Sanajahti-solver
Projektin dokumentaatio

Ryhmä: Sanajahti3

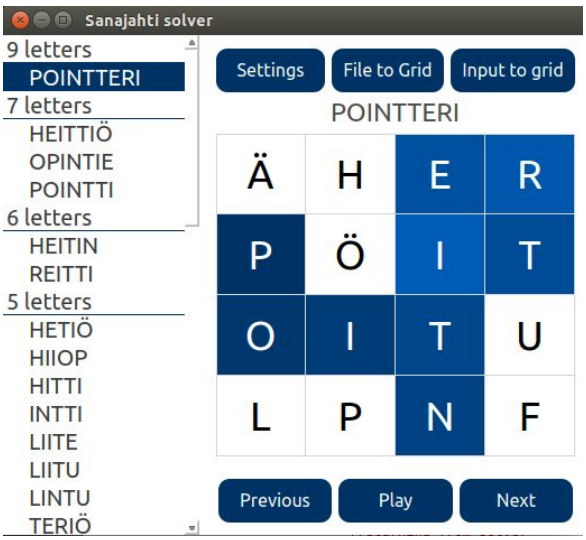
Matti Parkkila 480510
Lassi Heikkinen 291204
Timo Haario 428721
Kaisa Voutilainen 481629
Otto von Hellens 298870

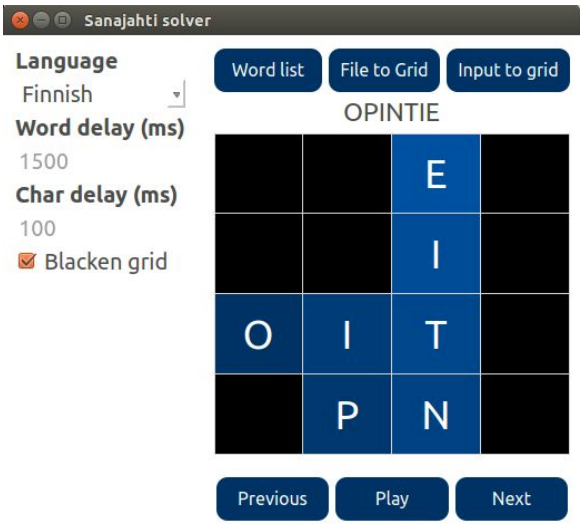
Yleiskatsaus

Projektin tavoitteena oli luoda ohjelma, joka auttaa käyttäjää huijaamaan Sanajahti-pelissä, antaen käyttäjälle selkeän edun muita pelaajia vastaan. Ohjelma ratkaisee Sanajahdin 4x4 kirjainruudukosta kaikki mahdolliset kirjainjonoista muodostuvat sanat.

Projektissa sisäisenä tavoitteenamme oli saada aikaiseksi Expert-tason työ. Ohjelman toiminnalisuuksiin kuuluu:

- nopea ja joustava ratkaisija
- helppokäyttöinen graafinen käyttöliittymä
- konenäöllä tapahtuva ruudukon lukeminen
- hyödyllinen kokonaisuus Sanajahdin liveratkaisemiseen





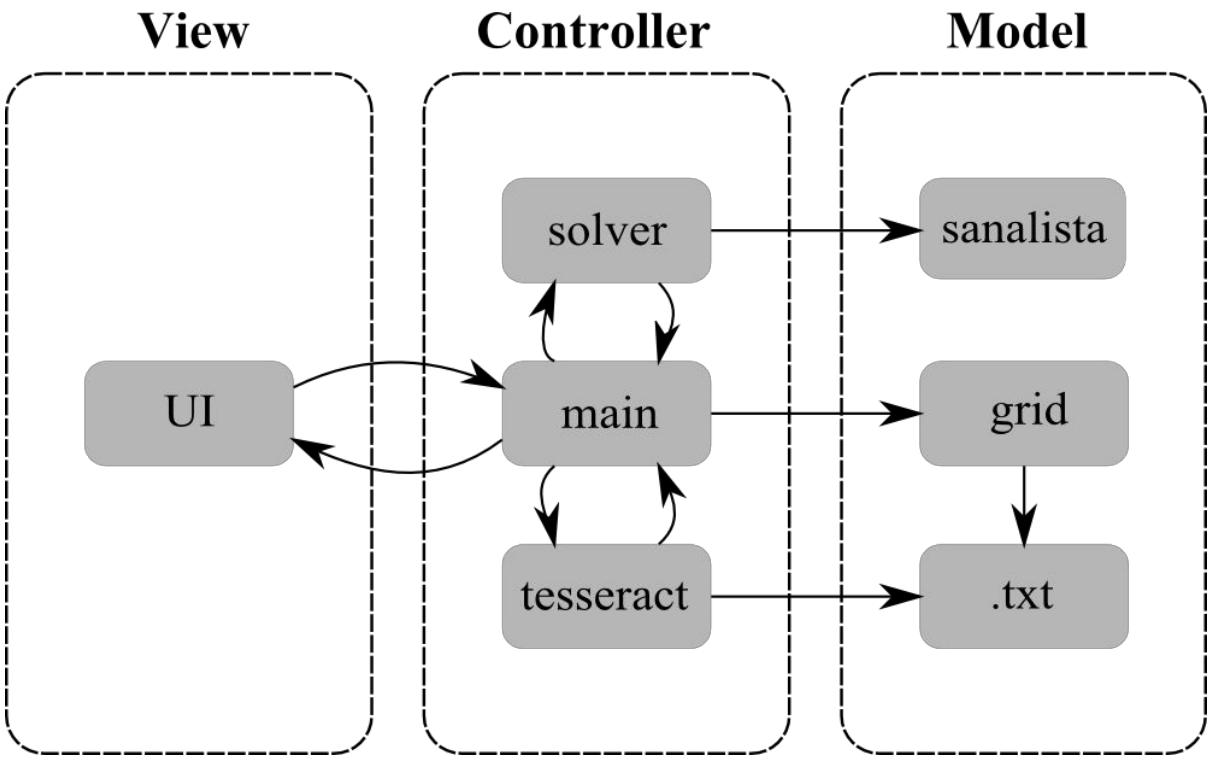
Lopputuloksena ohjelma on toimiva, selkeä ja tehokas. Käyttäjä voi helposti lisätä haluamansa Sanajahtiruudukon ohjelman ratkaistavaksi kolmella eri tavalla (Input näppäimistöltä, käyttöjärjestelmän oma tiedostoselain tai tiedoston drag&drop, joko .txt tai screenshot Sanajahdista muodossa .jpg tai png.)

Ohjelma hakee ruudukosta löytyvät sanat ja näyttää ne käyttäjälle. Käyttäjä voi vaihtaa näytettävää sanaa valitsemalla haluamansa sanan listasta tai painamalle Previous/Next painikkeita. Käyttäjä voi myös laittaa ohjelman toistamaan sanan toisensa perään painamalla Play-painiketta. Ohjelmassa on omat asetuksensa käytettävälle sanalistalle ratkaisuissa (Finnish/English), sekä sanojen piirtämisessä käytettäville muuttujille (Sanojen ja kirjainten välinen tauko, ruudukon mustaus poislukien sanan vaatimat kirjaimet.

Projekti aikataulutettiin niin, että saimme kaikki projektin osa-alueet toteutettua. Kokonaisuutena ohjelman suunnittelu ja toteutus täyttää asettamamme vaatimukset tarkoituksenmukaisesta ohjelmasta, joka toimii luontevasti useammassa eri käyttötarkoituksessa. Käyttöliittymä on rajattu 4x4-ruudukkoon Sanajahti-sovelluksen mukaisesti, mutta ratkaisualgoritmi tukee mielivaltaisen kokoisia ruudukoita. Jatkokehityksessä myös käyttöliittymä voitaisiin helposti laajentaa isommille ruudukoille.

Työosuudet jaoimme siten, että jokainen ryhmän jäsen sai riittävän haastavia, omaa osaamisaluettaan vastaavia tehtäviä.

Ohjelman rakenne



Projektissa käytimme kehitysympäristönä Qt:tä ja tekstintunnistukseen (OCR, optical character recognition) erillistä Tesseract OCR -kirjastoa. Tesseractin käyttö vaatii lisäksi kuvankäsittelyyn Leptonica-kirjaston. Graafinen käyttöliittymä (UI-luokka) toteutettiin käyttäen Qt-kehitysympäristöä. Qt:n avulla on helppo luoda käyttöliittymästä käyttäjäystävällisempi ja esteettisempi versio.

Projekti toteutettiin noudattamalla MVC-arkkitehtuuria (model-view-controller), jossa ohjelma jaetaan kolmeen osaan: malli (model), näkymä (view) ja käsittelijä (controller). Ohjelman osa-alueet toteutettiin erillisenä komponentteina, joiden toteutukset ovat toisistaan riippumattomia. Tämä mahdollisti komponenttien sisäisten toimintojen toteuttamisen ja muuttamisen vaivattomasti vaikuttamatta muihin komponentteihin.

Näkymä tai käyttöliittymä vastaa datan näyttämisestä sekä käyttäjän komentojen vastaanottamisesta. Käyttöliittymä välittää annetut käskyt käsittelijälle, joka vastaa tallennettujen tietojen lataamisesta tai muokkaamisesta. Tietojen lukemista/tallentamista sekä käyttöliittymää voitiin siis toteuttaa samanaikaisesti, kunhan niiden välissä oleva käsittelijä oli suunniteltu kokonaisuuden kannalta toimivaksi.

Ohjelma toteutettiin olio-ohjelmoinnin avulla, jossa erilliset komponentit toteutettiin omina luokkinaan, joiden instansseja käytetään ohjelman ajamisen aikana eri tarkoituksiin. Luokkien rajapinnat ja tärkeimmät funktiot määriteltiin MVC-arkkitehtuuria tukeviksi.

View

MainWindow-luokka vastaa datan esittämiseen graafisessa käyttöliittymässä sekä käyttäjän komentojen vastaanottamisesta. Datan esittämiseen käytetään hyväksi Qt-frameworkin omia tietotyyppejä, joiden avulla UI-elementit voidaan ohjeistaa päivittymään aina jos niille annetun muuttujan sisältö muuttuu. Luokka käsittelee pelkästään MainProgram-luokan muuttujia ja funktioita, jotka vastaavat ohjelman logiikasta ja tietojen tallentamisesta ohjelman ajamisen aikana. MainWindow-luokka vastaa siis enimmäkseen omien UI-elementtiensä logiikan yhdistämisestä MainProgram-luokan funktioihin ja muuttujiin, jotta data voidaan visualisoida ja käyttäjän komentoihin voidaan reagoida halutulla tavalla.

Controller

Main-luokka toimii rajapintana käyttöliittymän ja tallennetun datan välillä. Muiden luokkien instanssit (MainProgram&MainWindow) alustetaan ohjelman alkuvaiheessa Main-luokassa, ja myös yhdistetään kuuntelemaaan toistensa signaaleja. Main-luokka osaa käsitellä luokkia MainWindow sekä MainProgram, jotka vastaavat muiden luokkien tarkemmasta käsittelystä.
MainProgram-luokka pitää sisällään ohjelman tärkeimmän logiikan. Luokan tärkeimmät julkiset muuttujat ovat grid (Sanajahdin kirjaimet) sekä words (ratkaistut sanat kirjainten perusteella), jotka UI-luokka näyttää käyttöliittymässä. Niiden muokkaaminen MainProgram-luokassa päivittää käyttöliittymän näyttämät tiedot.
MainProgram-luokan tärkeimmät julkiset funktiot ovat solveGrid, joka ratkaisee ja näyttää annetun ruudukon sekä makeGrid, joka täyttää annetun tiedoston perusteella ohjelman kirjainruudukon. Tekstitiedosto voidaan lukea ja muuttaa oikeaan formaattiin grid-muuttujaan, kun taas kuvatiedosto käsitellään ImageReader-luokassa grid-muuttujan täyttämiseksi. Toisaalta solveGrid on yhteydessä aiemmin luotuun instanssiin Solver-luokasta, jolle annetaan kirjainruudukko, jonka avulla se täyttää MainProgramin words-muuttujaan löydetyt sanat.

Solver-luokka on vastuussa ratkaistun sanalistan tuottamisesta sanajahti-kirjainten perusteella. Sen tärkein julkinen funktio on solve, joka ottaa kirjainruudukon parametrina ja palauttaa ratkaistun sanalistan MainProgram-luokan käytettäväksi.
Solverin algoritmi lukee ja tallentaa ensin koko sanakirjan, josta poistetaan kaikki ne sanat, jotka sisältävät kirjaimia, joita ei löydy Sanajahtiruudukosta. Tämän jälkeen tallennetaan std::map tietorakenteeseen avain - arvo pareina avaimiksi ruudukon kirjaimet ja arvoiksi kyseisen kirjaimen koordinaatit. Solve funktio alkaa yksi kerrallaan käymään läpi sanakirjan jokaisen sanan, tutkien löytyykö sitä Sanajahtiruudukosta. Sanan tutkiminen aloitetaan

katsomalla kaikki mahdolliset lähtökohdat aikaisemmin määritellyn avain - arvo tietorakenteen avulla, mistä syvyyshaulla lähdetään etsimään sanaa lähtökirjaim(i)en ympäriltä. Tällä algoritmilla aikakompleksisuus on likimain lineaarinen Sanajahtiruudukon koon suhteen, sillä isommalla peliruudukolla on lineaarisesti enemmän sanojen ensimmäisiä kirjaimia (tutkimisen lähtökohtia).

ImageReader on luokka, joka osaa etsiä kuvatiedostosta mahdollisen Sanajahtiruudukon. Se osaa lukea siihen kirjoitetut kirjaimet sekä täyttää niiden perusteella MainProgramin grid-muuttujan. Sen tärkein funktio on initData, joka ottaa parametriksi tiedostopolun kuvaan, jonka ruudukko halutaan muuttaa tekstiksi.

Model

Tärkeimpänä luokkana Model-kokonaisuudessa on Word-luokka, jonka tehtävänä on jäsentää ratkaisijan löytämät sanat ohjelman kokonaisuuden kannalta järkevään muotoon. Word-olio tietää sisältämänsä sanan sekä sanan kirjainten paikat ruudukolla. Luokka perii Qt:n QTreeWidgetItem-luokan, jonka avulla Word-oliot voidaan näyttää käyttöliittymässä QTreeWidget-elementin avulla.

Model-kokonaisuuteen kuuluu myös logiikka ja rakenne tekstitiedostoille, joita käytetään sanalistojen ja Sanajahtiruudukon tallentamiseen. Tietojen tallentamiseen käytetään ulkopuolisia tekstitiedostoja, jotta tietojen tallentaminen on mahdollista myös ohjelman monien käyttökertojen välissä. Käyttäjä voi myös helposti tuottaa itse omia sanalistoja tai Sanajahtiruudukoita tekstitiedostoon, jotka ohjelma osaa käsitellä.

Ohjelman logiikka ja tärkeimmät funktioiden rajapinnat

Tärkeimpinä rajapintoina ohjelmassa on UI:n (MainWindow-luokka) sekä pääohjelman (MainProgram-luokka) funktiot, jotka ovat sidottu reagoimaan toisiinsa Qt-frameworkin signaleita ja slotteja käyttäen. UI:n tärkeimmät signaalit ovat:
void fileDropped(QString filePath), joka ilmoittaa pääohjelmalle uuden tiedoston avaamisesta (File to Grid tai drag&drop-tiedosto).
void requestSolve(QVector<QVector<QChar> > currentGrid), joka pyytää MainProgramilta ratkaisujen etsimistä nykyisen ruudukon perusteella (Esimerkiksi käyttäjän syöttämä data).

MainProgram kuuntelee mainittuja signaaleja ja toteuttaa UI:n pyynnöt sanojen etsimisen ja päivittämisen suhteen. fileDropped-signaalin yhteydessä MainProgram katsoo onko annettu tiedosto .txt vai .png/.jpg. Kuvatiedostot välitetään Solver-luokalle, joka tunnistaa niistä ruudukon kirjaimet TesseractOCR:n avulla (ImageReader::initData) ja päivittää MainProgramiin löydetyt kirjaimet ratkaisua varten. Tekstitiedostot MainProgram lukee itse ja päivittää ruudukon niiden perusteella. MainProgram ilmoittaa UI:lle olevansa valmis

void gridUpdated(bool) -signaalilla, jonka parametri kertoo UI:lle löytyikö ratkaisuja annetulla ruudukolla vai ei. Tämän avulla UI osaa päivittää ruudukon ja/tai löydetyt sanat käyttäjän käsiteltäväksi.

Ohjeita projektin kääntämiseen
Ohjelma tarvitsee kääntyäkseen Qt-frameworkin (https://www.qt.io/download/) ja muutaman kirjaston. Kaikki tarvittavat kirjastot löytyvät valmiina paketteina Ubuntulle ja sen johdannaisille:
sudo apt-get install libgl1-mesa-dev libtesseract-dev tesseract-ocr-deu libleptonica-dev

Kääntäminen ja ohjelman suorittaminen:
cd src/
qmake
make
./SanajahtiSolver

Projekti on kehitetty GCC 4.8.4:llä ja Qt 5.5:llä, aikaisempia versioita ei ole kokeiltu.

Basic user guide
Ohjelman suunnittelussa on mietitty kahta erilaista käyttötapausta. Tärkeimpänä käyttötapauksena ohjelmalle valittiin Sanajahti-pelin reaaliaikainen ratkominen ja avustaminen käyttäjän pelatessa Sanajahtia mobiililaitteellaan. Tässä tapauksessa käyttäjä avaa ohjelman ja valmistelee halutessaan Settings-välilehdeltä omat asetuksensa. Käyttäjä avaa Sanajahti-mobiilisovelluksen ja valmistautuu kirjoittamaan sovelluksen ristikon “Input to Grid”-painiketta painamalla. Käyttäjä kopioi sanajahtiruudukon ohjelmalle, jonka jälkeen ohjelma ratkaisee ruudukon.

Käyttäjä pystyy toistamaan ohjelman näyttämät sanat Sanajahti-sovellukselle. Oletuksena ruudukon täytettyä Input to Grid -toiminnolla ohjelma alkaa näyttämään löydettyjä sanoja käyttäjän tahtiin. Käyttäjä voi painaa välilyöntiä seuraavaan sanaan siirtymiseksi. On myös kuitenkin mahdollista laittaa ohjelma toistamaan sanoja automaattisesti asetuksista valitulla viiveellä painamalla Play-painiketta. Käyttäjä voi pelata erän Sanajahtia, ja aloittaa seuraavan erän painamalla Input to Grid -painiketta uudestaan. Asetuksia sanojen näyttämisestä on helppo muuttaa erien välisen tauon aikana Settings-välilehdeltä.

Toisena käyttötapauksena kuviteltiin käyttäjä, joka haluaa tarkistaa haluamastaan ristikosta kaikki mahdolliset sanat. Tällöin käyttäjä on joko kopioinut ristikon tiedot esimerkiksi tekstitiedostoon tai ottanut kuvakaappauksen Sanajahti-sovelluksesta. Käyttäjä voi siirtää tallentamansa ruudukon ohjelmalle File to Grid -toiminnolla (avaa käyttöjärjestelmän oman tiedostoselaimen) tai pudottamalla tiedoston ohjelmalle (Drag&Drop). Ohjelma ratkaisee itsestään annetun ruudukon sekä siitä löytyvät kirjaimet, jonka jälkeen käyttäjä voi selata löydettyjä sanoja vasemmalla olevan listan avulla. Käyttäjä voi halutessaan piilottaa tietyn mittaiset sanat tuplaklikkaamalla “n letters” otsikkoa listalla, tai navigoimalla siihen ja painamalla nuolinäppäintä vasemmalle. Käyttäjä voi myös muuttaa antamaansa ristikkoa klikkaamalla kirjaimesta ja kirjoittamalla siihen uuden kirjaimen.

Moduulien testaus

Tekstintunnistusta testattiin empiirisesti riittävän suurella määrällä 1536x2048 pikselin kokoisia Sanajahti-ruudunkaappauksia, että voitiin olla varmoja jokaisen aakkosen oikeasta tunnistuksesta.

Algoritmin toimivuutta testattiin empiirisesti Sanajahti-peliä pelaamalla. Löydetyt sanat perustuvat ohjelmalle annettuun sanalistaan, johon tehtiin tarkennuksia pelattujen pelien perusteella. Osa löydetyistä sanoista ei ole Sanajahti-pelin hyväksymiä, jolloin sanalistasta poistettiin virheellisiä sanoja. Kokeilujen perusteella ohjelma löytää jokaisesta ruudukosta kaikki sanat, jotka ovat sille annetussa sanalistassa. Sanalistaa ei saatu pelin kannalta täydelliseksi.

Työnjako

GUI&Luokat: Matti

Build: Lassi

Solver: Timo

Konenäkö: Kaisa ja Otto

Main ja Grid: Timo ja Lassi

Työtunnit viikoittain

	Viikko 45	Viikko 46	Viikko 47	Viikko 48	Viikko 49	Viikko 50
Matti	5	5	10	6	20	26
Lassi	15	0	5	4	25	20
Timo	0	0	15	2	22	18
Kaisa	5	3	4	3	16	4
Otto	4	0	2	4	18	6