Henkilötiedot

Jimi Väyrynen

102778188

Tietotekniikka

2024

28.4.2025

Yleiskuvaus

Projektityönä toteutin “Mökki Kasino”-korttipelin Scala-ohjelmointikielellä, käyttäen ScalaFX-kirjastoa graafisen käyttöliittymän toteuttamiseen. Pelissä useat pelaajat keräävät pisteitä keräämällä kortteja ja tekemällä “mökkejä”. Ensimmäinen 16 pistettä saavuttanut pelaaja voittaa. Peli tukee myös keskivaikean tietokone vastustajan lisäämistä. Työ toteutettiin vaativan tason määrittelyn mukaisesti.

Käyttöohje

Ohjelma käynnistetään IntelliJ IDEA:ssa ScalaFX-projektina ajamalla Main.scala. Alkunäytössä valitaan seuraavat asiat: Pelaajien määrä (1-4), Otetaanko mukaan Tekoäly vastustaja, sekä pelaajien nimet. Pelissä pelaajat valitsevat kädestään kortin ja ottavat yhdistelmän kortteja pöydältä tai asettavat kortin pöytään. Tietokonepelaaja tekee automaattisesti oman siirtonsa. Peli jatkuu, kunnes pelaaja saavuttaa 16 pistettä. Tiedostosta voi myös tallentaa ja ladata pelin (tekstitiedosto tallennus.txt)

Ohjelman rakenne

Main.scala: Vastaa graafisesta käyttöliittymästä (ScalaFX) ja koko pelin ohjaamisesta. Sisältää myös aloitusvalikon, jossa valitaan pelaajat ja tekoäly.

Game.scala: Sisältää pelin ydintoiminnot kuten pelaajien vuorottelun, pisteiden päivityksen, korttien jakamisen ja pelin loppumisen ehdot.

Player.scala: Määrittelee pelaajaluokan, joka kuvaa ihmispelaajaa ja ComputerPlayer-luokan, joka mallintaa tekoälyvastustajaa.

Board.scala: Hallinnoi pelipöytää ja siihen liittyvää pakkaa. Sisältää metodit korttien jakamiseen, pöytään lisäämiseen ja pakan nollaamiseen uuden kierroksen alkaessa.

Deck.scala: Vastaa korttipakan luonnista, sekoittamisesta ja korttien jakamisesta.

Card.scala: Mallintaa yksittäistä pelissä käytettävää korttia (arvo ja maa).

CardValidator.scala: Sisältää logiikan, joka tarkistaa onko siirto pöydältä laillinen (esimerkiksi saman arvoiset kortit tai summakombinaatiot).

Scoring.scala: Vastaa pisteiden laskemisesta jokaisen pelikierroksen päätteeksi (mökki, ässät, ruutu-10, pata-2 jne.).

FileHandler.scala: Mahdollistaa pelin tallentamisen ja lataamisen tekstimuotoiseen tiedostoon, joka sisältää pelaajien tilanteet, pöytäkortit ja pakat.

Algoritmit
Korttien ottaminen pöydältä: Pelaaja voi ottaa kortteja, jotka vastaavat kortin arvoa tai muodostavat summan

Pseudokoodina:

Syöte: Pelaajan kortti (pleyerCard), pöydällä olevat kortit (tableVards)

1. Laske pelaajan kortin numeerinen arvo (esim. jätkä = 11).
2. Etsi pöydältä kaikki kortit, joiden arvo on:

    a) Yhtä suuri kuin pelaajan kortin arvo.

    b) Tai muodostavat summana pelaajan kortin arvon.

3. Palauta lista kelvollisista korttiyhdistelmistä.

Toiminta:

- Luo tyhjä lista mahdollisista yhdistelmistä.

- Käy läpi kaikki pöytäkorttien osajoukot (ei-tyhjät).

- Jos osajoukon korttien summa == pelaajan kortin arvo, lisää osajoukko mahdollisena ottona.

- Palauta kaikki löydetyt mahdolliset korttiyhdistelmät.

Tekoälyn päätöksenteko: Tietokone valitsee ensisijaisesti siirron, jolla saa eniten kortteja tai pisteitä, välttäen jättämästä helppoja mökkejä.

Vuoronvaihto: Vuorot pyörivät järjestyksessä, tekoäly tekee siirtonsa automaattisesti

Tietorakenteet
List[PlayingCard]: Käytetään pelaajien käsissä olevien korttien, pöydällä olevien korttien ja pakan korttien hallintaan. Listaan kohdistuu operaatioita, kuten korttien lisääminen, poistaminen ja hakeminen. Valitsin tämän rakenteen, koska se on yksinkertainen, helposti hallittava ja soveltuu hyvin pieniin kokoelmiin, joissa operaatioiden nopeus ei ole kriittinen. Muina vaihtoehtoina olisivat voineet olla esimerkiksi Array tai Buffer, mutta List on kätevin ja selkein ratkaisu. List on muuttumaton (immutable), mikä helpottaa turvallista ja virheetöntä datan käsittelyä.

Set[PlayingCard]: Käytetään pelissä korttien valitsemiseen pöydältä pelivuoron aikana. Set tarjoaa tehokkaan tavan hallita valittuja kortteja ilman duplikaatteja ja mahdollistaa nopean

tarkistuksen, onko tietty kortti valittu. Vaihtoehtona olisi voinut käyttää myös List-tyyppiä, mutta Set on selvästi parempi, kun halutaan varmistaa, ettei samaa korttia valita kahdesti. Set on muuttumaton (immutable), mikä tekee datan käsittelystä turvallista ja virheriskeiltä suojattua.

Option[T]: Käytetään valinnaisten tietojen tai tapahtumien turvalliseen hallintaan, kuten esimerkiksi mahdollisesti tyhjien korttivalintojen tai tekoälyn siirtojen yhteydessä. Option mahdollistaa sen, että koodissa ei tapahdu virheitä tyhjien tai olemattomien arvojen käsittelystä. Sen avulla ohjelma voi turvallisesti käsitellä tilanteita, joissa dataa ei välttämättä ole saatavilla. Vaihtoehtona olisi ollut käyttää suoria arvoja tarkistuksilla (null-tarkistukset), mutta Option-rakenne on Scala-kielessä selvästi turvallisempi ja elegantimpi ratkaisu. Option on muuttumaton (immutable).

Näiden tietorakenteiden valinta perustuu niiden tarjoamaan turvallisuuteen ja käytettävyteen.

Tiedostot ja verkossa oleva tieto
Ohjelma käyttää tekstitiedostona (tallennus.txt) pelitilan tallentamiseen ja lataamiseen.

Tiedostoformaatti:

PLAYER:Nimi:Pisteet:Kortit

TABLE :Kortit

DECK: Kortit

TURN: Nimi

Esimerkki tiedostostosta:

PLAYER:Jimi:5:ace_of_spades,2_of_hearts

TABLE:5_of_clubs,10_of_diamonds

DECK:3_of_spades,4_of_hearts

TURN:Jimi

Testaus
Ohjelman testaus suoritettiin pääosin manuaalisesti graafisen käyttöliittymän kautta:

- Pelimekaniikan testaus (korttien pelaaminen ja kerääminen).
- Pistelaskun tarkkuus.
- Tekoälyn päätöksenteon toimivuus.
- Tiedostojen tallennuksen ja latauksen testaus.

Testauksen aikana löytyneet virheet korjattiin välittömästi. Yksikkötestejä ei erikseen laadittu, koska projekti keskittyi ensisijaisesti toiminnalliseen testaamiseen.

Testaus toteutettiin pitkälti suunnitelman mukaisesti.

Ohjelman tunnetut puutteet ja viat

Tekoäly voi toisinaan tehdä epätarkkoja tai strategisesti heikkoja siirtoja. Tätä voisi kehittää käyttämällä syvällisempää tekoälylogiikkaa tai koneoppimismenetelmiä.

Tiedostonlukutoiminto ei tarkista kaikkia mahdollisia virheellisiä tiedostoformaatteja. Virheenkäsittelyn parantaminen olisi hyvä seuraava askel.

3 parasta ja 3 heikointa kohtaa
Kolme parasta:

1. Pelilogiikan selekys ja toiminta: Pelin keskeiset säännöt ja toiminnot toteutettiin selkeästi ja luotettavasti. Korttien yhdistely ja pöydältä ottaminen toimivat tarkasti pelin sääntöjen mukaan, mikä tekee pelistä sujuvan ja mielekkään pelata.
2. Tekoälypelaajan toteutus: Keskivaikean tekoälypelaajan kehitys onnistui hyvin, ja tekoäly pystyy tekemään järkeviä ja taktisesti perusteltuja siirtoja, mikä lisää pelin monipuolisuutta ja haastetta.
3. Tiedostonhallinta (Tallennus ja lataus): Tiedostojen tallennuksen ja latauksen toteuttaminen onnistui käyttäjäystävällisesti ja selkeästi. Tekstitiedostoformaatti on helposti luettavissa ja muokattavissa, mikä helpottaa testausta ja virheiden jäljittämistä.

Kolme heikointa:

1. Graafiden käyttöliittymän visuaalinen toteutus: Käyttöliittymä toimii hyvin toiminnallisesti, mutta visuaalinen ilme on yksinkertainen ja sitä voisi edelleen kehittää houkuttelevammaksi lisäämällä grafiikkaa, animaatioita tai tarkempaa tyyliä.

   Korjausmahdollisuus: Graafista suunnittelua voisi kehittää käyttämällä enemmän aikaa käyttöliittymän visuaalisiin yksityiskohtiin ja mahdollisesti lisäämällä teemaan sopivaa grafiikkaa.

2. Virheenkäsittely tiedostonhallinnassa: Nykyinen toteutus ei käsittele kattavasti kaikkia mahdollisia virheitä tiedostojen lukemisen yhteydessä, mikä voi johtaa ongelmiin virheellisen tiedoston kanssa.

   Korjausmahdollisuus: Virheenkäsittelyyn voisi lisätä tarkistuksia ja palauttaa selkeitä virheilmoituksia käyttäjälle, jotta virhetilanteita voitaisiin hallita paremmin.

3. Tekoälyn strategian syvyys: Vaikka tekoäly tekee järkeviä siirtoja, sen päätökset perustuvat melko yksinkertaiseen heuristiikkaan. Tämä voi toisinaan johtaa epäoptimaalisiin ratkaisuihin.

Korjausmahdollisuus: Tekoälyn toimintaa voisi parantaa lisäämällä syvällisempää päätöksentekologiikkaa, esimerkiksi minimax-algoritmia tai muita edistyneempiä tekoälyn menetelmiä.

Poikkeamat suunnitelmasta, toteutunut työjärjestys ja aikataulu

Aluksi keskityin toteuttamaan pelin peruslogiikan, kuten korttien jaon, pelivuorojen hallinnan, pöytäkorttien yhdistämisalgoritmit ja alustavan pistelaskennan. Tässä vaiheessa toteutus pysyi suunnitelman mukaisena, ja aikataulu piti erittäin hyvin. Keskeiset luokat (Game.scala, Player.scala, Board.scala ja Deck.scala) toteutettiin ensin.

Seuraavaksi aloitin graafisen käyttöliittymän rakentamisen ScalaFX-kirjastolla (Main.scala). Tämä vaihe vei ennakoitua enemmän aikaa, koska käyttöliittymän yksityiskohdat, kuten animaatiot ja korttien valinta, osoittautuivat suunniteltua haastavammiksi. Jouduin myös käyttämään ylimääräistä aikaa käyttöliittymän testaukseen ja käytettävyyden parantamiseen. Tämä poikkesi hieman alkuperäisestä suunnitelmasta, sillä alun perin käyttöliittymän rakentamisen ei arvioitu vievän niin merkittävästi lisäaikaa.

Kolmannessa vaiheessa keskityin lisäämään peliin keskivaikean tekoälyn (ComputerPlayer.scala) ja pelin tallennus- ja lataustoiminnot (FileHandler.scala). Alkuperäisessä suunnitelmassa tekoälyn kehitykseen ei ollut varattu merkittävästi aikaa, mutta huomasin pian, että realistisesti toimivan tekoälyn toteutus vei enemmän aikaa kuin olin suunnitellut. Tämän johdosta projektin loppuvaiheessa käytin ylimääräistä aikaa tekoälyn siirtojen testaamiseen ja hienosäätöön.

Projektin aikana opin erityisesti aikataulun tarkemman arvioinnin tärkeyttä sekä sen, että käyttöliittymän ja tekoälyn kaltaiset monimutkaisemmat ominaisuudet vaativat realistisen ajanvarauksen. Opin myös jakamaan projektia tehokkaasti pienempiin osakokonaisuuksiin ja työskentelemään iteratiivisesti, jolloin mahdollisiin ongelmiin voitiin puuttua nopeasti.

Kokonaisarvio lopputuloksesta

Mökki Kasino -projekti saavutti asetetut tavoitteet ja on pelattavissa alusta loppuun ilman suurempia ongelmia. Pelin peruslogiikka on selkeä ja toimiva, ja käyttökokemus on intuitiivinen. Tekoälyn lisääminen toi projektiin merkittävää lisäarvoa ja paransi pelin haastavuutta sekä uudelleenpelattavuutta. Ohjelma onnistuu myös tallentamaan ja lataamaan pelitilanteita luotettavasti ja käyttäjäystävällisesti.

Parannusehdotukset tulevaisuudelle:

- Käyttöliittymän visuaaliseen ulkoasuun voisi käyttää enemmän aikaa ja mahdollisesti integroida pelin teemaan sopivaa grafiikkaa ja animaatioita.
- Tiedostonhallinnan virhetilanteiden käsittelyyn voisi lisätä kattavat tarkistukset ja selkeät virheilmoitukset.
- Tekoälyn strategista syvyyttä voisi lisätä käyttämällä edistyneempiä algoritmeja, kuten minimaxia tai muita strategisia päätöksentekomenetelmiä.

Ratkaisumenetelmien, tietorakenteiden ja luokkajaon arviointi:

Valitut tietorakenteet (List, Set ja Option) olivat toimivia ja tukivat hyvin pelin toiminnallisia vaatimuksia. Scala-kielen muuttumattomat rakenteet (immutable) tarjosivat turvallisen pohjan ohjelman rakentamiselle. Luokkajako ja ohjelman rakenne ovat loogisia ja hyvin organisoituja, mikä mahdollistaa ohjelman helpon ylläpidon ja laajennettavuuden. Luokkien väliset suhteet on toteutettu selkeästi, eikä merkittäviä muutoksia tarvitsisi tehdä myöhemmissä laajennuksissa.

Jos aloittaisin projektin uudelleen, käyttäisin enemmän aikaa alkuvaiheessa käyttöliittymän suunnitteluun ja visuaalisen toteutuksen suunnitteluun. Lisäksi ottaisin yksikkötestauksen tiiviimmin mukaan kehitysprosessiin alusta alkaen, jolloin pienet virheet ja ongelmat olisi mahdollista tunnistaa ja ratkaista nopeammin ja systemaattisemmin. Käyttäisin myös enemmän aikaa tekoälyn strategisen syvyyden suunnitteluun ja toteutukseen, jotta lopputuloksesta tulisi haastavampi ja älykkäämpi.

Kokonaisuutena projekti oli erittäin opettavainen ja kehitti erityisesti taitojani iteratiivisessa ohjelmistokehityksessä, aikataulunhallinnassa sekä ongelmanratkaisussa monimutkaisten kokonaisuuksien yhteydessä.

Viitteet ja muualta otettu koodi
Kirjallisuus ja verkkosivut:

- ScalaFX:n virallinen dokumentaatio:
  https://www.scalafx.org/docs/home/
  Käytin tätä lähdettä graafisen käyttöliittymän toteutuksen oppimiseen sekä ScalaFX-komponenttien käyttöön liittyvien ongelmien ratkaisemiseen.
- Scala-dokumentaatio:
  https://docs.scala-lang.org
  Käytin virallista dokumentaatiota varmistaakseni Scala-kielen ominaisuuksien oikean ja tehokkaan käytön ohjelmoinnissa.
- Stack Overflow:
  https://stackoverflow.com
  Käytin Stack Overflow -kysymyksiä ja vastauksia erityisesti ongelmatilanteissa, jotka koskivat ScalaFX:n käyttöä ja Scala-kokoelmien käsittelyä. En suoraan kopioinut koodia Stack Overflow -sivustolta, vaan hyödynsin vastauksia lähinnä ideoinnin ja ongelmanratkaisun tukena.

Käytetyt kirjastot:

- ScalaFX:
  Käytetty pelin graafisen käyttöliittymän rakentamiseen. ScalaFX tarjosi valmiit ja helposti muokattavat komponentit visuaalisten elementtien, kuten nappien ja korttien kuvien, toteuttamiseen.

Tekoälyn käyttö (ChatGPT):

- Ideoiden ja neuvojen kysymiseen erityisesti tekoälyn toteutuksen suunnittelussa ja algoritmien ideoinnissa.

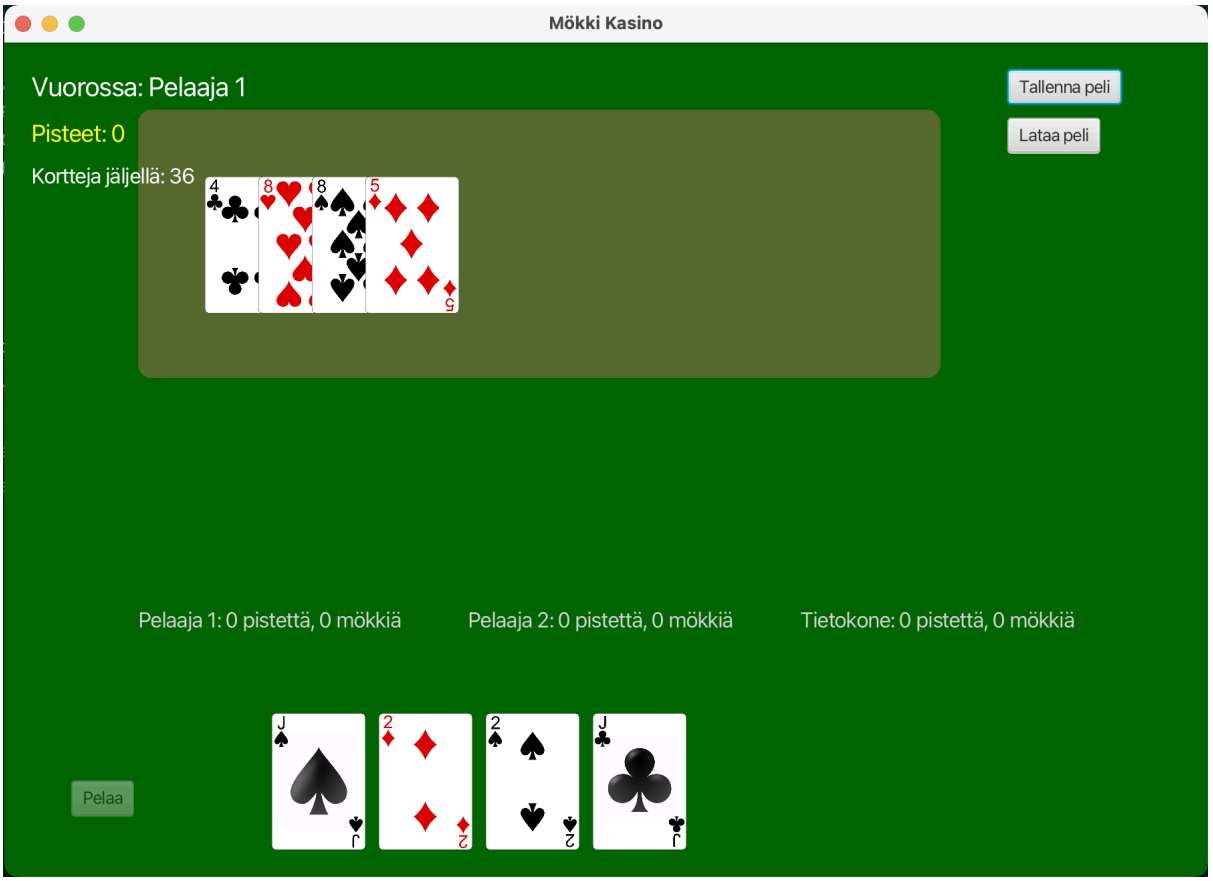
- Käytännön neuvoja tiedostonhallinnan toteutukseen ja ScalaFX-komponenttien käyttöön liittyvien ongelmien ratkaisuun.
- ChatGPT:n avulla generoin ajoittain pieniä esimerkkejä ja pseudokoodia, mutta kaikki tuotettu materiaali tarkistettiin ja sovellettiin oman projektini tarpeisiin.

Suoraa kopiointia toisilta opiskelijoilta ei tapahtunut, eikä suoraan valmista koodia kopioitu. Kaikki projektin koodi on itse kirjoitettua ja sovellettua.

Liitteet

Aloitusnäyttö:

Pelinäyttö:


Mökki Kasino
Vuorossa: Pelaaja 1
Pisteet: 0
Kortteja jäljellä: 36
Tallenna peli
Lataa peli
Pelaaja 1: 0 pistettä, 0 mökkiä
Pelaaja 2: 0 pistettä, 0 mökkiä
Tietokone: 0 pistettä, 0 mökkiä
Pelaa