Formula

Projektin dokumentti

Miro Aurela

585240

Tietotekniikan koulutusohjelma

Ensimmäinen vuosikurssi

26.4.2017

Yleiskuvaus

Toteutettu projekti on formula-aiheinen vuoropohjainen peli, jossa pelaajat siirtävät autojaan pelilautana toimivassa ruudukossa pelin sääntöjen mukaan. Pelin tarkoitus on kiertää pelilaudan rata ympäri nopeammin kuin vastustaja, jolloin pelin voittaa. Samalla pitää varoa, etteivät lailliset siirtomahdollisuudet lopu kesken, sillä jos näin käy, häviää pelin suoraan.

Vuorollaan pelaajalla on vaihtoehtonaan ensin vaihtaa auton vaihdetta yksi ylöspäin tai alaspäin, millä vaikutetaan auton nopeuteen ja sen kääntösäteeseen. Vaihteen voi myös pitää samana. Tämän jälkeen pelaaja voi valita kolmesta ruudusta siirtovaihtoehtonsa. Pelaaja voi jatkaa samaan suuntaan kuin auto oli aiemmin menossa tai kääntää autoa hieman vasemmalle tai oikealle. Se kuinka paljon auto kääntyy, riippuu auton vaihteesta.

Peli pyrittiin alun perin toteuttamaan vaikealla tasolla. Lopullinen versio täyttää kaikki helpon ja keskivaikean tason vaatimukset sekä suurimmilta osiltaan myös vaikean tason. Ominaisuuksiin kuuluu esimerkiksi: graafisen käyttöliittymän, ratojen lataamisen tiedostosta, kuljettajien eli pelaajaprofiilien tallentamisen ja lataamisen sekä ratojen ennätyslistojen ja pelaajien ennätysten tallentamisen ja lataamisen. Myös itse pelin toiminta on ohjeistuksen mukainen.

Vaikeaan tasoon kuuluvana omana extrana on toteutettu tehtävänannossa ehdotettu rataeditori sekä ylimääräiset maastotyypit. Näiden lisäksi peliin on toteutettu joitakin pieniä omia lisäyksiä, kuten esimerkiksi siirron perumismahdollisuus pelissä. Peliin luotiin neljä uutta maastotyyppiä: jää, öljy, hiekka ja syvä hiekka, jotka kaikki omien sääntöjensä mukaan rajoittavat liikkumista. Vastaavien uusien maastotyyppien lisääminen peliin on suhteellisen helppoa. Lisäksi peliin toteutettiin neljä erilaista maalityyppiä, jotka mahdollistavat sekä vaaka- että pystysuorien maaliviivojen tekemisen sekä radan kiertosuunnan säätämisen myötä- tai vastapäivään. Myös vaikeaan tasoon kuuluva tekoäly on toteutettu, mutta sen toiminta jäi osittain virheelliseksi. Tekoälypelaaja osaa kiertää rataa törmäämättä seiniin ja kulkee lisäksi suurimman osan ajasta järkeviin suuntiin. Kuitenkin koneen tekoälyn vaihteidenvaihto on omituista, eikä kone käytä isoja vaihteita, minkä takia se usein häviää pelin.

Käyttöohje

Peli käynnistetään ajamalla tiedosto Formula/src/gui/Ikkuna.scala (Run as… Scala application), jolloin ruudulle aukeaa ikkuna, jossa on pelin päävalikko. Ikkunassa on neljä nappulaa ”Uusi peli”, ”Profiilit ja radat”, ”Rataeditori” ja ”Ohjeet”. Ikkunan oikeassa yläreunassa on lisäksi kolmen dropdown-valikkoa nimeltään ”Pelaaja 1: Sininen”, ”Pelaaja 2: Punainen” ja ”Rata”. Ikkunan voi sulkea painamalla vasemman yläkulman rastia.

Kun käyttäjä haluaa pelata itse peliä, hän ensin valitsee oikean yläkulman valikosta kummallekin pelaajalle profiilin sekä radan, jota halutaan pelata. Pelaaja voi myös valita pelaavansa ilman profiilia, jolloin tuloksia ei tallenneta tai toiseksi pelaajaksi voidaan valita tekoäly. Kaikki kolme valikkoa ovat radio button -valikoita, minkä ansiosta käyttäjä voi valita kustakin valikosta vain yhden vaihtoehdon kerrallaan. Valitsemalla lopuksi ”Uusi peli” pystyy aloittamaan itse pelin pelaamisen. Peli luodaan valittujen tietojen perusteella tai annetaan virheilmoitus, jos valinnat olivat virheellisiä. Esimerkiksi samaa profiilia ei voi valita kummallekin pelaajalle.

Kun rata ja pelaajat ovat valittu, varsinainen peli alkaa. Ikkunassa näkyy ruuduista koostuva pelilauta, jossa on kaksi autoa sinisenä ja punaisena ympyränä kuvattuna. Pelilaudan oikealla puolella lukee vuorossa olevan pelaajan profiilinimi ja väri. Vuorossa olevalle pelaajalle näytetään kolme nykyisen vaihteen siirtovaihtoehtoa neliöillä kuvattuna. Iso neliö merkitsee ruutua, jonka pelaaja voi valita ja pieni neliö ruutua, johon pelaaja auton suunnan puolesta saisi liikkua, mutta on esimerkiksi radan ulkopuolella ja siksi laiton siirto. Vuorossa oleva pelaaja voi vaihtaa autonsa vaihdetta painamalla pelilaudan alapuolella olevia nuolinappuloita, jolloin ruudulla näkyvät siirtovaihtoehdot päivittyvät näyttämään tämän vaihteen mahdollisuudet. Kun pelaaja haluaa tehdä siirron, hän klikkaa ruutua, johon on mahdollista liikkua, jolloin auton paikka laudalla vaihtuu. Hyväksyttävän siirron jälkeen vuorossa olevaa pelaaja vaihdetaan.

Peli jatkuu näin, kunnes jompikumpi pelaajista ylittää maalin (tasoittava vuoro tarjotaan toiselle pelaajalle) tai vuorossa oleva pelaaja ei voi siirtää. Ikkunaan aukeaa ilmoitus pelin päättymisestä, josta valitsemalla Ok tai Cancel voi päättää pelin ja palata päävalikkoon tai jäädä peliin, perua siirtoja ja jatkaa peliä. Pelin perumiseen tarkoitettu nappula on vuorossa olevan pelaajan ilmaisevan tekstin alla ja sitä voi käyttää pelin aikana, myös ennen pelin päättymistä. Peru-nappulan alla on myös lopeta-nappula, jota painamalla aukeaa varmistusviesti, joka kysyy, haluaako käyttäjä varmasti lopettaa pelin kesken. Jos käyttäjä valitsee haluavansa, peli päättyy ja päävalikko asetetaan näkyviin, muutoin peli jatkuu edelleen.

Päävalikossa ollessaan voi käyttäjä halutessaan myös tarkastella profiilien ja ratojen ennätyslistoja valitsemalla ”Profiilit ja radat”. Käyttäjälle aukeaa tyhjä ruutu, jonka yläreunassa on kolme dropdown-valikkoa otsikoilla ”Menu”, ”Profiilien tiedot” ja ”Ratojen tiedot”. Menu-valikon avaamalla saa käyttäjä vaihtoehdoikseen ”Uusi profiili” ja ”Palaa päävalikkoon”. Uuden profiilin luodessa käyttäjä valitsee profiilille nimen ja jos nimi on hyväksyttävä, uusi profiili luodaan. Valitsemalla valikon profiili- tai ratavalikon aukeaa käyttäjälle lista valintansa mukaan profiileista tai radoista, joista valitsemalla ruudulle näytetään tiedot profiilin pelitilastosta tietyllä radalla tai tietyn radan lyhyimmät kierrosajat.

Päävalikosta käyttäjä voi halutessaan avata rataeditorin, jolloin ikkunaan aukeaa tyhjä ruutu, jonka yläreunassa on valikot ”Menu”, ”Uusi rata”, ”Kopioi rata”, ”Muokkaa rataa” ja ”Lisää maasto”. Käyttäjä voi eri valikoista valita luovansa uuden radan valitsemalla vaihtoehdoista tietyn koon (esimerkiksi 24x24) ja sitten nimeämällä radan. Tämän lisäksi käyttäjä voi muokata jo olemassa olevaa rataa suoraan, luomatta uutta rataa. Tai aloittaa vanhan radan pohjalta uuden radan eri nimellä. Kun muokattava rata on avattu, se piirtyy ruudulle samoin kuin peliä pelatessa, paitsi että radalla ei ole autoja. Käyttäjä voi nyt valita maastovalikosta maastotyypin, jota haluaa käyttää ja klikkaamalla ruutua vaihtaa sen ruudun maastotyypin valitsemakseen maastoksi. Kaikki käyttäjän tekemät muutokset tallennetaan välittömästi sekä pelin muistiin että tiedostoon, eikä erillistä tallennusnappulaa siis tarvita. Kun käyttäjä on valmis muokkauksessaan, valitsee hän menu-valikosta ”Palaa päävalikkoon”.

Viimeinen päävalikon neljästä napista on ”Ohjeet”, josta painamalla pelaajalle yksinkertaisesti aukeaa sivu, jossa kolmella lyhyellä kappaleella on pyritty selittämään itse pelin perussäännöt. Lisäksi sivulla on kuvattu peliin luotujen erikoismaastoihin liittyvät säännöt. Päävalikkoon voi palata valitsemalla menu-valikosta ”Palaa päävalikkoon”.

Ohjelman rakenne

Ohjelman luokat on jaettu viiteen pakettiin: gui, peli, pelikomponentit, siirrot, tietojenTallennus, jotka kaikki hoitavat omaa tehtäväänsä pelin toiminnassa. Alla on esitelty tarkemmin niihin kuuluvat luokat ja yksittäisoliot.

gui

gui-paketti sisältää kaikki peli käyttöliittymää koskevat luokat ja oliot, kuten käynnistysolion Ikkuna, pelilautaa graafisesti kuvaavan luokan Ruudukko, käyttöliittymän ja pelitilanteen välistä kommunikaatiota hallitsevan olion NappuloidenHallinta, sekä profiileille, radoille ja maastoille omat dropdown-valikot.

Ikkuna-oliossa on määritelty kaikki eri näkymät, jotka ohjelmaa käyttäessä voi kohdata, kuten päävalikon, itse pelin, profiilien ja ratojen tietoja esittelevän sivun, rataeditorin ja ohjeet. Rataeditorin ja pelilaudan näyttämiseen käyttää ikkuna hyväkseen luokkia Rataeditori ja Ruudukko. Osa dropdown-valikoista ovat määritelty NappuloidenHallinnassa, vaikka oikeastaan niillekin luonnollisempi paikka olisi ollut Ikkunassa. Dropdown radio button -valikot: RataMenu, MaastoMenu ja PelaajaMenu toteutettiin jokainen omaksi luokakseen, koska niissä oli joitain erilaisuuksia. Yhteyksiä on kuitenkin niin paljon, että niille olisi näin jälkikäteen ajatellen kannattanut toteuttaa vähintäänkin yhteinen yliluokka. Ikkuna-olio hallitsee sekä päävalikon että pelin nappuloiden painamiset, useimmiten ohjaamalla toteutuksen NappuloidenHallinta-oliolle.

NappuloidenHallinta luotiin alun perin nimenomaan säilyttämään tallessa funktioita, jotka toteuttaisivat nappuloiden painamisista aiheutuvia asioita. Myöhemmin sitä käytettiin myös muiden pelin ajan tapahtumien toteuttamiseen. Esimerkiksi siirtoihin pelilaudan ruutuja klikatessa ja pelitulosten tallentamiseen pelin päättyessä. Myös osa pelin valikoista ja niiden toiminnoista toteutettiin NappuloidenHallintaan. Loppujen lopuksi luokka on siis nimestään huolimatta lähinnä yhteys käyttöliittymän ja pelin logiikkaa ohjaavien luokkien välillä.

Ruudukko-luokka määritellään Pelitilanteen avulla ja sen tärkein tehtävä onkin näyttää visuaalinen kuva pelin tilanteesta käyttäjälle. Ruudukon metodit käsittelevät pelkästään eri asioiden (ruudukko, maastot, autot, siirtovaihtoehdot) piirtämistä ruudulle ja ruutujen klikkaamisten vaikutuksia. Aina pelilautaa painettaessa lasketaan hiireen sijainnin perusteella klikattu Koordinaatti ja siitä lähetetään NappuloidenHallinta-olion kautta tieto Pelitilanteelle, joka yrittää tehdä siirron. Pelaajan siirron jälkeen tarkistetaan Pelitilanteelta myös tieto siitä, onko peli päättynyt ja reagoidaan tarvittaessa siihen. Jos peli ei ole päättynyt, viimeisenä tarkistetaan, onko tietokonepelaajan vuoro pelata, ja ohjataan se tekemään siirtonsa, jos näin on.

Rataeditori-luokka on samoin kuin Ruudukko, tarkoitettu kuvaamaan tiettyä rataa. Vaikka pohjimmiltaan ne molemmat piirtävät Maastoja, hakee Ruudukko tietonsa Pelitilanteen Pelilaudan ja Ruudun kautta, kun taas Rata pääsee suoraan käsiksi Maastoihin. Itse piirtäminen toimii kuitenkin täsmälleen samalla tavalla. Lukuun ottamatta sitä, että Rataeditori ei piirrä autoja tai siirtovaihtoehtoja. Myös klikatun ruudun laskeminen toimii samalla tavalla, vaikka efekti on eri, ja tämän takia parempi tapa olisi ollut toteuttaa esimerkiksi yläluokan avulla, jolloin toisteisuudesta oltaisiin päästy eroon.

peli

peli-paketti pitää sisällään yksittäisolion Peli, joka kuvaa koko ohjelmaa ja luokan Pelitilanne, joka kuvaa yksittäistä ottelua. Nämä kaksi ovat Ikkunan ja NappuloidenHallinnan lisäksi kenties ohjelman

tärkeimmät luokat, sillä ne pitävät sisällään suurimman osan pelin keskeisestä toiminnasta. Lisäksi peli sisältää omat luokat Pelaajalle ja siihen mahdollisesti kuuluvalle AI:lle eli tekoälylle.

Yksittäisolio Peli lataa ohjelman käynnistyessä pelin tiedostoista tallennetut Radat ja Profiilit. Näiden tietojen lisäksi Peli pitää Option-kääreessä tallessa sen hetkistä Pelitilannetta, jos peli sillä hetkellä on käynnissä. Pelillä on metodit uuden pelin aloittamiselle sekä uuden profiilin tai radan tallentamiselle. Jälkimmäisessä kahdessa metodissa, kuten myös tietojen lataamisessa Peli tukeutuu tietojenTallennus-pakettiin.

Pelitilanne määrittyy Pelilaudan sekä Pelaaja-listan avulla. Se pitää näiden lisäksi myös kirjaa siitä, kumpi Pelaajista on vuorossa ja tekoälyn ollessa vuorossa. Käyttäjän siirrot toteutuvat kutsumalla NappuloidenHallinta-oliosta Pelitilanteen metodia siirraAutoa. Ja vastaavasti siirron perumiseen on oma metodinsa. Pelitilanteella on muun ohella metodi tarkistamaan, onko peli päättynyt ja kertomaan voittaja, jos on. Pelitilanteella on myös kumppaniolio, jonka kuvitteellinen-funktiolla, AI voi luoda pelitilanteestaan kopion, jonka avulla se voi suunnitella tulevaa peliään.

Pelaaja on luokka, jolla Profiili ja Auto yhdistetään toisiinsa. Pelaajalla saattaa olla myös määritelty tekoäly AI, jos kyseinen pelaaja on tietokoneen ohjaama. AI-luokka määritellään siihen liittyvän Pelitilanteen avulla, koska valitakseen siirtonsa, täytyy tekoälyn tietää, millainen tilanne pelissä on milläkin hetkellä. Tämän Pelitilanteen pitää olla määritelty ennen kuin AI voidaan luoda. Siis kun Peli-luokassa luodaan uuttaa Pelitilannetta, luodaan ensin Autot ja Pelaajat, joiden avulla määritellään Pelitilanne ja vasta sen jälkeen lisätään tekoäly tietokonepelaajalle Pelaaja-luokan lisaaTekoaly-metodilla.

pelikomponentit

pelikomponentit sisältävät Pelilaudan, Auton, Maaston ja Ruudun. Ne liittyvät läheisesti toisiinsa, sillä pelilauta koostuu ruuduista, jotka ovat jokainen tiettyä maastotyyppiä ja saattavat sisältää auton.

Pelilauta luodaan alun perin Rata-luokan pohjalta. Kuitenkin toisin kuin Rata, jolla on tallessa vain Maastot, Pelilauta koostuu Ruuduista, jotka pitävät tallessa tietoa sekä pysyvästä Maastosta että ruudussa mahdollisesti olevasta Autosta. Pelitilanteella on metodit autojen arpomiselle lähtöruutuihin pelin alussa sekä metodit autojen siirtämiselle. Tähän voidaan käyttää metodia siirraAutoaLaillisesti, joka palauttaa totuusarvon sen mukaan onko siirto laillinen ja siirtää autoa vain siinä tapauksessa, että se on sallittua tai metodia siirräAutoaPakolla, joka nimensä mukaisesti tekee siirron huolimatta siitä, onko se sääntöjen mukainen vai ei. Tietyn siirron laillisuus pystytään tarkistamaan, koska Pelilauta osaa laskea tietyn auton kaikki sallitut siirrot jokaisella hetkellä. Muita Pelilaudan tehtäviä ovat tietyn auton sijainnin etsiminen sekä maalinylitysten valvominen.

Ruutu on luokka josta, kuten aiemmin sanottiin, Pelilauta koostuu. Ruudun tehtävä on lähinnä helpottaa Pelilaudan toimintaa ja sen metodit liittyvät oman Maastotyypin kertomiseen, auton ruudussa olon selvittämiseen, auton poistoon ruudusta tai lisäämiseen ruutuun. Maastoihin liittyvät kysymykset luokka ohjaa eteenpäin itse Maastolle, joka on sealed abstract class. Samassa tiedostossa on määritelty kaikki pelin maastotyypit, niiden merkit Ratojen tiedostoformaatissa sekä säännöt liikkumiselle. Samassa paikassa olisi ollut järkevää myös määritellä graafisessa käyttöliittymässä käytettävä maasto väri, mutta tähän toteutukseen ei kyseinen ominaisuus päässyt mukaan. Sen sijaan värit määritellään piirtämisen yhteydessä erikseen sekä Ruudukko- että Rataeditoriluokissa, mikä aiheuttaa toisteisuutta koodiin.

Viimeinen pelikomponentti-pakettiin kuuluvista luokista on Auto, joka pitää kirjaa vaihteestaan sekä siirroistaan. Siirtojen tallennusta käytetään kokonaissiirtomäärän laskemiseen, auton suunnan määrittämiseen sekä vuorojen perumiseen. Auton tärkeimmät metodit ovat vaihteiden nostaminen ja laskeminen, jos se sääntöjen mukaan on sallittua sekä omien sallittujen suuntiensa päätteleminen.

Siirrot

Siirto-paketista löytyvät luokat: Koordinaatti, Siirto ja Suunta. Koordinaatti on case class, joka kuvaa yksinkertaisesti riviä ja koordinaattia merkitseviä kokonaislukujen yhdistelmää. Se ei luokkana ole kovin laaja, mutta se on siistimpi kuin vaihtoehtoinen tapa käyttää tyyppiä Tuple2[Int, Int] aina koordinaattien merkitsemiseen. Koordinaatilla on Pelilaudan parametriksi ottava metodi, joka tarkistaa, onko pelilaudalla kyseistä koordinaattia sekä metodit koordinaattien yhteen- ja vähennyslaskulle.

Siirto-luokka yhdistää kaksi koordinaattia: lähdön ja kohteen. Siirrosta on eri muuttujissa paljon tietoa tallessa, kuten x- ja y-suuntainen liike, siirtoon tarvittava vaihde sekä siirron kulma. Siirron tärkeimpiin ominaisuuksiin kuuluu: siirron kanssa mahdollisimman samansuuntaisen Suunnan laskeminen mille tahansa vaihteelle, siirron naapurisiirrot vaihteen pysyessä samana sekä koordinaatit, joiden läpi siirto kulkee. Siirron alalaji on Suunta, joka alkaa aina origosta ja sen siksi pystyy alkukoordinaatin antamalla muuttamaan vastaavaksi Siirroksi. Suunnan pystyy luomaan kulman ja vaihteen perusteella.

tietojenTallennus

tietojenTallennus taas sisältää luokat: Profiili, pelilaudan yleinen malli: Rata ja TiedostonHallinta, jota käytetään tiedostojen lukemiseen ja kirjoittamiseen.

Profiili-luokka tallentaa saman käyttäjän pelitulokset yhteen paikkaan. Se pitää tallessa pelattujen pelien määrää, voitettujen pelien määrää ja ennätyskierrosta, joka rataa kohden. Sen tärkein metodi on paivita, joka päivittää Profiilin tiedot tietyn pelin tuloksen perusteella ja sitten kutsuu TiedostonHallintaa päivittämään saman muutoksen tiedostoihin. Profiililla on myös kumppaniolio, jonka apply-metodi on hyödyllinen, kun Profiilien tietoja ladataan tiedostosta. Se ottaa parametreikseen Profiilin nimen sekä tallennetun tiedoston luetut rivit Vector[String]-muodossa ja luo niiden perusteella Profiilin, jolla on oikeat tiedot.

Radat toimivat hyvin samalla tavoin kuin Profiilit. Rata-luokalla on myös tallessa joitain tietoja, kuten nimi, radan maastot Vector[Vector[Maasto]]-muodossa sekä parhaat kierrosajat ja ne suorittaneet Profiilit. Lisäksi samoin kuin Profiililla, myös Radalla on paivita-metodi, joka lisää uudet kierrosajat Radan tietoihin ja tallentaa tiedot myös Rataa vastaavaan tiedostoon. Rata-luokan kumppanuusoliolla on kolme erilaista metodia uuden Radan luomista varten. Yksi tiedostosta haetuilla tiedoilla varten luomiseen, toinen annetusta Radasta kopion luomiseen ja kolmas tyhjän tietyn kokoisen Radan luomiseen. Kahta jälkimmäistä hyödynnetään rataeditorin yhteydessä.

TiedostonHallinta-olio hoitaa Ratojen ja Profiilien tietojen lataamisen tiedostoista ja tallentamisen tiedostoihin. Oliolla on erikseen metodit sekä Radoille että Profiileille, jotka sitten lukevat tai tallentavat annetut tiedot yhteisillä metodeilla lueTiedosto, lueRivit ja kirjoitaTiedostoon. Myös uuden Rata- tai Profiili-tiedoston luominen on mahdollista.

Algoritmit

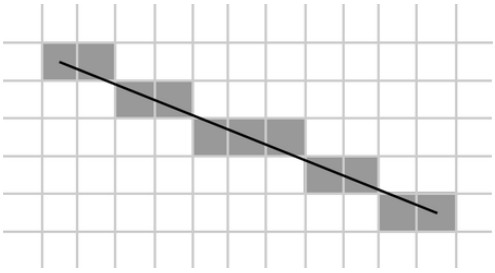
Sama suunta eri vaihteella

Yksi Siirto-luokan tehtävistä on laskea tietyn siirron jälkeen eri vaihteilla Suunta, joka on mahdollisimman lähellä tehdyn Siirron suuntaa. Tähän käytetään Siirron metodia samaSuunta sekä Suunnan kumppaniolion apply-metodeja. Jos vaihde pidetään samana, on Suunta, johon auto on kulkemassa selvästikin sama kuin edellisen Siirron suunta ja saadaan vain muuttamalla Siirto Suunnaksi. Jos taas vaihdetta on vaihdettu, täytyy pystyä etsimään sellainen Suunta, joka on mahdollisimman lähellä edellisen siirron suuntaa.

Tämä on toteutettu niin, että ensin lasketaan edellisen Siirron kulma väliltä 0-2*Pi. Tämän jälkeen käydään läpi yksi kerrallaan jokainen tietyllä vaihteella määritellyistä siirtosuunnista ja verrataan jokaisen tällaisen suunnan kulmaa edellisen kierroksen Siirron kulmaan. Pidetään muistissa jokaisen suunnan käytyä, millä jo käydyistä suunnista kulmien erotus oli pienin (eli lähimpänä Siirron suuntaa) ja jatketaan, kunnes kaikki tietyn vaihteen suunnat on käyty läpi. Kun koko lista on käyty läpi, ollaan saatu lähimmäksi osunut vaihtoehto. Algoritmi on mielestäni hyvin toimiva sekä järkevä ja sitä jopa suositeltiin tehtävänannossa antamalla vihjeeksi, että sama suunta on helpoin selvittää laskemalla siirtojen arkussinit.

Koordinaatit, joiden läpi Siirto menee

Toinen Siirto-luokan tärkeimmistä tehtävistä on selvittää jokaiselle Siirrolle sen läpäisevien Koordinaattien lista. Tämä tulkittiin projektia toteuttaessa niin, että siirto kulkee kaikkien niiden koordinaattien läpi, joiden ruutuja leikkaa lähtökoordinaatin keskipisteestä lähtevä ja kohdekoordinaatin keskipisteeseen päättyvä suora. Alkuperäinen idea algoritmille saatiin Bresenhamin algoritmista. Kuitenkin poiketen Bresenhamin algoritmista, jonka tulosta oheinen kuva esittää, projektin vaatimassa algoritmissa pitäisi läpäistäviksi ruuduiksi laskea myös ruudut, joita suora leikkaa vain vähän. Siis kuvan tilanteessa väritettyinä pitäisi olla myös ruudut koordinaateissa (2,2), (5,2), (7,4) ja (10,4), kun koordinaatiston origona pidetään vasemman yläreunan ruutua.



https://fi.wikipedia.org/wiki/Bresenhamin_algoritmi#/media/File:Bresenham.png,

25.4.17

Toteutetun algoritmi perusperiaate oli seuraava. Ensin alustettiin lista, johon koordinaatit tultaisiin säilömään. Sen jälkeen tarkistettiin, oliko kyseessä pystysuora siirto, sillä algoritmissa pitää laskea suoralle kulmakerroin. Koska pystysuoralla ei kuitenkaan ole kulmakerrointa, erotettiin tämä omaksi erikoistapauksekseen ja siihen kuuluvat koordinaatit saatiin yksinkertaisesti käymällä y-akselin arvot läpi aloituskoordinaatin y-arvosta lopetuskoordinaatin y-arvoon, pitäen x-arvo samana.

Jos kyseessä ei ole pystysuora, sille määritetään ensin kulmakerroin ja sen jälkeen funktio, joka x-koordinaatin perusteella palauttaa y-koordinaatin. Koska suoran pitää lähteä ruudun keskipisteestä, saatiin sen yhtälöksi: $y = kx - 0.5(k-1)$. Näin arvolla $x=0.5$ saadaan kulmakertoimesta riippumatta $y = 0.5k – 0.5(k-1) = 0.5k – 0.5k – 0.5(-1) = 0.5$.

Kun suoran funktio on tiedossa, käydään x=0.5 alkaen x:n arvoja lävitse aina x=0.5+leveys asti, jossa ’leveys’ on siirron leveys x-suunnassa. Leveys voi olla myös negatiivinen, jolloin x-arvoa vähennetään joka askeleella. Jokaiselle x:lle lasketaan suoran funktiosta sitä vastaava y-arvo, ja lisätään

kyseinen (x,y)-koordinaatti alussa määriteltyyn listaan, jos se ei vielä kuulunut sinne. X:n arvoja tulee käydä riittävällä tarkkuudella, sillä esimerkiksi oheisessa kuvan tilanteessa pitää aiemmin mainittujen koordinaattien löytämiseksi x:n arvon kulkea hyvin pienissä askelissa. Riittäväksi askelpituudeksi valittiin 0.01, joka testeissä osoittautui antavan oikeat tulokset.

Koska funktion arvot voivat olla mitä vain reaalilukuja, mutta tulokseksi haluttavat koordinaatit ovat kokonaislukuja, täytyy tämä muutos vielä huomioida koordinaatteja lisätessä. Positiivisia koordinaatteja pyöristäessä katkaisupyöristys toimii hyvin ja esimerkiksi luku 2,7 pyöristetään kokonaisluvuksi 2 aivan oikein. Kuitenkin negatiivisilla luvuilla esimerkiksi -0.6 pyöristyy katkaistaessa kokonaislukuun 0, mikä on väärin, sillä -0.6 ja 0.6 kuuluvat selvästikin eri ruutuihin. Siispä negatiivisista luvuista vähennettiin ennen katkaisua luku yksi, jolloin saadaan: -0.6-1 = -1.6 = -1. Kun koko liikkeen väli on käyty läpi, palautetaan saatu koordinaattilista.

Sallitut siirrot

Jotta pelin siirtojen laillisuutta pystytään valvomaan, täytyy tietää mihin suuntiin auto tällä vuorolla saa liikkua sekä, minkä ruutujen läpi siirtovaihtoehdot menevät. Esimerkiksi auto ei voi siirrollaan käydä radan ulkopuolella, vaikka siirto päättyisikin takaisin radalle. Näiden tarkistamiseen käytetään hyväksi tietoa siitä, mitkä suunnat ovat lähimpänä auton edellistä siirtoa sekä minkä pelilaudan koordinaattien läpi siirrot kulkevat. Nämä tiedot saadaan edellä selitettyjen algoritmien avulla.

Sallittujen siirtojen etsiminen on vaiheittain prosessi, jossa siirtovalikoimaa kutistetaan, kunnes lopuksi vain sallitut siirrot ovat jäljellä. Ensin selvitetään, mihin suuntaan vuorossa oleva auto on menossa ja mikä suunta mahdollisesti vaihdetulla vaihteella on lähinnä tätä. Saadun suunnan perusteella lasketaan sitten suunnat, joihin auto voi kääntyä. Auton sijainti etsitään pelilaudalta ja yhdistetään tieto suuntiin, jolloin saadaan selville jokaisen siirron lähtö- ja kohdekoordinaatit. Kaikista näin saaduista siirroista testataan, ovatko niiden kulkemat reitit sallittuja. Siirron sallittavuudessa otetaan huomioon, ettei siirto saa kulkea radan ulkopuolella, eikä myöskään sellaisen ruudun läpi, jossa toinen auto on. Lisäksi huomioidaan eri maastotyyppien vaikutukset liikkumiseen. Esimerkiksi siirto, joka kulkee hiekkaa sisältävän ruudun läpi, ei saa olla vaihteeltaan suurempi kuin edellisen siirron vaihde oli. Kaikki siirrot, jotka eivät toteuta jotakin ehdoista hylätään, kun taas ehdot toteuttavat siirrot palautetaan sallittuina siirtoina.

Kuvitteellinen Pelitilanne

Pelitilanteen kumppanioliolla on metodi, joka luo parametrina annettavan tekoälyn perusteella kopion Pelitilanne-luokasta, jossa tekoäly on mukana. Tekoäly käyttää kopiota hyväkseen, suunnitellessaan seuraavia siirtojaan. Kopion luominen on tarpeellista, etteivät kuvitteelliset muutokset vaikuta oikeaan pelitilanteeseen ja jotteivat tekoälyn laskemat pelin etenemisvaihtoehdot vilkkuisi visuaalisella pelilaudalla tekoälyn miettiessä.

Kopion valmistelu alkaa luomalla uusi Pelilauta-luokan ilmentymä sekä uudet autot. Autot asetetaan samaan tilanteeseen kuin ne oikeassa pelissä ovat käymällä siihen asti pelattu peli läpi siirto kerrallaan. Tämä onnistuu, koska jokainen Auton ilmentymä pitää tallessa pelin aikana tekemiään siirtoja. Ensin oikean pelitilanteen autojen lähtökoordinaatit selvitetään ja uudet autot asetetaan vastaaviin ruutuihin. Sitten autot toteuttavat vuorotellen oikean pelitilanteen autojen tekemät siirrot. Näin kuvitteellisessa tilanteessa ei ole mitään eroa oikeaan tilanteeseen. Uudet Pelaajat luodaan

väliaikaisten Profiilien ja uusien Autojen avulla ja lopuksi itse Pelitilanne luodaan Pelaajien ja Pelilaudan avulla.

Koska kuvitteellisessa pelitilanteessa ei tarvitse kaikkia tietoja siitä, miten peli siihen asti on edennyt, olisi tämän ominaisuuden voinut toteuttaa jollain toisellakin tavalla. Tekoälylle riittävä informaatio olisi autojen sijainti ja suunta sekä vuorossa oleva pelaaja. Siispä sen sijaan että autoja olisi konkreettisesti siirrelty Pelitilannetta luodessa, olisi autot vain voitu asettaa suoraan koordinaatteihin, joissa ne toisella laudalla ovat. Kun Autoille olisi vielä kopioitu siirtohistoriasta viimeisen siirron, olisi sen perusteella pystytty määrittämään seuraavat mahdolliset siirrot. Vaikka tämä tapa olisi kenties ollut yksinkertaisempi, sen sijaan valittiin edellisessä kappaleessa selitetty tapa, koska tuntui mielekkäältä käyttää jo olemassa olevia auton siirtämiseen tarkoitettua metodeja. Jälkikäteen ajatellen olisi metodi kuitenkin ollut siistimpi toteuttaa tämän vaihtoehtoisen tavan mukaan.

Tekoäly

Peliin on toteutettu tekoäly, jota vastaan ratoja pystyy pelaamaan. Valitettavasti kuitenkin valmiissa versiossa tekoälyn kehitys ei ehtinyt aivan niin pitkälle kuin oli tarkoitus, eikä siis pelaa kovin hyvin nykyisessä versiossa. Tässä esitellään kuitenkin tekoälyn parhaan siirron etsimisen algoritmi niin kuin se on tällä hetkellä toteutettuna.

Tekoälyn etsintä parhaalle siirrolle perustuu rekursiiviseen funktioon, joka etsii parhaan jatkon tietylle tilanteelle. Funktiolle annetaan parametriksi Pelitilanne ja lista, jossa on suunnitelmaan siihen asti kuuluvat Siirrot. Funktiossa alustetaan lista, johon eri jatkovaihtoehdot tullaan keräämään. Sen jälkeen kaikki tilanteen lailliset siirrot käydään läpi ja tarkastetaan niiden parhaat mahdolliset jatkot.

Paras mahdollinen jatko määritellään sen mukaan, kumpi pelaaja on vuorossa. Tekoälyn ollessa vuorossa on paras mahdollinen siirto sellainen, joka johtaa mahdollisimman hyvään lopputulokseen tekoälyn kannalta ja pelaajan ollessa vuorossa mahdollisimman hyvä tilanne on sellainen, josta tekoäly kärsii eniten. Jokaisen kuvitteellisesti tehdyn siirron jälkeen kutsutaan erillistä funktiota, joka arvostelee sen hetkisen tilanteen. Positiivisella luvulla palautetaan, jos kyseessä on voitto tietokoneelle ja negatiivinen, jos kyseessä on tappio. Lisäksi mitä pienempi luku on, sitä parempi tulos on: positiivisilla luvuilla tulos on voittoon tarvittavien siirtojen määrä, kun taas negatiivisen luvun itseisarvo kertoo, kuinka monta kierrosta kuluu ennen tappiota. Luvulla nolla merkitään tilannetta, jota ei ole todettu voitoksi tai tappioksi.

Funktio palauttaa aina sekä parhaan siirtosarjan että siihen liittyvän kokonaislukuarvion sarjan hyvyydestä. Kun kaikki tietyn tilanteen jatkot on käyty läpi, verrataan vaihtoehtoisten siirtosarjojen arvoja, valitaan vuorossa olevan pelaajan näkökulmasta paras ja palautetaan tieto siitä. Tätä rekursiivisesti jatkamalla saadaan lopulta paras siirtosarja alkuperäiselle pelitilanteelle.

Kuten alussa todettiin, algoritmi ei kokonaisuudessaan valmistunut. Osa tekoälyn tekemistä siirroista eivät vaikuta parhailta mahdollisilta ja tekoäly myös pääsee tähän tulokseen varsin hitaasti. Jälkimmäisen seikan seurauksena jouduttiin tekoälylle laittamaan rajoitus, joka estää sitä jatkamasta tilanteen tarkastelua yli viiden pelikierroksen, vaan tulkitsee tällaiset jatkot aina arvoltaan nollaksi. Tämä taas osaltaan vaikuttaa siihen miksi tietokoneen siirrot eivät ole täydellisiä.

Tietokoneen laskennan tehostamiseksi olisi voitu laskea pelkästään tietokoneen omia siirtoja, jolloin se olisi kohtuullisessa ajassa päässyt 10 kierroksen mietintään. Tämäkään ei kuitenkaan olisi

riittänyt koko pelikentän läpikäyntiin. Lisäksi toisen pelaajan siirtojen huomioimatta jättäminen olisi voinut aiheuttaa sen, että tekoäly ajaa laidan yli vastustajan siirtojen takia. Tämän takia algoritmin alun perin todettiinkin vaativan kummankin pelaajan siirtojen tarkastelun.

Olisi algoritmi toteutettu kummallaan vaan tavalla, on selvää, että itse periaate, jolla ongelmaa yritettiin ratkaista ei ollut toimiva. Vaihtoehtoinen lähestymistapa asiaan olisi voinut olla esimerkiksi laskea etukäteen kaikille Radan sijainnin ja Suunnan yhdistelmille arvo ja valita jokaisella vuoroilla vaihtoehtoisista siirroista se, joka päättyy parhaaseen ruutuun. Tilanteen hyvyys olisi voitu määritellä esimerkiksi arvolla, joka kertoo kuinka monta vuoroa tilanteesta kestää päästä maaliin. Tässäkin kuitenkin tulisi ongelmaksi vastustajan sijainnin huomioon ottaminen.

Tietorakenteet

Projektissa käytetään käyttötarkoituksesta riippuen tietorakenteena Vectoria, Arrayta, Bufferia, Listiä tai Mapia. Tietoa säilytetään Vectorina esimerkiksi Pelilaudan ruudukon tallentamisessa. Pelilauta nimittäin koostuu Ruutu-luokan ilmentymistä, jotka kokoelmassa pysyvät aina samana. Autojen liikkuminen ruudukossa kuvataan Ruudun Option-tyyppisenä muuttujana, joka kertoo ruudussa olevan auton, jos sellainen on. Tämän lisäksi lähes kaikki kokoelmia palauttavat metodit, palauttavat tietonsa Vector-tyyppisenä, sillä näitä palautusarvoja ei koskaan joudutaan muokkaamaan. Tämä tekee Vectorista kenties käytetyimmän tietotyypin.

Buffer-alkiokokoelmaa käytetään auton tallentaman kuljetun reitin merkitsemiseen, sillä sitä pitää pystyä helposti kasvattamaan joka vuorolla. Lisäksi Buffereita käytetään väliaikaisesti tietyissä algoritmeissa, vaikka niitä pysyvästi ei käytetäkään tiedon säilömiseen. Esimerkkejä tällaisista algoritmeista ovat tekoälyn parhaan siirron valinta sekä Siirto-luokan lasku siirron läpäisevistä Koordinaateista. Algoritmeja yhdistää se, että niissä silmukan aikana listan sisältöön jatkuvasti lisätään uusia alkioita. Koska Buffer-tietotyyppiin pystyy helposti lisäämään alkioita, on se erityisen sopiva edellä mainittuihin tilanteisiin.

Arrayta käytetään esimerkiksi Rata-luokassa. Jokainen Rata koostuu kaksiulotteisesta taulukosta Maastoja, jotka kuvaavat rataa. Koska Ratoja on tarkoitus pystyä muokkaamaan suoraan rataeditorista käsin, jopa ohjelman suorituksen aikana, valittiin sen tietotyypiksi juuri Array.

Listin käyttö rajoittuu tilanteisiin, joissa kokoelman n:ttä alkiota ei ole tarpeellista tietää, sillä kyseinen operaatio ei ole tehokas List-kokoelmalla. Kuitenkin alkioiden lisääminen listan loppuun on suhteellisen helppoa, ja ensimmäisen alkion hakeminen sitäkin helpompaa. Siispä ohjelmassa Listiä käytetään AI-luokassa tekoälyn siirronetsimisalgoritmissa eri siirtosarjojen välittämiseen rekursioiden välillä. Tähän olisi voitu myös käyttää Bufferia, mutta koska kokoelman keskimmäisiä arvoja ei haeta koskaan, ei siitä olisi ollut Listiä enempää hyötyä.

Edellä luoteltujen kokoelmien lisäksi myös tietorakennetta Map käytettään ohjelmassa. Profiileihin tallennetut tiedot pelattujen pelien määrästä tietyllä Radalla, voittojen määrästä tietyllä Radalla ja lyhimmästä kierrosajasta ovat kaikki tallennettu Map-tietorakenteella. Esimerkiksi ennätysten tapauksessa radan nimestä (String) saadaan lyhyimmän kierroksen siirtomäärä (Int). Mapin avaimen tyyppinä voitaisiin suoraan käyttää Rataa, mutta koska Profiilin sisältämät tiedot säilytään pelikertojen välissä tiedostossa, tulisi ongelmaksi Rata-olioihin osoittavien viittausten tallentaminen tiedostoon.

Tekstiä sen sijan on helppo tallentaa, ja koska ratojen samaksi nimeäminen on estetty, riittää Radan nimi yksiselitteiseksi tunnistetiedoksi.

Valmiiden tietorakenteiden lisäksi ohjelmassa on joitain luokkia, jotka voidaan tulkita tietorakenteiksi. Tällaisia ovat esimerkiksi Profiili, jossa pidetään tallessa jokaisen käyttäjän tiedot pelien voitoista ja henkilökohtaisista rataennätyksistä, Rata, jolla säilytetään tietoa kentän maastojen sijainneista, Pelilauta, jolla yhdessä Ruutu-luokan kanssa säilytetään tietoa laudalla olevien nappuloiden sijainneista sekä Auto, joka pitää kirjaa nykyisestä vaihteesta ja omista aiemmista siirroistaan. Nämä kaikki ovat ainakin osittain muuttuvatilaisia, vaikka esimerkiksi Profiilin tai Radan nimiä ei voikaan muuttaa. Sen lisäksi ohjelmassa on myös muuttamattomia rakenteita, kuten kahdesta kokonaisluvusta muodostuva Koordinaatti ja ja kahdesta Koordinaatista muodostuva Siirto.

Tiedostot

Ohjelman tiedostoissa piti tehtävän annon mukaisesti olla tallennettuna radat ja jokaisen radan nopeimpien kierrosten lista sekä pelaajaprofiileita, joissa on tallennettuna käyttäjän nimimerkki, pelattujen otteluiden määrä, voitettujen otteluiden määrä sekä ennätyskierroksen siirtomäärä kaikilta pelatuilta radoilta. Ratojen tiedot ovat tallennettuna kansiossa Formula/radat muotoa [radan_nimi].txt olevilla tiedostonimillä. Vastaavasti Profiilien tiedot ovat tallennettu kansioon Formula/profiilit samanlaisella nimeämistyylillä. Sekä Ratojen että Profiilien tiedostot ovat tavallisia tekstitiedostoja.

Profiilit

Profiilien tiedot ovat tallennettu hyvin yksinkertaiseen muotoon, joka on niin ihmisen kirjoitettavissa kuin luettavissakin helposti. Jokaisessa profiilitiedostossa on riveittäin tallessa sen kuvaaman Profiilin pelaamien ratojen tilastot järjestyksessä: radan nimi, voitot radalla, pelit radalla ja lyhyin kierros radalla. Radan nimi on tekstiä ja muut tiedot kokonaislukuja. Lyhyimmän kierroksen kohdalle voi olla merkitty myös viiva (’ - ’), joka tarkoittaa, ettei pelaaja ole koskaan läpäissyt rataa, vaikka onkin pelannut sillä. Tiedot ovat erotettu toisistaan yhdellä välilyönnillä. Profiilin nimi saadaan tiedoston nimestä poistamalla .txt-pääte. Alla esimerkki mahdollisesta profiilitiedostosta:

```
rata1 1 4 -
rata2 4 6 16
```

Tiedostosta käy ilmi, että rataa nimeltä ”rata1” on tällä profiililla pelattu 4 kertaa, joista vain yksi on ollut voitto. Pelaaja ei ole koskaan selvinnyt rataa loppuun asti, vaan peli on aina päättynyt siihen, että hän tai hänen vastustajansa ei voi enää liikkua. Radalla nimeltä ”rata2” taas 6 pelistä voittoja on 4 ja henkilökohtainen ennätys radalla on 16 vuoroa. Kun Profiilia luodaan tiedoston pohjalta kaikki rivit, jotka eivät toteuta esitettyä mallia jätetään huomiotta.

Radat

Yhtä Rataa kuvaava ruudukko ja Radan ennätyslista ovat tallennettuna samassa tiedostossa, toisin kuin alkuperäisen suunnitelman mukaan oli tarkoitus. Molemmat tiedot pystyttiin alkuperäisen ajatuksen vastaisesti yhdistämään helposti yhteen tiedostoon. Tallentaessaan tietoja ohjelma itse kirjoittaa tiedot niin, että ensin Radan tiedot ovat esitetty peräkkäisillä riveillä ja sen alla Radalla ajettua ennätyksiä kuvaavat rivit tiedoston lopuilla riveillä. Todellisuudessa järjestyksellä ei ole väliä, sillä luettaessa

tiedostoa kaikki rivit, jotka täyttävät tarvittavan muotoilun ollakseen ennätysrivejä, tulkitaan ennätysriveiksi, kun taas kaikki loput rivit tulkitaan maastoriveiksi.

Ennätysriveillä on aina ensin pelaajaprofiilin nimi ja sitten kierrosten määrä. Nimen täytyy olla tekstiä ja kierrosten määrän täytyy olla luku. Tiedot on erotettu toisistaan yhdellä välilyönnillä. Rivien ei tarvitse olla järjestyksessä, vaan ne järjestetään vasta kun Rata-luokka on luotu. Radan Maastot taas luetaan niin, että jokainen merkki maastorivillä kuvaa tiettyä maastotyyppiä. Jos joukossa on merkkejä, jotka eivät vastaa yhtäkään Maasto-tyyppiä, tulkitaan kaikki sellaiset merkit tavalliseksi tieksi. Myös jos ratarivien leveys ei ole sama täytetään lyhyimpien rivien loput tavallisella tiellä. Alla on esimerkki mahdollisesta ratatiedostosta:

```
##############################
##           <        ###########
#            <         ##########
#            <          #########
#        #########       ########
##        #########       #######
###        #########         ####
####       ##########          #
####                           #
#####                         ##
#######                      ###
##############################
Matti 20
Maija 14
```

Tiedoston #-merkit kuvaavat radan reunoja, kun taas tyhjä kohdat (välilyönnit) kuvaavat tavallista tietä. <-merkki kuvaa lähtö/maaliviivaa, jossa nuolen kärki osoittaa kiertosuunnan. Radan ovat läpäisseet Profiili nimeltä Matti kierrosajalla 20 ja Maija ajalla 14. Kuten tästä esimerkistä näkee, on tiedostomuoto suhteellisen luettava myös ihmiselle ja kirjoittaminenkin onnistuu, jos vain kärsivällisyyttä riittää merkkien naputteluun. Ratojen luomista varten peliin on toteutettu Rataeditori, jolla Ratojen muokkaaminen on kenties helpompaa. Jos Ratoja haluaa kuitenkin muokata tai luoda suoraan tekstitiedostoista käsin on alla esitelty kaikki merkit, joita eri Maastojen merkitsemiseen käytetään.

Tie = ’ ’

Jää = ’j’

Öljy = ’o’

Hiekka = ’h’

Syvä hiekka = ’H’

Reuna = ”#”

Maali (kiertosuunta ylös) = ’^’

Maali (kiertosuunta alas) = ’v’

Maali (kiertosuunta oikealle) = ’>’

Maali (kiertosuunta vasemmalle) = ’<’

Testaus

Testaussuunnitelman mukaisesti kaikki graafisen toteutuksen testit suoritettiin silmämääräisesti, koska se on yksinkertaisinta. Näihin kuuluivat esimerkiksi käyttöliittymän piirtofunktio sekä käyttöliittymän reagointi nappuloiden, ruudukon ja valikoiden painamiseen. Sen sijaan esimerkiksi mahdollisten siirtojen laskentaa ja sen saavuttamiseksi tehtäviä välivaiheita testattiin useammalla eri testillä. Kaikki koodillisesti toteutetut testit, joita käytettiin löytyvät Formula/src/testaus -paketista. Osassa testeistä on myös mukana lyhyitä kommentteja.

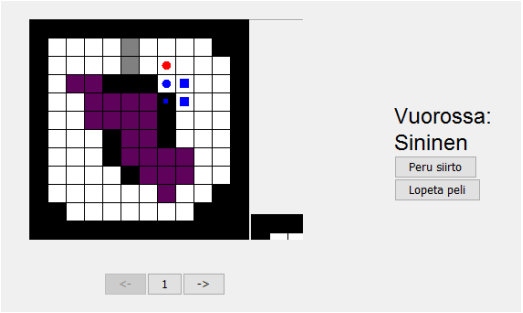
Ohjelman osia testattiin sitä mukaan kuin ne oltiin toteutettu ja seuraavaan vaiheeseen jatkettiin vasta kun edellinen asia näytti toimivan. Osa testeistä läpäistiin aluksi, mutta projektin myöhemmissä vaiheissa huomattiin, että itse asiassa osa ei toiminutkaan oikein. Tällöin testiin palattiin ja siitä pyrittiin tekemään kattavampi. Uudella testillä paikallistetut virheet korjattiin ja sitten palattiin tekemään sitä, mitä ennen ongelman havaitsemista oltiin tekemistä. Valmis ohjelma toteutti kaikki tehdyt testit ja osoitti toimivuutensa myös useissa testipeluutuksissa.

Suurinta osaa testatuista ohjelman osista testattiin tulostamalla metodien tuloksia eri lähtöarvoilla ja toteamalla olivatko tulokset oikein vai ei. Kuitenkin esimerkiksi siirron läpäisemiä koordinaatteja laskevalle metodille toteutettiin yksikkötestaus. Testeissä pyrittiin käymään läpi kaikki oleelliset eri tapaukset kuten pysty- ja vaakasuora siirto, 45 asteen kulman siirto, mielivaltaisten kulmien siirtoja sekä negatiiviseen y- tai x-suuntaan kulkevia siirtoja. Ohjelma selvisi näistäkin testeistä.

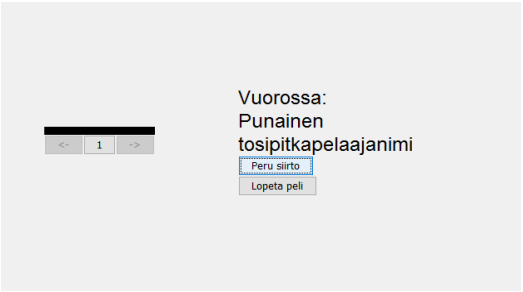
Ohjelman tunnetut puutteet ja viat

Peli-ikkunan graafiset ongelmat

Ensimmäinen pelin graafiseen esitykseen liittyvistä ongelmista näyttää esiintyvän nimenomaan pienikokoisilla kentillä kuten 12x12. Vieressä oleva kuva esittää tilannetta, jossa peli on pelattu ensimmäiset pari siirtoa. Kuvassa voi havaita todellisen kentän oikean yläkulman vieressä lyhyen viivan ja alakulman vieressä jotain mikä muistuttaa kentän vasenta ylänurkkaa. Itse lauta näkyy kokonaan ja sillä pystyy pelaamaan peliä, mutta omituiset jäljet häiritsevät näkymää hieman. Syytä tälle vialle ei ole tiedossa.



Toinen ongelma taas liittyy käyttäjien nimimerkkeihin. Jos käyttäjä valitsee itselleen nimimerkin, jonka pituus on pidempi kuin ”Vuorossa:”-teksti käyttäjänimen yläpuolella, alkaa pelilauta vaihtaa sijaintiaan ruudulla aina vuoron vaihtuessa. Tämä ilmeisesti johtuu siitä, että kenttä ja tekstit pyrkivät automaattisesti sijoittumaan peli-ikkunaan mahdollisimman tasaisesti. Pelaajanimen ollessa vielä pidempi voidaan joutua tilanteeseen, jossa pelilauta ei mahdu kokonaisuudessaan peli-ikkunaan ja katoaa näkyvistä. Kuten viereisestä kuvasta näkee, peli tulee tällöin pelaamiskelvottomaksi. Tämän ongelman voisi kuitenkin yksinkertaisesti korjata asettamalla sopivan pituusrajoituksen pelaajanimiin.



Etusivun valikon muokkaamattomuus

Toinen käyttöliittymää koskeva ongelma on etusivun dropdown-valikkojen muokkaamattomuus. Tämä aiheuttaa sen, että jos pelin aikana luodaan uusia Profiileja tai uusia Ratoja eivät ne vielä samalla pelikerralla näy vaihtoehtoina uutta peliä aloittaessa, vaan peli täytyy sammuttaa ja käynnistää uudelleen ensin. Ongelma perustuu siihen, että kyseinen valikko on määritelty val-muuttujaksi, jotta uutta peliä avatessa pystytään lukemaan radio button -valinnat valikon listoista. Vastaavaa ongelmaa esiintyy pienemmässä mittakaavassa myös muiden sivujen valikoilla. Muut valikot kuitenkin päivittävät itsensä pelkästään sivua vaihtamalla.

Alun perin etusivun valikko oli toteutettu funktiona, joka palauttaa valikosta aina uusimman päivitetyn version hakemalla tiedot Pelaajista ja Radoista aina uudestaan, kun funktiota kutsutaan. Tämä kuitenkin nopeasti osoittautui mahdottomaksi toteutustavaksi, sillä näin tietoon valikon valinnoista ei päässyt mitenkään käsiksi. Kun funktiota kutsuttiin uudestaan, saatiin luonnollisesti vain uusi versio valikosta. Vaikka parhaillaan käytössä oleva valikko olikin tallessa Ikkuna-olion menuBar-muuttujassa, ei keinoa sen sisällön valintojen tarkasteluun löydetty. Tästä johtuen uutta peliä ei pystytty luomaan valintojen pohjalta, ja oli pakko tyytyä muuttumattomaan valikkoon.

Projektin aikana tätä ongelmaa yritettiin korjata useampaan kertaan, mutta ei koskaan löydetty toimivaa toteutustapaa. Nyt jälkikäteen ajatellen ongelma ei vaikutakaan enää yhtään niin vaikealta. Ratkaisu ongelmaan voisi olla käyttää var-tyyppistä muuttujaa pelaaja1- pelaaja2 ja tasovalinta-muuttujien säilömiseen ja luoda erikseen funktio, jolla voitaisiin päivittää näitä arvoja. Kun funktiota kutsuttaisiin aina päävalikon auetessa, saataisiin dropdown-valikot päivittymään aina siinä kohtaa. Valikko pysyisi kuitenkin muissa väleissä samana ja pelin pystyisi avaamaan paaValikko-menun sisältämien tietojen avulla. Alla nykyinen toteutus oliosta NappuloidenHallinta:

```
val paaValikko = new MenuBar {
    val pelaaja1 = new PelaajaMenu("Pelaaja1: Sininen")
    val pelaaja2 = new PelaajaMenu("Pelaaja2: Punainen")
    val tasovalinta = new RataMenu("Rata")
    contents += pelaaja1
    contents += pelaaja2
    contents += tasovalinta
  }
```

Tekoäly

Tekoälyä, kuten aiemminkin on sanottu, ei saatu lopultakaan toimimaan oikein. Se osaa kyllä liikkua pelilaudalla hetken mietinnän jälkeen, eikä se testipeluutuksissa ole kertaakaan törmännyt seinään tai toiseen pelaajaan. Kuitenkinkin koska tekoäly osaa laskea vain viisi vuoroa eteenpäin, se ei osaa varsinaisesti suunnistaa kohti maalia, ennen kuin se viidellä siirrolla pääsisi sinne. Sen sijaan tekoäly näyttää seuraavaan kentän muotoja ja tekevän siirtovalintansa nimenomaan vain välttääkseen ulosajoa. Tästä voi seurata se, että leveärataisilla tai haarautuvilla kentillä tekoäly vaihtaa suuntaa kesken matkan, eikä koskaan pääsee tarpeeksi lähelle maalia, jotta osaisi navigoida itsensä sinne. Aiemmin mainitun lisäksi tekoäly ei myöskään tuntemattomasta syystä koskaan vaihda vaihdettaan vaihteen kaksi yläpuolelle, vaikka se selvästi niin pystyisi tekemään ilman ulosajamisen riskiä. Ja kaiken muun lisäksi tekoälyn ollessa toisena pelaajana ei normaali maalinylityksen testaus toimi, sillä tekoäly siirron jälkeen voittoa ei tarkisteta.

Osan näistä ongelmista korjaaminen olisi helppoa, jos projektia vielä jatkettaisiin. Esimerkiksi pelin voittamisen ongelma saataisiin ratkaistua yksinkertaisesti lisäämällä uusi voittotarkistus tekoälyn siirron jälkeen, sillä tällä hetkellä voitto tarkistetaan vain jokaisen pelaajan siirron jälkeen. Koko tekoälyn korjaamiseksi kuitenkaan ei riittäisi välttämättä edes vanhan koodin muokkaaminen, vaan luultavasti AI-luokan koko siirto-metodi pitäisi luoda uudelleen erilaista algoritmia käyttäen. Yksi tällainen vaihtoehto on selitetty Algoritmit-luvussa.

Kulmittaisten reunaruutujen välistä kulkeminen

Tällä tarkoitetaan tilannetta, jossa koordinaatissa (0,0) oleva auto voi liikkua koordinaattiin (1,1) huolimatta siitä, että ruudut (1,0) ja (0,1) ovat kumpikin radan reunaa. Visuaalisesti tämä saattaa näyttää siltä, että auto ajaisi reunan yli. Siirron läpäisevien ruutujen määritelmäksi kuitenkin alun perin valittiin, että ”siirto kulkee kaikkien niiden koordinaattien läpi, joiden ruutuja leikkaa lähtökoordinaatin keskipisteestä lähtevä ja kohdekoordinaatin keskipisteeseen päättyvä suora”. Tämän määritelmän mukaan ajaessa koordinaatista (0,0) koordinaattiin (1,1) ei viereisissä koordinaateissa käydä ollenkaan matkalla. Otin tämän esille ”ongelmana”, sillä se saattaa näyttää omituiselta pelattaessa. Koska kuitenkin mielestäni käytetty määritelmä on edelleen ainut järkevä, ei tämä minusta ole vika, vaan suunniteltu ominaisuus.

3 parasta ja 3 heikointa kohtaa

Heikkoudet

Samanlaiset luokat ja funktiot

Ensimmäisenä heikkoutena otan ohjelmastani esille toisteisuuden eräiden luokkien ja eräiden funktioiden välillä. RataMenu, PelaajaMenu ja MaastoMenu ovat keskenään hyvin samanlaisia. Samoin Ruudukkossa ja Rataeditorissa on paljon samankaltaisuutta. Tämä johtuu eniten kenties siitä, ettei näille ohjelman osille ei ollut etukäteen tehty yhtä selkeää suunnitelmaa kuin monelle muulle luokalla ja niiden välisille yhteyksillä. Myös NappuloidenHallinta-olion Dialog-kyselyt ovat kopioitu toisistaan muuttamalla vain tiettyjä yksityiskohtia.

RataMenu, PelaajaMenu ja MaastoMenu ovat dropdown-valikoita, joiden tarkoitus on hakea lista kaikista Radoista, Profiileista sekä Maastoista ja täyttää itsensä niiden nimillä. Näitä valikoita käytetään esimerkiksi pelin päävalikossa valitessa käytettäviä profiileita ja pelattavaa kenttää. Vaikka valikot hakevatkin eri tietoja, on niiden toiminta hyvin samanlaista. Jokainen luokka tullessaan luoduksi hakee tietyn listan ja yhdistää listan alkioiden nimen perusteella luodun Radio Buttonin itse lista alkioon Map-tietorakenteessa. Sen jälkeen luodaan ButtonGroup-luokan ilmentymä, johon Radio Buttonit lisätään. Lisäksi jokaisessa luokassa valitaan joku nappuloista oletuksena ja määritellään metodi palauttamaan sillä hetkellä valittua Radio Buttonia vastaavan ilmentymän Radasta Pelaajasta tai Maastosta. Koska luokissa on niin paljon samanlaisuutta voitaisiin osa niiden toiminnallisuudesta yhdistää yhteen yläluokkaan, jolloin ohjelmasta saataisiin hieman siistimpi.

Ruudukon ja Rataeditorin välillä yhteys on vielä selkeämpi. Metodit piirraRuudukko ja piirraMaasto ovat nimittäin täsmälleen samanlaisia kummassakin luokassa. Lisäksi sekä Ruudukossa että Rataeditorissa klikatun ruudun laskeminen suoritetaan samalla tavalla ja ainoa ero on siinä, mitä tiedolle tehdään. Koska Lähes kaikki Rataeditorin ominaisuuden ovat samoja kuin Ruudukon, mutta Ruudukolla ominaisuuksia on vain enemmän olisi loogista määritellä Rataeditori Ruudukon aliluokaksi. Näin

toistuvia metodeja ei tarvitsisi kirjoittaa moneen kertaan, vaan riittäisi, että paintComponent-metodi ja funktio, joka klikatulle koordinaatille suoritetaan olisi määritelty uudestaan aliluokassa.

Ikkunan ja NappuloidenHallinnan sekavuus

Samoin kuin edellinenkin ongelmakohta, tämäkin liittyy ohjelman käyttöliittymään ja olisi kenties voitu välttää paremmalla etukäteisellä suunnittelulla. Tämä toinen heikkous on eri käyttöliittymän komponenttien jakautuminen epämääräisesti eri tiedostoihin. Sen lisäksi, että komponentteja löytyy yksittäisoliosta Ikkuna, joka olisi looginen paikka kaikille komponenteille, on osa niistä sijoitettuna olioon NappuloidenHallinta.

NappuloidenHallinnan idea oli alun perin pitää käyttöliittymän graafista puolta kuvaavat luokat ja oliot mahdollisimman siisteinä. Sen sijaan että oliolle Ikkuna olisi määritelty pitkiä funktioita, ne sijoitettiin olioon NappuloidenHallinta, joka nimestään huolimatta loppujen lopuksi sisälsi kaiken käyttöliittymän ja pelin sisäisen logiikan välisen kommunikaation. Näin Ikkunasta pystyi tiettyä nappulaa painaessa kutsumaan efektejä, joita kuvaavat funktiot eivät kuitenkaan ollut vähentämässä Ikkuna-olion selkeyttä.

Tätä perusperiaatteita pidän itse edelleen järkevinä. Valitettavasti se ei kuitenkaan toteutunut projektissa niin kuin oli tarkoitettu. Ikkunaan luotiin useita metodeja eri näkymien vaihtamiselle ja NappuloidenHallintaan siirrettiin kaikki dropdown-valikot. Näille kaikille päätöksille oli alun perin syy, joka liittyi valikkojen päivittämiseen sekä rekursiivisten määritelmien välttämiseen, ja aikanaan tämä toteutustapa valittiin, sillä se oli ainut, joka silloin saatiin toimimaan. Jos projektia olisi kuitenkin jatkettu vielä, olisi näiden kahden luokan siistiminen ollut yksi seikka jota olisin ehdottomasti lähtenyt toteuttamaan.

Uskon, että on olemassa ratkaisu, jolla luokille saisi toteutettua tavoittelemani järkevän jaon. Ikkuna-oliossa tulisi olla määritelty kaikki graafiset komponentit, kuten nappulat ja valikot, kun taas NappuloidenHallinnassa olisi ollut metodeja, joilla viestittäisiin käyttöliittymän ja pelilogiikan välillä. Erikseen voisi vielä luoda olion, jossa olisi määriteltynä funktiot, joilla käyttöliittymän komponentteihin vaikutettaan, esimerkiksi tekstejä muuttamalla. Näin olio NappuloidenHallinta ei olisi niin sekava kuin se on nyt. Lisäksi NappuloidenHallinta-olion nimen voisi vaihtaa sen nykyistä tarkoitusta paremmin kuvaavaan.

Tekoäly

Kuten aiemminkin on tullut ilmi, tekoälyä ei saatu toimimaan tehtävänannon vaatimalla tavalla. Tämä on erityisen harmillista sen takia, että tekoäly oli omasta mielestäni yksi ehdottomasti kiinnostavimmista projektin osatehtävistä. Erityisesti tästä johtuen halusin ottaa aiheen vielä esille tällä listalle. Se on nimittäin omasta mielestäni kaikkein ikävin puute projektissa. Jos projektia jatkettaisiin, pyrkisin ehdottomasti saamaan tämän ominaisuuden toimimaan oikein.

Vahvuudet

Peru-nappula

Yksi projektin hienoista ominaisuuksista on peru-nappula. Vaikka se ei teknisesti olekaan monimutkainen, se pääsi listalle, sillä se oli ylimääräinen ominaisuus. Peru-nappula on käytännöllinen tilanteissa, jossa peliä pelatessa vahingossa klikkaa väärää ruutua tai kun huolimattoman siirron takia

ajaa yli pelin ja häviää. Tällaisissa tilanteissa jos peli haluttaisiinkin pelata loppuun sen sijaan että peli jouduttaisiin aloittamaan alusta.

Perumisen toimintaperiaate on hyvin yksinkertainen. Kumpikin auto pitää kirjaa tekemistään siirroista ja kun siirto halutaan perua, tarkistetaan peruttavan olevan auton viimeisin siirto ja siirretään auto takaisin siirron lähtökoordinaattiin. Ruutu, jossa auto viimeksi oli on varmastikin tyhjä, jos peruminen kohdistuu aina siihen autoon, joka edellisenä teki siirtonsa. Tämän takia siirtoa peruessa voidaan auto pakottaa edelliseen ruutuunsa ilman siirron laillisuuden tarkastamista. Siirron laillisuuden tarkastamatta jättäminen on oleellista, sillä auto ei missään tilanteessa voisi laillisesti siirtyä ruutuun, jossa oli viime vuorollansa. Jos auto siirrollaan ylitti maalin, tulee kierroslaskurinkin lisäys peruttua. Tämän jälkeen viimeisin siirto poistetaan auton listasta. Lopuksi auton vaihde palautetaan siihen vaihteeseen, jolla nyt auton siirtolistassa uusimpana oleva siirto on tehty.

Maalit

Erilaisten maastotyyppien osana peliin toteutettiin neljä erilaista maalityyppiä. Maalityyppien tarkoitus oli mahdollistaa erilaisten ratojen luominen. Alkuperäisessä versiossa maalityyppejä oli vain yksi ja sen takia maaliviivan piti aina olla pystysuora ja rataa piti aina kiertää vastapäivään. Uusilla maalityypeillä tehtiin mahdolliseksi se, että maaliviivan pystyi luomaan myös vaakasuoraan ja sen lisäksi sekä vaaka- että pystysuorissa maaliviivoissa pystyi säätämään kumpaan suuntaan maalin läpi kuljetaan. Tämä määrää sekä sen mihin suuntaan autot ovat menossa lähtiessään liikkeelle että sen mihin suuntaan maaliviiva pitää ylittää, jotta kierros lasketaan. Maaliviivan väärään suuntaan ylittämisetkin pystytään hallitsemaan, sillä väärään suuntaan maalin ohittaessa, kierroslaskuri nousemisen sijasta laskee, minkä jälkeen maali pitää ensin ylittää takaisin ja sen jälkeen vielä oikeasti kiertää koko kierros. Näin estettiin huijaaminen, jossa maali ylitetään ensin väärän suuntaan, jotta oikeaan suuntaan palatessa saisi kierroksen laskettua.

Kehitettävyys

Pelin kehityksen aikana pyrittiin mahdollisimman suuri osa valinnoista tekemään niin, että peliä pystyisi myöhemmin helposti kehittämään. Kaikilta osin tämä ei ole onnistunut niin hyvin kuin olisi haluttu, mutta muutamia hyviä esimerkkejä asiasta voidaan antaa. Ensinnäkin uusia maastotyyppejä voi suhteellisen pienellä vaivalla lisätä peliin. Toiseksi pelaajien määrä on pyritty pitämään yleisellä tasolla koodissa sen sijaan että pelaajien määrä oletettaisiin kahdeksi. Tulevaisuudessa olisi siis muutamilla muutoksilla mahdollista saada pelistä aikaan variaatio, jossa kilpailevia autoja onkin esimerkiksi kolme. Kolmanneksi vaikka voittoon tarvittavien kierrosten määrä on tällä hetkellä kaikissa radoissa yksi, viitataan tähänkin lukuun Pelitilanteessa aina muuttujana. Näin olisi myöhemmin mahdollista kehittää variaatioita, jossa tiettyä rataa tulee ajaa useampi kierros voittaakseen. Viimeisenä esimerkkinä sekä Radan että Profiilin tiedoston perusteella luominen on case-rakenteen ansioista varsin joustavaa, mikä mahdollistaa niin halutessa uusien tietojen tallentamisen Ratoihin ja Profiileihin.

Poikkeamat suunnitelmasta

Projektia tehdessä pyrittiin kaikki toteuttamaan mahdollisimman suunnitelmaa vastaavasti ja tämä näkyikin projektin lopputuloksessa. Suurin osa asioista toteutettiin täsmälleen siten kuten suunnitelmassa ja vielä samassa järjestyksessä. Ensimmäiset pari viikkoa suunnitelmaa noudatettiin sekä aikataulun että järjestyksen kanssa täsmällisesti. Tämän jälkeen kuitenkin muiden kurssien työmäärät alkoivat vähentää aikaa, jota projektiin käytettiin, mistä johtuen projektin työvaiheet alkoivat jäädä

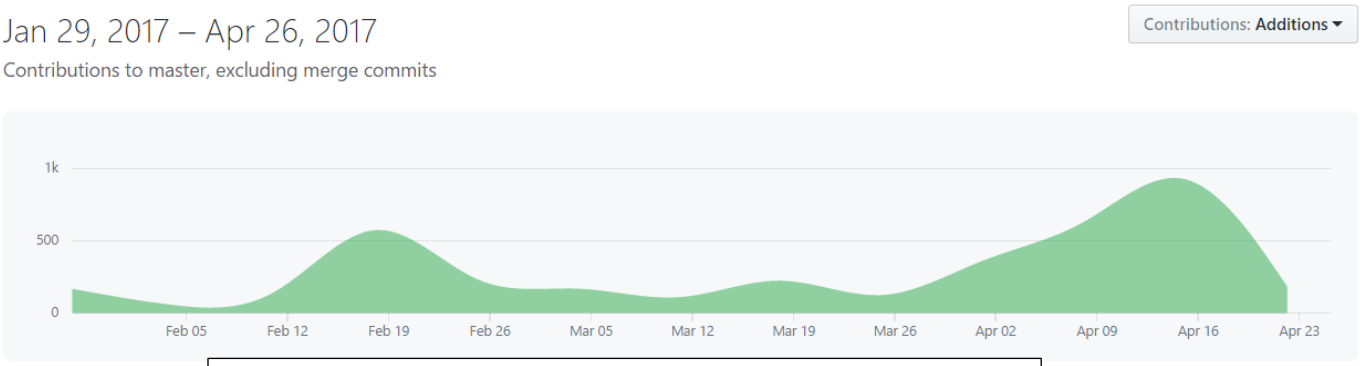
jälkeen suunnitellusta aikataulusta. Onneksi aikataulut oli kuitenkin laadittu hyvin optimistisesti ja sen tähden ylimääräistä aikaa oli sopivasti projektin valmistumiseen. Loppujen lopuksi työ tuli valmiiksi lähes sille tasolle kuin oltiin suunniteltukin.

Niiltä osin, kun luokkia oli suunniteltu etukäteen luokkajako ja luokkien väliset suhteet olivat täysin suunnitelman mukaisia. Uusia luokkia ja olioita kuitenkin tuli lisää projektin aikana erityisesti graafiselle puolelle, jota ei etukäteen oltu suunniteltu kovinkaan tarkasti. Projektin aikana päätettiin toteuttaa myös uudet luokat Koordinaatti, Siirto ja Suunta, joita alkuperäisessä suunnitelmassa ei ollut, mutta jotka lopulta osoittautuivat hyvin keskeisiksi osiksi projektin toteutusta. Uusien luokkien lisäksi vanhat luokat saivat monia uusia metodeja, joita ei oltu osattu suunnitella etukäteen. Lähes kaikki suunnitellut metodit kuitenkin toteutettiin suunnitelman mukaisesti.

Joitain yksittäisiä muutoksia alkuperäiseen suunnitelmaan nähden syntyi toteutuksen aikana. Esimerkiksi Rata-luokka oli alun perin tarkoitus määritellä Pelilaudan perustella, johon sitten pelin alkaessa olisi sijoitettu Autot. Tiedosta lataamisen kannalta kuitenkin huomattavasti järkevämmäksi tavaksi ilmeni, että Rata-luokka luotiin Array[Array[Maasto]] tiedon pohjalta. Radalle luotiin kumppanuusolio, joka taas apply-metodissa muutti riveittäin Vector[String]-muodossa annetun tiedoston Rata-luokan vaatimaan muotoon. Myös toisin kuin alkuperäisessä suunnitelmassa Radan muodosta tehtiin rataeditorin toimintaa varten muuttuvatilainen.

Toinen tällainen pieni muutos oli tekoälyn vuorontarkistuksen siirtäminen Ruudukko-luokkaan Pelitilanne-luokasta. Alkuperäisessä ideassa Pelitilanne olisi jokaisen vuoronvaihdon yhteydessä tarkistanut, onko seuraavaa pelaaja AI ja siirtänyt sen puolesta tarvittaessa. Vastuu tästä siirrettiin kuitenkin Ruudukkoon, koska ensimmäinen tapa aiheutti ongelmia tekoälyn suunnitellessa siirtojaan kuvitteellisella pelillä.

Toteutunut työjärjestys ja aikataulu



Projektiohjelman päivittäinen edistyminen koodiriveissä mitattuna

Projekti aloitettiin Ikkuna-luokalla ja swingillä piirtämisen kertailulla sekä luomalla muutamia keskeisempiä muita luokkia ja oliota, kuten Peli, Pelitilanne ja Pelilauta. Tätä vaihetta työstettiin jo ennen kuin edes suunnitelma projektista oli valmis. Sen jälkeen käytettiin muutama päivä yleisen ja teknisen suunnitelman kirjoittamiseen, kunnes projektin varsinainen toteutus aloitettiin suunnitelmien palauttamisen jälkeen 21. helmikuuta. Silloin työskenneltiin maastojen ja seuraavana päivänä myös

autojen piirtämisen kanssa. 22. helmikuuta luotiin kehykset muutamille uusille luokille kuten AI, Rata ja Pelaaja. Samaan aikaan aloitettiin Autoa siirtävän funktion sekä peli-ikkunan klikkauksen tunnistamisen kehittäminen.

23. helmikuuta klikkauksen tunnistaminen ja auton liikuttaminen yhdistettiin, jolloin autoja pystyi liikuttamaan pelilaudalla vuorotellen. Tässä vaiheessa tosin auton liikkumisella ei ollut vielä mitään rajoituksia. Kahden päivän päästä liikkuminen rajoitettiin vain vapaisiin tieruutuihin ja lisättiin ominaisuus vaihteiden vaihtamiselle, vaikkei se vielä liikkumiseen vaikuttanutkaan. Tämän prosessin rinnalla työskenneltiin myös ratojen lataamisen parissa ja 25. päivä helmikuuta oli ratojen tiedostosta lataaminen valmis. Ruudulle piirrettävää rataa ei kuitenkaan pystynyt vielä vaihtamaan.

Pienen tauon jälkeen seuraavan viikon loppupuolella luotiin luokat Siirto ja Suunta, joita myöhemmin tultaisiin käyttämään sallittujen siirtojen laskemisessa sekä liikkumisen uudessa toteutuksessa. Tätä seuraavan viikon maanantaina 6. maaliskuuta saatiin siirrot lopulta toimimaan oikein. Lailliset siirtovaihtoehdot näytettiin peliruudulla ja vain näihin ruutuihin pystyi siirtyä. Myös vaihteen vaihtamisen vaikutus siirtoon toimi oikein.

Seuraavat muutama päivä olivat muiden kurssien työmäärän takia projektin edistymisen kannalta katkonaisia. Projektin edistyi merkittävästi ensi kertaa tämän jälkeen vasta noin kahden viikon päästä 20. maaliskuuta, jolloin peliin luotiin päävalikko. Tämän lisäksi itse peli-ikkunaan lisättiin laudan viereen nappulat siirron perumiselle ja pelin keskeyttämiselle, joiden molempien toiminnallisuus myös saatiin toteutettua.

Heti päävalikon luomisen jälkeen aloitettiin aktiivisesti työskentelemään algoritmin parissa, joka laskisi tietyn siirron läpikulkemat koordinaatit. Tämän toteuttamiseen muiden kiireiden ohella meni projektin ajasta seuraavat kaksi viikkoa, kunnes 6. huhtikuuta funktio valmistui. Uutta ominaisuutta hyödyntäen ohjelmoitiin seuraavaksi peli tunnistamaan, jos jompikumpi pelaajista ylittää maaliviivan.

Sen jälkeen projektissa keskityttiin Profiileihin. Ajanjaksolla 7.-12. huhtikuuta toteutettiin Profiilien luominen tiedoston pohjalta sekä mahdollisuus Profiilin päivittämiseen. Lopulta toteutettiin käyttöliittymään sivu, jossa profiilien tietoja voi tarkastella ja uuden profiilin luoda. Käyttöliittymään lisättiin myös mahdollisuus valita pelissä käytettävät Profiilit sekä Rata.

Tässä vaiheessa projektia työskentelyn tahti alkoi kiihtyä deadlinen lähestyessä ja muiden velvoitteiden vähentyessä. Seuraavat 10 päivää työskenneltiinkin lähes täyspäiväisesti projektin parissa. Ensimmäisinä päivinä lisättiin pelin päättyminen voiton lisäksi tilanteeseen, jossa toinen pelaaja ei voinut liikkua ja lisättiin popup-huomautukset esimerkiksi pelin loppumiselle. 16. päivänä muutettiin autot aloittamaan satunnaisista lähtöruuduista määrättyjen lähtöruutujen sijaan. Samalla muokattiin maaliviivaa kuvaavaa Maasto-luokkaa niin, että se mahdollisti radan kiertosuunnan määrittämisen. Näiden lisäksi aikaa kului pelistä löytyneiden virheiden korjaamiseen.

Vasta viimeisen kokonaisen viikon tiistaina (18. huhtikuuta) aloitettiin vaikean tason vaatimusten toteuttaminen AI-luokalla. Keskiviikkona tekoäly saatiin liikkumaan ilman törmäilyä, mutta myöskään vaihteita vaihtamatta. Kun tätä yritettiin korjata, näytti tekoäly menevän yhä enemmän sekaisin. Ensimmäisessä versiossa se kaatoi ohjelman aina sen vuoron koittaessa ja kun tämä saatiin korjattua, se otti strategiakseen ajaa lähintä seinää päin heti pelin alettua.

20. päivä tekoälyn parista otettiin tauko ja lähdettiin toteuttamaan uusia maastotyyppejä. Päivän päätyttyä kaikki neljä pelin erikoismaastoa olivat toiminnassa niin Radan lataamisen, piirtämiseen kuin siirtovaihtoehtojen rajoittamisen osalta. Seuraavien kahden päivän aikana toteutettiin rataeditori valmiiseen toimintakuntoonsa. Uusia ratoja pystyi luomaan aloittaen tyhjästä tai vanhan radan pohjalta. Myös vanhoja ratoja pystyi muokkaamaan.

Viimeisenä itse projektiohjelman tekopäivänä 23. huhtikuuta yritettiin AI-luokkaa vielä korjata, sillä se oli ainut osa ohjelmasta joka ei vielä täyttänyt tehtävänantoa. Yrityksestä huolimatta tuloksena oli vain lievästi edellistä versiota parempi tekoäly. Viimeiset kolme päivää ennen projektin deadlinea 26. huhtikuuta käytettiin projektidokumentin kirjoittamiseen.

Arvio lopputuloksesta

Työ toteuttaa mielestäni täsmällisesti kaikki ohjelmalle annetut perusvaatimukset ja tarjoaa joitain ylimääräisiä ominaisuuksiakin. Helpon tason vaatimukset ratojen ja käyttäjien tietojen lataamisesta ja tallentamisesta ovat toteutettu. Näitä kumpiakin tietoja pystyy lisäksi tarkastelemaan ohjelmassa Profiilit ja Radat -sivulla. Profiileita ja ratoja pystyy kumpiakin myös luomaan uusia.

Keskitason vaatimuksena ollut graafinen käyttöliittymä on toteutettu ja toimii lähes täydellisesti. Käyttöliittymä on yksinkertainen, eikä sisällä erityisen laadukasta grafiikkaa, mutta mielestäni täyttää tehtävänannon vaatimukset tältä osin. Vaikean tason vaatimuksista toteutettuina on rataeditori ja erilaiset maastotyypit, jotka nekin mielestäni yltävät vaaditulle tason. Tietokonepelaaja sen sijaan ei täytä tehtävänannossa määritettyjä vähimmäisvaatimuksia, sillä vaikka se osaakin välttää häviämisen ulosajamisen takia, ei ole taattua, että se jokaisella radalla pääsisi maaliin asti, vaikka ihmispelaaja pelaisikin huonosti.

Ohjelman luokkajako ja tietorakenteiden käyttö on mielestäni pääosin sujuvaa, vaikka muutamassa aiemmin mainitussa luokassa tiettyjä korjauksia voitaisiinkin tehdä. Tällaisia ovat esimerkiksi luokkien RataMenu, PelaajaMenu ja MaastoMenu yhdistäminen yhteiseen yliluokkaan sekä Rataeditori-luokan toteuttaminen Ruudukon aliluokaksi. Lisäksi yksittäisolioiden Ikkuna ja NappuloidenHallinta sisältöjä voisi siistiä niin funktiota yhdistämällä toisteisuuden välttämiseksi kuin funktioiden sijoittelua näiden kahden olion välillä vaihtamalla. Myös osa tiedoista olisi ohjelmassa voinut olla tallennettuna parempaan paikkaan. Esimerkiksi tiedot Maaston väristä olisi voinut Ruudukko-luokan sijaan sijoittaa samaan paikkaan, jossa muut Maaston tiedot ovat määritelty.

Kuten edellisessä luvussakin todettiin on ohjelman laajentaminen ja kehittäminen ainakin tietyiltä osin helppoa. Tällaisiin asioihin kuuluvat uusien vastaavien maastotyyppien lisääminen, pelaajamäärän tai voittoon tarvittavan kierrosmäärän vaihtaminen sekä tallennettavien tietojen lisääminen Profiileihin ja Ratoihin. Sen sijaan esimerkiksi siirtovaihtoehtoja lisäävien maastotyyppien luominen ei olisi yhtä suoraviivaista, mutta onnistuisi luultavasti sekin.

Jos projektia lähdettäisiin jatkamaan korjattaisiin ensimmäisenä puutteet ja viat, joita edellä ja aiemmin dokumentissa on lueteltu. Tämän jälkeen voitaisiin keskittyä kehittämään uusia ominaisuuksia. Tällaisia voisivat olla yhden ja yli kahden pelaajan pelivaihtoehdot, eri tasoiset tietokonevastustajat sekä kierrosmäärän lisääminen joko jokaisen radan ominaisuudeksi tai pelin alussa valittavaksi vaihtoehdoksi. Myös pelin graafista puolta voisi kehittää. Esimerkiksi autoja voitaisiin ympyröiden sijaan kuvata autokuvilla ja autot voisivat siirrettäessä todella liikkua lautaa pitkin. Lisäksi pelin ohjeiden näyttämistä

voitaisiin päivittää niin, että kesken pelin klikatessaan tiettyä ruutua, peli osaisi kertoa, mikä maasto on kyseessä ja miten se vaikuttaa liikkumiseen. Voitaisiin myös lisätä mahdollisuus nähdä oman vuoronsa aikana mihin suuntaan vastustajan auto on kulkemassa.

Projekti onnistui mielestäni loppujen lopuksi oikein hyvin, vaikka tiettyjen ongelmien ratkaisemattomuus jäikin vähän häiritsemään. Se toteuttaa tekoälyä lukuun ottamatta kaikki annetut vaatimukset ja omasta mielestäni ansaitsee arvosanaksi vähintään 4.

Viitteet

Lähes kaikki projektissa käytetyt lähdemateriaalit olivat internetsivuja. Lisäksi esimerkiksi käyttöliittymää tehdessä tukena käytettiin viime syksynä kurssilla Ohjelmointi 1 tekemääni ryhmätyöprojektia Pentaminoes. Nettilähteistä suurin osa löydettiin etsimällä Googlesta ratkaisua johonkin tiettyyn ongelmaan, mutta pieni osa lähteistä oli myös tämän tai aiempien ohjelmointikurssien materiaaleista. Tällaiset kurssimateriaalit merkittiin lähteeksi silloin, kun niitä nimenomaisesti tätä projektia varten palattiin katsomaan.

Alla on listattu käytetyt lähteet luokkakohtaisesti. On mahdollista, että jonkin luokan lähteen tietoja on käytetty myös toisen luokan yhteydessä ilman merkintää , jos tieto alun perin etsittiin vain toista luokkaa varten.

gui.Ikkuna

- Pentaminoes-projektin (Ryhmämme työ Ohjelmointi 1 kurssilla) GameWindow-olio
- Ohjelmointi 1 kurssin materiaalin Swing-esimerkkipeli: sekä pdf että kooditiedostoista View-olio
- BoxPanel(Orientation.Vertical) -luokan käyttö: http://stackoverflow.com/questions/13380701/scala-swing-component-alignment
- Case-match rakenteeseen liittyvä ongelma: http://alvinalexander.com/scala/scala-unreachable-code-due-to-variable-pattern-message
- http://alvinalexander.com/java/jwarehouse/scala/src/swing/scala/swing/test/UIDemo.scala.shtml

gui.NappuloidenHallinta

- Dialog input: http://stackoverflow.com/questions/19603403/how-to-collect-input-from-user-with-dialog-box

gui.MaastoMenu, gui.PelaajaMenu, gui.RataMenu

- Radio button Menu: http://alvinalexander.com/java/jwarehouse/scala/src/swing/scala/swing/test/UIDemo.scala.shtml
- Radio button Menu: https://lampsvn.epfl.ch/trac/scala/browser/scala/trunk/src/swing/scala/swing/test/Dialogs.scala?rev=25699#L54

gui.Ruudukko, gui.Rataeditori

- Samoja lähteitä kuin Ikkunassa
- http://www.scala-lang.org/api/rc2/scala/swing/Dialog$.html
- Värien lukuarvot: http://colorizer.org/

- Java.awt.Color: http://www.users.csbsju.edu/~lziegler/apidocs/java.awt.Color.html

peli.AI

- Listojen min-funktion Ordering: http://www.scala-lang.org/old/node/7529

pelikomponentit.Maasto

- Ohjelmointi 1 -kurssin viikkotehtävän Chess luokka Piece

siirrot.Siirto

- Läpimentävien koordinaattien laskenta pohjautui: https://fi.wikipedia.org/wiki/Bresenhamin_algoritmi

tietojenTallennus.Profiili

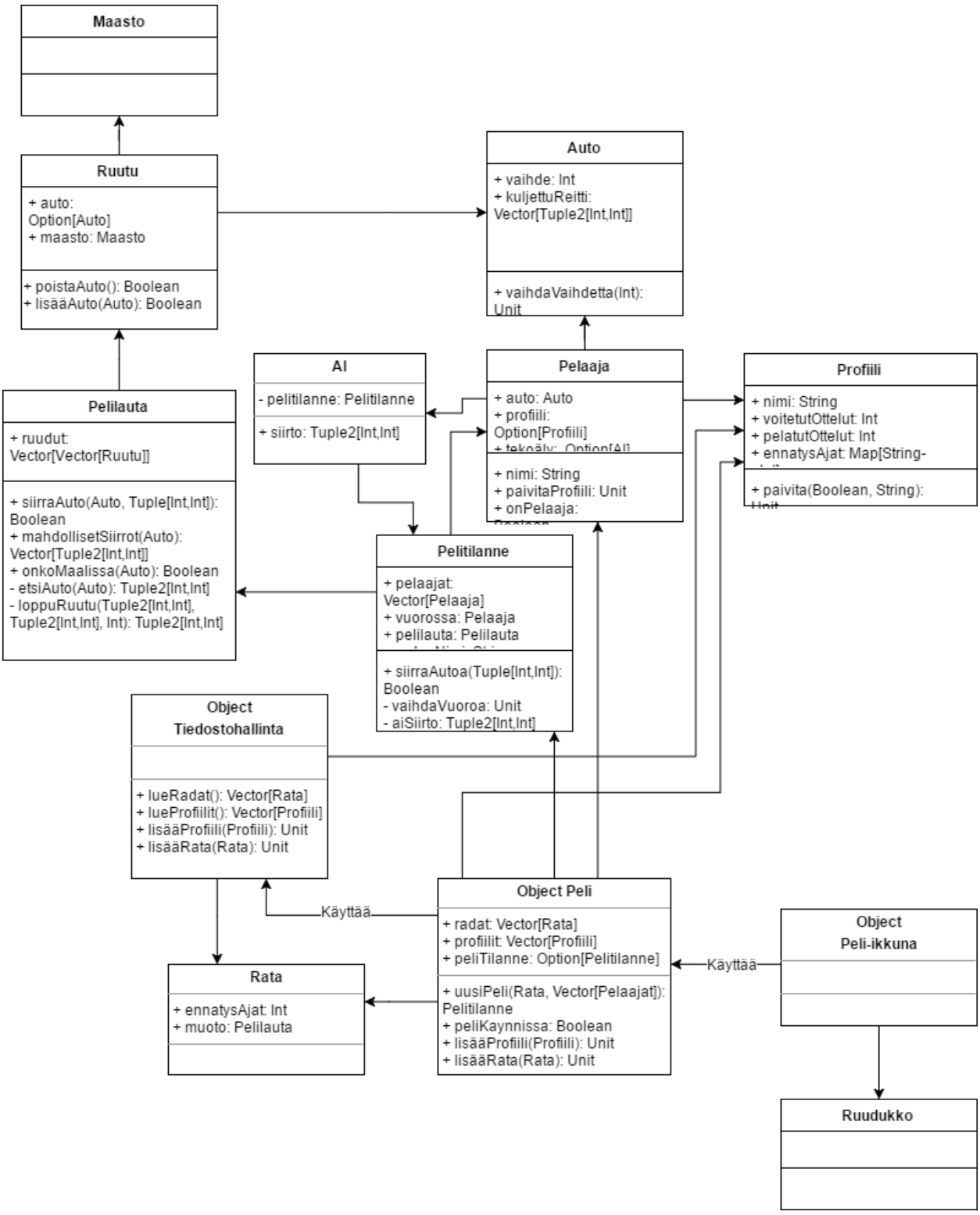
- Map: http://docs.scala-lang.org/overviews/collections/maps.html
- Regex matching: http://stackoverflow.com/questions/15119238/scala-regular-expressions-string-delimited-by-double-quotes
- Regex matching: https://www.tutorialspoint.com/scala/scala_regular_expressions.htm

tietojenTallennus.TiedostonHallinta

- Tiedostojen lukeminen ja hallinta, sekä listFiles()-komento: https://plus.cs.hut.fi/studio_2/2017/k15/osa03/
- Vastaus ongelmaan map-komennon kanssa: http://www.scala-lang.org/old/node/12090.html
- Tiedostojen kirjoittaminen: http://alvinalexander.com/scala/how-to-write-text-files-in-scala-printwriter-filewriter

Liitteet

Seuraavalla sivulla on esitetty projektin suunnitelmassakin liitteenä ollut luokkakaavio. Luokkakaavio on suhteessa valmiiseen projektiin hieman puutteellinen, sillä sinne ei ole merkitty kaikkia graafisen käyttöliittymän luokkia eikä luokkia Koordinaatti, Siirto ja Suunta. Myöskään kaikkien siinä olevien metodien palautustyypit eivät ole sama kuin lopullisessa projektissa. Kuvaa voi kuitenkin käyttää luokkien ja olioiden suhteiden havainnollistamiseen, sillä sellaisten luokkien ja olioiden suhteet, jotka kuvassa on esitetty, ovat paikkansa pitäviä. Tarkempia tietoja saadakseen kannattaa kuitenkin perehtyä lukuun Ohjelman Rakenne.


Maasto
Ruutu
+ auto:
Option[Auto]
+ maasto: Maasto
+ poistaAuto(): Boolean
+ lisääAuto(Auto): Boolean
Auto
+ vaihde: Int
+ kuljettuReitti:
Vector[Tuple2[Int,Int]]
+ vaihdaVaihdetta(Int):
Unit
Pelilauta
+ ruudut:
Vector[Vector[Ruutu]]
+ siirraAuto(Auto, Tuple[Int,Int]):
Boolean
+ mahdollisetSiirrot(Auto):
Vector[Tuple2[Int,Int]]
+ onkoMaalissa(Auto): Boolean
- etsiAuto(Auto): Tuple2[Int,Int]
- loppuRuutu(Tuple2[Int,Int],
Tuple2[Int,Int], Int): Tuple2[Int,Int]
AI
- pelitilanne: Pelitilanne
+ siirto: Tuple2[Int,Int]
Pelaaja
+ auto: Auto
+ profiili:
Option[Profiili]
+ nimi: String
+ paivitaProfiili: Unit
+ onPelaaja:
Profiili
+ nimi: String
+ voitetutOttelut: Int
+ pelatutOttelut: Int
+ ennatysAjat: Map[String-
+ paivita(Boolean, String):
Pelitilanne
+ pelaajat:
Vector[Pelaaja]
+ vuorossa: Pelaaja
+ pelilauta: Pelilauta
+ siirraAutoa(Tuple[Int,Int]):
Boolean
- vaihdaVuoroa: Unit
- aiSiirto: Tuple2[Int,Int]
Object
Tiedostohallinta
+ lueRadat(): Vector[Rata]
+ lueProfiilit(): Vector[Profiili]
+ lisääProfiili(Profiili): Unit
+ lisääRata(Rata): Unit
Käyttää
Object Peli
+ radat: Vector[Rata]
+ profiilit: Vector[Profiili]
+ peliTilanne: Option[Pelitilanne]
+ uusiPeli(Rata, Vector[Pelaajat]):
Pelitilanne
+ peliKaynnissa: Boolean
+ lisääProfiili(Profiili): Unit
+ lisääRata(Rata): Unit
Käyttää
Object
Peli-ikkuna
Rata
+ ennatysAjat: Int
+ muoto: Pelilauta
Ruudukko