1. Lentokenttäsimulaattori

Tämä harjoitustyö pohjautuu luennoilla käsiteltyyn lentokenttäsimulaattoriin ja siinä sovellettuihin jonotietorakenteisiin (Queue ja Extended_queue). Jälleen simulaattorista tehdään eri variaatioita inkrementaalisesti siten, että tietyn variaation tekeminen edellyttää aina sitä, että edeltävät variaatiot on myös tehty. Harjoitustyöstä saa pisteitä sen mukaan kuinka pitkälle variaatioiden tekemisessä pääsee eli arvostelu perustuu projektiin käytettyyn työmäärään. Mitään jo tehtyä ohjelmakoodia ei pidä heittää pois uutta variaatiota tehdessä; jos sille ei ole käyttöä uudessa variaatiossa voit kommentoida sen pois tai jos versionhallintaohjelman käyttö on sinulle tuttua edellisten variaatioiden lähdekoodit löytyvät vähintään vanhemmista versioista lähdekooditiedostoja.

Alta löydät vaatimukset eri variaatioille. Samoin alta löytyy lista palautukseen liittyvistä vaatimuksista

1.1. Yhden pisteen työn vaatimukset

Tee luennoilla käsitelty toimiva lentokenttäsimulaattori. Mitään muutoksia ei tarvitse tehdä sen toimintaan vaan luentokalvojen ”perusversio” riittää tässä. Toki joudut integroimaan eri jonotietorakenteisiin liittyvät lähdekoodit sekä muut tarvittavat lähdekoodit ml. header-tiedostot yhteen toimivaksi ohjelmaksi. Aja simulaattoria erilaisilla lentokoneiden saapumistiheyksillä (parkkipaikalta, muilta lentokentiltä) ja kirjaa niihin liittyvät simulointitulokset README.DOCX –tiedostoon (siitä kerrotaan myöhemmin tässä dokumentissa). Totea simulointitulosten järkevyys.

Yritä löytää testeissä saapumistiheyksien osalta sellaiset arvot, että lentokentän suorituskyky juuri ja juuri riittää kiitoradalle saapuvan liikenteen käsittelyyn (kiitoradan ”joutokäyntiaste” on alle 5 prosenttia ja simuloinnin loputtua saapuvien koneiden jonot eivät ole liian pitkiä eikä muille lentokentille ole käännytetty kohtuuttoman montaa lentokonetta). Totea testituloksista tämä kriittinen testitapaus (mitkä olivat sen syötteet ja millainen oli tulos).

Aja vielä edellinen kriittinen testitapaus lyhyemmillä lentokonejonoilla ja vastaavasti pidemmillä. Miten simulaattorin ajotulokset nyt käyttäytyvät olettaen että syöte on sama kuin edellä kriittisessä testitapauksessa.

1.2. Kahden pisteen työn vaatimukset

Edellä tehtyä lentokenttäsimulaattoria muutetaan siten, että lentokentällä on käytössä kaksi kiitorataa. Toista käytetään yksinomaan laskeutumisiin ja toista nousuihin.

Vertaa testaamalla tämän simulaattorin toimintaa yhden pisteen simulaattorin toimintaan samoilla lähtötiedoilla. Pystytäänkö vertailusi perusteella lentokentän kapasiteetti (= aikayksikössä käsiteltyjen lentokoneiden määrä) käsitellä lentokoneita kaksinkertaistamaan tällä versiolla lentokentästä ? Tai jopa yli kaksinkertaistamaan.

Koska yhden pisteen lentokenttäsimulaattorin ominaisuuksia ei ole tarkoitus nyt hävittää voisi olla järkevää aloittaa ohjelman ajo valikolla, jossa kysytään mitä versiota lentokenttäsimulaattorista ajetaan. Ja jos jokin luokka/aliohjelma ohjelmoidaan kahden pisteen työssä **eri tavoin** kuin yhden pisteen työssä voisi olla järkevää säilyttää ohjelmassa molemmat luokat/aliohjelmat variantteina;

esim. kiitorataluokan tapauksessa luokkina Runway_1 (käytetään yhden pisteen työssä) ja Runway_2 (käytetään kahden pisteen työssä; jos eroaa joltain osin Runway_1:stä). Saman Runway-luokan käyttäminen molemmissa tapauksissa voi johtaa hankaliin haarautumisrakenteisiin; esim. if-lauseilla yritetään erottaa eri variantit toisistaan. Tämä puolustaisi Runway_1:n ja Runway_2:n tekemistä ihan selkeyden takia. Toki voit hoitaa tämän homman niin hienolla tavalla kuin vain keksit! Huom: Runway yllä on vähän huono esimerkki luokan varioinnista (ks. alla olevaa tekstiä) mutta sama pätee muihinkin simulaattorin luokkiin ja itse pääohjelmaankin (voit ohjelmoida geneerisen main():in, josta kutsutaan tarpeen mukaan main_1():sta, main_2():sta jne. tehdyn variaation ja käyttäjän syötteen mukaan.

Vihje: mieti voisiko perusversion Runway-luokkaa käyttää samanlaisena myös tässä variantissa? Runway-luokan toista jonoahan ei ole pakko käyttää lainkaan jos ko. kiitoradan osalta koneita tulee kiitoradalle vain yhdestä jonosta; esim. parkkipaikalta. Muutokset ohjelmaan voisi tehdä tämän luokan palveluita kutsuvissa ohjelman osissa (ylemmän abstraktiotason koodissa). Huomaa myöskin, että tässä variantissa yhden aikayksikön sisällä tapahtuu yhtä aikaa tapahtumia kahdella kiitoradalla (jos niille on tulossa koneita jostakin).

1.3. Kolmen pisteen työn vaatimukset

Tässä lenttokenttävariaatiossa kentällä on kaksi kiitorataa. Toista käytetään pääsääntöisesti laskeutumisiin ja toista nousuihin. Jos jompikumpi jono on jonain ajankohtana tyhjä niin molempia kiitoratoja voidaan tuona aikana käyttää toisen jonon purkamiseen. Esimerkiksi jos laskeutuvia koneita ei ole jollain kellonlyömällä voi kaksi konetta lähteä nousuun yhtä aikaa tuolloin.

Lisäksi lentokentän toimintaa muutetaan sillä tavoin, että jos saapuvien koneiden jono on täynnä laskuja varten varatulle kiitoradalle ja uusi kone saapuu muualta tälle kentälle ko. kone ohjataankin nousuja varten varatulle kiitoradalle. Täksi ajaksi nousut lentokentältä pysäytetään. Muualta tulevia koneita ohjataan nousuja varten varatun kiitotien jonoon niin kauan että laskeutumisia saadaan purettua sen verran että kaikki laskeutuvat koneet mahtuvat laskeutumista varten varatun kiitotien jonoon (puretaan liika laskeutumisruuhka). Koko tämän ajan kentän molemmat kiitotiet palvelevat laskeutumisia ja nousut on silloin kielletty.

Pystytkö osoittamaan testaamalla tämän lentokenttävariaation paremmuuden kahteen edelliseen nähden kun siinä käytetään kiitoratoja vieläkin tehokkaammin ? Miten asia näkyy ajotuloksissa ?

1.4. Neljän pisteen työn vaatimukset

Peruslentokenttävariaation toimintaa (yhden pisteen työ) muutetaan seuraavasti:

- lentokentällä on kolme kiitorataa
- yksi niistä on varattu yksinomaan laskeutumisiin
- toinen on varattu yksinomaan nousuihin
- kolmatta kiitotietä käytetään laskeutumisiin. Jos ilmassa ei ole yhtään konetta odottamassa laskeutumisvuoroaan tätä kolmatta kiitotietä voidaan käyttää myös nousuihin.

Huomaa tämän variantin kohdalla että tiettynä aikana lentokentällä voi olla yhtäaikaista toimintaa kolmella kiitotiellä.

1.5. Viiden pisteen työn vaatimukset

Lentokenttäsimulaattorin perusversion toimintaan tehdään seuraavat muutokset:

- lentokentällä on yksi kiitorata
- kun lentokone saapuu muualta tälle lentokentälle koneelle arvotaan jäljellä oleva polttoaineen määrä. Määrä ilmaistaan aikayksiköinä, jonka verran kone voi olla vielä ilmassa odottamassa laskua mukaan lukien laskeutumiseen tarvittava polttoaine (polttoainetta on varattava yhden yksikön verran tähän). Tämä tieto on lisättävä koneen yksityisiin tietojäseniin. Voit olettaa saapuvalla koneella olevan polttoainetta vähintään sen verran että se kykeneen välittömään laskeutumiseen (arvottu polttoainemäärä on aina vähintään yksi).
- jos saapuvalle koneelle arvottu polttoainemäärä ei riitä ilmassa odottelemiseen koneen annetaan laskeutua välittömästi. Tämän vuoksi ilmassa odottavien koneiden odotusaika kasvaa ja tästä syystä jonkun/joidenkin koneiden polttoaine voi loppua odottaessa.
- ilmassa olevia koneita ei tarvitse skannata lävitse jos niiden odotusaika kasvaa vaan voit tarkistaa aina laskeutumisen yhteydessä polttoaineen riittävyyden. Jos se loppuu nyt tai loppui jo voit rekisteröidä yhden lentokoneen putoamisen kentälle.
- myös muualle käännytettävät saapuvat koneet voivat tippua siitä syystä, ettei niiden polttoainemäärä riitä tavoittamaan mitään lähellä olevaa kenttää. Mieti tähän jokin sopiva polttoaineen määrän kynnysarvo (vakio), jota pienemmällä polttoainemäärällä muualle lähteminen ei onnistu.

Kuinka paljon lentokenttää voidaan kuormittaa saapuvalla ja lähtevällä liikenteellä ennen kuin koneet alkavat tippua ilmasta ?

1.1. Vaatimuksia palautukselle

- palautus tehdään tabulaan kurssisivulla osoitettuun paikkaan
- palautetaan joko koko ohjelmointiprojekti Microsoft Visual Studio 2013:n ymmärtämässä muodossa tai sitten tarvittavat lähdekoodit siten, että niistä saa toimivan ohjelman tekemällä pelkästään tyhjän Win32 console project –tyyppisen projektin, johon tiedostot liitetään ja saadaan siten käännettyä toimivaksi ohjelmaksi.
- palautukseen (= pakattuun tiedostoon) liitetään mukaan myös README.DOCX –tiedosto (huom: Word-dokumentti), johon on dokumentoitu:
    - ohjelman käännös-/linkkaus-/asennus-/ajo-ohje niin, että testaaja pääsee testaamaan ohjelmaa seuraamalla näitä ohjeita
    - ohjelman jokaisen tason (1-5) yhteydessä ajetut testitapaukset, joilla ohjelman toiminnasta on varmistuttu. Käytännössä dokumentoitte annetun syötteen (cmd-ikkunasta / tiedostosta). Jos syötteenä on käytetty tiedostoa, jossa on isompi sisältö, koko tiedosto pitää liittää pakattuun tiedostoon mukaan ja se käyttö testauksessa pitää ohjeistaa (=> testaaja pystyy testaamaan helposti uudestaan testitapauksenne). Kutakin syötettä vastaava tulos on dokumentoitava raporttiin esim. näytönkuvana tai tekstimuodossa kopioituna ohjelman ajotuloksista.
    - mitä pistemäärää tavoittelet palautetulla harjoitustyöllä. Tämän tiedon pitää olla realistinen ja oikea; jos tavoittelet esim. viittä pistettä on ohjelmastasi löydyttävä kaikki viisi variaatiota.
    - palauttajan yhteystiedot: sähköpostiosoite ja puhelinnumero

- tuntikirjanpito työn tekemiseen kuluneesta ajasta tyyliin pvm, käytetty aika puolen tunnin tarkkuudella, mitä tuona aikana tehtiin. Tällaisia rivejä kannattaa kerätä esim. Excel-tiedostoon, jonka sisällön liitätte README.DOCX -tiedostoon.
- selitä lyhyesti sanallisesti kussakin variaatiossa käyttämäsi suunnitteluratkaisu. Eli mikä on kyseisellä tasolla tekemäsi oman ohjelmakoodin idea lyhyesti: miten ratkaisit ohjelmointiongelman sanoin selitettynä ? Tässä tietoa pitää tiivistää; jokaista ohjelmariviä ei pidä kertoa sanallisesti. Esim. voit kertoa jonkin algoritmin vaikka pseudokoodina tai ihan sanallisesti tai vaikkapa vuokaavion tapaisena kuvana. Muutoinkin jos asian selittämistä auttaa kuva niin liitä sellainen mukaan palautukseen (jonkinlainen UML-kaavio tms.). Keskity tässä kohdassa kertomaan **variaatiossa tehdyt muutokset** verrattuna lentokenttäsimulaattorin luennoilla esitetyn version toteutukseen. Pitäydy ylemmällä abstraktiotasolla; jokaista muutettua ohjelmariviä ei pidä kertoa.

README.DOCX-tiedoston ulkoasun on oltava TAMKin kirjallisten töiden raportointiohjeen mukainen (löydät tämän intran hakutoiminnolla; tähän on tehty valmis Word-template).