Eerik Kukkonen

Verkkojen piirtäminen Silverlightilla

Tietotekniikan
kandidaatin tutkielma
29. maaliskuuta 2012

Jyväskylän yliopisto

Tietotekniikan laitos

Jyväskylä

Tekijä: Eerik Kukkonen
Yhteystiedot: eerik.kukkonen@jyu.fi
Työn nimi: Verkkojen piirtäminen Silverlightilla
Title in English: Graph drawing with Silverlight
Työ: Tietotekniikan kandidaatin tutkielma
Sivumäärä: 25
Tiivistelmä: Silverlight-teknologialla voidaan tehdä mm. näyttäviä web- ja mobiilisovelluksia. Tutkielmassa selvitetään Silverlightin soveltuvuutta verkkojen piirto-ohjelman tekemiseen. Tutkielmassa ohjelmoitiin piirto-ohjelma, joka havainnollistaa ratsun polun etsintää shakkilaudalla. Ohjelman kehitysprosessin aikana huomattiin, että Silverlight soveltuu tutkielman mittaluokan verkoille. Silverlight-sovellus saattaa kuitenkin hidastua, jos sovellukseen tuodaan reaaliaikaisia tapahtumia vaativia tai pelimäisiä elementtejä. Tällöin voi olla järkevää vaihtaa XNA-kehitykseen Windows-ympäristössä.
English abstract: With Silverlight technology you can create engaging web and mobile applications among other things. In this study the suitability of Silverlight for creating a graph drawing program is measured. In the study a graph drawing program was programmed that presents the search of a knight's tour on the chess board. During the development process of the program it was noticed that Silverlight is suited for the graphs of the scale used in the study. Although a Silverlight application might get slowed down if real time event based or gamelike elements is brought in to the application. In that case it might be reasonable to change to XNA development in the Windows environment.
Avainsanat: Silverlight, Verkot, Verkkoteoria, Ratsun polku
Keywords: Silverlight, Graphs, Graph theory, Knight's tour

Sisältö

1 Johdanto

Silverlight on teknologia, joka mahdollistaa mm. näyttävien web- ja mobiilisovellusten tekemisen. Tutkielma vastaa kysymykseen, voiko Silverlight-teknologialla tehdä web-selaimessa toimivan verkkojen piirto-ohjelman ja voidaanko tätä sovellusta käyttää shakin ratsun polun havainnollistamiseen. Tutkielmassa ohjelmoidaan verkkojen piirto-ohjelma ja kaksi ratsun polkua etsivää algoritmia. Piirto-ohjelman kehittämisprosessia arvioidaan ohjelmoinnin ja teknologian kannalta.

Tutkielmassa käydään ensin läpi verkkoteorian käsitteitä. Verkkoteorian ohella esitetään muutamia algoritmeja ratsun polun etsimiseen. Tämän jälkeen kerrotaan Silverlight-teknologiasta. Tutkielman keskivaiheilla kerrotaan ohjeita ohjelmoidun piirto-ohjelman käyttöön ja ohjelman toiminnasta ohjelmakoodin tasolla. Lopuksi arvioidaan piirto-ohjelman kehittämisprosessia.

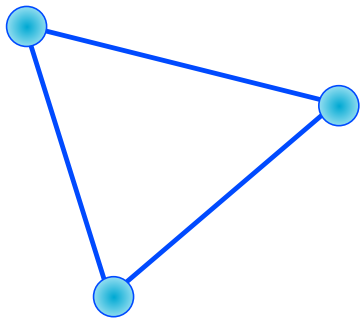
2 Verkkoteoriaa

Verkkoteorian peruskäsitteet ovat helppotajuisia, ja tästä syystä verkoilla voidaan esittää asioiden välisiä suhteita selkeästi [15, s.1-3]. Tässä tutkielmassa mainitaan muutamia peruskäsitteitä, mutta enemmän asiasta kiinnostuneille suosittelen verkkoteorian kurssin käymistä tai tutustumista aiheesta löytyvään kirjallisuuteen. Verkkoteorian käsitteitä on paljon, ja monille asioille ei ole vakiintuneita termejä, vaan samaa asiaa saatetaan kutsua monella nimellä. Verkkoteoriaa voidaan soveltaa erittäin moniin kohteisiin [17]. Tällaisia voivat olla esim. sosiaaliset verkot, verkot kaupungeista (kauppamatkustajan ongelma), verkko luokkien välisistä riippuvuuksista ohjelmakoodissa (kts. kuva 4.3) tai tiedon louhinnalla koostettu verkko, joka suosittelee tiettyyn oireeseen tiettyä lääkettä [16].

2.1 Käsitteitä

Verkko G koostuu *solmujen* joukosta $V = \{v_1, v_2, ..., v_n\}$ ja *välien* joukosta $E = \{e_1, e_2, ..., e_m\}$. Kukin väli e yhdistää jotkin solmut v_1 ja v_2 ja niitä kutsutaan *naapurisolmuiksi*. Välistä käytetään merkintää $e = \{v_1, v_2\}$ tai pelkästään $e = v_1v_2$. Yleensä verkosta käytetään merkintää $G = (V, E)$ [17, s.1]. Verkolla on aina kaaviokuva [17, s.2], jossa solmut piirretään ympyröinä ja niitä yhdistävät välit piirretään viivoina.

Tasoverkko on verkko, joka voidaan piirtää tasoon niin, ettei yksikään väli leikkaa toista väliä [17, s.109]. Jos verkko ei ole tasoverkko, se on *avaruusverkko*. Avaruusverkkoa ei siis voida piirtää tasoon ilman, että sen välit leikkaisivat toisiaan.



Kuva 2.1: Suuntaamaton verkko

Luuppi tarkoittaa väliä, joka yhdistää solmun itseensä eli välin alku- ja loppusolmu

ovat samoja [17, s.4]. *Silmukka* tai *sykli* on polku, jolla on sama alku- ja loppusolmu [17, s.21]. Silmukan täytyy sisältää vähintään kolme solmua. Jos kahdella tai useammalla välillä on sama alku- ja loppusolmu, niin näitä välejä kutsutaan *rinnakkaisiksi*. Verkko on *yksinkertainen*, jos siinä ei ole luuppeja eikä rinnakkaisia välejä [17, s.4].

Hamiltonin polku on polku, joka kulkee verkon jokaisen solmun kautta ainoastaan kerran [17, s.75]. Kun edellä mainitun polun loppusolmusta päästään takaisin alkusolmuun, kutsutaan polkua *Hamiltonin silmukaksi*.

Suunnatun verkon jokaisella välillä on myös suunta [17, s.3]. Jos yhteenkään väliin ei ole merkitty suuntaa, on kyseessä *suuntaamaton verkko* kuten kuvassa 2.1.

2.2 Ratsun polku ja algoritmit

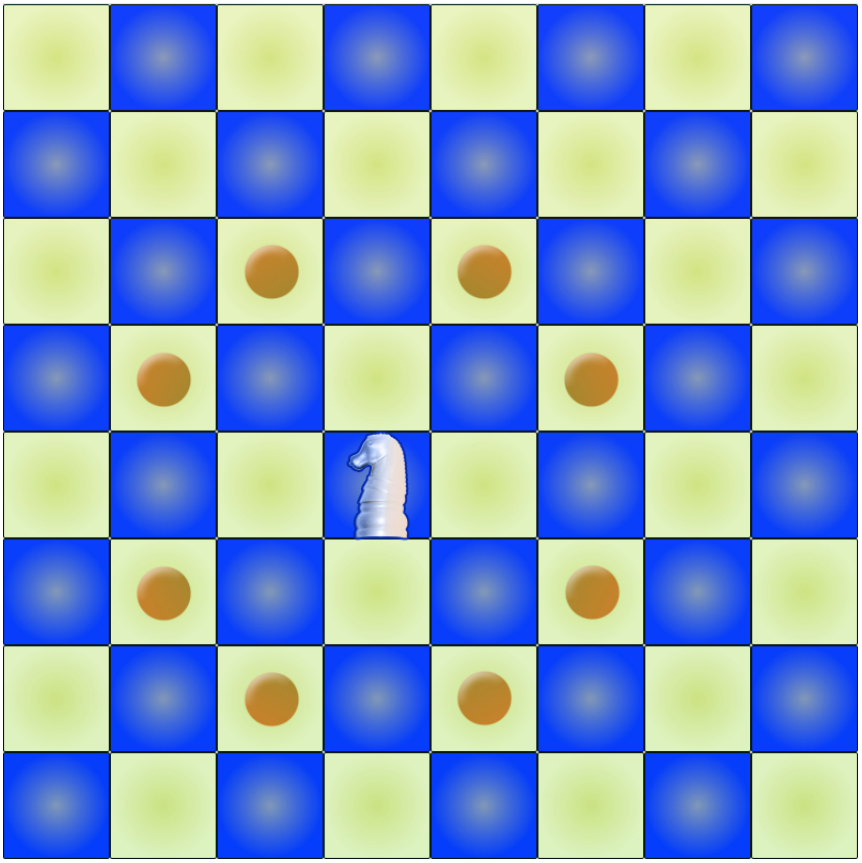
Ratsun polku on hyvin tunnettu ja vanha verkkoteorian ongelma [2, s.22]. Verkkoteorian kannalta ratsun polun etsiminen on Hamiltonin polun etsimisen erikoistapaus [12]. Ratsu on shakin hevosta kuvaava nappula, joka siirtyy kaksi askelta ylös, alas, vasemmalle tai oikealle ja tämän jälkeen yhden askeleen kohtisuoraan. Näin ratsulla on shakkilaudan keskiosassa kahdeksan siirtomahdollisuutta kuten kuvassa 2.2. Ratsun polku on polku, jossa ratsu käy jokaisessa shakkilaudan ruudussa ja käy jokaisessa ruudussa vain kerran. Ratsun polkua sanotaan suljetuksi, jos lähtöruutu ja viimeinen ruutu ovat yhden siirron päässä toisistaan eli polku muodostaa (Hamiltonin) silmukan. Muuten polkua sanotaan *avoimeksi* (eli Hamiltonin poluksi).

2.2.1 Warndorfin heuristiikka

Yleisimmin tunnetuista ratsun polun algoritmeista lienee vuoden 1843 peruutuksen sisältävä syvyyssuntainen haku, jossa käytetään Warndorfin heuristiikkaa. Tässä algoritmissa valitaan ratsun siirroksi se ruutu, jolla on vähiten siirtomahdollisuuksia eteenpäin. Yleisemmin Warndorfin heuristiikkaa voidaan kutsua MRV-heuristiikaksi (Minimum Remaining Values) [11, s.34].

2.2.2 Selbyn algoritmi

Lähteen [3, s.225] mukaan seuraavan algoritmin on kehittänyt alunperin Selby [14] suuntaamattomille verkoille Hamiltonin polun etsimiseen. Algoritmia kutsutaan monipolkumetodiksi [3, s.225]. Algoritmi löytää kaikki verkon Hamiltonin polut. Tässä algoritmissa muodostetaan polkua S_0 ja ohessa muodostuu polkuja $S_1, S_2, ..., S_n$. Algoritmi perustuu useiden polkujen yhtäaikaiseen etsimiseen, missä polut $S_1, S_2, ..., S_n$



Kuva 2.2: Ratsun mahdolliset siirrot

nopeuttavat Hamiltonin polun löytämistä tai sen osoittamista, ettei Hamiltonin polkua ole olemassa, jolloin voidaan tehdä peruutus.

Käydään ensin läpi hieman algoritmin perusideoita. Otetaanpa esimerkin vuoksi käsittelyyn mikä tahansa näiden polkujen keskellä oleva solmu, joka ei ole polun aloitussolmu tai lopetussolmu. Nyt jos tämä solmu on jo yhdistetty polkuun kahdella välillä, niin kaikki muut solmuun tulevat välit voidaan poistaa, koska ollaan etsimässä Hamiltonin polkua. Jos nyt S_0 ei ole ainoa olemassa oleva polku eli Hamiltonin polkua ei ole vielä löydetty, sellaiset välit, jotka yhdistävät jonkin polun aloitussolmun ja saman polun lopetussolmun, voidaan poistaa, koska muuten välit aiheuttaisivat ei haluttuja silmukoita.

Näiden välien poistamisen jälkeen verkkoon G jää jäljelle monia solmuja, joihin tulee vain kaksi väliä. Kaikki nämä keskellä olevat solmut ja välit poistetaan verkosta G ja sen sijaan kutakin polkua merkitään yhdellä välillä. Tämä tehdään kuitenkin niin, että polusta nähdään kaikki solmut, joiden kautta polku kulkee. Tästä saadaan ns. redusoitu verkko $G_k = (V_k, E_k)$, jossa k on indeksi, joka ilmaisee haun tasoa.

Tarkastellaan solmun lisäämistä polkuun S_0. Laajennetaan polkua S_0 niin, että

polun loppuun (tai alkuun) lisätään solmu x_i, johon on väli polun viimeisestä (tai ensimmäisestä) solmusta. Solmun lisäämisen jälkeen tehdään kolme seuraavaa vaihetta:

1. Poistetaan kaikki epäsoveliaat välit verkosta G_k seuraavasti:
   (a) Poistetaan kaikki välit solmusta x_i polun alkusolmuun (tai loppusolmuun).
   (b) Jos x_i on jonkin muun polun S_i aloitussolmu, poistetaan välit polun S_i loppusolmusta polun S_0 aloitussolmuun.
2. Merkitään verkkoa välien poiston jälkeen $G'_k = (V_k, E_k)$.
   (a) Jos johonkin polun keskellä olevaan solmuun tulee kaksi väliä, poistetaan kaikki muut solmuun tulevat välit.
   (b) Jos johonkin solmuun tulee kaksi väliä ja niistä ei ole vielä muodostettu polkua, lisätään uusi polku. Laajennetaan myös kaikkia muita polkuja, jos huomataan, että jonkin polun alkuun tai loppuun voidaaan lisätä vain yksi tietty solmu eli lisättävään solmuun tulee vain yksi väli jonkin polun alku- tai loppusolmusta. Poistetaan kaikki välit, jotka johtavat jonkin polun alkusolmusta saman polun loppusolmuun.

   Toistetaan vaihetta 2 niin kauan kunnes ei voida enää poistaa enempää välejä.
3. Poistetaan verkosta G'_k kaikki polkujen keskellä olevat solmut. Tämä poistaminen tapahtuu kuten keskellä olevien solmujen poistaminen selitettiin aiemmin. Tuloksena saadaan uusi redusoitu verkko G_{k+1}, joka korvaa aiemman verkon G_k.

Jos solmun x_i lisäämisen jälkeen algoritmin vaiheen 2 lopussa johonkin solmuun tulee vain yksi väli, voidaan todeta ettei Hamiltonin polkua löytynyt [3, s.226]. Tällöin solmu x_i hylätään ja toinen solmu x_i valitaan lisättäväksi polkuun S_0. Jos kaikki vaihtoehdot lisättäväksi solmuksi ollaan käyty läpi, tehdään peruutus. Peruutus vaatii, että kohdissa 1 ja 2 poistetuista väleistä tallennetaan tarpeeksi tietoa jokaisella haun tasolla k jotta voidaan muodostaa verkko G_k verkosta G_{k+1} millä tahansa k, kun peruutus tapahtuu.

Jos huomataan että yksi polku sisältää kaikki verkon solmut, voidaan tarkistaa onko Hamiltonin polku löydetty. Jos Hamiltonin polkua ei löydetty, niin voidaan peruuttaa tai jos löydettiin Hamiltonin polku, mutta halutaan löytää useampia, niin peruutetaan.

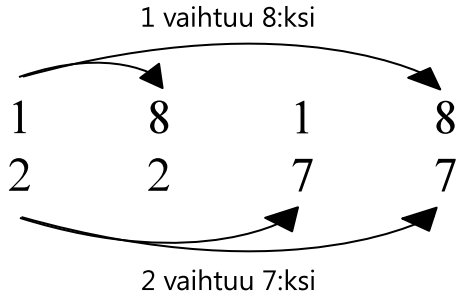
2.2.3 Vandermonden metodi

Vandermonden metodissa yksinkertaistetaan ongelmaa symmetrian avulla [2, s.25]. Yksinkertaistamisesta johtuen tämä algoritmi löytää joitain tietynlaisia ratsun polkuja,

muttei kaikkia. Tässä algoritmissa on tarkoitus järjestää shakkilaudan 64 ruutua $\frac{b}{a}$

$$\begin{array}{cccccccccccccc} 1 & 1 & 1 & 1 & \cdots & 2 & 2 & 2 & \cdots & 3 & 3 & \cdots & 8 \\ 1 & 2 & 3 & 4 & \cdots & 1 & 2 & 3 & \cdots & 1 & 2 & \cdots & 8 \end{array}$$

siten, että niiden järjestys kuvaa jotakin ratsun polkua. Kaksi peräkkäistä ruutua tulee järjestää niin, että ylänumeroiden ero on 1 ja alanumeroiden 2 tai että ylänumeroiden ero on 2 ja alanumeroiden 1. Tällöin jokainen siirto on siis laillinen ratsun siirto. Nyt voidaan etsiä 16 ruudun joukko, jolle edellä kuvattu sääntö pätee ja muut ratsun polun osat saadaan symmetrialla. Aluksi valitaan 64 ruudusta satunnaisesti mikä tahansa esim. ruutu $\frac{1}{2}$. Tämän jälkeen kirjoitetaan ylös tämän ruudun kaikki symmetriaversiot $\frac{8}{2}$, $\frac{1}{7}$ ja $\frac{8}{7}$ (kts. kuva 2.3).



Kuva 2.3: Kaikki ruudun $\frac{1}{2}$ symmetriaversiot

Symmetriaversiot saadaan käymällä kaikki lukuparista 1 ja 2 saatavat permutaatiot läpi ja yleisesti kaavan (2.1) mukaan. Kaavassa 1 vaihdetaan 8:ksi, 2 vaihdetaan 7:ksi, 3 vaihdetaan 6:ksi, 4 vaihdetaan 5:ksi jne.

$$\begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 \end{array} \tag{2.1}$$

Tämä vastaa tilannetta, että ruutu $\frac{1}{2}$ peilataan shakkilaudan keskipisteen kautta kulkevan pystysuuntaisen akselin suhteen (ruutu $\frac{8}{2}$), vaakasuuntaisen akselin suhteen (ruutu $\frac{1}{7}$) ja halkaisijaa pitkin kulkevan akselin suhteen (ruutu $\frac{8}{7}$). Poistetaan valitut ruudut 64:stä. Sitten valitaan satunnaisesti jokin laillinen ratsun siirto ruuudusta $\frac{1}{2}$ esim. ruutuun $\frac{3}{1}$. Nyt taas kirjoitetaan ylös ruudun $\frac{3}{1}$ symmetriaversiot $\frac{6}{1}$, $\frac{3}{8}$ ja $\frac{6}{8}$. Nämä poistetaan nyt 60:stä ja jatketaan samalla tavalla kunnes saadaan 4 erilaista 16 ruudun symmetristä polkua. Lopuksi polut sitten yhdistetään sopivasti yhdeksi kokonaiseksi ratsun poluksi. Jos ruutuja valitaan täysin satunnaisesti, voidaan joutua peruuttamaan.

Uudempia ratkaisuja ratsun polun etsimiseen ovat muurahaisyhdyskuntaoptimointialgoritmit [10] ja neuroverkkoalgoritmit [13].

3 Silverlight

Silverlight on kehitysalusta, joka mahdollistaa näyttävien web-, työpöytä- ja mobiilisovellusten tekemisen. Silverlight toimii eri alustoilla kuten Macilla ja Linuxilla ja myös webissä eri selaimilla [6]. Silverlight-sovellusten tekemiseen soveltuu Visual Studio -ohjelmointiympäristö. Blend on enemmän graafisen suunnittelijan työkalu, mutta sillä voidaan myös lisätä tominnallisuutta komponentteihin. Näiden lisäksi voidaan käyttää pelkkään komponenttien piirtämiseen Design-sovellusta. Silverlightissa on tuki eri ohjelmointikielille kuten C#:lle ja Visual Basic:lle.

Nykyään Silverlight ja XNA Framework ovat ensisijaisia alustoja Windows Phone -sovelluksille [4]. Myös Symbian-puhelimille on olemassa tuki Silverlightille.

3.1 XAML-kuvauskieli

Silverlight-sovellusten käyttöliittymä ja käyttöliittymän komponentit kuvataan ensisijaisesti XAML:lla, joka pohjautuu XML-kieleen. Silverlightin XAML on likimääräisesti osajoukko WPF:n XAML:sta [9]. Kaikki mitä voidaan määritellä XAML-muodossa voidaan kuitenkin kirjoittaa myös ohjelmakoodina. Jokainen Silverlightin user control sisältää aina XAML-pohjan ja näin jokaisella komponentilla, joka luodaan tästä User control -luokasta, on sama pohja, jota voidaan muokata myöhemmin.

XAML toimii myös välittäjänä käyttöliittymäkomponenttien siirtämisessä sovelluksesta toiseen. Esim. Designissa piirretty käyttöliittymäkomponentti voidaan siirtää Visual Studioon XAML-muodossa. Joidenkin asioiden kuten esim. monimutkaisten muotojen piirtäminen on kätevämpää määritellä XAML:ssa kuin kirjoittaa koodina.

XAML:ia käytetään kuvaamaan Silverlight-sovelluksen ulkonäkö, mutta toiminnallisuus puolestaan kerrotaan ohjelmakoodissa. Näin sovelluksen ulkonäköä voidaan muokata koskematta ohjelmakoodiin. Kuvassa 3.1 on annettu esimerkki yksinkertaisesta XAML-tiedostosta.

XAML sisältää erilaisia säiliöitä käyttöliittymän komponenttien sijoitteluun. Grid-elementti sijoittaa automaattisesti komponentteja ruudukkomaisesti. Canvas-elementin kanssa joudutaan huolehtimaan komponenttien sijoittelusta itse, mutta toisaalta tämä antaa enemmän vapautta komponenttien sijoitteluun. Canvas-elementtiä käyttämällä komponenttien sijainti saadaan määriteltyä pikselin tarkkuudella.

Esimerkin kaltainen tiedosto kuvassa 3.1 luodaan automaattisesti Silverlight-projektin

```
<UserControl x:Class="SilverlightApplicationNappula.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
    d:DesignHeight="300" d:DesignWidth="400">

    <Grid x:Name="PaaGrid" Background="White">
        <Button Name="nappulainen" Content="Paina tästä" Height="23" Width="75"/>
    </Grid>
</UserControl>
```

Kuva 3.1: Esimerkki XAML-tiedostosta

luomisen yhteydessä. Tiedoston nimi on MainPage.xaml [5]. Tämä on päätiedosto, joka sisältää kaikki muut sovelluksen käyttöliittymäkomponentit. Esimerkissä uloin XAML-elementti on User Control, joka sisältää oletuksena Grid-elementin. Grid-elementin sisälle laitetaan ohjelman varsinaiset komponentit kuten tässä esimerkissä Button-elementti, joka on tavallinen nappula. Elementeille voidaan määritellä ominaisuuksia esim. Button-elementin korkeudeksi on määritelty nyt 23 pikseliä ja leveydeksi 75 pikseliä.

3.2 User control -käyttöliittymäkomponentti

User control on Silverlightin muokattava käyttöliittymäkomponentti [8]. User control sisältää komponentin käyttöliittymän ja toiminnallisuuden. User controllia on hyvä käyttää, kun halutaan tehdä oma käyttöliittymäkomponentti, jota mahdollisesti käytetään sovelluksessa useita kertoja tai halutaan tehdä uudelleenkäytettävä komponentti muita projekteja varten. Ohjelmointikielen näkökulmasta user control on luokka, joka on peritty Control-luokasta.

Silverlightissa on monia valmiita käyttöliittymäkomponentteja, joita voidaan käyttää tehdessä omaa user controllia.

4 Verkkojen piirto-ohjelma

Tutkielman aikana on ohjelmoitu verkkojen piirto-ohjelma käyttäen Silverlightia. Ohjelma toimii käyttäjän web-selaimessa ohjelman nettisivulla[1]. Ohjelmassa piirretään yksinkertaisia ja suuntaamattomia verkkoja. Ohjelmassa ei siis pysty piirtämään rinnakkaisia välejä. Ohjelmassa voidaan myös tarkastella ratsun polusta muodostunutta verkkoa shakkilaudalla. Ohjelman käyttöliittymästä on pyritty tekemään yksinkertainen.

4.1 Ohjelman käyttö

Solmut ovat ohjelmassa sinisiä ympyröitä, jotka ohjelma nimeää automaattisesti kirjaimin. Välit ovat sinisiä viivoja.

Verkkojen piirtämiseen käytetään ohjelmassa vain hiirikomentoja. Solmujen piirtäminen tapahtuu klikkaamalla vasemmalla hiiren napilla mistä tahansa sovelluksen sinertävästä taustasta. Solmuja poistetaan klikkaamalla oikealla hiiren napilla poistettavan solmun päällä. Solmuja voidaan siirtää vasemmalla hiiren napilla klikkaamalla ja raahaamalla solmuja niiden oransseista kahvoista. Oranssi kahva ilmestyy, kun hiiren kursori viedään solmun ylle.

Välien piirtämiseen vaaditaan vähintään kaksi solmua, joiden välille väli voidaan piirtää. Väli piirretään painamalla vasen hiiren nappi alas ensimmäisen solmun kohdalla ja vapauttamalla nappi toisen solmun kohdalla. Välejä poistetaan klikkamalla niitä oikealla hiiren napilla. Jos poistetaan solmu, josta lähtee välejä muihin solmuihin, niin samalla myös solmusta lähteneet välit poistetaan. Ohjelman näkymän oikeassa alakulmassa on nappi "Puhtaaksi", josta klikkaamalla kaikki piirretyt solmut ja välit poistetaan.

Ohjelman käytön aikana voidaan myös käyttää web-selaimen omia zoomaus-nappuloita, jolloin Silverlight-ohjelman komponentit skaalautuvat automaattisesti oikeaan kokoon.

4.1.1 Ratsun polku

Ohjelman näkymän vasemmassa yläkulmassa on nappi "Ratsun polku". Tästä klikkaamalla ohjelma näyttää shakkilaudan. Ohjelma havainnollistaa etsittyä ratsun polkua

[1] Verkkojen piirto-ohjelma `http://users.jyu.fi/~eemikukk/kandi/`

piirtämällä solmut shakkilaudan ruutuihin. Ohjelma nimeää shakkilaudalle piirretyt solmut numeroilla, jotka kertovat monesko ruutu on kyseessä ratsun polusta. Luvut ovat siis väliltä 1–64. Kun shakkilauta on näkyvissä, voidaan hiirellä klikata mitä tahansa ruutua ja ohjelma aloittaa ratsun polun etsimisen. Tämän jälkeen siirtoja voidaan selata shakkilaudan yläpuolella olevista nuolinäppäimistä.

Tällä hetkellä shakkilaudan kokoa voidaan muuttaa vain ohjelmakoodista. Ohjelma kuitenkin voi löytää ratsun polun myös muilla kuin $8 * 8$-kokoisilla laudoilla. Myös algoritmi, jolla ratsun polkua etsitään, tulee valita ohjelmakoodin puolella.

4.2 Ohjelman rakenne

Monet verkkojen piirto-ohjelman käyttöliittymän osat ovat olioita. Ohjelma on tapahtumapohjainen. Eli kun käyttäjä tekee jonkin toiminnon esim. napsauttaa hiirellä, niin ohjelma huomaa tämän ja tekee siihen liittyvän toimenpiteen.

4.2.1 Luokkahierarkia ja luokkien sisältö

Ylimpänä luokkana ohjelmassa on MainPage-luokka, jonka niminen luokka luodaan automaattisesti Silverlight-projektin luomisen yhteydessä. Tämä luokka sisältää attribuutteinaan (kts. kuva 4.1) mm. Verkot-luokan solmuille ja väleille ja ShakkiLauta-luokan shakkilaudalle. Luokka sisältää attrbituutteinaan myös DispatcherTimer-ajastimet ratsun polun piirtämisen havainnollistamiseksi lähinnä ohjelman debuggausta varten. MainPage-luokka piirtää ohjelman näkymän kulmissa sijaitsevat nappulat ruudulle. MainPage-luokka huolehtii solmujen piirtämisestä ruudulle, kun hiiren nappulaa painetaan ja myös ohjelman kulmissa sijaitsevien nappuloiden tominnallisuudet.

```
public partial class MainPage : UserControl
    {
        private Verkot verkot;
        private ShakkiLauta shakkiLauta;
        private Graafi graafi;
        private Grid nappulaPaneeli;
        private SoivaLuokka soitanto;
        private DispatcherTimer ajastin;
        private DispatcherTimer moneskoRuutuAjastin;
        private Laskuri laskuri;
        private Boolean shakkiPainettu = false;
        ...
    }
```

Kuva 4.1: MainPage-luokan attribuutit

Ohjelman alussa luodaan uusi Verkot-olio ja ShakkiLauta-olio, jonka ruutumäärä voidaan määrittää sen konstruktorissa. Shakkilauta asetetaan aluksi kuitenkin piiloon käyttäjältä ohjelman taustan taakse kuten kuvassa 4.2 ja asetetaan näkyviin vasta, kun Ratsun polku-nappulaa painetaan. Piilotus tapahtuu vaihtamalla ShakkiLauta-olion Canvas.ZIndex:ä pienemmäksi kuin ohjelman taustan Canvas.ZIndex. Tällöin ShakkiLauta-olio piirtyy ohjelman taustan taakse eli siirtyy syvyyssuunnassa z-akselilla ohjelman taustasta kauemmaksi.

```
private void alustaShakkiLauta()
        {
            Canvas.SetZIndex(shakkiLauta, -1);          // asetetaan shakkilauta piiloon
            LayoutRoot.Children.Add(this.shakkiLauta);  // laitetaan shakkilauta LayoutRootin lapseksi
        }
```

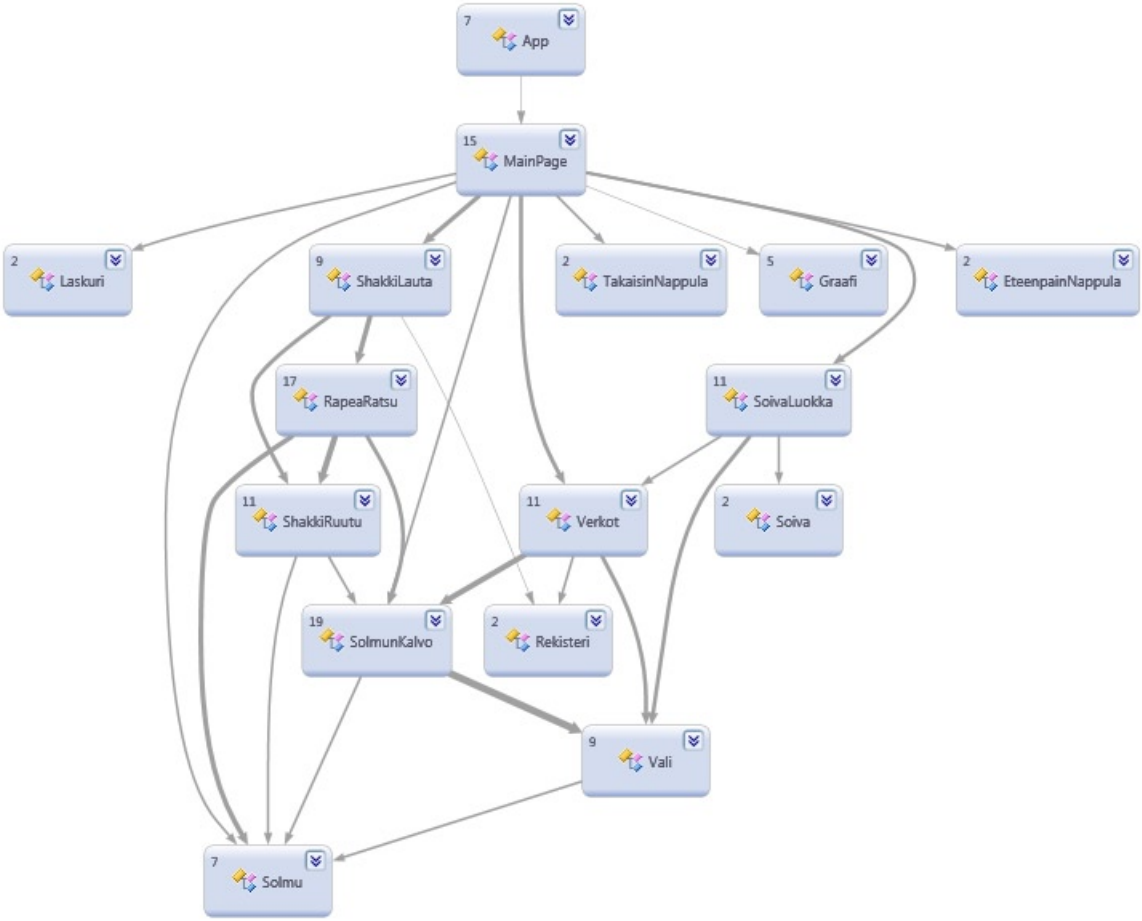
Kuva 4.2: Shakkilaudan piilotus ohjelman alussa

Verkot-luokka on perinteinen C#-luokka siinä mielessä, että sillä ei ole graafista XAML:ssa määritettyä ulkonäköä kuten useimmilla tämän ohjelman luokilla on. Verkko-luokka sisältää attribuutteinaan mm. solmunKalvot-listan, joka on lista SolmunKalvo-olioita ja valit-listan, joka on lista kaikista ohjelmassa piirretyistä väleistä.

SolmunKalvo-luokka on ehkä oleellisin luokka toiminnaltaan verkkojen piirto-ohjelmassa. SolmunKalvo-luokka sisältää mm. Solmu-olion, joka on oikeastaan vain solmun ulkonäkö ja valit-listan, joka on viite Verkot-luokan kaikkien välien listaan. SolmunKalvo-luokka on graafisesti läpinäkyvä ympyrä, joka piirretään jokaisen Solmu-olion päälle. SolmunKalvo-luokka sisältää solmun varsinaisen toiminnallisuuden, koska käyttäjä tulee aina klikanneeksi SolmunKalvo-olioita klikatessaan ohjelman solmua. SolmunKalvo-luokka ottaa kiinni tapahtumat, kun aletaan piirtää välejä ja huolehtii myös oranssin kahvan piirtämisestä, kun hiiri viedään solmun päälle. Luokka huolehtii myös solmujen liikuttelusta hiirellä.

Kuva 4.3 on suunnattu verkko luokkien välisistä riippuvuuksista. Solmut kuvaavat ohjelman luokkia ja välit luokkien välisiä metodikutsuja. Välin suunta ilmoittaa kutsun suunnan ja välin paksuus kuvaa sitä, kuinka monta kutsua kohdeluokkaan tehdään. Jos verkossa olisi rinnakkaisia välejä, ne kuvaisivat luokkien välisiä molempiin suuntiin kulkevia kutsuja. Ohjelmassa ei siis ole molempiin suuntiin kulkevia metodikutsuja ollenkaan. Solmun vasemmassa yläkulmassa oleva luku tarkoittaa luokan omien metodien määrää.

Solmun piirtämistä Canvas-säiliöön esitellään kuvassa 4.4 sekvenssikaaviona. Esitelty metodi sijaitsee MainPage-luokassa. Piirtäminen alkaa, kun hiiren oikea nappi vapautetaan. Tämän jälkeen tulee if-lohko, jossa tarkistetaan onko tapahtuman herät-



Kuva 4.3: Kaavio luokkien riippuvuuksista

täjänä MainPagen Canvas-olio nimeltä kangas. Jos on niin luodaan uusi Solmu-olio ja mennään Solmu-luokan konstruktoriin. Tämän jälkeen luodaan SolmunKalvo-olio ja annetaan SolmunKalvo-luokan konstruktorille aiemmin luotu Solmu-olio parametrina. Näin saadaan SolmunKalvo-olio tietämään oma solmunsa. Tämän jälkeen kutsutaan Verkot-olion Add-metodia ja annetaan sille parametriksi aiemmin luotu SolmunKalvo-olio. Add-metodissa asetetaan SolmunKalvo-oliolle pääkangas piirtämistä varten ja asetetaan valit, jotta SolmunKalvo-olio tietäisi kaikki ohjelmassa luodut välit. Yhä Add-metodissa pyydetään Rekisteri-oliolta getTunnus-metodilla solmulle oma tunnus eli aakkosten kirjain ja asetetaan tunnus Solmu-olion tekstikenttään. Viimeiseksi lisätään SolmunKalvo-olio solmunKalvot-listaan. Lopulta palataan MainPagen metodiin ja lopetetaan käskyllä return.

4.3 Ratsun polun algoritmien tarkastelua

Ohjelmaan on kirjoitettu kaksi algoritmia ratsun polun etsimiseksi. Kumpikin algoritmeista etsii avoimia ratsun polkuja. Ensimmäisessä algoritmissa seuraava ratsun siirto valitaan satunnaisesti. Tällöin kokonaisen ratsun polun löytäminen saattaa olla vaikeaa. Toinen algoritmi etsii ratsun polkua Warndorfin heuristiikan periaatteilla. Käytän termejä "Warndorfin heuristiikan periaatteilla" ja "Warndorfin tavoin", koska itselläni ei ole alkuperäistä Warndorfin algoritmia eikä näin kirjoittamani algoritmi luultavasti toimi aivan kuten Warndorf on algoritminsa alunperin määritellyt.

Satunnaisesti etsivässä algoritmissa on myös yksinkertainen peruutus. Peruutus tapahtuu, jos Ruutu-olion ehdottama ruutu löytyy jo tähän asti etsitystä ratsun polusta. Koska peruutus on yksinkertainen eikä pidä muistissa aiempia peruutuksia, saattaa ratsu jäädä jumiin eikä luonnollisestikaan löydä ratsun polkua. Kirjoittamaani Warndorfin algoritmiin ei ole toteutettu peruutusta, mutta sellaista ei välttämättä tarvita jotta ratsun polku voitaisiin löytää.

Molemmissa algoritmeissa Ratsu-olio kysyy Ruutu-oliolta eli ruudulta, jossa tällä hetkellä ollaan, seuraavaa ruutua, johon ratsu siirretään. Tähän Ruutu-olio vastaa palauttamalla seuraavan siirron ruudun, kunnes Ruutu-olio on etsinyt sen valitulla algoritmilla.

4.3.1 Ratsun polkua satunnaisesti etsivä algoritmi

Kuten kuvassa 4.5 satunnaisessa algoritmissa $hevosenHyppy$-muuttujaan arvotaan kokonaisluku ja $siirrot$-muuttujaan tallennetaan mahdollisten siirtojen määrä. Muuttuja $hevosenHyppy$ saa arvon väliltä $0 \leq hevosenHyppy < siirrot$. Seuraavaksi siirroksi otetaan listasta ruutu $hevosenHyppy$-arvon kohdalta. Tämän jälkeen satunnaisesti valittu ruutu poistetaan tämänhetkisen ruudun mahdollisista siirroista, ja näin ruudun siirrot voivat loppua kesken ennen kuin ratsun polkua ollaan löydetty. Tämän takia jos tämänhetkisen ruudun siirrot loppuvat eikä ratsun polkua ole löydetty, lisätään ruudulle siirtoja kaikkiin mahdollisiin ruutuihin ja peruutetaan. Tämänkaltainen siirtojen lisääminen voisi tuottaa ongelmia Warndorfin tapaan toimivassa algoritmissa, jos ratsun polun etsimiseen käytettäisiin molemman kaltaisia algoritmeja yhtäaikaa.

4.3.2 Ratsun polkua Warndorfin tavoin etsivä algoritmi

Warndorfin tavoin ratsun polkua etsivässä algoritmissa on keskeistä siirtyä seuraavaksi siihen ruutuun, jolla on vähiten etenemismahdollisuuksia verrattuna muihin ruutuihin. Jos kahdella ruuudulla sattuu olemaan saman verran ruutuja, joihin edetä, tarkastel-

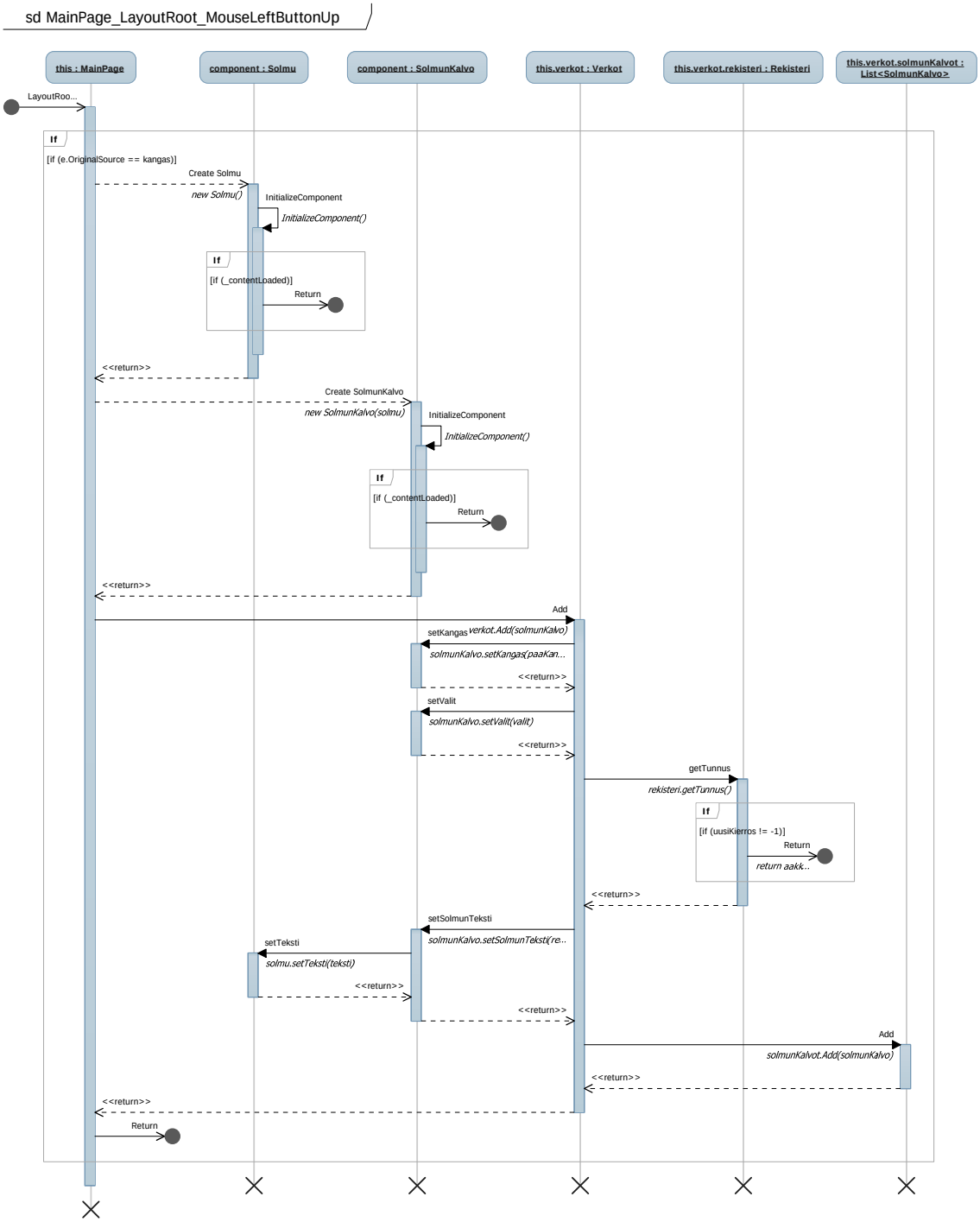
laan etenemismahdollisuuksia näitä ruutuja seuraavista ruuduista. Ratsun polun etsiminen lopetetaan kunnes tämänhetkisestä ruudusta ei ole yhtään mahdollista siirtoa eteenpäin. Tavallisissa tilanteissa tällöin ollaan löydetty ratsun polku.

Kuvassa 4.6 haetaan ruutua, jolla on vähiten etenemismahdollisuuksia. Muuttujaan *pieninSiirtoMaara* on aluksi tallennettu suuri luku. Aina kun *pieninSiirtoMaara* sisältää suuremman arvon kuin tämänhetkisellä ruudulla on mahdollisia siirtoja, tallennetaan ruutu *seuraavaSiirto* muuttujaan. Samalla pidetään tallennettuna ruutua, jolla on toiseksi vähiten etenemismahdollisuuksia, siltä varalta että kahdella ruudulla on sama määrä mahdollisia siirtoja. Jos todetaan että kahdella ruudulla on sama määrä etenemismahdollisuuksia, katsotaan eteenpäin kummalla ruudulla on sellainen ruutu, jolla on vähiten mahdollisia siirtoja. Sitten siirrytään siihen ruutuun, jolla on edessään ruutu, jolla on vähiten siirtoja.

Tässä kohdassa tulee huomata, että ohjelmassa otetaan huomioon ainoastaan sellainen tilanne, jossa vain kahdella ruudulla on sama määrä mahdollisia siirtoja, mutta joskus tällaisia ruutuja voi olla enemmänkin kuin kaksi. Tällöin voi olla, ettei ratsun polkua saada löydettyä vaikka sen pitäisi olla mahdollista.

4.4 Ratsun polun ja verkkoteorian soveltaminen musiikin säveltämiseen

Ohjelma voidaan laittaa soittamaan nuotteja piirrettyjen verkkojen mukaan. Soitettujen nuottien korkeus riippuu solmujen sijainnista. Tilassa jossa verkkoja voidaan piirtää vapaasti, kunkin nuotin korkeus riippuu solmun sijainnista leveysakselilla. Puolestaan ratsun polun verkossa nuotti soitetaan solmun sarakkeen mukaan. Ohjelmassa ei voi laittaa soimaan useampaa kuin yhtä verkkoa kerrallaan.



Kuva 4.4: Sekvenssikaavio solmun piirtämisestä

```
public ShakkiRuutu Sokea_SeuraavaSiirto(Random villiHevonen, List<ShakkiRuutu> ratsunPolku,
                                        List<Solmu> solmut, int ruudunNumero)
        {
            int hevosenHyppy;
            ShakkiRuutu seuraavaSiirto;
            do
            {
                int siirrot = mahdollisetSiirrot.Count;
                // Jos kaikki mahdolliset siirrot on tehty eikä ratsun polkua ole
                // saatu muodostettua, niin 1) lisätään mahdollisia
                // siirtoja kaikkiin ympärillä oleviin ruutuihin ja 2) peruutetaan.
                if (siirrot == 0 && ratsunPolku.Count <= ruutujenMaara)
                {
                    return siirrotLoppu(ratsunPolku);
                }
                // Ratsun polku löydetty!
                if (ratsunPolku.Count >= ruutujenMaara) return null;
                hevosenHyppy = villiHevonen.Next(siirrot);
                seuraavaSiirto = mahdollisetSiirrot[hevosenHyppy];
                mahdollisetSiirrot.Remove(seuraavaSiirto);
            } while (ratsunPolku.Contains(seuraavaSiirto));

            return seuraavaSiirto;
        }
```

Kuva 4.5: Ratsun polun satunnaista etsintää

```
foreach (ShakkiRuutu ruutu in mahdollisetSiirrot)
            {
                if (pieninSiirtoMaara >= ruutu.mahdollisetSiirrot.Count)
                {
                    pieninSiirtoMaara = ruutu.mahdollisetSiirrot.Count;
                    toinenSeuraavaSiirto = seuraavaSiirto;
                    seuraavaSiirto = ruutu;
                }
            }
```

Kuva 4.6: Warndorfin tavoin etsitään ruutua, jolla on vähiten etenemismahdollisuuksia

5 Ohjelman kehittämisprosessin arviointia

5.1 Silverlight ja XAML

Mielestäni itse Silverlight auttaa kehittäjää tekemään sovelluksia nopeasti eikä teknologian käyttäminen aiheuttanut suurempia ongelmia. Erityisesti grafiikan piirtämisessä pääsee hyvin alkuun Silverlightin valmiiden piirto-ominaisuuksien ja Design-ohjelman tuen ansiosta. Esim. ohjelmaa tehdessä en tarvinnut käyttää kuvatiedostoja grafiikan piirtämisessä vaan XAML:lla määritetyt komponentit riittivät. Myös XAML:n käyttö vaikuttaa mielestäni järkevältä ratkaisulta. XAML-tiedostoista ei tule kohtuuttoman pitkiä, kun ohjelma jaetaan useampiin luokkiin. Näin XAML-tiedostojen lukemisesta ei tule hankalaa, toisin kuin mahdollinen XAML:n tai XML:n saama kritiikki on antanut olettaa. Komponenttien luominen dynaamisesti koodin puolella myös keventää XAML:n vastuuta ja näin parantaa XAML-tiedostojen luettavuutta. Joidenkin asioiden muokkaaminen XAML:in avulla voi myös olla nopeampaa kuin muutoksen tekeminen ohjelmakoodiin. Saamani kokemuksen mukaan myös komponenttien lisääminen dynaamisesti ohjelmakoodissa ajon aikana toimii moitteetta.

5.2 Käyttöliittymä

Ohjelman graafisen ilmeen tein suurimmaksi osaksi kuten itse näin hyväksi. Kuitenkin solmujen kahvojen väriä mietin käyttöliittymän kannalta niin, että väri olisi selkeä ja tulisi esiin taustan väristä. Verkkojen piirtämisen yhteydessä pidin tärkeänä, ettei verkkojen piirtämiseen tarvittaisi kuin hiirtä. Tämä siis täyttää pyrkimystäni tehdä käyttöliittymästä yksinkertainen.

5.3 Ohjelmointi

Verkkojen piirto-ohjelma on toteutettu olio-ohjelmoinnin periaatteita käyttäen [1]. Mielestäni on hyvä, että Silverlightia ohjelmoidaan nimen omaan C#:n kaltaisella kielellä, joka on esim. lähellä Javaa ja joka on myös sama kieli, jolla tehdään Windowsin työpöytäsovellukset ja XNA-alustan pelit. Tämä voi helpottaa Silverlight-teknologiaan siirtymistä ja Silverlightin yhdistämistä esim. XNA:n kanssa.

Verkkojen piirto-ohjelman toiminnallisuutta pyrin jaottelemaan luokkiin ja jaka-

maan kullekin luokalle omat tehtävänsä. Asioiden abstrahointi voi onnistua paremmin C#:n kaltaisella olio-ohjelmointikielellä kuin webissä usein käytetyillä skriptikielillä. Kuitenkin voi olla aiheellista kritisoida sitä, että ohjelmassa ei ole selvemmin jaoteltu erikseen komponenttien piirtämistä ja toiminnallisuutta. Koodin uudelleenkäyttö saatta tästä johtuen kärsiä.

Debuggausta tuli tehdä erityisesti ratsun polun etsimisen kanssa, jolloin oli selkeämpää saada nähdä reaaliajassa kuinka algoritmin suorittama etsiminen sujui. Käytin siis ohjelmaa itseään ratsun polun piirtämiseen sitä mukaa, kun tietokone eteni etsinnässä. Pelkästä konsoliin tulostetusta tiedosta voi olla hankalaa nähdä kuinka etsiminen edistyy tämänkaltaisessa ongelmassa.

Joissain kohdin kuten esim. nappuloiden luomisen yhteydessä tein metodin, jolla voi luoda nappuloita antaen nappulan nimen ja muita tarvittavia tietoja. Näin koodin muokkaaminen helpottuu eikä yhteen kohtaan tule paljon samannäköistä koodia.

Muutosten tekeminen koodiin saattaa silti olla hankalaa. Tämä johtunee siitä, että ohjelma olisi vaatinut enemmän suunnittelua alun perin tai koodin arviointia kehittämisprosessin aikana. Esim. lisättäessä shakkilaudalle välienpiirto-ominaisuus täytyy tehdä uudenlainen piirtometodi, koska alkuperäinen metodi ei sovellu tähän tarkoitukseen. Tämä johtuu siitä, että alkuperäinen metodi sopii välien piirtämiseen Canvas-säiliölle muttei Grid:lle. Silverlightin kanssa juuri tämä Canvasin ja Gridin yhteen sovittaminen saattaa tuottaa ongelmia, mutta toisaalta niiden erilaiset ominaisuudet ovat hyödyllisiä käyttää, mikä voi johtaa niiden käyttämiseen samanaikaisesti. Konkreettisesti ongelman huomaa silloin, kun shakkilaudan kokoa muutetaan, niin kovakoodatut välien piirtämiseen tarkoitetut koordinaatit aiheuttavat, että välit piirtyvät vääriin kohtiin.

C#:n interfaceja käytin siinä vaiheessa, kun tarvitsin vapaasti piirrettäville verkoille ja ratsun polun verkoille eri tavat laskea soitettavan nuotin korkeus.

5.3.1 Ratsun polun algoritmit

Ratsun polun algoritmeista satunnainen algoritmi oli helpompi toteuttaa ja on myös koodissa lyhyempi Warndorfin tavoin toimivaan algoritmiin verrattuna. Osa molempien algoritmien koodista näyttää kuitenkin samalta ja niitä olisi ehkä mahdollista refaktoroida. Molemmissa niin Ratsu-luokassa kuin Ruutu-luokassa on algoritmien välillä samankaltaisuutta. Muokattavuuden kannalta ohjelmaan olisi ehkä hyvä rakentaa järjestelmä, jolla olisi helpompi lisätä ohjelmaan uusi algoritmi, joka toimisi vähin muokkauksin.

Satunnaisesti etsivän algoritmin peruutus jäi yksinkertaiseksi. Tämän kehittäminen

jo tehtyjen peruutuksien tallentamisella voisi parantaa ratsun polkujen löytymistä. Tällöin algoritmi alkaisi muistuttaa enemmän raa'an voiman algoritmia, mutta joka käy eri vaihtoehtoja läpi satunnaisesti.

Mielenkiintoista olisi ollut nähdä, olisiko ollut mahdollista käyttää satunnaista ja Warndorfin heuristiikkaan perustuvaa algoritmia yhtäaikaisesti. Tällöin esim. muutama ruutu olisi voitu valita satunnaisesti ja loput Warndorfin heuristiikan mukaisesti.

5.4 Suorituskyky

Teknologian suorituskyky ei tämän mittaluokan ohjelmassa juuri tullut vastaan. Ohjelmaa käytettäessä webistä eniten latausaikaa menee äänitiedostojen siirtämiseen eikä niinkään ohjelman itsensä käynnistämiseen. Suurten tiedostojen siirtämiseen on myös vaihtoehtoisia ratkaisuja eikä niitä tarvitse siirtää palvelimelta käyttäjän koneelle heti ohjelman käynnistysvaiheessa. Jos isojen tiedostojen siirtoa lykättäisiin, saataisiin ohjelman käynnistys nopeutumaan.

Itse en ole testannut, kuinka suurien shakkilautojen ratsun polkujen etsimisessä ohjelmalla alkaa kulua enemmän aikaa. $8 * 8$-kokoisen shakkilaudan ratkaiseminen vie ainakin käyttäjän näkökulmasta nykyaikaisilta koneilta vain hetken.

Selvimmin huomattavat viiveet ohjelman suorituskyvyssä nähdään, kun aletaan soittamaan nuotteja. Aikaväli, jolla nuotit soitetaan, voi huomata vaihtelevan vaikka nuottien pitäisi soida tasaisin väliajoin eli tietyssä tempossa. Tämä voi osaltaan johtua siitä, että Silverlight ei ole yhtä kykenevä animaatioiden ja äänten toistamiseen kuin esim. XNA [7]. Tällaisessa tilanteessa voi jo oppia sitä, että mitä enemmän pelimäisemmäksi ja ohjelman nopeaa reagoimista käyttäjän antamiin syötteisiin sovellus vaatii, sitä enemmän alkaa olla aihetta siirtyä Windows-ympäristössä XNA-alustaan.

6 Yhteenveto

Tutkielmassa on otettu selvää, soveltuuko Silverlight web-selaimessa ajettavan verkkojen piirtotyökalun tekemiseen. Tutkielmassa ohjelmoidun piirto-ohjelman kehityksestä saatiin selville, että Silverlight sopii tutkielmassa käytetyn mittaluokan verkkojen piirtämiseen. C# ja XAML tukevat kokemukseni perusteella sovelluskehitystä. XAML lisää eri kehitystyökalujen yhteentoimivuutta.

Ohjelmoidut ratsun polun algoritmit saatiin toimimaan. Warndorfin tavoin etsivä algoritmi löytää ratsun polun tavallisimmissa tapauksissa. Ratsun polun etsimistä ja verkkoteoriaa voitiin myös soveltaa musiikin säveltämiseen. Kuitenkin kun sovellukseen lisättiin reaaliaikaista toimintaa, voitiin huomata sovelluksen hidastumista tai toimimista epätahtiin musiikin toistamisessa. Reaaliaikaisia tapahtumia vaativia sovelluksia voi siis olla parempi kehittää XNA-alustalla.

7 Lähteet

[1] Deborah J. Armstrong. The quarks of object-oriented development. *Commun. ACM*, 49:123–128, February 2006.

[2] N.L. Biggs, Lloyd E.K., and Wilson R.J. *Graph Theory 1736-1936*. Oxford University Press, 1976.

[3] N. Christofides. *Graph Theory An Algorithmic Approach*. Academic, London, 1975.

[4] Microsoft Corporation. Application platform overview for Windows Phone. `http://msdn.microsoft.com/fi-fi/library/ff402531(v=VS.92).aspx`. Haettu 29.3.2012.

[5] Microsoft Corporation. How to: Create a new silverlight project. `http://msdn.microsoft.com/en-us/library/cc838164(v=vs.95).aspx`. Haettu 29.3.2012.

[6] Microsoft Corporation. Silverlight. `http://msdn.microsoft.com/en-us/library/cc838158(VS.95).aspx`. Haettu 29.3.2012.

[7] Microsoft Corporation. The Silverlight and XNA Frameworks for Windows Phone. `http://msdn.microsoft.com/en-us/library/ff402528(v=vs.92).aspx`. Haettu 29.3.2012.

[8] Microsoft Corporation. Usercontrol class. `http://msdn.microsoft.com/en-us/library/system.windows.forms.usercontrol.aspx`. Haettu 29.3.2012.

[9] Microsoft Corporation. Xaml overview. `http://msdn.microsoft.com/en-us/library/cc189036(v=VS.95).aspx`. Haettu 29.3.2012.

[10] P. Hingston and G. Kendall. Ant colonies discover knight's tours. In *AI 2004: Advances in Artificial Intelligence*, number 3339 in Lecture Notes in Computer Science, Berlin, 2004. Springer.

[11] P. Hämäläinen. *Tekoäly*. Jyväskylän yliopisto, 2011.

[12] J. Kyppö. Hamiltonin silmukka. `http://users.jyu.fi/~jorma/graphs/vk22.htm`. Haettu 29.3.2012.

[13] KC. Lee and Y. Takefuji. Finding knight's tours on an $M \times N$ chessboard with $O(MN)$ hysteresis McCulloch-Pitts neurons. *IEEE Transactions On Systems Man And Cybernetics*, 24(2):300–306, 1994.

[14] G.R. Selby. *The use of topological methods in computer-aided circuit layout.* PhD thesis, Imperial College of Science and Technology, University of London, 1970.

[15] K. Sugiyama. *Graph Drawing and Applications for Software and Knowledge Engineers.* World Scientific, River Edge, NJ, USA, 2002.

[16] Cornell University. Graph theory to repurpose existing drugs. `http://blogs.cornell.edu/info2040/2011/09/23/graph-theory-to-repurpose-existing-drugs/`. Haettu 29.3.2012.

[17] C. Vasudev. *Graph Theory with Applications.* New Age International, Daryaganj, Intia, 2006.