Tekijä: Egor Kovalenko

Opiskelija numero: 1032378

Koulutusohjelma: Teknistieteellinen kandidaatti ja diplomi-insinööri (Tietotekniikka)

Vuosikurssi: 2022 (2021?)

Päiväys: 26.04.2023

Yleiskuvaus:

Projektin tarkoitus on ollut tehdä diagrammeja luova/käsittelevä ohjelma. Projekti pohjautuu THL-sivuston tarjoamaan Suomen koronatilannetta kuvaavaan dataan. Ohjelma mahdollistaisi saatavan datan visualisoinnin/tallentamisen, jälkikäteisen analysoinnin. Lyhyesti sanottuna ohjelman tarkoitus on olla niin sanotusti Excelin pikkuveli. Projektia luodessa monet suunnitelmat muuttuivat, mutta perimmäinen tarkoitus on kuitenkin pysynyt samana ja toteutui. Ohjelmani osaa parsata dataa Circe frameworkin kautta, ja muuntaa se lukukelpoiseksi. Parsaaminen tosiaan luo yksittäisiä datapisteitä, mikä mahdollistaa datan tarkan seurannan datapisteiden värien muuttamisen, kommenttien lisäämisen jne. Ohjelma tarjoaa mahdollisuuden käsitellä yksittäistä datapistettä monien mittareiden kautta, kuitenkin riippuen paikasta, sillä tietoturvasyistä, THL ei voi antaa tiettyihin paikkoihin ulottuvaa dataa. Esimerkiksi kunnista ja sairaanhoitopiireistä ei saa tietoja liittyen kuolemantapauksiin, kun taas koko Suomen alueesta niitä saa. Ohjelma osaa analysoida avattua diagrammia neljän kriteerin kautta: keskihajonta, summa, maksimi ja minimipointti. Analysointi tapahtuu käyttäjän valitsemalla intervallilla ja paikoilla. Diagrammin muodostuminen perustuu viiteen tarjoamaan mittariin (asukasmäärän lukuun ottamatta) (kuva 1), joista on tarkoitus valita kuitenkin vain yksi. Kaikki muu mahdollinen data tulee kuitenkin mukaan Additional infona itse datapisteeseen. Käyttäjä voi valita paikat kunnittain, sairaanhoitopiireittäin tai kaikki alueet yhteensä. Diagrammeja on kolmea eri tyyppiä: ympyräkaavio, viivakaavio ja pylväsdiagrammi. Niiden toiminta poikkeaa hieman toisistaan, mutta kokonaisuudessaan ne antavat samaa dataa (kuvat 2–4). Ohjelmassa on toteutettu Tallentaja ja fileenlukijat, jotka ottavat mukaan kaikki käyttäjän tekemät muutokset ja tallentavat sekä kuvaavat ne, kun file avataan uudestaan. Yksittäisinä pisteinä pidetään pylväät pylväsdiagrammissa, ympyrän segmentit ympyräkaaviossa ja pisteet viivakaaviossa. Kuvista voi myös huomata merkinnät, jotka lyhyesti kuvaavat datapisteitä. Esimerkiksi oranssi väri pylväsdiagrammissa viittaa viikkoon 100 (viikkojen indeksien tulkinta on mahdollista saada selville dokumentin käyttöohjeesta tai painamalla setting -> help ohjelman sisällä), ympyräkaaviossa värillä kuvataan yhtä aluetta ja siitä annetaan kuvaus muodossa (paikka – viikko – mittarin antama arvo) -> (Helsinki – 109–4763). Työni taso on mielestäni haastava, sillä olen toteuttanut kaikki siellä ja keskivaikean vaatimuksissa olevat kriteerit paitsi Datapisteiden / komponenttien valinta suorakulmion valintatyökalulla.

Käyttöohjeet:

Ohjelman käynnistettyä käyttäjälle aukeaa graafinen liittymä, jonka yläpalkissa näkyvät seuraavat kohdat:

- file
- settings
- update
- compare
- analyze

Kun ohjelmaa ajetaan ensimmäistä kertaa, tulee käyttäjän painaa file -> New, minkä jälkeen näytölle ilmestyy Creation menu. Creation menussa käyttäjän on täytettävä kriteerit, joiden mukaan luodaan diagrammi (kuva 5). Ensin valitaan intervalli 0–208 (joka muuttuu oletettavasti vuonna 2024, sillä viikko 209 kuvaa viimeistä vuoden 2023 viikkoa). Intervalli ei ole pysyvä ja jokaisen käynnistyksen jälkeen päivittyy automaattisesti. Intervalli valinnan jälkeen tulee käyttäjän valita paikka, joka voi olla Sairaanhoitopiiri, Koko Suomi (Kaikki Alueet tai Regions) tai kunta, joka kuuluu yllä valittuun Sairaanhoitopiiriin. Paikan valinnan jälkeen ohjelma päivittää valittavien mittareiden ikkuna, josta voidaan valita sopivin. Lopuksi pitää vielä valita diagrammin tyyppi valintalaatikosta, jonka yllä lukee “Choose type of Diagram”. Kun valinta on tehty, tiedot on lisättävä painikkeella “Add”, joka sijaitsee vasemmassa alalaidassa. Tämän jälkeen voi joko heti luoda diagrammi tai lisätä muu kiinnostava paikka. Kuitenkin silloin on toteutettava seuraavat ehdot:

1) Mittarin ja diagrammin tyypin on oltava samat kuin sen olion/paikan, joka oli ensimmäisenä lisätty
2) N:n paikan intervallin arvo pitää olla sellainen, että sen yksi piste joko sisältyy johonkin aikaisemmin lisättyyn intervalliin, tai että se kaataa kaikki aikaisemmin lisätyt intervallit (sisältä kaikki taulussa jo olevat intervallit).

Kun nämä ehdot toteutuvat toimii “Add”-näppäin oikein ja lisää kyseisen paikan listaan, muuten se ei tee mitään. Kun paikat on lisätty listaan “Create Diagram” nappulan kautta voi luoda diagrammi, joka tulee keskelle ikkuna. Ikkunaa voi säätää ja diagrammiobjektit silloin säädetään automaattisesti. Jos diagrammin arvoja haluaa vaihtaa tässä vaiheessa, voidaan sen edelleen tehdä creation menu ikkunassa. Mikäli se on jo laitettu kiinni, voi paina ylälaidassa oleva “update”-> “Update” nappula, jolloin näytölle tulee takaisin Creative stage – ikkuna, jolla on täysin samat kyvyt.

“Remove” painikkeella voi poistaa taulusta valittu paikka. Jos jokin paikka poistetaan siinä vaiheessa, kun diagrammi on jo luotu, tulee toisen kerran painaa “Create diagram”, jonka jälkeen diagrammi päivittyy.

Kun diagrammi on luotu, voi datapisteitä analysoida tarkemminkin. Jokainen datapiste on olio, ja kun siitä painetaan, tulee näytölle ikkuna, jossa on lyhyesti selostettu paikan ja sitä vastaavan viikon tiedot (kuva 6). Kuvassa 6 näkyy data ja pari nappulaa sekä tekstikenttä. Tekstikenttään voi kirjoittaa kommentteja ja sitten tallentaa ne painamalla “save” nappula. “color” - nappulaa painamalla näkyy diagrammissa, että valitun paikan väri muuttuu. “get comments” nappulaa painamalla vastaavasti näkyy kaikki kommentit, jotka käyttäjä on aikaisemin lisännyt.

Tässä vaiheessa, kun diagrammi on luotu ja käsitelty, sitä voi analysoida “analyze” painikkeen kautta, joka sijaitsee ylälaidassa oikealla. Kun painetaan “analyze” -> “Analyze”, tulee näytölle ilmoitus siitä, että analyysi tapahtuu ainoastaan tällä hetkellä olevan diagrammin kanssa. Eli jos Analyze ikkunan avaamisen jälkeen diagrammiin tehdään joitain muutoksia, tulee analyze tool avata uudelleen. Muutoin analyysi tehdään koko ajan avalyze-ikkunan avauksen aikana olemassa olevan diagrammin ja sen datan mukaan. Analyze-tool antaa laskea diagrammin tiettyyn aikaväliin kuuluvien komponenttien eri arvoja. Esimerkkinä: keskihajonta, keskiarvo, mittarin mukainen summa tietystä intervallista, max ja min arvo.

Analyze-toolia käyttäessä pitää ensin valita mittari, jonka kautta tehdään analyysi. Ohjelma tarjoaa vain ne mittarit, jotka sisältyvät diagrammiin. Eli jos analysoitavaksi oli valittu vain kunta, tulee 1 mittari käsittelyyn. Tämän jälkeen valitaan aukeaa kriteeri-ikkuna, josta voi valita kiinnostavin kohde. Seuraavaksi käyttäjän tulee valita oikealla olevasta laatikosta, jonka vieressä lukee “Choose place” ne paikat, joita hän haluaisi analysoida ja yksitellen lisätä ne “Add place” painikkeella, jonka jälkeen voi painaa "create statistic stage”, joka puolestaan luo näkyville ikkunan, josta näkyy kyseinen laskutulos ja tiedot siitä.

Kun kaikki aikaisemmat toiminnot on tehty ja on aika tallentaa file johonkin. Tulee käyttäjän paina file -> save ja kirjoittaa tiedoston nimi, johon se haluaa tallentaa fileen. Tiedosto on txt muodossa, mutta kenttään (kuva 8) sitä ei tarvitse kirjoittaa, muuten ohjelma tallentaa tiedoston muotoon “nimi.txt.txt” mikä ei periaatteessa haittaa, sillä tiedosto on edelleen lukukelpoinen. Näin ei kuitenkaan suositella tehtäväksi. Kuvassa 8 näkyy millä nimellä kannattaa tallentaa tiedosto. HUOMAUTUS! jos kuvan 8 tekstikenttään kirjoittaa jo olemassa olevan fileen nimen. Fileen aikaisempi data menee hukkaan ja sen tilalle tulee se data, jota sillä hetkellä tallennetaan. Fileen nimiä voi varmuuden vuoksi tarkistaa “SavedTxtFiles” kansiosta, joka sijaitsee projektin juuressa. Jos käyttäjä antaa virheellisen nimen, kuten “\23jrt??//.wuihf.html.txt\\//” tallennettavan diagrammin dataa tallentuu automaattisesti “out.txt” nimiseen tiedostoon. Nimi on sen takia virheellinen, koska siinä on “/” merkkejä, joiden on tarkoitus täsmentää fileen polkua, eivät johda mihinkään kuin paikkaan, jossa ei ole mitään (mahdollisesti).

Kun uusi tiedosto on luotu ja käyttäjä lähtee hörppimään teetä, mutta muistaa yhtäkkiä unohtaneensa vaihtaa Helsingin ja Uuden maan SHP:n viikon 109: väriä vihreäksi ja kirjoittaa kommentin “I love butterflies” hän voi avata tallennetun tiedoston, kirjoittaen sen (tiedoston, joka toki voi olla sama kuin käyttäjän nimi tarvittaessa”) nimen JA .txt loppuun. Tämä on tärkeää, sillä muuten tiedosto ei aukea (kuva 9).

Tässä vaiheessa käyttäjä voi mahdollisesti haluta lisätä jonkin paikan diagrammiin. Toisin sanoen muokkailla diagrammia jälkikäteen. Se on mahdollista kuten on jo aikaisemmin mainittu “update”-painikkeella, josta aukeaa ikkuna, joka on muodoltaan sama kuin Creation Stage ja sisältää jo dataa, jota tällä hetkellä käsitellään (näkyy ruudussa)

ToolLaidasta huomaa myös “compare”- nappulan. Sen tarkoitus on luoda erilliset Stageit, jotka sisältävät käyttäjän määräämät diagrammit muissa ikkunoissa kuin PrimeStage-ikkunassa. Tämä työkalu mahdollista monien diagrammien avaaminen samaan aikaan ja monipuolistaa datan analysointia.

TÄRKEÄ HUOMAUTUS!

Aina kun painetaan file -> new analysoitava data / diagrammit menevät hukkaan ja kaikki update-ikkunat menevät myös kiinni. Tämä johtuu mahdollisista virhetilanteista, jotka muuten voisivat tulla, jos sitä ei tehtäisi. Tätä voisi pitää shortcuttina uuden diagrammin luomiseen, sen sijaan, et poistettaisiin kaikki paikat käsin update ikkunassa.

Lopuksi mainitsemisen arvoinen on vielä se fakta, että settings-painikkeessa (ylälaidassa) on kaksi nappula "Change axis/chart's name". Niiden kautta voi vaihtaa diagrammin / akselin nimen. Jos tämän jälkeen diagrammi tallennetaan johonkin tiedostoon ja avataan uudelleen aikaisempien ohjeiden mukaan, kaikki nimet säilyvät, ja näkyvät avatussa diagrammissa.

Mittarit

- 1 - **COVID-19 -tapausten lukumäärä**
- 2 - **COVID-19 testausmäärät**
- 3 - **Koronaan ajallisesti liittyvät kuolemat (30 vrk), Tartuntatautirekisteri**
- 4 - **Koronasta johtuvat kuolemat, kuolintodistus (alustava tieto)**
- 5 – **Kuolemat, joissa korona myötävaikuttavana tekijänä, kuolintodistus (alustava tieto)**

Ohjelman rakenne:

Ohjelma on jaettu viiteen kokonaisuuteen – Datan parsaaminen ja lukukelpoiseksi muuntaminen, Graafinen käyttöliittymä, Diagrammit, Datan tallentaminen ja tiedostosta lukeminen, jälkikätinen muokkaaminen.

Liitteenä kuva 11.

1) Datan lukeminen ja parsaaminen on pääsääntöisesti toteutettu luokassa DataReader. Koko ohjelmassa tästä luokasta on luotu vain yksi olio, mikä viitaa siihen, että luokan sijaan DataReader olisi tosiaan voinut hyvinkin olla objekti tai toimia rakenteella kumppani-olio case class luokan kanssa. Tämä selvisi minulle vasta projektin loppuvaiheessa, sillä aikaisemmin oletin, että aina dataa parsattaessa tullaan tarvitsemaan erillinen Readerin olio. Tämä ei kuitenkaan vaikuta mitenkään projektin toimintaan, vaan on lähinnä huomautus itselleni. Datareader siis parsaa dataa THL:n sivustolta ja luo DataPoint olioita, jotka kuvaavat yksittäisiä pisteitä diagrammissa. DataReader:llä on kolme olennaisinta metodia kunta_Creator, SHP_creator_week kaikki_alueet_creator, jotka nimensä mukaisesti luovat datapisteitä tietylle paikalle, tietylle aikavälille. Kaikki nämä kolme metodia palauttavat Array[Datapoint], niin että voi varmistaa, että tuleva käsittely ei muuta Arrayn kokoa. DataReaderin toinen oleellisin metodi on kuntaData. KuntaData toimii parissa kahden muun rakenteen kanssa. Niistä toinen on DataReader luokassa esiintyvä object CirceUtil, jonka kautta voi muuttaa tulevan JSON-datan kenttien nimiä. Tämä on tosi tärkeää toteutuksen kannalta, koska suurena ongelmana oli alussa se, että dataoliot sisältävät kentän, jonka nimi on "class". Miksi se on ongelma? Sen takia, että scalan konstruktoriparametrin nimeksi ei voi asettaa class, koska se on scalakielen syntaksi, joka kuvaa luokan luomista, ja sitä vastaan taisteltiin nimenomaan objektin CirceUtil:n ainoalla "renameField" metodilla. Toinen niistä Parsers-packagessa oleva Kunta case class luokka, joka on toteutettu tavalla, että se nimestään huolimatta

kaataa myös SHP ja Kaikkien alueiden data-olioita, jotka ovat saatavilla THL:n sivustolla. KuntaData parsaa THL:n tarjoamaa JSON-dataa ja luo Kunta objekti, jota sitten käsitellään kunta_Creator, SHP_creator_week, kaikki_alueet_creator, metodeissa.

2) Diagrammit. Diagrammeja on kolmea tyyppiä ja ne perivät StatisticBoardPoints piirteen, joka vaatii konstruktoriksi Array[DataPoints] diagrammin muodostamiseen. Diagrammien tyypit/nimet: ViivaKaavio,Circle, Pylväs. Nimiä olisi tosiaan hyvä vähän muokata, mutta mennään kuitenkin niillä. Kaikki diagrammit sisältävät luomisen kannalta neljä tärkeintä metodia: makePic, color_in, vaihtoehtoisesti myös apu_legend_colorChange ja XYchartCreator. makePic ei itsessään vielä tuota mitään kuvaa, vaan se antaa diagrammissa olevan chartin muuttujualle arvon, jota käyttävät muut metodit. makePic myös muuttaa legend-taulun värejä pieChartissa(Circle-class) selkeyttämisen vuoksi. color_in metodia käytetään, kun luetaan tallennettua tiedostoa ja muunnetaan se diagrammiksi. Ensin ideana oli initialisoida värit jo luomisen vaiheessa, mutta osoittautui, että siinä tarvittava getNode metodi ei toimi siihen asti, kunnes luodaan Chart loppuun ja sen takia, initialisointi tapahtuu vasta diagrammin luomisen jälkeen color_in:n kautta. Aikaisemmin mainittu XYchartCreator luo XYchartSeries ja XYDatapoints pylväs- ja viivakaavioissa, jotka kuvaavat tiettyjä datapisteitä ja niiden yhtenäisyyksiä. Esimerkiksi kaikki viikon 55 alueet ja niiden tapaukset. Kuvassa 2 XYcarSeries olisi esimerkiksi viikko 103 ja XYdatapoints olisivat Helsingin ja Uudenmaan SHP sekä niiden mittarien arvot. Jälkikätinen diagrammin muokkaaminen ja käsitteleminen on mahdollista metodien clicker, mover, color_change_of_piece, add_comment metodien kautta. Kun painetaan luodun diagrammin yksittäistä datapistettä, toimii clicker metodi, joka avaa kuvassa 6 olevan ikkunan. “mover”-nimisessä metodissa, hiiren liikkuessa tietyn datapisteen päälle. Tulee pisteen tiedot näkyville, ja samalla näkyy kuvassa 6 oikealla (tummateksti 8782 – Helsinki). Ja hiiren poistuessa, myös kirjoitus katoaa kuvan kyseisestä kohdasta. Color_change_of_piece toteuttaa kuvassa 6 näkyvän Color-buttonin toiminnan, eli vaihtaa datapisteen värin, kun painiketta painetaan. AddComment lisää kaikki kuvassa 6 tekstikentässä olevat tekstit. GetComments nappula palauttaa stagein, joka sisältää kaikki lisätyt kommentit. GetComments on yleismetodi, joka on kuvattu itse tratitissa.

3) Datan lukeminen tiedostosta ja tallentaminen tiedostoon tapahtuu itse GUI:ssä actionEventien kuvaamisessa (fileItems.onAction, SaveDiagram:onAction), mikä jälkikäteen ajatellen ei ole ehkä paras ratkaisu, vaan olisi kannattanut luoda joko metodit tai erillinen objekti sitä varten. Se säästäisi tilaa ja olisi helpommin muokattavissa tulevaisuudessa. Muuten lukeminen ja tallentaminen tapahtuu JSON-formaatissa uudelleen käyttäen circe-en semidecodereita/-encodereita ja dataParsers packagen FileToWroten case classia datan lukukelpoistamiseksi

4) 5) GUI. Graafinen käyttöliittymä on melkein täysin kuvattu GUI-objektissa, joka perii JFXApp3. GUI yhdistää periaatteessa kaikki ohjelman luokat ja toiminnot. Kuva tuotetaan overridetussa start metodissa, joka siis sisältää oleellisimman koodin. GUI-objektista muun muassa voi huomata Diagram case classin. Diagram kuvaa tulevaa, joka tulee luoduksi DataReaderin kautta. Näin Diagram sisältää paikan, sen tunnuksen, intervallin, mittarin ja muuta tämän kuvauksen kannalta ei niin oleellista dataa. Myös GUI sisältää Buffer[Diagram] bufferia, joka siis pitää kirjaa, sitä, mitä diagrammin on tarkoitus tällä hetkellä kuvata. Kun painetaan kuvan 5 Create diagram nappula tulee toimimaan ActionEventin kuvaaminen, joka lukee aikaisemmin mainittua bufferia ja suodattaa diagrammeja tavalla, jolla DataReader käyttää jokaista diagrammia kohtaan oikeaa metodia, jotka kuvattiin kohdassa 1. Näin esimerkiksi Diagram("Helsinki", "kunta", (1,100),1.....) :ia varten suodatin käyttäisi kunta_Creator metodia. GUI sisältää myös seuraavat rivit : diagramHimself, diagrams, rootPane, tabulaaaar, updateStage. DiagramHimself sisältää chartin kuvan, diagrams on nimenomaan se bufferi, josta oli puhetta aikaisemmin, rootPane, kuvaa ohjelman sijaintia propertiesillään: center, left, right, top, bottom. Näin esimerksi ylälaidassa olevat menu-painikkeet ovat laitettu rootPanein top:iin, tabulaar on talu, joka näkyy, kun luodaan diagrammia tai kun painetaan update. UpdateStage on tietysti kopio creationStageista, joka mahdollista jälkikätisen muokkaamisen ja on muun muassa toteutettu luokassa AfterWordMaintainer. Miksi ne on erikseen viety niin korkealle tasolle? Sen takia, että niitä voi käsitellä myöhemmin muissa olioissa. Näin esimerkiksi, kun painetaan "Update" luoduksi tulee AfterWordMaintainer olio, joka käyttää gettereita ja settereita luokasta GUI ja voi melkein suoraan muokata sen dataa. AfterWordMaintainer on periaatteessa kopio newStage:n toteutuksesta, mutta ilman TableView:n luomista.

Algoritmit:

DataReaderin SHP-creator_week käyttää yhtä algoritmia, joka perustuu THL:n datan kuvakseen. Mittareiden arvot riippuvat niiden sijoituksessa URL:ssä. Data menee sillä tavalla, että ensin kuvataan

Paikka – aika kaikkine mittareineen - aika2 kaikkine mittareineen - aika2 kaikkine mittareineen...

Paikka2 – aika kaikkine mittareineen - ….

Kun dataa parsataan, sieltä löytyy value niminen lista, jossa on kaikkien viikkojen määrä * kaikkien mittareiden määrä * kaikkien paikkojen määrä. Esimerkiksi on 3 kaupunkia 2 mittaria ja 5 viikkoa. Tällöin ensimmäisen kaupungin arvot löytyvät välistä 0 – 5 *2, toisen kaupungin dataa löytyy indekseillä (5 *2 +1) - 5*2, kolmannen kaupungin vastaavasti ((5 *2 +1) - 5*2) +1 – 5*2 jne. Paikkojen järjestys tosiaan selviää label nimisessä listasta. Kun yhdistetään tämä lasku kuntien järjestyksen aikaisemmin esitetyn laskun nojalla voi tietää, että joka toinen arvo vastaa mittaria yksi ja arvo + 1 vastaa mittaria kaksi. Tämä toiminta on toteutettu SHP_apu_creatorissa ja itse SHP_Creatorissa, sekä kunta_Creatorissa ja apu_kunta_creatorissa.

Analyze Stage sisältää metodit, suodattavat dataa ja muodostavat sen nojalla laskuarvot. Näin esimerkiksi keskihajonnan lasku tapahtuu sillä, että lasketaan ensin keskiarvo ottamalla kaikkien

intervalliin kuuluvien pisteiden arvot, laskemalla ne yhteen ja jakamalla niiden arvolla. Seuraavaksi jokaisesta arvosta lasketaan varianssi ottamalla pois jokaisesta arvosta keskiarvo ja korottamalla toiseen ja sitten summaamalla kaikki tällaiset arvot. Tämän jälkeen saatu arvo jaetaan pisteden määrällä - 1.

Color_change_of_piece kaikissa Statistic_board luokissa. Värin muuttaminen vaatii seuraavan väriListan elementtia, mutta mitä jos listan pituus on 10 ja seuraavan värin indeksi olisi 10? Color_change_of_piece tarkistaa, mikä väri tulee seuraavaksi, kun Color nappulaa kuvassa 6 on painettu ja jos se yltää listan pituuden, se hyppää listan alkuun ja lähtee uudelle kierrokselle

XY_chartCreator Pylväs ottaa diagrammista kaikki Array[DataPoint] olioiden nimet ja poistaa duplikaatteja. Jokaista nimeä käydään for cyclessa. Ensin se suodattaa Datapointit niin, että jäljelle jäävät vain, ne, joilla on sama nimi kuin for loopin arvolla. Tämän jälkeen se muodostaa XYSeries, joka ottaa nimeksi for:in arvo ja tekee suodatuista datapointeista yhtä paljon XYData, jotka, ottavat viikon ja viikkoa vastaavan mittariarvon mukaan. Viivakaaviossa on sama periaate, mutta XYSeries:n nimeksi tulee viikko ja XYDatan arvoiksi paikan nimi ja mittarin arvo.

Circlen makePic sisältää lopussa sellaisen foreach haaran, jossa taas otetaan dataPontsien nimet ja poistetaan duplikaatit. Jokaista paikan nimeä vastaa monta datapistettä, joilla kuitenkin on vain ainutlaatuinen viikko ja jokin arvo. Näin Array[Datapoint] suodattaa foreachissa olevan nimen mukaan ja sitten jokaiselle arvolle suodatetussa listassa annetaan sama väri. Tarkempi toiminta on kuvattu apu_legend_colorChange metodissa

Tietorakenteet:

Tiedostossa käytetään tarkoituksesta riippuen mitä erilaisimpia kokoelmatyyppejä. DataPoint:it esimerkiksi tallennetaan Array[DataPoints], jolla halutaan korostaa sen muuttumattomuutta. Bufferia käytetään dynaamiseen dataan, eli esimerkiksi jatkuvasti muuttuvaan Creation/Update ikkunan muutokset voivat olla kirjattu diagrams nimiseen bufferiin (Buffer[Diagram]). Scalafx:n GUI:n elementit kuten combobox,vbox,hbox,TableView,TableColumn käyttävät ObservableBuffereita tai ObservableArrayit tallennetaakseen dataa. En ole tutkinut, mihin niiden toiminta perustuu, mutta uskon, että new ovat Array:in ja Bufferin modifikaatiot. Comboboxit voivat ottaa listat initialisoinnissa, eli joskus voi heti uuteen comboboxin parametriksi laittaa List[Tyyppi], kun comboboxin arvoa määritellään ensin. Kaikki Chartit käyttivät ObservableNode tallentaakseen dataan. Muita tietorakenteita en joutunut luomaan itse.

Tiedosto ja verkossa oleva tieto:

Ohjelmani toimina periaatteessa perustuu datan lukemiseen internetistä. Data haetaan THL:n sivustolta .json formaatisssa ja sitä käsitellään circe frameworkin kautta. Riippuen siitä miten säädetään URL:ää datan muoto/luokka/tyyppi/määrä muuttuu. Näin voidaan valita erikseen kunnat/shpt/kaikki alueet/aikavälit koodinsa mukaan. Käyttäjän ei tarvitse huolehtia linkeistä, vaan ne ovat ennalta määritelty ja niitä voi tarkistaa DataReaderin muuttujista alussa, tai DataPointien luojissa. Tallennuksen vaiheessa diagrammien dataa muunnetaan JSON muotoon

kirjoittamani GUI:ssa sijaitsevien encodereiden kautta ja sitten muutetaan Stringiin, joka Javan BufferedWriterit tallentaa käyttäjän määräämään fileeseen (kuva 8). Jos haluaa suoraan itse kirjoittaa jokin txt tiedoston testaamiseen, sen pitää olla muotoa:


```
{
  "dg_type" : "Pie Chart",
  "measure_key" : "1",
  "in_start" : {
    "Helsingin ja Uudenmaan SHP" : 3,
    "Helsinki" : 2
  },
  "in_end" : {
    "Helsingin ja Uudenmaan SHP" : 44,
    "Helsinki" : 60
  },
  "add_mes" : {
    "Helsingin ja Uudenmaan SHP" : "",
    "Helsinki" : ""
  },
  "kunta_color_changes" : [
    {
      "name" : "Helsingin ja Uudenmaan SHP",
      "week_color" : {
        "14" : "-fx-background-color: #d8bfd855",
        "41" : "-fx-background-color: #2f4f4f"
      }
    },
    {
      "name" : "Helsinki",
      "week_color" : {
        "47" : "-fx-background-color: #fffacd",
        "58" : "-fx-background-color: #2f4f4f"
      }
    }
  ],
  "kunta_comment_changes" : [
  {
    "name" : "Helsingin ja Uudenmaan SHP",
    "week_comment" : [
      {
        "week" : 20,
        "comments" : [
          "wefwqf",
          "werh45t6uhj"
        ]
      }
    ]
```

```
  }
],
  "charName" : "Pie Chart",
  "axisName" : ""
}
```

Mutta lähinnä sanoisin, että sen pitää toteuttaa FileWrittenTo case classia ja sen kenttiä. Helpompaa olisi kuitenkin ensin itse luoda jokin diagrammi, tallentaa se ja tehdä siihen, joitakin muutoksia. Tämä kuitenkin voi vahingoittaa ohjelmaa, sillä jos tiedosto on kirjoitettu vääriin, niin ohjelma voi kaatua. Näin ei käy, jos siihen ei tee muutoksia ulkopuolelta.

Tässä vielä toinen esimerkillinen data:

```
{
  "dg_type" : "Line Chart",
  "measure_key" : "1",
  "in_start" : {
    "Kaikki Alueet" : 90,
    "Helsingin ja Uudenmaan SHP" : 95,
    "Espoo" : 92
  },
  "in_end" : {
    "Kaikki Alueet" : 110,
    "Helsingin ja Uudenmaan SHP" : 105,
    "Espoo" : 107
  },
  "add_mes" : {
    "Kaikki Alueet" : "",
    "Helsingin ja Uudenmaan SHP" : "",
    "Espoo" : ""
  },
  "kunta_color_changes" : [
  ],
  "kunta_comment_changes" : [
  ],
  "charName" : "Stacked Line Chart",
  "axisName" : "tapaukset"
}
```

Vielä esimerkki

```
{
  "dg_type" : "Bar Chart",
  "measure_key" : "1",
```

```
  "in_start" : {
    "Helsingin ja Uudenmaan SHP" : 2,
    "Lapin SHP" : 5,
    "Pohjois-Karjalan SHP" : 4
  },
  "in_end" : {
    "Helsingin ja Uudenmaan SHP" : 26,
    "Lapin SHP" : 53,
    "Pohjois-Karjalan SHP" : 7
  },
  "add_mes" : {
    "Helsingin ja Uudenmaan SHP" : "",
    "Lapin SHP" : "",
    "Pohjois-Karjalan SHP" : ""
  },
  "kunta_color_changes" : [
  ],
  "kunta_comment_changes" : [
  ],
  "charName" : "Stacked Line Chart",
  "axisName" : "tapaukset"
}
```

```
case class FileToWroten(dg_type : String,measure_key: String,in_start : Map[String,Int],in_end: Map[String,Int],
                        add_mes : Map[String, String],kunta_color_changes: List[Kunta_color_changes],
                        kunta_comment_changes: List[Kunta_comment_changes],charName: String,
                        axisName: String)

case class Kunta_color_changes(name: String, week_color: Map[Int,String])
case class Week_Comment(week: Int, comments: List[String])
case class Kunta_comment_changes(name: String, week_comment: List[Week_Comment])
```

Yllä esitetyssä esimerkeissä on attribuutit, jotka kuvaavat:

- Dg_type –diagrammin tyyppi
- Measure key – mittari, jonka kautta tehdään analyysi
- In_start/end - kuvaavat alku-loppuintervallia
- Add_mes kuvaa Additional meaasureita.”Tämä ei ole käytössä nyt”, mutta käsittelyn kannalta jätän sen tähän, sillä tulevaisuudessa haluaisin kehittää projektia ja tämä rivi on tarpeen silloin (voi käsitellä dataa useamman muun mittarin kautta, jos sen laatu muuttuu).

- kunta-color-changes sisältää listan Kunta_color_Changes objekteja, jotka kuvaavat yhtä paikkaa ja kaikille datapisteille tehtyjä värimuutoksia tietyillä viikoilla. Viikko on tulkittu avaimeksi Map[Int,String]:ssa. Yllä esitetyssä PieChart esimerkissä näkyy hyvin miltä muutettu data näyttää
- kunta_comment_changes sisältää listan Kunta_comment_changes olioita, jotka samalla tavalla kuin värien muutokset kuvaavat kommenttien muutoksia, PieChartin esimerkki kuvaa hyvin. Jokaista week-oliota vastaa kunnan nimi – name : String Week_Comment sisältää tuplesit, joissa on avaimena week: Int muuttuja ja kaikki siihen kuuluvat kommentit comments: List[String]
- CharName/axisName tallentavat tiedon chartin nimen tai akselin nimen muutoksesta.

Muutama sana THL:n datan lukemisesta. Viitteessä on neljä linkiä, joista kaksi ovat raa'at Jsonit ja toiset visualisoidut Jsonit. Jos haluaa tarkistaa niitä tarkemmin, onnistuu Postamnin kautta. Datan tarkoitus on vain olla luettu, siksi ainoat käytössä olevat välineet DataReaderissa ovat decodereita ja dataParsersin "Kunta" luokka, joka kuvaa samaa, mitä näkyy postmanissa tai suoraan avatussa linkissä. Dataa käsitelleessäni käytin useasti Hcursor, jonka kautta pystyin navigoimaan datassa. Eli jos halutaan päästä johonkin tiettyyn paikkaan, kuten esimerkiksi valuet: tauluun, joka sisältää tarvittavat mittariarvot, ei ole paikko luoda erillistä KuntaObjektia vaan voi operoida heti Hcursorin downfield:nimisellä metodilla, joka ottaa parametriksi muuttujan nimen ja oikeilla asetuksilla palauttaa tiedostosta tarvittavan kohdan. Sellainen metodi on esimerkiksi käytetty region_week_spotter metodissa, joka palauttaa dataa Suoraan JSonista, luomatta ScalaObjekteja.

Testaus:

Testaus poikkesi suunnitelmastani aika paljon. Päätin, että käytän aikaani lähinnä toiminnallisuuden lisäämiseen. Mutta esimerkiksi viimeisistä commitesista voi nähdä, kuinka paljon printlineitä minulla oli. Ohjelman toiminnallisuus on taattu Graafisella liittymällä. Olen tehnyt sen niin, että virhetilanteita vältetään jo silloin kun diagrammia luodaan. Se voidaan nähdä esimerkiksi Creation ikkunassa. Jos käyttäjä yhtäkkiä vaihtaa paikan, niin mittarin arvo nollautuu. Diagrammi ikkuna ei myöskään päästä eri mittareita samaan tauluun eikä käyttäjä esimerkiksi tule valitsemaan, minkä metodin kautta luodaan Diagrammi, vaan se on automatisoitu ja hyvin käsitelty. Niin Datareaderissa on monta muuttujaa, jotka sisältävät dataa (joka päivittyy aina kun ohjelmaa ajetaan uudestaan) paikkojen tyypeistä, niiden koodit, aikavälit ja niiden koodit, jne. Nämä muuttujat mahdollistavat kunnon suodattamisen luomisen vaiheessa. Moneen ohjelman kohtaan on kuitenkin laitettu try catch blockit, jotka kuvaavat ongelmia tarpeeksi hyvin ja samalla mahdollistavat sen, että ohjelma ei kaadu suorituksen aikana, monet try catchit ovat luotu requirments määräämisen jälkeen, mikä näkyy hyvin GUI:n toteutuksessa. Ohjelma on testattu myös monilla eri arvoilla, eikä tähän asti ole heittänyt yllättäviä virheitä tai luonut älyttömän outoja diagrammeja. Tämä ei kuitenkaan tarkoita, että se toimii täydellisesti, mutta kertoo kuitenkin paljon siitä, että se on ensimmäiseksi projektiksi riittävän hyvänlaatuinen.

Ohjelman puutteet ja viat:

- Yksinkertaisuudessaan – liian paljon toistuvaa koodia. Varsinkin huomaa GUI:n ja sen apu-luokista, kuinka paljon samanlaista koodia niissä on. Tätä virhettä olisi pitänyt miettiä, kun olin luomassa new diagram nappulan toimintaa ja siellä oleva createStagetta. Tajusin kuitenkin ongelman vasta silloin, kun aloin jo implementoida GUI:n toimintaa muiden ratkaisujen kanssa. Selvisin kuitenkin luomalla jokaista nappulaa varten oman luokan, joiden toiminta ei pääsääntöisesti poikkeaa toisistaan, mutta niiden tallentama data menee eri paikkoihin. Näin esimerkiksi AfterWordStage muuttaa GUI:n dataa GUI:n getterimetoidella. Tämä on ollut myös syynä, miksi oleellisimmat rivit kuten "diagramHimself, tabulaaaar ja diagrams" ovat niin korkealla tasolla GUI-objektissa. CompareStagen toiminta ei poikkea mitenkään GUI:n ja AfterWordStagen toiminnasta paitsi, että se tallentaa dataa paikallisesti ja luo uuden ikkunan diagrammista. CompareStagessa ei ole myöskään kaikkia niitä menu-esineitä, jotka ovat saatavilla PrimaryStageessa, sillä sen on tarkoitus olla apu-stage eikä main stage. Jälkikäteen ajatellen kolmesta mainituista stagesta olisi voitu tehdä vain yksi erillinen luokka, jota voisi operoida metodien kautta, jotka määräisivät Stageen oikean sijainnin ja toiminnan. Näin säästäisin varmaan noin 400-500 riviä? Ja helpottaisin koodin lukua sekä sen jälkikäteistä käsittelemistä.
- Jos käyttäjä päättääkin itse muokkailla tallennetussa tiedostossa olevaa dataa ja tekee jonkin virheen, ohjelma voi kaatua siihen. Alun perin KÄYTTÄJÄN EI OLE TARKOITUS VUOROVAIKUTTAA DATAN KANSSA SUORAAN TIEDOSTOSSA, vaan jos halua muokata jotakin, se kannattaa tehdä itse ohjelmassa Update ja Save työkaluilla. Jos kuitenkin käyttäjä on hyvin perehtynyt ohjelman toimintaan, voi siihen tehdä muutoksia. Esimerkiksi voi vaihtaa Diagrammin tyyppiä "Bar Chart":sta "Line Chart"iin, ja tiedoston avaamisen nappula "Open from File" kautta, diagrammi muuttuu LineChartiksi, mutta kuten aikaisemmin on mainittu näin ei ole ohjelman suorituksen kannalta suositeltavaa.
- Pieni puute myös liittyy siihen, jos käyttäjä käynnistää ohjelman ilman verkkoyhteyttä. Silloin ohjelma heittää virheen, eikä diagrammia muodostu. Eli kannattaa olla netti päällä silloinkin, kun avataan diagrammi tiedostosta, sillä data on dynaaminen.
- Yksi abstraaktipuute on itse käyttöliittymä. Se kykenee ehdottomasti vuorovaikutukseen, mutta ohjelma näyttää siltä, kuin se olisi tehty 40 vuotta sitten ja on designereiden tarpeessa! Tämä on kuitenkin makukysymys ja teknisen toiminnan kannalta ei ehkä ole niin oleellinen.
- Ohjelman puute on suorakolmion valintatyökalu. En ole ehtinyt implementoimaan sitä, koska en ymmärtänyt ajoissa mitä siinä vaaditaan. Ratkaisuna olisi kysyä vähän aikaisemmin assistentilta, mitä se oikein tarkoittaa.
- Itse sanoisin, että puutteena on myös Datareadrin metodien monipuolisuus. Oikeastaan SHP ja Kunta hakua voisi yhdistää yhteen metodiin THL:n datasta päätellen. Huomasin sen kuitenkin vasta silloin, kun jo olin luonut ne metodit ja siksi jatkoin ohjelman kirjoittamista niitä käyttäen. Tämä ei ole suuri puute, mutta taas tulevan käsittelyn kannalta olisi helpompi, jos pitää korjata yhtä metodia kahden sijaan.

3 Parasta ja kolme heikointa kohta:

Hyvät:

- Diagrammit. Diagrammien toiminta on mielestäni hyvin implementoitu. Diagrammit sisältävät metodeja, jotka auttavat käyttäjää vuorovaikuttamaan itse diagrammin ja kaikkien datapisteiden kanssa samaan aikaan. Jokaista datapistettä voi tutkia erikseen, mikä on toteutettu clicker/mover metodein kautta. “mover” auttaa selvittämään datapisteen peruskuvauksen, mikä on hyvin tehokasta varsinkin, kun on aika paljon pisteitä näkyvillä."clicker” näyttää kaikki tiedot datapisteestä painettaessa. Lisäksi se mahdollistaa värien muutokset, kommenttien lisäämisen sekä niiden saamisen tietystä pisteestä. Diagrammit sisältävät filelukemiseen tarvittavan color_in metodin, joka täyttää pisteet väreillä, jotka oli tallennettu tiedostoon aikaisemmin.
- FileReader ja FileWriter sekä DataReader. Olen tehnyt tallentamista ja lukemista varten erillisen FileWrittenTo luokan, joka toimii rinnakkain “FileReader”n ja FileWriter”n kanssa. Ne eivät ole kuitenkaan metodeja, vaan suora implementaatio Save ja Open from file ActionEventeille. Tässä haluaisin huomauttaa, että olisi parempi tehdä Readerista ja Writerista kuitenkin erillinen/erilliset objektit. Tämä ensisijaisesti mahdollistaisi niiden käytön monissa muissa ohjelman paikoissa ja helpomman käsittelemisen jälkikäteen, mutta muuten ne toimivat aina oikein, ellei dataa vahingoiteta ulkopuolelta. Tämä on tullut mahdolliseksi johtuen implicit encdoereiden ja decodereiden implementaatiosta, sekä Javan BufferedWrittrerin/Readerin kautta, joiden toiminta opetettiin kurssin alkuvaiheessa. Tarkemmin datan tallentamisprosessi on kuvattu kohdassa “Tiedosto ja verkossa oleva tieto”. DataReader on täytetty parsatulla THL:n Json-datalla, sekä metodeilla, joiden toiminnallisuus kaataa paikkakuntien ja niiden koodien määräämisen, datan luomisen tietyillä aikaväleillä, Kuntien tyyppien selvittäminen. DataReader muun muassa käyttää algoritmeja luodessaan DataLisoja. Algoritmi on kuvattu osiossa Algoritmit ihan ensimmäisenä.
- Monimutkaisuudestaan huolimatta GUI: monipuolisuus, joka kaataa aika monta toimintaa. GUI mahdollista jälkikätisen ohjelman käsittelyn, monien eri diagrammien muodostamisen samaan aikaan, analysoinnin eri mittareiden nojalla. GUI myös yhdistää Diagrammit, AnalyzeStage Readerit ja writerit, sekä kaikki muut funktiot, jotka ohjelma sisältää, minkä takia GUI: funktionaalisuus on mielestäni implementoitu erittäin mainiosti.

Huonot:

- GUI: monipuolisuus. GUI monipuolisuus eduistaan huolimatta vaatii todella ison koodimäärän kirjoittamista. Update toolin toimintaa varten on näin esimerkiksi luotu erillinen AfterWordStage luokka. Compare nappula on taattu CompareStagen toiminnalla. Tämä on toisaalta aika huono valinta, koska kuten jo aikaisemmin olen maininnut se nostaa ohjelman lukemisen vaikeustasoa ja lisää tarpeettoman monta riviä. Ratkaisua tähän pohdin kohdan “Ohjelman puutteet ja viat” alussa.
- LineChartin pikku buggi. Line chart kuten, myös muut diagrammit, toimii aivan hyvin kaikkien arvojen kanssa. Olen kuitenkin huomannut yhden pienen heikon kohdan, että kun diagrammiin on jo lisätty jokin määrä paikkoja tutkittaviksi, niin mikäli seuraavan

paikan alkuintervalli on pienempi kuin täällä hetkellä taulussa oleva pienin alkuintervalli Uuden chartin arvot voivat sijoittua oudosti diagrammiin. Näin esimerkiksi viikkoa kolme vastaava datapiste voi mennä 50 ja 55 väliin. Näin käy useimmiten yhdelle tai kahdelle pisteelle vain, eikä se kovin paljoa häiritse ohjelman suoritusta, mutta se ei ole kuitenkaan mukavaa. Oletan, että ongelmana on tässä se, että ViivaKaavio saa datapisteet, jonka aikaa kuvaava “date” pari sisältää luvut String muodossa, eikä ohjelma osaa kunnolla suorittaa niitä diagrammille. Toisaalta tämä teoria rikkoutuu joskus, koska jos luodaan muut diagrammit, joiden alkuintervalli onkin pienempi kuin pienin taulussa, mutta tämä intervalli kokonaisuudessaan kaataa kaikki muut intervallit, niin näkyville tulee ihan oikea Line Chart, kuten voi huomata. Oletan, että ratkaisua kannattaa etsiä XYChart_creator – metodissa ViivaKaavio –luokassa ja korjaus olisi suodatus päivämäärille ennen XYSeriesin luomista. Muiden diagrammien kohdalla ei ole samaa ongelmaa.

- Try-catch haarat GUI:n AddButtonin ActionEventtiä kuvaavassa toteutuksessa ovat erittäin vaikeanlukuisia ja vaativat, että niitä oikeasti otetaan kiinni erikseen, jos haluaa selvittää ongelman. Sanoisin vielä, että olisi syytä laittaa enemmän tekstiä settingsin help-tooliin, sekä tehdä enemmän Alertsia kun käyttäjä tekee jotakin ohjelman toimintaa mahdollisesti vahingoittavaa.

Poikkeamat suunnitelmasta, toteutunut työjärjestys ja aikataulu:

Lyhyesti tulkittuna päivä meni kaikkien frameworkien ja kirjastojen laatimiseen, puolitoistaviikkoa meni DataReadaerin luomiseen. Neljä päivää meni StatisticBoardPoints ja sitä periviin luokkiin alustavaan tekemiseen, ja 2 päivää niiden lopulliseen täydentämiseen. Viikko meni GUI:n kanssa ja mielestäni se oli vaikein kohta. Suunnitelmani ajankäyttö olisi ehkä järkevä, jos olisin heti ryhtynyt tekemään projektia, eikä olisi viivästyttänyt sen aloittamista 3 viikolla. Olisin saanut nyt projektin paljon parempaan kuntoon. Readerin kanssa meni niin paljon aikaa ei sen takia, että en ymmärtäisi jotakin, vaan siitä johtuen, että olin lomalla ja käytössä oli Macbook air, jonka kanssa on ihan mahdotonta tehdä yhtään mitään ohjelmointiin liittyvää. Siksi periaatteessa Data Readeria tein noin neljä päivää, ja loput ajasta kärsin luodakseni ainakin jotakin mackbookilla. Suunnitelmassa ei lue, mutta tarkoitus oli myös hakea päivittäistä dataa diagrammeja luodessa. Sen takia esimerkiksi metodi SHP_week_creator kuvaa nimenomaan sitä, että se on tarkoitettu viikoittaiselle SHP:n datalle ja DataPointin getData palauttaa toisena parametrinaan (Option[String],Option[String]), joista ensimmäinen kuvaa päivää ja toinen viikkoa, mutta käytetyksi tuli kuitenkin ainoastaan toinen. Olen tässä vaiheessa kuitenkin varmaa, että lähtisin tekemään projektia ihan samanlaisessa järjestyksessä, sillä näen sen järkevänä ja toimivan jopa lyhyellä aikavälillä erittäin hyvin. Kuitenkin varaisin vähän enemmän aikaa. Tiedän tosi hyvin, että jos minulla olisi vain yksi ylimääräinen viikko, niin projekti olisi ihan täydellinen ja luultavasti paremman näköinen. Ehtisin tekemään suorakolmiovalintatyökalun ja vähän muotoilemaan graafista liittymää, sekä lisätä aikaisemmin mainitun päivittäisen datahaun. Eli lopuksi voisi sanoa, että olen oppinut uudestaan, että hommia pitää tehdä mahdollisimman pian, varsinkin kun luvassa on niin paljon työtä. Jos asia stressaa, niin sen aloittamista ei kuitenkaan kannata välttää tai lykätä. Pitää vain ryhtyä tekemään ja oppimaan asioita, jolloin stressi häipyy itsestään.

Kokonaisarvio lopputuloksesta:

Tämä on ensimmäinen niin suuri projekti minulle, joka on tosiaan avannut silmät hyvin monien asioiden suhteen. Niin isot projektit pitää oikeasti miettiä hyvin etukäteen, suunnitella toiminnallisuutta ja mieluummin kirjoittaa se johonkin ylös, eikä vain lähteä tekemään ilman varsinaista suunnitelmaa. Olen varmaa, että ohjelmassani on liian paljon koodia, mikä tekee siitä vähän vaikeamman lukuisen sekä ei niin helpon maintainanceen suhteen. Jos kirjoittaisin samaa koodia uudelleen, miettisin GUI:n rakennetta uudelleen. Mielestäni Creation Stagetta olisi voitu tehdä heti erillisenä luokkana ja muokkailla niin, että se myös toimisi update-stagena, se sijaan, että updettia varten luon vielä yhden erillisen AfterWordUpdate luokkaa. Näin säästäisin hermoja, tilaa ja tekisin koodista vähän maintanablempi. Olen kuitenkin tosi ylpeä DataReaderista ja siitä, kuinka olen perehtynyt Circeeseen. Mielestäni on tosi näppärää, että yksi metodi kunta_Creator voi luoda dataOlioita siitäkin huolimatta, että se tietyissä tapauksissa saa erilaista dataa. Myös algoritmien toiminta kunta_Creatorissa ja SHP_week_creatorissa ovat todella hyviä. Olen myöskin tosi ylpeä, että löysin chartien metodeista, miten voi muokata värejä legend-tasolla (kuvassa 4 alempi laattikko, jossa paljon tekstiä). Olen oppinut eventHandlereiden ja ActionEventien käyttöä ja toteuttanut ne digrammien clicker ja mover metodeissa niin, että ne toimivat jokaisessa diagrammissa moitteettomasti. Circe-perehdykseni mahdollisti myös hyvän encodereiden ja decodereiden luomisen guihin. Ohjelmani osaa lukea dataa ja tallentaa sitä kaikkine muutoksineen, mikä on minusta suuri saavutus. Kuitenkaan en ryhtyisi kirjoittamaan filreadereita ja writereita taas suoraan GUI:n action eventeihi, vaan loisin mieluummin erillisen objektin, jonka GUI käynnistäisi.

Viitteet:

https://stackoverflow.com/ - en tähän laittaa jokaista linkkiä erikseen, koska tästä tullisi sitten varman 3 sivua lisää

https://thl.fi/fi/tilastot-ja-data/aineistot-ja-palvelut/avoin-data/varmistetut-koronatapaukset-suomessa-covid-19- -THL

https://sampo.thl.fi/pivot/prod/fi/epirapo/covid19case/fact_epirapo_covid19case.json?row=hcdmunicipality2020-445222&column=dateweek20200101-508933&column=measure-444833 - Json data (yksi esimerkeistä)

https://sampo.thl.fi/pivot/prod/fi/epirapo/covid19case/fact_epirapo_covid19case?row=hcdmunicipality2020-445222&column=dateweek20200101-508933&column=measure-444833 - Sama data kuin yllä, mutta visualisoitu

https://circe.github.io/circe/ - Circeen dokumentaatio

https://github.com/scalafx/scalafx - Scalafx:n github hienoine esimerkkeineen

https://docs.oracle.com/javafx/2/ui_controls/table-view.htm - TableView javafx

https://docs.oracle.com/javafx/2/charts/jfxpub-charts.htm - Charts, toimii samalla periaatteella scalafx:ssa

https://www.postman.com/ - Json lukemiseen. Antaa paremman visualisoinnin datasta.

https://javadoc.io/doc/org.scalafx/scalafx_2.12/latest/index.html - scalafx: dokumentaatio

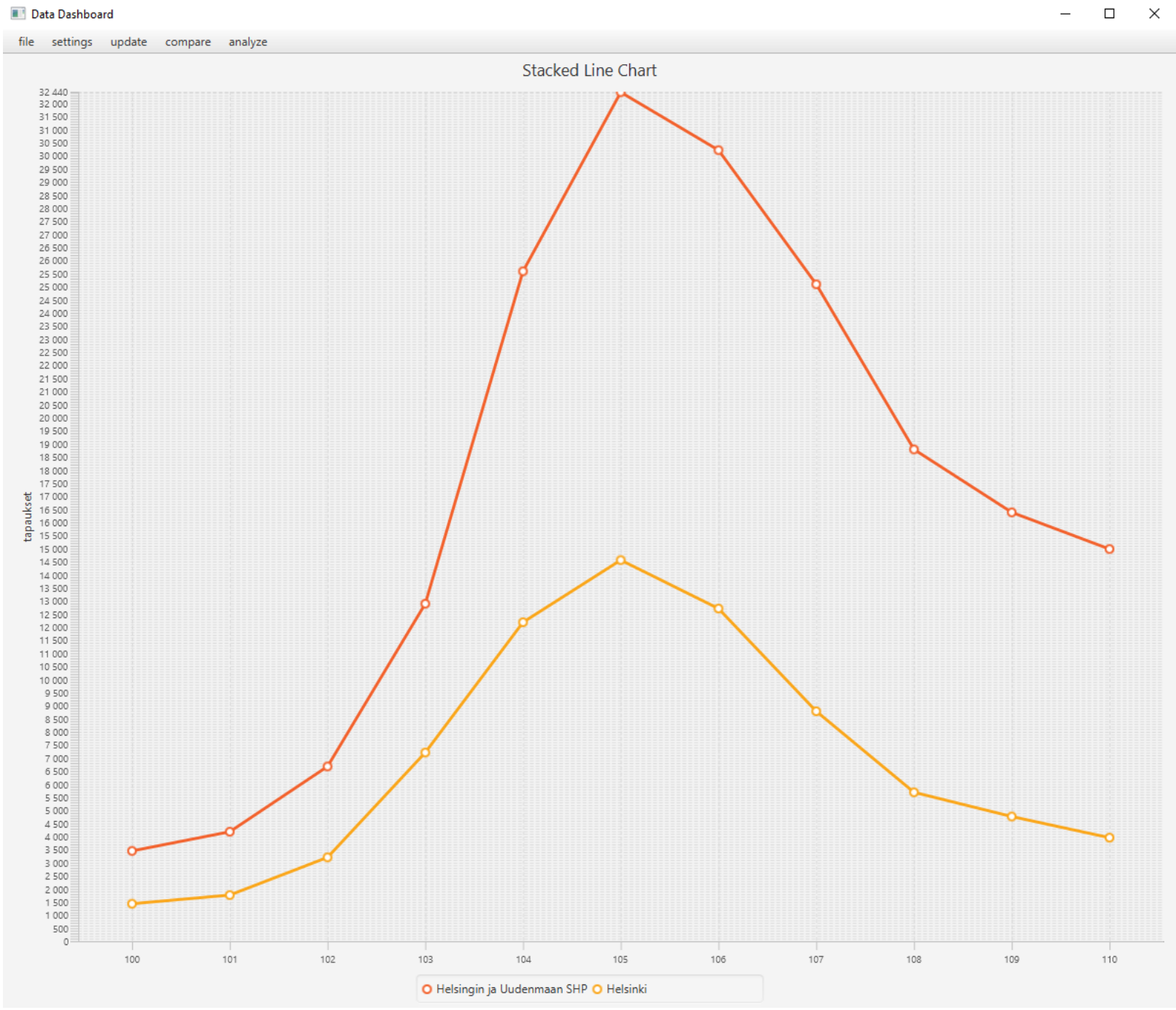
https://plus.cs.aalto.fi/studio_2/k2023/w15/ch04/ - Oli tosi hyödyllinen kappale

- Kuva 1

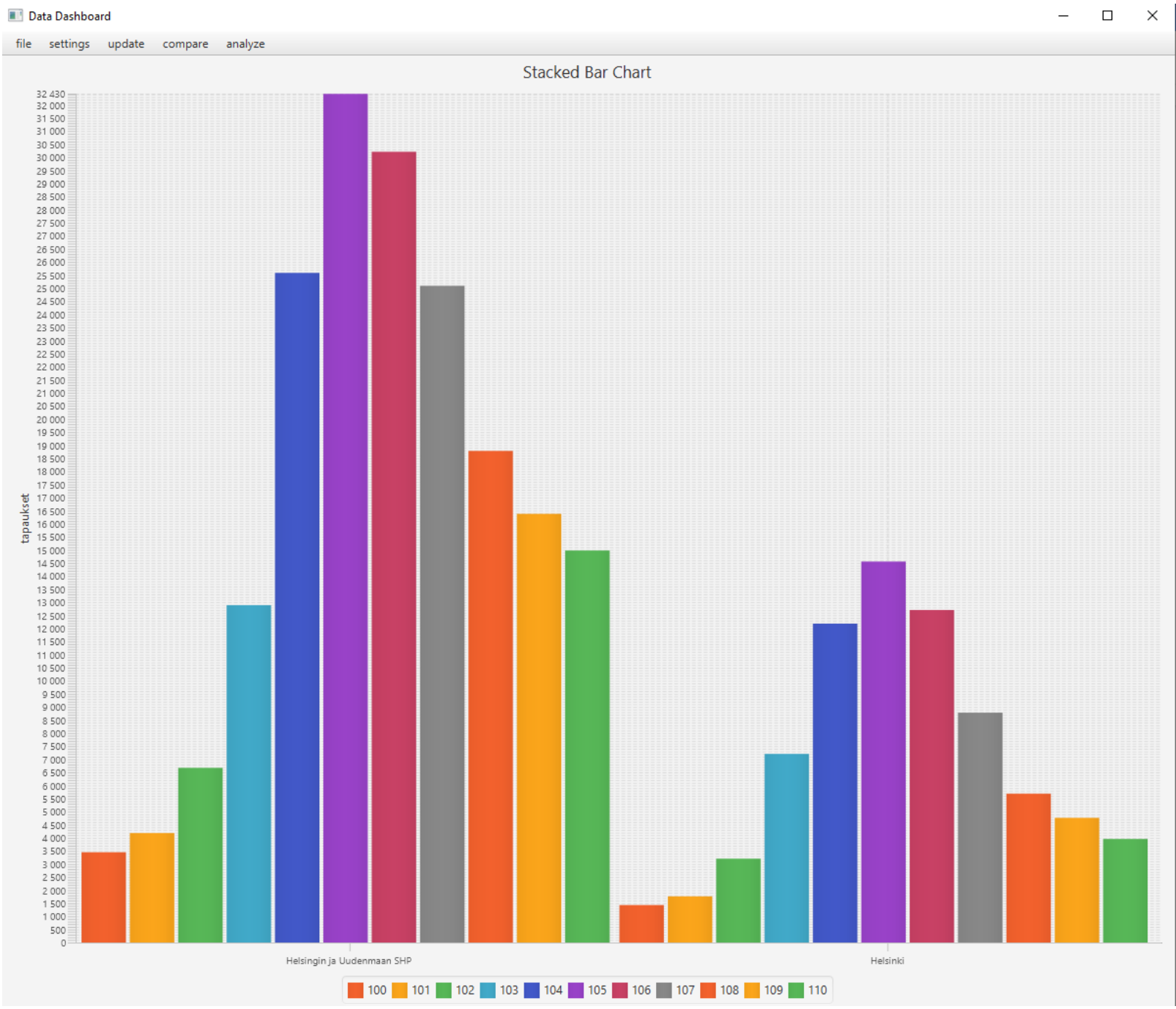
Mittarit

Mittari	Mittarin tunniste	Kuvaus	Lähde
COVID-19 -tapausten lukumäärä	n_covid_ttr	Tapaukseksi lasketaan tartuntatautirekisteriin ilmoitettu COVID-19 tauti. Yhdellä henkilöllä voi olla maksimissaan yksi tautitapaus 12 kuukauden sisällä. Ajallisesti tapaukset asetetaan tartuntatautirekisterin tilastointipäivälle.	Tartuntatautirekisteri.
COVID-19 testausmäärät	n_labtest	Laboratoriot ilmoittavat päivittäin, kuinka monta COVID-19 liittyvää testiä kyseisessä laboratoriossa tehdään kyseisenä päivänä. Ajallisesti testit asetetaan testin tekopäivälle.	Laboratorioilmoitus kyselylomakkeella
Koronaan ajallisesti liittyvät kuolemat (30 vrk), Tartuntatautirekisteri	n_deaths_ttr	1. Väestötietojärjestelmän kuolintieto, jonka päivämäärä on 30 päivän sisällä COVID-19 -tapauksen tilastointipäivämäärästä. 2. Tartuntatautirekisterin kuolintieto, kun lääkäri on ilmoittanut tartuntatautirekisteriin COVID-19 hoidon lopputulokseksi kuoleman. Ajallisesti kuolemat asetetaan tartuntatautirekisterin tilastointipäivälle.	Väestötietojärjestelmä ja tartuntatautirekisteri.
Asukaslukumäärä	sum_pop_week	Viikoittain päivittyvä asukasmäärä.	Väestötietojärjestelmä.
Koronasta johtuvat kuolemat, kuolintodistus (alustava tieto)	n_deaths_covid_due_kt	Kuolemat, joissa koronavirus on kirjattu kuolemansyyksi, eli kuoleman katsotaan aiheutuneen koronasta. Ajallisesti kuolemat asetetaan kuolinpäivälle.	Kuolintodistukset, alustava THL:n kokoama tieto. Tilastokeskus vastaa virallisesta tilastoinnista.
Kuolemat joissa korona myötävaikuttavana tekijänä, kuolintodistus (alustava tieto)	n_deaths_covid_ctb_kt	Kuolemat, joihin koronaviruksen katsotaan olleen kuolemaan myötävaikuttavana tekijänä, mutta kuolemansyy on jokin muu. Ajallisesti kuolemat asetetaan kuolinpäivälle.	Kuolintodistukset, alustava THL:n kokoama tieto. Tilastokeskus vastaa virallisesta tilastoinnista.

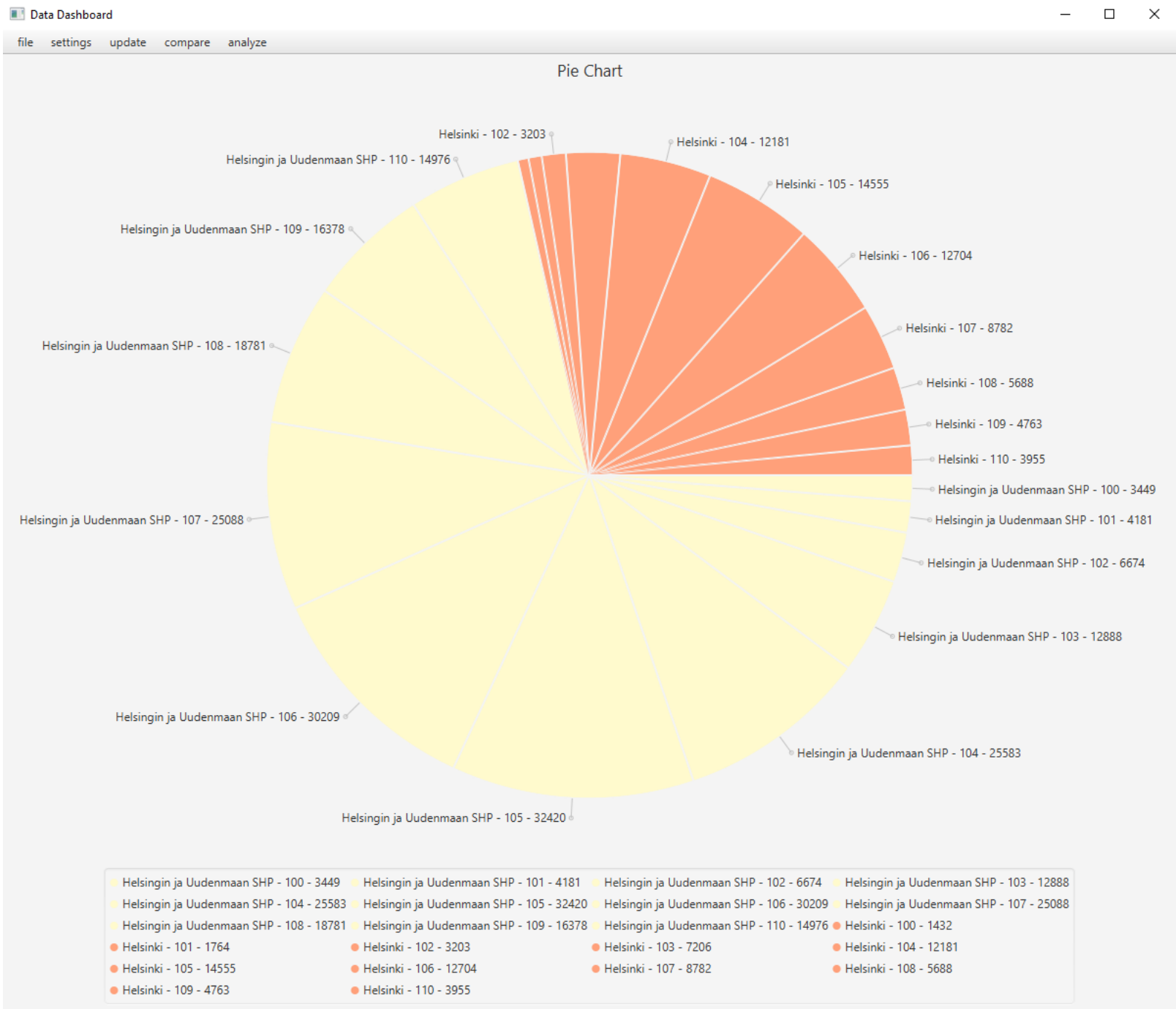
- Kuva 2.


Data Dashboard
file
settings
update
compare
analyze
Stacked Line Chart
tapaukset
32 440
32 000
31 500
31 000
30 500
30 000
29 500
29 000
28 500
28 000
27 500
27 000
26 500
26 000
25 500
25 000
24 500
24 000
23 500
23 000
22 500
22 000
21 500
21 000
20 500
20 000
19 500
19 000
18 500
18 000
17 500
17 000
16 500
16 000
15 500
15 000
14 500
14 000
13 500
13 000
12 500
12 000
11 500
11 000
10 500
10 000
9 500
9 000
8 500
8 000
7 500
7 000
6 500
6 000
5 500
5 000
4 500
4 000
3 500
3 000
2 500
2 000
1 500
1 000
500
0
100
101
102
103
104
105
106
107
108
109
110
Helsingin ja Uudenmaan SHP
Helsinki

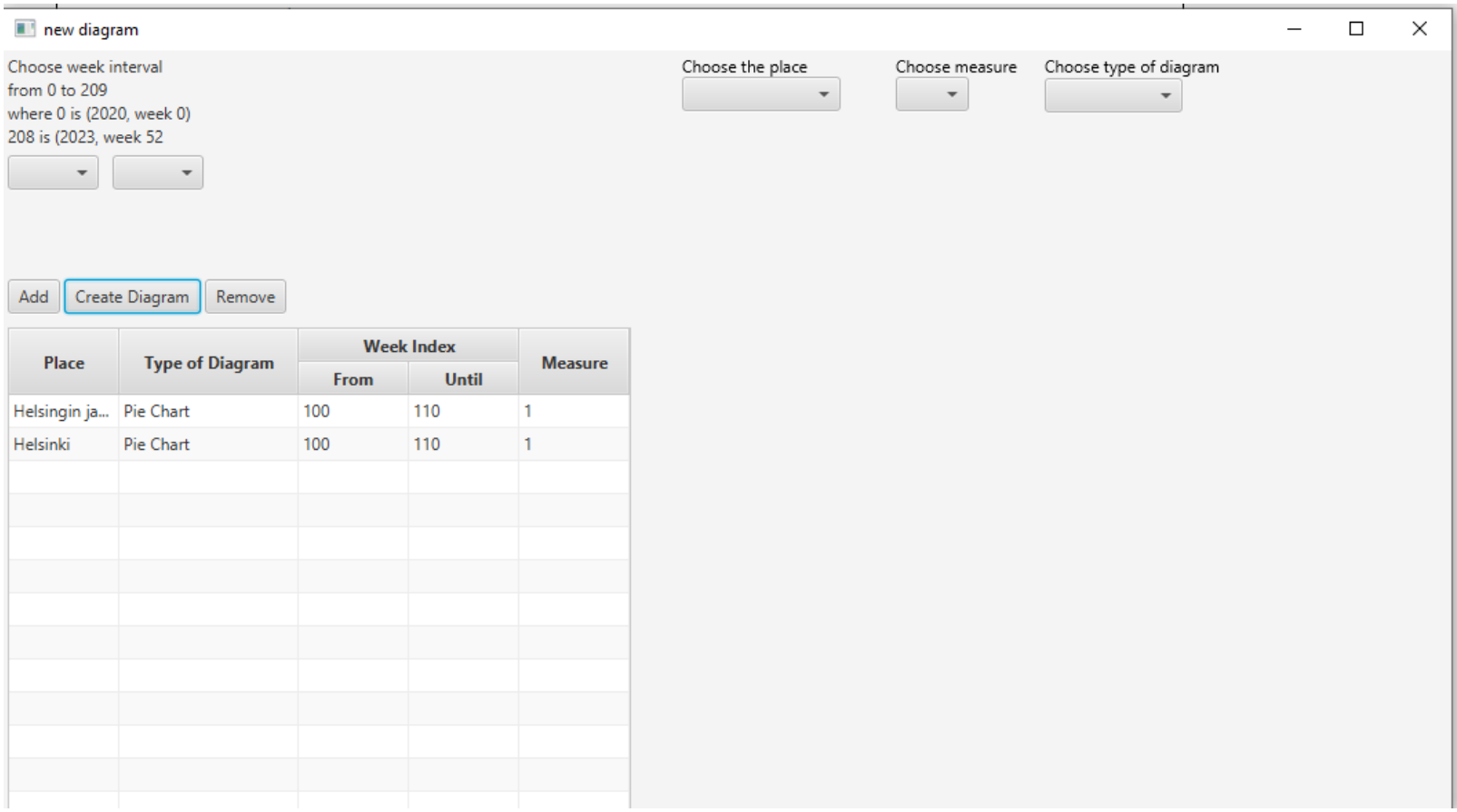
- Kuva 3.


Data Dashboard
file
settings
update
compare
analyze
Stacked Bar Chart
tapaukset
Helsingin ja Uudenmaan SHP
Helsinki
100
101
102
103
104
105
106
107
108
109
110

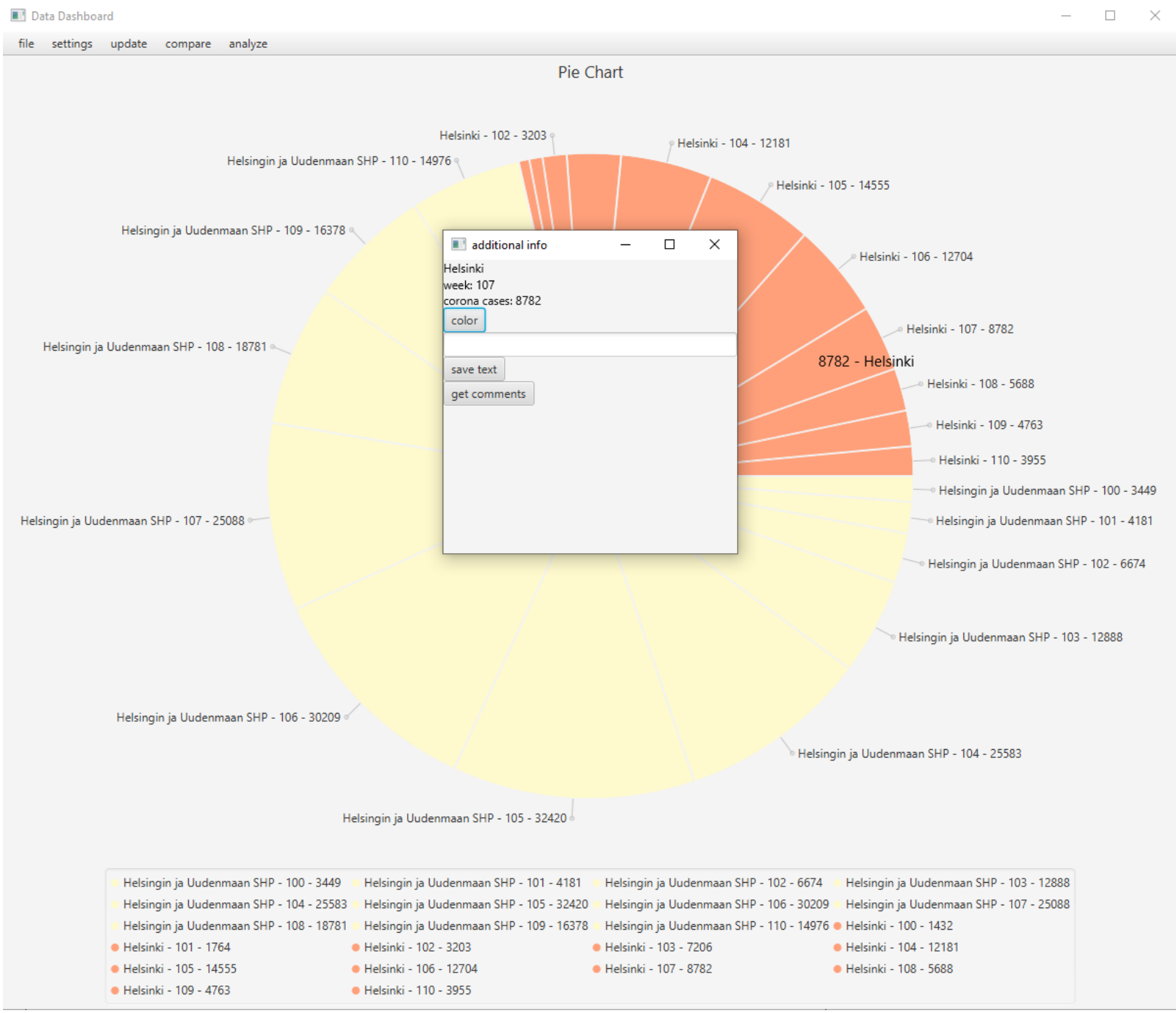
- Kuva 4.


Data Dashboard
file
settings
update
compare
analyze
Pie Chart
Helsinki - 102 - 3203
Helsinki - 104 - 12181
Helsingin ja Uudenmaan SHP - 110 - 14976
Helsinki - 105 - 14555
Helsingin ja Uudenmaan SHP - 109 - 16378
Helsinki - 106 - 12704
Helsinki - 107 - 8782
Helsingin ja Uudenmaan SHP - 108 - 18781
Helsinki - 108 - 5688
Helsinki - 109 - 4763
Helsinki - 110 - 3955
Helsingin ja Uudenmaan SHP - 100 - 3449
Helsingin ja Uudenmaan SHP - 107 - 25088
Helsingin ja Uudenmaan SHP - 101 - 4181
Helsingin ja Uudenmaan SHP - 102 - 6674
Helsingin ja Uudenmaan SHP - 103 - 12888
Helsingin ja Uudenmaan SHP - 106 - 30209
Helsingin ja Uudenmaan SHP - 104 - 25583
Helsingin ja Uudenmaan SHP - 105 - 32420
Helsingin ja Uudenmaan SHP - 100 - 3449
Helsingin ja Uudenmaan SHP - 101 - 4181
Helsingin ja Uudenmaan SHP - 102 - 6674
Helsingin ja Uudenmaan SHP - 103 - 12888
Helsingin ja Uudenmaan SHP - 104 - 25583
Helsingin ja Uudenmaan SHP - 105 - 32420
Helsingin ja Uudenmaan SHP - 106 - 30209
Helsingin ja Uudenmaan SHP - 107 - 25088
Helsingin ja Uudenmaan SHP - 108 - 18781
Helsingin ja Uudenmaan SHP - 109 - 16378
Helsingin ja Uudenmaan SHP - 110 - 14976
Helsinki - 100 - 1432
Helsinki - 101 - 1764
Helsinki - 102 - 3203
Helsinki - 103 - 7206
Helsinki - 104 - 12181
Helsinki - 105 - 14555
Helsinki - 106 - 12704
Helsinki - 107 - 8782
Helsinki - 108 - 5688
Helsinki - 109 - 4763
Helsinki - 110 - 3955

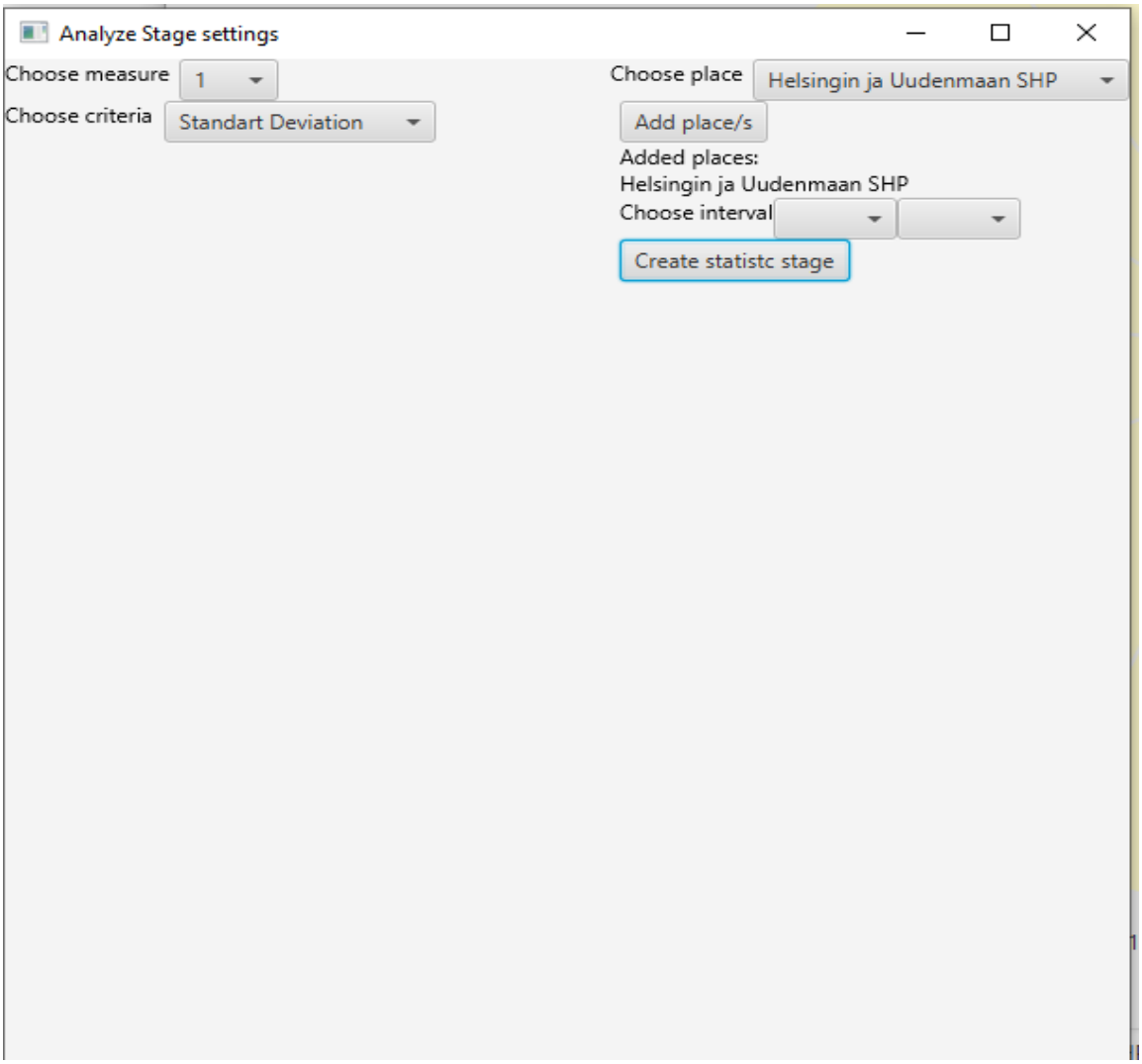
- Kuva 5.


new diagram
Choose week interval
from 0 to 209
where 0 is (2020, week 0)
208 is (2023, week 52
Choose the place
Choose measure
Choose type of diagram
Add
Create Diagram
Remove
Place
Type of Diagram
Week Index
From
Until
Measure
Helsingin ja...
Pie Chart
100
110
1
Helsinki
Pie Chart
100
110
1

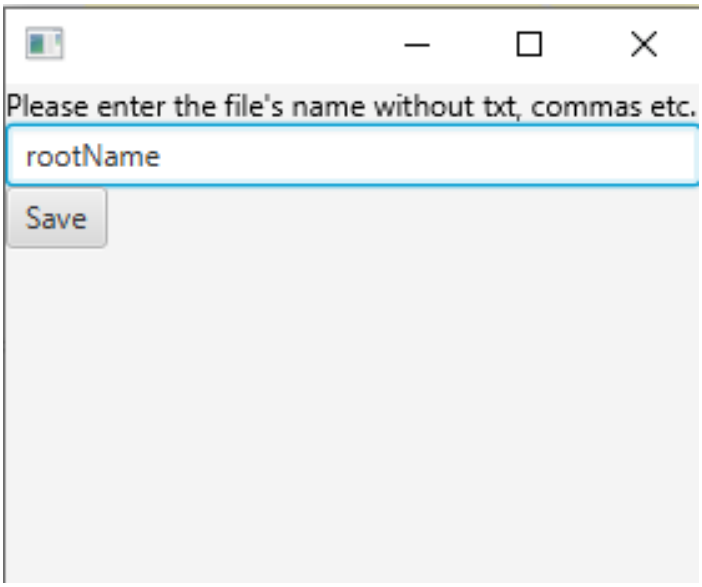
- Kuva 6.


Data Dashboard
file
settings
update
compare
analyze
Pie Chart
Helsinki - 102 - 3203
Helsinki - 104 - 12181
Helsingin ja Uudenmaan SHP - 110 - 14976
Helsinki - 105 - 14555
Helsingin ja Uudenmaan SHP - 109 - 16378
Helsinki - 106 - 12704
Helsinki - 107 - 8782
Helsingin ja Uudenmaan SHP - 108 - 18781
8782 - Helsinki
Helsinki - 108 - 5688
Helsinki - 109 - 4763
Helsinki - 110 - 3955
Helsingin ja Uudenmaan SHP - 100 - 3449
Helsingin ja Uudenmaan SHP - 107 - 25088
Helsingin ja Uudenmaan SHP - 101 - 4181
Helsingin ja Uudenmaan SHP - 102 - 6674
Helsingin ja Uudenmaan SHP - 103 - 12888
Helsingin ja Uudenmaan SHP - 106 - 30209
Helsingin ja Uudenmaan SHP - 104 - 25583
Helsingin ja Uudenmaan SHP - 105 - 32420
additional info
Helsinki
week: 107
corona cases: 8782
color
save text
get comments
Helsingin ja Uudenmaan SHP - 100 - 3449
Helsingin ja Uudenmaan SHP - 101 - 4181
Helsingin ja Uudenmaan SHP - 102 - 6674
Helsingin ja Uudenmaan SHP - 103 - 12888
Helsingin ja Uudenmaan SHP - 104 - 25583
Helsingin ja Uudenmaan SHP - 105 - 32420
Helsingin ja Uudenmaan SHP - 106 - 30209
Helsingin ja Uudenmaan SHP - 107 - 25088
Helsingin ja Uudenmaan SHP - 108 - 18781
Helsingin ja Uudenmaan SHP - 109 - 16378
Helsingin ja Uudenmaan SHP - 110 - 14976
Helsinki - 100 - 1432
Helsinki - 101 - 1764
Helsinki - 102 - 3203
Helsinki - 103 - 7206
Helsinki - 104 - 12181
Helsinki - 105 - 14555
Helsinki - 106 - 12704
Helsinki - 107 - 8782
Helsinki - 108 - 5688
Helsinki - 109 - 4763
Helsinki - 110 - 3955

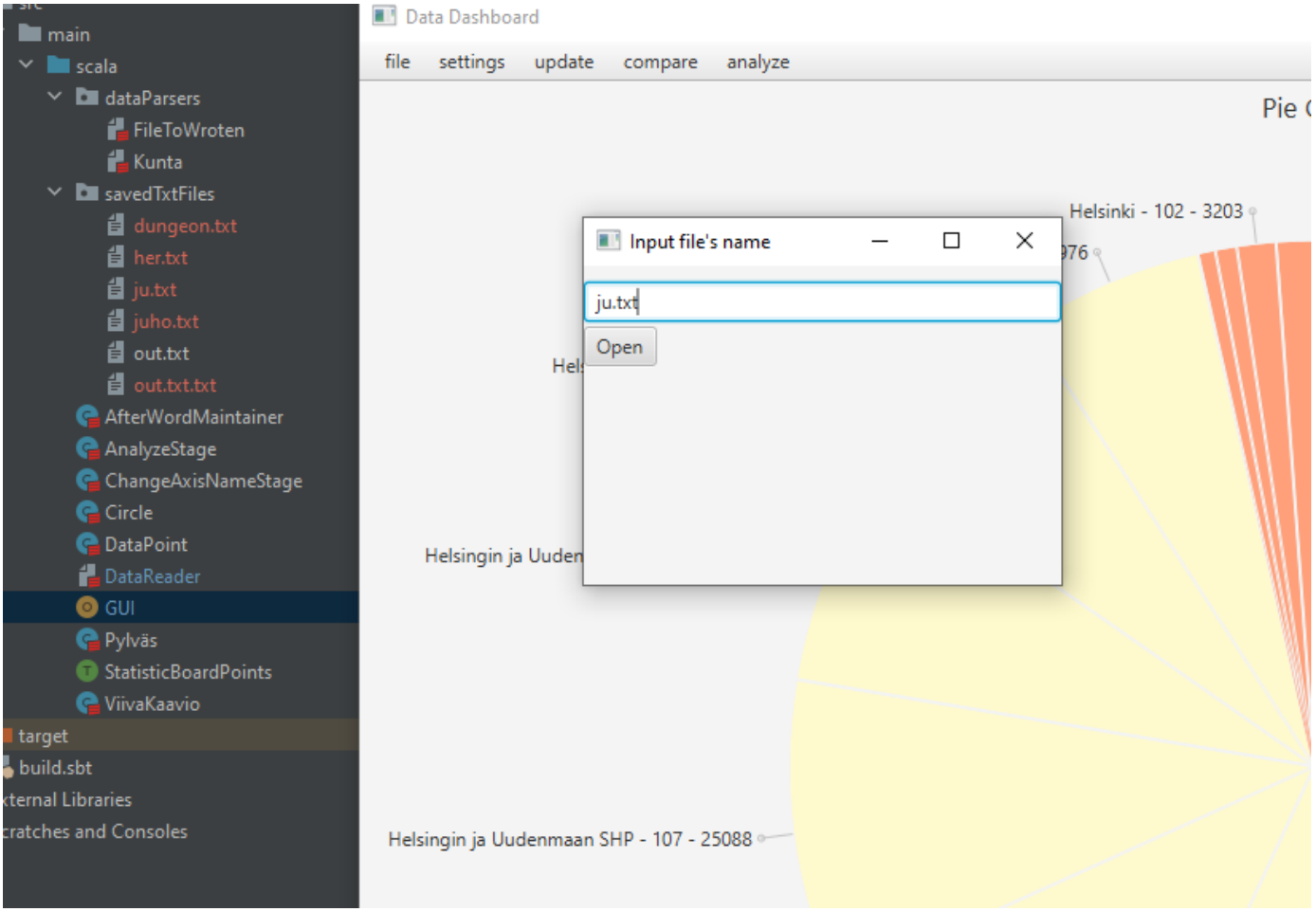
- Kuva 7


Analyze Stage settings
Choose measure
1
Choose criteria
Standart Deviation
Choose place
Helsingin ja Uudenmaan SHP
Add place/s
Added places:
Helsingin ja Uudenmaan SHP
Choose interval
Create statistc stage

- Kuva 8.


Please enter the file's name without txt, commas etc.
rootName
Save

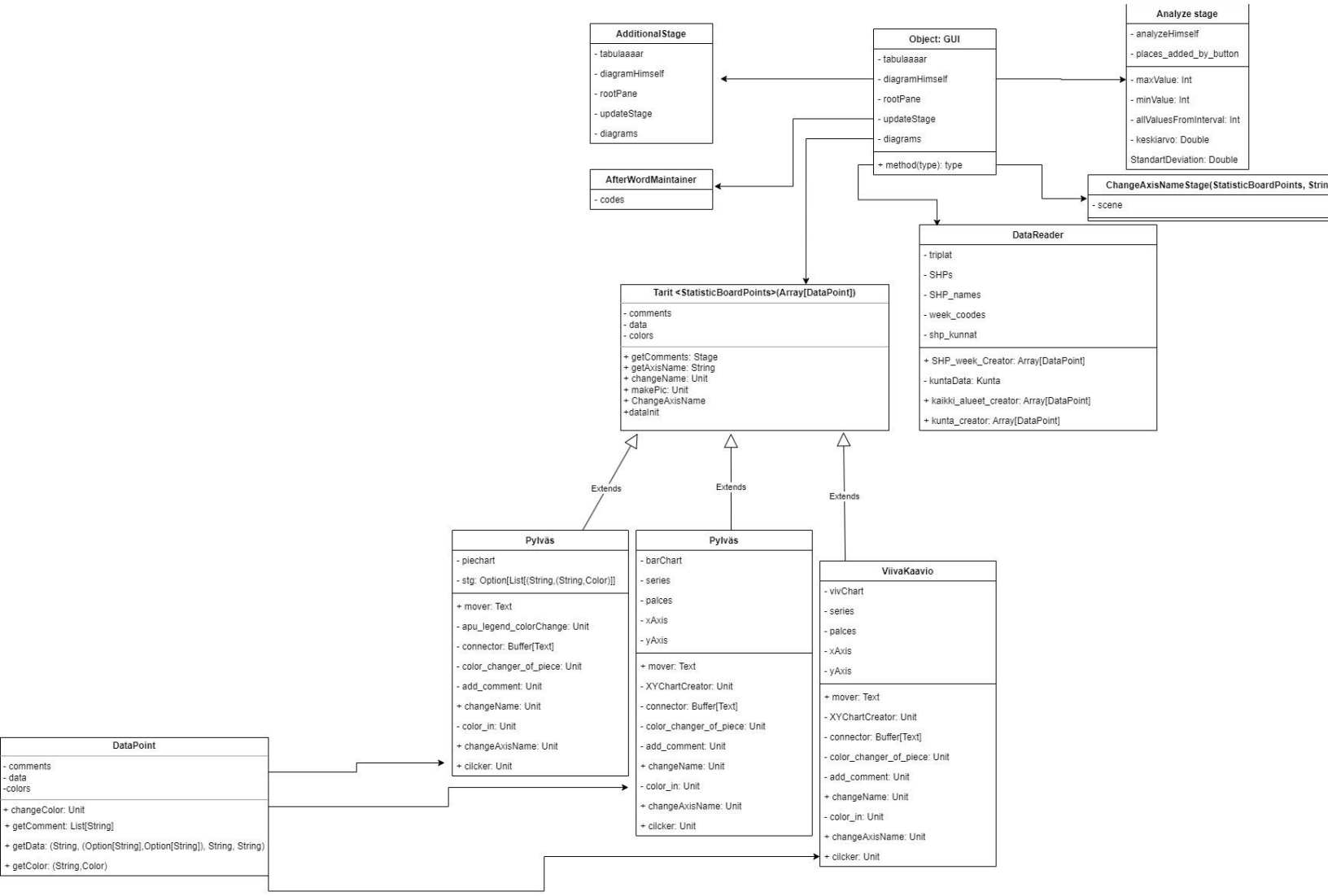
- Kuva 9.


main
scala
dataParsers
FileToWroten
Kunta
savedTxtFiles
dungeon.txt
her.txt
ju.txt
juho.txt
out.txt
out.txt.txt
AfterWordMaintainer
AnalyzeStage
ChangeAxisNameStage
Circle
DataPoint
DataReader
GUI
Pylväs
StatisticBoardPoints
ViivaKaavio
target
build.sbt
Data Dashboard
file
settings
update
compare
analyze
Input file's name
ju.txt
Open
Helsinki - 102 - 3203
Helsingin ja Uudenmaan SHP - 107 - 25088

- Kuva 10

	Vuosi 2020 Viikko 01		Vuosi 2020 Viikko 02		Vuosi 2020 Viikko 03		Vuosi 2020 Viikko 04		Vuosi 2020 Viikko 05		Vuosi 2020 Viikko 06	
	Testausmäärä	Tapausten lukumäärä	Testausmäärä	Tapausten lukumäärä	Testausmäärä	Tapausten lukumäärä	Testausmäärä	Tapausten lukumäärä	Testausmäärä	Tapausten lukumäärä	Testausmäärä	Tapausten lukumäärä
Ahvenanmaa										0		
Varsinais-Suomen SHP										0		
Satakunnan SHP										0		
Kanta-Hämeen SHP										0		
Pirkanmaan SHP										0		
Päijät-Hämeen SHP										0		
Kymenlaakson SHP										0		
Etelä-Karjalan SHP										0		
Etelä-Savon SHP										0		
Itä-Savon SHP										0		
Pohjois-Karjalan SHP										0		
Pohjois-Savon										0		

- Kuva 11


AdditionalStage
- tabulaaaar
- diagramHimself
- rootPane
- updateStage
- diagrams
Object: GUI
- tabulaaaar
- diagramHimself
- rootPane
- updateStage
- diagrams
+ method(type): type
Analyze stage
- analyzeHimself
- places_added_by_button
- maxValue: Int
- minValue: Int
- allValuesFromInterval: Int
- keskiarvo: Double
StandartDeviation: Double
AfterWordMaintainer
- codes
ChangeAxisNameStage(StatisticBoardPoints, Strin
- scene
DataReader
- triplat
- SHPs
- SHP_names
- week_coodes
- shp_kunnat
+ SHP_week_Creator: Array[DataPoint]
- kuntaData: Kunta
+ kaikki_alueet_creator: Array[DataPoint]
+ kunta_creator: Array[DataPoint]
Tarit <StatisticBoardPoints>(Array[DataPoint])
- comments
- data
- colors
+ getComments: Stage
+ getAxisName: String
+ changeName: Unit
+ makePic: Unit
+ ChangeAxisName
+datalnit
Extends
Extends
Extends
Pylväs
- piechart
- stg: Option[List[(String,(String,Color)]]
+ mover: Text
- apu_legend_colorChange: Unit
- connector: Buffer[Text]
- color_changer_of_piece: Unit
- add_comment: Unit
+ changeName: Unit
- color_in: Unit
+ changeAxisName: Unit
+ cilcker: Unit
Pylväs
- barChart
- series
- palces
- xAxis
- yAxis
+ mover: Text
- XYChartCreator: Unit
- connector: Buffer[Text]
- color_changer_of_piece: Unit
- add_comment: Unit
+ changeName: Unit
- color_in: Unit
+ changeAxisName: Unit
+ cilcker: Unit
ViivaKaavio
- vivChart
- series
- palces
- xAxis
- yAxis
+ mover: Text
- XYChartCreator: Unit
- connector: Buffer[Text]
- color_changer_of_piece: Unit
- add_comment: Unit
+ changeName: Unit
- color_in: Unit
+ changeAxisName: Unit
+ cilcker: Unit
DataPoint
- comments
- data
-colors
+ changeColor: Unit
+ getComment: List[String]
+ getData: (String, (Option[String],Option[String]), String, String)
+ getColor: (String,Color)