Aalto-yliopisto

CS-A1121 - Ohjelmoinnin peruskurssi Y2

Kolmiopalapeli: Dokumentaatio

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

4. huhtikuuta 2019

1 Yleiskuvaus

Toteutin kolmiopalapelin, jossa kolmion muotoiset palat on sijoitettava pelilaudalle niin, että jokaisten vierekkäisten kolmioiden yhteisellä sivulla on sama väri. Myös pelilaudan reunoilla on oma väri, joten niiden vieressä olevan kolmion sivu on valittava sen mukaan. Käytössä on 12 väriä, jotka ovat lainattu ColorBrewer paleteista [1].

Toteutin projektin vaativien vaatimuksien mukaan, jotka olivat siis:

- Graafinen käyttöliittymä. Ohjelma arpoo aluksi ratkaisun. Ratkaisussa ei saa olla kahta samanlaista kolmiota (ota huomioon, että kolmioita voidaan pyörittää). Kelpaavan ratkaisun löydyttyä ohjelma purkaa kolmiot pinoon tai pöydälle näkyville ja sekoittaa niiden järjestyksen ja suunnan.
- Kolmioita voi sekä sijoittaa laudalle että poistaa siltä. Keksi itse, miten koordinaatit annetaan kätevästi.
- Kolmioita tulee voida pyörittää (laudalla saa ratkaisuvaiheessa olla ristiriitaisia sijoituksia).
- Ohjelman tulee havaita automaattisesti onnistunut ratkaisu.
- Pelin tallennus ja lataus (tässä voi käyttää tietysti omaa formaattia)
- Näennäisen älykäs algoritmi, jolla tietokone voi yrittää pelata pelin loppuun pelaajan näin halutessa. Algoritmin tulee kyetä ratkaisemaan peli ilman esitietoja ja pelaajan apua. Algoritmi ei siis esimerkiksi saa hyödyntää ohjelman alussa arvottua ratkaisua omassa ratkaisussaan.

2 Käyttöohje

Ohjelma on kirjoitettu Python 3.7:lla ja se hyödyntää PyGame-grafiikkakirjastoa [2]. Ohjelma käynnistetään ajamalla main.py. Jos peli käynnistetään ekaa kertaa, niin se luo uuden pelilaudan. Jos peliä on pelattu aiemmin, niin se jatkuu siitä mihin viimeksi jäi.

Pelaaja voi vuorovaikuttaa ohjelman kanssa oikealla hiirinäppäimellä ja A/D näppäimillä. Hiirellä pelaaja voi valita halutun kolmiopalan ja sijoittaa sen tyhjälle paikalle. A/D näppäimillä pyöritetään kolmiopaloja vastapäivään/myötäpäivään.

Pelaaja voi myös painaa kolmea peliruudun nappia: Check solution, Solve ja New game. Check Solution tarkistaa pelilaudan ja kertoo, että onko ratkaisu valiidi vai ei. Solve ratkaisee pelaajan puolesta pelilaudan. New game generoi uuden pelilaudan.

3 Ulkoiset kirjastot

Ohjelma hyödyntää PyGame-grafiikkakirjastoa [2]. Sen avulla piirretään grafiikka ja luetaan käyttäjäsyöte.

4 Ohjelman rakenne

Ohjelman grafiikka luodaan PyGame:lla. PyGame tarjoaa kaikki tarvittavat luokat graafisella puolella. Käyttöliittymän elementtejä kutsutaan piirrettäväksi ainoastaan pelisilmukan sisältä. Ohjelman sisäinen luokkarakenne:

- Game – Pelikokonaisuus, joka pitää kirjaa pelilaudasta ja pelipaloista. Olio luodaan kerran pelin alussa. Varsinainen pelisilmukka main.py:ssä vuorovaikuttaa vain tämän luokan kanssa. Game-olio vuorovaikuttaa kaikkien muiden luokkien kanssa.
  - gen_board: luo satunnaislukugenraattorin avulla pelilaudan
  - switch: vaihtaa pelinkentän kahden elementin paikkaa
  - solve: ratkaisee pelilaudan dynaamisen algoritmin avulla
  - is_valid: tarkistaa, onko pelilaudalla oleva ratkaisu valiidi
  - shuffle: sekoittaa luodun pelilaudan pelaajaa varten ja siirtää pelipalat hyllylle
- Element – Yläluokka kolmelle luokalle: Blank, Wall, Piece. Pitää kirjaa sijainnista.
- Blank – Edustaa tyhjää paikkaa pelilaudalla. Sen paikalle voisi sijoittaa pelipalan.
  - draw: piirtää itsensä ruudulle tietyllä tavalla, riippuen sijainnista
- Wall – Kuvaa pelikentän staattista seinää.
  - draw: piirtää itsensä ruudulle tietyn värisenä
- Piece – Kuvaa pelikentän pelipalaa.
  - rotate_right: kiertää pelipalan myötäpäivää
  - rotate_left: kiertää pelipalan vastapäivää
  - draw: piirtää itsensä ruudulle ja värittää jokaisen sivun tietyn väriseksi

Käytössä ovat myös res.py, dynamics.py ja test.py. res.py on saatavilla kaikille luokille ja se sisältää kaikki pelin staattiset koordinaatit ja värit. dynamics.py sisältää pelilaudan luku/kirjoittamisen funktiot, pelilaudan ratkaisemisessa käytetyn algoritmin ja debuggaustyökalun. test.py hyödyntää kaikkia luokkia ja on puhtaasti sisäisen pelilogiikan testaamista varten.

Luokkajaon olisi voinut tehdä monella eri tavalla. Koska ohjelma ei ole kovin laaja, yksinkertainen luokkarakenne on riittävä ja se on helposti luettava. res.py oli hyvä valinta, sillä melkein kaikki luokat hyödyntävät jollain tavalla pelin staattisia ominaisuuksia, joten niitä ei ole järkeä listata jokaiselle luokalle erikseen. dynamics.py oli tarpeen luettavuuden edistämiseksi. Kaikki funktiot olisi voinut sisällyttää Game-olion tiedostoon, mutta siitä olisi tullut todella pitkä. Lisäksi, dynamics.py funktiot edustavan omia kokonaisuuksia, joten on järkeä pitää ne myös omassa tiedostossa.

5 Algoritmit

- Peli luodaan valmiiksi pelilaudalle, jonka jälkeen palat sekoitetaan pelaajaa varten. Jokaisen palan uusi orientaatio ja sijainti hyllyllä valitaan satunnaisesti. Jokaiselle pelilaudan palalle algoritmi arpoo yhden kuudesta orientaatioista. Jokaiselle palalle algoritmi arpoo myös sijainnin hyllyltä. Algoritmilla on tiedossa hyllyn varatut paikat, joten törmäyksiä ei voi sattua. Satunnaisuuden takaa satunnaislukugeneraattorin siemen, joka on pelihetken kellonaika. Sekoituksen voisi toteuttaa myös systemaattisesti, mutta se johtaisi huonoon pelikokemukseen, sillä palat sekotettaisi joka kerta samalla tavalla.

- Kun pelaaja painaa Solve nappia, ohjelma ratkaisee pelin ja näyttää pelaajalle lopputuloksen. Algoritmi ratkaisee ongelman syvyyshaulla ja dynaamisella optimoinnilla. Ensin se rakentaa matriisin M, jossa jokainen sarake vastaa yhtä pelipalaa ja jokainen rivi vastaa yhden pelipalan kaikkia mahdollisia orientaatiota. Matriisin koko on täten 26×6.

  Algoritmi käy ylhäältä alas ja vasemmalta oikealla jokaisen vapaan paikan pelilaudalla läpi. Jokaisen tyhjän paikan kohdalla se luo listan l kaikista mahdollista paloista ja orientaatioista, jotka sopisivat värien puolesta siihen. Algoritmi valitsee ensimmäisen alkion listalta, poistaa sitä pelipalaa vastaavan rivivektorin matriisista M (samaa palaa ei voi käyttää uudestaan) ja etenee seuraavalle vapaalle paikalle. Jos jossain vaiheessa l on tyhjä ja pelilauta ei ole vielä täynnä, niin jossain kohtaan on tehty väärä valinta. Algoritmi palaa rekursiossa taaksepäin ja valitsee toisen alkion listalta l kun se on mahdollista. Jos l on tyhjä ja pelilaudassa ei ole enää vapaita paikkoja, niin ratkaisu on löydetty ja se palautetaan rekursiossa taaksepäin.

  Algoritmi voisi toimia myös eri tavalla. Kaikki mahdolliset kombinaatiot voisi käydä läpi, kunnes valiidi ratkaisu on löydetty. Pelilaudalla on 6.8794×10^{46} mahdollista kombinaatiota, joten on turvallista sanoa, että tämä lähestymistapa ei johtaisi ratkaisuun pöytätietokoneella. Toteutettu algoritmi on tehokas ja nopea millä tahansa tietokoneella.

  Algoritmin ajonopeuteen vaikuttavat pelilaudan koko ja käytettyjen värien lukumäärä. Mitä suurempi pelilauta, sitä pidempi ajoaika. Mitä vähemmän värejä käytetään,

sitä nopeammin algoritmi löytää ratkaisun, sillä mahdollisten ratkaisujen määrä on valtava. Jos värejä on paljon, niin lista l on joka kerta pieni, sillä pelipaloja on paljon erilaisia. Sillä l on pieni, niin ratkaisu löydetään nopeasti ja se on usein myös uniikki. Kokemuksen mukaan, jos käytettyjä värejä on vähemmän kuin 4 tai enemmän kuin 10, niin ajoaika on keskimäärin 0.1 sekunttia. Kun värien määrä on väliltä 4 ja 10, niin ajoajat voivat olla jopa 5 minuutin luokkaa. Tämä on algoritmin heikkous, mutta koska tässä pelissä käytetään 12 eri väriä, niin algoritmi on nopea eikä aiheuta viivettä pelikokemuksessa.

6 Tietorakenteet

Ohjelma hyödyntää listoja ja monikkoja. Listat ovat muuttuvia ja monikot muuttumattomia. Listoja käytetään pääosin pitämään kirjaa pelilaudalla sijaitsevista elementeistä. Listat soveltuvat tähän tehtävään, sillä pelikenttää pitää pystyä muuttamaan helposti.

Monikkoja hyödynnetään pitämään kirjaa pelin ja pelikentän staattisista ominaisuuksista kuten koordinaateista ja väreistä rgb-muodossa. Monikkoja on käytetty kun on haluttu säilyttää informaatiota mitä ei tarvitse ohjelman aikana muuttaa. Ratkaisemiseen käytetty algoritmi hyödyntää sekä listoja että monikkoja.

Sanakirjoja olisi voitu käyttää listojen ja monikkosen sijasta/lisäksi, mutta se olisi hidastanut ohjelmaa tuomatta mitään lisäarvoa.

7 Tiedostot

Ohjelma hyödyntää yhtä ulkoista board.txt tiedostoa. Kun ohjelma käynnistetään, niin tiedostosta yritetään lukea vanhaa pelilautaa. Jos se ei onnistu, niin luodaan uusi pelilauta. Kun ohjelma suljetaan, niin pelilauta tallenetaan tähän tiedostoon. board.txt on myös helposti muokattavissa ja luettavissa, sillä se sisältää rivi riviltä kaikki pelilaudan elementit sekä niiden ominaisuudet.

Myös .csv tiedostoa olisi voitu käyttää, mutta sen käyttö ei olisi tuonut mitään lisäarvoa.

Esimerkit ja selitykset tiedoston riveistä:

(nimi;x-koordinaatti;y-koordinaatti)
blank;100;500

(nimi;x-koordinaatti;y-koordinaatti;ylä-väri;vasen-väri;ala-väri;oikea-väri)
wall;100;600;(1, 1, 1);(255, 187, 120);(1, 1, 1);(1, 1, 1);(255, 187, 120)

(nimi;x-koordinaatti;y-koordinaatti;ylä-väri;vasen-väri;ala-väri;oikea-väri;orientaatio)
piece;100;1000;(44, 160, 44);(23, 190, 207);(255, 0, 0);(227, 119, 194);0

Elementeillä on neljä rgb-muodossa olevaa väriä neljälle suunnalle. Sekä seinäpaloilla että pelipaloilla on neljä suuntaa sisäisen ohjelmointilogiikan helpottamiseksi, eikä niitä kaikkia välttämättä piirretä näkyviin. Orientaatio on binäärimuodossa: 1 - ylös osoittava kolmio, 0 - alas osoittava kolmio.

8 Testaus

Ohjelman graafista puolta ja käyttäjäsyötteeen toimivuutta testattiin pelaamalla pelia. Lopullisessa versiossa grafiikka toimi niin kuin pitää ja käyttäjäsyötteellä ei saanut ohjelmaa kaadettua.

Yksikkötestausta hyödynnettiin kehittämisen loppuvaiheessa. Kun ohjelman rakenne alkoi olemaan selvä, niin yksikkötestit oli järkevä kirjoittaa. Ne löytyvät test.py tiedostosta. Testien rakenne:

- test_is_valid: tunnistaako ohjelma onnistuneesti oikean ja väärän ratkaisun.
- test_dump_read: onko pelilauta identtinen sen jälkeen kun se on tallennettu ja luettu ulkoisesta tiedostosta.
- test_gen_board: luodaanko pelilauta oikein, onko arvottu pelilauta valiidi ja onko pelipaloja oikea määrä oikeissa paikoissa.
- test_algothm_solve: ratkaiseeko algoritmi pelin oikein. Tässä hyväksytään kaikki valiidit ratkaisut vaikka ne eivät olisi identtisiä alkuperäisen ratkaisun kanssa (useampi oikea ratkaisu mahdollisesti olemassa).

Ohjelma läpäisee kaikki esitetyt testit. Ohjelman testaus vastasi ja jopa ylitti suunnitelmassa esitettyä. Testien kattavuus on PyCharmin mukaan 81%, joista puuttuvat 19% ovat pääosin grafiikan alustamista ja piirtämistä, mitä testattiin käsin.

9 Ohjelman tunnetut puutteet ja viat

Ohjelmassa ei ole tunnettuja puutteita tai vikoja. Käytetty algoritmi on nopea, mutta pelin satunnaisuuden vuoksi kokeellisella tasolla on esiintynyt anomalioita, joissa algoritmilla kestää 0.1 sekunnin sijaan noin kaksi sekunttia.

10 3 parasta ja 3 heikointa kohtaa

3 parasta kohtaa: pelin visuaalinen ilme, pelin vaikeustaso ja pelin satunnaisuus. Pelin visuaalinen ilme on minimalistinen ja käytetyt värit ovat mietitty väriopin kannalta tarkkaan (Kuva 1, 2). Ne ovat pääosin peräisin ColorBrewer paleteista. Käytettyjä värejä on 12, jotta pelin algoritmi olisi tarpeeksi vakaa ja nopea. Peli on vaikea, sillä pelaajan on tehtävä paljon arvauksia ja muistaa valintojen järjestys. Pelin on tarkoituksella vaikea ja se palkitsevaa kun sen saa pelattua loppuun. Pelin satunnaisuus on laadukas, eikä anna pelaajan kyllästyä värimaailmaan tai sekoitusjärjestykseen.

3 heikointa kohtaa: käytetty grafiikkakirjasto, pelikokemuksessa olisi vielä hiottavaa, algoritmin ajoaika. Käytetty PyGame-grafiikkakirjasto on kankea, eikä tarjoa helppoja työkaluja piirtää muotoja. Tämän vuoksi grafiikka on usein pikselöityä ja heikentää pelin visuaalista ilmettä. Pelikokemuksessa olisi vielä hiottavaa. Pelaajalle voisi esimerkiksi näyttää, minkä pelipalojen takia ratkaisu ei ole valiidi. Pelaajalle voisi myös antaa mahdollisuuden valita vaikeustaso, minkä mukaan määräytyisi pelikentän koko ja/tai käytettyjen värien määrää. Nykyisellä pelikentän koolla ja käytettyjen värien määrällä algoritmin on suhteellisen vakaa ja useimmiten todella nopea. Anomalioita voi teorian tasolla kuitenkin sattua, jolloin ajoaika olisi kohtuuttoman pitkä pelikokemuksen näkökulmasta. Algoritmia voisi siis nopeuttaa entisestään loogisella ja teknisellä tasolla. Esimerkiksi usein käytetty deepcopy-funktio on hidas ja vie keskimäärin 65% ajoajasta PyCharmin mukaan.

11 Poikkeamat suunnitelmasta

Pelikentän topologiaan tuli kehityksen aikana muutoksia. Pelikentän kokoa ensin kasvatettiin pelikokemuksen edistämiseksi, jonka jälkeen sitä taas pienennettin pelin helpottamiseksi ja algoritmin nopeuttamiseksi. Pelin helpottamisen vuoksi pelilaudan keskelle sijoitettiin kaksi staattista oikein sijoitettua pelipalaa. Pelilaudalta poistettiin pelipalojen pyörittämisen mahdollisuus, sillä se voisi hämmentää pelaajia ja johtaa heidät harhaan. Pelaajan on tarkoitus pyörittää pelipaloja hyllyllä ennen pelilaudalle sijoittamista.

Työjärjestys toteutui suunnitelman mukaan, mutta aikataulun puolesta edettiin huomattavasti nopeammin kuin mitä oli suunniteltu.

Viikko 11 (10h)

Graafisen ilmeen ja luokkajaon suunnittelu.

Viikko 12 (30h)

Ensin keskivaikean ja sitten vaativa tason vaatimuksien toteuttaminen siinä järjestyksessä, mikä on tehtävänannossa esitetty. Testien kirjoittaminen silloin kun se oli järkevä tehdä.

Viikko 13 (8h)

Dokumentaation kirjoittaminen, koodin siivoaminen ja kommentointi.

Viikko 14 (1h)

Ohjelman koodin ja dokumentaation tarkistaminen ja palauttaminen.

12 Arvio lopputuloksesta

Ohjelma on vakaa ja ehyt kokonaisuus. Se täyttää kaikki vaativan tason vaatimukset. Sen lisäksi ohjelmalla on hyvä visuaalinen ilme ja se on mielyttävä pelikokemus. Ohjelman voisi parantaa antamalla pelaajalle mahdollisuus valita vaikeustaso, minkä mukaan määräytyisi pelikentän koko ja/tai käytettyjen värien määrää. Siinä tapauksessa ratkaisemiseen ei käytettäisi algoritmia mahdollisesti huonon ajoajansa vuoksi, vaan ratkaisu tallennettaisi luomishetkellä ulkoiseen tiedostoon.

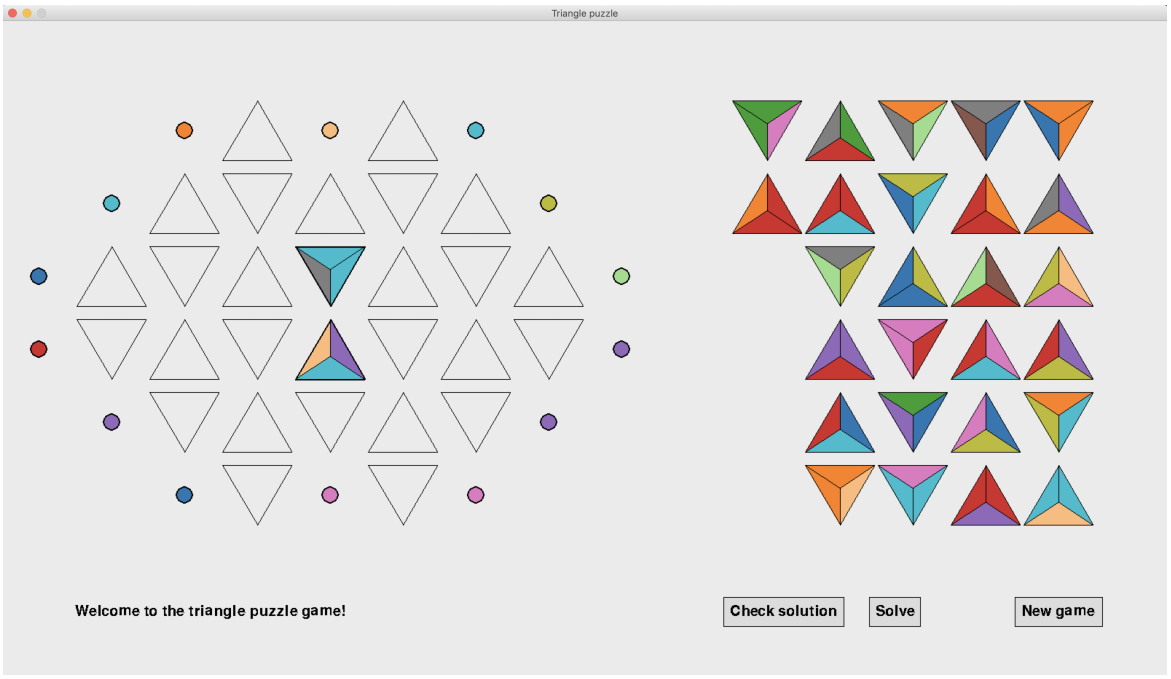
Ohjelman ratkaisumenetelmät, tietorakenteet tai luokkajako ovat perusteltuja ja onnistuneesti toteutettuja. Ohjelma on kirjoitettu modulaariseksi, joten se on helposti laajennettavissa ja skaalattavissa.

Viitteet

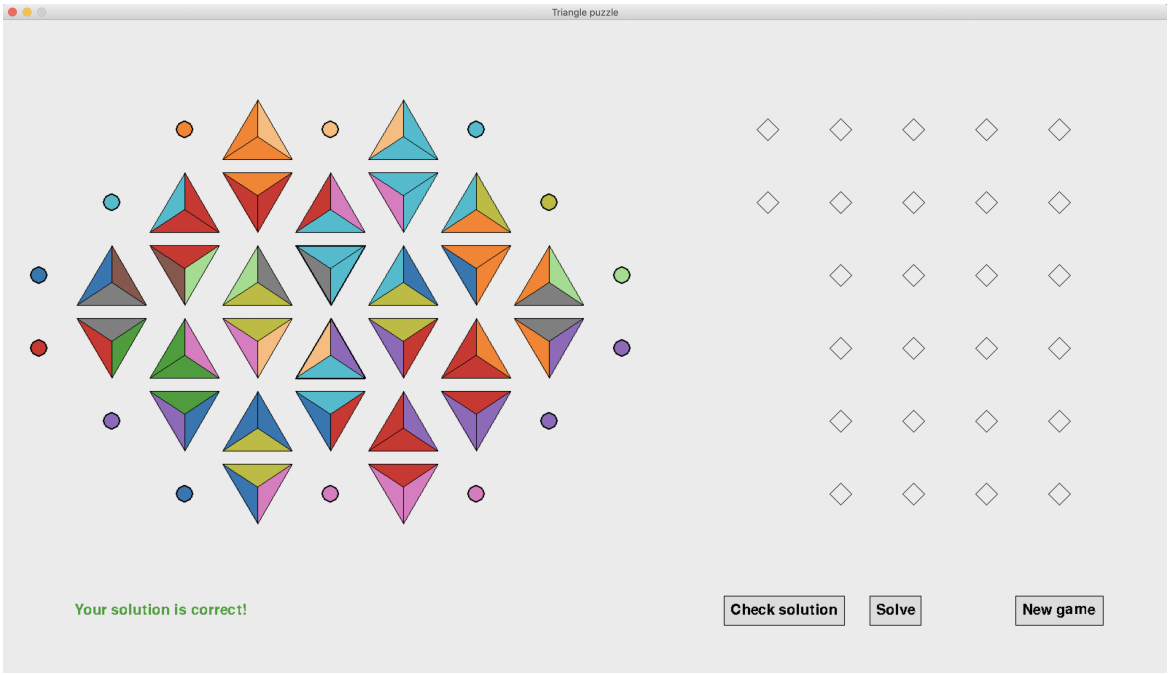
[1] ColorBrewer, https://observablehq.com/@d3/color-schemes

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

A Kuvakaappaukset



Kuva 1: Pelin alkunäkymä.



Kuva 2: Algoritmin ratkaisema peli.