan, mutta epätasainen aikataulutus ei ole optimaalista ennustettavuuden kannalta. ScalaFX:ään liittyvien asioiden selvittelyyn meni luultavasti kokonaisuudesta eniten aikaa (logiikkaa joutui myös jonkin verran miettimään, mutta se oli odotettavissakin). Tähän liittyen opin tiettyjä suhtautumistapoja asioihin joista lienee tulevaisuudessa hyötyä jumien estämiseksi. Esimerkiksi pala kerrallaan -lähestyminen tuo aina edes jotakin tuloksia, ja UI:n runkoon otin mallia olemassaolevasta ScalaFX-tutoriaalista alkuun pääsemiseksi.

Tiivistyksenä noudatin pitkälti alkuperäistä toteutussuunnitelmaa, ja sain mielestäni pääpiirteissään varsin toimivan ja selkeän tuotteen aikaiseksi. Aikataulutukseen voi tulevissa projekteissa miettiä mahdollisia parannuksia.