Maria Tudose

427780

Tietotekniikka 4vk

26.04.2018

## Kolmiopalapeli – Dokumentaatio

**Yleiskuvaus**

Olen tehnyt graafisen ohjelman, jossa voi ratkoa kolmiopalapelejä. Palapelissä kolmion muotoiset palat sijoitetaan kuusikulmion muotoiselle laudalle niin, että yhteensopivat kuviot ovat kolmioiden yhteisellä laidalla. Ohjelma arpoo ratkaisun ja asettaa uniikit palat laudalle sekalaisesti, sekoittaen niiden järjestyksen ja suunnan. Käyttäjä voi pyörittää ja liikuttaa paloja laudalla, sekä tallentaa ja ladata pelin. Kun kaikki palat ovat asetettu oikein, ohjelma tunnistaa ratkaisun ja onnittelee siitä. Projekti on toteutettu keskivaikealla tasolla.

Tärkeä esitieto dokumentaation lukemiseen: pelilaudan koordinaateista puhuttaessa on kyse 0,0 – 4,7 välistä eli itse sijainneista pelilaudalla. Peliruudun koordinaatit taas tarkoittavat palojen sijainnista koko graafisella käyttöliittymällä.

**Käyttöohje**

Opastus ohjelman käyttöön: miten ohjelma käynnistetään? Mitä sillä voi tehdä? Mitä komentoja käyttäjällä on valittavanaan? Jne.

Ohjelma käynnistetään ajamalla TriPuzzle.scala tiedosto ja pelin voi aloittaa heti. Palat ovat valmiiksi luotu ja sekoitettu ja käyttäjä voi liikuttaa niitä vasemmalla hiiren näppäimellä ja pyörittää oikealla näppäimellä. Käyttäjä voi asettaa paloja peliruudukolle sekä tallentaa ja ladata pelin yläpalkin menusta.

**Ohjelman rakenne**

Ohjelmassa on luokat TriPuzzle, Board ja Piece. TriPuzzle luokassa luodaan graafinen liittymä ja hoidetaan pelin aloitus, tallennus ja lataus. TriPuzzlen startGame metodi käynnistää koko prosessin: se luo uuden pelilaudan ja sen palat, sekä piirtää koko liittymän kutsuen drawPieces ja drawMenu metodeita. drawPieces hoitaa palojen ja laudan piirtämisen ja samoin niiden reagoinnit hiiren painalluksille. drawPieces kutsuu Piece luokan setPos ja setCoord metodeita, jotka asettavat palan paikan pelilaudalla ja koordinaatit ruudulla.

Board luokka hoitaa palojen luomisen createPieces metodilla ja pitää tiedot palojen sijainnista pelilaudalla. Toinen Board luokan keskeinen metodi on checkIfSolved, jossa käydään asetettuja paloja läpi ja tarkistetaan vastaavatko naapuripalojen sivut toisiaan. Piece luokka hoitaa kaikki palakohtaiset toiminnot. neighborFound tarkistaa onko naapuripalaa olemassa siihen suuntaan mitä tutkitaan ja createSides käyttää tätä hyväksi luodessaan palan sivut. Board myös käyttää neighborFound metodia checkIfSolved metodissa, jotta ei mentäisi laudan laidoista yli.

**Algoritmit**

Tässä ohjelmassa on vain muutama algoritmi. Palojen luonti on tehty createPieces metodissa, jossa luodaan palat aloittaen keskeltä taulukkoa ja lähdetään spiraalimaisesti luomaan palat sisältä ulospäin. Näin saadaan helposti luotua palat joiden sivut vastaavat toisiaan siten että uusin pala vain katsoo ympäröivien naapureiden sivuja ja sen omat sivut luodaan vastaamaan naapureiden sivuja. Tämä toiminto näkyy createSides metodissa jota createPieces käyttää.

Oikean ratkaisun tarkistus on tehty tallentamalla kaikki palat ja niiden asemat pelilaudalla sanakirjaan, josta käydään jokainen pala läpi ja tarkistetaan sen naapurit. Tämä on yksinkertaisin tapa saada selville onko palapeli ratkaistu oikein

Palojen napsahduksessa paikoilleen pelilaudalle on käytetty ruudukon palan koordinaattia hyväksi ja palan koordinaatiksi asetetaan erotus liikutettavan palan ja ruudukon palan koordinaateille.

**Tietorakenteet**

Käytin paljon muuttuvatilaisia kokoelmatyyppejä, sillä ohjelmassa on paljon muuttuvaa dataa. Käytin useassa kohdassa muuttuvatilaisia sanakirjoja palojen ja niiden kordinaattien tallennukseen. Tästä oli hyötyä palojen paikan tutkimiseen ja niiden läpikäymiseen. Palojen piirtämisessä käytin muutamaa ArrayBuffer tietorakennetta, jotta saisin talteen piirrettyjen kuvioiden tietoja myöhempää käyttöä varten.

**Tiedostot**

Ohjelmani käsittelee pelkästään yhtä tiedostoa, tallennustiedostoa. Tiedosto on tekstitiedosto nimeltä save.txt ja siihen on tallennettu palojen keskeisimmät tiedot. Eri palat on erotettu puolipisteellä ja palojen sisäiset tiedot pilkulla. Palan tiedot tallennetaan seuraavasti: alkuperäiset koordinaatit pelilaudalla, koordinaatit peliruudulla, tämänhetkinen sijainti pelilaudalla ja palan kirjaimet. Esim. ”3,0,187.5,50.0,5,0,aAC;” pala on alun perin luotu laudalle kohtaan ”3,0”, sen nykyinen sijainti peliruudulla on ”187.5,50.0” ja se on asetettu pelilaudalle kohtaan ”5,0”, sekä palan kirjaimet ovat ”aAC”.

**Testaus**

Testasin ohjelmaa rakentaessa printlineilla ja asettamalla erilaisia arvoja itse koodissa, jotka myöhemmin poistin. Tein vasta ohjelman lopuksi kunnolliset testit, jotka jäivät mielestäni hieman puutteelliseksi. Minulle tuli loppupuolella vastaan ongelma TriPuzzlen metodien testaamisessa, sillä ne vaativat Pane luokan children muuttujaa toimiakseen. TriPuzzlea olisi ollut mielestäni tärkeä testata perusteellisesti ja erityisesti loadGame metodia. Luodut testit testaavat Piece ja Board luokkien metodeita ja pääsevät kaikki läpi. Jos tarkoituksella rikon ohjelman niin ne epäonnistuvat.

**Ohjelman tunnetut puutteet ja viat**

Tunnetut viat ovat pelin latauksessa tapahtuva pieni palojen liike ruudussa, jota olisin lisäajalla korjannut tutkimalla tarkemmin kuinka palan koordinaatit asetetaan. Palat voivat myös mennä reunoista yli mikä oli alkuperäisessä suunnitelmassa tarkoitus estää. Tämän korjaaminen vaatisi hienosäätöä ja koordinaattien tutkimista.

**3 parasta ja 3 heikointa kohtaa**

Parhaat 3 kohtaa ohjelmassani on mielestäni se kuinka hyvin palat napsahtavat paikalle peliruutuun, kuinka uudet palat luodaan ja asetetaan kentälle, sekä kuinka tarkistan onko ratkaisu oikea. Näitä on avattu lisää kohdissa ”Ohjelman rakenne” ja ”Algoritmit”

Suurin heikko kohta on mielestäni TriPuzzlen epäselvä rakenne, jota yritin korjata asettelemalla käyttöliittymän piirtämisen metodeihin, mutta tuntuu vieläkin sotkuiselta. Toinen heikko kohta on kun pelin tallentaa ja lataa uusiksi niin palat ovat liikkuneet hieman alkuperäisistä asennoista, mitä en saanut millään korjattua. Kolmas heikko kohta on kuinka otan tiedot ladatusta tiedostosta. Ajan loppumisen takia en keksinyt parempaa tapaa.

**Poikkeamat suunnitelmasta**

Toteutin melkein kaiken suunnitelman mukaisesti lukuunottamatta palojen rajoitusta kentän sisäpuolelle (mikä osoittautui yllättävän vaikeaksi palojen muodon takia) ja palapelin ratkaisun luomisen, jolle löytyikin yksinkertaisempi ratkaisu kuin mitä olin suunnitelmassa ajatellut. Ajankäyttöarviosta ja toteutusjärjestyksestä lisää seuraavassa osiossa.

**Toteutunut työjärjestys ja aikataulu**

Osa-aikaisen työn takia aloitin projektin vasta kun muiden kurssien tentit olivat ohi eli noin 3 viikkoa sitten. Loin ensiksi suunnitellut luokat ja niiden metodit pääpiirteisesti. Jatkoin luomalla graafisen käyttöliittymän ja kehittämällä pelilaudan määrittelyä ja palojen asettelua siihen. Seuraavaksi tein palojen napsahduksen peliruudukolle ja niiden pyörimisen, jonka jälkeen tein pelin tallennuksen ja latauksen. Lopuksi tein testit ohjelmalle. Poikkesin suunnitelmasta siinä, että tein pelin tallennuksen ja latauksen sekä testit vasta lopuksi.

**Arvio lopputuloksesta**

Mielestäni ohjelmani laatu on hyvä. Graafinen liittymä on siisti ja selkeä ja koodissa on hyvä jako. Kommentointia voisi olla hieman enemmän ja palojen asetusta olisi voinut hioa lisää pelin latauksessa. Ohjelmaa voisi parantaa esim. käyttämällä kuvia kirjainten sijaan ja antamalla mahdollisuuden pyörittää paloja myös eri asentoihin.

**Viitteet**

Scalafx dokumentaation puutteellisuuden takia käytin paljon javafx dokumentaatiota: <https://docs.oracle.com/javase/8/javafx/api/toc.htm>

Otin mallia erilaisista tutoriaaleista javafx käytöstä:

<https://www.tutorialspoint.com/javafx/2dshapes_polygon.htm>

<https://docs.oracle.com/javafx/2/canvas/jfxpub-canvas.htm>

<https://docs.oracle.com/javafx/2/ui_controls/menu_controls.htm>