Aalto University

CS-A1121 - Ohjelmoinnin peruskurssi Y2

Kolmiopalapeli: Tekninen suunnitelma

Rustam Latypov (474461)
Teknillisen fysiikan ja matematiikan koulutusohjelma
3. vuosikurssi

3. maaliskuuta 2019

1 Ohjelman rakennesuunnitelma

Ohjelman graafinen ilme on varsin yksinkertainen, joten se luodaan sen PyGame-kirjastolla [1]. Sillä onnistuu käyttäjänsyötteen (hiiren) toimintojen lukeminen vaivattomasti. PyGame tarjoaa kaikki tarvittavat luokat graafiselle puolelle. Ohjelman sisäinen logiikka tulee viemään eniten aikaa ja sen luokkarakenne on hyvä miettiä valmiiksi. Sitä varten luodaan seuraavat luokat:

- Game – Pelikokonaisuus, joka pitää kirjaa pelilaudasta, ”hyllystä” ja pelipaloista. Olio luodaan kerran pelin alussa ja se voidaan tallentaa/ladata hyödyntäen ulkoista tiedostoa.
  - saveTo: tallentaa pelikokonaisuuden ulkoiseen tiedostoon.
  - loadFrom: lataa pelikokonaisuuden ulkoisesta tiedostosta.
  - toBoard: siirtää palan hyllyltä pelilaudalle.
  - toShelf: siirtää palan pelilaudalta hyllylle.
  - solveGame: ratkaisee pelin dynaamisen ohjelmoinnin algoritmilla.
- Board – Varsinainen pelilauta, joka on aluksi tyhjä ja näyttää pelaajalle pelkät raamit.
  - checkValidity: testaa siirron laillisuutta.
  - addItem: lisää pelipalan pelilaudalle.
  - removeItem: poistaa pelipalan pelilaudalta.
- Shelf – Hylly, missä säilytetään käyttämättömät pelipalat.
  - addItem: lisää pelipalan hyllylle.
  - removeItem: poistaa pelipalan hyllyltä.
- Item – Pelipala, jolla on kolme orientaatiota ja joka kuuluu sijoittaa pelilaudalle.
  - rotate: vaihtaa kerran palan orientaatiota $360°/3 = 120°$ astetta.
  - position: palauttaa palan sijainnin.

Näiden olioiden avulla peli pelataan ja PyGamen avulla ne viedään graafisen liittymän puolelle. Oliot liittyvät toisiinsa niin, että Game luodaan alustetulla Boardilla ja Shelfilla ja Itemeillä. Board ja Shelf vaihtavat pelipaloja keskenään pelin aikana.

2 Käyttötapauskuvaus

Kun pelaaja käynnistää pelin, häneltä kysytään, haluaako hän aloittaa uuden pelin vai jatkaa edellistä. Pelaajan valinnan perusteella ohjelma joko arpoo uuden pelitilanteen tai lataa vanhan pelitilanteen ulkoisesta tiedostosta. Molemmissa tapauksissa luodaan Game-olio.

Peli alkaa tyhjästä pelilaudasta ja täydestä hyllystä, jossa on sekalaisessa järjestyksessä pelipaloja. Pelipalat ovat kolmioita, joilla jokainen sivu on väritetty jollain värillä. Pelilauta on kuusikulmio, minne mahtuu 24 pelipalaa ja jonka seinämät ovat myös väritetty. Painamalla vasenta hiirinäppäintä pelaaja voi sekä valita tietyn pelipalan että pudottaa sen halutulle paikalle. Hiiren oikealla hiirinäppäimellä pelaaja voi muuttaa pelipalan orientaatiota. Pelin päämäärä on täyttää pelilauta tarjotuilla pelipaloilla niin, että jokaisella sivuparilla ja sivuseinäparilla on sama väri. Jos pelaaja yrittää tehdä laittomia pelisiirtoja, ruudulle ilmestyy kirjallinen selitys virhetilanteelle ja pelipala palautetaan hyllylle.

Pelin aikana pelaaja voi aina tallentaa pelitilanteen ja poistua ohjelmasta, aloittaa suoraan uuden pelin tai vaihtoehtoisesti pyytää ohjelmaa ratkaisemaan pelin. Valinnoista riippumatta, Game-olio toimii rajapintana pelaajan käskyjen ja varsinaisten funktiokutsujen välillä. Game-olion kautta ohjataan pelilautaa ja hyllyä. Pelilauta ja hylly toisaalta ohjaavat varsinaisia pelipaloja.

3 Algoritmit

Projektini sisältää kaksi pientä algoritmia jotka luovat ratkaisun ja sekoittavat pelipalat pelaajaa varten. Niiden toiminta on suoraviivainen. Ratkaisun luominen tapahtuu täyttämällä pelilauta värittömillä pelipaloilla ja valitsemalla satunnaisesti värin, millä sivupari tai sivuseinäpari väritetään. Tämä toistetaan, kunnes pelilaudassa on ratkaisu. Ratkaisun sekoittaminen hyödyntää myös satunnaisuutta, sillä se arpoo jokaiselle palalle orientaation ja sijainnin hyllylle.

Projektin keskeinen algoritmi on pelin ratkaisun löytäminen hyödyntämättä mitään esitietoja. Sillä pelilauta ei ole kovin suuri, eikä orientaatioiden määrä ole suuri, voidaan käyttää dynaamisen optimoinnin lähestymistapaa. Sillä pelilaudan reunoilla on oma väri, tehtävä helpottuu. Algoritmi rakentaa ratkaisua aina laudan tai muiden palojen reunoja pitkin. Algoritmi rakentaa ratkaisun rekursiivisesti käymällä läpi kaikki mahdolliset vaihtoehdot tietylle sijainnille. Jos tietylle sijainnille ei löydy värien puolesta sopivaa palaa, ratkaisu on virheellinen ja rekursiossa palataan takaisin. Algoritmin rakenne näyttää alaspäin aukeavalta puulta, missä jokainen solmu on valinta käyttää tiettyä palaa tietyssä orientaatiossa.

4 Tietorakenteet

Ohjelmassa ei ole syytä käyttää muita tietorakenteita kun listoja. Osa listoista tulee olemaan dynaamisia, osa ei. Pelilaudan koordinaatisto tulee olemaan staattinen matriisi (lista listoja), mutta hylly tulee olemaan pelipaloja sisältävä dynaaminen lista. Toinen vaihtoehto voisi olla dictionary-tietorakenne, mutta sen käyttö ei tuo mitään etua listoihin verrattuna.

Ulkoinen tiedosto mihin tallennetaan tai mistä ladataan pelitilanne on .csv tiedosto, tarvittavalla määrällä rivejä ja sarakkeita kuvaamaan pelipalojen määrää ja ominaisuuksia. Tätä formaattia käytetään, sillä paloja on aina vakiomäärä ja niillä on aina vakiomäärä ominaisuuksia. CSV-formaattia on lisäksi helppo

käsitellä pandas kirjastolla [2]. Toinen vaihtoehto olisi .txt, mutta sen käyttö ei tuo mitään etua .csv verrattuna.

5 Aikataulu

Viikko 10 (8h)

Käytettävien grafiikkakirjostoihin tutustuminen ja PyCharm-Gitlab yhteyden muodostaminen. Tärkeimpien olioiden suunnittelu ja luominen. Pelin graafisen liittymän rungon toteuttaminen.

Viikko 11 (20h)

Muiden tarpeellisten olioiden luominen ja graafisen liittymän hiominen. Pelin tallentamisen ja lataamisen mahdollistavien tietorakenteiden suunnittelu.

Viikko 12 (20h)

Keskivaikean vaatimuksien toteuttaminen. Pelin tallentamisen ja lataamisen mahdollistavien ominaisuuksien toteuttaminen.

Viikko 13 (20h)

Pelin vaativan vaikeustason keskeisen algoritmin kehittäminen ja toteuttaminen.

Viikko 14 (10h)

Pelin testien luominen ja niiden avulla vikojen degubbaaminen.

Viikko 15 (4h)

Pelin degubbaaminen, testaaminen ja palauttaminen.

6 Yksikkötestaussuunnitelma

Pelin keskeisimpiä ominaisuuksia ovat ratkaistavissa olevan pelitilanteen luominen satunnaisesti, ratkaisun oikeaoppinen ja satunnainen sekoittaminen ja pelin algoritminen ratkaiseminen. Ensin täytyy testata algoritminen ratkaisu antamalla varmasti ratkaistavissa olevan ja oikein sekoitetun pelitilanteen. Vasta kun se vaihe toimii varmasti, niin ohjelma voidaan testata ratkaistavissa olevalla pelitilanteella ilman sekoittamista. Tämän jälkeen voidaan testata koko paketti putkeen: ratkaisun luominen, pelitilanteen sekoittaminen ja pelin ratkaiseminen.

Game-olion alku- ja lopputilannetta täytyy verrata pelin tallentamisen ja lataamisen jälkeen. Käyttäjäsyötettä on syytä myös testata. Käyttäjältä täytyy estää laittomat siirrot, kuten pelipalojen pinoaminen ja pelipalojen pelialueen ulkopuolelle vieminen. Niiden sattuessa täytyy myös tulostua oikeat virheilmoitukset käyttäjälle. Luonnollisesti, täytyy myös testata, että pelin aikana pelipalojen määrä pysyy vakiona.

Viitteet

[1] PyGame dokumentaatio, https://www.pygame.org/docs

[2] Pandas dokumentaatio, https://pandas.pydata.org