Haaga-Helia
ammattikorkeakoulu Oy

Datan visualisointi 3D-ympäristössä Unity-pelimoottorilla

Olli Sorjonen, Sami Sorjonen

Opinnäytetyö
Tietojenkäsittelyn koulutusohjelma
2018

Haaga-Helia ammattikorkeakoulu Oy

Tiivistelmä

8.5.2018

Tekijä(t)
Olli Sorjonen, Sami Sorjonen

Koulutusohjelma
Tietojenkäsittelyn koulutusohjelma

Opinnäytetyön otsikko
Datan visualisointi 3D-ympäristössä Unity-pelimoottorilla

Sivu- ja liitesivumäärä
89 + 9

Opinnäytetyön otsikko englanniksi
Data visualization in 3D environment with Unity game engine

Tuotetun datan määrä kasvaa koko ajan. Jotta tätä dataa voitaisiin käyttää paremmin hyödyksi, olisi olennaista löytää tehokkaampia tapoja sen tutkimiseen ja esittämiseen. Datan visualisoinnista 3D-ympäristössä pelimoottoreita käyttämällä löytyy suhteellisen niukasti tietoa, vaikka pelimoottorit tarjoavat erittäin hyvät työkalut vuorovaikutteisen kolmiulotteisen sisällön toteuttamiseen.

Tässä opinnäytetyössä tavoitteena oli selvittää datan visualisoinnin toteuttamista 3D-ympäristössä Unity-pelimoottorilla. Opinnäytetyö tehtiin parityönä ja se on tyypiltään toiminnallinen. Opinnäytetyöllä ei ollut toimeksiantajaa. Visualisoinnissa keskityttiin tilastodatan visualisointiin. Opinnäytetyön ulkopuolelle rajattiin mm. datan muokkaamiseen kehitettävät työkalut ja GPU-laskenta.

Työn tietoperusta jakautuu kahteen osaan. Ensimmäisessä osassa käsitellään visualisointia, sen merkitystä, datan visualisoinnin historiaa ja nykytilaa, visualisointiprosessia, aiheeseen liittyviä havaintopsykologian teorioita ja datan visualisointitapoja. Toisessa osassa kerrotaan Unity-pelimoottorista ja sen historiasta, sekä käsitellään sen ominaisuuksia datan visualisointiin soveltuvuuden näkökulmasta.

Opinnäytetyön toiminnallisessa osassa käsitellään datan kolmiulotteisten visualisointien suunnittelua ja toteuttamista Unityssä. Selvitetään, kuinka dataa saadaan tuotua Unityyn, ja minkälaisia valmisteluja sekä työvaiheita tämä vaatii. Lisäksi kerrotaan, kuinka kolmiulotteisia visualisointeja voidaan luoda Unityssä ja mitä työvaiheita tähän sisältyy.

Lopputuloksena syntyi kaksi Windows-ympäristössä toimivaa, pelimoottorilla toteutettua 3D-visualisointiprototyyppiä, joilla voidaan esittää muutamia tavallisimpia datan visualisointeja kolmiulotteisessa, vuorovaikutteisessa ympäristössä.

Datan tuominen Unityyn vaatii oletetusti tapauskohtaisia ratkaisuja, eikä valmiita työkaluja datan käsittelyyn ole. Visualisoinneissa usein tarvittavat asiat eivät ole tuettuna ja ne tulee toteuttaa itse. Myös käyttöliittymän toteuttaminen vaatii paljon manuaalista työtä, kun käytetään Unityn omaa UI-kirjastoa. Haasteista ja rajoitteista huolimatta Unityn todettiin soveltuvan kolmiulotteisten vuorovaikutteisten visualisointien toteuttamiseen.

Opinnäytetyö toteutettiin 2018 keväällä.

Asiasanat
Unity, data, visualisointi, kolmiulotteisuus, tilastografiikka, ohjelmistokehitys

Sisällys

1 Johdanto 1
1.1 Tavoitteet 1
1.2 Menetelmät ja rakenne 2
1.3 Rajaukset 2
1.4 Keskeiset käsitteet 3
2 Visualisointi 5
2.1 Visualisoinnin ja tietokonegrafiikan ero 6
2.2 Datan visualisoinnin merkitys 6
2.3 Visualisoinnin edut 7
2.4 Datan visualisoinnin historia 8
2.5 Datan visualisointi nykyaikana 10
2.6 Datan visualisointi ja datatyypit 10
2.7 Visualisointiprosessi 12
2.8 Näköaisti ja näköhavainto 12
2.9 Hahmolait 14
2.10 Visuaaliset muuttujat 15
2.11 Esitystavan merkitys 17
2.12 Kuvioroina 18
2.13 Värien käyttö 18
2.14 Harhaanjohtavat esitykset 19
2.15 Tavallisimpia tilastokuvioita 21
2.16 Kolmiulotteiset kuviot 21
3 Unity-pelimoottori 23
3.1 Yrityksen kasvu 23
3.2 Versiohistoria 24
3.3 Saatavuus 25
3.4 Lähestyttävyys 25
3.5 Ohjelmointi 26
3.6 Editori 26
3.7 Grafiikkaominaisuudet 27
3.8 Muut ominaisuudet ja oheispalvelut 29
3.9 Laite- ja alustatuki 29
3.10 Komponenttiajattelu 30
3.11 Käyttöliittymien rakentaminen 31
3.12 Puutteet 32
3.13 Soveltuvuus datan visualisointiin 32
3.14 Visualisointitekniikat ja skaalautuvuus 33
3.15 Unityn tukemat tiedostoformaatit 34

4 Visualisointiprotyyppien suunnittelu 36
4.1 Suunnittelu ja vaatimukset 36
4.2 Käytetyt työkalut ja ohjelmistot 38
5 Pylväsdiagrammivisualisointi 39
5.1 Datan lataaminen ja parserointi 39
5.2 Datan visualisoinnin valmistelu 42
5.3 Pylväsdiagrammin toteutustapa 43
5.4 Datan piirtäminen ruudulle 43
5.5 Koordinaattiruudukko 47
5.6 Kamera 48
5.7 Tooltip- ja RayCastItems-luokat 51
5.8 Graafinen käyttöliittymä 52
5.9 Äänet 57
6 Parvikuviovisualisointi 58
6.1 Datan lataaminen ja parserointi 58
6.2 Koordinaattiruudukko 60
6.3 Koordinaattiakselit 62
6.4 Datan rajojen visualisointi 64
6.5 Datapisteet 65
6.6 Datan piirtäminen ruudulle 65
6.7 Kamera 69
6.8 Graafinen käyttöliittymä 70
6.9 Visualisoinnin apuluokat 71
7 Tuotos 73
8 Pohdinta 75
8.1 Tavoitteiden saavuttaminen 76
8.2 Oppiminen 76
8.3 Johtopäätökset 77
8.4 Jatkokehitysajatuksia 79
Lähteet 81
Liitteet 90
Liite 1. Pylväsdiagrammivisualisoinnin scene-rakenne 90
Liite 2. Parvikuviovisualisoinnin scene-rakenne 91
Liite 3. Pylväsdiagrammivisualisoinnin kuvia 92
Liite 4. Parvikuviovisualisoinnin kuvia 93

1 Johdanto

Tuotetun digitaalisen datan määrä kasvaa koko ajan. Jotta tätä dataa voitaisiin käyttää paremmin hyödyksi, olisi olennaista löytää tehokkaampia tapoja sen tutkimiseen ja esittämiseen. Datan kolmiulotteiset visualisoinnit kuvina ja animaatioina ovat yleisiä. Myös keinotodellisuusvisualisointeja on käytetty eri tutkimuksen aloilla ja niitä on tutkittu akateemisesti jo pitkän aikaa (Bryson 1993; Cruz-Neira, Sandin, DeFanti, Kenyon & Hart 1992, 64-72; Cruz-Neira ym. 1993; Feiner, MacIntyre, Haupt & Solomon 1993, 144-155). Pelimoottorit ovat yleiskäyttöisiä, ja niillä voi toteuttaa monipuolista vuorovaikutteista kaksi- ja kolmiulotteista sisältöä. Esimerkiksi Unityllä on toteutettu pelien lisäksi erilaisia koulutusohjelmia, animaatioita, arkkitehtuurisia visualisointeja, taideinstallaatioita sekä VR- ja AR-ohjelmia. Vastaavaa vuorovaikutteisuutta ei olisi välttämättä lainkaan mahdollista toteuttaa visualisointityökaluilla. Unity, Unreal Engine ja CryEngine ovat mahdollistaneet pelimoottorien käytön lähes kenelle tahansa, johtuen niiden joustavista lisensointimalleista, jotka sallivat pelimoottorin käytön aloittamisen ilmaiseksi. Ne ovat madaltaneet kynnystä päästä toteuttamaan erilaisia projekteja, pelimoottorin hoitaessa vaikeimmat asiat. Tästä huolimatta pelimoottoreilla toteutettuja datan visualisointeja ei kuitenkaan tuntunut löytyvän kovin montaa. Tämä alkuasetelma huomioiden halutaan selvittää mitä ongelmia, vaatimuksia ja haasteita liittyy visualisoinnin toteuttamiseen pelimoottorilla.

1.1 Tavoitteet

Tämän opinnäytetyön tavoitteena on selvittää työvaiheet, joilla dataa voidaan tuoda Unity-pelimoottoriin ja kuinka toteuttaa vuorovaikutteinen visualisointi tästä datasta. Tavoitteen saavuttamiseksi on määritelty seuraavia tutkimuskysymyksiä, joihin pyritään vastaamaan:

1. Mitkä ovat haasteet datan tuomisessa pelimoottoriin ja datan esittämisessä 3D-tilassa?
   - Mitkä ovat työvaiheet datan tuomisessa pelimoottoriin?
2. Mitä tulee huomioida, jotta dataa voidaan esittää 3D-tilassa?
   - Mitkä ovat vaatimukset datalle, jotta sitä voidaan esittää 3D-tilassa?
   - Mitkä ovat vaatimukset 3D-grafiikalle liittyen datan esittämiseen?
3. Mitkä ovat datan esittämiseen 3D-tilassa liittyvät edut ja haitat?
   - Miten 3D-tilassa voidaan esittää dataa niin, että ymmärrettävyys säilyy?

Aihevalinta mahdollistaa opinnäytetyöntekijöiden useamman kiinnostuksen kohteen soveltamisen: 3D-grafiikka, Unity-pelimoottori ja opiskellessa käsitellyt asiat kuten data ja datan visualisointi. Halutaan myös päästä käytännössä kokeilemaan ja syventämään opittuja ohjelmointitaitoja. Lisäksi tarjoutuu mahdollisuus perehtyä datan visualisoinnin taustalla olevien teorioiden perusteisiin, jotka ovat osittain vieraita. Lisäksi voidaan hyödyntää käytettävyyteen liittyviä oppeja visualisointien käyttöliittymien toteutuksessa.

1.2 Menetelmät ja rakenne

Opinnäytetyö on produktityyppinen, jonka tavoitteena on tuottaa kaksi visualisointiprototyyppiä. Opinnäytetyö toteutetaan parityönä, tekijöinä Olli Sorjonen ja Sami Sorjonen. Opinnäytetyöllä ei ole toimeksiantajaa. Opinnäytetyö koostuu teoriaosasta ja toiminnallisesta osasta.

Teoriaosa alkaa luvusta 2, jossa käsitellään visualisointia, sen merkitystä, datan visualisoinnin historiaa ja nykytilaa, visualisointiprosessia, aiheeseen liittyviä havaintopsykologian teorioita ja datan visualisointitapoja. Tarkoituksena on muodostaa kokonaiskäsitys hyvän ja huonon visualisoinnin piirteistä, eli mitkä asiat tulee huomioida visualisointia toteuttaessa. Luvussa 3 käsitellään Unity-pelimoottoria, sen historiaa ja ominaisuuksia. Tämän jälkeen tarkastellaan Unityn soveltuvuutta datan visualisointiin.

Toiminnallisessa osassa (luvut 4-7) keskitytään vuorovaikutteisten visualisointien suunnitteluun ja toteuttamiseen Unity-pelimoottorilla, käyttämällä C#-ohjelmointikieltä. Selvitetään, miten valittuja tekniikoita ja työkaluja käytetään visualisointiprototyyppien toteuttamiseen. Kokeillaan datan tuomista ja parserointia pelimoottorin käyttöön. Lisäksi dokumentoidaan tähän liittyvät vaatimukset, haasteet ja havainnot. Työkaluina käytetään Unity-pelimoottorin lisäksi Visual Studio Code -editoria, Exceliä ja tekstieditoreita. Lisäksi käytetään ryhmätyöskentelyyn sopivia työkaluja, kuten Git ja Google Docs. Tarvittavien 3D-kappaleiden toteutukseen käytetään Blender 3D-ohjelmaa.

Toiminnallisen osan lopputuloksena syntyy kaksi Unity-pelimoottorilla toteutettua Windows-ympäristössä ajettavaa vuorovaikutteista visualisointiprototyyppiä, joilla voidaan esittää muutamia tavallisimpia datan visualisointeja kolmiulotteisessa ympäristössä.

Tämän opinnäytetyön tiedoista voi olla hyötyä niille, jotka aikovat kehittää kolmiulotteisia visualisointeja Unityssä. Syntynyttä tietoa voidaan käyttää apuna, niin ettei yleisimpiä virheitä tehdä ja pystytään toteuttamaan ohjelman rakenne, joka on Unityn käytäntöjen mukainen. Lukijat voivat saada myös ajatuksia siitä, mitkä ovat mahdollisia graafisten esitysten toteutustapoja.

1.3 Rajaukset

Koska aihepiiri on suhteellisen laaja, datan visualisoinneissa keskitytään tilastografiikkaan, joka on yleisesti tunnettua. Tilastografiikkaakin voidaan esittää lukuisilla eri tavoilla, joten esitystavat rajataan muutamaan yleisimpään. Opinnäytetyössä keskitytään visuali-

sointien toteuttamiseen, vaikka voitaisiin myös tutkia datan lataamiseen ja muokkaamiseen tarvittavia työkaluja tai niiden rakentamista. Näiden toteuttaminen voisi osoittautua liian työlääksi. Toteutettaville visualisointiohjelmille ei tehdä käytettävyystestausta, testaaminen rajataan itse toteutettuun visualisointien toimivuuden toteamiseen. Myöskään reaaliaikaisen datan esittämistä ei käsitellä, eikä myöskään ns. big datan visualisointia, koska se vaatisi todennäköisesti GPU-laskentaa, jonka toteuttaminen voi osoittautua liian työlääksi ja vaativaksi opinäytetyöntekijöille.

1.4 Keskeiset käsitteet

Data	Joukko tietoa tai arvoja, joka ei välttämättä ole jäsentynyttä.
Visualisointi	Datan tai tiedon esittäminen graafisesti, voi tarkoittaa lopputulosta tai prosessia.
Unity	Suljetun lähdekoodin kaupallinen pelimoottori, jolla voi toteuttaa 2D- ja 3D-grafiikkaa käyttäviä tietokonepelejä tai -ohjelmia useille eri kohdealustoille.
Pelimoottori	Ohjelmistokehys, jota käyttämällä on mahdollista toteuttaa tietokonepelejä, simulaatioita tai ohjelmia eri kohdealustoille.
Informaatio	Yleiskäsite, jolla viitataan jäsennettyyn dataan.
Tilastografiikka	Visualisointi, joka välittää tilastotietoja.
Tietokonegrafiikka	Tietokoneen avulla tuotettu digitaalinen kuva, kuvitus tai animaatio.
3D-grafiikka	Tietokonegrafiikkaa, jossa kolmiulotteisten mallien pohjalta on tuotettu kuvia tai animaatiota.
Asset	Unityn käyttämässä merkityksessä viittaa Unity-projektissa käytettävään sisältöön kuten tiedostot, mallit ja tekstuurit.
C#	Microsoftin kehittämä yleiskäyttöinen olio-ohjelmointikieli.

Luokka	Olio-ohjelmoinnin peruskappale, johon voidaan koostaa haluttu toiminnallisuus. Luokasta voidaan luoda olioita.
Metodi	Komentoja sisältävä osio koodia, jota voidaan kutsua sen nimellä.
Virtuaalitodellisuus	(engl. virtual reality, VR.) Tietokonesimulaatio, jonka avulla voidaan tuottaa keinotekoinen ympäristö.
Lisätty todellisuus	(engl. augmented reality, AR.) Näkymä, johon on lisätty tietokonegrafiikalla tuotettuja elementtejä nähdyn ympäristön osaksi.
GPU	(engl. graphics processing unit.) Grafiikan nopeaan piirtämiseen näytölle erikoistunut piiri.
Shaderi	Pieni GPU:lla ajettava ohjelma, jossa lasketaan pikselien värejä erilaisilla algoritmeilla tai funktioilla, yleensä käyttämällä syötettyjä valojen ja materiaaliparametrien arvoja.
Verteksi	3D-mallissa oleva piste. Pisteitä yhdistämällä voidaan luoda monikulmioita, jotka muodostavat 3D-mallin pinnan.
UV-koordinaatit	3D-malliin lisätty koordinaattidata, jolla kaksiulotteinen pintakuviointi voidaan siirtää 3D-mallin pinnalle.

2 Visualisointi

Vaikka visualisointi käsitteenä olisi vieras, visualisointeihin voi törmätä lähes kaikkialla - liikennemerkki, pörssikursseja kuvaava käyrä, sääkartta, 3D-kuva tai -animaatio talosta ja huonekalun kasausohjeet kaikki ovat visualisointeja. Visualisointi on laaja käsite. Erään määritelmän mukaan visualisointi tarkoittaa tiedon välittämistä graafisia esityksiä käyttämällä (Ward, Grinstein & Keim 2010, 1). Näin kaikenkattavista määritelmistä johtuen, visualisoinnin voidaan ymmärtää tarkoittavan monia asioita.

Visualisointitutkija Robert Kosara on yrittänyt määritellä visualisointi-käsitettä kirjoituksessaan, jossa hän käsittelee erityisesti tiedon visualisointia. Kosaran (2007, 2) mukaan visualisoinnille ei ole yhtä selkeää yleisesti hyväksyttyä määritelmää. Kosara on kuitenkin määritellyt seuraavat kriteerit, joita voidaan pitää minimivaatimuksena kaikille visualisoinneille:

- Visualisointi perustuu (ei-visuaaliseen) dataan.
- Visualisointiprosessin lopputuloksena on kuva.
- Lopputulos on luettavissa ja tunnistettavissa.

Visualisoinnilla tarkoitetaan siis abstraktin tiedon esittämistä ja tämän tiedon välittämistä katsojalle. Koposen, Hildenin ja Vapaasalon mukaan tietoa välittävät kuvat jaetaan usein kahteen pääkategoriaan viestinnällisen funktion mukaan, joko visualisointeihin tai infografiikoihin. Lähteestä riippuen näille käsitteille annetut merkitykset vaihtelevat paljonkin. Visualisointi on eksploratiivista (engl. exploratory), uusia piirteitä aineistosta paljastavaa grafiikkaa. Sitä voidaan käyttää uuden tiedon löytämiseksi, sen ensisijaisena pyrkimyksenä ei ole siis tekijöiden ennalta määrittelemän viestin välittäminen. Infografiikka on selittävää (engl. explanatory), viestintää tukevaa grafiikkaa. Infografiikan tehtävänä on ennen kaikkea tiedon välittäminen ihmiseltä toiselle. Jako visualisointeihin ja infografiikkaan ei ole heidän mielestään myöskään mustavalkoinen, vaan näitä voidaan ajatella jatkumona tarkkarajaisten kategorioiden sijaan. (Koponen, Hilden & Vapaasalo 2016, 20-22.)

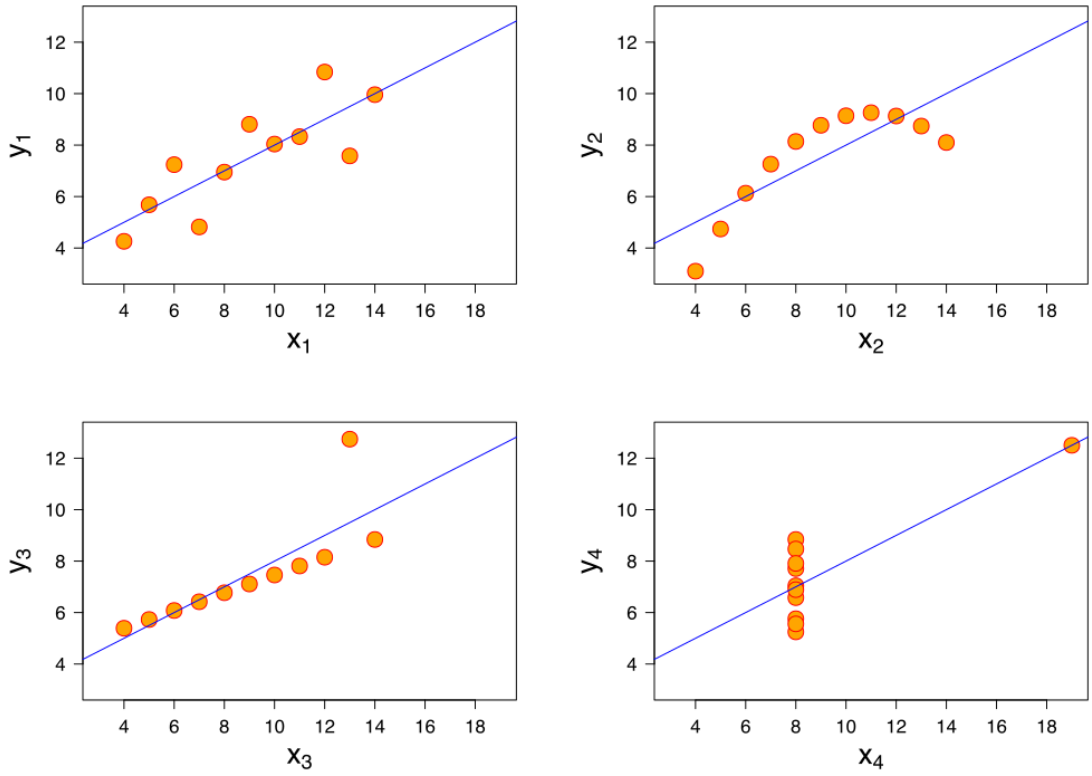
Sanoja “visualisointi” tai “visualisoiminen” voidaan käyttää yleisemmin kuvaamaan sekä prosessia, jossa data muutetaan visuaaliseen muotoon, että tämän prosessin lopputulosta. Tällöin visualisointi on miellettävä kattotermiksi, jonka alle kuuluvat kaikki varsinaiset visualisoinnit ja useimmat infografiikat. Kaikessa visuaalisessa viestinnässä ei myöskään ole aina kyse tiedon välittämisestä, vaan viestinnälliset tavoitteet voivat liittyä mm. mielikuviin, arvoihin tai muihin vähemmän konkreettisiin asioihin. (Koponen ym. 2016, 23.)

2.1 Visualisoinnin ja tietokonegrafiikan ero

Ward, Grinstein ja Keim kertovat, että visualisointi on aluksi mielletty tietokonegrafiikan osa-alueeksi pääasiassa siitä syystä, että visualisoinnissa käytetään tietokoneen tuottamaa grafiikkaa. Visualisointi sivuaa monia muita tieteenaloja, kuten ihmisen ja tietokoneen vuorovaikutusta, havaintopsykologiaa, tietokantoja, tilastoja ja tiedon louhintaa. Tietokonegrafiikka keskittyy pääasiassa interaktiivisten synteettisten kuvien ja animaatioiden luontiin kolmiulotteisista kappaleista, visuaalisen realismin ollessa usein yksi tietokonegrafiikan tavoitteista. Toissijainen tietokonegrafiikan käyttötarkoitus on taiteessa ja viihteessä, kuten videopelit, animaatioelokuvat, mainokset ja elokuvien erikoistehosteet. Visualisointi sen sijaan ei korosta visuaalista realismia, vaan tehokasta tiedon välittämistä. Monet visualisointityypit eivät käsittele fyysisiä kappaleita, ja ne jotka käsittelevät, kommunikoivat usein kappaleiden näkymättömiä ominaisuuksia, kuten esimerkiksi materiaalin jännitys tai nesteen virtaus. Näin ollen, vaikka tietokonegrafiikka ja visualisointi jakavat monia samoja käsitteitä, työkaluja ja tekniikoita, taustalla olevat mallit ja tavoitteet ovat pohjimmiltaan erilaisia. (Ward ym. 2010, 21-22.)

2.2 Datan visualisoinnin merkitys

Ehkä yleisin ja tunnetuin esimerkki datan visualisoinnin tärkeydestä on tilastotieteilijä Francis Anscomben kehittämä neljän datasetin ryhmä, jota kutsutaan nykyisin nimellä Anscomben kvartetti (kuvio 1). Tilastotieteessä käytettyjen matemaattisten funktioiden avulla vertailemalla ne vaikuttavat hyvin samankaltaisilta, mutta visualisoituna ne paljastuvat hyvin erilaisiksi. Anscomben mukaan monet tilastolliset laskutavat nojaavat olettamuksiin datan käyttäytymisestä. Nämä olettamukset voivat olla vääriä, ja siten laskelmista voi tulla harhaanjohtavia. Sen takia Anscomben mukaan on aina tärkeää pyrkiä tarkastamaan ovatko olettamukset kohtuullisen oikeita - tähän voidaan käyttää apuna visualisointia. (Anscombe 1973, 17.) Anscomben kvartetin keskeinen ajatus on siis esittää datan visualisoinnin hyödyllisyys datan analysoinnissa.



Kuvio 1. Tilastollisesti hyvin samankaltaiselta vaikuttavat Anscomben kvartetin arvosarjat visualisoituna (Wikipedia 2018a)

2.3 Visualisoinnin edut

Visualisointi on myös tehokas tapa välittää informaatiota, koska ihminen kerää valtaosan informaatiosta näköaistinsa avulla. Lisäksi ihmisen aivot käsittelevät kerättyä visuaalista dataa tehokkaasti. Yksi kuva voi sisältää runsaasti tietoa, ja se voidaan käsitellä huomattavasti nopeammin kuin sivullinen sanoja. Tämä johtuu siitä, että kuvan tulkinta tapahtuu rinnakkaisena toimintona havaintojärjestelmässämme, kun taas tekstin analysointia rajoittaa lukuprosessi, joka muodostuu sarjasta perättäisiä havaintoja. Kuvat voivat olla myös niiden tulkitsijan kielestä riippumattomia. (Ward ym. 2010, 1.)

Colin Waren mukaan tiedon visualisoinnin yksi suurimmista eduista on se, että valtava määrä tietoa voidaan tulkita nopeasti, jos se on esitetty hyvin. Esimerkkinä Ware käyttää meren pohjasta tehtyä kolmiulotteista visualisointia, jossa on visualisoituna noin miljoona mittauspistettä. Tämän visualisoinnin avulla ihminen pystyy havainnoimaan mittaukset välittömästi ja helposti. Waren mukaan visualisoinnilla on useita etuja: sen lisäksi, että visualisointi mahdollistaa suuren datamäärän hahmottamisen, se mahdollistaa myös emergenttien ominaisuuksien havainnoinnin - visualisoinnin avulla voidaan helposti havaita erilaisia kuvioita, joiden pohjalta voidaan taas tehdä uusia havaintoja. Usein visualisointi

mahdollistaa datassa esiintyvien ongelmien selkeän havaitsemisen - se ei välttämättä pelkästään paljasta tietoja datasta, vaan myös tavasta, jolla dataa on kerätty. Oikealla tavalla valittu visualisointi voi myös paljastaa datan virheet hyvin ilmeisesti. Tästä syystä visualisointi on arvokas työkalu datan laatua arvioidessa. Visualisointi voi myös auttaa sekä suuren että pienen mittaskaalan ominaisuuksien havainnointia datasta. Lisäksi visualisointien avulla voidaan löytää uutta tutkittavaa sekä muodostaa tutkimuskysymyksiä. (Ware 2013, 3-4.)

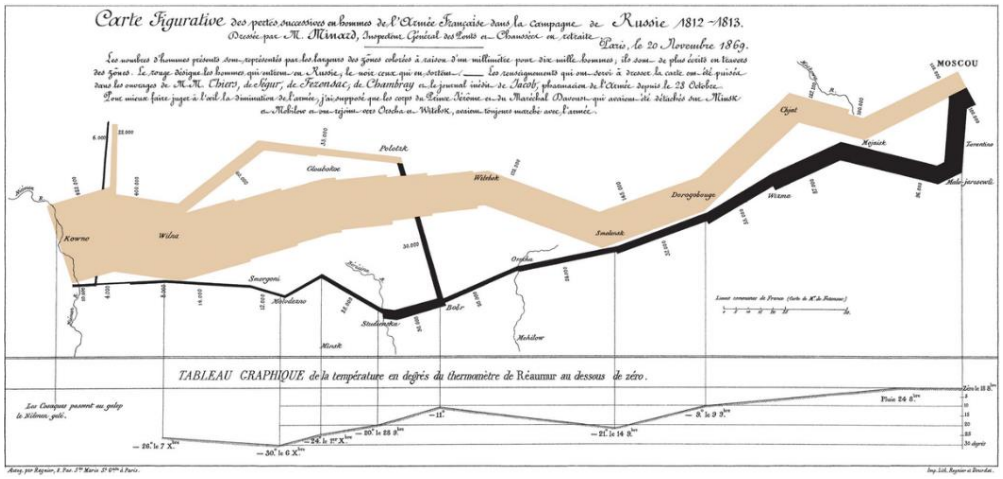
2.4 Datan visualisoinnin historia

Datan visualisointi ei ole uusi keksintö. Vanhimpia tunnettuja esimerkkejä tiedon visuaalisesta esittämisestä ovat luolien seinien kuvitukset, kuten Chauvet-Pont-d'Arc -luola, joka sijaitse lähellä Vallon-Pont-d'Arc:ia eteläisessä Ranskassa. Luolassa on yli 250 maalausta, jotka on tehty noin 30 tuhatta vuotta sitten. (Ward ym. 2010, 7.)

Varhaisia visualisointeja tehtiin tarpeeseen, ne liittyivät matkustamiseen, kaupankäyntiin, uskontoon ja viestintään. Peutingerin kartta Rooman ajalta kuvaa 70 tuhatta mailia Rooman valtakunnan teitä. Siitä käy ilmi teiden pituuksia, maamerkkejä ja etäisyyksiä tärkeiden kohteiden välillä. (Ward ym. 2010, 9.)

Koordinaatiston käyttö abstraktin datan esittämiseen oli yksi tiedon visualisoinnin läpimurroista. Tämä mahdollisti eri parametrien sijoittamisen koordinaatistoon sen sijaan, että olisi sijoiteltu vain kappaleita kartan koordinaatistoon. William Playfair, yksi datan visualisoinnin pioneereista, laati mm. viivakuvion, joka esitti valtion velan kehittymistä aikajanalla. X-akseli kuvasi aikaa ja y-akseli valtionvelan suuruutta. (Ward ym. 2010, 13.)

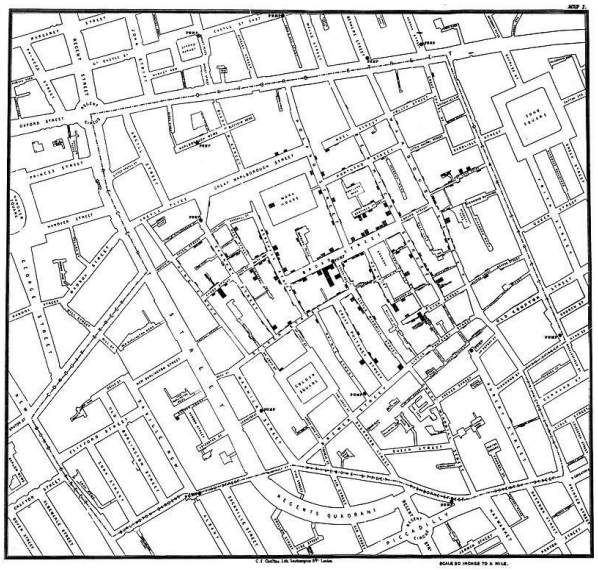
Eräs kuuluisa esimerkki datan monipuolisesta visualisoinnista on Joseph Charles Minardin kartta. Minardia voidaan myös pitää yhtenä ensimmäisistä moderneista infografiikan tekijöistä. Minardin laatima kaavio Napoleonin armeijan marssista kohti Moskovaa esitti aikasarjadataa maantieteellisiin paikkoihin yhdistettynä karttakuvana (kuvio 2). Kartasta käy ilmi, kuinka Napoleonin joukot kärsivät tappioita eri vaiheissa Venäjän kampanjaa. Kartan puumainen viiva kuvasi armeijan kokoa kyseisessä paikassa ja viivan väri ilmaisi joukkojen liikkeen suuntaa (kohti tai poispäin), myös lämpötila oli merkitty eri kohdissa joukkojen perääntymistä. (Ward ym. 2010, 12.)



Kuvio 2. Minardin kartta Napoleonin kampanjasta (Wikipedia 2018b)

Florence Nightingale laati 1800-luvulla kaavion, joka näytti armeijan joukkojen kuukausittaiset kuolemat Krimin sodassa. Hänen laatimastaan datan visualisoinnista kävi ilmi, kuinka paljon joukkoja kuoli kunakin kuukautena vuodessa ja kuolivatko joukot haavoihin, tauteihin tai jostakin muusta syystä. (Ward ym. 2010, 14; Biography 2018.) Hänen laatimaansa kaaviota kutsutaan coxcombin kaavioksi (engl. coxcomb chart), joka on eräänlainen pylväskuvion ja piirakkakuvion yhdistelmä.

Aikainen esimerkki eksploratiivisesta visualisoinnista on lontoolaisen lääkärin John Snown kartta koleraepidemiasta (kuvio 3). John Snow keräsi dataa kartoittaakseen koleraepidemian leviämistä Lontoossa vuonna 1854. Snow merkitsi kolerakuoleman uhrit karttaan Sohon alueella, ja onnistui vakuuttamaan viranomaiset siitä, että epidemia oli lähtenyt liikkeelle saastuneesta vesikaivosta Broad Street kadulla. (Ferster 2013, 3-4; UCLA Department of Epidemiology.)



Kuvio 3. John Snown kartta Lontoon kolerakuolemista (Wikimedia commons 2018)

2.5 Datan visualisointi nykyaikana

Viime vuosisatoina visualisointia on käytetty pääasiassa erilaisten tilastojen esittämiseen paperille piirrettyinä tai painettuina kuvina. Nykyään visualisoinnit toteutetaan useimmiten tietokoneella ja ne voivat myös usein olla vuorovaikutteisia. Tietokoneiden ja tietokonegrafiikan myötä visualisointien käyttäminen on yleistynyt huomattavasti. Tietokoneiden laskentateho ja tallennuskapasiteetti ovat mahdollistaneet valtavien tietomäärien tarkan visualisoinnin, kuten erilaiset maantieteellisten mittausten visualisoinnit sekä tieteelliset ja arkkitehtuuriset visualisoinnit. Visualisointeja käytetään nykyaikana paljon monilla tutkimuksen ja tieteen aloilla sekä myös monilla muilla aloilla. Myös erilaiset topografiset kartat liikennejärjestelmistä kuten metro- tai bussilinjoista ovat yleisiä, ja ne muistuttavat Harry Beckin Lontoon 1931 vuoden metrokarttaa (Glancey 2015). Vastaavanlaisia visualisointeja tehdään myös erilaisista prosesseista sekä järjestelmistä teollisuuden ja tuotannon aloilla. Myös koulukirjoissa ja oppimateriaalissa käytetään usein visualisointeja selventämään erilaisia abstrakteja asioita. Internetin myötä saataville on tullut lukuisia uusia tapoja tehdä ja käyttää visualisointeja. Kirjastoilla kuten D3.js on mahdollista laatia erilaisia datan visualisointeja verkkosivuille. Google maps -palvelulla voi selata karttoja. Ohjelmilla kuten Tableau ja PowerBI on mahdollista luoda erilaisia datan visualisointeja, joista käyttäjä voi koostaa päivittyviä dashboard-näkymiä organisaationsa käyttöön.

2.6 Datan visualisointi ja datatyypit

Dataa voi olla usean tyyppistä ja datatyyppien määritelmiä ja luokitteluja on myös useita. Ben Shneiderman (1996, 337-339) määritteli vuonna 1996 kirjoituksessaan “The Eyes Have It: Task by Data Type Taxonomy for Information Visualizations” seitsemän eri datatyyppiä, joiden Shneiderman totesi itse määrittelevän, kuinka ne tulisi visualisoida. Hän määritteli ne seuraavasti, niille sopivan visualisointitavan mukaan:

- Yksiulotteinen data (engl. one-dimensional data).
- Kaksiulotteinen data (engl. two-dimensional data).
- Kolmiulotteinen data (engl. three-dimensional data).
- Ajallinen data (engl. temporal data).
- Moniulotteinen data (engl. multidimensional data).
- Puurakenne (engl. tree data).
- Verkostorakenne (engl. network data).

Heer, Bostock ja Ogievetsky (2010, 61-67) määrittelevät artikkelissaan “A Tour Through the Visualization Zoo” visualisointityypit datan ominaisuuksien pohjalta:

- Aikasarjadata (engl. time-series data).
- Tilastolliset jakaumat (engl. statistical distributions).
- Kartat (engl. maps).
- Hierarkiat (engl. hierarchies).
- Verkostot (engl. networks).

Ward, Grinstein ja Keim (2010, 46-47) kirjassaan “Interactive data visualization” taasen määrittelevät datan tyypit seuraavasti: data voi olla järjestyksellistä (engl. ordinal) tai nimellistä (engl. nominal). Heidän määritelmässään järjestyksellinen data jakautuu alityyppeihin binäärinen data (0 tai 1), diskreetti data (esim. 2, 4, 6) ja jatkuva data (esim. luvut väliltä 0-5). Nimellinen data taasen jakautuu seuraaviin alatyyppeihin: kategorinen data (engl. categorical data) tarkoittaa dataa, joka on valittu tietystä joukosta ennalta määriteltyjä arvoja (esim. punainen, sininen, vihreä), luokiteltu data (engl. ranked data), joka tarkoittaa kategorista dataa, jolla on järjestystä vihjaava merkitys (esim. pieni, keskisuuri, suuri) ja satunnainen data (engl. arbitrary data), joka sisältää satunnaisia arvoja, joilla ei ole mitään ilmeistä järjestystä.

Tämän lisäksi he määrittelevät toiseksi tavaksi kategorisoida datan muuttujia niiden matemaattisen skaalan. Heidän mukaansa kolme attribuuttia jotka määrittävät muuttujaa ovat:

- Järjestys. Tapa jolla data voidaan järjestää.
- Etäisyys. Tallennetun datan yksikköjen välillä voidaan mitata etäisyyksiä.
- Absoluuttisen nollapisteen olemassaolo. Järjestyksellisellä datalla on jokin tietty pienin arvo (painolla voi olla absoluuttinen nolla-arvo).

Heidän mukaansa skaala (kuvio 4) on tärkeä attribuutti, jota tulee tarkastella, kun suunnitellaan datan visualisointia, koska kaikkiin käytössä oleviin graafisiin ominaisuuksiin liittyy myös skaala.

MITTA-ASTEIKKOJEN NELJÄ RYHMÄÄ

Poliisin tietoon tullut rikollisuus eräissä Kainuun kunnissa 2012–2015

Kunta	Sija	2012	2013	2014	2015
Kajaani	1.	3 517	3 161	3 054	3 009
Sotkamo	2.	946	861	762	909
Kuhmo	3.	490	531	427	465
Suomussalmi	4.	407	397	352	305
Paltamo	5.	279	280	302	257

Vuosiluvut muodostavat **välimatka-asteikon**

Kuntien nimet muodostavat **laatuasteikon**

Kunnat on asetettu **järjestysasteikolle** rikosten määrän mukaan

Rikosten kappalemäärät sijoittuvat **suhdeasteikolle**

Kuvio 4. Mitta-asteikkoja esittelevä taulukko (Koponen ym. 2016, 48)

2.7 Visualisointiprosessi

Colin Ware (2013, 4) on jakanut visualisoinnin neljään perusvaiheeseen, jossa on muutamia palautesilmukoita. Vaiheet ovat seuraavat:

1. Tiedon kerääminen ja tallentaminen.
2. Esikäsittelyvaihe (engl. preprocessing stage), jossa data muunnetaan helpommin käsiteltävään muotoon. Yleensä tähän vaiheeseen kuuluu jonkinlaista datan vähentämistä, niin että haluttuja piirteitä saadaan näkyviin datasta. Datan eksploraatio (engl. exploration) on prosessi, jossa datan tarkkailtavaa ulottuvuutta tai alaryhmää vaihdetaan.
3. Datan sijoittaminen (engl. mapping) visuaaliseen esitykseen, joka saadaan aikaiseksi käyttämällä tietokonealgoritmeja, jotka tuottavat kuvan ruudulle. Käyttäjän syötteet voivat muuttaa datan sijoittamista, korostaa tiettyjä alaryhmiä tai muuttaa esityksen ulkoasua. Yleensä tämä vaihe tehdään käyttäjän tietokoneella.
4. Ihmisen havainnointi- ja kognitiivinen järjestelmä.

Visuaalisen Processing-ohjelmointikielen kehittäjä Ben Fry (Ferster 2013, 35) on kuvannut interaktiivisen visualisoinnin vaiheet. Fry on jakanut prosessinsa seitsemään vaiheeseen:

1. Hankkiminen (engl. acquiring). Datan hankkiminen aloittaa prosessin, jossa raakaa dataa tai informaatiota hankitaan digitaalisessa muodossa.
2. Jäsentely (engl. parsing). Jäsentelyvaiheessa raaka data muunnetaan muotoon, jossa sen elementit ovat esimerkiksi tietokannan kenttinä tai taulukkolaskentaohjelman sarakkeissa.
3. Suodatus (engl. filtering). Datasetistä poistetaan data, joka ei ole visualisoinnin kannalta kiinnostavaa.
4. Louhinta (engl. mining). Suodatetusta datasta voidaan löytää toistuvia kuvioita tilastollisia menetelmiä, kuten korrelaatiota ja regressiota käyttämällä. Tällaisia toimenpiteitä voivat olla esimerkiksi miesten ja naisten tulojen vertailu ajan kuluessa.
5. Esitys (engl. representing). Datan esittäminen visuaalisesti mahdollistaa datasta louhintavaiheessa löytyneiden kuvioiden kommunikoinnin erilaisia visualisointitapoja käyttämällä.
6. Jalostus (engl. refining). Esitystä jalostetaan iteratiivisesti palaamalla prosessin aiempiin vaiheisiin, riippuen siitä minkälaisia havaintoja on tehty.
7. Vuorovaikutus (engl. interacting). Visualisointiin on lisätty ominaisuuksia, jotka mahdollistavat käyttäjän muokata esityksen ulkoasua ja esitystapaa, niin että käyttäjät voivat tehdä havaintoja riippuvuussuhteista datassa ja ymmärtää paremmin esitettävää dataa.

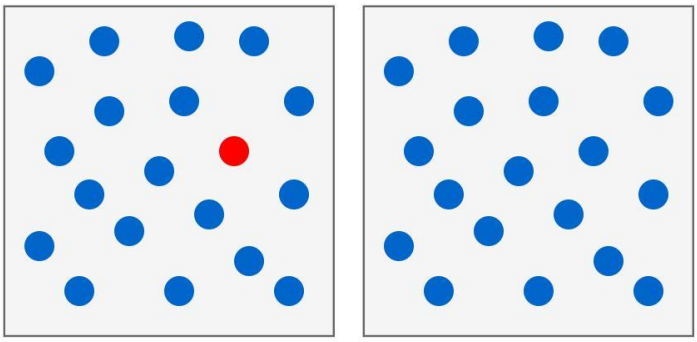
2.8 Näköaisti ja näköhavainto

Ihmisillä ja ihmisapinoilla on poikkeuksellisen kehittynyt näköaisti, mikä erottaa meidät useimmista muista nisäkkäistä, joille näön sijaan haju- ja kuuloaistilla on suurempi merkitys ympäröivän maailman hahmottamisessa. Ihmisen näköaivokuori on suurempi kuin kaikkien muiden aistimusten käsittelyä suorittavat aivojen osat yhteensä ja uuden tiedon omaksuminen on yleensä nopeinta juuri visuaalisessa muodossa. (Koponen ym. 2016, 17.) Ihmisen silmät näkevät sähkömagneettisesta spektristä ns. näkyvän valon alueen, eli alueen 380 nanometristä läheltä ultraviolettia aina 700 nanometriin asti, lähelle infrapunaaluetta. Näkökyvyn kannalta ihmisen silmän olennaisimmat osat ovat sarveiskalvo, iiris, pupilli, linssi ja verkkokalvo. Silmä on nesteen täyttämä pallo, jossa on valoherkkiä soluja

toisella puolella, ja toisella puolella aukko josta pääsee valoa silmään. Tämän lisäksi kunkin silmän liikettä kontrolloi kuusi lihasta. Silmän yhdistää aivoihin näköhermo. Silmän optinen järjestelmä muistuttaakin ominaisuuksiltaan kahdesta linssistä koostuvaa kamerajärjestelmää. (Ward ym. 2010, 78-80.)

Näköhavainto alkaa silmän verkkokalvolle päätyvästä valoärsykkeestä. Koska silmän verkkokalvo kehittyy suoraan aivojen soluista, sillä on myös kyky esikäsitellä verkkokalvolle syntyvää kuvaa. Verkkokalvo tekeekin jonkinlaista kuvan pakkausta. Tämä välitettävän tiedon pakkaaminen on tarpeen, koska näköhermoja on noin miljoona kappaletta, mutta näköhavaintoja vastaanottavia verkkokalvon sauva- ja tappisoluja on satakertainen määrä. (Ward ym. 2010, 86.) Havaintokykymme voi myös usein tehdä vääriä tulkintoja visuaalisista esityksistä, jos ne eivät vastaa havaintokykymme toimintaa, tai ne ovat tehty harhauttamaan havaintokykyämme (Ward ym. 2010, 74).

Ihmisen näköjärjestelmän toimintaa ja sen kuvien analysointikykyä on tutkittu pitkän aikaa. Yksi tutkimuksissa tehty tärkeä havainto on sellaisten visuaalisten piirteiden joukko, jotka visuaalinen järjestelmämme havaitsee matalalla tasolla erittäin nopeasti (kuvio 5). Näistä ominaisuuksista käytettiin alkuun käsitettä esitietoinen havainto (engl. preattentive processing), koska niiden havaitseminen vaikutti edeltävän keskittynyttä huomiota. Nykyisin tiedetään, että huomiolla on kriittinen rooli siinä, mitä ihminen havainnoi näköaistin avulla, mutta esitietoisen havainnon käsitettä käytetään edelleen, koska se antaa intuitiivisesti käsityksen nopeudesta ja helppoudesta, jolla kyseiset ominaisuudet tunnistetaan. (Healey & Enns 2012, 1170-1171.)

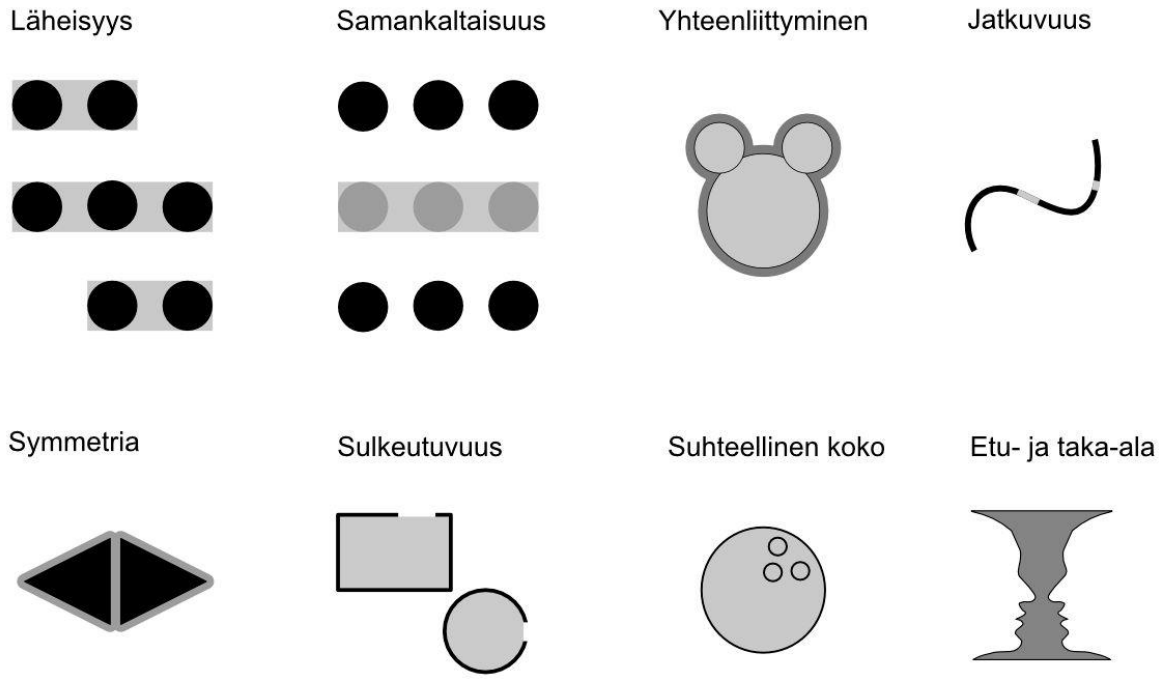


Kuvio 5. Näköjärjestelmämme havaitsee tiettyjä asioita paremmin kuin toisia

2.9 Hahmolait

Näköjärjestelmämme tunnistaa myös kappaleita ja niiden ryhmittymiä tietyllä tavoin ja tämän ymmärtäminen voi auttaa visualisointeja suunnitellessa. Ryhmä saksalaisia psykologeja perusti Gestalt-koulukunnan vuonna 1912 Saksassa. He tutkivat kuinka ihmiset tunnistavat kuvioita. Gestalt-koulukunta päätyi kahdeksaan hahmolakiin (kuvio 6), jotka määrittelevät kuvioiden tunnistamista. Nämä lait ovat seuraavat (Ferster 2013, 113):

- Läheisyys (engl. proximity). Kappaleet jotka ovat lähellä toisiaan, tulkitaan kuuluvaksi samaan ryhmään.
- Samankaltaisuus (engl. similarity). Kappaleet jotka ovat samankaltaisia, tulkitaan kuuluvaksi samaan ryhmään. Samankaltaisuus voi syntyä väristä, muodosta, koosta, pintakuviosta tai asennosta.
- Yhteenliittyminen (engl. connectedness). Kun kappaleet ovat liittyneet fyysisesti toisiinsa, ne tulkitaan kuuluvaksi samaan ryhmään.
- Jatkuvuus (engl. continuity). Toisiaan leikkaavat kappaleet tulkitaan yhdeksi kappaleeksi.
- Symmetria (engl. symmetry). Symmetriset kappaleet tulkitaan olevan osa samaa ryhmää.
- Sulkeutuvuus (engl. closure). Kappaleet joiden ääriviiva on osin toisen kappaleen peittämä, tulkitaan kokonaiseksi.
- Suhteellinen koko (engl. relative size). Kuvion pienet osat tulkitaan usein kappaleiksi, kun taas isot osat taustaksi.
- Etu- ja taka-ala/hahmo ja maa (engl. figure and ground). Edellä mainitut lait vaikuttavat kuvion tunnistukseen, mutta ne ovat riippuvaisia siitä, mikä tulkitaan etualaksi ja taustaksi.

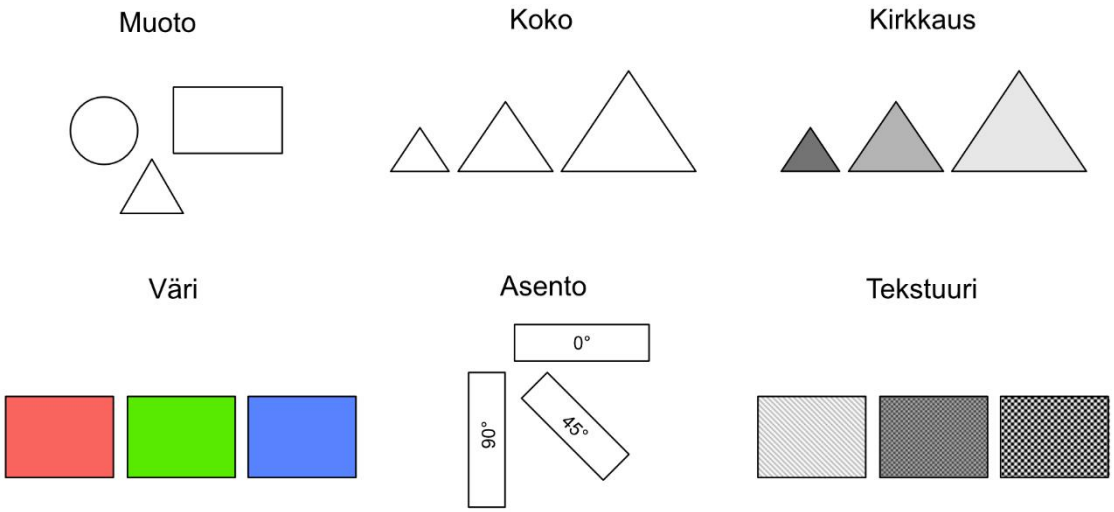


Kuvio 6. Havainnekuvat kahdeksasta hahmolaista

2.10 Visuaaliset muuttujat

Ranskalainen kartografi Jacques Bertin esitteli 1960-luvulla visuaalisten muuttujien teorian. Näillä visuaalisilla muuttujilla tarkoitetaan visuaalisia keinoja, joilla visualisoinnin tietopisteitä kuvaaviin elementteihin voidaan liittää tietoa. Havaintokykymme ominaispiirteistä johtuen näillä kuvailluilla visuaalisilla muuttujilla on paremmuusjärjestys, joka määrittää, kuinka tarkasti voimme lukea näillä muuttujilla koodattua tietoa. Alkujaan teoria oli kehitetty kartografian käyttöön, mutta sitä voidaan soveltaa muihinkin tiedon esittämisen tapoihin. (Koponen ym. 2016, 94.)

Ward, Grinstein ja Keim ovat käsitelleet tätä teoriaa visualisointiin liittyen, ja hieman laajentaneet tätä määritelmää. He määrittelevät kahdeksan eri tapaa, jolla graafiset kappaleet voivat koodata itseensä tietoa (kuvio 7). Näitä kaikkia ominaisuuksia voidaan käyttää visualisoinnin tiedonvälityksen tehokkuuden maksimointiin. He järjestävät nämä kahdeksan visuaalista muuttujaa seuraavalla tavalla tärkeysjärjestykseen: paikka, muoto, koko, kirkkaus, väri, asento, tekstuuri ja liike. (Ward ym. 2010, 137.) Alkuperäisessä teoriassa liike oli jätetty pois, koska teoria on laadittu ennen vuorovaikutteisten visualisointien yleistymistä (Koponen ym. 2016, 94).

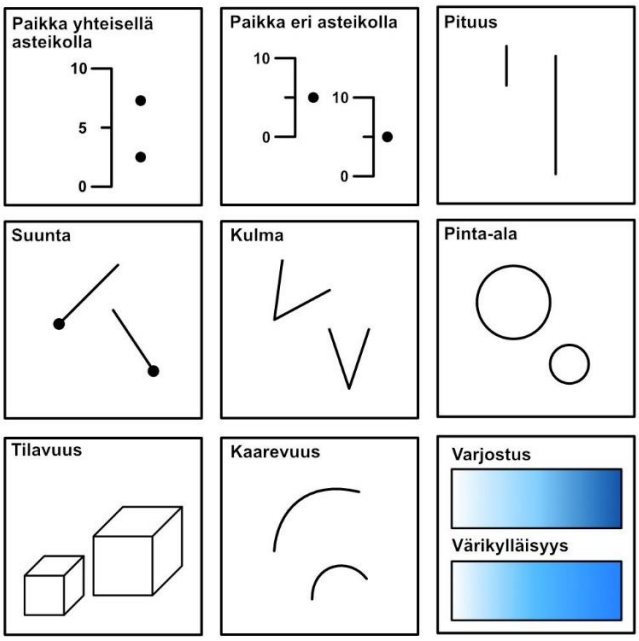


Kuvio 7. Havainnekuva yleisimmistä visuaalisista muuttujista

Cleveland ja McGill (1984, 531-533) ovat tutkineet visuaalisia muuttujia 1980-luvulla, heidän tutkimuksensa koski datan esittämistä ja tilastokuvioiden havainnointia. Tutkimuksessaan he esittelevät “elementary perceptual tasks” käsitteen. Heidän tavoitteenaan oli tunnistaa perushavainnot sekä niiden vuorovaikutus, jotta tätä havainnointia jota teemme voitaisiin ymmärtää ja ennustaa paremmin. He päätyivät tutkimuksensa perusteella kymmeneen perushavaintoon, joista yhtä tai useampia suoritamme, kun yritämme tulkita määrällistä tietoa kuvioista (kuvio 8). Nämä suoritettavat vaiheet eivät välttämättä ole itsenäisiä

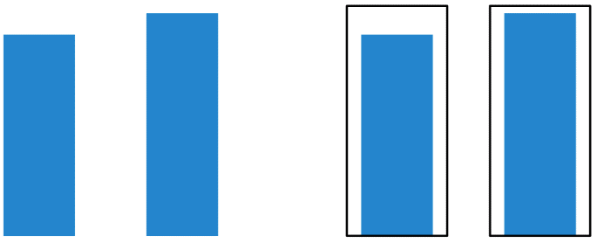
vaan osittain päällekkäisiä toimia, esimerkiksi kulman ja suunnan arvioiminen riippuvat toisistaan. Värien ja pintakuvion havainnointi oli jätetty tästä listasta pois. Heidän mukaansa, esimerkiksi kun katsomme piirakkakuviota, päätehtävä numeerisen tiedon saamiseksi on kulman havainnointi, mutta arvioimme myös todennäköisesti pinta-alaa ja kaaren pituutta piirakkakuvion viipaleista. He järjestivät nämä havainnot tutkimuksensa perusteella, kuinka tarkasti ihmiset keskimäärin pystyvät kutakin ominaisuutta havainnoimaan, järjestykseen tarkin-epätarkin:

- Paikka samalla mitta-asteikolla.
- Paikat erillisillä mitta-asteikoilla.
- Pituus, suunta, kulma.
- Pinta-ala.
- Tilavuus, kaarevuus.
- Varjostus, värin kylläisyys.



Kuvio 8. Mukaelma Clevelandin ja McGillin perushavainnoista (1984, 532)

Hahmotamme siis helpommin esimerkiksi vierekkäisiä pylväiden pituuksia, mutta hahmotamme huomattavasti huonommin vaikkapa ympyröiden tai neliöiden pinta-aloja (kuvio 9). Vielä pinta-aloja heikommin pystymme arvioimaan kolmiulotteisten kappaleiden välisten tilavuuksien eroja. Nämä asiat on syytä pitää mielessä erilaisia kuvioita ja visualisointeja laatiessa.



Kuvio 9. Havaintokykymme erottaa paremmin suhteellisia kuin absoluuttisia kokoja

2.11 Esitystavan merkitys

Yhtenä tärkeänä tekijänä esitystavan valinnassa voidaan pitää katsojaa. Cole Nussbaumer Knaflic on kirjassaan “storytelling with data” todennut, että ennen kuin visualisointia aletaan varsinaisesti luomaan, on tärkeää ensin ymmärtää tiedon kommunikoimisen konteksti. Knaflicin mukaan on hyvin tärkeää ymmärtää eksploratiivisen ja selittävän analyysin ero. Eksploratiivista analyysia tehdään, kun halutaan ymmärtää mitä huomionarvoista tai mielenkiintoista datassa voisi olla. Kun tietoa halutaan kommunikoida yleisölle, on tärkeää olla selittävässä tilassa, niin että esityksessä on selkeästi jotakin selitettävää tai tarina, joka halutaan kertoa. (Knaflic 2015, 33.)

Visualisointien tarkoitus on monesti auttaa päätöksenteossa tai saada katsoja ymmärtämään jokin abstrakti asia. Dataa voidaan esittää usein monilla eri tavoilla, mutta on tärkeää valita oikea visuaalinen esitystapa, niin että data saadaan koodattua graafisiin ominaisuuksiin kuten paikkaan, kokoon, muotoon tai väriin sopivalla tavalla.

Graafisten esitysten havainnointia tutkittaessa on huomattu, että tila-avaruudellinen sijainti johtaa täsmällisimpään numeerisen datan dekoodaukseen ja on yleisesti ottaen suositeltava vaihtoehto kulmaan, yksiulotteiseen pituuteen, kaksiulotteiseen pinta-alaan, kolmiulotteiseen tilavuuteen ja värikylläisyyteen verrattuna. Tästä johtuen valtaosa yleisesti käytetyistä datan visualisoinneista käyttää sijaintiin pohjautuvaa koodaustapaa. (Heer ym. 2010, 60.)

Vaikka esteettisyys ei ole visualisoinnin itsetarkoitus, on sen merkitys kuitenkin huomattava. Ward, Grinstein ja Keim (2010, 365-366) mukaan parhaat visualisoinnit ovat sekä informatiivisia, että miellyttäviä silmille. Toisaalta visualisointi voi jopa olla niin luotaantyöntävä, että se estää tiedon välittymisen. Esteettisesti miellyttävä visualisointi voi houkutella katsojan tutkimaan sitä tarkasti. Seuraavana on listattu asioita joita he ovat lainanneet graafisen suunnittelun ja taiteen aloita, joita tulisi huomioida viehättävän visualisoinnin toteutuksessa:

- Fokus. Katsojan huomio tulisi ohjata visualisoinnissa sen tärkeimpiin asioihin. Jos tärkeitä yksityiskohtia ei ole korostettu tarpeeksi, tämä voi hankaloittaa näiden tutkimista.
- Tasapaino. Ruudun pinta-ala tulisi käyttää tehokkaasti, ja tärkeimpien asioiden tulisi olla keskellä.
- Yksinkertaisuus. Yhteen näkymään ei tule yrittää ahtaa liian paljon tietoa. Myöskään kaikkia käytössä olevia graafisia keinoja ei tule käyttää itsetarkoituksellisesti.

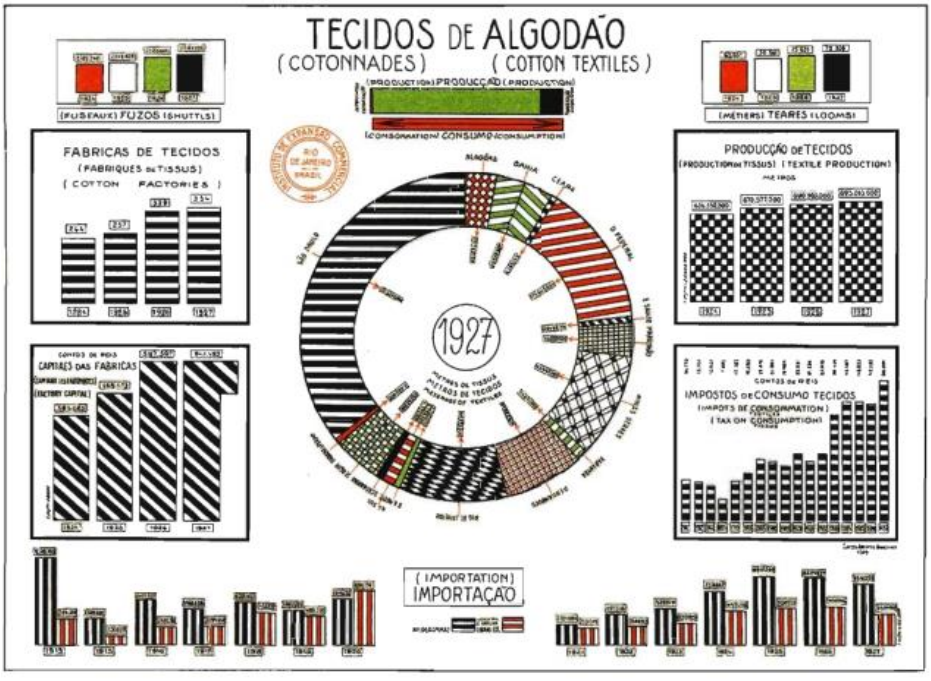
Edward Tufte (2001, 13) on listannut erinomaisen tilastografiikan olennaisimmat piirteet:

- Näytä data.
- Pyri saamaan katsoja ajattelemaan esityksen sisältöä, eikä sen esitystapaa.
- Vältä vääristämästä datan sanomaa.

- Esitä paljon numeroita pienessä tilassa.
- Tee suurista dataseteistä johdonmukaisia.
- Kannusta katsojan silmää vertailemaan dataa.
- Esitä data usealla eri tarkkuustasolla, yleiskuvasta yksityiskohtiin.
- Tarjoa selkeä tarkoitus - kuvaaminen, eksploraatio, taulukointi tai koristelu.
- Integroi grafiikka tiukasti datasetin tilastollisten ja sanallisten kuvausten kanssa.

2.12 Kuvioroina

Vaikka tiedon esittäminen on visualisoinnin keskeinen tehtävä, on esitystavan ja esteettisyyden lisäksi myös esityksen selkeydellä suuri merkitys tämän onnistumisessa. Kuvioroina (engl. chart junk) on Tuften määrittelemä käsite, jolla hän tarkoittaa kaikkea ylimääräistä visualisoinneissa, joka ei liity itse datan esittämiseen (kuvio 10). Tuften mukaan koristelun syyt vaihtelevat, tarkoituksena voi olla yritys esityksen saamisesta tieteellisemmän oloiseksi, elävöittää esitystä tai antaa graafiselle suunnittelijalle mahdollisuus käyttää taitojaan. Useimmiten Tuften mukaan kuvioroina ei kuitenkaan ole visualisointiin liitettyä taiteellista sisältöä vaan erilaista datan tarkastelua häiritsevää pintakuviointia ja muita silmää ärsyttäviä ratkaisuja, kuten esimerkiksi tahattomasti syntyneet optiset illuusiot. (Tufte 2001, 107-108.) Tufte kehottaakin välttämään kaikenlaista kuvioroinaa ja keskittymään itse esitettävään dataan (Tufte 2001, 121).



Kuvio 10. Esimerkki kuvioroinasta (Tufte 2001, 108)

2.13 Värien käyttö

Myös värien käyttöä tulisi miettiä tarkkaan visualisointia suunnitellessa. Värien käytöstä on kirjoitettu paljon graafisen suunnittelun sekä kuvataiteiden aloilla, ja samat lainalaisuudet

pätevät pitkälti visualisointeihin. Väreillä voidaan antaa merkityksiä asioille, niillä voidaan korostaa ja ryhmitellä kappaleita tai siirtää huomio haluttuun kohteeseen.

Koposen, Hildenin ja Vapaasalon mukaan värien käytön tulee olla johdonmukaista ja väriratkaisujen tulisi olla perusteltuja. Värit on myös valittava sisällön mukaan. Voimakkaita aksenttivärejä tulisi käyttää tärkeimpiin yksityiskohtiin, tausta- ja toissijaista informaatiota voidaan kuvata vaaleammilla ja vähemmän kylläisillä sävyillä. Värejä ei koskaan myöskään havaita absoluuttisesti, koska vierekkäiset värit vaikuttavat toisiinsa. Luotettavimmin havaitaan värit, jotka eroavat taustasta ja toisistaan riittävästi. Lisäksi värit eivät saa olla niin voimakkaita, että ne aiheuttavat häiritseviä kontrasti-ilmiöitä. (Koponen ym. 2016, 115.)

Värien käyttöä visualisointiin suunniteltaessa tulee muistaa, että osa ihmisistä voi kärsiä heikon näön lisäksi myös värinäön poikkeamista (kuvio 11). On arvioitu, että noin kymmenen prosenttia kaikista miehistä kärsii jonkin asteisista värinäön vajaavaisuuksista. Tätä ongelmaa voidaan lievittää visualisointeja toteuttaessa mm. huomioimalla seuraavia asioita (Ward ym. 2010, 364):

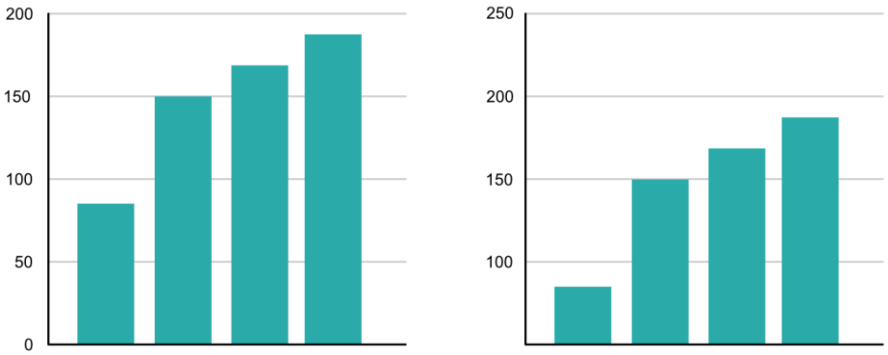
- Jos visualisoinnin tulkitseminen vaatii absoluuttista arviointikykyä, pidä eri vertailtavien arvojen määrä pienenä.
- Käytä useampaa keinoa arvojen esittämiseen. Jokin arvo voidaan esittää värin lisäksi koolla.
- Jos numeraalista tietoa välitetään värisävyillä, tulee huomioida, että sekä värisävy että kirkkaus vaihtuvat kussakin esityksen alkiossa.
- Lisää otsikoidut väriavaimet, joiden avulla käyttäjä voi tulkita näkemänsä värit.

Kuvio 11. Havainnekuva värinäön poikkeamista

2.14 Harhaanjohtavat esitykset

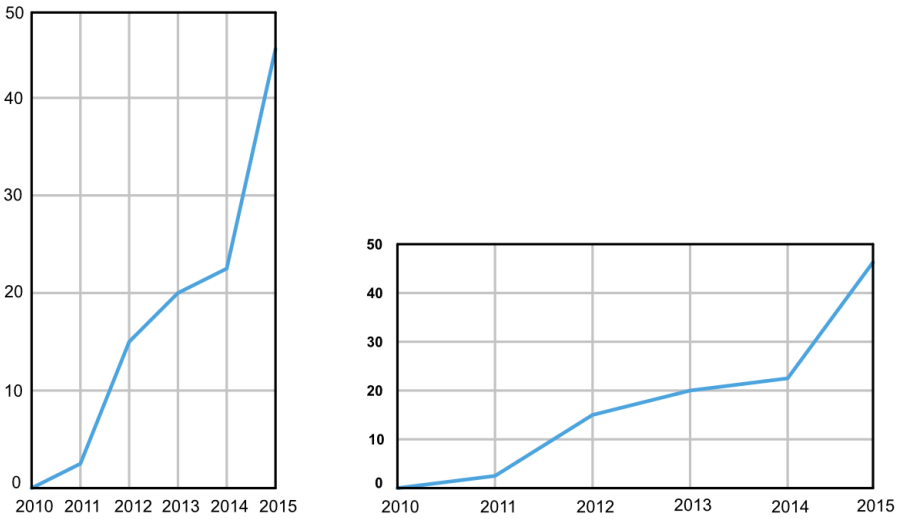
Dataa voi myös helposti esittää virheellisesti. Tämä voi tapahtua vahingossa tai tarkoituksellisesti, niin että datasta saadaan halutunlainen vaikutelma. Dataa voidaan myös valita vahingossa tai tarkoitushakuisesti niin, että esityksen lopputulos vääristyy, kun esimerkiksi tietyt ihmisryhmät puuttuvat datasta, jolloin esityksen lopputulos ei edusta esimerkiksi

koko populaatiota. Myös näennäisen riippuvuussuhteen esittäminen voi johtaa katsojaa harhaan. Mitta-asteikkojen leikkaaminenkaan ei ole suotavaa (kuvio 12). Esimerkiksi tilastoja pylväskuviolla esittäessä määräasteikon olisi syytä alkaa nollasta, koska asteikkoa typistettäessä pinta-alojen ja lukujen suhde vääristyy (Kuusela 2000, 113):



Kuvio 12. Pylväsdiagrammin katkaiseminen vääristää pylväiden keskinäisiä suhteita

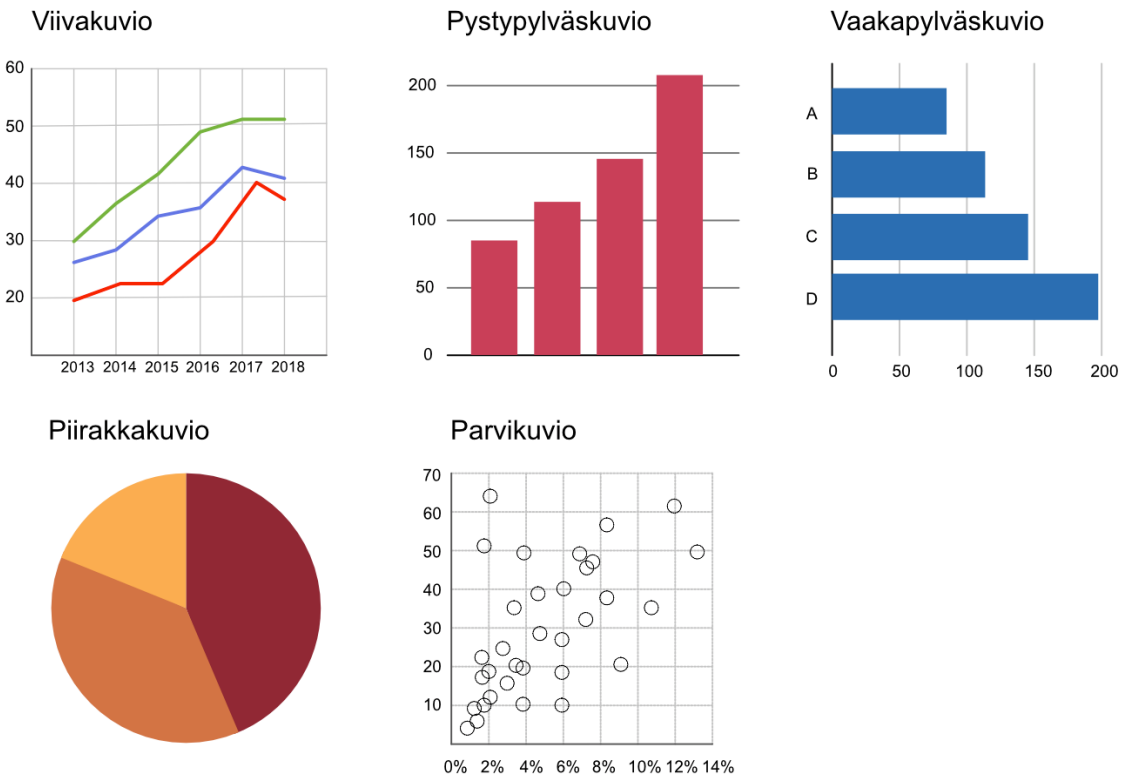
Myös kuvasuhde voi vaikuttaa kuvioiden esittämästä datasta tehtyihin havaintoihin. Viivakuvion välittämään sanomaan eniten vaikuttava tekijä on x- ja y-akselien asteikkojen välinen suhde, eli aspektisuhde (engl. aspect ratio). Jos aspektisuhdetta muutetaan, voidaan sama kuviossa esitetty muutos saada näyttämään suurelta tai vähäiseltä (kuvio 13). Yleinen ohje aspektin osalta on, että esitettävää kehitystä eli trendiä kuvatessa aspektisuhteen pitäisi olla sellainen, että tasaista muutosta kuvataan 45 asteen kulmassa kulkevalla viivalla. Tällaista aspektisuhdetta käytettäessä herätetään oikeantyyppinen mielikuva muutoksen luonteesta. (Kuusela 2000, 90-91.)



Kuvio 13. Aspektisuhteen valinta vaikuttaa esitettävän datan havainnointiin

2.15 Tavallisimpia tilastokuvioita

Erityyppisiä visualisointeja on lähes loputon määrä, mutta tässä opinnäytetyössä keskitytään visualisoimaan pääasiassa tilastokuvioita. Kuuselan mukaan suurin osa yleisimmin datan esittämiseen käytetyistä tilastokuvioista kuuluu viiva-, pystypylväs-, vaakapylväs- ja piirakkakuvioiden kategoriaan (Kuusela 2000, 49). Alla kuvassa nähdään tavallisimpia tilastokuvioita, Kuuselan listaamien lisäksi parvikuvio (kuvio 14).



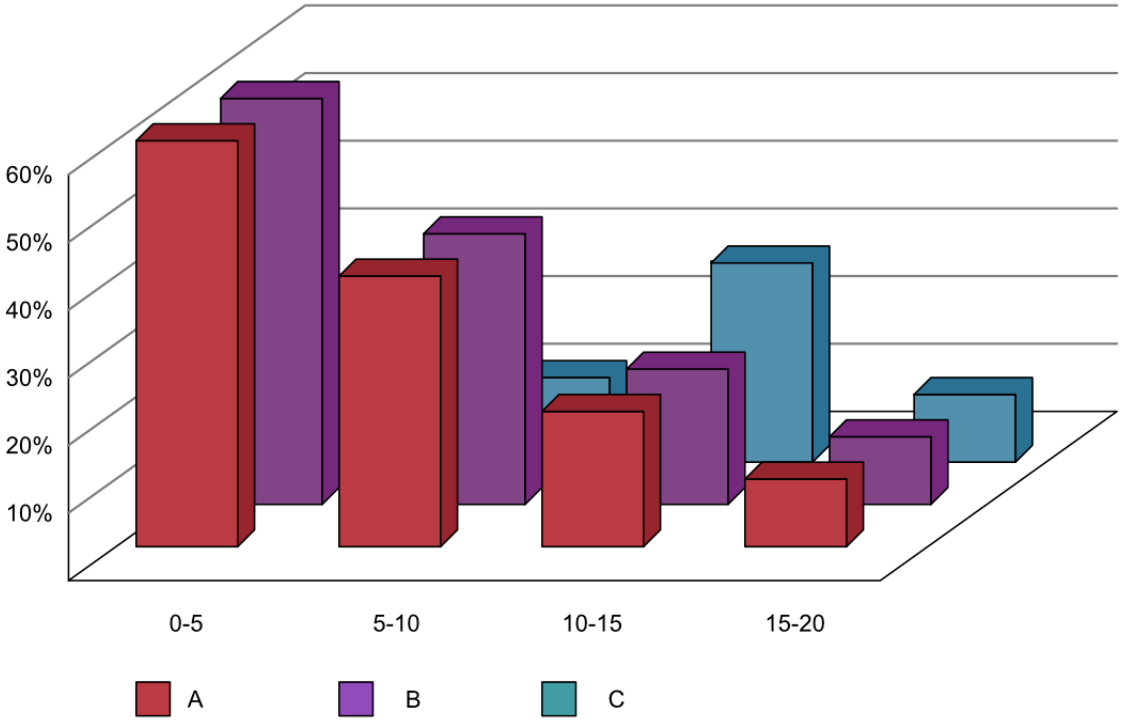
Kuvio 14. Tavallisimpia tilastokuvioita

2.16 Kolmiulotteiset kuviot

Tämän opinnäytetyön visualisoinnit toteutetaan kolmiulotteisella grafiikalla. Kolmiulotteisella grafiikalla voidaan esittää monia asioita, joita kaksiulotteisella grafiikalla ei ole mahdollista esittää. Kolmiulotteinen grafiikka voi olla näyttävämpää, mutta lisäulottuvuus tuo myös toteutettavaan visualisointiin haasteita, kuten aiemmin teoriaosiossa on jo tuotu ilmi. Kolmiulotteisuuden käyttöä tulee harkita ja sen ominaispiirteisiin liittyvät asiat tulee huomioida.

Koponen, Hilden ja Vapaasalo toteavat, että tilavuuksien hahmottaminen on ihmiselle vaikeaa. Esimerkiksi kolmiulotteisten pylväiden hahmottaminen onnistuu hyvin, mutta muiden muotojen ei. He suosittelevatkin, että visualisoinneissa on useimmiten syytä käyttää kaksi-

ulotteisia esityksiä. Ongelmia syntyy heidän mukaansa myös kolmiulotteisuudesta itsestään, perspektiivin vuoksi osa kappaleista voi kuvakulmasta riippuen jäädä piiloon toisten kappaleiden taakse (kuvio 15). (Koponen ym. 2016, 80.)



Kuvio 15. Kolmiulotteisessa esityksessä osa kuvioista voi peittyä toisten kuvioiden taakse

3 Unity-pelimoottori

Unity-pelimoottorin taustalla on yritys Unity Technologies SF. Yritys on perustettu alkujaan Tanskassa vuonna 2004 nimellä Over the Edge I/S (Crunchbase; CVR API). Sen nimi muutettiin vuonna 2006 muotoon Unity Technologies ApS. Yrityksen kasvun myötä se muutti pääkonttorinsa USA:han San Franciscoon ja muutti nimensä Unity Technologies SF:ksi vuonna 2009. Yrityksen perustajajäseniin kuului David Helgason, Nicholas Francis ja Joachim Ante (Haas 2014, 1). Kaikki kolme perustajaa vaikuttavat edelleen yrityksen toiminnassa. Alkujaan pelimoottori oli ryhmän oman pelin (GooBall) käyttöön tehty, mutta pelin epäonnistuttua kaupallisesti yritys arvioi tavoitteitaan uudestaan ja tunnisti pelimoottorin potentiaalin ja päätti alkaa myymään sitä pelien sijaan (Dahlberg 2017).

Unityn menestys on liitetty usein sen tarjoamaan mobiililaitetukeen ja yrityksen missioon pelinteon demokratisoinnista (Marketwired 2011). Unityä pidetään myös helposti lähestyttävänä. Yleinen konsensus vaikuttaa olevan se, että johtavista pelimoottoreista Unreal Engine on visuaalisella saralla johtava, Unityn tullessa perässä tältä osin, keskittyen enemmän mobiilialustoihin ja kattavampaan alustatukeen. Unity Technologies alkoikin tarjota tukea alustoille, joissa muut pelimoottorien valmistajat eivät vielä nähneet potentiaalia. Vuonna 2008 Unity oli yksi ensimmäisistä pelimoottoreista, joka tuki iPhone-puhelinta (Haas 2014, 9). Unity onkin ollut erityisen suosittu pienyritysten ja ns. indie-pelinkehittäjien parissa, jotka pyrkivät julkaisemaan ensimmäisenä tuotteita uusille alustoille. Unityn käyttökohteet eivät kuitenkaan rajoitu peleihin tai mobiililaitteisiin. Sitä on käytetty paljon erilaisten koulutusohjelmien, simulaatioiden, animaatioelokuvien, taideinstallaatioiden, VR- ja AR-ohjelmien toteuttamiseen.

Unity Asset Store on Unityn sisältökauppa. Unity Asset Store lienee myös yksi Unityn menestyksen tekijöistä. Se on käyttäjien välinen markkinapaikka, jossa käyttäjät voivat myydä itse tuottamaansa sisältöä ja ohjelmakomponentteja Unity projekteissa käytettäväksi, Unityn ottaessa prosenttiosuuden myynneistä. Unity Asset Store on avattu 2010 marraskuussa. 2014 huhtikuuhun mennessä sillä oli 600 000 rekisteröityä käyttäjää, jotka ostivat tai hakivat Unityn mukaan sieltä noin 500 000 asset-pakettia kuukaudessa. (Campbell 2014.)

3.1 Yrityksen kasvu

2012 Unity-pelimoottorin saavutettua suosiota, Unity kertoi, että yrityksellä on miljoona rekisteröitynyttä käyttäjää. Muutama vuosi myöhemmin vuonna 2015 Unity kertoi, että heillä

on 4.5 miljoonaa käyttäjää ja miljoona kuukausittain aktiivista käyttäjää. Yritys kasvoi kovasti suosion mukana.

Unity osti Applifier-nimisen yrityksen 2014 maaliskuussa. Tämän myötä Everyplay-palvelusta tuli Unityn Everyplay. (Kumparak 2014.) Applifierin GameAds-palvelusta tuli UnityAds.

2014 Unity osti myös Playnomics:in, joka oli data-analytiikka-alustaa tarjoava yritys (Spence 2014). Playnomicsista tuli Unity Analytics.

2014 Unity osti myös Tsugi:n, josta tuli sittemmin Unity Cloud Build palvelu (Andrew 2014).

2014 lokakuussa yrityksen kasvaessa, David Helgason siirtyi sivuun toimitusjohtajan roolista ja entisestä Electronic Arts:in toimitusjohtajasta, John Riccitiellosta, tuli Unityn toimitusjohtaja. Helgason jatkoi yrityksen "Executive Vice President" roolissa. (Conditt 2014.)

2015 Elokuussa Unity osti SilkCloud:in, Shanghailaisen e-commerce -kehittäjän, joka sittemmin on toteuttanut Unityn verkkopalveluiden infrastruktuuria (Jianding 2015).

3.2 Versiohistoria

Unity oli aluksi Mac-ohjelmisto ja siitä oli saatavilla erilliset iPhone- ja Mac-ohjelmistokehitykseen suunnatut versiot (Haas 2014, 8-9). Versiosta kolme alkaen eri alustojen versiot on integroitu yhteen editoriin ja Mac-version rinnalle tuli Windows-ympäristössä toimiva editori. Unityn versiohistoriaa ei kuvata tässä tarkemmin, mutta alla olevassa taulukossa on listattuna Unityn versioiden julkaisupäivämääriä (taulukko 1).

Taulukko 1. Unityn versioita (Internet Archive; Unity 2018a)

Unity 1.0.0	6. kesäkuuta 2005
Unity 2.0.0	10. lokakuuta 2007
Unity 3.0.0	27. syyskuuta 2010
Unity 4.0.0	13. marraskuuta 2012
Unity 5	3. maaliskuuta 2015
Unity 2017.1.0	10. heinäkuuta 2017
Unity 2018.1	2. toukokuuta 2018

3.3 Saatavuus

Unity on tällä hetkellä hyvin edullinen alusta pelien ja muun 2D- ja 3D-sisällön tuottamiseen. Alkujaan pelimoottori oli kertamaksullinen moduuleina ostettava kaupallinen tuote, mutta tällä hetkellä lisensointimallissa on siirrytty tilaajamalliin (Downie 2016). Editorista on tällä hetkellä tarjolla seuraavat versiot (Unity 2018b):

- Personal (ilmainen).
- Plus ($35/kk).
- Pro ($125/kk).
- Enterprise.

Pelimoottorin kaikki ominaisuudet ovat saatavilla jokaisessa versioissa (Unity 2018b). Kaikilla versioilla on mahdollista julkaista ohjelmia eri alustoille ilmaiseksi, mutta Unity Technologiesin liiketoimintamalli perustuu siihen, että tuotteen julkaisevan yrityksen tulot rajoittavat versioiden käyttöä. Esimerkiksi Plus-tason vuosittainen tuloraja tai joukkorahoituksesta saatujen tulojen raja on $200 000, jonka jälkeen on siirryttävä maksamaan Pro-versiosta. Pro-versio ei sisällä tulorajoituksia. Maksulliset versiot sisältävät myös joitain etuja - Unity Asset Store -sisältökaupasta on saatavissa alennuksia Plus- ja Pro-käyttäjille, ja oheispalvelut sisältävät laajemmat ominaisuudet. (Unity 2018b.) Enterprise-versio lienee suurimmalle osalle Unity-kehittäjiä tuntematon, se on käytännössä suunnattu suuremmille yrityksille, jotka tarvitsevat premium-tukea ja haluavat saada pääsyn Unity-pelimoottorin lähdekoodiin.

3.4 Lähestyttävyys

Unity-pelimoottorin käyttökynnys on matala. Unity-editorin voi ladata itselleen suoraan yrityksen sivuilta ilman rekisteröintiä. Kun editori avataan ensimmäisen kerran, tulee luoda käyttäjätili. Myös ilmainen Personal-versio vaatii rekisteröitymisen. Unityn suosituin ohjelmointikieli C# on suhteellisen helposti lähestyttävä. C# on Java-kieltä läheisesti muistuttava ohjelmointikieli (McCown 2011). C#-ohjelmointikielelle löytyy sen yleisyydestä johtuen runsaasti dokumentaatiota Unityn dokumentaation ulkopuolelta. Unity-editorista pääsee suoraan myös yrityksen sivuilla sijaitsevaan kattavaan ohjekirjaan docs.unity3d.com osoitteessa, josta löytyy ohjeet sekä editorin käyttöä että API-rajapintoja varten. Tämän lisäksi Unityllä on forum.unity.com osoitteessa sijaitseva suuri ja aktiivinen kehittäjäyhteisö, sekä kysymys ja vastaus -tyyppinen answers.unity.com-palvelu. Tämän lisäksi Unity-ohjelmointia käsitteleviä artikkeleita, videoita ja kirjallisuutta on runsaasti saatavilla.

3.5 Ohjelmointi

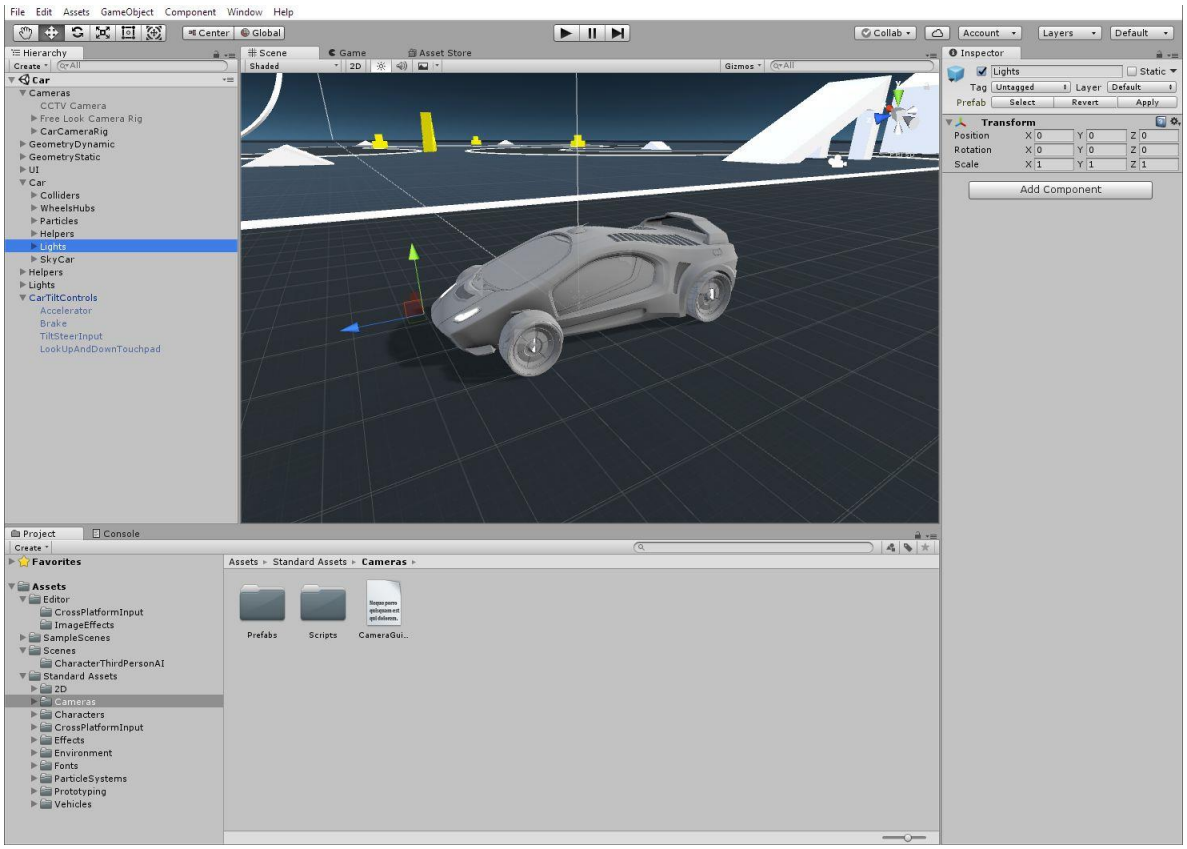
Unityn editori jolla ohjelmien kehittäminen tapahtuu, on saatavilla Windows-, Mac- ja Linux-alustoille, joskin Linux tuki on kokeellinen. Unity on pohjimmiltaan C++ runtime-ytimeen pohjautuva pelimoottori. Ohjelmointiin Unityn C++ API:a varten on ollut saatavilla UnityScript-, Boo- ja C#-kielet, joilla ohjelmakoodi yleisimmin kirjoitetaan. Unity on sittemmin lopettanut sekä UnityScript- että Boo-kielien tuen, C#-ohjelmointikielen otettua ylivoimaisesti suosituimman kielen paikan Unity käyttäjien keskuudessa (Fine 2017). Unity on rakenteeltaan suljetun lähdekoodin pelimoottori (Johansen 2009), toisin kuin Unreal Engine tai CryEngine -pelimoottorit. Tästä huolimatta, Unitystä on saatavilla joitain osia avoimena lähdekoodina, kuten esimerkiksi osa Unityn UI-koodista.

Unity -projektit luodaan ns. asset-tiedostoista editoria käyttämällä (Unity 2018c). Tämä käsite viittaa käytännössä projektin käytössä oleviin tiedostoihin, eli kaikki projektiin tuodut 3D-mallit, kuvat, äänet, tekstitiedostot ja muu data ovat asset-tiedostoja Unity -projektissa (Unity 2018d). Ohjelmalogiikan toteuttaminen tapahtuu Unityn tukemia ohjelmointikieliä käyttämällä. Ohjelmointi tehdään Unityn editorin ulkopuolella integroidussa ohjelmointiympäristössä. Unity tukee kehitysalustasta riippuen eri integroituja ohjelmointiympäristöjä, kuten Visual Studio Community Edition, Visual Studio Code tai MonoDevelop. MonoDevelop-tuki on loppumassa (Paczkowski 2018). Myös muiden koodieditorien käyttäminen on mahdollista. Valmiit ohjelmat käännetään Unityllä halutulle kohdealustalle. Kullekin alustalle kääntämistä varten tarvitaan myös kyseisen alustan ohjelmistokehityspaketti (software development kit, SDK) asennettuna. Kaikki kohdealustoja varten tarvittavat komponentit on mahdollista asentaa Unityn asennusohjelmaa käyttämällä automaattisesti.

3.6 Editori

Unityn editori on visuaaliselta ulkoasultaan hyvin pelkistetty. Perusnäkymä sisältää Project-ikkunan, Inspector-ikkunan, Scene-ikkunan, Hierarchy-ikkunan ja Toolbar-palkin (Unity 2018e). Käyttöliittymän toiminta on hyvin selkeä. Project-ikkunaan voi pudottaa tiedostoja, jotka kopioituvat projektin hakemistoon asset-tiedostoiksi (Unity 2018f). Hierarchy-ikkunaan voi tiputtaa Project-ikkunasta 3D-kappaleita, jolloin ne tulevat osaksi aktiivisena olevaa scene-tiedostoa. Scene-tiedosto on ikään kuin maailma johon kulloinenkin sisältö koostetaan, scene-tiedostoja voi olla niin monia kuin tarve vaatii. Editor-ikkuna tarjoaa näkymän scene-tiedoston 3D-tilaan, jossa 3D-kappaleita, 2D-kappaleita ja muita elementtejä voi sekä manipuloida että liikutella (kuva 1). Inspector-ikkuna mahdollistaa Project, Scene tai Hierarchy-ikkunoissa valitun kappaleen ominaisuuksien muokkaamisen ja ohjelmakomponenttien lisäämisen kappaleisiin. Game View -ikkuna tarjoaa näkymän, jonka loppukäyttäjä näkee ruudullaan.

Unityn editorin ikkunoiden asettelu on käyttäjän vapaasti muokattavissa. Ikkunoita voi siirrellä, lisätä, poistaa ja piilottaa modernien editorien tapaan kuten vaikkapa Photoshopissa. Lisäksi vähemmän käytettyjä ikkunoita eri tehtäviin löytyy menupalkin Window-menusta. Unityn editoria on myös mahdollista laajentaa C#-ohjelmointikielellä kirjoitetuilla editori-skripteillä. Mahdollisuuksia on monia, käytännössä lähes jokainen asia, mitä Unityn käyttöliittymästä ja editori-ikkunasta löytyy, on mahdollista toteuttaa skriptauksella (Sohail 2017). Editorityökalujen suurin ero muihin ohjelman sisältöön käytettyihin C#-skripteihin on se, että ne sijoitetaan Editor-kansioihin projektissa, jolloin ne eivät myöskään tule mukaan varsinaiseen julkaistuun tuotteeseen. Unity tarjoaa itse myös osan mainostamistaan ominaisuuksistaan Unity Asset Store -sisältökaupasta ladattavina paketteina, sen sijaan että ominaisuudet olisivat kiinteästi integroituna osaksi editoria.



Kuva 1. Unityn editori ja sen ikkunat

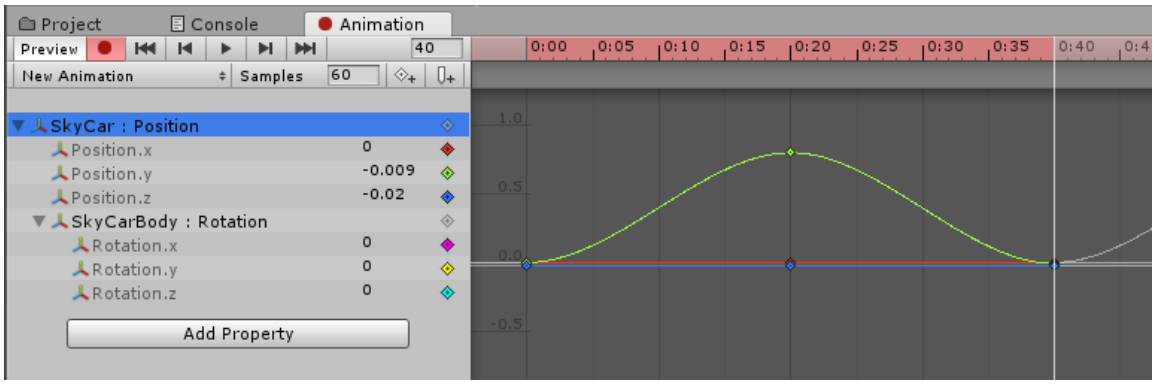
3.7 Grafiikkaominaisuudet

Unity käyttää eri grafiikkarajapintoja riippuen mille kohdealustalle ohjelma julkaistaan. Direct3D (Windows, Xbox One), OpenGL (Linux, Mac, Windows), OpenGL ES (Android, iOS), WebGL (verkkoselaimet) kuuluvat tuettuihin rajapintoihin, mutta myös erilaisia suljet-

tuja rajapintoja (Proprietary APIs) on tuettu eri konsoleilla. Myös matalan tason grafiikkarajapinnat (low-level API) kuten Metal (iOS) ja Vulkan (Android, Linux, Windows) ovat tuettuna jossakin määrin. Sisällöntuottajan tai pelintekijän ei tarvitse ymmärtää grafiikkarajapinnoista välttämättä juurikaan mitään. Näihin liittyviä asetuksia voi tarvittaessa säätää julkaisuasetuksista (Unity 2018g). Suurin osa Unityn graafisista ominaisuuksista toimii kaikilla kohdealustoilla, mutta kohdealustasta ja sen rajoituksista riippuen tiettyjä ominaisuuksia ei ole tuettu. Rajoitukset käyvät ilmi ohjekirjasta. Unity osaa myös emuloida valittua kohdealustaa editorissa (Unity 2018h).

Unityn grafiikkaominaisuudet ovat kohtuullisen kattavat, se tarjoaa modernit perusominaisuudet. Unity on luonteeltaan pääasiassa 3D-sisältöön painottunut pelimoottori. Alkujaan 2D-tuki oli heikko, mutta nykyisin Unity tarjoaa suhteellisen kattavat 2D-ominaisuudet. Aivan viimeisimpänä Unity lisäsi 2D-tiilikartta tuen (Shafir 2018). Käytännössä Unityn tarjoama 2D-grafiikka on 3D-monikulmioilla toteutettua, eikä pelimoottori sisällä varsinaista 2D-renderöijää kuten esim. Godot Engine. Sekä 2D- ja 3D-ominaisuuksia laajentavia asset-paketteja on saatavilla joko ilmaiseksi tai maksua vastaan Unity Asset Store -sisältökaupasta. Animaatiota varten Unity tarjoaa 2D- ja 3D-fysiikkamoottorit (Box2D, PhysX) (Unity 2018i). Unity tarjoaa myös partikkelieditorin, jolla käyttäjä voi luoda haluamansa kaltaisia partikkelianimaatioita. Unityn editori sisältää myös työkaluja valaistukseen ja 3D-ympäristöön liittyen. Lightmap-valaistustyökalut mahdollistavat valokarttojen ominaisuuksien määrittelyn ja niiden laskemisen (Unity 2018j). Reflection Probe ja Light Probe -työkalujen avulla on mahdollista muokata valaistuksen ominaisuuksia. Lisäksi editori tarjoaa työkaluja navigaatio-ominaisuutta varten (NavMesh) ja maasto-ominaisuuksia varten (Terrain). Kameratyökalut (Cinemachine), värinkäsittelytyökalut (Post Processing Stack) ja lisätyökalut sisällön rakentamiseen (ProBuilder ja Polybrush) löytyvät Unity Asset Store -sisältökaupasta ilmaisina paketteina (Unity Asset Store 2018).

Unityn editori tarjoaa myös animaatio-ominaisuudet (kuva 2). Animaatio-ominaisuudet ovat suhteellisen alkeelliset verrattuna vaikkapa 3D-animaatio-ohjelmien ominaisuuksiin, mutta ne mahdollistavat 3D- sekä 2D-elementtien, partikkelien tai näiden komponenttien arvojen animoinnin. Tämän lisäksi Unity tarjoaa myös Timeline-työkalun NLE-tyyppiseen animaatioleikkeiden yhdistelyyn (Low 2017). Monimutkaisemmat animaatiot, kuten ihmishahmojen animaatiot on syytä tuoda 3D- tai 2D-animaatio-ohjelmistosta Unity-projektiin käyttämällä Unityn tukemia 3D- ja animaatioformaatteja, kuten Filmbox tai Collada.



Kuva 2. Unityn Animation-ikkuna

3.8 Muut ominaisuudet ja oheispalvelut

Unity tarjoaa myös moninpelituen backend-palvelulla varustettuna (Unityn palvelimilla), joskaan tätä ei ole pakko käyttää. Unityn editori tarjoaa myös suhteellisen kattavat ääniominaisuudet. Audio Source -kappaleita voi tiputella scene-tiedostoihin, lisäksi editorin käyttöliittymä tarjoaa Audio Mixer -ikkunan, jolla on mahdollista ryhmitellä ja muokata äänilähteiden ääniä eri efekteillä, äänimikserien tapaisesti (Unity 2018k).

Unity tarjoaa keskustelufoorumin (Unity 2018l). Tämän lisäksi Unityllä on monia muitakin palveluja. Suurin osa palveluista, kuten Unity Asset Store -sisältökauppa on suoraan saatavilla editorista. Palveluja on tällä hetkellä saatavilla ainakin alla mainitut (Unity 2018m):

- Unity Ads (integroidut mainokset, iOS ja Android).
- Unity Analytics (analytiikka integraatio ja dashboard).
- Unity Performance Reporting (suorituskykyanalytiikka).
- Unity Everyplay (sosiaalisia ominaisuuksia ja videotoisto).
- Unity IAP (mikromaksut, iOS, Android ym.).
- Unity Certification (ohjelmointiosaamisesta kertova sertifikaatti).
- Unity Collaborate (sisäänrakennettu versionhallinta).
- Unity Cloud Build (continuous integration pilvessä).
- Unity Multiplayer (moninpeli-backend).

3.9 Laite- ja alustatuki

Unityn tarjoama alustatuki on vaihdellut ajan saatossa. Jatkuvasti tuettuina ovat olleet Windows- ja Mac-työpöytäversiot. Pelikonsoleiden ja mobiililaitteiden (puhelin tai tabletti) tuki on vaihdellut konsolien ja mobiililaitteiden elinkaaren mukaan. Joitain äly-TV-laitteita on tuettuna. Uusimpina tuettuina laitteina ovat VR- ja AR-laitteet, joiden laitetuessa Unity on ollut eturintamassa verrattuna toisiin kaupallisiin pelimoottoreihin. Tällä hetkellä Unityn verkkosivujen mukaan 28 eri alustaa on tuettuna (Unity 2018n). Vanhempia alustoja poistuu niiden suosion tai elinkaaren mukaisesti, uusien korvatessa ne. Esimerkiksi Unityllä oli valtavan suosittu Unity Web Player, jonka 139 miljoonaa pelaajaa oli asentanut verk-

koselaimeensa vuonna 2008 (Takahashi 2012). Web Player on sittemmin poistunut käytöstä, ja se on osittain korvattu WebGL-tuella. Samoin Adoben aikanaan suositun Flash-alustan tuki on poistunut jo käytöstä (Helgason 2013). Vaikka tuotteiden julkaisu eri alustoille onkin ilmaista, osa tuetuista alustoista vaatii julkaisusopimuksen tai luvan julkaisualustan omistajalta.

Unityllä tehtyjä ohjelmia ja tuotoksia on mahdollista julkaista usealle eri alustalle kohtuullisella vaivalla Unityn pääosin automatisoidun build-prosessin ansiosta. Unity tarjoaa pilvipalveluna Cloud Build -palvelua (Minotti 2015). Cloud Build -palvelua käyttämällä on myös mahdollista yksinkertaistaa build-prosessia alustoilla, jotka saattavat vaatia enemmän osaamista kuten iOS, jolla build-prosessin loppuosa muutoin tapahtuu Applen XCode-ohjelmointiympäristössä.

Unity on panostanut uusien alustojen ja tekniikoiden tukeen kovasti. Vision Summit 2016 -tapahtumassaan helmikuussa 2016 Unity Technologies kertoi perustamastaan Unity Labs osastosta, joka keskittyy Unityn VR- ja AR-ominaisuuksien kehittämiseen (Chevalier 2016). Erilaisten VR- ja AR-kokeilujen tai sovellusten toteuttaminen Unityllä on helppoa, koska Unity ja laitevalmistajat ovat julkaisseet kehitystyökaluja näitä alustoja varten. Unity tukeekin lukuisia VR- ja AR-alustoja, alla lista tämänhetkisistä tuetuista alustoista (Unity 2018n):

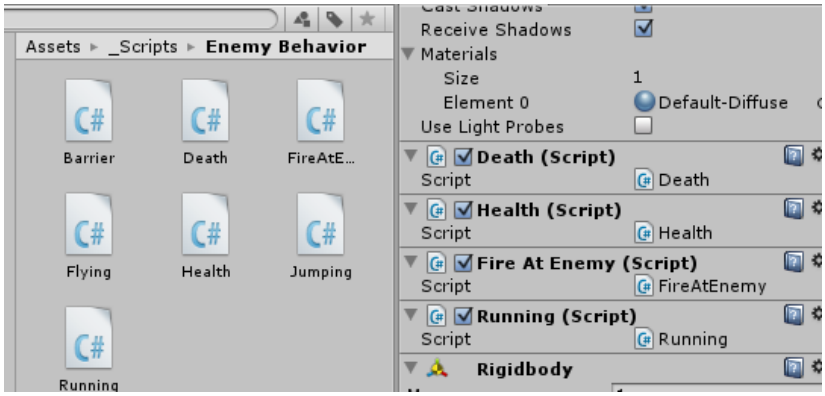
- Oculus VR (PC).
- SteamVR (PC/Mac).
- Apple ARKit (iOS).
- Google Cardboard (Android/iOS).
- Google ARCore (Android).
- Playstation VR (Sony Playstation).
- Windows Mixed Reality (PC).
- Gear VR (Samsung-laitteet).
- Daydream (Android).

Joillekin alustoille on saatavilla Unity Remote -tuki (kuten Apple iOS). Unity Remote on ohjelma, joka ladataan laitteeseen. Tämän jälkeen sen ruudulle voidaan siirtää ajettavan ohjelman kuva Unityn editorista, ja laitteesta siirretään käyttäjän tekemät syötteet takaisin editoriin. Näin ohjelmaa voidaan testata ilman kääntämistä. Unity Remote -alustatuki on muuttunut ajan kuluessa, ajantasaista tietoa löytyy Unityn omilta sivustoilta. (Unity 2018o.)

3.10 Komponenttiajattelu

Unity käyttää komponenttiajattelu-lähestymistapaa (Porter 2013). Tästä johtuen tarvittavat toiminnallisuudet eivät useinkaan vaadi suurta määrää koodia. GameObject-kappaleisiin voidaan lisätä valmiita tai itse kirjoitettuja komponentteja (kuva 3), jotka kaikki pohjautuvat

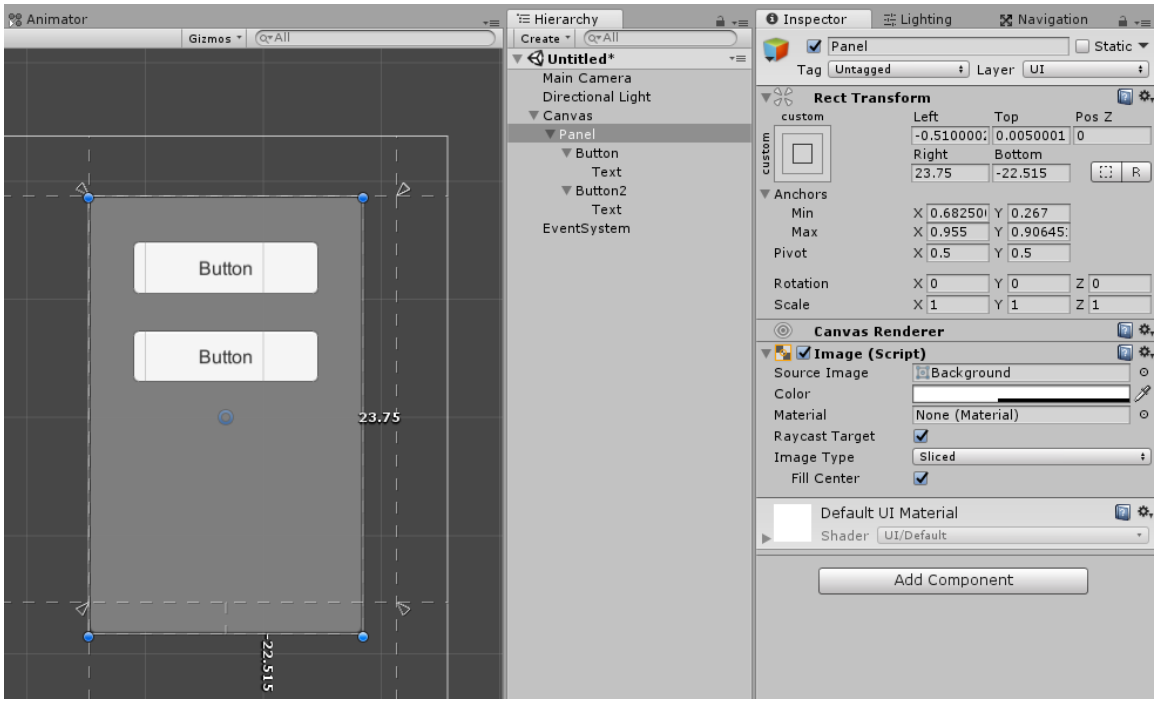
MonoBehaviour-luokkaan (Unity 2018p). Komponenttiajattelu mahdollistaa myös useiden erilaisten loogisten rakenteiden ja toiminnallisuuksien kierrättämisen toiseen projektiin tai muokkaamisen uuteen käyttötarkoitukseen. Myös erilaisten ohjelmointikehysten kuten Entitas (Schmid) tai mallien kuten MVC soveltaminen on mahdollista (Dunstan 2015). Ohjelman esitystapa voidaan esimerkiksi rakentaa vaihdettavaksi, niin että käytössä on 3D-malleja käyttävä esitys tai vaikkapa kokonaan GPU-ratkaisulla toteutettu esitys. Kirjoitetut komponentit myös käännetään automaattisesti editorissa (Bergeron 2017). Lisäksi ohjelmaa voidaan ajaa editorissa, joten erilaisia muutoksia voidaan kokeilla nopeasti, ilman että koko ohjelmaa tarvitsisi kääntää toistuvasti.



Kuva 3. GameObject-kappaleisiin voidaan rakentaa toiminnallisuutta MonoBehaviour-komponenteilla (Porter 2013)

3.11 Käyttöliittymien rakentaminen

Unityn käyttöliittymä-komponentit (UI) mahdollistavat erilaisten otsikoiden, tekstien, nappien, numeroiden, listojen, popup-ikkunoiden sekä muiden graafisten käyttöliittymäkomponenttien lisäämisen 3D- ja 2D-tilaan (kuva 4) (Unity 2018q). Jos koetaan että esityksestä puuttuu jokin UI-komponentti, voidaan tällainen toteuttaa UI-järjestelmää laajentavana komponenttina ja lisätä vaikkapa ruudulla esitettäviin kappaleisiin. Unityn UI-komponentit eivät kuitenkaan kilpaile laadullaan Qt-ohjelmistokehyksen tai vastaavien 2D-käyttöliittymiin erikoistuneiden ohjelmistokehysten kanssa. Vaikka Unityn ominaisuudet ovat lähinnä suunnattu erilaisten vuorovaikutteisten mittaristojen ja menujen toteuttamiseen, kohtuullisen monimutkaistenkin graafisten käyttöliittymien toteuttaminen on mahdollista.



Kuva 4. Unityn UI-järjestelmä käytössä

3.12 Puutteet

Vaikka Unity sisältääkin paljon ominaisuuksia, on se saanut myös kritiikkiä puutteistaan. Ominaisuuksien määrä näyttää välillä menneen tärkeysjärjestyksessä tarjottujen työkalujen laadun ohi. Alla listatut puutteet ovat pitkälti opinnäytetyön kirjoittajien omia havaintoja ja näkemyksiä. Esimerkiksi 3D-pintamateriaalien mallintamiseen tarkoitettua visuaalista shader-editoria ei ole ollut vielä toistaiseksi saatavilla. Animaatio-editori on ollut puutteellinen. Hahmojen animointityökalut on koettu puutteellisiksi. Valaistustyökalut ovat olleet puutteellisia tai bugisia. Visuaalista ohjelmointia vuokaavion tapaan ei ole ollut tarjolla. Lähes kaikkiin näihin on kuitenkin löytynyt Unity Asset Store -sisältökaupasta jonkinlaisia kolmansien osapuolien toteuttamia ratkaisuja. Lisäksi suurimpia osia näistä puutteista on vuoden 2018 huhtikuuhun mennessä yritetty paikata yrityksen toimesta Unityn beta-versioissa.

3.13 Soveltuvuus datan visualisointiin

Unityn pelimoottori-ominaisuudet mahdollistavat hyvin erilaisen visualisointien kehitys- ja toteutustavan, verrattuna perinteisiin tietokoneella toteutettuihin visualisointiratkaisuihin. Unity mahdollistaa uusien toiminnallisuuksien nopean kokeilun vuorovaikutteisesti ja se mahdollistaa myös visualisointiohjelmien julkaisun mobiililaitteista AR- ja VR-laitteisiin. Unityn Editor-ikkuna mahdollistaa 3D-kappaleiden lisäämisen, muokkaamisen, poistamisen ja muiden muutosten tekemisen vuorovaikutteisesti editorissa myös ohjelmaa ajettaessa, joskin rajoituksin. Erilaisia editorin käyttäjälle toiminnallisuuksia lisääviä skriptejä

voidaan myös ohjelmoida, jolloin editorissa voidaan näyttää erilaisia Gizmo-ominaisuuksilla toteutettuja mittareita ym. joita ei välttämättä julkaistussa visualisoinnissa edes ole. Ohjelmakoodia voidaan tarvittaessa ajaa myös editoritilassa, kun editorin toimintaa tarvitsee muokata tai laajentaa. Unityn verkko-ominaisuudet mahdollistavat datan lataamisen ajon aikana reaaliaikaisesti verkosta.

Pelimoottorina Unity tarjoaa paljon ominaisuuksia vuorovaikutteisuuden toteuttamiseen. Unity tukee useita erilaisia peliohjaimia ja laajennuksien avulla myös muitakin rajapintoja (kuten MIDI), joiden avulla erilaisia ohjainlaitteita voidaan käyttää Unityssä. Peliohjainten ja muiden ohjainlaitteiden toimintoja on helppo asettaa kontrolloimaan haluttuja parametreja, näitä voidaan käyttää esimerkiksi visualisoinnissa esitettävän datan manipulointiin tai kuvakulman muuttamiseen ohjelman ajon aikana. Lisäksi Unity sisältää tai siihen on saatavilla Unity Asset Store -sisältökaupasta sekä Unityltä että kolmansilta osapuolilta erilaisia hahmo- ja interaktiokontrollereita. Näiden avulla voidaan toteuttaa esimerkiksi 3D-tilassa tapahtuvaa liikkumista, kappaleiden ja kuvakulman manipulointia, kosketus- ja elekontrolleja, ilman että tällaisia järjestelmiä tarvitsisi suunnitella ja toteuttaa tyhjästä.

3.14 Visualisointitekniikat ja skaalautuvuus

Unity tukee myös shaderien ja kuvaefektien toteuttamista korkeantason rajapinnan (high-level shading language, HLSL) kautta (Unity 2018r). Shadereita voidaan testata reaaliaikaisesti Unityn editorissa, ilman tarvetta kääntää sovellusta. Uusimmassa versiossa saataville tulee myös visuaalinen materiaalieditori, Shader Graph (Cooper 2018). Nämä ominaisuudet mahdollistavat graafisen ulkoasun muokkaamisen ja erilaisten esitystapojen kehittämisen visualisointiin sopivaksi. Unityssä tällaisten muokkausten ja laajennusten toteuttaminen on todennäköisesti huomattavasti yksinkertaisempaa, kuin näiden toteuttaminen yksittäiseen visualisointiohjelmaan.

Unityn scene-tiedostot ja lähes kaikki ruudulla näytetyt 3D-kappaleet rakentuvat GameObject-rakenteiden päälle (Unity 2018s). GameObject-rakenne on itsessään suhteellisen raskas suorituskyvyn kannalta. Tästä johtuen Unity ei pysty esittämään ruudulla kovin montaa kappaletta kerrallaan. Unity selviää useista sadoista tai vaikkapa noin tuhannesta GameObject-kappaleesta PC-tietokoneella (wibble82 2015). Se ei kuitenkaan selviä kymmenistä tuhansista samanaikaisista GameObject-kappaleista. Tällaisissa tilanteissa ruudunpäivitysnopeus todennäköisesti tippuu huomattavasti tehokkaallakin työasemalla. Esimerkiksi suppeamman tilastodatan esittämiselle tästä ei välttämättä tule estettä. Toisaalta jos tarkoituksena on tehdä big datasta visualisointeja ja pyrkimyksenä on saada kymme-

niä tuhansia, satoja tuhansia tai kenties miljoonia dataelementtejä ruudulle, joudutaan kehittämään erilaisia ratkaisuja, joilla tällaiset visualisoinnit voidaan toteuttaa. LOD-järjestelmät (level of detail, tarkkuustaso) ja tavat segmentoida esitettävää dataa ovat mahdollisia toteuttaa C#-kielellä, mutta tällaisia toiminnallisuuksia ei ole valmiiksi tarjolla Unityssä. GPU-laskennan hyödyntäminen Unityssä on mahdollista käyttämällä mm. compute shadereita, jolloin laskentaa voidaan siirtää GPU:lle (Unity 2018t). Tämä mahdollistaa suuremmat kappalemäärät rinnakkaislaskennan avulla, mutta tämä vaatii käytännössä luopumista ainakin osin Unityn tarjoamasta GameObject-mallista. Tähän liittyen, Unityyn on tulossa uutena ratkaisuna Unity Technologies EntityComponent ja Job-System -ominaisuudet, jotka tuovat helpotuksia mm. suurien kappalemäärien esittämiseen (Rincon 2018).

3.15 Unityn tukemat tiedostoformaatit

Unity tukee tällä hetkellä pääasiassa pelin graafisen sisällön, animaatioiden ja äänien pelimoottoriin tuomiseksi tarvittavia tiedostoformaatteja, mutta hyvin rajoittuneesti muun tyyppisen sisällön tiedostoformaatteja. Tuetut 3D-tiedostoformaatit pystyvät lataamaan pääasiassa vain kappaleiden ulkoasuun liittyviä tietoja kuten verteksien paikat, monikulmiot, pinnan varjostus, UV-koordinaatit ym. joten niiden hyödyntäminen muuhun käyttöön voi vaatia luovuutta.

3D-sisällön tiedostoformaateista Unity pystyy lukemaan .fbx (Filmbox), .dae (Collada), .3ds (3D Studio), .obj (Wavefront) ja .skp (SketchUp) tiedostomuotoja. Lisäksi Unity pystyy tuomaan 3D-dataa 3ds Max, Maya, Blender, Cinema 4D, Modo, Lightwave ja Cheetah3D ohjelmistoista niin, että tiedostoja tuodessa ne muunnetaan .fbx muotoon. Näitä jälkimmäisenä mainittuja tiedostomuotoja tuodessa kyseisen ohjelman täytyy olla asennettuna tietokoneella, johon Unity on asennettu. (Unity 2018u.)

Puhtaasti dataan keskittyviä formaatteja ei ole juurikaan tuettuna (kuten CSV tai XML). Tämä ei kuitenkaan tarkoita sitä, että dataa ei saataisi Unityyn, mutta käytännössä on turvauduttava Unityn ohjelmointirajapinnan ulkopuolisiin C#-ohjelmointikielen ominaisuuksiin, C#-kirjastoihin sekä Unity Asset Store -sisältökaupasta saataviin laajennuksiin tai ohjelmoitava itse tarpeeseen sopiva laajennus.

Tekstidataa voidaan ladata Unityn omaa Text Asset -luokkaa käyttämällä, mutta vaikka Text Asset tukee näennäisesti useita eri tiedostoformaatteja kuten txt, XML ja JSON, on tiedostojen sisällön parserointi kuitenkin toteutettava itse. JsonUtility-luokan avulla voidaan lukea JSON-dataa (Unity 2018v). Käytännössä JSON-datan muodon tulee vastata tarkalleen luokan rakennetta, johon data ladataan (Unity 2018w). JsonUtility ei siis ole

yleiskäyttöinen. Unityn WWW-luokka, joka on käytännössä POST- ja GET-pyyntöjä toteuttava luokka, voidaan käyttää datan lataamiseen internetistä (Unity 2018x). Sitä voidaan käyttää datan hakemiseen tietokannasta esimerkiksi PHP-kielellä rakennetun backendin kautta.

Dataa Unityyn tuodessa voidaan käyttää hyödyksi myös C#-ohjelmointikielen ominaisuuksia. Tekstimuotoista dataa voidaan ladata Unityyn esimerkiksi käyttämällä C#-ohjelmointikielen System.IO-nimiavaruuden StreamReader-luokkaa (Microsoft 2018). C# tukee myös lukuisia tyypillisiä rakenteita kuten Dictionary, List, ArrayList jne. joita voidaan hyödyntää ladatun datan käsittelyssä. Lisäksi Mono C# sisältää myös Microsoftin Linq -ominaisuuden (Language-Integrated Query), jolla datarakenteista voidaan tehdä SQL-kyselyjen kaltaisia hakuja (Microsoft 2015). Unityyn on saatavilla myös lukuisia kolmansien osapuolien kaupallisia, ilmaisia tai avoimen lähdekoodin laajennuksia erilaisten datatieteen alalla käytettävien tiedostoformaattien lataamiseen kuten CSV, JSON, XML ja Excel.

4 Visualisointiprotyyppien suunnittelu

Tässä toiminallisen osuuden ensimäisessä luvussa käsitellään Unity-pelimoottorilla toteutettavien 3D-visualisointiprototyyppien suunnittelua, vaatimuksia sekä käytettyjä työkaluja ja ohjelmistoja.

Toteutettavien prototyyppien avulla on tarkoitus selvittää, mitä etuja ja mahdollisuuksia liittyy 3D-ympäristössä toteutettuun datan visualisointiin. Ohjelmointiin käytetään C#-ohjelmointikieltä ja Unity-pelimoottorin ominaisuuksia. Lopputuloksena on tuotos, joka on kaksi vuorovaikutteista 3D-visualisointiprototyyppiä, sekä näiden luomista kuvaava raportti.

Toiminnallinen osuus jakautuu osiin: Tämän luvun alaluvussa suunnittelu (4.1) kuvataan toteutettavien 3D-visualisointiprototyyppien suunnittelua, rajauksia ja toiminnallisten vaatimusten määrittelyä, joiden avulla saadaan muodostettua kuva siitä, minkälaisia näistä prototyypeistä tulee. Seuraavassa alaluvussa (4.2) käsitellään toteutukseen käytettyjä työkaluja ja ohjelmistoja, sekä kuvataan lyhyesti niiden valintaperusteita. Tämän jälkeen kahdessa seuraavassa luvussa (5 ja 6) kerrotaan 3D-visualisointiprototyyppien toteutuksesta. Näissä luvuissa käydään läpi kaikkien olennaisien komponenttien toteutus ja toiminta sekä kerrotaan, miksi näihin ratkaisuihin päädyttiin. Tuotos luvussa (7) kuvataan, minkälainen kokonaisuus 3D-visualisointiprototyypeistä saatiin aikaiseksi, sekä arvioidaan niiden laatua ja käytettyjä ratkaisuja.

4.1 Suunnittelu ja vaatimukset

Alustavien kokeilujen perusteella päätettiin tehdä kaksi erityyppistä visualisointia. Datan esitystavoiksi valittiin pylväs- ja parvikuvio, koska ne ovat hyvin yleisiä ja tunnettuja tilastokuvioita. Lisäksi niiden toteuttaminen on suhteellisen yksinkertaista, eikä käytettävien muotojen toteuttamiseen 3D-tilassa tarvita monimutkaista matematiikkaa tai algoritmeja. Erilaisten hierarkioita esittävien graafien (kuten sosiaaliset verkostot ja rakenteet) visualisointi rajattiin pois, koska niiden käyttäminen vaatisi esimerkiksi voimaohjattujen graafinpiirtoalgoritmien toteuttamista, joka voisi osoittautua liian haastavaksi.

Visualisoinneissa käytettiin yleisiä ja helposti saatavilla olevia datasettejä. Pylväsdiagrammivisualisointiin datasetiksi valittiin avoindata.fi kautta aluesarjat.fi palvelusta haettua Helsingin seudun väestötieto-dataa. Parvikuvion visualisointiin käytettiin Ronald Fisherin iris-datasettiä, joka on erittäin hyvin tunnettu multivariaatti-datasetti, jota käytetään usein erilaisissa harjoituksissa, jotka liittyvät datan tutkimiseen, tilastotieteeseen tai koneoppimiseen (UCI Machine Learning Repository).

Näiden 3D-visualisointiprototyyppien toiminnalle määriteltiin seuraavia sekä toiminnallisia että ei-toiminnallisia vaatimuksia:

Toiminnalliset vaatimukset:

- Visualisoinnin on kyettävä lataamaan esitettävää dataa tiedostosta.
- Sen on kyettävä esittämään data kolmiulotteisesti ruudulla.
- Sen on tarjottava käyttäjälle kontrollit, joilla 3D-tilassa olevaa dataa voidaan tarkkailla eri kuvakulmista.
- Käyttäjän on voitava rajata esitettävää dataa ja/tai vaihtaa sen esitystapaa.
- Visualisoinnin tulee esittää selkeästi ja reaaliaikaisesti valitun dataelementin tiedot.
- Visualisoinnin tulee esittää mitta-asteikko tai koordinaattiruudukko, joka antaa apua datan tulkitsemiseen.
- Graafisen käyttöliittymän ja grafiikan on huomioitava kohdelaitteen kuvasuhde.

Ei-toiminnalliset, laadulliset vaatimukset:

- Visualisoinnin on oltava helposti lähestyttävä ja pelkistetty.
- Visualisoinnin käytön on oltava itsensä selittävä.
- Käyttöliittymän on oltava selkeä.
- Kameran kontrollien on oltava helppokäyttöiset.
- Visualisoinnin väripaletin on oltava miellyttävä.
- Värisuunnittelussa otetaan huomioon värinäön poikkeamat.
- Ruudunpäivityksen tulee olla sulavaa niin, ettei vuorovaikutteisuus kärsi.

Päätimme käyttää vaatimusmäärittelyn tukena käyttäjäpersoonia. Luotiin karkeat käyttäjäpersoonat, joiden avulla pyrittiin tekemään visualisointeja suunnitellessa ja toteuttaessa oikeanlaisia ratkaisuja. Käyttäjäpersoonat eivät esiinny tässä raportissa myöhemmin, niitä käytettiin työskentelyn apuna.

Käyttäjä A - tavalliset lehtiä ja uutisia lukevat ihmiset, joille visualisoinnit voivat tarjota lisäarvoa tai virikkeitä aiheeseen lisää perehtymiseen. Tälle kohderyhmälle suunnitellun ohjelman käyttötarkoitus on pystyä tuottamaan lehdissä tai verkkojulkaisuissa nähtyjen kaltaisia datan visualisointeja.

Käyttäjä B - datan visualisoinnista kiinnostuneet käyttäjät. Ohjelma tarjoaa uuden tavan katsella ja tutkia dataa 3D-tilassa. Tällä käyttäjäryhmällä on mahdollisuus ottaa ohjelman lähdekoodi ja muokata sitä haluamallaan tavalla jonkin toisentyyppisen datan visualisointiin.

Pylväsdiagrammivisualisointi toteutettiin suomen kielellä, ja yleiskäyttöisemmäksi kaavailtu parvikuviovisualisointi englannin kielellä.

4.2 Käytetyt työkalut ja ohjelmistot

Toiminnallinen osa aloitettiin työkalujen päivittämisestä ajan tasalle. Uusia työkaluja ei tarvinnut asentaa, koska kaikkia oli käytetty jo aiemmin. Unity päivitettiin hieman aiemmasta versiosta versioon 2017.3.1. Toiminnallista osaa aloittaessa Unityn viimeisin versionumero oli 2017.3.1. Opinnäytetyötä viimeistellessä ilmestyi versio 2018.1, mutta ei nähty syytä vaihtaa versiota kesken projektin. Unity-käyttäjätilejä ei tarvinnut luoda, koska molemmat opinnäytetyöntekijät olivat käyttäneet Unityä aiemmin.

Unity-pelimoottorin lisäksi tarvittiin ohjelmointia varten jokin integroitu kehitysympäristö. Käyttöön valittiin Visual Studio Code, lähinnä omien mieltymysten perusteella. Työn luonteesta johtuen esimerkiksi debug-toimintoja ei tarvittu, joten hieman yksinkertaisempi peruseditori riitti käyttöön, eikä tarvetta vaihtaa sitä kesken työn muodostunut. Visual Studio Code päivittyy itsestään, mutta sen versio oli lähes koko opinnäytetyön tekemisen ajan 1.22.1.

Lisäksi käytettiin Exceliä taulukkomuotoisen datan katseluun ja muutamia eri tekstieditoreita, kuten Atom ja Notepad++ datan käsittelyyn. Koska kyseessä on parityönä toteutettu opinnäytetyö, käytettiin apuna myös ryhmätyöskentelyyn sopivia työkaluja, kuten Google Docs, joiden avulla ryhmätyötä pystyttiin tekemään sujuvasti. Versionhallintaan käytettiin Git-versionhallintajärjestelmää. Tekijöiden aiemman kokemuksen perusteella sen on todettu olevan hyvä vaihtoehto pienen työryhmän versionhallinnan työkaluksi, erityisesti siitä johtuen, että se ei tarvitse palvelinta. Lisäksi tarvittavien 3D-kappaleiden toteutukseen käytettiin Blender 3D-ohjelmaa.

5 Pylväsdiagrammivisualisointi

3D-visualisointiprototyyppien toteuttaminen päätettiin aloittaa yksinkertaisemmasta eli pylväsdiagrammivisualisoinnista. Vaikka pylväsdiagrammi ei välttämättä lähtökohtaisesti pystyisi toimimaan parhaiten 3D-tilassa, haluttiin tätä kokeilla. Visualisoinnin scene-tiedoston rakenne selviää liitteestä 1.

Visualisoinnin tekeminen aloitettiin luomalla tyhjä Unity 3D-projekti. Sinänsä ei ole merkitystä mikä projektityyppi Unityssä valitaan, kyse on vain muutamasta editorin oletusasetukseen ja ulkonäköön vaikuttavasta asetuksesta. Visualisointia varten luotiin tyhjä scene-tiedosto, joka lisättiin saman tien Build Settings -ikkunan Scenes in Build -listaan. Nämä scene-tiedostot tulevat sisällytetyksi lopulliseen käännettyyn peliin tai ohjelmaan. Pääteltiin, että kumpikin visualisointi tulisi tarvitsemaan vain yhden scene-tiedoston, joten tähän asetukseen ei tarvinnut sittemmin koskea kummassakaan visualisoinnissa.

Seuraavaksi tehtiin kansiot eri tiedostotyypeille: Audio, Data, Graphics, Materials, Objects, Prefabs, Scripts, Shaders ja Textures. Kansioiden nimet voivat olla vapaavalintaisia, mutta nämä on havaittu hyviksi tiedostojen jäsentelyyn. Kansioiden nimet projektin sisällä muuttuivat hieman työn tekemisen aikana, mutta pääpiirteittäin käytettiin yllämainittuja.

5.1 Datan lataaminen ja parserointi

Koska visualisoinnin lähtökohtana on data, jota esitetään ruudulla, aloitettiin ohjelman komponenteista ensimmäisenä datan lataajasta. Käyttöön tuleva datasetti oli tyypiltään CSV-tiedosto, jolla tarkoitetaan käytännössä pilkulla tai jollain muulla välimerkillä erotettua, riveittäin tallennettua dataa tekstitiedostossa. Pylväsdiagrammivisualisoinnin projektin Data-kansioon kopioitiin aluesarjat.fi-palvelusta tulostettu tiedosto, jossa on Helsingin väestötieto ikäryhmittäin vuosilta 2008-2017, tiedoston nimi oli 02UM_Vakiluku_ikaryhmittain.txt.

Datan lataamista varten tehtiin DataParser C#-luokkatiedosto Scripts-kansioon. Sen yliluokaksi jätettiin Unityn perusasetuksena tarjoama MonoBehaviour-luokka. Tämä tarkoittaa sitä, että nämä luokat on mahdollista kiinnittää komponentteina Unityn scene-tiedostoissa ja projektikansioissa oleviin GameObject-tyyppisiin asset-tiedostoihin. Scene-tiedostoon lisättiin tyhjä GameObject-kappale nimeltään barsVisualizer, johon DataParser-luokka kiinnitettiin. Teoriaosassa oli jo selvitetty vaihtoehtoja datan lataamiselle. Tekstitiedoston lataamiseen Unityyn päätettiin käyttää TextAsset-tyyppistä muuttujaa.

Datatiedosto olisi voitu sijoittaa myös Resources-kansioon ja ladata se käyttämällä .NET-ohjelmistokehyksen luokkia, mutta public-tyyppinen TextAsset-muuttuja mahdollisti tekstitiedoston lisäämisen Data-kansiosta Inspector-ikkunan kenttään. Tekstitiedosto ladattiin suoraan projektin kansiosta, mutta jos kyse olisi ollut jostakin muusta lähteestä, olisi voitu käyttää WWW-luokkaa datan lataamiseen esimerkiksi internetistä. TextAsset mahdollisti siis sen, että .NET-ohjelmistokehyksen System.IO nimiavaruudessa sijaitsevia StreamReader- tai File-luokkia ei tarvittu tekstin lukemiseen.

CSV-datan arvojen erottelua varten lisäsimme seuraavat muuttujat: separator-muuttujaa käytettiin datan yksiköt erottavan merkin määrittelyyn (puolipilkku) ja lineEnd-muuttujaa taasen käytettiin rivinvaihtojen tunnistamiseen (kuva 5).

```
public string separator = ";";
public string lineEnd = @"\r\n";
```

Kuva 5. Muuttujat CSV-datan lukemista varten

CSV-tiedoston rakenne oli seuraavanlainen: ensimmäinen rivi sisältää sarakkeiden otsikot. Kaikki seuraavat rivit tiedoston viimeiseen riviin asti sisältävät eri ikäryhmien vuosittaiset tiedot. Data ei siis ollut pelkkää sarakedataa, jossa otsikkorivit olisivat määritelleet kunkin alemman rivin sarakkeen sisällön. Se ei siis ollut tallennettu tavalla: kaupunki, ryhmä, vuosi ja määrä. Alla ote CSV-tiedostosta (kuva 6).

```
1   kaupunki;ryhma;2008;2009;2010;2011;2012;2013;2014;2015;2016;2017
2   Helsinki;0-6-vuotiaat;36922;37971;39170;40064;41127;42224;43444;44552;45379;45807
3   Helsinki;7-12-vuotiaat;30001;29693;29410;29318;29506;30054;30680;31523;32684;33730
4   Helsinki;13-15-vuotiaat;16853;16743;16268;15840;15336;15128;15067;14927;14991;15069
5   Helsinki;16-19-vuotiaat;24365;24928;24994;24901;24817;24290;23802;22971;22458;22429
6   Helsinki;20-29-vuotiaat;99115;100952;102121;103172;104422;106018;107095;108369;108914;109434
7   Helsinki;30-39-vuotiaat;86147;87502;89694;91535;94560;98296;102290;105432;107669;109121
8   Helsinki;40-49-vuotiaat;82380;83362;83241;82449;81694;80766;79456;78850;78935;79559
9   Helsinki;50-64-vuotiaat;112067;112830;113661;113204;112237;111812;112002;112382;113114;114040
10  Helsinki;65-74-vuotiaat;42179;43771;45478;48163;51417;54439;57061;58976;60865;61173
```

Kuva 6. CSV-dataa puolipilkuilla eroteltuna

Dataa olisi voitu pivotoida ja käsitellä eri tavoin esimerkiksi Excelissä, mutta data päätettiin kuitenkin pitää siinä muodossa kuin se saatiin. Todettiin, että Unity ei tarjoa juuri ollenkaan työkaluja tällaisen taulukko- tai tilastodatan käsittelyyn ja muokkaamiseen. Erilaisia ratkaisuja etsiessä selvisi, että esimerkiksi Python-kielelle on olemassa mm. Pandas-kirjasto, jolla datalle voi tehdä erilaisia muokkaustoimintoja komentoriviltä, hieman samaan tapaan kuin Excelissä on mahdollista tehdä graafisen käyttöliittymän kautta.

Luettava data päätettiin tallentaa datan sisältöä vastaavaan luokkaan. Käytettävä luokka on tavallinen C#-luokka, jonka ilmentymiin kukin rivi tietoineen tallennetaan (kuva 7).

```
[System.Serializable]
public class ItemData
{
    public string kaupunki;
    public string ikaRyhma;
    public int vuosi;
    public int maara;
}
```

Kuva 7. C#-luokka luettavaa dataa varten

Lisäksi tarvittiin rakenteita, joihin ladattu data tallennetaan. List-luokan ilmentymä parsedData tulisi pitämään sisällään rivien datasta tehdyt ItemData-oliot. List on C#-ohjelmointikielen mukainen geneerinen luokka, ja <> merkkien välissä sen tallentavan datan tyypiksi on määritelty ItemData-luokka. Keys-nimistä Array-rakennetta vuorostaan käytettiin selkeyden vuoksi apuna otsikkorivin sarakkeiden sisältöön viitatessa. Lisäksi ageGroups-niminen List-rakenne lisättiin, jotta lista ikäryhmistä olisi kokonaisuudessaan helposti saatavilla visualisointia varten. Seuraavassa kuvassa nähdään käytettyjä rakenteita (kuva 8):

```
public List<ItemData> parsedData;
public string[] keys;
public List<string> ageGroups;
```

Kuva 8. Rakenteita ladattua dataa varten

Tämän jälkeen DataParser-luokkaan lisättiin Parse-metodi, joka vastaa TextAsset-tiedoston lataamisesta ja muuntamisesta ItemData-olioiksi. CSV-datan otsikkorivi pilkotaan ja tallennettaan keys Array-rakenteeseen. Tämän jälkeen for-silmukalla käydään läpi kaikki tekstitiedoston datarivit. Kukin rivi pilkotaan separator-muuttujan perusteella, käyttämällä C#-ohjelmointikielen Regex-luokan Split-toimintoa. Kaksi ensimmäistä saraketta joka rivillä määrittävät kaupungin ja ikäryhmän. Tämän jälkeen kunkin rivin seuraavat sarakkeet määrittävät vuosikohtaisen väkimäärän tälle ryhmälle. Tätä varten for-silmukan sisälle lisättiin toinen for-silmukka, jossa kerätään datat ja luodaan ilmentymiä ItemData-luokasta, ja tallennetaan nämä parsedData List-rakenteeseen.

Käytännössä DataParser-luokasta tuli hyvin tapauskohtainen, sillä ei oikeastaan voi ladata kuin samalla rakenteella varustettua dataa eri vuosilta. Se ei tarjoa mitään yleistä ratkaisua taulukkodatan lataamiseen, koska ItemData-luokka ja yksinkertainen for-silmukka eivät salli juurikaan eroja ladattavalle datalle. Tavoitteena oli keskittyä visualisointiin, ei

datan lataamiseen tai datarakenteisiin. Toisaalta se on kuitenkin helposti muunnettavissa samankaltaisen datan lataamiseen.

5.2 Datan visualisoinnin valmistelu

Jotta ladattua dataa pystyttäisiin esittämään 3D-tilassa, tarvittiin jokin luokka, jolla voidaan muuttaa List-datana oleva data 3D-tilassa esitettävään muotoon. Ensin scene-tiedostoa muokattiin hieman. Kamera siirrettiin visualisoinnin kannalta sopivaksi oletettuun paikkaan. Unityn perusasetuksena näkyvä horisonttitausta poistettiin asettamalla kameran GameObject-kappaleen Camera-komponentista Clear Flags “Solid Color” tyyppiseksi ja sen väri muutettiin neutraalin harmaaksi. Tämän lisäksi pääteltiin heti, että kameran FOV-asetus (field of view, näkökenttä) muutetaan vähemmän laajakulmaiseksi perspektiivivääristymän poistamiseksi. Tilastovisualisoinneista on yleisesti todettu, että perspektiivivääristymä voi aiheuttaa virhearviointeja. Tämän lisäksi scene-tiedostossa tulee oletusarvoisesti mukana valonlähde, joten sen paikkaa muutettiin vastaamaan paremmin visualisoinnin tarvetta.

Tämän jälkeen lisättiin barsVisualizer GameObject -kappaleeseen uusi BarVisualizer-luokka. BarsVisualizer GameObject -kappaleeseen oli jo aiemmin lisätty DataParser-luokka. Unityssä on monia tapoja luoda uusia luokkia, tässä tapauksessa uusi luokka lisättiin Inspector-ikkunan kautta, barVisualizer-GameObject valittuna valittiin “Add Component”, “New Script” ja nimen antamisen jälkeen painettiin “Create and Add”.

Aivan ensimmäiseksi BarVisualizer-luokkaan lisättiin GameObject- ja Transform-tyyppiset muuttujat. Koska visualisointi päätettiin toteuttaa käyttämällä GameObject-kappaleita, lisättiin seuraavat kentät, joiden avulla Unityn Inspector-paneelista voidaan lisätä kyseiselle luokalle viittauksia sen tarvitsemiin scene-tiedostossa oleviin GameObject-kappaleisiin, tai niiden komponentteihin (kuva 9).

```
[Header("Geometry")]
public Transform cameraTra;
public Transform visualizerTra;
public GameObject cube;
public GridVisualizer grid;
```

Kuva 9. Muuttujia visualisoinnin grafiikan esittämistä varten

CameraTra-kenttään lisättiin Inspector-ikkunasta linkki kameraan. VisualizerTra-kenttään lisättiin linkki tyhjään GameObject-kappaleeseen, joka tulisi toimimaan parent-kappaleena

pylväsdiagrammin pylväille. Grid-muuttuja taasen on pylväiden alle piirrettävää koordinaattiruudukkoa varten. Tätä varten lisättiin Unityn GameObject-menun 3D-alimenusta scene-tiedostoon Plane-peruskappale, jolle annettiin nimeksi gridVisualizer. Sekä visualizerTra että gridVisualizer GameObject -kappaleiden varmistettiin olevan scene-tiedoston koordinaatiston 0, 0, 0 kohdassa. Tämän jälkeen ne kiinnitettiin VisualizationManager GameObject -kappaleen alle.

5.3 Pylväsdiagrammin toteutustapa

Seuraavaksi aloitettiin miettimään visualisoinnin rakennetta. Datan pohjalta päätettiin, että 3D-tilassa x-akselille laitettaisiin ikäryhmät nuorimmasta vanhimpaan ja z-akselille (syvyys) laitettaisiin eri vuodet. Pylväsdiagrammin pylväät kussakin x- ja z-akselin leikkauskohdassa kuvaavat siten tiettyä ikäryhmää tiettynä vuotena ja pylväsdiagrammin pylväiden korkeus, eli pituus y-akselilla kuvaa tämän väestöryhmän väkimäärää kyseisenä vuonna. Koska visualisointi on kolmiulotteinen, on mahdollista, että pylväät voivat jäädä toistensa taakse piiloon, esimerkiksi pylväiden koosta tai kameran kuvakulmasta johtuen. Tätä varten päätettiin jo suunnitellessa, että toteutetaan kontrollit, joilla käyttäjä voi valita sekä ikäryhmän alku- ja loppuarvon, että esitettävän visualisoinnin alku- ja loppuvuoden. Näin pylväsdiagrammin esitystä voi muokata sen mukaan mitä ikäryhmiä, aikaväliä tai näiden yhdistelmiä käyttäjä haluaa tarkastella.

5.4 Datan piirtäminen ruudulle

Koko visualisointi käynnistyy BarVisualizer-skriptin Unityn Start-metodista, jota Unity kutsuu automaattisesti ensimmäisessä päivityksessä (engl. frame), kun ohjelma ajetaan. Start-metodissa kutsutaan ensin DataParser-luokan Parse-metodia (kuva 10).

```
DataParser.Instance.Parse();
```

Kuva 10. Visualisointi aloitetaan kutsumalla Parse-metodia

DataParser-luokkaan lisättiin pelkistetty singleton-ratkaisu, jolla päästään staattisen muuttujan kautta DataParser-luokan ainoaan scene-tiedostossa olevaan ilmentymään käsiksi. Seuraavaksi kutsutaan Parse-metodia, joka lataa ja tallentaa CSV-väestödatan DataParser-luokan List-rakenteeseen. Tämän jälkeen kutsutaan InitVisualization-metodia. InitVisualization-metodissa asetetaan aiemmin mainitut ikäryhmä- ja vuosirajat (kuva 11).

```
public void InitVisualization ()
{
    // Time data
    var startTime = int.Parse(DataParser.Instance.keys[2]);
    var endTime = int.Parse(DataParser.Instance.keys[ DataParser.Instance.keys.Length-1]);

    // Age data
    var startAge = 0;
    var endAge = DataParser.Instance.ageGroups.Count-1;

    DoQuery(startTime, endTime, startAge, endAge);
}
```

Kuva 11. Arvojen asettaminen ja kyselyn tekeminen

Tämän jälkeen kutsutaan DoQuery-metodia, joka toteuttaa kaksi asiaa. Se tallentaa ikäryhmien ja vuosien alku- ja loppuarvot. Tämän jälkeen se kutsuu DataParser-luokan Query-metodia. Tämä metodi päädyttiin lisäämään, jotta näytettävän datan valitsemista voitaisiin helpottaa. Sen sijaan että vuosien rajaamista olisi toteutettu if-lauseilla tai switch-rakenteilla, etsittiin joustavampia vaihtoehtoja ja päädyttiin Linq-luokan kokeilemiseen tähän tarkoitukseen. Microsoftin .NET-ohjelmistokehyksen tarjoama System.Linq-luokka mahdollistaa listassa olevasta datasta hakujen tekemisen SQL-kielen hakujen kaltaisesti. Tässä tapauksessa käytetty data oli valmiiksi käyttötarkoitukseen sopivassa järjestyksessä, mutta tarvittaessa Linq mahdollistaa myös valittujen elementtien järjestämisen, kun tulos palautetaan. Linq ei ollut tuttu, joten siihen perehdyttiin ensin hieman. Alustavien kokeilujen jälkeen päädyttiin seuraavaan kyselyyn (kuva 12):

```
public IEnumerable<ItemData> Query (int yearStart, int yearEnd)
{
    IEnumerable<ItemData> query =
    from item in parsedData
    where (item.vuosi >= yearStart && item.vuosi <= yearEnd)
    orderby item.vuosi
    select item;

    return query;
}
```

Kuva 12. Query-metodi palauttaa kyselyn tuloksen parseroidusta datasta

Käytännössä tämä metodi ottaa vastaan alku- ja loppuvuoden sekä palauttaa parsedData List-rakenteesta kaikki solut, joiden alku- ja loppuvuosi ovat annettujen parametrien välissä. Tämän jälkeen data järjestetään vuoden mukaan ja palautetaan pyytäjälle IEnumerable-tyyppisenä kokoelmana. IEnumerable on käytännössä C#-ohjelmointikielen rajapinta (engl. interface), jonka tarjoaman kokoelman sisältöä voi lukea eri tavoin, kuten foreach-silmukalla. Tämän jälkeen palataan takaisin DoQuery-metodiin (kuva 13).

```
public void DoQuery (int startTime, int endTime, int startAge, int endAge)
{
    this.startTime = startTime;
    this.endTime = endTime;
    this.startAge = startAge;
    this.endAge = endAge;

    var data = DataParser.Instance.Query(startTime, endTime);

    VisualizeData(data);
}
```

Kuva 13. DoQuery-metodi tekee käyttäjän pyytämän kyselyn

Kun data on saatu data-nimiseen muuttujaan Query-metodilta, voidaan jatkaa VisualizeData-metodiin, joka vastaa pylväiden lisäämisestä scene-tiedostoon. VisualizeData-metodi muutti muotoaan useamman kerran, mutta tässä opinnäytetyössä käsitellään vain sen lopullista versiota.

Ensin VisualizeData-metodi poistaa kaikki pylväitä kuvaavat GameObject -kappaleet. Tämän jälkeen VisualizeData-metodi käy läpi yksi kerrallaan arvoja saamastaan IEnumerable-tyyppisestä data-muuttujasta, joka sisältää ItemData-luokan ilmentymiä. Käytännössä saadussa datassa on lähtökohtaisesti koko pyydetty vuosiväli, ikäryhmien rajaus tehdään vasta VisualizeData-metodissa. Tätä ei tehty DataParser-luokan Query-metodissa, koska ikäryhmät ovat datassa String-tyyppisiä kenttiä, eivätkä numeroarvoja.

Linq-luokka palauttaa yksiulotteisen listan, mutta visualisoinnissa halutaan esittää kaksiulotteinen kenttä pylväitä. Jotta pystytään etenemään z-akselilla, tarkastellaan ItemData-olioiden vuosia. Kun vuosiarvo vaihtuu, siirrytään seuraavaan z-koordinaattiin, nollataan x-koordinaatti ja lisätään z-koordinaatin arvoa (kuva 14).

```
if (item.vuosi != edellinen.vuosi)
{
    x = 0;
    z++;
}
```

Kuva 14. Yksiulotteisen taulukon rakenteen kiertäminen

Pylväät on toteutettu Cube-peruskappaleella, joka on yksi Unityn tarjoamista 3D-peruskappaleista. Cube-peruskappale ei sellaisenaan ole sopiva pylvääksi, koska sen pivot, eli

transformaatioiden keskipiste on kappaleen keskellä. Lisäksi pylvästä ei voi tehdä muuttamalla Cube-peruskappaleen mittoja, koska se ei ole parametrinen kappale, kuten eivät muutkaan Unityn tarjoamista peruskappaleista. Tätä varten Cube-peruskappaleen kopio kiinnitettiin tyhjään GameObject-kappaleeseen, jossa on ainoastaan Transform-komponentti. Tämä GameObject-kappale sijoitettiin Cube-peruskappaleen pohjan keskelle, jolloin tämän skaalaaminen mahdollistaa pylvään pituuden kasvattamisen. Lisäksi pylvään parent GameObject-kappaleeseen lisättiin DataContainer MonoBehaviour -skripti. Tämä on yksinkertainen luokka, joka mahdollistaa ItemData-luokan ilmentymän tallentamisen GameObject-kappaleeseen, jolloin luotaviin pylväisiin voidaan tallentaa pylvästä koskevat tiedot. Tässä ajatuksena on, että kun käyttäjä vie hiiren kursorin pylvään päälle, näkee hän nämä tiedot tooltip-paneelissa, joka ilmestyy hiiren kursorin viereen. Tätä varten tehtiin OnMouseOverHilite MonoBehaviour -luokka, joka lisättiin itse Cube-peruskappaleeseen. Tämä mahdollistaa kappaleen valinnan korostamisen, kun hiiren kursori viedään sen päälle. Jotta hiirellä valinta tulisi toimimaan, piti varmistaa vielä, että kappaleessa on jokin törmäystarkastuksen mahdollistava Collider-tyyppinen komponentti. Unityn Raycast- ja fysiikkajärjestelmä vaativat, että kappaleessa tulee olla Collider-komponentti, jotta se voidaan rekisteröidä fysiikkajärjestelmässä. Lopuksi pylvään Cube GameObject -kappaleeseen lisättiin bar-pintamateriaali ja pylväästä tehtiin Prefab-asset “bar”, joka tallennettiin projektin Prefabs-kansioon.

Tämän jälkeen seuraa silmukka, jossa tarkistetaan, onko kyseinen pylväs piirrettävä. Tämä rajaus tehdään käyttäjän asettaman ikäryhmän alku- ja loppuarvoilla. Jos nämä ehdot täyttyvät foreach-silmukan if-ehdon sisällä, luodaan dataa vastaava pylväsdiagrammin pylväs. Pylvään paikka määräytyy siis silmukan x- ja z-muuttujien mukaan ja pylvään korkeus määräytyy väkimäärän mukaan. Koska datan lukuarvot ovat kymmeniä tuhansia, pylvään korkeutta skaalattiin kertoimella, niin että ne mahtuvat ruudulle. Seuraavaksi pylvään GameObject-kappaleeseen lisättiin sen DataContainer-skriptiin ItemData. Aivan lopuksi pylväälle lasketaan väriarvo, jotta visualisointi ei näyttäisi yksiväriseltä. BarVisualizer-luokkaan lisättiin vielä List-rakenne, jossa pylväiden käyttämät väriliu’ut määritellään (kuva 15).

```
public List<Gradient> colorRamps;
```

Kuva 15. Pylväiden käyttämät väriliu’ut kuvataan Gradient-luokkaa käyttämällä

Tähän List-rakenteeseen on tallennettu valmiiksi suunniteltuja väriliukuja, joita Unityn editorissa voi luoda hieman samaan tapaan kuin vaikkapa Photoshopissa. Unity osaa piirtää Gradient-tyyppiset muuttujat Inspector-ikkunassa (kuva 16).

Kuva 16. Gradient-luokalla tehdyt väriliu'ut List-rakenteessa Inspector-ikkunassa

Pylväiden värittyminen toteutettiin niin, että väri määräytyy pylvään x-akselin paikan mukaan. Värisävy pylväälle saadaan Gradient-luokasta käyttämällä Evaluate-metodia, jolle annetaan arvoja 0-1 välillä. Näin ensimmäinen ikäryhmä saa Gradient-luokan tallentaman väriliu'un vasemman laidan arvon, kun taas x-akselin viimeisen jonon pylväät saavat väriliu'un oikean laidan arvon. Pylväiden väriarvot laskettiin seuraavalla tavalla (kuva 17):

```
float fraction = (float)x/(DataParser.Instance.keys.Length -2);
var color = colorRamps[colorSetNow].Evaluate(fraction);
bar.GetComponentInChildren<Renderer>().material.color = color;
```

Kuva 17. Väriarvon laskeminen ja asettaminen pylväille

5.5 Koordinaattiruudukko

Pylväsdiagrammin alla haluttiin esittää jonkinlainen koordinaattiruudukko. Koordinaattiruudukon haluttiin keskittyvän automaattisesti pylväiden alle. Tässä osoittautui ongelmaksi, että kun pylväät luodaan, eivät niiden GameObject-kappaleet ole ilmeisesti vielä Unityn samassa Update-metodin syklissä päivittyneet, mikä aiheutti ongelmia pylväiden keskipisteen laskemiselle ruudukkoa päivittäessä. Unity kuitenkin tarjoaa tällaisten tapausten varalta LateUpdate-metodin, jota se kutsuu kaikkien Update-metodikutsujen jälkeen. LateUpdate-metodia käyttämällä tästä tilanteesta selvittiin asettamalla pylväiden piirtämisen jälkeen VisualizeData-metodin lopussa totuusarvon tallentavan dirty-muuttujan arvoksi true. Kun Unityn sisäinen päivitys tulee BarVisualizer-luokan LateUpdate-metodiin, tiedetään dirty-muuttujan ollessa true, että koordinaattiruudukkoa tulee päivittää.

Koordinaattiruudukon tietoja päivittävä UpdateGrid-metodi kutsuu GetGeoCenter-apumetodia. GetGeoCenter-metodi laskee kaikkien pylväiden paikat yhteen (Vector3 arvot) ja jakaa tämän Vector3-muuttujan arvon pylväiden lukumäärällä (kuva 18). Näin saadaan tietää pylväiden keskipiste. Unity ei itse tarjoa tällaista tietoa suoraan, joskin se tarjoaa tietoja esimerkiksi yhden GameObject-kappaleen Renderer- tai Collider-komponenttien mitoista ja keskipisteistä.

```
public Vector3 GetGeoCenter()
{
    var center = Vector3.zero;
    var numPoints = 0;

    foreach (Transform child in visualizerTra)
    {
        if (child != transform)
        {
            center += child.transform.position;
            numPoints++;
        }
    }

    return center = center / numPoints;
}
```

Kuva 18. Ratkaisu kappaleiden yhteisen keskipisteen laskemiseen

Koordinaattiruudukolle on oma GridVisualizer-luokka, jonka kopio on kiinnitetty koordinaattiruudukko GameObject-kappaleeseen. Tämän SetPosition-metodia kutsutaan ja se asettaa koordinaattiruudukon pyydettyyn paikkaan, pylväiden alle, niiden keskipisteeseen. UpdateGrid-metodi asettaa myös koordinaattiruudukon x- ja y-akselien otsikot, joiden asettamiseen käytetään myös GridVisualizer-luokan SetXAxisLabel- ja SetYAxisLabel-metodeja.

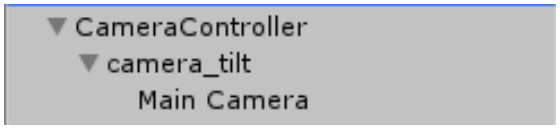
5.6 Kamera

Kun pylväät oli saatu ruudulle, pystyttiin sekä kameran paikkaa että valaistusta säätämään. Kamera siirrettiin sopivalle paikalle, niin että pylväät näkyvät hieman yläviistosta. Kameran linssin asetuksia säädettiin myös niin, että kuva on suhteellisen lähellä isometristä esitystä, niin että pylväiden keskinäiset suhteet eivät vääristy liiaksi perspektiivistä johtuen.

Visualisoinnin käyttäjän on myös pystyttävä tarkastelemaan ja katselemaan pylväsdiagrammia eri kuvakulmista, jota varten tarvitaan toiminnallisuutta toteuttava skripti kameran liikutteluun. Unity tarjoaa Asset Store -sisältökaupasta siirrettävissä olevissa demo-materiaalipaketeissa erilaisia kamera-kontrollereita. Lisäksi Unity tarjoaa myös ilmaiseksi Asset Store -sisältökaupasta siirrettävän Cinemachine-kameratyökalun. Todettiin, että tarve on kuitenkin hyvin tapauskohtainen ja tarkasti määriteltävissä, joten itse ohjelmoitu ratkaisu olisi sopivampi. Kameraa tulisi pystyä pyöritellä pylväsdiagrammin ympäri ja kamera saisi jäädä liikkumaan hetkeksi käyttäjän lopetettua hiiren napin painamisen, kunnes se hetken päästä pysähtyy pehmeästi. Tämän lisäksi kameraa tulisi pystyä nostamaan ja laskemaan

niin, että kamera osoittaa jatkuvasti kohti pylväiden keskikohtaa. Lisäksi hiiren rullaa pyörittämällä tulisi pystyä siirtämään kameraa lähemmäs tai kauemmas pylväsdiagrammista.

Näillä määrittelyillä ja rajoituksilla oli tarkoitus tehdä kameran käytöstä mahdollisimman helppoa, niin että käyttäjä voi keskittyä itse datan tarkasteluun ja että kameran kohdistamiseen tarvitsee käyttää mahdollisimman vähän huomiota. Kameran liikuttaminen mahdollistuu CameraController-luokan toiminnoilla. Tämän lisäksi kameran liikuttelun mahdollistamiseksi toivotulla tavalla luotiin seuraavanlainen GameObject-hierarkia (kuva 19):



Kuva 19. GameObject-hierarkia kameran toiminnallisuutta varten

CameraController-luokan Unity Update-metodissa jokaisella päivityskerralla ajetaan sekä RotateObject- että ZoomCamera-metodit. RotateObject-metodi vastaa käyttäjän hiiren liikkeellä hallittavasta kameran pyörittelystä pylväsdiagrammin ympäri (kuva 20). Käytännössä pyörittely voisi tapahtua pelkästä hiiren liikkeestä mutta todettiin, että käyttöliittymän ruutunappien käytön kannalta olisi parempi, että kamera ei liiku, kun käyttäjä hakeutuu nappien luo hiiren kursorilla. Siksi päätettiin testaamisen tuloksena asettaa kameran pyörittely tapahtumaan vain, kun hiiren oikea nappi on painettuna pohjaan. Kuten aiemmin todettiin, haluttiin että kamera ei pysähdy äkillisesti, vaan pehmeästi liukuen. Tällä on pääasiassa esteettinen merkitys - kuvion pyörittelystä tulee miellyttävämpää, kun kameran liike ei ole töksähtelevää. Toisaalta yksi tavoite hyvälle vuorovaikutteiselle visualisoinnille on se, että siirtymien ja muutoksien olisi suotavaa olla tasaisia. Tämä ominaisuus lisättiin kuitenkin vain sivuttaisliikkeeseen, jottei tästä efektistä tule liian häiritsevää.

```
void RotateObject()
{
    if (Input.GetMouseButtonDown(1))
    {
        previousPosition = Input.mousePosition;
    }
    else if (Input.GetMouseButton(1))
    {
        distX = -(previousPosition - Input.mousePosition).x * Time.deltaTime * rotationSpeed;
        distY = (previousPosition - Input.mousePosition).y * Time.deltaTime * rotationSpeed;

        rotateTra.Rotate(Vector3.up, distX);
        tiltTra.Rotate(Vector3.right, distY);

        previousPosition = Input.mousePosition;
    }
    else
    {
        // Free spin
        if (Mathf.Abs(distX) > 0f)
        {
            if (distX < 0)
                distX += friction * Time.deltaTime;
            else
                distX -= friction * Time.deltaTime;
        }

        rotateTra.Rotate(Vector3.up, distX);
    }
}
```

Kuva 20. Kameran pyörittelyn toteuttava koodi

ZoomCamera on hieman harhaanjohtava nimi metodille, koska käytännössä kameraa siirretään lähemmäs tai kauemmas pylväsdiagrammista, kyseessä on siis kamera-ajo (kuva 21). Kameran siirtäminen päätettiin toteuttaa niin, että hiiren rullan pyörittäminen ei suoraan siirrä kameraa, vaan asettaa kameralle uuden kohde-etäisyyden. Kamera käyttää tätä etäisyyttä apuna ja päättelee, tulisiko sen siirtyä vai ei. Jos etäisyyttä on tarpeeksi, kamera alkaa siirtyä kohti uutta kohde-etäisyyttä. Tämä siirtyminen on toteutettu käyttämällä Unityn Vector3-luokan Lerp-funktiota, joka mahdollistaa tasaisen siirtymisen kahden pisteen välillä. Näin myös kameraa siirretään lähemmäs tai kauemmas kohteesta tasaisesti kameran pyörittelyn tavoin. Kameran x-akselin ympäri tapahtuva liike tuottaa kameran kallistuksen pystysuunnassa. Jotta tämä olisi onnistunut vaivattomasti ilman ongelmia (mahdollinen gimbal lock tai äkilliset kääntymiset), käytettiin erillistä camera_tilt GameObject-kappaletta, joka on hierarkiassa kameran CameraController GameObject -kappaleen ja kameran välissä. Tämä mahdollistaa sen, että kamera voi liikkua itsenäisesti eteen- ja taaksepäin, samalla kun camera_tilt voi kontrolloida kameran korkeutta, niin että kamera osoittaa kohti CameraController GameObject -kappaletta. CameraController taasen voi pyöriä y-akselinsa ympäri, pylväsdiagrammin ympäri.

```
void ZoomCamera ()
{
    zoomDistNow = (dollyTra.position-rotateTra.position).magnitude;

    if (Input.GetAxis("Mouse ScrollWheel") != 0)
    {
        var zoom = Input.GetAxis("Mouse ScrollWheel") * 2f * zoomStep;

        zoomDistTarget = zoomDistTarget - zoom;
    }

    var newPos = dollyTra.position;
    zoomDelta = Mathf.Abs(zoomDistNow - zoomDistTarget);

    if (zoomDistTarget < zoomDistNow && zoomDelta > limit)
    {
        newPos += dollyTra.forward * zoomDistNow;
    }
    else if (zoomDistTarget > zoomDistNow && zoomDelta > limit)
    {
        newPos -= dollyTra.forward * zoomDistNow;
    }

    dollyTra.position = Vector3.Lerp(dollyTra.position, newPos, Time.deltaTime);
}
```

Kuva 21. Kameran etäisyyttä kohteesta muokkaava ZoomCamera-metodi

5.7 Tooltip- ja RayCastItems-luokat

Jotta käyttäjä voisi tarkkailla pylväsdiagrammin arvoja muutenkin kuin vertailemalla niiden korkeuksia, tarvitaan jokin tapa lukea pylväisiin tallennettuja ItemData-luokan ilmentymien sisältöjä. Tätä varten kirjoitettiin Tooltip- ja RayCastItems-luokat. Tooltip ja RayCastItems MonoBehaviour -luokkien kopiot ovat kiinnitettynä UI-panel-elementtiin. RayCastItems-luokka toimii seuraavanlaisesti: jos käyttäjä ei pidä hiiren oikeaa nappia pohjassa, eli ei pyöritä näkymää, niin tällöin tehdään kutsu DoRaycastCheckForData-metodiin. DoRaycastCheckForData-metodi tekee hiiren kursorin kohdalta ruudun pinnasta Unityn Physics-luokan Raycast-metodia käyttäen säteen ampumalla tarkastuksen, osuuko se johonkin kappaleeseen, jossa on Collider-komponentti. Jos Raycast osuu Collider-komponenttiin, tarkastetaan tässä tapauksessa, onko kyseisessä GameObject-kappaleessa DataContainer-luokka, joka on lisätty vain pylväisiin. Jos näin on, tiedetään että kyseessä on pylväs, jossa on myös näytettävää ItemData-tyyppistä dataa, DataContainer MonoBehaviour -luokan muuttujan sisällä. RayCastItems-luokka kutsuu Tooltip-paneelin näyttämistä ja muuttaa sen tekstejä, käyttämällä Tooltip-luokan SetText-metodeja. Näin voidaan siis hiiren kursori pylvään päälle viedessä näyttää kyseisen pylvään tietoja.

Tooltip-paneeli asetti omia haasteita. Unity ei tarjoa valmiina mitään vastaavaa toiminnallisuutta. Tooltip-toiminnallisuutta varten rakennettiin UI-elementeistä paneeli, johon lisättiin Text-elementtejä. RayCastItems-luokka kytkee tarvittaessa tämän paneelin näkyviin tai

piilottaa sen. Myös paneelin ruudulla näkyvissä pysyminen piti ratkaista. Kun kursori on tarpeeksi lähellä ruudun ylälaitaa tai oikeaa laitaa, niin hiiren yläoikealla näytetty paneeli rajautuu jossain vaiheessa pois kuvaruudusta. Tämä ratkaistiin seuraavasti: Unityn UI-luokkiin perehtymällä selvisi, että RectTransformUtility-luokkaa hyödyntämällä voidaan muuntaa hiiren kursorin paikka UI-järjestelmän käyttämän Canvas-elementin koordinaatistoon. Käytännössä kaikki UI-elementit ovat Canvas-elementin hierarkiassa. UI-järjestelmän käyttämä ruudun peittävä Canvas-elementti ei siis käytä samoja koordinaatteja kuin Screen-luokka. Näin voitiin selvittää hiiren kursorin paikka UI-järjestelmän kankaalla, jolloin pystyttiin myös laskemaan Tooltip-paneelin mahtuminen ruudulle. Tämän lisäksi tuli myös huomioida ruudun todellinen pikselikoko, joka selvisi Canvas Scaler -komponentin tarjoamasta tiedosta. Alla Tooltip-paneelin ruudulla näkyvissä pitävä koodi (kuva 22):

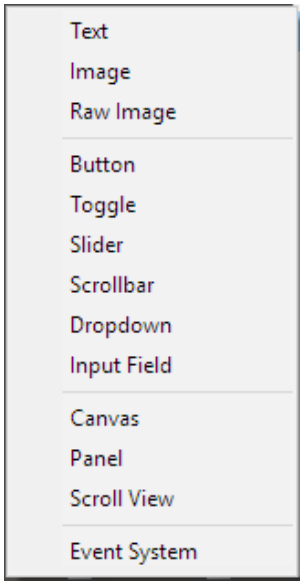
```
void LateUpdate ()
{
    if (isVisible)
    {
        Vector2 mousePos = Vector3.zero;
        RectTransformUtility.ScreenPointToLocalPointInRectangle(canvasRect,
        Input.mousePosition, null, out mousePos);

        var halfWidth = scaler.referenceResolution.x * 0.5f;
        var halfHeight = scaler.referenceResolution.y * 0.5f;

        testPos  = mousePos;

        var offsetFinal = offset;

        if (mousePos.x + tooltipRect.rect.width >= halfWidth)
        {
            offsetFinal.x = -tooltipRect.rect.width - offset.x;
        }

        if (mousePos.y + tooltipRect.rect.height >= halfHeight)
        {
            offsetFinal.y = -tooltipRect.rect.height - offset.y;
        }

        tooltipRect.localPosition = mousePos + offsetFinal;
    }
}
```

Kuva 22. Tooltip-paneelin ruudulla pysymisen mahdollistava koodi

5.8 Graafinen käyttöliittymä

Jotta käyttäjä voisi muokata pylväsdiagrammivisualisoinnin asetuksia, tarvitaan käyttöliittymä. Jo visualisointia suunnitellessa päätettiin, että toteutetaan graafinen käyttöliittymä (graphical user interface, GUI). Unity on aiemmin tarjonnut hieman vanhanaikaisen graafisten käyttöliittymien toteutustavan. Tätä aiempaa ns. Immediate Mode GUI (IMGUI) käyttöliittymien ohjelmointitapaa suositellaan käytettäväksi enää vain Unityn editorilaajennusten ohjelmointiin. Vaikka aiempi Unityn käyttöliittymien toteutustapa on ollut rajoittunut, ei uudempi käyttöliittymien toteutustapa sekään ole täydellinen. Käyttöliittymiä rakennetaan

GameObject-kappaleista, joihin on kiinnitetty RectTransform-komponentti, joka korvaa Unityn GameObject-kappaleiden ainoan pakollisen komponentin, eli Transform-komponentin. Kaikkien UI-järjestelmän osana olevien GameObject-kappaleiden tulee sijaita hierarkiassa Canvas-komponentilla varustetun GameObject-kappaleen alla, ja niissä tulee olla RectTransform-komponentti. RectTransform-komponentti mahdollistaa UI-järjestelmän kankaalle (Canvas) lisätyille kappaleille kaksiulotteiset venyttely- ja sijoitteluominaisuudet Inspector-paneelissa. Tämän lisäksi Unity UI-järjestelmä tarjoaa erilaisia komponentteja, joita voi lisätä näihin kappaleisiin (kuva 23). Näitä komponentteja on suhteellisen pieni määrä, käytännössä vain olennaisimmat.



Kuva 23. Unityn tarjoamat UI-komponentit

Tämän lisäksi UI-järjestelmä tarjoaa joitakin luokkia sommittelun tekoon, ja näidenkin valikoima on hyvin rajoittunut. Kaikki UI-komponentit teksti mukaan lukien koostuvat kahdesta tai useamman kolmion muodostamista nelikulmioista, joten suuri määrä elementtejä aiheuttaa myös päällekkäispiirtoa ja lisää näytönohjaimella piirrettävien 3D-elementtien määrää.

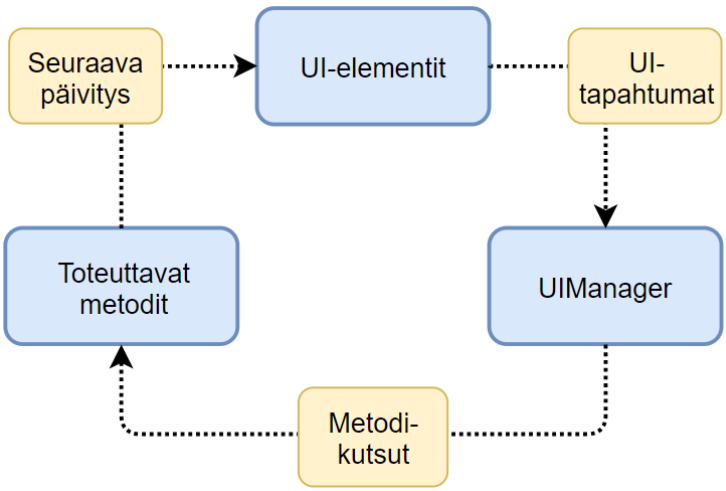
Vaatimusmäärittelyvaiheessa listattiin asioita, joita käyttäjän on kyettävä tekemään graafisella käyttöliittymällä:

- Ohjeiden näyttäminen tulisi onnistua info-napista paneeli avaamalla.
- Pylväiden arvojen tarkastelun tulisi onnistua Tooltip-luokkaa ja tooltip-paneelia käyttämällä.
- Näytettävien arvojen valinta tulisi onnistua arvojen valintapaneelia käyttämällä.
- Visualisoinnin väripaletin vaihtaminen tulisi olla mahdollista.
- Asetusten muuttaminen (koordinaattiruudukon ja akselien näyttäminen) ja ohjelmasta poistuminen tulisi olla mahdollista asetuspaneelista.
- Visualisoinnin asetusten nollaaminen tulisi onnistua reset-nappulasta.

Tämän pohjalta päädyttiin seuraavanlaiseen sijoitteluun (kuva 24):

Kuva 24. Graafisen käyttöliittymän sijoittelu

Käytännössä graafisen käyttöliittymän toteuttaminen tapahtuu Unityssä pelkistetyimmillään rakentamalla UI-elementeistä paneeleita ja erilaisia interaktiivisia elementtejä. Tämän jälkeen on tehtävä UI-elementtien kytkeminen ohjelmalogiikkaan, joka onnistuu usealla tavalla. Tätä toimintaa varten luotiin UIManager MonoBehaviour -luokka, joka lisättiin tyhjään GameObject-kappaleeseen Canvas-elementin alle. Tähän luokkaan kerättiin kaikki UI-elementtien tarvitsemat metodit. Visualisoinnin graafisen käyttöliittymän toimintaa on kuvattu kuviossa 16:



Kuvio 16. Graafisen käyttöliittymän rakenne ja toimintaperiaate

Jokaiselle Button-elementille lisättiin Inspector-ikkunassa sen OnClick-osioon metodikutsu, joka osoittaa UIManager-luokassa sille varattuun metodiin. Esimerkiksi quit-napin kutsuma metodi näyttää seuraavanlaiselta UIManager-luokassa (kuva 25):

```
public void OnQuitButtonPressed ()
{
    UserUIAudio.Instance.PlayButtonClickedClip();
    Debug.Log("OnQuitButtonPressed");
    Application.Quit();
}
```

Kuva 25. Esimerkki UI-kutsuun vastaavasta metodista

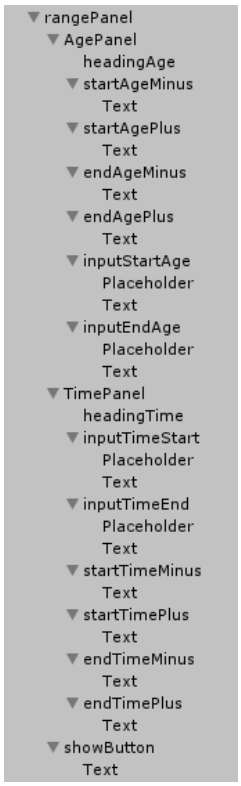
UIManager-luokan metodit välittävät Button-elementtien kutsut eteenpäin niiden toimintojen toteuttamisesta vastaaville metodeille eri luokissa. Lisäksi UIManager-luokka vastaa myös klikkauksille lisättyjen äänien soittamisesta. Yksi isoimmista Inspector-ikkunan kautta tehtyjen UI-metodikutsujen ongelmista liittyy Unityn editorin serialisointiin ja kutsuttavien luokkien tai metodien uudelleennimeämiseen. Jos kutsuttava metodi tai sen luokka nimetään uudestaan tai poistetaan, häviää tämän yhteys välittömästi Button-tyyppisiin ja muihin UI-elementteihin asetetuista OnClick:in kaltaisista kutsuista. Tämä voi aiheuttaa yllättäviä UI-elementtien toiminnan loppumisia jotka eivät anna virhettä ohjelma ajettaessa, joten näiden paikallistaminen voi olla työlästä. Jos nimeämisen aiheuttamia haittoja halutaan välttää, voidaan tätä kiertää käyttämällä nimetyissä muuttujissa FormerlySerializedAs-attribuuttia.

Käyttöliittymän ikoneina käytettiin CC 3.0 lisenssillä jaettavia “Modern UI Icons” projektin ikoneita, jotka ovat saatavilla osoitteesta: https://github.com/Templarian/WindowsIcons/. Tämä kokoelma on Windows-laitteille suunnattu kirjasto yleisiä ikoneita, ja ne sopivat hyvin tämän prototyypin käyttöön.

Esitettävän datan valinnan mahdollistavan UI-paneelin rakentaminen osoittautui melko työlääksi. Paneelien ja elementtien sijoittelua varten ei ole varsinaista editoria, jossa niitä voisi asetella peräkkäin, vaan ne tulee venytellä ja asemoida useimmiten yksi kerrallaan Unityn editorin 3D-tilassa. Vaikka tarjolla on myös 2D-näkymä, ei elementtien sijoittelu ole kovin käytännöllistä. Layout-elementeillä sijoittelua voi automatisoida, mutta ne soveltuvat pääasiassa useita samantyyppisiä UI-elementtejä sisältävien paneelien ja näkymien toteuttamiseen. Toteutettu paneeli näyttää seuraavanlaiselta (kuva 26):

Kuva 26. Esitettävän datan valinnan mahdollistava paneeli

Kun UI-elementeistä tehdään erilaisia rakenteita, tulee harkita jokaisen elementin sijaintia Unityn scene-tiedoston Hierarchy-ikkunassa. UI-elementtien järjestys Hierarchy-ikkunassa määrittelee myös niiden piirtojärjestyksen (ylhäältä alas -piirtojärjestys). Samoin jokaisen UI-elementin asetukset tulee laittaa kohdilleen Inspector-ikkunasta, kuten myös UI-elementtien toiminnallisuudesta vastaavat metodikutsut tulee määritellä Inspector-ikkunasta. Nämä kaikki asetukset voidaan toki tehdä koodilla, mutta käytännössä näin pienen graafisen käyttöliittymän toteuttamisen kannalta ei tällaisella ratkaisulla olisi saavutettu ajallista tai muutakaan etua. Esitettävän datan valintapaneelin hierarkia näyttää seuraavanlaiselta Hierarchy-ikkunassa (kuva 27):



Kuva 27. Esitettävän datan valinnan mahdollistavan paneelin rakenne

5.9 Äänet

Osa graafista käyttöliittymää on myös äänivaste, joka saadaan nappeja painettaessa. Äänien soittaminen toteutettiin UserUIAudio-luokassa, joka on lisätty samaan GameObject-kappaleeseen UIManager-luokan kanssa. Tämä koostuu kolmesta eri metodista, joita kutsumalla voidaan soittaa määriteltyjä ääniä Unityn AudioSource MonoBehaviour -luokkaa käyttämällä. Nämä äänet ovat menun avaus- ja sulkuäänet, napin painallusääni ja yleinen klikkausääni. Äänien soittamista varten lisättiin scene-tiedostoon yksi tyhjä GameObject-kappale, johon lisättiin AudioSource-komponentti, joka myös lisättiin Inspector-ikkunasta UserUIAudio-luokan kenttään. UserUIAudio-luokan metodeja kutsuttiin UIManager-luokasta seuraavalla tavalla (kuva 28):

```
public void OnEndMinusPressed ()
{
    if (endYearNow > startYearNow)
    {
        UserUIAudio.Instance.PlayDatapointClickedSound();
        endYearNow--;
        inputTimeEnd.text = endYearNow.ToString();
    }
}
```

Kuva 28. Äänien soittaminen kutsumalla UserUIAudio-luokan metodia

Äänien käyttäminen Unityssä on suhteellisen suoraviivaista. Jokaisen scene-tiedoston kamerassa on valmiina Audio Listener -komponentti, joka ns. kuuntelee ääniä, joita scene-tiedoston aktiiviset AudioSource MonoBehaviour -luokan omaavat GameObject-kappaleet voivat tuottaa. AudioSource-luokan avulla voi soittaa jatkuvaa ääntä, kuten musiikkia tai kertaluonteisia ääniä, kuten klikkaukset tai muut ääniefektit. Tämän lisäksi Unity tarjoaa Audio Mixer -assetin, jonka läpi on mahdollista kuljettaa ja samanaikaisesti muokata eri ääniefekteillä osaa tai kaikkia scene-tiedoston äänilähteiden äänistä. Rajallisten äänivaatimusten takia emme kuitenkaan tarvinneet Audio Mixer -assettia.

6 Parvikuviovisualisointi

Toisen 3D-visualisointiprototyypin aiheeksi valittiin parvi- eli hajontakuvio, jonka toteutusta käsitellään tässä luvussa. Parvikuvio on yksi yleisimmistä tilastokuvioista, ja siten sopii hyvin aiheeksi datan visualisointiin. Visualisoinnin scene-tiedoston rakenne selviää liitteestä 2.

Parvikuviovisualisointia varten valittu iris-datasetti on Brittiläisen tilastotieteilijä ja biologi Ronald Fisherin vuonna 1936 esittelemä datasetti, joka pohjautui Edgar Andersonin työhön. Kyseinen datasetti koostuu viidestäkymmenestä mittaustuloksesta kustakin kolmesta eri Iiris-kukkalajista (Setosa, Versicolour ja Virginica). Jokaisesta kukasta mitattiin verho- (sepal) ja terälehtien (petal) pituus sekä leveys senttimetreinä. Iris-datasetti on suhteellisen kevyt, siinä on vain 150 riviä ja kussakin tietueessa on viisi arvoa (pituudet, leveydet ja laji). Tätä datasettiä käyttämällä on myös helppo vertailla visualisoinnin toiminnan mahdollisia virheitä, koska iris-datasetistä on saatavilla sekä tietoa että visualisointeja lukuisilta verkkosivustoilta. Lisäksi iris-datasetti on saatavilla useammastakin lähteestä CSV-muodossa, käytimme Curran Kelleherin GitHub-sivulta löytyvää CSV-tiedostoa: https://gist.github.com/curran/a08a1080b88344b0c8a7. Alla ote iris-datasetistä (kuva 29):

```
sepal_length,sepal_width,petal_length,petal_width,species
5.1,3.5,1.4,0.2,setosa
4.9,3.0,1.4,0.2,setosa
4.7,3.2,1.3,0.2,setosa
```

Kuva 29. Iris-datasetin CSV-datan rakenne

6.1 Datan lataaminen ja parserointi

Iris-datasetti on rakenteeltaan erilaista verrattuna pylväsdiagrammivisualisoinnin väestötietodataan. Iris-datasetissä kukin rivi kuvaa yhtä mittaustulosta. Tästä johtuen tämän visualisoinnin tapauksessa päätettiin tehdä hieman yleiskäyttöisempi DataParser-niminen datan lataaja. Päädyttiin ratkaisuun, jossa ensin kerätään tiedoston otsikkoriviltä sarake-tyypit. Tämän jälkeen käsitellään kukin datarivi ja tallennetaan jokaisen sarakkeen tiedot. Tuetuiksi datatyypeiksi määriteltiin float- ja String-tyyppinen data. Tämä mahdollistaisi käytännössä minkä tahansa vastaavaa dataa sisältävän datasetin lataamisen visualisointiohjelmaan. DataParser-luokan muuttujat ovat hyvin samankaltaiset kuin pylväsdiagrammivisualisoinnnissa. Luettava data saadaan TextAsset-tyyppisestä muuttujasta tekstitiedostona. SeparatorChar-muuttuja kertoo käytettävän erotinmerkin, koska kyseessä on CSV-data, tässä tapauksessa käytetään pilkkua. Alla kuva DataParser-luokan muuttujista (kuva 30):

```
[Header("Data file")]
public TextAsset dataFile;
public char separatorChar = ',';
public string newLine = @"\r\n";

[Header("Read data")]
public List<string> labels;
public List<Col> data;
```

Kuva 30. CSV-datan parserointiin käytetyt muuttujat

Data tallennetaan List-tyyppiseen data -muuttujaan, Col-luokan ilmentymiin. Koska tiedot voivat olla joko float- tai String-tyyppistä dataa, toteutettiin Col-luokkaan dataString- ja dataFloat-nimiset List-tyyppiset rakenteet. Tämän lisäksi päätettiin tallentaa sarakkeen tiedot label- ja type-kenttiin. Näin pystytään selvittämään sarakkeen nimi ja sen tallentama datatyyppi. Ohjelman toteuttamisen aikana lisättiin Col-luokkaan myös muutama apumetodi, joilla voi saada sarakkeen nimen ja pituuden selville. Col-luokan datarakenne ei välttämättä ole paras, mutta osasyy tähän ratkaisuun oli se, että ladattua dataa pystyttäisiin näkemään myös Unityn Inspector-ikkunassa. Jos olisi käytetty esimerkiksi C#-ohjelmointikielen geneerisiä luokkia, näiden piirtäminen Unityn Inspector-ikkunassa olisi vaatinut Custom Inspector -luokan tai PropertyDrawer-luokan käyttöä.

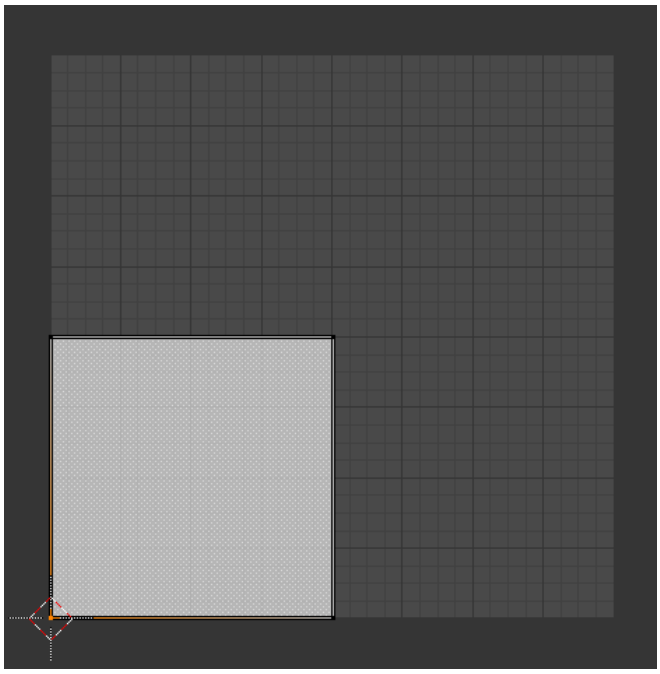
Pylväsdiagrammivisualisoinnin tapaan DataParser-luokka vastaa datan lataamisesta. Ensimmäisenä DataParser-luokka pilkkoo tiedoston kaikki rivit erillisiksi olioiksi käyttämällä tässäkin tapauksessa Regex-luokan Split-funktiota. Seuraavaksi ensimmäiseltä riviltä kerätään sarakkeiden tiedot List-tyyppiseen labels-muuttujaan. Kaikkiin rivien pilkkomisoperaatioihin käytetään itse kirjoitettua SplitRow-metodia, joka pilkkoo rivin käyttämällä String-luokan Split-metodia ja palauttaa sen List<String>-tyyppisenä listana takaisin. Tämän jälkeen varmistetaan, että data-niminen List-rakenne on tyhjä ja aloitetaan käsittelemään ja tallentamaan kukin tiedoston sarake. Saraketta varten luodaan tyhjä Col-olio. Tämän jälkeen selvitetään, onko kyseinen sarake float- vai String-tyyppinen. Koska kaikki data on tässä vaiheessa vielä String-tyyppistä, käytetään tähän float.TryParse-funktiota. Jos kyseinen data saadaan muunnettua float-tyyppiseksi luvuksi, kyseessä on float-tyyppistä dataa, muutoin kyseessä on String-tyyppinen data. Koko sarake käsitellään viimeiseen riviin asti, ja tallennetaan sarakkeena List-tyyppiseen data-nimiseen muuttujaan. Parseri etenee samaan tapaan kaikkien sarakkeiden läpi. Toteutettu parseri on jo aavistuksen monikäyttöisempi, se voi käytännössä ladata vapaamuotoisen määrän sarakkeita float- tai String-tyyppistä dataa. Mutta koska opinnäytetyössä keskityttiin pääasiassa visualisointiin eikä datan lataamiseen, ei tästä luokasta yritetty tehdä kovin virheenkestävää. Käytännössä virheellinen data ja puuttuvat arvot jätettiin huomioimatta. Tällaiseen parserointiin tuntuisi

järkevämmältä harkita käytettävän jotakin esimerkiksi Python-kielelle saatavilla olevan Pandas-kirjaston kaltaista datan käsittelyyn tarkoitettua kirjastoa, sen sijaan että datan lataaja kirjoitettaisiin itse.

6.2 Koordinaattiruudukko

Parvikuviovisualisoinnin koordinaattiruudukosta päätettiin tehdä joustavampi kuin pylväsdiagrammivisualisoinnin ruudukosta. Koordinaattiruudukon tasot päätettiin toteuttaa 3D-malleina. Koordinaattiruudukon haluttiin skaalautuvan tarpeen mukaan. Lisäksi koordinaattiruudukon esityksen haluttiin olevan hallittavissa niin, että sen ulkoasua voitaisiin säätää helposti esimerkiksi ruudukon tiheyden, viivojen paksuuden, värien ja muiden parametrien osalta. Näin koordinaattiruudukosta saataisiin monikäyttöisempi, ja sitä voitaisiin käyttää vaihtelevien datasettien esittämiseen.

Näistä vaatimuksista johtuen päätettiin yrittää toteuttaa koordinaattiruudukko tekstuurin sijaan dynaamisesti shaderilla renderöimällä. GridShaderUV-shaderissa koordinaattiruudukon renderöinti toteutettiin verteksi/fragmentti shaderia käyttämällä, niin ettei Unityn surface-ohjelma ollut käytössä. Näin pystyttiin pelkistämään shaderin toimintaa ja myös saamaan esitys ulkoasultaan halutunlaiseksi. Vaikka koordinaattiruudukon piirtäminen onnistuikin shaderilla, hieman asiaa tutkittua koordinaattiruudukon skaalautumisen toteuttaminen osoittautui kuitenkin puhtaasti shaderilla toteutettuna käytettävään aikaan nähden monimutkaiseksi. Koordinaattiruudukon tiilitystä (tiheyttä) pystyy toki muuttamaan helposti shaderiin lisätyllä Tiling-parametrillä, mutta tämä ei mahdollista nollakohdan määrittelyä. Jos tiilitystä muutetaan, niin koordinaattiruudukko kuitenkin alkaa kunkin tason vasemmasta laidasta. Tästä johtuen, yksinkertaisena ratkaisuna tasot päätettiin jakaa niin, että kukin taso koostuu neljästä neliöstä. UV-koordinaatit sijoitettiin seuraavalla tavalla Blender-ohjelmassa: UV-koordinaatiston vasemmassa alalaidassa 0, 0 kohdassa sijaitseva verteksi on kunkin koordinaattitason neljän monikulmion jaettu verteksi (kuva 31).

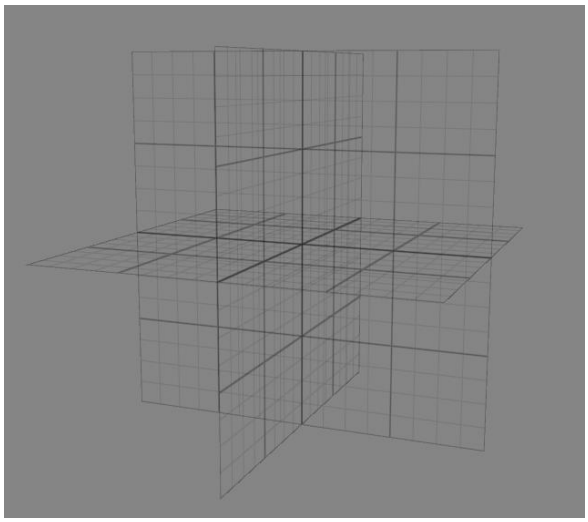


Kuva 31. UV-koordinaattien sijoittelu Blender-ohjelmassa

Tämä mahdollisti sen, että kun shaderin Tiling-parametriä muutetaan, koordinaattiruudukko alkaa odotetusti kunkin koordinaattitason keskipisteestä, tiilityksen määrästä riippumatta. Haittapuolena tässä ratkaisussa on se, että koska koordinaattiruudukko on toteutettu neljänneksistä, ei numeroarvoja voisi tällä koordinaattiruudukolla esittää. Suunniteltuun käyttötarkoitukseen shaderin ja mallin yhdistelmä kuitenkin riitti.

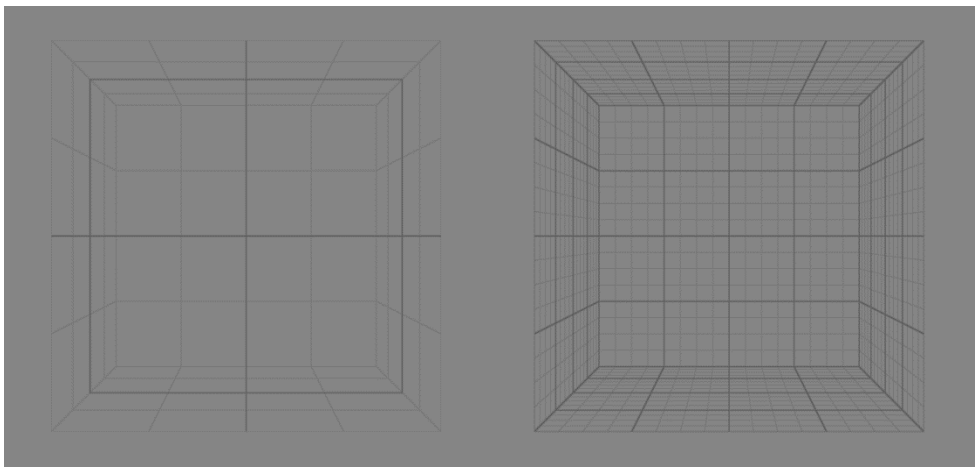
Koordinaattiruudukon esitystavaksi valittiin testauksen jälkeen kuutio, jossa todettiin olevan hyviä puolia. Se ei peitä dataa ja se mahdollistaa dataa sisältävän kuution sisällä liikkumisen, niin ettei ruudukko ole häiritsevästi tiellä. Toisaalta myös koordinaattiruudukkoa kuution seinillä voi käyttää viitteellisenä mitta-asteikkona. Lisäksi yksipuoleisena shaderina toteutettuna kuution sivut ovat automaattisesti poissa tieltä, kun kuutiota katsotaan sen ulkopuolelta. Näin on mahdollista tarkkailla dataa niin, ettei koordinaattiruudukko häiritse havainnointia.

Koordinaattiruudukon esitystapana testattiin myös kolmea ristikkäistä neliötä. Tämä vaikutti alkuun hyvältä ajatukselta, mutta se osoittautui ongelmalliseksi. Ristikkäiset pinnat näyttävät sekavilta, kun ne leikkaavat toisiaan. Koordinaattiruudukon osat voivat myös törmätä esitettyyn dataan. Välistä myös origon toisella puolella olevat kappaleet jäävät ruudukon peittämäksi. Lisäksi tällainen koordinaattiruudukko vaatii myös kaksipuoleisen shaderin, jotta ruudukko näkyisi joka suunnasta katsottuna. Kuvassa 32 nähdään testattu esitystapa, jota ei otettu käyttöön.



Kuva 32. Kolme ristikkäistä neliötä ei sopinut hyvin käyttötarkoitukseen

kuvassa 33 nähdään kuution muodossa esitetty koordinaattiruudukko asetuksilla 2 ja 4, origo pysyy koordinaattiruudukon koosta riippumatta keskellä jokaista kuution sivua.



Kuva 33. Koordinaattiruudukko eri arvoilla piirrettynä

Unity ei siis itsessään sisällä minkäänlaista renderöitävää ruudukkoa, pois luettuna editorin ruudukko, jota ei voi kuitenkaan renderöidä Game-ikkunassa. Tämä tarkoittaa siis sitä, että jos visualisoinnissa tarvitaan koordinaattiruudukkoa tai jonkinlaista mitta-asteikkoa, täytyy tällaiseen käyttöön ohjelmoida shaderi tai käyttää jotakin muuta toteutustapaa, mikä voi osoittautua kohtuullisen haastavaksi.

6.3 Koordinaattiakselit

Visualisointiin haluttiin myös 3D-mallinnusohjelmissa usein käytetty esitystapa koordinaattiakseleista. Yleinen tapa esittää koordinaattiakselit ja niiden suunta kuvaan nähden on

piirtää origossa sijaitsevasta pisteestä lähteviä viivoja. Punaisella (x), vihreällä (y) ja sinisellä (z) viivalla voidaan ilmaista kunkin koordinaattiakselin sen hetkistä suuntaa kuvassa. Akselien piirtämisen olisi voinut toteuttaa 3D-mallilla, mutta silloin 3D-kappaleen skaalaaminen olisi aiheuttanut sen, että akselien viivat piirtyvät paksumpina. Sama olisi myös tapahtunut kameran etäisyydestä riippuen, jolloin akselien viivat olisivat piirtyneet eri paksuisina, mitä ei haluttu. 3D-mallien käyttämistä haluttiin myös rajoittaa, niin että valtaosa sisällöstä ei olisi riippuvaista Unityn ulkopuolisista työkaluista tai 3D-mallien tiedostoista, joten tästä syystä päädyttiin toteuttamaan akselien esitys puhtaasti Unityssä.

Unityssä on tällä hetkellä muutamia tapoja, joilla viivoja voidaan piirtää. Yksi tapa on käyttää Line Renderer -komponenttia. Line Renderer -komponentti on kuitenkin suunnattu pääasiassa jatkuvien viivojen piirtämiseen, joten jos halutaan piirtää kolme irtonaista viivaa (x-, y-, z-viivat), olisi tähän tarvittu kolme erillistä Line Renderer -komponenttia. Tämän visualisoinnin suorituskyvyn kannalta tällä seikalla ei olisi ollut mitään merkitystä, mutta turhaa monimutkaisuutta haluttiin välttää.

Toinen toteutustapa olisi ollut se, että Mesh-luokkaa käyttämällä luodaan kokonaan itse esitettävä geometria. Unityn Mesh-luokkaa käytettäessä viivat olisi pitänyt esittää neljästä verteksistä ja kahdesta kolmiosta muodostuneesta kappaleesta. Lisäksi jos viivat halutaan esittää laadukkaasti, pitäisi ne luoda niin, että viivaa esittävän suorakaiteen pinta osoittaa kohtisuoraan kameraa päin joka kerta, kun ruutu päivittyy. Lisäksi kuvan perspektiivi vaikuttaisi viivan paksuuteen, joten tämäkin pitäisi huomioida, jos viivasta haluttaisiin samanlevyistä kuvaruudun pinnalla. Tämä olisi vaatinut jonkin verran pohtimista ja lukuisia laskutoimituksia, joten toteutus olisi voinut osoittautua tarpeettoman monimutkaiseksi.

Kolmas ja tämän visualisoinnin kannalta helpoimmalta tuntuva ratkaisu jota pohdittiin, oli matalan tason GL-luokan käyttö, joka mahdollistaa muutamien yksinkertaisten graafisten elementtien piirtämisen halutulla materiaalilla. GL.lines-toimintoa käyttämällä piirrettävät viivat ovat saman paksuisia ruudulla niiden etäisyydestä kameraan riippumatta ja ne piirtyvät ilman aliasoitumis-artifakteja (kuvan sahalaitaisuus) tai muitakaan ongelmia, joten tämä vaihtoehto koettiin sopivimmaksi tähän tarkoitukseen.

Koordinaattiakselien piirtoa varten luotiin DrawAxes-luokka, joka pitää kiinnittää komponenttina kameraan, jotta viivat piirtyvät ohjelmaa ajettaessa. Piirtoa toteuttavaa DrawLines-metodia kutsutaan Unityn kameran OnPostRender-metodissa. Tätä GL.lines-toteutustapaa käyttämällä oli myös mahdollista toteuttaa viivanpiirto niin, että viiva piirretään sekä ohjelmaa ajettaessa, että tarvittaessa myös Unityn editori-ikkunassa. Editorissa tapahtuva akselien piirto tapahtuu kutsumalla DrawLines-metodia Unityn OnDrawGizmos-

metodissa, joka on tarkoitettu editorissa näkyvien debug-grafiikoiden piirtämiseen. Näin koordinaattiakselien piirtämisestä saatiin monikäyttöisempi. Koordinaattiakseleihin käytettiin shaderia, joka mahdollistaa akselien piirtämisen kaikkien kappaleiden päälle, niin etteivät ne jää esimerkiksi dataelementtien taakse.

Koordinaattiakseleille on lisäksi myös sijoitettu tekstit, jotka esittävät datan suurimman arvon kullakin akselilla. Näiden tekstien toteuttamiseen vaadittiin Unityn UI-järjestelmän Text-elementit, jotka seuraavat akselien paikkoja. Pylväsdiagrammivisualisoinnin tapaan GridVisual-luokka vastaa akselien tekstien päivittämisestä. Tässä visualisoinnissa haluttiin, että nämä tekstit ovat selkeästi luettavissa. Tekstien haluttiin olevan ruudulla samankokoisia, paikasta riippumatta sekä osoittavan koko ajan kohti kuvapintaa. Tähänkään ei ollut valmista ratkaisua Unityn tarjoamana, joten päädyttiin käyttämään UI-luokkiin kuuluvaa RectTransformUtility-luokan ScreenPointToLocalPointInRectangle-metodia 3D-tilassa sijaitsevien akselien otsikoiden paikkaa kuvaavien pisteiden paikkojen selvittämiseen Canvas-elementin RectTransform-komponentin koordinaatistossa. Lopullinen ratkaisu tällaisten elementtien esittämiseen on seuraavanlainen (kuva 34):

```
Vector2 sp = Camera.main.WorldToScreenPoint(axisX.transform.position);
RectTransformUtility.ScreenPointToLocalPointInRectangle(canvasRect, sp, null, out localPoint);
textX.rectTransform.localPosition = localPoint;
```

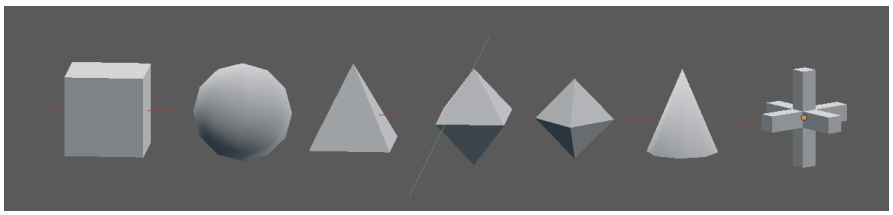
Kuva 34. 3D-kappaleiden paikan siirtäminen Canvas-koordinaatistoon

6.4 Datan rajojen visualisointi

Myös datasetin rajat haluttiin visualisoida, jotta katselija voi saada helpommin käsityksen siitä, minne asti esitetty data ulottuu 3D-tilassa. Tällainen ominaisuus mahdollistaa myös virheellisten tai ryhmästä erillään olevien datapisteiden havaitsemisen helpommin. Visualisointitavaksi valittiin 3D-mallinnusohjelmissa yleisesti käytetty tapa, jossa kappaleen (tai ryhmän kappaleita) ympärille piirretään viivagrafiikalla laatikko suurimpien ulottuvuuksien mukaisesti. Koska vaatimukset olivat tämän esitystavan osalta pitkälti samanlaiset kuin koordinaattiakselien osalta, päätettiin toteuttamiseen käyttää samaa matalan tason GL-luokan viivojen piirtoa. Tehtävää varten luotu DrawBounds-luokka kiinnitettiin kameraan. Sen toiminta on hyvin samankaltainen koordinaattiakseleja piirtävän DrawAxes-luokan kanssa. DrawBounds-luokkaan toteutettiin SetBoundaries-metodi, jolla voidaan asettaa kappaletta ympäröivän laatikon ulottuvuudet uudestaan, esimerkiksi kun datan esitystapaa vaihdetaan. CalcPositions-metodi hoitaa piirrettävän laatikon verteksien paikan laskemisen annettujen tietojen mukaisesti ja DrawBoundary-metodi hoitaa laatikon piirtämisen.

6.5 Datapisteet

Koska visualisoinnissa esitetään parvikuvio, tarvittiin datapisteiden esittämiseen jokin tapa. Todettiin, että jos datapisteille halutaan antaa myös koko-ominaisuus (scale), ei pelkkien pisteiden käyttäminen ole riittävää. Tästä johtuen päätettiin mallintaa muutamia erilaisia kolmiulotteisia kappaleita, jotka ovat siluetiltaan selkeästi toisistaan poikkeavia. Näitä mallintaessa pyrittiin pitämään kappaleiden verteksi- ja monikulmiomäärä mahdollisimman pienenä (kuva 35). Jos datapisteitä olisi useampia kuin iris-datasetissä, tulee piirrettävien monikulmioiden suuresta määrästä nopeasti ohjelman ruudunpäivitystä hidastava tekijä.



Kuva 35. Datapisteitä esittävät mallinnetut 3D-kappaleet

Nämä kappaleet mitoitettiin sivultaan noin metrin mittaiseksi, josta kappaleet on helppo skaalata haluttuun kokoon. Lisäksi luotiin myös piste-esitystapa, jolla data voidaan esittää kuutioina, jotka pysyvät aina saman kokoisena ruudulla katseluetäisyydestä riippumatta. Tämän visuaalisen efektin toteuttaa piste-elementtiin liitetty PointScaler-luokka, joka päivittää kappaleen kokoa kameran etäisyyden mukaisesti aina kun ruutua päivitetään.

Pylväsdiagrammivisualisoinnin tavoin tässäkin visualisoinnissa on käyttäjän mahdollista tarkkailla datan arvoja hiiren kursori elementin päälle viemällä. Tätä varten DataContainer MonoBehaviour -luokan ilmentymä lisättiin kuhunkin datapistettä esittävään 3D-malliin. DataContainer-luokka voi tallentaa data -muuttujaansa PointData-luokan ilmentymiä, joihin kunkin parvikuvion pisteen tiedot voidaan tallentaa myöhemmin luettavaksi.

6.6 Datan piirtäminen ruudulle

Pylväsdiagrammivisualisoinnin kaltaisesti Visualizer-luokka vastaa datan renderöinnistä ruudulle. Start-metodissaan Visualizer-luokka kutsuu Init-metodiaan, joka tekee seuraavia asioita. Se asettaa väripaletin visualisoinnille ja kutsuu DataParser-luokan Parse-metodia, joka lataa datan. Tämän jälkeen Init-metodi täyttää käyttöliittymän DropDown-listoja, joista käyttäjä voi tehdä datan esittämiseen liittyviä valintoja. Viimeisenä kutsutaan VisualizeDimensions-metodia (kuva 36).

```
void Init ()
{
    colorRampUse = colorRamps[0];

    dataParser.Parse();

    DimensionSelector.Instance.SetupDropdown(dataParser);
    InspectorManager.Instance.SetupDropdown(dataParser);

    VisualizeDimensions(0, 1, -1, 0, -1, -1, -1);
}
```

Kuva 36. Init-metodi kutsuu eri metodeja visualisoinnin esittämiseksi

VisualizeDimensions-metodi vastaa datan piirtämisestä ruudulle. Käytännössä tämä tarkoittaa sitä, että datan arvoja voidaan kuvata datapisteitä esittävien 3D-mallien ominaisuuksilla. Näitä ovat tässä tapauksessa pisteen x-, y- ja z-koordinaattipaikka 3D-tilassa, kappaleen muoto, väri, koko ja ryhmä. Toisin sanoen, iris-datasetin terälehden pituus-arvot voidaan esittää vaikkapa x-akselilla paikkana, värinä tai kappaleen kokona. VisualizeDimensions-metodi mahdollistaa sen, että datan eri sarakkeiden arvoja voidaan esittää lähes vapaavalintaisesti näiden seitsemän parametrin puitteissa.

Ensimmäisenä VisualizeDimensions-metodi poistaa mahdolliset aiemmat datapisteitä esittävät GameObject-kappaleet ruudulta. Visualisoinnin kannalta on vaikeaa ennalta tietää, miltä väliltä esitettävän datan arvot ovat, suurin arvo voi olla vaikkapa 10 tai 10 miljoonaa. Koska Unity käyttää peleille tyypillisiä ratkaisuja 3D-tilan esittämiseen, ei tarkkuus ole ensimmäinen prioriteetti vaan pääasiassa suorituskyky. Käytetty liukulukujen tarkkuus aiheuttaa sen, että käytännössä jo muutaman kilometrin päässä origosta sijaitsevien kappaleiden paikka voi olla epätarkka. Liian suuri mittaskaala aiheuttaa myös ongelmia kameran toiminnalle - etäällä olevat kappaleet alkavat aiheuttamaan myös ns. z-fighting ilmiötä, jossa renderöijä ei osaa päättää kumpi etäisistä kappaleista on kulloinkin piirrettävässä kuvassa lähempänä kameraa. Toisaalta myös liian laaja datan esitystila voi olla hankala hahmottaa.

Näistä seikoista johtuen, päädyttiin ratkaisuun, jossa kaikki esitettävä data skaalataan aina kuution sisään, jonka sivu on yhden metrin mittainen. Tällöin voidaan olla varmoja, että 3D-tilan esitystarkkuus kappaleille on hyvä, sekä myös valot ja varjot toimivat oletetusti, eikä käyttäjän myöskään tarvitse etsiä datapisteitä tarpeettoman kaukaa. Skaalaus ratkaistiin seuraavalla tavalla: kaikkien datapisteiden paikat tarkistetaan CalculateDataLargestDimension-metodissa. Riippumatta siitä mikä x-, y- tai z-akseleista on kyseessä, etsitään suurin ulottuvuus millä tahansa akselilla. Suurinta ulottuvuutta käytetään jakajana,

jolloin data saadaan skaalattua mahtumaan käytetyn kuution sisään. Alla kuva käytetystä ratkaisusta (kuva 37):

```
largestDim = CalculateDataLargestDimension(dimX, dimY, dimZ);
scale = (1f / largestDim);
```

Kuva 37. Datan skaalauskerroin mahdollistaa datan esittämisen halutussa koossa

Kun skaala tiedetään, käydään kaikki DataParser-luokan data-muuttujan arvot läpi. Koska data on tallennettuna yhtä pitkiin sarakkeisiin, käytetään yhtä saraketta laskurina ja käsitellään datan rivit yksi kerrallaan. Samalla asetetaan data haluttuihin datapisteiden ominaisuuksiin. Ensimmäiseksi määritellään ruudulle piirrettävän GameObject-kappaleen x-, y- ja z-koordinaatit eli paikka 3D-tilassa. Arvot tallennetaan väliaikaisiin muuttujiin (kuva 38).

```
if (dimX != -1)
{
    var typeX = dataParser.data[dimX].type;

    if (typeX == FieldType.Float)
        valX = dataParser.data[dimX].dataFloat[r];
}

if (dimY != -1)
{
    var typeY = dataParser.data[dimY].type;

    if (typeY == FieldType.Float)
        valY = dataParser.data[dimY].dataFloat[r];
}

if (dimZ != -1)
{
    var typeZ = dataParser.data[dimZ].type;

    if (typeZ == FieldType.Float)
        valZ = dataParser.data[dimZ].dataFloat[r];
}
```

Kuva 38. Datan arvojen käyttäminen 3D-paikkakoordinaatteina

Tämän jälkeen määritellään kappaleen väri. Jos data on float-tyyppistä, voidaan sitä käyttää kappaleen värin asettamiseen. Linq-luokkaa käyttämällä selvitetään kyseisen sarakkeen pienin ja suurin lukuarvo, joita käyttämällä lukuarvot voidaan normalisoida 0-1 välille. Normalisoituja lukuarvoja käyttämällä voidaan hieman myöhemmin poimia Gradient-luokalla määritellyistä väriliu'uista sen Evaluate-funktiolla väri. Jos sarakkeessa on String-tyyppisiä arvoja float-tyyppisten sijaan, lasketaan eri sisältöisten String-olioiden määrä ja katsotaan, kuinka mones näistä tämä kyseinen String-tyyppinen arvo on. Näin voidaan

antaa eri String-tyyppisille arvoille omat väriarvot - eli Setosa-, Versicolour- ja Virginica-lajit saavat eri värit väriliu'usta. Muussa tapauksessa kappaleelle annetaan perusväriarvo. Alla koodi, jolla datan arvot muutetaan väriarvoiksi (kuva 39):

```
if (dimColor != -1)
{
    var typeCol = dataParser.data[dimColor].type;

    if (typeCol == FieldType.Float)
    {
        minColor = dataParser.data[dimColor].dataFloat.Min();
        maxColor = dataParser.data[dimColor].dataFloat.Max();

        normalizedColor = (dataParser.data[dimColor].dataFloat[r] - minColor) / (maxColor - minColor);
        valColor = dataParser.data[dimColor].dataFloat[r];
    }
    else
    {
        var kinds = dataParser.data[dimColor].kinds.Count;
        var key = dataParser.data[dimColor].dataString[r];
        var idx = dataParser.data[dimColor].GetKindIndex(key);

        normalizedColor = (float)idx /  (kinds-1);
    }
}
else
{
    normalizedColor = 1f;
}
```

Kuva 39. Datan arvojen muuttaminen väriarvoiksi

Tämän jälkeen kerätään skaalaa varten arvo, joskin GameObject-kappaleen koko asetetaan myöhemmin. Jos visuaaliselle ryhmittelylle on annettu arvo metodikutsussa, mahdollistaa se String-tyyppiselle datalle 3D-kappaleen muodon valitsemisen sen ryhmän mukaan. Visualisoinnin käytössä on seitsemän eri 3D-kappaletta, joten iris-datasetin käytössä ollessa sen kolme eri lajia pystytään siis esittämään eri 3D-muodoilla. Toki kappaleita voisi olla vaikkapa 16 erilaista, mutta käytännössä erinäköisten kappaleiden tunnistaminen tulee visualisoinnin katsojalle sitä vaikeammaksi, mitä enemmän niitä on.

Seuraavaksi VisualizeDimensions-metodi luo GameObject-kappaleen prefab-assetista kukuvaamaan kyseistä datapistettä käyttämällä GameObject-luokan Instantiate-metodia. Luotu GameObject-kappale sijoitetaan paikalleen, sille määritellään parent-kappale sekä asetetaan laskettu väri. GameObject-kappaleeseen lisätään vielä DataContainer MonoBehaviour -luokan kopio, jolloin esitettävästä datasta voidaan tallentaa tietoja PointData-luokkaa käyttämällä GameObject-kappaleeseen. Lisäksi GameObject-kappaleeseen lisätään OnMouseOverHilite-komponentti, joka mahdollistaa korostuksen värillä, kun hiiren kursori viedään sen päälle.

SetDataVisualsScale-metodia käytetään määrittelemään datapisteiden esitettävä koko ja muutamia muita asetuksia, kuten koordinaattiruudukon koko. Liian pienet tai suuret luvut aiheuttavat ongelmia, jos niiden arvoja esitetään 3D-kappaleiden kokoa käyttämällä. Tästä syystä päädyttiin normalisoimaan esitettävien 3D-kappaleiden koot haluttujen minimi- ja maksimiarvojen väliin. SetDataVisualScale-metodi kutsuu CalculateGeoVisualScaling-metodia, joka tekee tämän 3D-kappaleiden kokojen normalisoinnin. Tämä metodi laskee ensin sarakkeen pienimmän ja suurimman arvon CalculateDataScaleRange-apumetodia käyttämällä. Tämän jälkeen luvut normalisoidaan seuraavaa kaavaa käyttämällä. Lisäksi käytetään scaling-arvoa, joka tallentaa graafisen käyttöliittymän scale-liukusäätimen arvon. Se on kerroin, jolla ohjelman käyttäjä voi vaikuttaa kappaleiden yleiseen kokoon. Alla koodi, joka toteuttaa normalisoinnin (kuva 40):

```
for (int i = 0 ; i < geoPointObjects.Count ; i++)
{
    var size = DataParser.Instance.data[scaleCol].dataFloat[i];
    var z = (size - minMax.x) / (minMax.y - minMax.x);
    var z2 = z * (fmax - fmin) + fmin;

    geoPointObjects[i].transform.localScale = Vector3.one * z2 * scaling;
}
```

Kuva 40. Datan arvojen normalisoinnin toteuttava kaava

Aivan viimeisenä VisualizeDimensions-metodi kutsuu vielä kameran keskityksen dataan tekevää CameraController-luokan FocusCameraToData-metodia. Visualizer-luokassa on lisäksi muutamia pieniä apumetodeja, jotka liittyvät pääosin datan ulottuvuuksien mittaamiseen 3D-tilassa.

Visualizer-luokan toteutus toi ilmeiseksi, mitä ongelmia 3D-tilasta seuraa datan esittämiselle. Datan sisältämiä lukuarvoja ei välttämättä voi sellaisenaan esittää 3D-tilassa paikkoina tai kokoina. Liian pienet ja suuret lukuarvot aiheuttavat ongelmia käytettäessä lukuarvoja sekä x-, y-, z-koordinaattien että 3D-kappaleen koon määrittelyyn. Esitettävän datan skaalan normalisointi on myös tarpeen, jos halutaan välttää kameralle liukuluvuista aiheutuvia ongelmia, koska liian etäiset tai suuret kappaleet voivat aiheuttaa koko kameran näkymän muuttumisen käyttökelvottomaksi. Unity ei tarjoa näihin toimenpiteisiin oikeastaan mitään valmiita työkaluja, vaan nämä ongelmat täytyy ratkaista itse.

6.7 Kamera

Datan tarkastelun eri kuvakulmista mahdollistavan kameran liikuttelun koodin pohjana käytettiin pylväsdiagrammivisualisoinnista lainattua CameraController-luokkaa. ZoomCamera- ja RotateObject-metodien lisäksi tarvittiin muutama uusi metodi, koska käyttäjälle

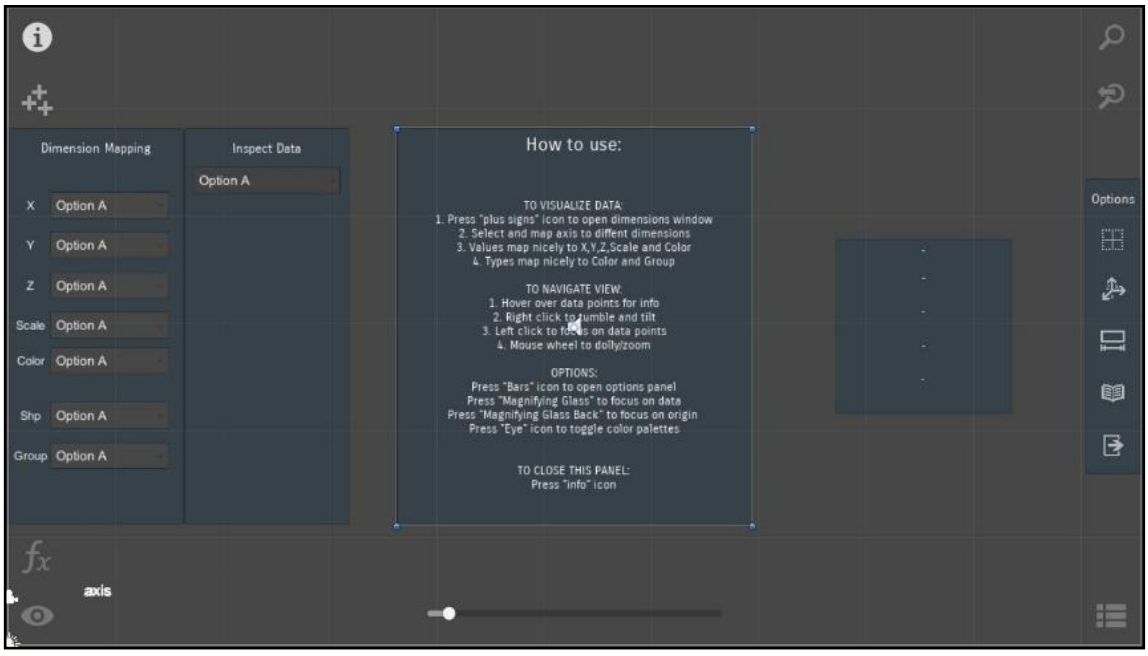
haluttiin mahdollistaa kuvan fokusointi haluttuun datapisteeseen. Lisättiin FocusCameraTo-metodi, joka mahdollistaa sen, että käyttäjä voi klikkailla eri datapisteitä, pyöritellä ja zoomata näkymää halutun pisteen ympäri. Lisättiin myös FocusCameraToData- ja FocusCameraToZero-metodit, jotka mahdollistavat näkymän palauttamisen datan keskipisteeseen tai origoon, jos käyttäjä jostain syystä hukkaa paikan. Näiden metodien toteuttamissa siirtymissä vältettiin äkillisiä liikkeitä, jottei käyttäjä menetä paikan tajuaan. Kukin metodi liikuttaa kameraa tasaisesti kohteeseensa, käyttämällä Coroutine-tyyppistä MoveTowards-metodia (kuva 41).

```
IEnumerator MoveTowards (Vector3 pos)
{
    while (Vector3.Distance(rotateTra.position, pos) > 0.01f)
    {
        rotateTra.position = Vector3.MoveTowards(rotateTra.position, pos, Time.deltaTime * focusSpeed);
        yield return null;
    }
}
```

Kuva 41. Tasaisen liikkeen mahdollistava Coroutine-tyyppinen MoveTowards-metodi

6.8 Graafinen käyttöliittymä

Visualisoinnin graafisesta käyttöliittymästä tuli hyvin samankaltainen kuin pylväsdiagrammivisualisoinnissa. Käyttäjä voi suorittaa kaikki toimenpiteet hiirellä, graafista käyttöliittymää käyttämällä. Ruudulla näkyvien toimintojen lisäksi lisää toimintoja löytyy ikoneita klikkaamalla aukaistavista ikkunoista ja palkeista. Alla graafisen käyttöliittymän elementit (kuva 42):



Kuva 42. Parvikuviovisualisoinnin graafisen käyttöliittymän elementit

Olennaisinta visualisoinnin toiminnallisuudessa on Dimensions Mapping -ikkuna ja sen toiminnot. Käyttäjä voi valita tästä ikkunasta mitä arvoja 3D-kappaleiden parametreissa näytetään. Tästä toiminnallisuudesta vastaa pääosin DimensionSelector-luokka, yhdessä käyttöliittymäelementtien kanssa. Kun käyttäjä tekee valinnan DimensionSelector-luokan täyttämistä DropDown-valikoista, DropDown-elementti kutsuu pylväsdiagrammivisualisoinnin tapaan UserUI-luokan DimensionSelector-luokan OnDimensionChanged-metodia, joka puolestaan kutsuu valituilla arvoilla Visualizer-luokan VisualizeDimensions-metodia. Tämän jälkeen käyttäjä voi nähdä muutoksen visualisoinnissa (kuva 43).

```
public void OnDimensionChanged ()
{
    if (!initDone)
        return;

    var x = dimDropDownX.value != 0 ? dimDropDownX.value-1 : -1;
    var y = dimDropDownY.value != 0 ? dimDropDownY.value-1 : -1;
    var z = dimDropDownZ.value != 0 ? dimDropDownZ.value-1 : -1;

    var color = colorDropDown.value != 0 ? colorDropDown.value-1 : -1;
    var scale = scaleDropDown.value != 0 ? scaleDropDown.value-1 : -1;
    var grouping = groupDropDown.value != 0 ? groupDropDown.value-1 : -1;

    var shape = shapeDropDown.value;

    visualizer.VisualizeDimensions(x,y,z, shape, color, scale, grouping);
}
```

Kuva 43. OnDimensionChanged-metodi kutsuu VisualizeDimensions-metodia käyttäjän tekemien valintojen arvoilla

6.9 Visualisoinnin apuluokat

Parvikuviovisualisointiin on toteutettu pylväsdiagrammivisualisoinnin tavoin UserUI- ja UserUIAudio-luokat käyttöliittymän tapahtumien seuraamiseen ja äänien soittamiseen. UI-elementtien toimintalogiikka on hyvin samankaltainen kuin pylväsdiagrammivisualisoinnissa. Myös Tooltip- ja RayCastItems-luokat toimivat hyvin pitkälti samalla tapaa kuin pylväsdiagrammivisualisoinnissa. Käyttäjä voi tarkkailla näiden avulla esitetyn datan arvoja viemällä hiiren kursorin datapisteen päälle.

Suunnitellusta poiketen visualisointiin kirjoitettiin DescriptiveStatistics-luokka, jossa on metodeja, joilla voidaan laskea kuvailevia tunnuslukuja datasta. Tunnuslukuja voidaan käyttää apuna, kun dataa halutaan tutkia. Niiden avulla voidaan saada nopeammin käsitys siitä, mitä data on. Toteutettuja matemaattisia funktioita ovat:

- Minimi (min).
- Maksimi (max).
- Keskiarvo (mean).

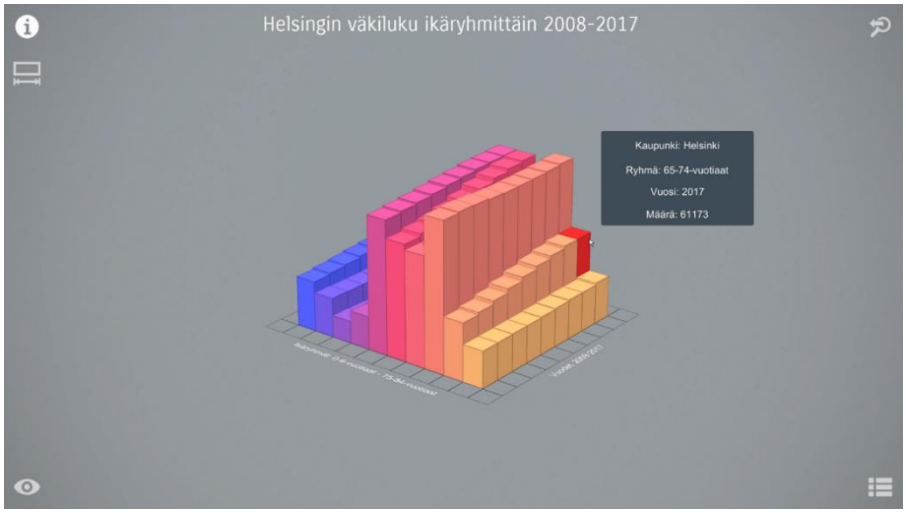
- Mediaani (median).
- Moodi (mode).
- Standardipoikkeama (standard deviation).
- Vaihteluväli (range).
- Varianssi (variance).
- Kvartiili (quartile).
- Kvartiiliväli (interquartile range).

Kyseisten funktioiden toimintaa ei koestettu kovin kattavasti, mutta ne suoriutuivat koetapauksista tuottaen oikean tuloksen. Tarkuutta vaativaan käyttötarkoitukseen suunniteltuun ohjelmaan olisi järkevintä ottaa käyttöön kuitenkin jokin tunnettu ja hyväksi todettu matematiikkakirjasto, jossa on tarvittavat tilastotieteelliset funktiot. Luokan esittämiä tietoja voidaan katsella Inspect Data -ikonia klikkaamalla aukeavasta ikkunasta. Tämän datan näyttämisestä vastaa InspectorManager MonoBehaviour -luokka. InspectorManager-luokan CalculateValues-metodi kutsuu DataParser-luokan dataa käyttämällä DescriptiveStatistics-luokan metodeja, ja tulostaa niistä String-tyyppisen tulosteen, joka näytetään Inspect Data -paneelissa.

7 Tuotos

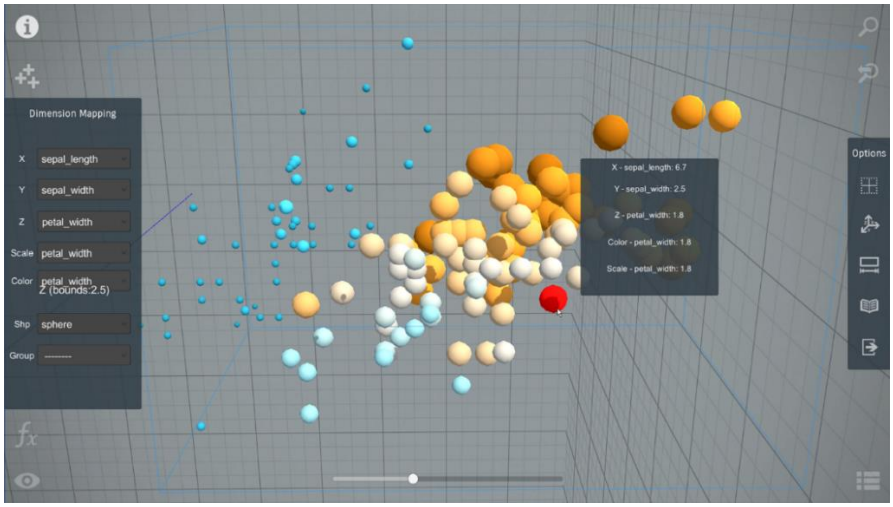
Toiminnallisessa osuudessa tehtiin kaksi visualisointiprototyyppiä, jotka esittävät ladattua dataa kolmiulotteisesti. Ensimmäisen visualisoinnin aiheeksi valittiin pylväsdiagrammi ja toisen visualisoinnin aiheeksi parvi- tai hajontakuvio. Molemmat visualisointiprototyypit täyttävät kaikki niille asetetut toiminnalliset vaatimukset. Visualisointiprototyypit toimivat Windows PC-ympäristössä ajettavina ohjelmina. Ohjelmien graafisia käyttöliittymiä voi kontrolloida hiirellä, lisäksi nappien painallukset tuottavat äänivihjeen. Ohjelmien resoluution ja kuvasuhteen voi valita vapaasti ohjelmien käynnistysvalikoista, ja kaikki graafisten käyttöliittymien elementit pysyvät ruudulla kuvasuhteesta riippumatta. Alustavasti mietittiin myös kosketuskäyttöliittymien toteuttamista, mutta tästä ajatuksesta luovuttiin. Käyttäjä ei voi itse myöskään valita ladattavaa dataa kummassakaan visualisoinnissa, vaan ladattava data on asetettu Unityssä visualisointiohjelmien käyttöön.

Pylväsdiagrammivisualisoinnissa Helsingin väestötietoja voidaan tarkastella vuorovaikutteisesti vuosien 2008-2017 osalta. Pylväsdiagrammivisualisointia voidaan pyöritellä ja siitä voi siirtyä kauemmas tai lähemmäs hiiren rullaa pyörittämällä eri suuntiin. Käyttäjä voi myös suunnitellun toiminnallisuuden mukaisesti rajata näytettäviä pylväitä sekä vuosien että ikäryhmien mukaan. Visualisoinnin voi myös nollata, jolloin pylväät palautetaan alkutilanteeseen, kameran paikka nollataan ja sen liike pysäytetään. Lisäksi pylväiden päälle hiiren kursorin kuljettamalla käyttäjä voi tarkastella eri vuosien väestötietoja kyseisen ikäryhmän osalta tooltip-ikkunaa käyttämällä. Kaksiulotteisen paperilla tai verkkosivulla katsottavan kuvion sijaan on mahdollista katsoa pylväitä kaupunkimaisena 3D-kuviona. Vaikka osa kuviosta voi paikoitellen peittyä toisten osien alle, voi kuvakulmaa vapaasti vaihtamalla tai näytettäviä arvoja rajaamalla korjata tällaista tilannetta. Kuvassa 44 pylväsdiagrammivisualisointi (lisää kuvia liitteessä 3).

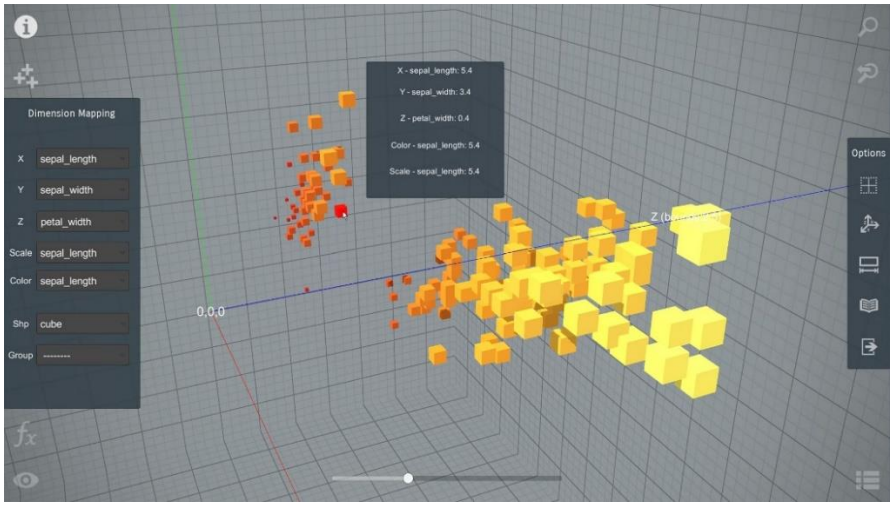


Kuva 44. Näkymä pylväsdiagrammivisualisoinnista

Parvikuviovisualisoinnin tuloksena saatiin vuorovaikutteinen iris-datasetin eksploratiivisen analysoinnin mahdollistava prototyyppi. Visualisoinnissa on suunnitellusti mahdollista vapaavalintaisesti esittää datasetin viisi eri ulottuvuutta datapisteitä esittävien 3D-kappaleiden x-, y- z-koordinaatteina, kokona, värinä ja joissain tapauksissa myös muotona. Suunniteltujen ominaisuuksien lisäksi toteutettiin erilaisia tilastollisia lukuarvoja datalle näyttävä Inspector-paneeli. Jokin 3D-tilassa toteutettu datan arvoja esittävä kuvaaja olisi ollut hyvä lisä parvikuvion visualisointiin, mutta tätä ei ehditty kuitenkaan toteuttaa. Kuvissa 45 ja 46 parvikuviovisualisointi (lisää kuvia liitteessä 4).



Kuva 45. Näkymä parvikuviovisualisoinnista



Kuva 46. Kuvakulman ja kameran keskityksen vaihtaminen on mahdollista hiirtä käyttämällä

Tuotosta ei voitu kokonaisuudessaan sisällyttää opinnäytetyön liitteeksi, koska Unity-projektia ei voi esittää luettavassa muodossa tekstinä. Lisäksi koodia on noin 3000 riviä. Raportissa käydään kuitenkin olennaisimmat ratkaisut yksityiskohtaisesti läpi.

8 Pohdinta

Tämän opinnäytetyön tarkoituksena oli selvittää datan 3D-visualisoinnin toteuttamista Unity-pelimoottoria käyttämällä. Lisäksi haluttiin selvittää asiaan liittyviä haasteita ja ongelmia.

Opinnäytetyön tekemiseen varattu aika oli suhteellisen rajallinen (noin yksi kuukausi), joten tämä aiheutti tiettyjä rajoitteita työn toteuttamiselle ja laajuudelle. Opinnäytetyön varsinainen toteuttaminen aloitettiin ensin tietoperustan kirjoittamisesta, jolloin tutustuttiin tarkemmin aihepiiriä käsittelevään kirjallisuuteen ja muihin tietolähteisiin. Näin saatiin selvitettyä aihepiiriin liittyvät peruskäsitteet, sekä myös kasvatettua tietoisuutta erilaisista teoreettisista asioista, jotka liittyvät visualisointiin. Tietoperustaa varten kerättiin runsaasti materiaalia, ehkä juuri tästä johtuen keskeisten käsitteiden hahmottaminen vei runsaasti aikaa.

Tämän jälkeen siirryttiin suunnittelemaan jo alustavasti mietittyä tuotosta, eli vuorovaikutteisia datan 3D-visualisointiprototyyppejä. Suunnitteluvaihe eteni suhteellisen yllätyksettömästi. Prototyyppien toteutuksen laajuutta hahmoteltiin, sekä arvioitiin eri tekniikoiden ja vaihtoehtojen etuja ja riskejä, niin että pystyttäisiin rajaamaan kokonaisuus, joka olisi toteutettavissa käytettävissä olevan ajan rajoissa. Tämä tarkoitti tiettyjen kiehtovalta vaikuttavien algoritmien ja lähestymistapojen rajaamista pois.

Tuotoksen tekemisessä tuli hieman kiire, koska aikataulu lyheni vielä alun perin suunnitellusta jonkin verran. Tästä johtuen pyrittiin etenemään suunnitelmallisesti, ja jakamaan molemman visualisoinnin osalta mahdollisimman paljon koodia. Vastaan tulleisiin ongelmiin etsittiin ratkaisuja pääasiassa Unityn keskustelufoorumilta ja hakukoneilla.

Opinnäytetyön tekemiseen käytetty työmäärä jakautui tekijöiden kesken suhteellisen tasaisesti. Ohjelmakoodia kirjoitettiin molemman toimesta. Joitain tehtäviä valittiin osaamisen mukaan. Olli Sorjonen esimerkiksi toteutti shader-ohjelmointia, kun taas Sami Sorjonen toteutti tiettyjä käyttöliittymään liittyviä asioita.

Opinnäytetyön tekstiä kirjoittivat molemmat. Tekstiä kirjoitettiin samanaikaisesti, jonka jälkeen tehtiin tekstin jäsentelyä parityönä. Molemmat opinnäytetyöntekijät olivat kirjoittaneet suhteellisen vähän asiatekstiä, joten opinnäytetyön tekstin viimeistely vei huomattavan osan käytettävissä olevasta ajasta.

8.1 Tavoitteiden saavuttaminen

Opinnäytetyön tavoitteissa onnistuttiin mielestämme kohtuullisen hyvin, sekä opinnäytetyön teksti että visualisoinnit saatiin toteutettua suunnitellulla tavalla aikataulussa. Visualisointeihin saatiin luotua myös muutamia toiminnallisuuksia, joita ei ollut mietitty toteutettavaksi.

Toteutetuista datan lataajista tuli suhteellisen kertakäyttöisiä, etenkin pylväsdiagrammivisualisoinnin osalta. Parvikuviovisualisoinnin lataaja on kuitenkin hieman yleiskäyttöisempi, käytetty Linq-luokka osoittautui hyödylliseksi näytettävän datan valitsemiseen SQL-hakujen kaltaisesti ladatusta datasta.

Vaikka visualisointien mitta-asteikot tai koordinaattiruudukot ovat viitteellisiä, voi esimerkiksi akselien suuntaisia kuvakulmia ja tooltip-paneelia käyttämällä tutkia datan arvoja suhteellisen tarkasti. Tarkka datan visualisointi vaatisi kohtisuorat koordinaattiakselien suuntaiset näkymät sekä näiden päälle piirretyt ruudukot, jolloin voidaan kyseenalaistaa osittain koko 3D-esityksen hyöty tällaiseen käyttötarkoitukseen.

Visualisoinnit jäivät kuitenkin käytännössä tutkielmiksi. Jotta näistä ohjelmista olisi saatu oikeasti käyttökelpoisia työkaluja, olisi näihin pitänyt toteuttaa huomattava määrä lisää erilaisia toiminnallisuuksia. Tällaista projektia ja sen löydöksiä voisi mielestämme lähinnä käyttää kattavamman pelimoottorissa toteutetun visualisointiohjelmistokehyksen suunnittelun ja aiheen jatkotutkimuksen lähtökohtana. Joka tapauksessa, ratkaisut joihin päädyttiin visualisointeja tehdessä, ovat mielestämme käyttökelpoisia visualisointien toteuttamiseen Unityssä. Visualisointien käyttöliittymät saatiin toteutettua suhteellisen käytettäviksi, ja kameran liikuttelu sekä datan katselu tuntui myös suhteellisen luontevalta.

8.2 Oppiminen

Opinnäytetyötä tehdessä tutustuttiin Unityn ominaisuuksin, joihin ei ole ollut mahdollisuutta perehtyä aikaisemmin. Vaikka opinnäytetyössä ei päädytty käyttämään esimerkiksi Unityn 3D-mallien instansointi-ominaisuuksia, perehdyttiin näiden toimintaan, niin että pystyttiin muodostamaan näkemys, mitä niillä pystyttäisiin tekemään visualisoinneissa, jos dataa olisi enemmän.

Myös asiatyylisen tekstin kirjoittamiseen saatiin lisää harjoitusta. Opinnäytetyötä tehdessä visualisoinnin suomenkielistä termistöä jouduttiin hieman miettimään ja etsimään luotettavia lähteitä sanastolle. Tieto näkyväksi kirjan laatijat ovat tehneet visualisoinnin sanaston

(Koponen ym. 2016, 364-369), josta oli hyötyä suomenkielisen termistön selvittämisessä. Aihepiirin sanasto tuli tutummaksi työn tekemisen myötä. Sama koskee myös sanastoa 3D-grafiikan osalta. Vaikka molemmat opinnäytetyöntekijät ovat työskennelleet pitkään 3D-grafiikan ja pelien parissa, haluttiin selvittää suomenkieliseksi vakiintunutta sanastoa, jota etsittiin pääasiassa opinnäytetöistä. Yhtenä lähteenä toimi Janne Joensuun opinnäytetyö (Joensuu 2016), jonka aiheena on 3D-alan sanasto.

8.3 Johtopäätökset

Vaikka Unityä onkin pyritty sitä kehittävän yrityksen toimesta muokkaamaan viimeisen muutaman vuoden aikana monipuolisempiin käyttötarkoituksiin soveltuvaksi, on tässä vielä paljon tekemistä. Tuntuu siltä, että visualisointeja ei ehkä ole selkeästi ajateltu yhtenä Unityn käyttökohteista.

Datan tuomiseen pelimoottoriin liittyy haasteita. Unityssä ei ole vielä sisäänrakennettuna toimintoja, joilla datan lataaminen onnistuisi helposti. Tiedostoformaattituki datan lataamiseksi Unityyn on kovin rajoittunut ja pääosin tuetaan geometrisia formaatteja (Buyuksalih ym. 2017, 165). Erilaisia kolmannen osapuolen ratkaisuja tähän tarkoitukseen kuitenkin löytyy. Näitä ei kuitenkaan ehditty kokeilemaan tätä opinnäytetyötä tehdessä.

Datan esittämiselle on myös useita haasteita. Monia visualisointiin tarvittavia komponentteja ei ole valmiina saatavilla. Unitylle on myös tällä hetkellä saatavilla niukasti kolmansien osapuolten valmistamia työkaluja, joiden avulla voisi toteuttaa ammattimaisia datan 3D-visualisointeja.

Jotta dataa voidaan esittää 3D-tilassa, tulisi huomioida, että visualisointien käyttöön tulevan datan on hyvä olla esikäsiteltyä, niin että sen esittämisessä ei tarvitsisi keskittyä virheiden huomioimiseen. Kyseinen data tulee ensin esikäsitellä jossakin muussa ohjelmistossa kuin Unity tai rakentaa työkaluja tähän käyttöön, koska näitä toiminnallisuuksia ei ole saatavilla valmiina ominaisuuksina. Käytössä olevaa dataa voidaan ladata internetistä, tiedostoista tai suoraan Unityn Inspector-ikkunassa asetetuista komponenttien kentistä. Unityn käyttämä C#-ohjelmointikieli tarjoaa lukuisia tyypillisiä datarakenteita tiedon tallentamiseen ohjelmaa varten, jotka ovat hyvin samankaltaisia kuin esimerkiksi Java-kielessä. Tietyt rakenteet kuten List tai Dictionary voivat sopia paremmin käyttötarkoitukseen riippuen datan rakenteesta. On myös mahdollista tehdä omia käytössä olevan datan tallentamiseen soveltuvia luokkia. Joissain tapauksissa näiden esittäminen Unityn Inspector-ikkunassa voi vaatia Unityn editorilaajennuksien kirjoittamista.

Datan esittämiseen 3D-grafiikalla liittyy useita vaatimuksia. Mittaskaalan normalisointi voi olla tarpeen, ja sitä tehdään usein, kun datan visualisointia valmistellaan (Bartke 2005, 4). Tämä onkin yksi vaatimuksista, joka tulee huomioida, jotta dataa voidaan esittää 3D-tilassa tarpeeksi joustavasti. Pelimoottorit eivät pysty useinkaan esittämään hyvin pieniä tai suuria kappaleita ongelmitta, eikä Unity ole poikkeus. Unity ei sisällä toiminnallisuuksia arvoiltaan suuresti vaihtelevan datan tai 3D-mallien koon normalisointiin, joten tähän pitää myös itse rakentaa tarvittavat toiminnallisuudet. Myös mitta-asteikot ovat usein tarpeen visualisoinneissa. Tasavälisten asteikkojen käyttö onkin yksi visuaalisen vertailtavuuden keskeisistä edellytyksistä (Kuusela 2000, 34). Jos koordinaattiruudukko tarvitaan, sen piirtäminen pitää toteuttaa alusta lähtien, koska tällaista ei ole tarjolla kuin Unityn editori-ikkunassa. Datapisteiden esittämiseen vaaditaan myös jonkinlaisia 3D-malleja, mutta Unity tarjoaa vain muutamia erilaisia 3D-peruskappaleita. Näissä tapauksissa pitää myös miettiä käytettävien kappaleiden tarkkuutta suhteessa esitettävien kappaleiden määrään.

Yksi haitoista datan esittämiselle Unityssä on se, että Unityssä ei ole myöskään sisäänrakennettuna tilastotieteen matemaattisia funktioita ja toimintoja, joita todennäköisesti tarvittaisiin visualisoinnin tueksi. Tämä vaatisi ulkopuolisten kirjastojen käyttöönottoa tai ylipäätänsä tutkimista, mitä Unityn kanssa yhteensopivia kirjastoja on olemassa ja kuinka työlästä niiden integroiminen Unityyn on. Unity ei tarjoa myöskään parametrisiä 3D-malleja, joten datan esittämiseen käytettävät 3D-mallit tulee mallintaa itse, tai käyttää jotain ratkaisua parametristen 3D-mallien luomiseen.

Kaikista puutteista huolimatta, datan esittämiseen 3D-tilassa liittyy myös lukuisia mahdollisia etuja. Opinnäytetyöntekijöille syntyi vaikutelma, että 3D-tilan hyödyntäminen pelimoottoria käyttäen visualisointiin voi olla hyödyllistä. Unity tarjoaa paljon etuja, joita perinteiset visualisointiohjelmat tai -työkalut eivät tarjoa. Ward, Grinstein ja Keim ovat kirjassaan “Interactive Data Visualization” korostaneet, että datan eri ulottuvuuksia voidaan esittää käyttämällä kappaleiden muotoa, väriä, skaalaa ja asentoa hyödyksi (Ward ym. 2010, 25). Unityn GameObject-kappaleita käyttämällä onkin erittäin helppoa esittää datapisteitä, palkkeja ja tasoja eri tavoin 3D-tilassa. Myös partikkeleita voidaan käyttää datapisteiden esittämiseen, mutta tätä ei otettu mukaan tehtyihin visualisointeihin, joskin tätä kokeiltiin onnistuneesti. Unity tarjoaa hyvät mahdollisuudet toteuttaa laadukkaasti renderöityä grafiikkaa, sulavalla ruudunpäivityksellä. Näitä kehittyneitä renderöinti- ja valaistustekniikoita käyttämällä voidaan toteuttaa esityksiä, jotka vastaavat enemmän ihmisen normaaleja aistihavaintoja ympäristöstä, jolloin datan tarkastelusta saadaan luonnollisemman kaltaista. Ware on kirjoittanut mm. luonnonmukaisen spekulaarisen valaistuksen, varjostuksen ja tekstuurin merkityksestä datan ymmärrettävyyden kannalta ja kuinka ne voivat auttaa tekemään paremmin havaintoja (Ware 2013, 39-40, 247-248, 253). Unityn avulla on myös

mahdollista tehdä visualisoinneista vuorovaikutteisia. Erilaisten toiminnallisuuksien rakentaminen on suhteellisen vaivatonta. Lisäksi Unitystä on mahdollista saada visualisointi esitettäväksi lukuisille eri laitealustoille. Tämä ei todennäköisesti onnistu, jos visualisointeja tehdään jollain muulla ohjelmistolla.

Vaikka alan kirjallisuudessa käsitellään kolmiulotteisuuteen liittyviä ongelmia, myös 3D-tilassa voidaan esittää dataa niin, että ymmärrettävyys säilyy, kun huomioidaan asioita, jotka liittyvät 3D-tilaan ja siinä navigointiin. Esimerkiksi kiinnostuksen kohteen ympäri pyörivä katselupiste voi auttaa hahmottamaan datavisualisoinnin syvyyssuhteita, erityisesti kun dataelementit eivät ole kiinnitettyinä muihin kappaleisiin 3D-tilassa (Ware 2013, 258). Aivan samoin kuin 2D-visualisointien ollessa kyseessä, kuvan rajauksen ja paikan muutokset tulisi toteuttaa siirtymin, jottei katsojan tilanhahmotusta hankaloiteta. Etenkin esitettävien datamäärien kasvaessa on todennäköistä, että 3D-tilassa esitetyistä visualisoinneista saattaa olla hyötyä, kun dataa tutkiva katsoja voi tehdä erilaisia havaintoja 3D-tilassa liikkuessaan. Benjamin Resnick (2017) on kirjoittanut samankaltaisesta lähestymistavasta.

8.4 Jatkokehitysajatuksia

Jos toteutetuista visualisointiprototyypeistä haluttaisiin kehittää oikea visualisointijärjestelmä, tarvittaisiin datan lataamisen mahdollistavia ominaisuuksia. Esikäsittely on huomattavan laaja työvaihe. Kovinkaan laajoja esikäsittelytoimintoja ei kannattaisi todennäköisesti toteuttaa Unityssä. Lähinnä erilaiset tiedostojen siivoamiseen (välimerkkien vaihto ym.) tai ladattavan datan valintaan liittyvät toiminnallisuudet voisivat olla hyödyllisiä. Lataamiselle voitaisiin suunnitella toteutettavan useamman eri tiedostoformaatin tuki, sekä tehdä lataajista toiminnallisuudeltaan sen tasoisia, että ne selviytyvät esikäsitellyn datan tuomisesta ohjelmaan.

Näihin prototyyppeihin ei otettu käyttöön Unityn erilaisia visuaalisia efektejä, joita voitaisiin käyttää avuksi datan luettavuuden parantamisessa. Esimerkiksi ambient occlusion -tekniikoita (SSAO, HBAO tms.) käyttämällä voitaisiin erityisesti isommista datamassoista tuoda huomattavasti paremmin esille erilaisia muotoja. Ihmisen näkökyvyn tehokkaita, luontaisia ominaisuuksia voitaisiin käyttää avuksi, esimerkiksi tietynlaisten poikkeamien havaitsemiseen, joita olisi vaikeampi havaita alkeellisemmin esitetyistä kappaleista. Mm. Tarini, Cignoni ja Montani (2006) sekä Eichelbaum, Scheuermann ja Hlawitschka (2013) ovat tutkineet ambient occlusion -tekniikoiden käyttöä datan visualisoinnissa. Lisäksi myös tila-avaruuden hahmottamista voitaisiin parantaa erilaisilla valinnaisilla kuvaefekteillä. Tällaisia

ovat Waren mukaan esimerkiksi sumu (2013, 268) ja syväterävyys (2013, 255-256), jotka tarjoavat parempia syvyysvihjeitä.

Koska ns. big datan visualisoinnin merkitys kasvaa jatkuvasti (Clark 2014), olisi syytä miettiä, millä tavoin suuria datamääriä voidaan esittää. Suurien datamäärien lataamista muistiin pitäisi segmentoida jollakin tavalla, esimerkiksi pitämällä vain osa tarkasteltavasta datasta näkyvissä täydellä tarkkuudella ja muut datan alueista esitettäisiin pelkistettynä. Vaikka näytönohjainten ja pelimoottorien teho on kasvanut viime vuosina, suuria datamääriä visualisoidessa joudutaan joka tapauksessa tekemään optimointeja, jotta dataelementit saadaan esitettyä ruudulla reaaliaikaisesti. Datan esittämiseen voitaisiin käyttää esimerkiksi GPU-instansoituja kappaleita, jolloin 3D-tilaan voitaisiin piirtää huomattavasti suurempia määriä kappaleita, kuin GameObject-kappaleita tai partikkeleita käyttämällä. Myös geometria-shadereilla saatettaisiin saavuttaa samankaltaista nopeutusta ohjelman toimintaan, mutta niitä käytettäessä ei voida höydyntää Unityn sisäänrakennettuja valaistustoimintoja, jolloin tuki kyseisille toiminnallisuuksille on rakennettava itse. Näitä instansointi-menetelmiä käytettäessä jouduttaisiin kuitenkin miettimään ratkaisua, miten esimerkiksi kappaleiden valitseminen voidaan toteuttaa. Nämä instanssit eivät ole Unityn fysiikka-järjestelmän havaittavissa, eikä esimerkiksi niiden kohdalla ruudulla tehtyjä hiiren painalluksia tai kosketuksia pystytä rekisteröimään Unityn sisäänrakennetuilla toiminnoilla.

Virtuaalitodellisuudelle tuen lisääminen voisi parantaa eksploratiivista datan analysointia. VR-lasien käyttäjä voisi katsella luonnollisesti eri puolille dataa, ja esimerkiksi dataelementtejä koskettamalla siirtyä niitä lähemmäs. Virtuaalitodellisuus-ympäristössä erilaiset näköhavaintoihin liittyvät asiat, kuten liikkeen havainnointi, liikeparallaksi ja kappaleiden sijainnin hahmottaminen voivat olla helpompia. Näin käyttäjä voisi mahdollisesti myös havainnoida datassa esiintyviä piirteitä paremmin. Mm. Olshannikova, Ometov, Koucheryavy ja Olsson (2015, 21) ovat tutkineet tätä.

Lisätty todellisuus sen sijaan todennäköisesti tarjoaisi datan esittämiseen toistaiseksi eniten mahdollisuuksia mainos- tai esittelykäytössä. AR-käytössä puhelin tai tabletti on kuitenkin hyvin pieni katseluikkuna 3D-tilaan, joten näiden ei voida olettaa tarjoavan kovin immersiivistä kokemusta. Toisaalta AR-lasien yleistyessä voitaisiin käyttää samanlaisia mahdollisuuksia datan seassa liikkumiseen, kuten VR-järjestelmiä käytettäessä.

Lähteet

Andrew, K. 2014. Unity acquires Tsugi to launch Unity Cloud Build. Luettavissa: http://www.pocketgamer.biz/news/59747/unity-acquires-tsugi-to-launch-unity-cloud-build/. Luettu: 17.4.2018.

Anscombe, F. 1973. Graphs in Statistical Analysis. The American Statistician, 27, 1, s.17-21. Luettavissa: http://www.sjsu.edu/faculty/gerstman/StatPrimer/anscombe1973.pdf. Luettu: 17.4.2018.

Bartke, K. 2005. 2D, 3D and High-Dimensional Data and Information Visualization. Luettavissa: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.83.3421&rep=rep1&type=pdf. Luettu: 18.4.2018.

Bergeron, C. 2017. Unity 2017.3: Speeding up Compile Time with the New Assembly Definitions. Luettavissa: https://chuckbergeron.io/blog/2017/12/20/speeding-up-unity-compile-time-with-the-new-assembly-definitions.html. Luettu: 17.4.2018.

Biography 2018. Florence Nightingale Biography. Luettavissa: https://www.biography.com/people/florence-nightingale-9423539. Luettu: 20.4.2018.

Bryson, S. 1993. Virtual Environments in Scientific Visualization. Luettavissa: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.27.6474&rep=rep1&type=pdf. Luettu: 2.5.2018.

Buyuksalih, I., Bayburt, S., Buyuksalih, G., Baskaraca, A., Karim, H. & Rahman A. 2017. 3D modelling and visualization based on the unity game engine - advantages and challenges. ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, 4, s. 161-166. Luettavissa: https://www.isprs-ann-photogramm-remote-sens-spatial-inf-sci.net/IV-4-W4/161/2017/isprs-annals-IV-4-W4-161-2017.pdf. Luettu: 18.4.2018.

Campbell, C. 2014. The billion-dollar game dev trading post. Luettavissa: https://www.polygon.com/2014/4/7/5575972/the-billion-dollar-game-dev-trading-post. Luettu: 17.4.2018.

Chevalier, E. 2016. Introducing Unity Labs. Luettavissa: https://vrscout.com/news/introducing-unity-labs/. Luettu: 17.4.2018.

Clark, D. 2014. Data Visualization Is The Future - Here's Why. Luettavissa: https://www.forbes.com/sites/dorieclark/2014/03/10/data-visualization-is-the-future-heres-why/. Luettu: 2.5.2018.

Cleveland, W. & McGill, R. 1984. Graphical Perception: Theory, Experimentation, and Application to the Development of Graphical Methods. Teoksessa Journal of the American Statistical Association, 79, 387, s. 531-554. Luettavissa: http://www.diliarana-sirova.com/assets/PSYC579/pdfs/03.2-Cleveland&McGill.pdf. Luettu: 17.4.2018.

Conditt, J. 2014. Former EA CEO John Riccitiello is now CEO of Unity. Luettavissa: https://www.engadget.com/2014/10/22/former-ea-ceo-john-riccitiello-is-now-ceo-of-unity/. Luettu: 17.4.2018.

Cooper, T. 2018. Introduction to Shader Graph: Build your shaders with a visual editor. Luettavissa: https://blogs.unity3d.com/2018/02/27/introduction-to-shader-graph-build-your-shaders-with-a-visual-editor/. Luettu: 17.4.2018.

Crunchbase. Unity. Luettavissa: https://www.crunchbase.com/organization/unity-technologies. Luettu: 17.4.2018.

Cruz-Neira, C., Leigh, J., Papka, M., Barnes, C. Cohen, S., Das, S., Engelmann, R., Hudson, R., Roy, T., Siegel, L., Vasilakis, C., DeFanti, T. & Sandin, D. 1993. Scientists in Wonderland : A Report on Visualization Applications in the CAVE Virtual Reality Environment. Luettavissa: https://pdfs.semanticscholar.org/08fa/42648a1ab67b6e45168a2aa1dac134ec7679.pdf. Luettu: 2.5.2018.

Cruz-Neira, C., Sandin, J., DeFanti, T., Kenyon, R. & Hart, J. 1992. The Cave. Audio Visual Experience Environment. Communications of the acm, 35, 6, s. 64-72. Luettavissa: https://www.evl.uic.edu/documents/cacm92-cave-cruz-neira.pdf. Luettu: 2.5.2018.

CVR API. Over the edge I/S. Luettavissa: http://cvrapi.dk/virksomhed/dk/over-the-edge-is/27877575. Luettu: 17.4.2018.

Dahlberg, A. 2017. Unity technologies gets a $400 million investment. Luetavissa: https://www.insidescandinavianbusiness.com/article.php?id=15. Luettu: 17.4.2018.

Downie, C. 2016. Evolution of our products and pricing. Luettavissa: https://blogs.unity3d.com/2016/06/16/evolution-of-our-products-and-pricing/. Luettu: 17.4.2018.

Dunstan, J. 2015. A Model-View-Controller (MVC) Pattern for Unity. Luettavissa: https://jacksondunstan.com/articles/3092. Luettu: 7.5.2018.

Eichelbaum, S., Scheuermann, G. & Hlawitschka, M. 2013. PointAO — Improved Ambient Occlusion for Point-based Visualization. Luettavissa: https://www.researchgate.net/publication/259904409_PointAO_-_Improved_Ambient_Occlusion_for_Point-based_Visualization. Luettu: 2.5.2018.

Feiner, S., MacIntyre, B., Haupt, M. & Solomon, E. 1993. Windows on the World: 2D Windows for 3D Augmented Reality. Teoksessa UIST ’93 (ACM Symp. on User Interface Software and Technology). s. 145-155. ACM. New York. Luettavissa: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.39.7881&rep=rep1&type=pdf. Luettu: 2.5.2018.

Ferster, B. 2013. Interactive Visualization. The MIT Press. Cambridge.

Fine, R. 2017. UnityScript’s long ride off into the sunset. Luettavissa: https://blogs.unity3d.com/2017/08/11/unityscripts-long-ride-off-into-the-sunset/. Luettu: 17.4.2018.

Glancey, J. 2015. The London Underground map: the design that shaped a city. Luettavissa: http://www.bbc.com/culture/story/20150720-the-london-underground-map-the-design-that-shaped-a-city. Luettu: 17.4.2018.

Haas 2014. A History of the Unity Game Engine. Luettavissa: https://web.wpi.edu/Pubs/E-project/Available/E-project-030614-143124/unrestricted/Haas_IQP_Final.pdf. Luettu: 17.4.2018.

Healey, C. & Enns, J. 2012. Attention and Visual Memory in Visualization and Computer Graphics. IEEE transactions on visualization and computer graphics, 18, 7, s. 1170-1188. Luettavissa: https://www.csc2.ncsu.edu/faculty/healey/download/tvcg.12a.pdf. Luettu: 17.4.2018.

Heer, J., Bostock, M. & Ogievetsky, V. 2010. A Tour Through the Visualization Zoo. Communications of the acm, 53, 6, s. 59-67. Luettavissa: https://idl.cs.washington.edu/files/2012-VisualizationZoo-CACM.pdf. Luettu: 17.4.2018.

Helgason, D. 2013. Sunsetting Flash. Luettavissa: https://blogs.unity3d.com/2013/04/23/sunsetting-flash/. Luettu: 17.4.2018.

Internet Archive. Editor Version Release Dates. Luettavissa: http://web.archive.org/web/20141015144227/http://docs.unity3d.com/Manual/ReleaseDates.html. Luettu: 18.4.2018.

Jianding, Z. 2015. Zhang Junbo: why SilkCloud chose to be acquired by Unity. Luettavissa: http://prog3.com/article/2015-08-27/2825563. Luettu: 17.4.2018.

Joensuu, J. 2016. 3D-alan sanasto: 3D-grafiikan termit suomeksi. Luettavissa: http://urn.fi/URN:NBN:fi:amk-2016060612045. Luettu: 17.4.2018.

Johansen, R. 2009. Why You Probably Don't Need a Source Code License. Luettavissa: https://blogs.unity3d.com/2009/03/20/why-you-probably-dont-need-a-source-code-license/. Luettu: 17.4.2018.

Knaflic, C. 2015. Storytelling with data. John Wiley & Sons. Hoboken.

Koponen, J., Hilden, J. & Vapaasalo, T. 2016. Tieto näkyväksi. Aalto ARTS Books. Helsinki.

Kosara, R. 2007. Visualization Criticism - The Missing Link Between Information Visualization and Art. Luettavissa: https://pdfs.semanticscholar.org/88fa/9dda1aeb4e8673a2a9ed4b431c0195a1a728.pdf. Luettu: 17.4.2018.

Kumparak, G. 2014. Unity Acquires Applifier To Bring Shareable Instant Replays To More Games. Luettavissa: https://techcrunch.com/2014/03/13/unity-acquires-applifier-to-bring-shareable-instant-replays-to-more-games/. Luettu: 17.4.2018.

Kuusela, V. 2000. Tilastografiikan perusteet. Edita. Helsinki.

Low, S. 2017. Timeline: Keeping the ADAM films on track. Luettavissa: https://blogs.unity3d.com/2017/12/06/adam-timeline/. Luettu: 17.4.2018.

Marketwired 2011. Unity Technologies Lands $12 Million in Series B Funding Led by WestSummit Capital and iGlobe Partners. Luettavissa: http://www.marketwired.com/press-release/unity-technologies-lands-12-million-series-b-funding-led-westsummit-capital-iglobe-partners-1540593.htm. Luettu: 17.4.2018.

McCown, F. 2011. Java and C# Comparison. Luettavissa: https://www.harding.edu/fmccown/java_csharp_comparison.html. Luettu: 17.4.2018.

Microsoft 2015. Introduction to LINQ (C#). Luettavissa: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/introduction-to-linq. Luettu: 17.4.2018.

Microsoft 2018. StreamReader Class. Luettavissa: https://msdn.microsoft.com/en-us/library/system.io.streamreader. Luettu: 20.4.2018.

Minotti, M. 2015. Unity’s Cloud Build is coming to PC, Mac, and Linux — features now free for all. Luettavissa: https://venturebeat.com/2015/09/21/unitys-cloud-build-is-coming-to-pc-mac-and-linux-features-now-free-for-all/. Luettu: 17.4.2018.

Olshannikova, E., Ometov, A., Koucheryavy, Y. & Olsson, T. 2015. Visualizing Big Data with augmented and virtual reality: challenges and research agenda. Luettavissa: https://link.springer.com/content/pdf/10.1186/s40537-015-0031-2.pdf. Luettu: 2.5.2018.

Paczkowski, L. 2018. Replacing MonoDevelop-Unity with Visual Studio Community starting in Unity 2018.1. Luettavissa: https://blogs.unity3d.com/2018/01/05/discontinuing-support-for-monodevelop-unity-starting-in-unity-2018-1/. Luettu: 17.4.2018.

Porter 2013. Unity: Now You're Thinking With Components. Luettavissa: https://gamedevelopment.tutsplus.com/articles/unity-now-youre-thinking-with-components--gamedev-12492. Luettu: 17.4.2018.

Resnick, B. 2017. Visualizing High Dimensional Data In Augmented Reality. Luettavissa: https://medium.com/inside-machine-learning/visualizing-high-dimensional-data-in-augmented-reality-2150a7e62d5b. Luettu: 3.5.2018.

Rincon, C. 2018. Get early access to Unity 2018.1: the beta is out. Luettavissa: https://blogs.unity3d.com/2018/01/10/get-early-access-to-unity-2018-1-the-beta-is-out/. Luettu: 17.4.2018.

Schmid, S. Entitas - The Entity Component System Framework for C# and Unity. Luettavissa: https://github.com/sschmid/Entitas-CSharp. Luettu: 17.4.2018.

Shafir, O. 2018. The 2D tool that’s changing how levels are built. Luettavissa: https://blogs.unity3d.com/2018/01/22/the-2d-tool-thats-changing-how-levels-are-built/. Luettu: 17.4.2018.

Shneiderman, B. The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations. Luettavissa: https://www.cs.umd.edu/~ben/papers/Shneiderman1996eyes.pdf. Luettu: 17.4.2018.

Sohail, A. 2017. Unity Editor Scripting (A kick-starter guide) – Part 2. Luettavissa: https://devcrew.io/2017/04/27/unity-editor-scripting-a-kick-starter-guide-part-2/. Luettu: 17.4.2018.

Spence, E. 2014. Unity Technologies Bulks Up On Utilities With Acquisition Of Playnomics. Luettavissa: https://www.forbes.com/sites/ewanspence/2014/04/18/unity-technologies-bulk-up-on-utilities-with-acquisition-of-playnomics/. Luettu: 17.4.2018.

Takahashi, D. 2012. Game developers, start your Unity 3D engines (interview). Luettavissa: https://venturebeat.com/2012/11/02/game-developers-start-your-unity-3d-engines-interview/. Luettu: 17.4.2018.

Tarini, M., Cignoni, P. & Montani, C. 2006. IEEE transactions on visualization and computer graphics, 12, 5. Luettavissa: http://preview.cgl.ucsf.edu/chimera/data/ambient-jul2014/Tarini_FinalVersionElec.pdf. Luettu: 2.5.2018.

Tufte, E. 2001. The Visual Display of Quantitative Information. Graphics Press. Cheshire.

UCI Machine Learning Repository. Iris Data Set. Luettavissa: https://archive.ics.uci.edu/ml/datasets/iris. Luettu: 20.4.2018.

UCLA Department of Epidemiology. Removal of the pump handle. Luettavissa: http://www.ph.ucla.edu/epi/snow/removal.html. Luettu: 20.4.2018.

Unity 2018a. Unity download archive. Luettavissa: https://unity3d.com/get-unity/download/archive. Luettu: 2.5.2018.

Unity 2018b. Unity store. Luettavissa: https://store.unity.com/. Luettu: 20.4.2018.

Unity 2018c. Assets, Objects and serialization. Luettavissa: https://unity3d.com/learn/tutorials/topics/best-practices/assets-objects-and-serialization. Luettu: 20.4.2018.

Unity 2018d. Common types of Assets. Luettavissa: https://docs.unity3d.com/Manual/AssetTypes.html. Luettu: 20.4.2018.

Unity 2018e. Learning the interface. Luettavissa: https://docs.unity3d.com/Manual/LearningtheInterface.html. Luettu: 4.5.2018.

Unity 2018f. Asset Workflow. Luettavissa: https://docs.unity3d.com/Manual/AssetWorkflow.html. Luettu: 20.4.2018.

Unity 2018g. How Do I Change The Supported Graphics APIs For My Game? Luettavissa: https://support.unity3d.com/hc/en-us/articles/209004446-How-do-I-change-the-supported-Graphics-APIs-for-my-game-. Luettu: 20.4.2018.

Unity 2018h. Graphics hardware capabilities and emulation. Luettavissa: https://docs.unity3d.com/Manual/GraphicsEmulation.html. Luettu: 20.4.2018.

Unity 2018i. A feature-rich and highly flexible editor. Luettavissa: https://unity3d.com/unity/editor. Luettu: 20.4.2018.

Unity 2018j. Lightmapping Quickstart. Luettavissa: https://docs.unity3d.com/Manual/Lightmapping.html. Luettu: 20.4.2018.

Unity 2018k. Audio Overview. Luettavissa: https://docs.unity3d.com/Manual/AudioOverview.html. Luettu: 20.4.2018.

Unity 2018l. Forum. Luettavissa: https://forum.unity.com/. Luettu: 20.4.2018.

Unity 2018m. World-class services for monetizing, making and improving your game. Luettavissa: https://unity3d.com/services. Luettu: 20.4.2018.

Unity 2018n. Build once, deploy anywhere. Luettavissa: https://unity3d.com/unity/features/multiplatform. Luettu: 20.4.2018.

Unity 2018o. Unity Remote. Luettavissa: https://docs.unity3d.com/Manual/UnityRemote5.html. Luettu: 20.4.2018.

Unity 2018p. MonoBehaviour. Luettavissa: https://docs.unity3d.com/ScriptReference/MonoBehaviour.html. Luettu: 20.4.2018.

Unity 2018q. UI. Luettavissa: https://docs.unity3d.com/Manual/UISystem.html. Luettu: 20.4.2018.

Unity 2018r. Writing vertex and fragment shaders. Luettavissa: https://docs.unity3d.com/Manual/SL-ShaderPrograms.html. Luettu: 20.4.2018.

Unity 2018s. GameObject. Luettavissa: https://docs.unity3d.com/Manual/class-GameObject.html. Luettu: 20.4.2018.

Unity 2018t. Compute shaders. Luettavissa: https://docs.unity3d.com/Manual/ComputeShaders.html. Luettu: 20.4.2018.

Unity 2018u. 3D formats. Luettavissa: https://docs.unity3d.com/Manual/3D-formats.html. Luettu: 20.4.2018.

Unity 2018v. JsonUtility. Luettavissa: https://docs.unity3d.com/ScriptReference/JsonUtility.html. Luettu: 20.4.2018.

Unity 2018w. JsonUtility.FromJson. Luettavissa: https://docs.unity3d.com/ScriptReference/JsonUtility.FromJson.html. Luettu: 20.4.2018.

Unity 2018x. WWW. Luettavissa: https://docs.unity3d.com/ScriptReference/WWW.html. Luettu: 20.4.2018.

Unity Asset Store 2018. Unity Technologies. Luettavissa: https://assetstore.unity.com/publishers/1. Luettu: 20.4.2018.

Ward, M., Grinstein, G. & Keim, D. 2010. Interactive Data Visualization. CRC Press. Boca Raton.

Ware, C. 2013. Information visualization. Elsevier. Waltham.

Wibble82 2015. How to have a scene with millions of gameObject? Luettavissa: https://answers.unity.com/questions/1113147/how-to-have-a-scene-with-millions-of-gameobject.html. Luettu: 17.4.2018.

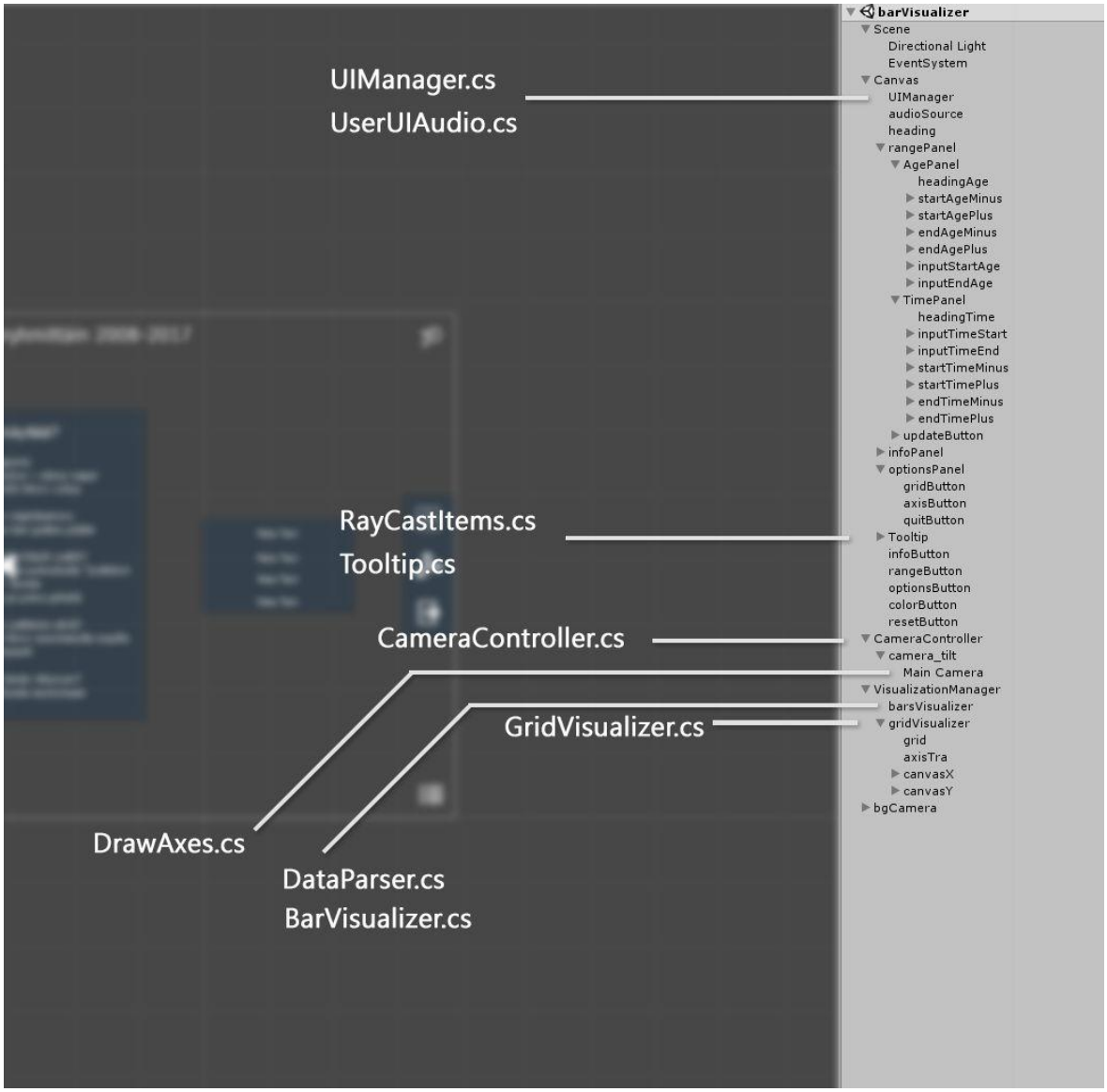
Wikimedia commons 2018. File:Snow-cholera-map-1.jpg. Luettavissa: https://commons.wikimedia.org/wiki/File:Snow-cholera-map-1.jpg. Luettu: 20.4.2018.

Wikipedia 2018a. Anscombe's quartet. Luettavissa: https://en.wikipedia.org/wiki/Anscombe's_quartet. Luettu: 20.4.2018.

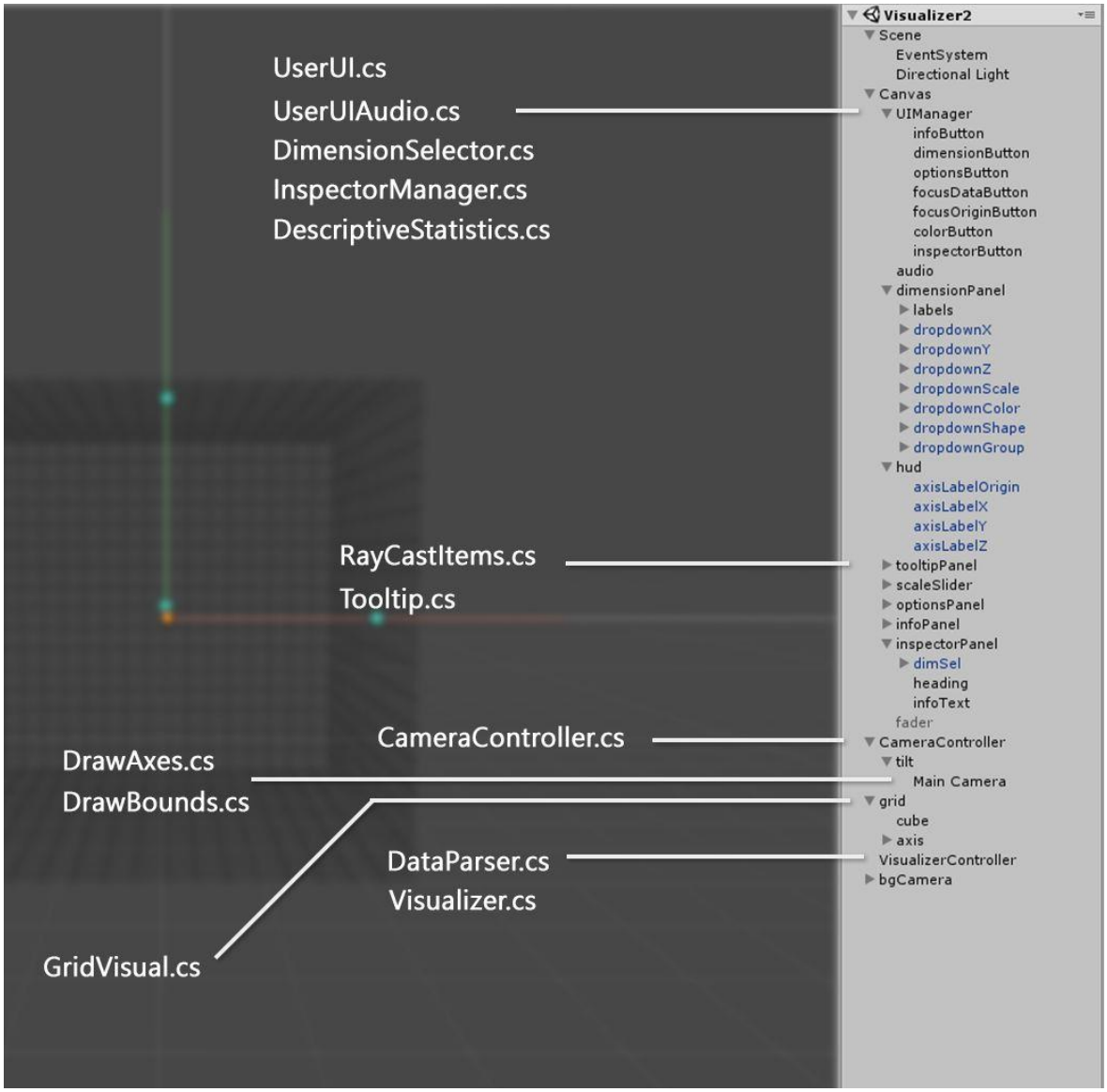
Wikipedia 2018b. Charles Joseph Minard. Luettavissa: https://en.wikipedia.org/wiki/Charles_Joseph_Minard. Luettu: 20.4.2018.

Liitteet

Liite 1. Pylväsdiagrammivisualisoinnin scene-rakenne



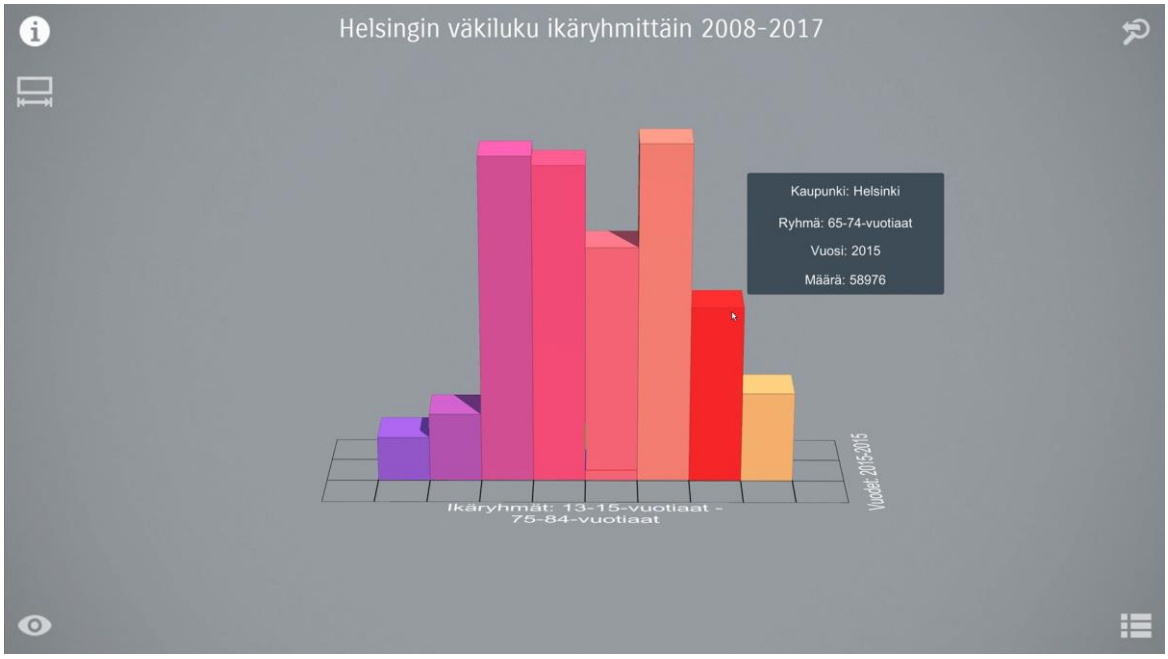
Liite 2. Parvikuviovisualisoinnin scene-rakenne



Parvikuviovisualisoinnin scene-rakenne ja MonoBehaviour-luokat.

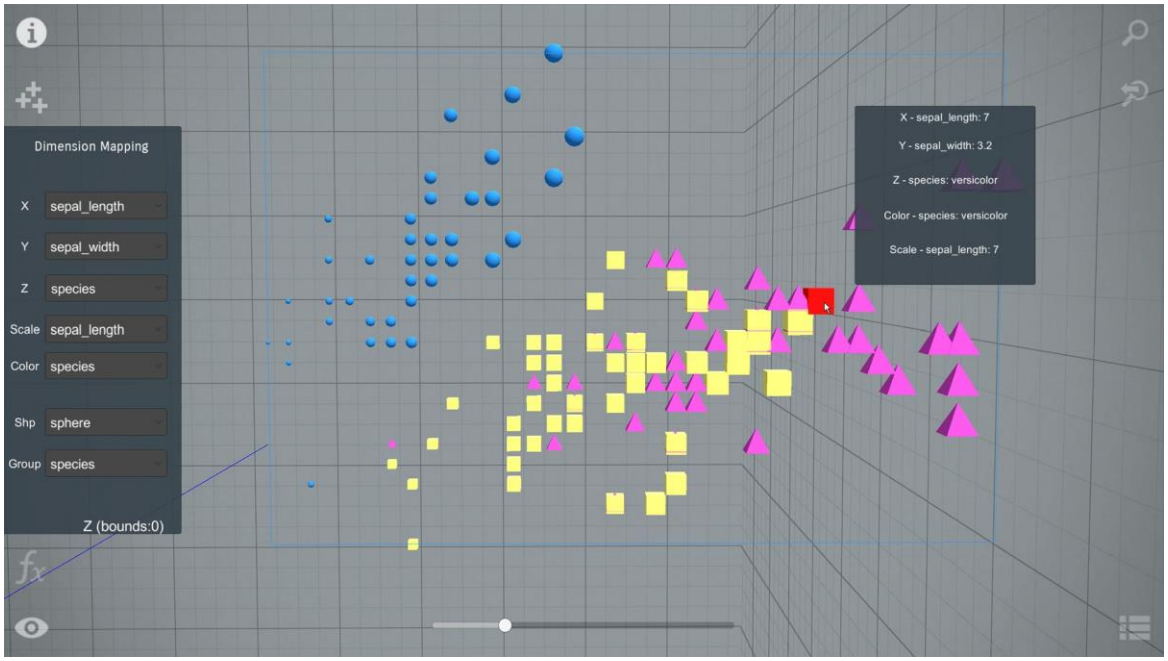
Liite 3. Pylväsdiagrammivisualisoinnin kuvia

Pylväsdiagrammin näytettävää dataa voi valita ikäryhmien ja vuosien perusteella.

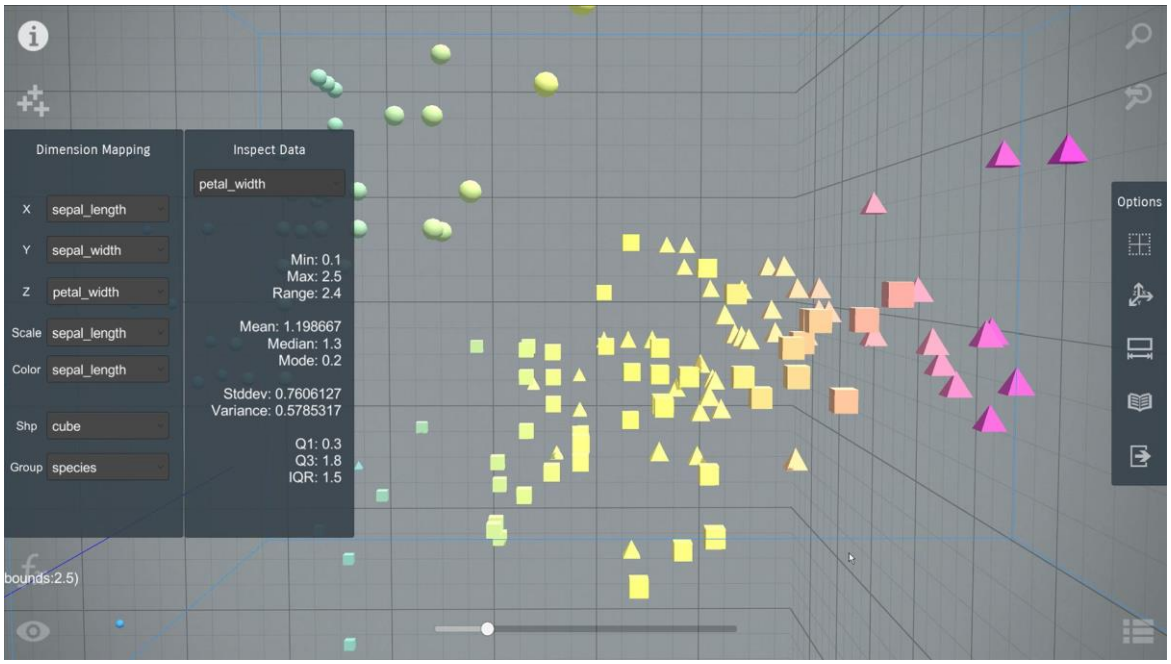


Pylväsdiagrammivisualisoinnin kameran kuvakulmaa ja etäisyyttä voi vaihtaa hiirtä käyttämällä.

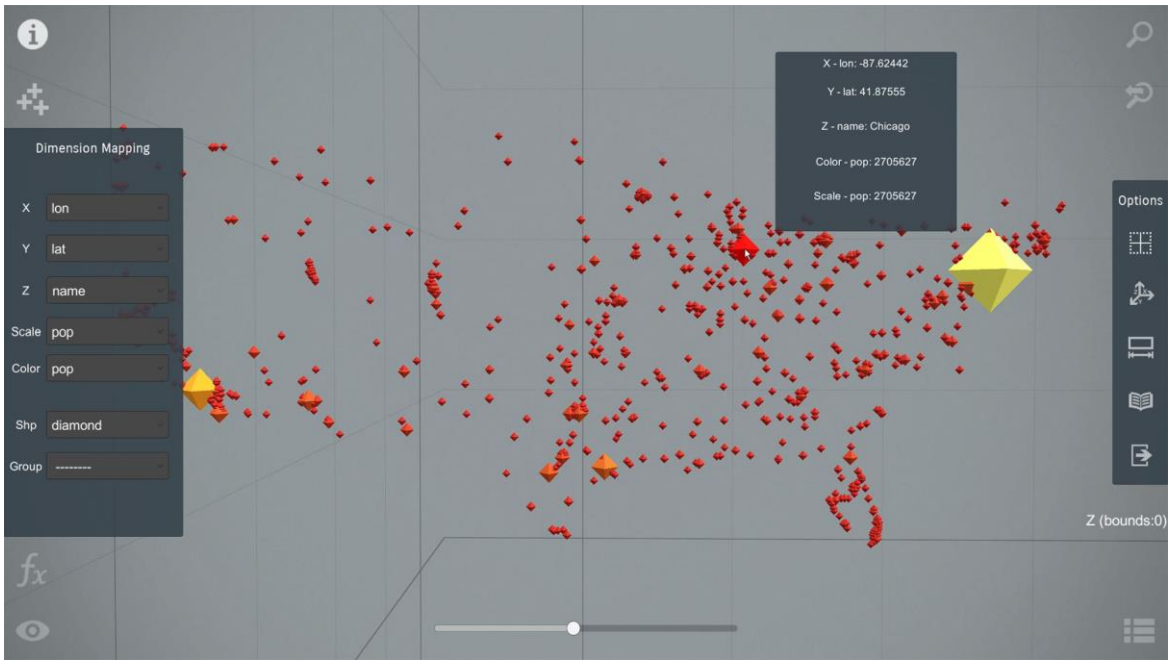
Liite 4. Parvikuviovisualisoinnin kuvia



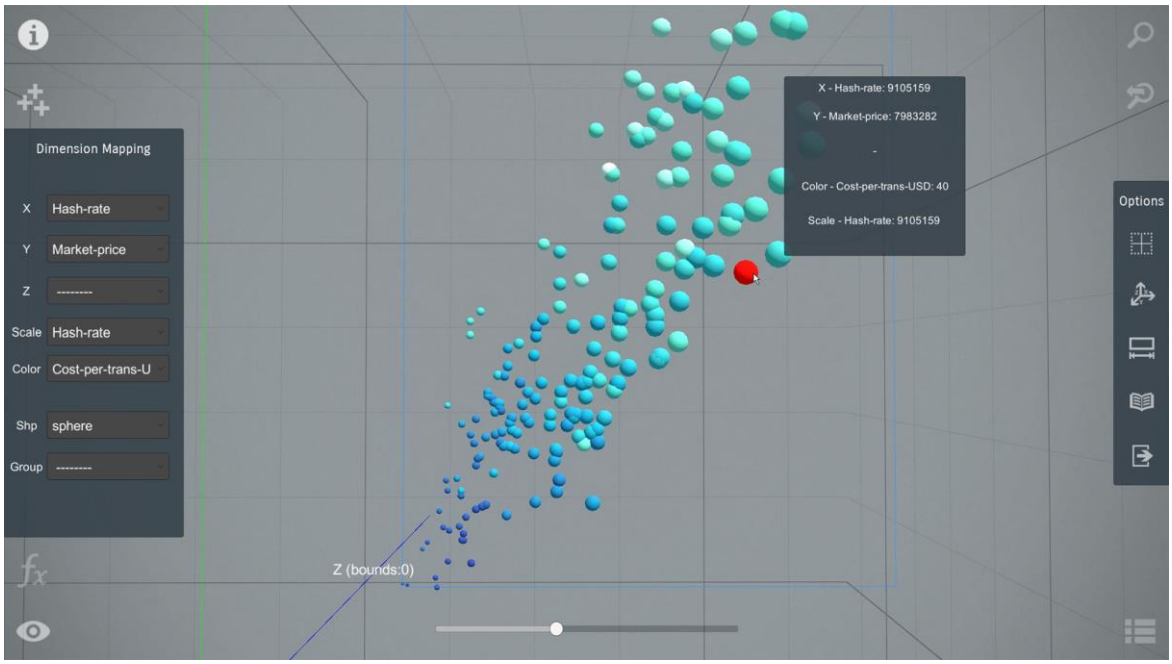
Parvikuviovisualisoinnin Dimension Mapping -paneelin asetuksia muuttamalla iris-data-setin eri ulottuvuuksia voidaan visualisoida eri tavoin. Kappaleiden muodot havainnollistavat Iiris-kukkien lajeja.



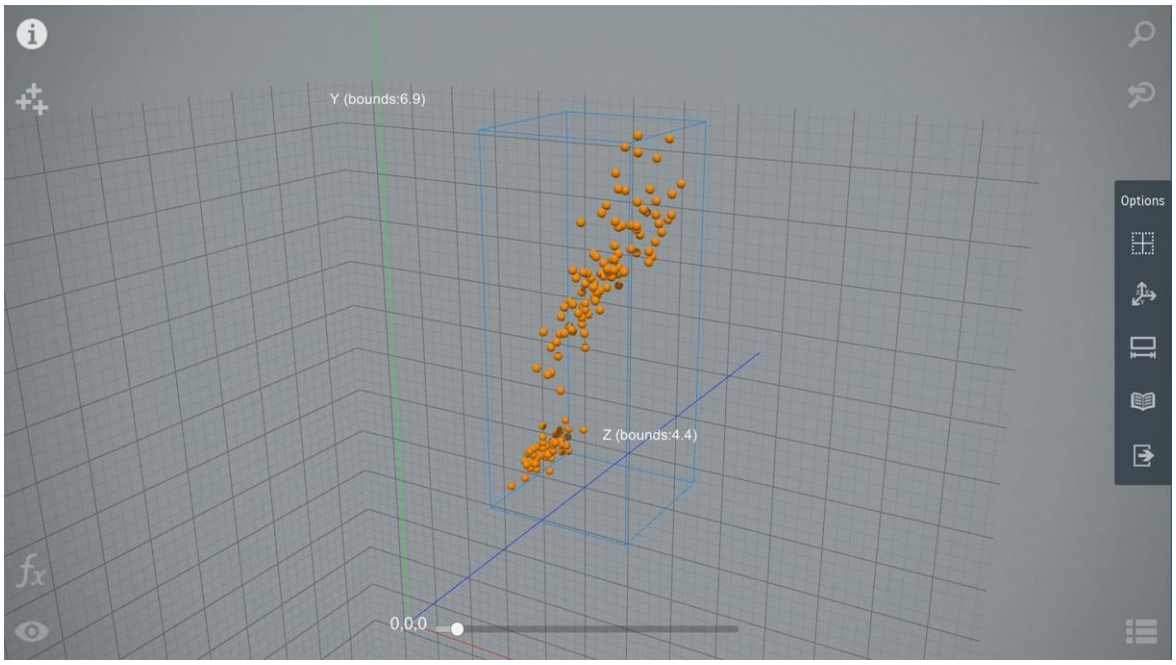
Inspect Data -paneelissa voidaan näyttää datan ulottuvuutta kuvailevia tunnuslukuja.



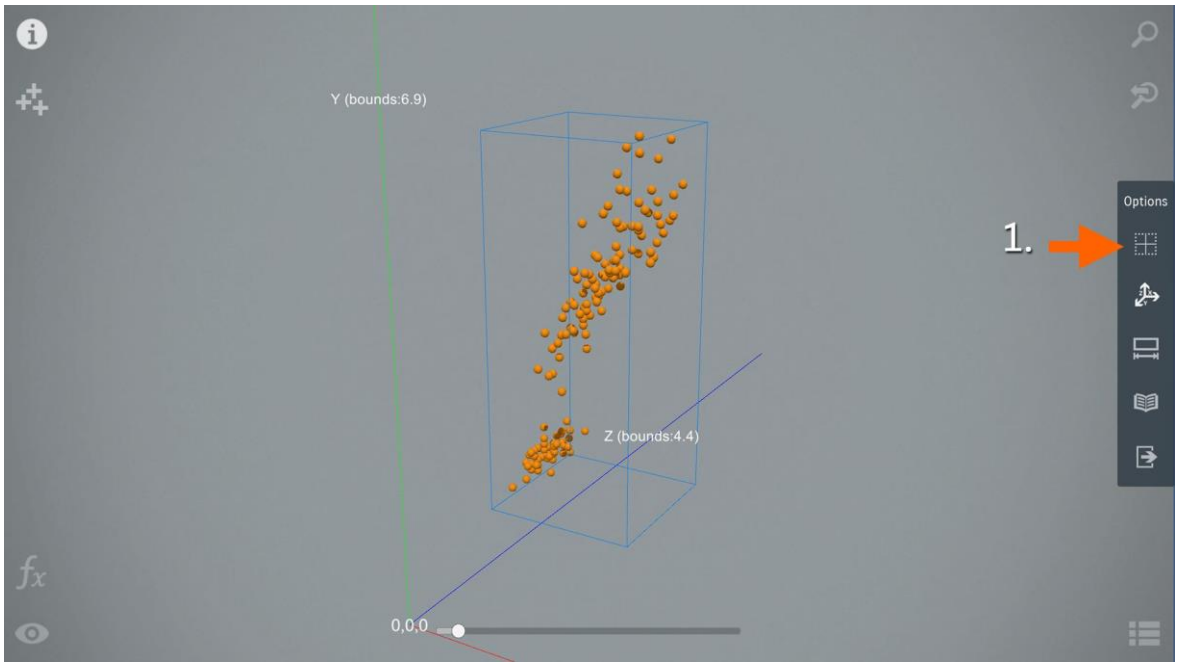
Datan lataajan koestamista. Amerikan yhdysvaltojen tuhannen suurimman kaupungin väkiluvut visualisoituna. Maantieteellinen sijainti on esitettynä x- ja y-akselilla. Väkiluvut on esitetty kappaleiden kokoa ja väriä käyttämällä.



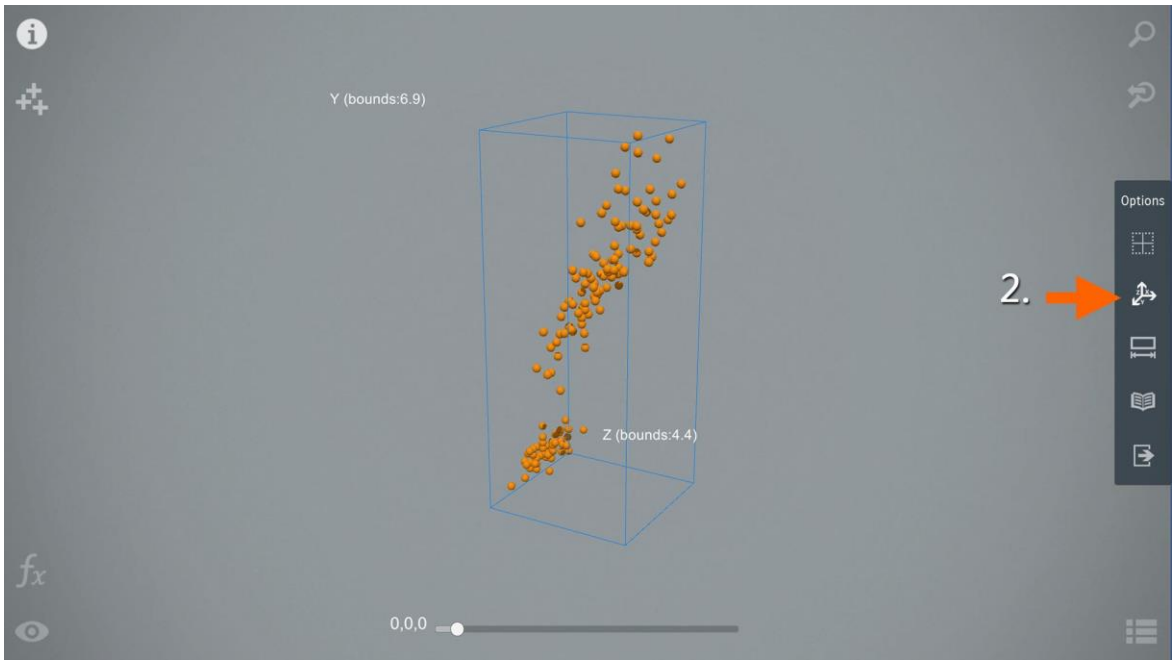
Bitcoin-kryptovaluutan louhintaa kuvaava datasetti visualisoituna. X-akselilla hash-rate ja y-akselilla markkinahinta.



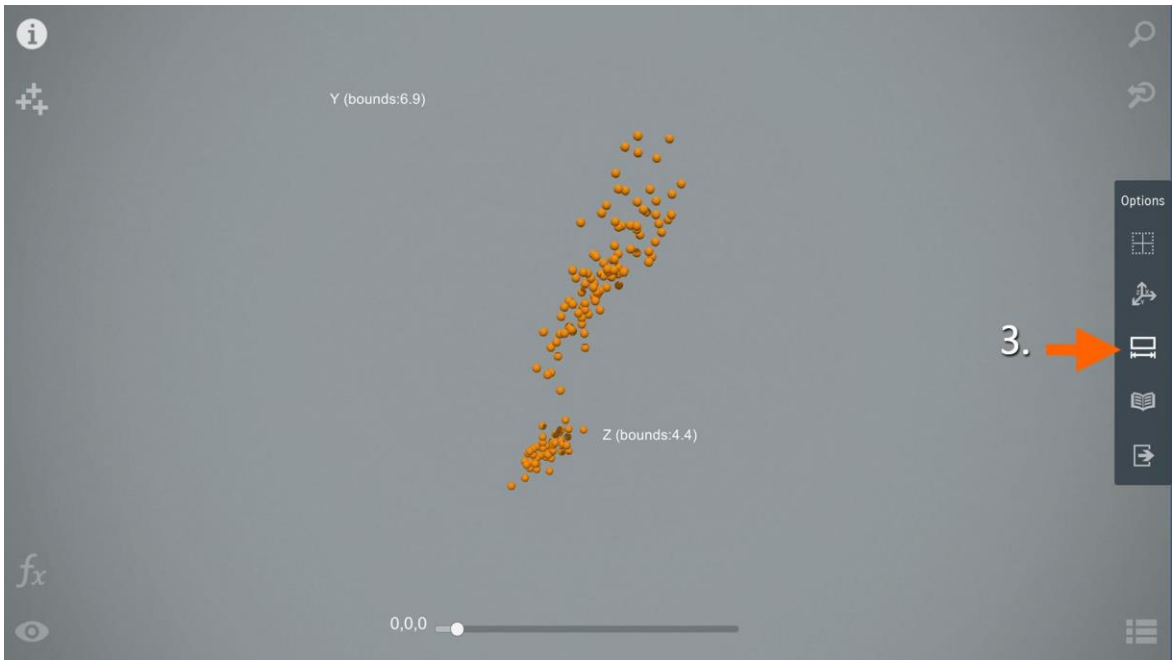
Parvikuviovisualisoinnin kuva-asetuksia voi muuttaa tarpeen mukaan;



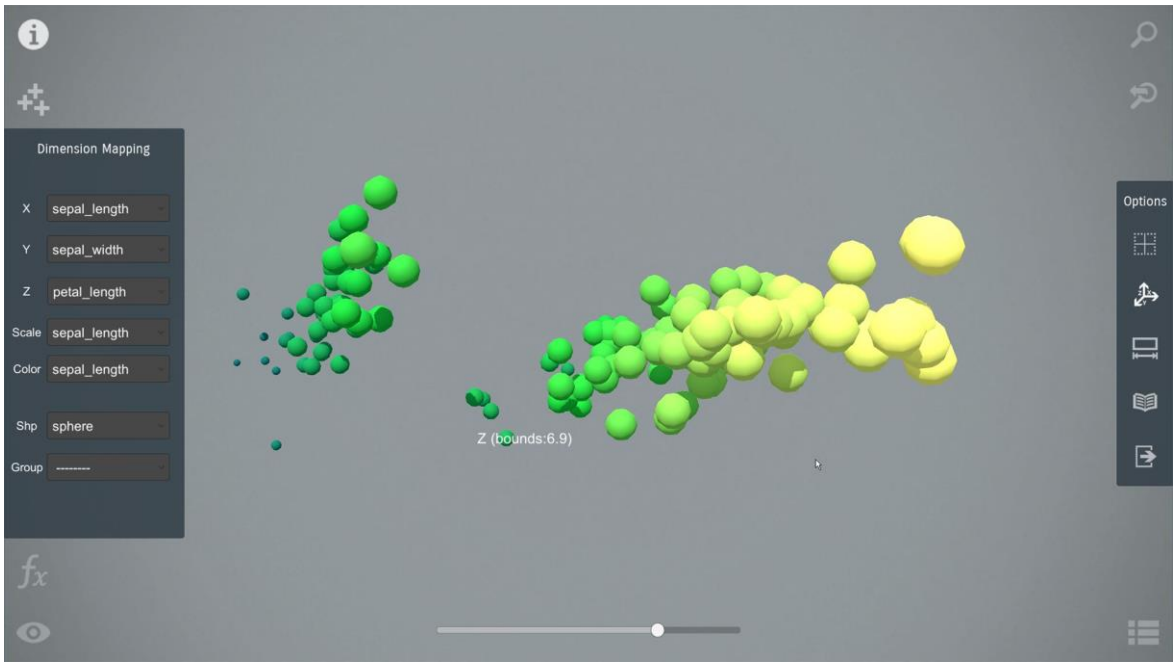
1. Koordinaattiruudukon voi kytkeä pois päältä tai päälle klikkaamalla ruudukko-ikonia.



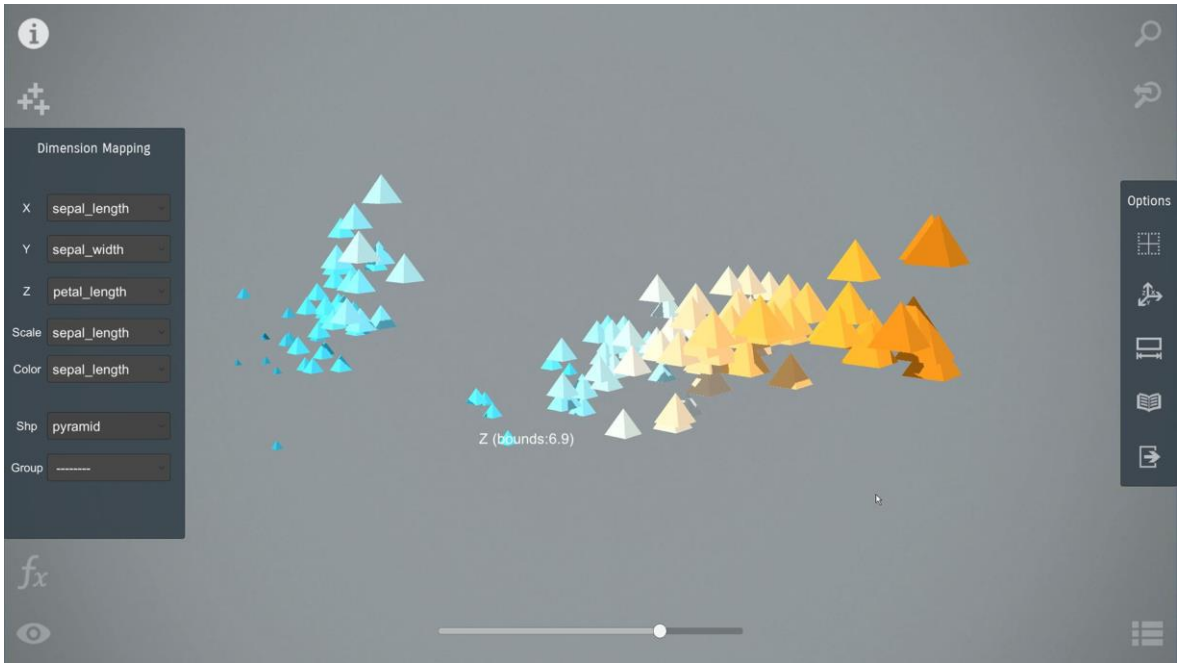
2. Koordinaattiakselin voi kytkeä pois päältä tai päälle klikkaamalla akselit-ikonia.



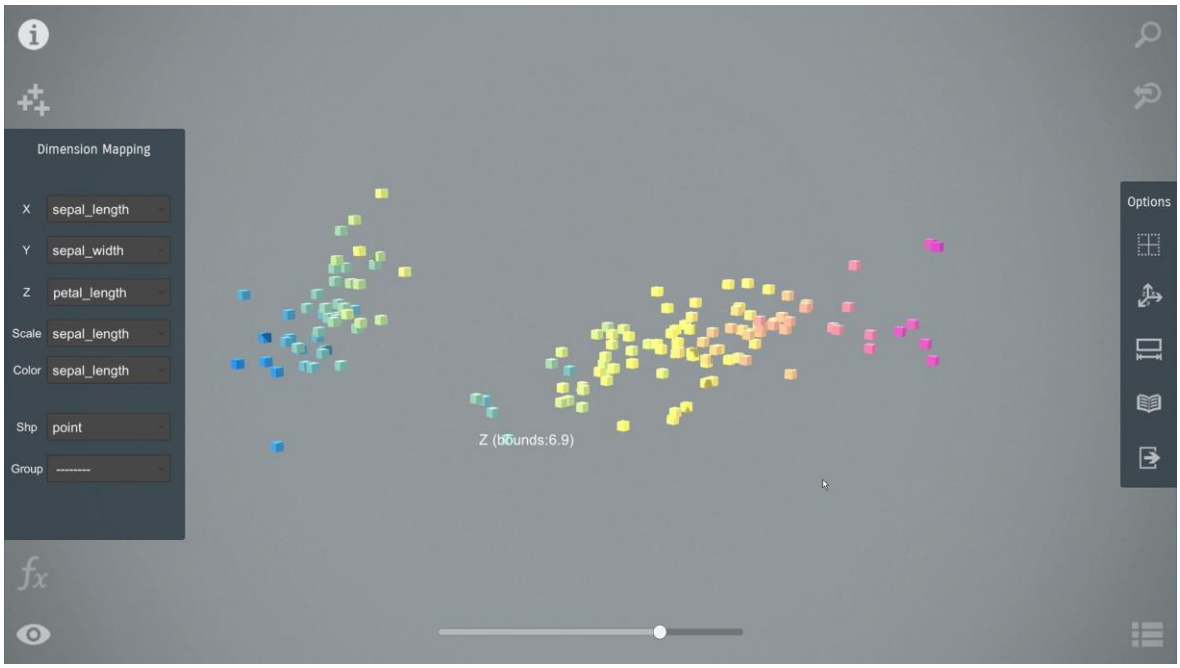
3. Datan ulottuvuuksia kuvaavan laatikon voi kytkeä pois päältä tai päälle klikkaamalla tilavuus-ikonia.



Datapisteitä voidaan esittää eri muodoilla ja väreillä. Tässä kuvassa käytössä vihreä-keltainen väriliuku ja pallot.



Tässä kuvassa käytössä sini-oranssi väriliuku ja pyramiidit.



Tässä kuvassa käytössä sini-kelta-violetti väriliuku ja piste-kappaleet, jotka säilyttävät kokonsa kameran etäisyydestä riippumatta.