Sanajahti-solver
Projektisuunnitelma

Ryhmä: Sanajahti3

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

Yleiskuvaus

Projektin tavoitteenamme on luoda ohjelma, joka auttaa käyttäjää huijaamaan Sanajahti -pelissä, luoden täten lyömättömän taktisen edun muita kanssapelaajia vastaan. Ohjelma ratkaisee Sanajahti -pelistä löytyvän 4x4 kirjainruudukon (ohjelma tukee myös muiden kokoisten ruudukkojen tutkimista), etsien kaikki mahdolliset kirjainjonoista muodostuvat sanat.

Projektin sisäisenä tavoitteenamme on saada aikaiseksi Expert -tason työ. Ohjelman toiminnallisuuksiin tulee siis tärkeimpinä kuulumaan:

- nopea ja joustava ratkaisija
- helppokäyttöinen graafinen UI
- konenäöllä toteutettu ruudukon lukeminen

Olemme aikatauluttaneet projektin niin, että saamme mahdollisimman varmasti työn kaikki osa-alueet toteutettua. Työosuudet jaoimme siten, että työn luonne vastaa mahdollisimman hyvin jokaisen ryhmän jäsenen omaa osaamisaluetta.

Käyttötapaukset ja käyttöliittymä

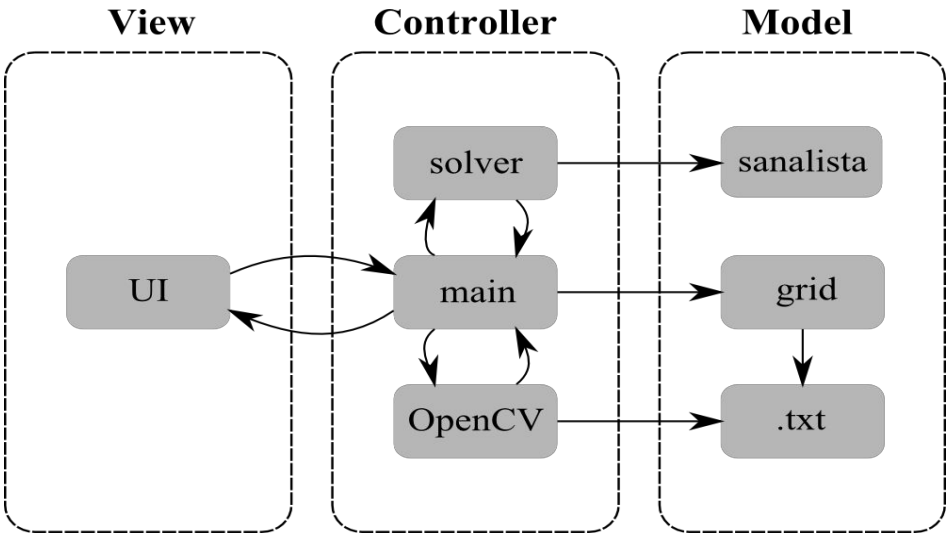
Käyttäjä pelaa Sanajahtia ja haluaa saada apua pelin voittamiseksi, joten hän avaa Sanajahti Solverin. Käyttäjä ottaa kuvan sanajahtiruudukosta ja raahaa/pudottaa kuvan Solverin päälle (Drag&Drop). Ohjelma automaattisesti etsii kuvasta ruudukon ja tulkkaa sen tekstiksi, jonka jälkeen se ratkaisee ruudukosta löytyvät sanat käyttäjälle ja näyttää ne listattuna. Käyttäjä painaa “Play”-painiketta, jonka jälkeen Solver alkaa näyttää ruudukosta löytyviä sanoja yksi kerrallaan automaattisesti, jotta käyttäjä voi samalla valita pelissä ratkaisut sana kerrallaan. Käyttäjä voi painaa “Pause”-painiketta, ja laittaa automaattisen toiston pauselle ja vaihtaa sanoja manuaalisesti “Previous”- ja ”Next”-painikkeilla tai klikkaamalla haluamaansa sanaa listasta. Sanalistan järjestystä voi muuttaa käyttöliittymästä sanan pituuden mukaan suurimmasta pienimpään tai päinvastoin.

Toisessa käyttötapauksessa käyttäjä ei ohjelmaa käyttäessään pelaa Sanajahtia, vaan haluaa tarkastella ruudukosta löytyviä sanoja ajan kanssa. Käyttäjä voi luoda uuden ruudukon “File to Grid”-painikkeella ja valita käyttöjärjestelmän omalla tiedostoselaimella joko haluamansa tekstitiedoston tai kuvan jonka perusteella ruudukko luodaan käyttöliittymään. Käyttäjä voi myös klikata jotain käyttöliittymän ruudukon kirjaimista ja korvata sen painamalla näppäimistöstä haluamaansa kirjainta, tai vaihtoehtoisesti painaa “Input to Grid”-painiketta, joka käy käyttöliittymän ruudukon läpi

vasemmalta oikealle, ylhäältä alas vaihtaen ruutua käyttäjän painettua kirjainta näppäimistöllään.

Käyttöliittymä mahdollistaa myös asetusten muuttamisen ohjelmassa. Käyttäjä voi painaa “Settings”-painiketta, josta hän voi muuttaa sanalistaa, jonka perusteella ohjelma etsii ratkaisuja ruudukosta. Jatkokehityksessä ohjelmaan voisi implementoida myös muita asetuksia, kuten “Autoplay”-ominaisuuden nopeuden muuttamisen.

Ohjelman rakennesuunnitelma



Projektissa on päätetty käytettävän kehitysympäristönä Qt:tä ja erillistä OpenCV-kirjastoa. OpenCV:n lisäksi tarvitaan Tesseract OCR -ohjelmistoa, joka toimii apuvälineenä tekstintunnistuksessa. Graafinen käyttöliittymä (UI-luokka) toteutetaan käyttäen Qt-kehitysympäristöä. Qt:n avulla on helppo luoda käyttöliittymästä käyttäjäystävällisempi ja esteettisempi versio. Kuvantunnistuksessa puolestaan käytetään apuna OpenCV-kirjastoa ja Tesseract-ohjelmistoa, jotka mahdollistavat kuvatiedostosta ruudukon tunnistamisen ja siten myös kirjainten erottamisen kuvasta.

Projekti toteutetaan noudattamalla MVC-arkkitehtuuria (model-view-controller), jossa ohjelma jaetaan kolmeen osaan: malli (model), näkymä (view) ja käsittelijä (controller). Ohjelman osa-alueet toteutetaan erillisenä komponentteina, joiden toteutukset ovat toisistaan riippumattomia. Tämä mahdollistaa 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 voidaan siis toteuttaa samanaikaisesti, kunhan niiden välissä oleva käsittelijä on suunniteltu kokonaisuuden kannalta toimivaksi.

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

View
UI-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 Main-luokan muuttujia ja funktioita, jotka vastaavat ohjelman logiikasta ja tietojen tallentamisesta ohjelman ajamisen aikana. UI-luokka vastaa siis enimmäkseen omien UI-elementtiensä logiikan yhdistämisestä Main-luokan funktioihin ja muuttujiin.

Controller
Main-luokka toimii rajapintana käyttöliittymän ja tallennetun datan välillä. Main-luokka osaa käsitellä luokkia UI, Solver ja OpenCV. Luokan tärkeimmät julkiset muuttujat ovat grid (sanajahdin kirjaimet) sekä words (ratkaistut sanat kirjainten perusteella), jotka UI-luokka näyttää käyttöliittymässä. Niiden muokkaaminen Main-luokassa päivittää käyttöliittymän näyttämät tiedot.
Main-luokan tärkeimmät julkiset funktiot ovat solveGrid, joka ratkaisee ja näyttää annetun ruudukon sekä overloadattu makeGrid, joka tuottaa annetun datan perusteella uuden Grid-olion. Mainittu makeGrid luo uuden Grid-olion erilaisen datan perusteella. Se voi olla joko parametriton tai se ottaa parametrikseen joko tiedostopolun tekstitiedostoon tai kuvaan. Tekstitiedosto voidaan lukea ja muuttaa oikeaan formaatiin Grid-olion luomiseksi, kun taas kuvatiedosto käsitellään OpenCV-luokassa Grid-olion luomiseksi. Grid-olio voidaan tehdä myös suoraan käyttäjän syötettä lukien. Toisaalta solveGrid on yhteydessä aiemmin luotuun instanssiin Solver-luokasta, joka ottaa ratkaisee Grid-olion avulla sanalistan Main-luokan käytettäväksi.

Solver-luokka on vastuussa ratkaistun sanalistan tuottamisesta sanajahti-kirjainten perusteella. Sen tärkein julkinen funktio on solve, joka ottaa Grid-olion parametrina ja palauttaa ratkaistun sanalistan Main-luokan käytettäväksi.

OpenCV on luokka, joka osaa etsiä kuvatiedostosta mahdollisen sanajahtiruudukon. Se osaa lukea siihen kirjoitetut kirjaimet sekä tuottaa niistä tekstitiedoston, jonka perusteella luodaan Grid-olio Main-luokan käsiteltäväksi. Sen tärkein funktio on readGrid, joka ottaa parametriksi tiedostopolun kuvaan, jonka ruudukko halutaan muuttaa tekstiksi.

Model
Tärkeimpänä luokkana Model-kokonaisuudessa on Grid-luokka, jonka tehtävänä on jäsentää sanajahtiruudukon kirjaimet ohjelman käytettäväksi tekstitiedostoa järkevämpään tietotyyppiin. Main-luokka osaa luoda Grid-olion käyttöliittymän ilmoittamilla paramatreilla (tekstitiedosto/kuva/käyttäjän syöte).

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.

Projektiaikataulu ja työnjako

Alustava aikataulu:

4.11. Projektin aloitus Kokous, päätetään työnjaosta, hahmotellaan projektin rakennetta ja aikataulua
6.11. Suunnitelman palautus
11.11. Build valmiina, Solver toimii Git käytössä, ryhmän jäsenillä Qt ja tarvittavat kirjastot käyttövalmiina, Solver optimointia vaille valmis.
15.11. Projektin rakenne ja perusteet valmiina Projektin kaikki luokat ovat ainakin jossain vaiheessa, ja meillä on toimiva ohjelma.
16.11. Checkpoint Assistentin tapaaminen, katsotaan mitä ollaan saatu aikaiseksi ja tarkennetaan myöhempää aikataulua sen mukaisesti.
24.11. Checkpoint 2 Tehdään tilannekatsaus projektista, päätetään mitä vielä ehdimme tehdä, muokataan tarpeen mukaan aikataulua, tavoitteita ja työnjakoa.
4.12. Raportti valmiina, ohjelma refatorointia ja debugaista vaille valmis
6.12. Projekti viimeistelty Lähdekoodi refaktoroitu ja tunnistetut bugit korjattu

Suunniteltu työnjako:

GUI: Matti
Build: Lassi
Solver: Timo
Konenäkö: Kaisa ja Otto
Main ja Grid: Timo ja Lassi