Kasino

Ohjelmointistudio 2 projekti

Nicholas Colb

591302

Informaatioverkostot 2016 - SCI

22.4.2017

I. Yleiskuvaus

Kyseessä on Scalalla tehty kasino-korttipeli, jossa on Scala Swing graafinen käyttöliittymä. Pelissä kerätään kortteja ja niillä pisteitä, jotka lasketaan jokaisen pelikierroksen lopussa. Peli jatkuu aina siihen asti kunnes joku pelaajista saavuttaa 16 pistettä. Käytössä on perinteiset kasinon säännöt, jotka voi kerrata a+:sta:
https://plus.cs.hut.fi/studio_2/2017/studioprojekti/109/
Pelaajia voi olla 2-4. Pelaaja voi tallentaa pelin jokaisen kierroksen välissä. Peliä voi pelata 1-3 robottivastustajaa vastaan. Roboteilla on erilaisia luonteita, ja näin ne tekevät erilaisia siirtoja kussakin tilanteessa. Peli on toteutettu vaativa-vaikeustasolla.

II. Käyttöohje

Peli on scala-projekti. Käynnistääkseen ohjelman käyttäjän tulee avata ui-paketti ja ajaa tiedosto play.scala. Pelaaja voi aloitusnäytöllä lukea pelin ohjeet, aloittaa uuden pelin tai ladata jo tallennetun pelin. Tekemällä jommankumman kahdesta viimeisestä pelaaja pääsee pelihuoneeseen, jossa hän voi lisätä vastustajia ja aloittaa pelin. Pelitilanteessa pelaaja voi omalla vuorollaan valita kortteja, tehdä siirtonsa tai katsoa tämänhetkisen pelitilanteen. Kierroksen päätyttyä pelaaja voi tallentaa pelin, jolloin ohjelma sulkeutuu.

III. Ohjelman rakenne

Ohjelma on jaettu melko selkeästi juuri suunnitelman UML-kaavion mukaisesti. Projektissa on viisi pakettia, joilla kullakin on omat osa-alueensa ja tehtävänsä. Files-paketissa on kaikki tarvittavat tiedostot peliin - siis kuvat ja äänet. Kasino.io huolehtii pelin tallentamisesta, ja sieltä löytyy reader, writer sekä tekstitiedosto, johon tallennetaan string-arvo. Io-toteutus muistuttaa hyvin pitkälti tämän kurssin Chess-projektin tallennusharjoitusta. Apumetodit chunkkien lukemiseen on lainattu suoraan kyseisestä projektista. Kasinocode-paketissa on kaikki pelin toimintaan liittyvä koodi. Paketissa on määritelty luokat keskeisille objekteille, joita pelissä on, kuten Card, Deck, Hand, Player ja RobotOpponent. Lisäksi kasinocode:ssa on määritelty tiedosto robottien tekoälyn luonteelle Personality.scala, jossa on erinäisiä case-luokkia, jotka ovat robottien “aivot”. Kasinocode-pakettiin kuuluu myös Logic.scala, joka toimiikoko pelin aivoina. Logic.scalassa tarkastetaan siirtojen laillisuus, lasketaan pisteitä ja määritetään voittajat. Logic.scalassa on myös metodi combinations2, jota robotit käyttävät löytääkseen kaikki mahdolliset siirtokombinaatiot. Paketissa on myös pelin “keskus” Table.scala, joka on luokka, joka ikään kuin kokoaa ja säilyttää pelitilanteessa tarvittavia objekteja muista luokista. Uusi table luodaan tallennetusta string arvosta kasino.io:ssa. Table.scala kommunikoi hyvin suoraviivaisesti projektin neljännen paketin, ui:n, kanssa, joka vastaa siitä, mitä käyttäjälle näytetään, eli mitä ruudulle piirretään. Ui:ssa on tiedostoja, jotka säilyttävät ruudulle piirrettävät vakiotekstit ja napit case-objekteina (esim. main menu -nappi), sekä useasti toistuvia, samankaltaisia objekteja case-luokkina (esim itse kortit, joissa samat ominaisuudet, mutta vain kuva vaihtuu.). Ui-paketissa on myös määritelty Screen.scala, joka on xy-koordinaatistoinen paneeli. Screen.scala hallinnoi varsinaisesti, mitä

ruudulla näytetään kullakin hetkellä, ja tekee tiivistä yhteistyötä table.scalan kanssa. Projektin viides paketti, test, sisältää projektiin liittyvät yksikkötestit. Testit on jaettu ohjelmatesteihin sekä io-tallennustesteihin.

Keskeisiä metodeja

- **ui/Screen.openRoom()**
    - Asettaa pelaajan huoneeseen ja avaa huoneen. Metodi varmistaa, että käyttäjä on antanut nimensä inputtina, ennen kuin huone piirretään näytölle. Tarkistus tehdään rekursiivisella check-metodilla ja ready-muuttujalla.
- **ui/Screen.restart()**
    - Lisää huoneessa näytölle kaikki aloitusobjektit. Tätä metodia kutsutaan aina, kun näyttö refreshataan, esimerkiksi kun pelaajien kortit vaihtuvat. Jotkut objektit täytyy kuitenkin poistaa, jos peli on kesken, metodilla gameOnClean().
- **ui/Screen.start()**
    - Aloittaa kierroksen, kun pelaaja painaa start- nappia. Määrittää table-objektiin dealerin.
- **ui/Screen.drawHand**
    - Piirtää Hand-objektia (objekti, joka säilöö kortti-objekteja) vastaavat kuvat näytölle. Parametriksi annetaan ensimmäisen kortin x ja y koordinaatit, itse Hand-objekti, boolean-arvo siitä, halutaanko kortit näkyviksi vai ei, boolean-arvo siitä, saako kyseisen

korttiryhmän kortteja klikata ja valita useamman kuin yhden kerrallaan sekä boolean arvo siitä- piirretäänkö kortit kokonaan näkyviin vai osittain päällekäin. Tätä samaa metodia voidaan parametrien ansiosta käyttää pöytäkorttien piirtämiseen(monta saa olla valittuna), pelaajan korttien piirtämiseen(vain yksi valittuna kerrallaan), robottien korttien piirtämiseen(korttien selkäpuoli näkyvissä) sekä tulosten piirtämiseen (kortteja sen verran paljon, että kortit tulee piirtää osittain toistensa päälle).

- **ui/Screen.makeMove()**
    - Pelin siirtämiseen liittyvä metodi, jota kutsutaan aina, kun peli alkaa tai edellisen pelaajan siirto on loppunut. Jos vuorossa on pelaaja, metodi alkaa rekursiivisesti tarkistamaan ajastimen kanssa, onko pelaaja painanut make-move-nappia, eli siis valinnut kortit ja pyytänyt siirrontekoa. Metodi tarkistaa tämän jälkeen, onko siirto laillinen Logic.scalan kautta. Laillinen siirto siirretään Player.scalaan ja toimitetaan loppuun asti, laiton siirto aloittaa puolestaan rekursion uudestaan. Robotin tapauksessa toimitaan melko samalla tavalla, paitsi että siirto tehdään RobotOpponent.scalassa, ja siirron jälkeen odotetaan 2 sekuntia, kunnes vuoro lopetetaan. (Tuo luontevuutta peliin).
- **kasinocode/RobotOpponent.findBestMove**
    - Tekoälyn keskeisin metodi, joka selvittää Logic.combinations-metodin kautta kaikki mahdolliset siirrot ja tämän jälkeen valitsee robotin oman persoonallisuuden mukaisen parhaan siirron. Käytännössä käydään for-loopissa läpi jokaisen robotin

kädessä olevan kortti. Loopissa kerätään kortin mahdolliset siirrot taulukkoon ja taulukko taitetaan robotin persoonallisuuden metodille **prioritize**, joka valitsee kahdesta siirrosta paremman. Kun kortin paras siirto on määritetty, sitä verrataan muiden kädessä olevien korttien parhaimpaan siirtoon ja niistä valitaan paras.

IV. Algoritmit

- **Siirtojen tarkistusalgoritmi**
    - **Logic.isLegit2**
        - Ottaa parametrikseen pöydältä valitut kortit taulukossa sekä kortin, joka on valittu kädestä. Tarkistetaan, voiko taulukon kortit jakaa osajoukoiksi apumetodin canFormSubSetin avulla. Näin vältytään esim. siltä, että ässällä saisi ottaa kortit (K,Q ja 3) mutta esimerkiksi (K,A,Q,2) olisi laillinen siirto ässällä (osajoukot K,A ja Q,2).
- **Mahdolliset siirrot pelitilanteessa**
    - **Logic.combinations2**
        - Käydään rekursiivisesti läpi taulukkoa, jossa on kortteja (pöytäkortit) ja kerätään mahdollisia kombinaatioita niin, että korttien summa = parametriksi annettu Int. Jos haettu summa ylittyy, returnataan pois rekursiosta ja pyritään etsimään seuraavaa kombinaatiota. Kun korttien summa = param Int, testataan vielä, että Bufferin kortit saa jaettua osajoukoiksi apumetodin canFormSubSetin avulla.

- **Tekoäly**

    - Hieman toistoa, mutta : Logic.combinations-metodin kautta selvitetään kaikki mahdolliset siirrot ja tämän jälkeen valitaan robotin oman persoonallisuuden mukainen paras siirto. Käytännössä käydään for-loopissa läpi jokaisen robotin kädessä oleva kortti. Loopissa kerätään kortin

mahdolliset siirrot taulukkoon ja taulukko taitetaan robotin persoonallisuuden metodille **prioritize**, joka valitsee kahdesta siirrosta paremman. Kun kortin paras siirto on määritetty, sitä verrataan muiden kädessä olevien korttien parhaimpaan siirtoon ja niistä valitaan paras.

- Robottien luonteella on myös metodi **oneOfFour,** joka määrittää robotin korteista sen, mikä kannattaa laittaa pöydälle, jos muuta siirtoa ei voida tehdä. Metodi käy läpi eri tilanteet (max 4, koska kädessä aina 4 korttia) ja selvittää, mikä niistä on edunmukaisin robotille.

V. Tietorakenteet

Päätin alusta alkaen käyttää muuttuvia ArrayBuffereita, koska pelissä on todella paljon muuttuvia tekijöitä (pelaajat, kortit kädessä, kortit pöydällä, kaapatut kortit, voittajat) yms. Aluksi toteutukseni Hand.scalaan oli sellainen, että Hand oli scala.collection.mutable.Buffer:in ilmentymä. Halusin kuitenkin muuttaa tietorakenteen sellaiseksi, että Handit ovat itsenäisiä, ainutlaatuisia olioita, vaikka niissä olisi samat kortit. Näin vältytään esimerkiksi sellaisilta virheiltä, että jos jossain ohjelman tilassa vaikkapa pöydällä ja kädessä on sama kortti (ennen kuin se on ehditty kädestä poistaa), niin Hand oliot eivät mene sekoitu. Siis jokaisella Hand-oliolla on oma parametri components, joka on ArrayBuffer.

VI. Tiedostot

Kuvat ovat png-tiedostoja, äänet wav-tiedostoja. State.txt toimii pelin tallennuksessa tekstitiedostona, johon tallennetaan yksi string-arvo. String näyttää esim tältä:

PLR06jaakko0005PLR04asta1108ROB05jorma0009ROB14klausVonHerzen0015END00

- PLR/ROB -tunniste kertoo, onko kyseessä pelaaja vai robotti.
- Seuraavat 2 numeroa kertovat nimen pituuden ‘k’.
- Seuraavat k-kirjainta ovat pelaajan nimi.
- Seuraava binäärinumero kertoo, onko pelaaja dealeri vai ei
- Seuraava binäärinumero kertoo, oliko pelaaja ensimmäisenä huoneessa (Tämä sen takia, että ensimmäinen pelaaja on aina ruudun alaosassa)
- Seuraavat kaksi numeroa kertovat pelaajan pisteet

Tunniste END00 kertoo, että tiedosto on päättynyt.

VII. Testaus

Tekoälyni ja tarkistusalgoritmini oli ensin puutteellinen. Olen jättänyt muutamaan kohtaan koodia viallisen tekoälyn koodia, (tägeillä REDUNDANT), jotta voin tutkiskella niitä myöhemmin. Aliarvioin tarkistusalgoritmin haastavuuden, ja tutkin sitä ensin pelkän modulon avulla..

Testaus tapahtui yksinkertaisilla yksikkötesteillä. Pyrin painottamaan, kuten suunnitelmassa totesinkin, mahdollisia, absurdeja pelitilanteita. Tilanteita on varmasti paljon enemmänkin, ja aina voisi tehdä enemmän testejä. Ohjelma läpäisee kaikki suunnitelmassa esitetyt testit. Testasin ohjelmaa alussa enemmän yksikkötesteillä, koska graafista

käyttöliittymää ei ollut. Kun GUI alkoi olla valmis, pystyin nähdä tulokset ruudulla, kun ajoin ohjelman. Kuitenkin esimerkiksi “backend” puolen, kuten tekoälyn, testausta piti yksikkötesteissä tarkastella.

Puutteet ja viat:

Combinations- algoritmi ei ole mitenkään kovin tehokas, ja toistoja tulisi aivan liikaa, jos taulukoiden koot kasvaisi. Kuitenkin tradeoff ratkaisuna tässä pelissä algoritmi toimii mielestäni hyvin.

Nimi saa olla enintään 10 merkkiä pitkä, koska muuten nimet alkavat mennä yli ruudun. Onkohan tämä ratkaistu juuri näin markkinoilla olevissa peleissä?

Maksimissaan neljä pelaajaa, koska muuten tuloksia tarkasteltaessa ruutu näyttäisi erittäin ahtaalta. Painotin tässä esteettisyyttä.

Peliä ei voi tallentaa kesken kierroksen. Tämä oli asia missä säästin aikaa, vaikka koodi ei olisi ollut vaikea toteuttaa. Vaatimuksena oli “pelitilanteen” tallentaminen ja lataus, ja ohjelma täyttää sen kuitenkin tallentamalla pisteet, pelaajat ja sen, kuka on jakaja. Tämän voisi ehkä lisätä peliin lisäkehityksenä.

Pelaajien kaappaamia kortteja ei pysty tarkastelemaan muulloin kuin omalla vuorolla. Lisäsin ominaisuuden pelattavuuden takia lisäominaisuutena, enkä ehtinyt lisätä sitä, että esimerkiksi robotit tekisivät vuorojaan samalla, kun pelaaja tarkastelee kaapattuja kortteja. Tämän tulen vielä lisäämään peliin.

Main menu -nappia painaessa pelin pitäisi pysähtyä välittömästi. Jos on robotin vuoro, vuoro kuitenkin jatkuu loppuun asti. Vuoroa ei kuitenkaan lopeteta gameOn parametrin takia (koska se on false siinä vaiheessa). Tätä olisi voinut hiota jollain, sillä koodissa on nyt try-catch rakenne makeMove:n ympärillä Screen.scalassa. Mitenköhän tämä on toteutettu markkinapeleissä?

Main menua painaessa koko peli haihtuu olemattomiin, jos sitä ei ole tallennettu. Tähän olisi voinut lisätä jonkun “are you sure you want to quit”-popupin.

3 parasta kohtaa:

- Erilaiset robottivastustajat
- Ui, musiikki ja pelattavuus. Sain tästä runsaasti positiivista palautetta ja pyrin panostamaan tähän.
- Ohjelma antaa käyttäjälle melko paljon anteeksi. Pyrinkin ottamaan tätä huomioon. Esimerkiksi nimen ylipituus on ennaltaehkäisty. Lisäksi jos 2 ihmispelaajaa pelaavat koko pelin niin, että kortteja vain laitetaan pöytään, kortit eivät mene näytön ulkopuolelle, vaan muodostavat rivejä pöydällä. Ohjelma poistaa valinnat automaattisesti, jos yritetään tehdä väärää siirtoa. Käyttäjän ei myöskään itse tarvitse poistaa edellistä valintaa omasta kortistaan, jos haluaakin valita toisen kortin, vaan ohjelma huolehtii siitä, että vain yksi pelaajan kortti voi olla valittuna. Robottien kortteja yms ei pysty valitsemaan. Command-teksti ohjeistaa pelaajaa melko selkeästi.

Huonot kohdat löytyvät tuosta ylempää. Otsikon "puutteet ja viat" alta.

VIII. Poikkeamat suunnitelmasta

Jätin animaation tekemättä, koska peli oli mielestäni yksinkertaisuudessaan hieno näin. (Olin toteuttanut animaation jo blackjack-projektissani, joten vaiva ei olisi ollut suuri.) Se ei mielestäni tuonut blackjackiin kovinkaan paljoa lisäarvoa, vaan teki siitä sekavamman näköisen.

Projekti viivästyi hieman, koska innostuin ui:n hiomisesta liikaa. Suunnitelmani siis aliarvioi ajankäyttöä. Tein ensin tekstiversion, kuten suunnitelmassani totesinkin. Tämän jälkeen tein ui:n perustoiminnallisuudet, minkä jälkeen aloin hioa esteettisyyttä.

IX Arvio lopputuloksesta

Olen erittäin tyytyväinen lopputulokseen. Ui näyttää upealta, ja pelin perustoiminnallisuudet ovat kunnossa. Ylitin tavoitteeni tekemällä useita erilaisia robotteja ja minusta tuntuu, että olen oppinut erittäin paljon projektin ohella. Projektia on ollut ennen kaikkea hauska tehdä.

Aliarvioin tarkistusalgoritmin haastavuuden, sillä tarkistin ensin siirrot pelkän modulon ja korttien suuruuden avulla. Esim. algoritmini päästi siirron 8 ->(4,4,8) läpi, koska 16 mod 8 = 0. Algoritmi ei päästänyt siirtoa 4->4,4,8 läpi, koska pöydästä otettu 8 olisi tällöin

suurempi kuin kädessä oleva kortti. Kuitenkin moduloalgoritmi osoittautui pian vialliseksi, koska esimerkiksi siirto A ->(K,Q,3) ei ole laillinen, vaikka modulo onkin nolla. Jouduin siis tekemään uuden algoritmin hieman kiireellä. Note to self: Mieti ne algoritmit aina loppuun asti! Kuitenkin nyt vaikuttaisi siltä, että loppu hyvin kaikki hyvin, vaikkei algoritmi kaikista tehokkain olekaan. Peliin se sopii, sillä taulukon koko on rajallinen.

Koodia voisi siistiä, ja pieniä lisäominaisuuksia voisi lisätä. (Esimerkiksi se, että main menu-nappia painaessa koko peli haihtuu olemattomiin jos sitä ei ole tallennettu. Tähän voisi lisätä jonkun popupin Are you sure -tyylisesti.) Ohjelman hyvänä puolena pidän sitä, että se huomioi pelattavuutta. Esimerkiksi, nimen ylipituus on ennaltaehkäisty. Lisäksi jos 2 ihmispelaajaa pelaavat koko pelin niin, että kortteja vain laitetaan pöytään, kortit eivät mene näytön ulkopuolelle, vaan muodostavat rivejä pöydällä. Ohjelma myös poistaa valinnat automaattisesti, jos yritetään tehdä väärää siirtoa. Käyttäjän ei myöskään itse tarvi poistaa edellistä valintaa omasta kortistaan, jos haluaakin valita toisen kortin, vaan ohjelma huolehtii siitä, että vain yksi pelaajan kortti voi olla valittuna. Robottien kortteja yms ei pysty valitsemaan, eikä ohjelmaa näin sekoittamaan. Command-teksti ohjeistaa pelaajaa melko selkeästi läpi pelin. Pelin tallentaminen on myös mielestäni hoidettu tehokkaasti, vaikka tottakai olisi voinut lisätä sen, että pelitilanteen voi tallentaa kesken kierroksenkin.

Uskon, että ohjelmasta voisi tehdä toisen korttipelin suhteellisen pienellä vaivalla, eli siinä mielessä se sopii laajennukseen. Toisaalta, jos ohjelmasta haluttaisiin tehdä esimerkiksi online-nettipeli, olisi hiottavaa vielä paljon.

Viitteet: Stackoverflow sekä Wikipedia. Äänet soundbiblesta, musiikki youtubesta https://www.youtube.com/watch?v=jX8HxO-ylHU