Ohjelmointistudio 2: Projekti

Lennonjohtopeli, Tekninen suunnitelma

Oskari Mäkinen

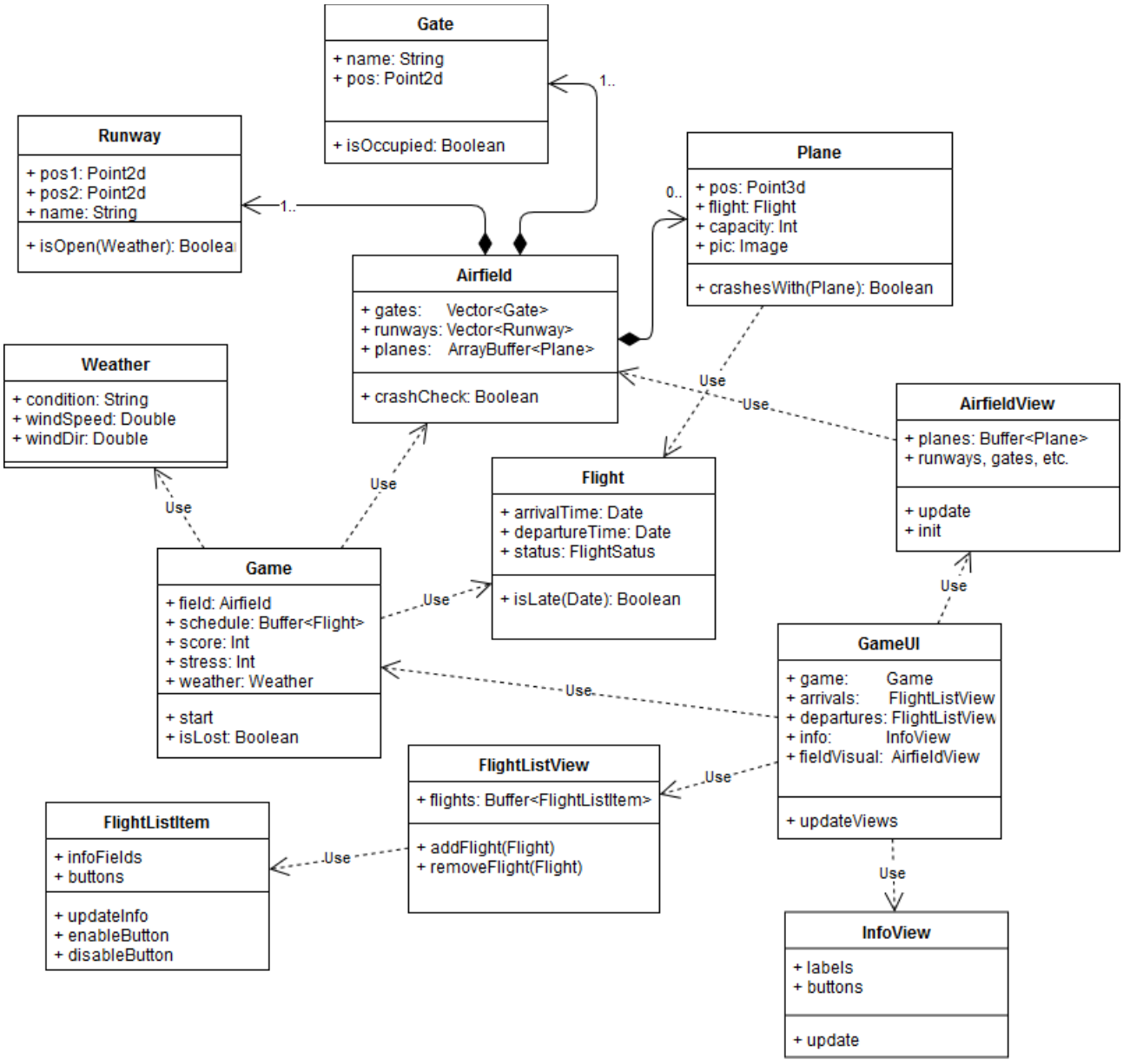
477510

EST/ELEC 4. vuosikurssi

14.2.2019

Ohjelman rakennesuunnitelma

Ohessa UML-kaavio alustavasta luokkarakenteesta. **Game**-luokan on tarkoitus huolehtia pelin logiikasta, **GameUI** vastuulla taas on käyttöliittymä. Peli- ja käyttöliittymälogiikka on pyritty erottamaan mahdollisimman hyvin, jotta näitä voitaisiin kehittää toisistaan riippumatta. Se, mitä tietoja kukin luokka tulee tarvitsemaan ei ole tässä vaiheessa vielä täysin selvää, joten luokkien väliset suhteet, voivat vielä muuttua.



Käyttötapauskuvaus

Ohjelman käynnistyessä käyttäjä voi valita pelaavansa jollain valmiilla kentällä tai satunnaisesti generoidulla kentällä. Hän myös valitsee vaikeustason, joka määrittää aikataulun kiireisyyden, sääolojen haastavuuden sekä vuoron pituuden. Tämän jälkeen peli alkaa, ja GameUI-luokka tekee tarvittavat toimenpiteet näkymien alustukseen. Pelitilannetta päivitetään jatkuvasti, ja eri näkymät päivittyvät sen mukaisesti. InfoView-näkymässä pelaaja näkee yleiset pelin tiedot, ja voi keskeyttää, jatkaa tai lopettaa pelin painikkeiden avulla.

Saapuvien ja lähtevien lentojen FlightListView-näkymissä on esitetty listassa lentojen tiedot FlightListItemien avulla. Jokaiseen FlightListItemiin kuuluu myös painikkeita, joita antamalla voi antaa lentokoneille käskyjä. Käskyn antaminen kutsuu Game-luokassa tarvittavia funktioita. Tilanteesta riippuen tietyt painikkeet voivat olla poissa käytöstä. Esimerkiksi jos yhdelle koneelle ollaan antamassa laskeutumislupaa, ei muille ilmassa oleville koneille voi samaan aikaan kommunikoida. FlightListItem-elementtien painikkeet ovat pelaajan ainoa tapa kontrolloida pelin tapahtumia.

Game-luokka tuo peliin uusia lentoja joko ennalta tai satunnaisesti määritellyn aikataulun perusteella. Lennot luetaan tiedostosta tai arvotaan pelin alussa, ja ne ilmestyvät peliin määritettynä aikana. Lennon ilmestyminen tarkoittaa siis lentokoneen saapumista kentän alueelle. Kun kone saapuu, se alkaa kiertämään kenttää ilmassa ja odottaa käskyjä. Game-luokka valvoo myös jatkuvasti, ettei peli ole hävitty. Peli voidaan hävitä, jos tapahtuu yhteentörmäys, tai stressitaso nousee liian korkeaksi. Stressitaso nousee ruuhkan ja myöhästymisten takia.

AirfieldView on graafinen esitys lentokentän tilasta. Näkymää päivitetään jatkuvasti lentokoneiden sijainnin perusteella. Näkymän tarkoitus on tuoda pelaajalle tietoa lentokentän tilasta, jotta hän voi välttää törmäykset ja määrittää kiitoradat ja portit järkevästi.

Peli loppuu, kun pelaajan työvuoro on lopussa tai jokin häviämisehto täyttyy. Pelaajalle näytetään kerätyt pisteet ja ranking, huippupisteet tallennetaan ja tämän jälkeen siirrytään takaisin aloitusruutuun.

Algoritmit

Pelissä ei todennäköisesti tulla tarvitsemaan monimutkaisia algoritmeja. Esimerkiksi törmäyksentarkistusalgoritmi voidaan toteuttaa vertaamalla kunkin koneen sijaintia muihin koneisiin ja laskemalla sijaintien välinen etäisyys sqrt(x^2 + y^2 + z^2). Aikavaativuus tulee olemaan O(n^2) (n=koneiden määrä), mutta tämä ei tule olemaan ongelma, sillä törmäys tarkistetaan muutaman kerran sekunnissa ja pelissä ei samanaikaisesti ole kovin montaa konetta.

Tietorakenteet

En näe tässä vaiheessa pelissä tarvetta itsetehdyille tietorakenteille. Pelin ajan vakiona pysyvät kiitoradat ja portit voidaan säilyttää muuttumattomissa Vector-kokoelmissa. Lennoille ja lentokoneille taas luontevinta on käyttää ArrayBuffereita. Lennot, jotka odottavat tuloaan peliin lienee järkevintä säilyttää prioriteettijonossa (PriorityQueue), joka on määritelty saapumisajan perusteella.

Aikataulu

Pelilogiikan luokat ja metodit: 10-20h
Yksikkötestien luominen: 10h
Scala swing -kirjaston opiskelu: 30h
Käyttöliittymän toteutus: 30h
Debuggaaminen: 10-20h
Pelikokemuksen hiominen/ lisäominaisuuksien toteutus: 30h

Yhteensä siis 120-140 tuntia, voi tosin olla huomattavasti enemmänkin.

Tarkoitus on aloittaa keskeisen pelilogiikan toteuttamisella, ja tämän jälkeen alkaa asteittain rakentamaan käyttöliittymää. Tästä eteenpäin eri osa-alueita tullaan todennäköisesti kehittämään vuorotellen.

Yksikkötestaussuunnitelma

Törmäyksentunnistusta voidaan testata luomalla joukkoja koneita. Osassa joukoista on törmäystilanne, osassa ei. Tunnistusalgoritmin tulee huomata törmäystilanteet.

Tietojen päivittymistä eri käyttöliittymän elementeissä on syytä myös testata. Yksi kohde on esimerkiksi saapuvien lentojen lista; listan koon pitäisi kasvaa ja lennon ilmestyä sinne, kun lento saapuu peliin. Laskeutuneen lennon tulisi poistua listasta.

Tämän lisäksi monia luokkiin tulevia apumetodeja varten on aiheellista kirjoittaa yksikkötestejä, jotta ohjelman määritystenvastainen toiminta voidaan huomata.

Kirjallisuusviitteet ja linkit

Yleistietoa lennonjohtamisesta:

https://en.wikipedia.org/wiki/Air_traffic_control
http://krepelka.com/fsweb/learningcenter/airtrafficcontrol/atcindex.htm

Inspiraationa toimiva peli: https://en.wikipedia.org/wiki/Air_Traffic_Chaos

ATC sanastoa: http://krepelka.com/fsweb/learningcenter/airtrafficcontrol/atcglossary.htm

Käyttöliittymäohjelmointi Scalassa ja Swingillä:

http://otfried.org/scala/gui.html
http://hello-scala.com/252-scala-swing.html
https://www.scala-lang.org/api/2.9.1/scala/swing/package.html
https://docs.oracle.com/javase/tutorial/uiswing/

Graafisten ja äänielementtien hankintaan:

https://opengameart.org/