LAPIN AMK
Lapland University of Applied Sciences

PROCESSING-OHJELMOINTIYMPÄRISTÖ KUVATAITEILIJAN TYÖKALUNA

Ohjelmoimalla tuotetut animaatiot

Ahvenainen Heikki

Opinnäytetyö

Kuvataiteen koulutus
Kuvataiteilija (AMK)

2020

LAPIN AMK
Lapland University of Applied Sciences

Opinnäytetyön tiivistelmä

Kuvataiteen koulutus
Kuvataiteilija (AMK)

Tekijä	Heikki Ahvenainen	Vuosi	2020
Ohjaaja	Eija Rajalin		
Työn nimi	Processing-ohjelmointiympäristö kuvataiteilijan työkaluna, Ohjelmoimalla tuotetut animaatiot		
Sivu- ja liitesivumäärä	32 + 24		

Tämän opinnäytetyön tavoitteena oli tutkia, miten ohjelmoimalla voi tuottaa kuvia ja animaatioita. Toisena päämääränä oli selvittää, minkä takia nimenomaan kuvataiteilijan kannattaisi käyttää ohjelmointia työkalunaan. Tutkimuksessa tutustuttiin erityisesti Processing-ohjelmointiympäristöön. Työn teososuudessa toteutettiin kymmenen animaatioteosta.

Opinnäytetyön tarkoituksena oli kehittää omaa taiteellista työskentelyäni. Kyseessä oli siis taiteellinen tutkimus, joka painottui kehittämisprosessini kuvailemiseen sekä kehittämistyöni seurauksena syntyneen aineiston analysointiin. Luonnoksia ja valmiita teoksia analysoitiin formalistisen kuva-analyysin keinoin. Kuva-analyysin avulla pystyin siis määrittelemään, minkälaisia kuvia ja animaatioita ohjelmoimalla pystyy tuottamaan.

Opinnäytetyön tulosten mukaan ohjelmoimalla voi tuottaa kuvia sekä visuaalisista lähtökohdista vähäisilläkin tiedoilla että loogisesti haastavammista lähtökohdista kattavammilla taidoilla. Johtopäätösteni mukaan ohjelmointia kannattaa käyttää työvälineenä varsinkin silloin, kun haluaa olla muiden tuottamista ohjelmista riippumaton ja digitaalisesti vapaampi toimimaan, sekä silloin, kun haluaa tuottaa teoksia, joiden tuottaminen ei muilla tekniikoilla olisi mahdollista.

Avainsanat	ohjelmointi, kuva, animaatio, taide
Muita tietoja	Työhön liittyy kymmenen animaatioteosta.

LAPIN AMK
Lapland University of Applied Sciences

Abstract of Thesis

Degree Programme in Visual Arts
Bachelor of Culture and Arts

Author Heikki Ahvenainen Year 2020
Supervisor Eija Rajalin
Subject of thesis Processing Development Environment as a Tool of an Artist, Animations Produced through Programming
Number of pages 32 + 24

The objective of this thesis was to study how images and animations can be created with programming. Another goal was to find out why artists specifically should use programming as their tool. The study was concentrated on the Processing Development Environment. In the functional part of the study, ten animations were created.

The authors's artistic work was elaborated in this thesis. The study concentrated on describing the development process and analysing the material created as a result in this process. The drafts and the finished pieces were analyzed with formalistic analysis. Using the formalistic analysis, it was possible to define what kind of images and animations can be produced with programming.

According to the results images and animations can be created with programming by both artists who have little programming skills and those who have more advanced skills and more challenging starting points. The thesis concluded that it is beneficial to use programming as a tool, especially when the artist wants to be independent of commercial software and free to work digitally. Programming is also suitable for creating works of art that are not possible to create using some other techniques.

Key words programming, image, animation, art
Special remarks The thesis includes ten animations.

SISÄLLYS

1 JOHDANTO

1.1 Lähtökohdat

Ohjelmointi on huomisen lukutaito, ja sitä tarvitaan ennen pitkää lähes kaikissa työtehtävissä. Vuonna 2016 ohjelmoinnin opettaminen aloitettiin myös peruskouluissa, ja jotkut uskovat ohjelmistoalan olevan jopa kansantaloutemme pelastaja. Ohjelmoinnista puhuttaessa onkin alettu käyttää niinkin mahtipontisia sanoja kuin oikeus ja velvollisuus, ja monien mielestä kyseessä on jopa tasa-arvoasia. (Saariketo 2015.)

Opinnäytetyöni käsittelee tätä tulevaisuuden kansalaistaitoa kuvataiteilijan näkökulmasta. Tutustun erityisesti Processing-ohjelmointiympäristöön ja tutkin toiminnallisesti, miten kuvataiteilija voi tuottaa kuvia ja animaatioita ohjelmoimalla. Tutkimustehtävänäni on vastata seuraaviin tutkimuskysymyksiin:

- Miten ohjelmoimalla voi tuottaa kuvia ja animaatioita?
- Miksi tuottaa kuvia ja animaatioita ohjelmoimalla?

Opinnäytetyön teoriaosuudessa määrittelen, mitä ohjelmointi ja ohjelmoimalla tuotettu taide oikeastaan ovat ja käyn läpi taidemuodon historiaa sekä sen sijaintia kuvataiteen kentällä. Määrittelen teoriaosuudessa myös Processing-ohjelmointiympäristön toimintaperiaatteet ja tarkastelen sen tarjoamia mahdollisuuksia kuvataiteilijalle.

Henkilökohtaisella tasolla ohjelmoimalla tuotettava taide on ennestään täysin tuntematonta aluetta, ja olenkin aikaisemmin tuntenut ohjelmointia kohtaan jopa jonkinasteista vastenmielisyyttä. Valitsin kuitenkin aiheekseni ohjelmoimalla tuotettavan taiteen juuri sen takia, että haluan haastaa itseni sekä laajentaa osaamisaluettani tutustumalla itselleni täysin uudenlaiseen tapaan tuottaa kuvia ja animaatioita. Toisaalta olen kiinnostunut aiheesta myös siksi, että se yhdistää useimmiten toistensa vastakohtina nähdyt luovan taiteellisen työskentelyn sekä loogisen päättelyn. Koen ohjelmoimalla tuotettavan taiteen myös puhtaasti visuaalisen tyylinsä takia puoleensavetävänä, ja olen myös kiinnostunut

matemaattisia funktioita sisältävän ohjelmointikielen mahdollisista yhteyksistä luonnossa ja abstrakteissa maalauksissa piilevään matematiikkaan.

1.2 Työn tavoite ja menetelmät

Opinnäytetyön tarkoituksena on kehittää omaa taiteellista työskentelyäni. Kyseessä on siis taiteellinen tutkimus, joka painottuu kehittämisprosessini kuvailemiseen sekä kehittämistyöni seurauksena syntyneen aineiston analysointiin. Käytännössä kehittämistyöni noudattaa spiraalimallia, jossa työn suunnittelua seuraavat toiminta, havainto sekä reflektointi. Tämän jälkeen vaiheet toistuvat ja arvioin edellisen vaiheen tulokset aina uudelleen ja uudelleen. (Salonen 2013, 15.) Tutkimuksen aikana pyrin myös havainnoimaan omia työtapojani sekä pohtimaan, miten ohjelmoimalla luominen eroaa muista taiteen tekniikoista.

Luonnoksia ja valmiita teoksia analysoidessani käytän analyysimenetelmänä formalistista kuva-analyysia. Formalistinen analyysimenetelmä sopii abstraktien animaatioiden tarkasteluun, koska se keskittyy kuvan muotoihin sekä rakenteellisiin ominaisuuksiin. Sen avulla saan siis selville, minkälaisia kuvia ja animaatioita ohjelmoimalla pystyy tuottamaan. Koska kyseessä on taiteellinen tutkimus, tietoperustan analysointi ei perustu mihinkään perinteiseen analyysimenetelmään, vaan se koostuu lähinnä omien ajatusteni ja teoriapohjan välisestä vuoropuhelusta syntyneistä ideoista ja tulkinnoista.

Tutkimuksen tulokset eli vastaukset tutkimuskysymyksiini sekä teososuuden tuotokset syntyvät oman kehittämistyöni seurauksena sekä teoriaosuudessa läpikäytyyn tietoperustaan pohjautuen. Saatuani vastaukset tutkimuskysymyksiini pohdin myös, ovatko vastaukseni yleistettävissä vai ainoastaan omiin kokemuksiini perustuvia henkilökohtaisia näkemyksiä.

2 OHJELMOINTI KUVATAITEILIJAN TYÖKALUNA

2.1 Mitä on ohjelmointi?

Ohjelmointi on sellaisen tietokoneohjelman suunnittelua ja rakentamista, joka suorittaa käynnistettäessä jonkin tietyn tehtävän. Useimmiten tämä rakentaminen toteutetaan kirjoittamalla lähdekoodi eli erilaisia toimintaohjeita jollakin ohjelmointikielellä. Ohjelmointikieli tarkoittaa sellaista kieltä, joka sisältää joukon sääntöjä, joiden avulla voidaan tuottaa erilaisia lopputuloksia. Kun valmis ohjelma käynnistetään, ohjelman lähdekoodi muunnetaan tietokoneen ymmärtämäksi konekieleksi kääntäjällä. Yleensä kääntäjä on integroituna samaan ohjelmaan, jossa varsinainen ohjelmointi eli lähdekoodin kirjoittaminenkin tapahtuu. (Gookin 2014, 23.)

Vaikka lähdekoodia voi vallan hyvin kirjoittaa perinteisissä tekstieditoreissakin, on se nykypäivänä kuitenkin tapana toteuttaa niin sanotuissa ohjelmointiympäristöissä. Tällöin ohjelmointi tapahtuu juuri siihen tarkoitukseen luodussa ohjelmassa, joka yhdistää kaikki ohjelmointiin tarvittavat työkalut yhteen pakettiin. Yleensä pakettiin kuuluu tekstieditorin lisäksi ainakin kääntäjä sekä virheenjäljitin eli debuggeri. Tekstieditorissa siis kirjoitetaan varsinainen lähdekoodi ja kääntäjällä käännetään kieli konekieleksi. Debuggeri sen sijaan toimii ohjelmoijan apuna virheiden jäljittämisessä. Käyttäessään ohjelmointiympäristöä ohjelmoija voi täten keskittyä varsinaisen ohjelman logiikan rakentamiseen, kun ohjelmointiympäristö automatisoi muut ohjelmointiin ja lopullisen ohjelman luontiin liittyvät järjestelyt. Usein ohjelmointiympäristöt tarjoavat käyttäjälle myös mahdollisuuden laajentaa ympäristön toimintamahdollisuuksia sallimalla erilaisten lisäosien ja kirjastomoduulien asentamisen. (Rouse 2016.)

Käytännössä ohjelmointi koostuu lähdekoodin kirjoittamisen lisäksi myös ohjelman testaamisesta ja virheiden korjaamisesta. Ohjelmointi on siis prosessi, jossa ensin kirjoitetaan jollakin ohjelmointikielellä sääntöjä sekä yhdistellään näitä sääntöjä loogisiksi kokonaisuuksiksi. Tämän jälkeen korjataan virheet ja hiotaan toiminnallisuutta kehittämällä uusia sääntöjä. (Gookin 2014, 23–24.)

2.2 Mitä on ohjelmoimalla tuotettu taide?

Ohjelmoimalla tuotettu taide tarkoittaa sellaista taidetta, joka on luotu käyttämällä jotain ohjelmointikieltä ilmaisullisiin ja taiteellisiin tarkoituksiin. Kun ohjelmoija siis perinteisesti toimii käytännöllisesti ja tiettyjä ongelmia ratkaisten, antaa ohjelmoimalla luotu taide mahdollisuuden toimia luovasti ja käytännöllisyydestä välittämättä. (Wu 2020) Yksinkertaistettuna ohjelmoimalla tuotettu taide on muotojen, värien ja liikkeiden luomista kirjoittamalla tietokoneelle käskyjä (FLOSS Manuals 2010).

Terminä ohjelmoimalla tuotettu taide ei ole kovinkaan usein käytetty, vaan usein puhutaankin esimerkiksi luovasta ohjelmoinnista, kooditaiteesta, generatiivisesta taiteesta tai algoritmisesta taiteesta. Käytän kuitenkin tässä opinnäytetyössä termiä "ohjelmoimalla tuotettu taide", koska se vastaa mielestäni parhaiten omaa tekemistäni. Historiallisesti katsoen ohjelmoimalla tuotettu taide luetaan osaksi generatiivista taidetta (Pearson 2011, 6) ja käsittelenkin tätä laajempaa kokonaisuutta tarkemmin luvussa 2.4.

Vaikka ohjelmoimalla tuotettua taidetta voidaan käytännössä luoda kaikilla ohjelmointikielillä ja -ohjelmilla, tuotetaan sitä useimmiten jossakin erityisesti siihen tarkoitukseen kehitetyssä ohjelmassa. Taiteen luominen tapahtuu kuitenkin samalla tavoin kuin mikä tahansa muukin ohjelmointityö eli ohjelmointikieltä kirjoittaen. Lopullinen taideteos ei kuitenkaan ole koodi itsessään, vaan kirjoitetun ohjelman käynnistyessä syntyvä animaatio tai kuva. Toisaalta teos voi olla myös interaktiivinen tai jopa jollain tavalla fyysinen. Tällöin kyseessä saattaa olla esimerkiksi ohjelmoitu piirustusrobotti tai tietyn logiikan mukaan vilkkuvista valoista koostuva installaatio. Periaatteessa ohjelmoimalla tuotetun taiteen mahdollisuudet ovatkin rajattomat ja lopullinen taideteos voi olla lähes mitä tahansa. (Dufva 2020.) Tässä opinnäytetyössä keskitytään kuitenkin ainoastaan kuvien ja animaatioiden tuottamiseen ohjelmoimalla.

Erityisesti visuaalisen taiteen luomiseen sopivia ohjelmointikieliä ovat esimerkiksi Max, openFrameworks, vvvv sekä Processing. Näistä jokaisen verkkosivuilta on ladattavissa paketti, jonka asentaessaan käyttäjä saa käyttöönsä kyseisen kielen ohjelmointiympäristön. Jokaisessa ympäristössä on omat vahvuutensa, mutta

Processingin valttina on sen samanaikainen helppokäyttöisyys sekä potentiaali tuottaa ammattimaisia tuloksia (Pearson 2011, 14).

2.3 Ohjelmoimalla tuotetun taiteen historia

Ohjelmoimalla tuotetun taiteen historiaa on jossain määrin vaikea määrittää, koska teokset ja työkalut ovat niin vahvasti virtuaalisia, että ne lopulta vain katoavat, eivätkä välttämättä jää mihinkään talteen samalla tavoin kuin fyysiset taideteokset (Neal 2018). Ihmisen tarve itseilmaisuun tuntien voidaan kuitenkin olettaa, että ohjelmointia on käytetty taiteen tuottamiseen lähes yhtä kauan kuin varsinaiseen ohjelmointiinkin. Varmuudella voidaankin sanoa, että jo vuonna 1956 taiteilijat käyttivät Yhdysvalloissa ensimmäistä kaupalliseen tarkoitukseen valmistettua UNIVAC-tietokonetta nimenomaan itseilmaisullisiin tarkoituksiin (Greenberg 2007, 13).

Koska kyseessä on niin laaja kenttä, ohjelmoimalla tuotetun taiteen historiaa ei ole vielä oikeastaan koottu minnekään. Nealin mielestä historiaa voisi kuitenkin lähteä purkamaan esimerkiksi ohjelmoimalla tuotetun taiteen työkaluista, teoksista, tekijöistä tai yhteisöistä. (Neal 2018.) Työkalut ovat siinä mielessä hyvä lähtökohta, että niiden historiaa ja julkaisuvuosia löytyy jonkin verran dokumentoituna. Verkkoa selaamalla selviääkin, että suurin osa ohjelmoimalla tuotettavan taiteen luomiseen keskittyvistä ohjelmista on julkaistu 2000-luvun taitteessa tai sen jälkeen. Nealin mielestä ohjelmien historiaa on kuitenkin hyvin hankalaa kartoittaa kattavasti, koska osasta ohjelmista on saatavissa niin vähän tietoja. Myös teosten ja tekijöiden menneisyyden hahmottaminen on tietojen katoamisen takia vaikeaa, joten Neal ehdottaakin, että ohjelmoimalla tuotetun taiteen historian kirjoittaminen täytyisi olla jollain tavalla yhteisöllinen projekti. (Neal 2018.)

Jos joka tapauksessa mennään ajassa hieman taaksepäin ja oletetaan, että ohjelmoimalla tuotettavan taiteen luomiseen tarvittavia ohjelmia on alettu julkaista siis vuosituhannen vaihteessa, voidaan myös otaksua, että jotain on tarvinnut olla sitä ennen. Vaikka kirjoitettua faktaa asiasta ei tietääkseni ole, uskon, että ohjelmoimalla tuotetun taiteen edeltäjänä voidaan pitää 80- ja 90-lukujen alakulttuuria vallinnutta demoskeneä. Kyseisen kulttuurin piirissä

ulosmitattiin sen ajan tietokoneiden potentiaali ja luotiin näyttäviä visuaaleja sekä musiikkia nimenomaan ohjelmoimalla (Reunanen 2010, 1).

2.4 Generatiivinen taide

Ohjelmoimalla tuotettu taide sijoitetaan taiteen kentällä yleensä osaksi generatiivista taidetta. Tyypillisesti termejä käytetään myös sekaisin ja generatiivisesta taiteesta puhuttaessa tarkoitetaankin usein nimenomaan ohjelmoimalla tuotettua taidetta. Todellisuudessa generatiivinen taide on siis kuitenkin kattavampi yleiskäsite, ja Galanterin (2003) mukaan generatiivista taidetta onkin kaikki sellainen taide, jossa taiteilija tuottaa teoksia käyttäen apunaan jonkinlaista autonomista systeemiä. Määritelmänä generatiivinen taide ei siis viittaa mihinkään tiettyyn teknologiaan vaan on laajempi taiteen tuotantotapoihin liittyvä käsite. Käsitteen alle voidaankin sijoittaa visuaalisen taiteen lisäksi myös esimerkiksi musiikkia, arkkitehtuuria ja kirjallisuutta. (Galanter 2003.)

Generatiivisen taiteen historian voidaan hieman yllättäen sanoa olevan yhtä pitkä kuin taiteen ylipäätänsä, eikä generatiivisen taiteen tekemiseen itse asiassa tarvita edes tietokonetta. Näin on siksi, että jo niin sanotut primitiiviset kansat käyttivät toistuvia geometrisia kuvioita sekä symmetriaa esimerkiksi tekstiileissään ja juuri tämä symmetria on Galanterin mukaan merkki autonomisen systeemin tuottamasta taiteesta. (Galanter 2003.) Kun siis generatiivista taidetta lähestytään historiallisesta näkökulmasta, laajeneekin perspektiivi pelkästä informaatioajan kooditaiteesta huomattavasti avarammaksi. Pearson on tästä sikäli samaa mieltä, että toteaa taidemuodon konseptin olleen olemassa jo pitkään. Hän kuitenkin lisää, että varsinaisesti termiä generatiivinen taide on käytetty vasta 1960-luvulta alkaen ja että sen takia generatiivisen taiteen historia voidaan käytännössä laskea vain vuosikymmenissä. (Pearson 2011, 7.)

Koska generatiivista taidetta luodaan nykypäivänä pääosin loogisesti käyttäytyvällä ohjelmointikielellä, on Pearsonin mukaan aiheellista pohtia, kuinka tällä tavoin tuotettua sisältöä voi edes kutsua taiteeksi. Taiteen määritelmä on tietysti jo itsessään vahvasti jäsentymätön, mutta onhan ohjelmoimalla luotuja taideteoksia vaikea kuvitella inhimillisesti koskettaviksi. (Pearson 2011, xxvii.)

Yksityiskohtaisemmin tarkasteltuna nähdään kuitenkin myös ihmisessä sekä luonnossa ylipäätään hyvinkin loogisesti toimivia kokonaisuuksia, eikä tietokoneilla luotu generatiivinen taidekaan ole loppujen lopuksi vain loogista ohjelmointia. Teosten pohjalla olevat autonomiset systeemithän luovat kuitenkin aina loppujen lopuksi luontoa jäljittelevää epäjärjestystä ja kaaosta. Oleellista onkin sisällyttää teoksiin juuri oikea määrä epäjärjestystä sekä järjestystä. (Pearson 2011, xxxii.)

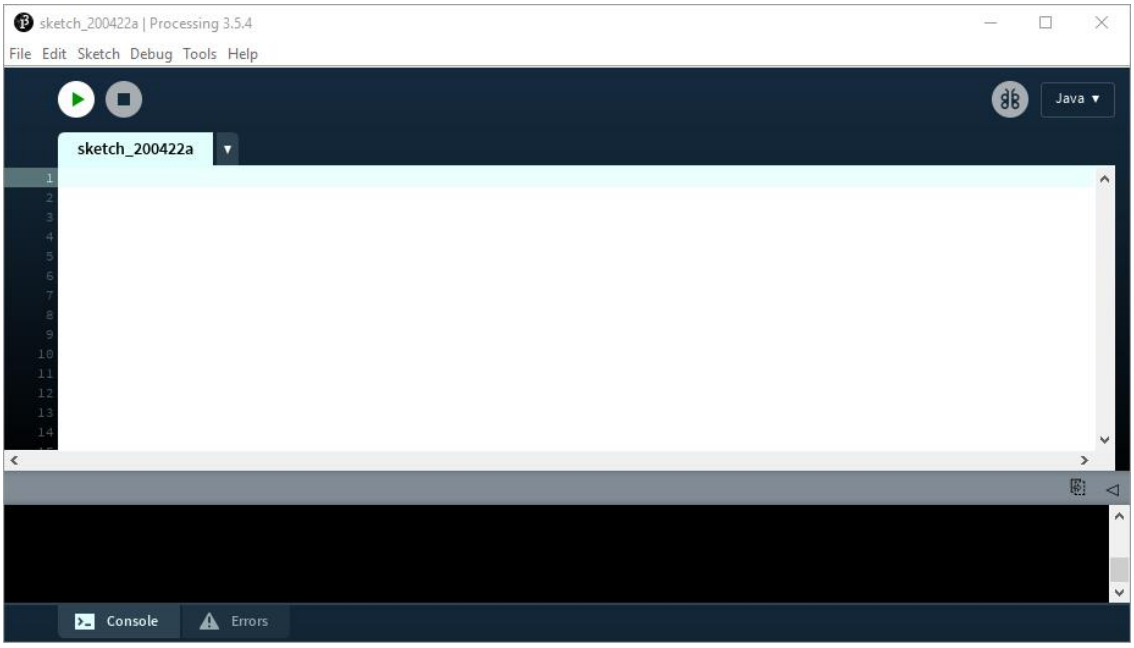
Generatiivisella taiteella on vahva side myös erääseen toiseen logiikan osa-alueeseen. Taidemuodon tuottamiseen tarvittavat työkalut tai tekemisen tavat nimittäin perustuvat aina pohjimmiltaan matemaattisiin algoritmeihin. (Pearson 2011, 9.) Algoritmien ja modernin taiteen välillä kiinnostavin linkki on sen sijaan Jackson Pollock, jonka maailmankuulujen roiskemaalauksien on jälkikäteen todistettu koostuvan algoritmeihin perustuvista fraktaalikuvioista. Pollockin teosten ”fraktaalipitoisuus” on myös ajan myötä kasvanut. (Galanter 2003.) On siis mielenkiintoista huomata, kuinka ilmeisen logiikasta ja matematiikasta vapaa työskentelykin on kuitenkin pohjimmiltaan nimenomaan matematiikkaan perustuvaa toimintaa.

3 PROCESSING-OHJELMOINTIYMPÄRISTÖ

3.1 Processing-ohjelmointiympäristön toimintaperiaate

Processing on Casey Reasin ja Ben Fryn vuonna 2001 kehittämä ohjelmointiympäristö taiteilijoille. Sen alkuperäisenä tarkoituksena on ollut toimia ohjelmoinnin opetustarkoituksessa visuaalisesti orientoituneille ihmisille, ja se on suunniteltu lähtökohtaisesti niin, että erilaisten graafisten elementtien kuten viivojen, ellipsien ja nelikulmioiden piirtäminen olisi mahdollisimman helppoa. Kyseessä on vapaan lähdekoodin ohjelma, jonka voi ladata ilmaiseksi osoitteesta www.processing.org/download. Kontekstista riippuen sanalla Processing voidaan tarkoittaa joko itse ohjelmaa, ohjelmointiympäristöä tai ohjelmointikieltä. Processing perustuu Java-ohjelmointikieleen, ja siitä on saatavissa versiot Mac-, Windows- ja Linux-käyttöjärjestelmiin sekä nykyään myös Android-laitteille. (Processing 2020a.) Ohjelmointiympäristön viimeisin vakaa versio on kirjoitushetkellä julkaistu tammikuussa 2020.

Käytännössä Processingia käytetään kirjoittamalla koodia eli ohjelmointikieltä Processing-ohjelman editoriin. Editorin yläpuolella sijaitset välilehdet, työkalurivi sekä päävalikko. Editorin alapuolelta löytyvät viestialue ja konsoli. (Reas & Fry 2007, 8.) Ohjelman käyttöliittymä on kuvattu kuvassa 1.



Kuva 1. Processing-käyttöliittymä

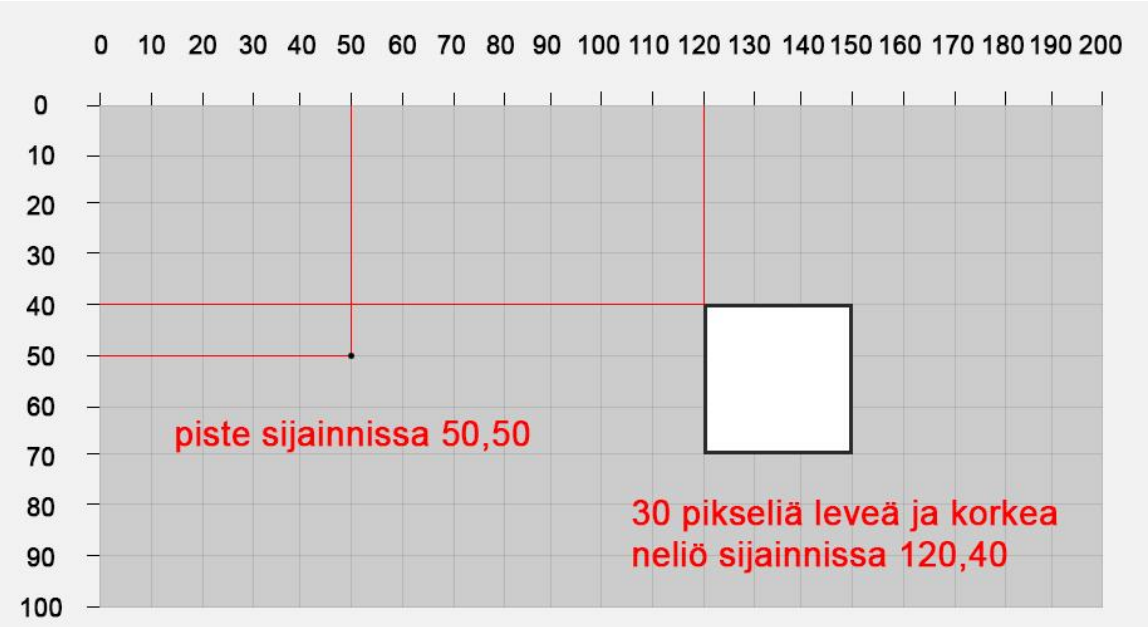
Processingissa kirjoitettuja ohjelmia kutsutaan "sketcheiksi". Kun valmis ohjelma käynnistetään työkalurivin run-komennolla, aukeaa uusi piirtoikkuna, johon ohjelmoidut kuvat tai animaatiot piirtyvät. Piirtoikkuna suljetaan joko työkalurivin stop-painikkeella tai ikkunan yläkulmasta löytyvällä close-komennolla. (Processing 2020b.) Esimerkki tyhjästä piirtoikkunasta on esitetty kuvassa 2.

Kuva 2. Processingin piirtoikkuna

Kirjoitetut ohjelmat tallennetaan kovalevylle .pde-päätteisinä. PDE on lyhenne sanoista Processing Development Environment. Tallentamisen lisäksi ohjelmat voidaan myös pakata Java-applikaatioiksi. (Processing 2020b.)

3.2 Processing-ohjelmointikielen perusteet

Tässä opinnäytetyössä ei ole tarkoituksenmukaista tutkia Processingia ohjelmointikielenä vaan ennemminkin määritellä se, miten Processing käytännössä toimii ja minkälaisia mahdollisuuksia se kuvataiteilijalle tarjoaa. Lähtökohtana voidaan pitää sitä, että kuvia tai animaatioita luodakseen kuvataiteilija kirjoittaa ohjelmointikielellä yksinkertaisia komentoja. Esimerkiksi komento point(50,50); piirtää pisteen kohtaan 50,50 ja komento rect(120,40,30,30); 30 pikseliä leveän ja korkean neliön sijaintiin 120, 40. Kuvioiden sijainti määritellään koordinaatistolla, jonka origo sijaitsee piirtoikkunan vasemmassa yläkulmassa. (Reas & Fry 2007, 23–28.) Edellä kuvattujen käskyjen ja koordinaatiston toimintaa on havainnollistettu kuvassa 3.



Kuva 3. Processingin piirtoikkunan koordinaatisto

Point- ja rect-komentojen lisäksi tyypillisimpiä käskyjä ovat esimerkiksi line(x1, y1, x2, y2);, ellipse(a, b, c, d); sekä triangle(x1, y1, x2, y2, x3, y3);. Muotojen koko ja sijainti määritellään antamalla sulkujen sisässä oleville parametreille halutut arvot. (Reas & Fry 2007, 23–28.) Kaikki Processingin tarjoamat komennot perusmuodoille on esitetty taulukossa 1.

Taulukko 1. Perusmuotojen piirtokomennot Processingissa

point(x,y);	Piste
line(x1,y1,x2,y2);	Viiva
rect(x,y,a,b);	Suorakulmio
square(x, y, a)	Neliö
quad(x1, y1, x2, y2, x3, y3, x4, y4)	Nelikulmio
triangle(x1, y1, x2, y2, x3, y3)	Kolmio
arc(a, b, c, d, s1, s2)	Kaari
circle(x, y, a)	Ympyrä
ellipse(a, b, c, d)	Ellipsi/Ympyrä

Piirrettävien muotojen värit määritellään fill- ja stroke-komennoilla. Fill määrittää täyttövärin ja stroke viivan värin. Sulkeiden sisään annetaan halutut RGB-arvot sekä mahdollinen läpinäkyvyysarvo. Viivan paksuus määritellään strokeWeight-komennolla. (Reas & Fry 2007, 32–34.) Väreihin ja viivanpaksuuteen liittyvät komennot on esitetty taulukossa 2.

Taulukko 2. Väreihin ja viivanpaksuuteen liittyvät komennot Processingissa

fill(r,g,b,a);	Täyttövärin määrittäminen
noFill();	Ei täyttöväriä
stroke(r,g,b,a);	Viivan väri
noStroke();	Ei viivaa
strokeWeight(a);	Viivan paksuuden määrittäminen
background(r,g,b);	Taustan väri

Processingin piirtoikkunan koko määritellään size-komennolla. Komennolle annettavat parametrit ovat leveys ja korkeus. (Reas & Fry 2007, 24.) Esimerkkejä erilaisista size-komennoista on esitetty kuvassa 4.

```
size(100,100);
size(500,500);
size(1920,1080);
```

Kuva 4. Erilaisia size-komentoja

Varsinaisen toiminnallisuuden lisäksi Processingissa voi kirjoittaa koodin sekaan myös kommentteja. Kommenttien avulla ohjelman rakenne on helpompi hahmottaa varsinkin silloin, jos samaa ohjelmaa on työstämässä useampi henkilö. (Reas & Fry 2007, 17–18.) Erilaiset kommentointitavat on esitetty kuvassa 5.

```
/*
   Tämä on monirivinen kommentti, joka voi ulottua
   useammalle riville.
*/

line(0, 50, 100, 50);  // Tämä on kommentti yhdelle riville
```

Kuva 5. Kommentointitavat Processingissa

Edellä kuvatut komennot ovat vain pintaraapaisu siitä, mitä Processingilla voi tehdä. Ne antavat kuitenkin kuvan siitä, miten ohjelmoimalla piirtäminen käytännössä toteutetaan. Varsinaisia ohjelmia kirjoittaessa piirtämistä ei kuitenkaan juuri koskaan toteuteta antamalla piirtofunktioille eksakteja arvoja, vaan piirtäminen perustuu muuttujiin ja erilaisiin rakenteisiin sekä usein myös jonkinlaiseen matematiikkaan. Kaikkiin Processingin sisältämiin komentoihin ja niiden toiminnallisuuteen voi tutustua tarkemmin osoitteessa https://processing.org/reference/.

3.2.1 Ohjelman rakenteet

Piirto-ominaisuuksien lisäksi ohjelmoijan on olennaista hahmottaa ohjelman rakenteen merkitys. Processingissa kirjoitetun ohjelman rakenne koostuu aina funktioista, jotka ovat tavallaan pienempiä ohjelmia ohjelman sisällä. (Reas & Fry 2007, 181.)

Kaksi Processingin oleellisinta funktiota ovat setup- ja draw-funktiot. Setup-funktio sijoitetaan yleensä ohjelman alkuosaan ja siinä määritellään ohjelman kannalta yleisiä ominaisuuksia kuten piirtoikkunan koko ja muuttujien lähtöarvot. Ohjelman käynnistyessä setup-funktio luetaan vain kerran. Heti setup-funktion perään sijoitettavaa draw-funktiota sen sijaan luetaan loputtomasti, kunnes ohjelma pysäytetään. Draw-funktio onkin olennainen nimenomaan animaatioiden tuottamisen kannalta. (Reas & Fry 2007, 173–180.)

Setup- ja draw- sekä muiden funktioiden lisäksi ohjelmien rakenteeseen voidaan vaikuttaa muillakin rakenteilla. Näitä rakenteita ovat esimerkiksi erilaiset

ehtolauseet, toistorakenteet sekä olio-ohjelmointi. Ehtolauseilla voidaan toteuttaa kuvainnollisesti ehtoja, jolloin osa koodista luetaan vain tiettyjen ehtojen täyttyessä. Toistorakenteilla sen sijaan luetaan sama osa koodista useampaan kertaan, mutta niin, että jokin parametri muuttuu jokaisella lukukerralla. Olio-ohjelmoinnilla taas tarkoitetaan ohjelmointitapaa, jossa ohjelma rakennetaan valmiiksi määritellyistä elementeistä. (Reas & Fry 2007, 53, 61, 395.) Kaiken kaikkiaan Processingissa tuotettujen ohjelmien rakenteet on valtava kokonaisuus, jonka hahmottaminen on pitkä prosessi. Kuvataiteilijalle olennaista on kuitenkin heti alkuun oivaltaa, että luodessaan kuvia tai animaatioita ohjelmoimalla pelkkä yksinkertaisten komentojen ymmärtäminen ei riitä, vaan mielenkiintoisten lopputulosten saavuttamiseksi on hallittava myös ohjelmien rakenteellisuus.

3.2.2 Animaatiot

Animaatioiden tuottamiseksi täytyy Processingissa kirjoitetun ohjelman synnyttää piirtoikkunaan jonkinlaista liikettä. Yksinkertaisimmillaan liike saavutetaan muuttamalla yksittäisen pisteen sijaintia aina kun draw-funktio luetaan uudelleen. Esimerkki tällaisesta tilanteesta on havainnollistettuna kuvassa 6. Pisteen x-koordinaattia siis kasvatetaan draw-funktion sisällä jokaisen piirtokerran jälkeen yhdellä yksiköllä. Background-komennolla maalataan piirtotilan tausta valkoiseksi ennen uuden pisteen piirtämistä, jotta vanhat pisteet eivät jää näkyviin. Koska setup-funktiossa on määritelty ”framerateksi” 30, liikkuu piste siis oikealle 30 yksikköä sekunnissa. (Reas & Fry 2007, 174–175.)

Kuva 6. Yksinkertainen animaatio Processingissa

Jotta animaatiot voidaan tallentaa videomuotoon, täytyy ne ensin tallentaa yksittäisiksi "freimeiksi" eli kuvasarjoiksi. Yksittäisten kuvien tallentaminen

kuvatiedostoiksi Processingissa onkin helppoa. Lisäämällä draw-funktion sisään komento save(kuva.png"); ohjelma tallentaa piirtoikkunan kuvan png.-tiedostoksi. Myös JPEG, TIFF ja TGA-muodoissa tallentaminen on mahdollista. Kun sen sijaan halutaan tallentaa animaatioita, muutetaan komento muotoon saveFrame("kuva_####.png");. Tällöin ohjelma tallentaa jokaisen "freimin" ja nimeää sen antamalla järjestysnumeron ristikkomerkkien tilalle. Näin tallennetut "animaatiofreimit" voidaan muuntaa animaatioksi esimerkiksi After Effectissä tai jossakin muussa tarkoitukseen soveltuvassa videoeditointiohjelmassa. Tallennetut kuvat voidaan muuntaa animaatioksi myös Processingin päävalikon Tools-kohdasta löytyvän Movie Maker -työkalun avulla. Työkalu on toimiva, mutta antaa käyttäjälle oikeaa videoeditointiohjelmaa huomattavasti vähemmän mahdollisuuksia vaikuttaa videon laatuun ja pakkaukseen. (Shiffman 2016.)

3.3 Processing-ohjelmointiympäristön mahdollisuudet

Kuvien ja animaatioiden luomisen lisäksi Processingilla voi tehdä paljon muutakin. Taiteilijalle oleellisimpia lisäominaisuuksia ovat mahdollisuus luoda ääntä sekä interaktiivisia teoksia, mutta Processingilla voi myös kirjoittaa verkko- ja puhelinsovelluksia sekä ohjelmoida erilaisia elektronisia laitteita. (FLOSS Manuals 2010.) Ohjelmointiin erikoistunut kuvataiteilija voi siis toteuttaa esimerkiksi ympäristöönsä reagoivan piirustusrobotin tai täysin itsekirjoitettuihin sääntöihin perustuvan maalaussovelluksen.

Processingin toimintamahdollisuuksia voi laajentaa entistä pidemmälle hyödyntämällä yhteisön kirjoittamia lisäosia eli kirjastoja. Kirjaston asentaminen onnistuu valitsemalla päävalikosta Sketch -> Import Library… -> Add Library… sekä avautuneesta valikosta haluttu kirjasto ja Install. Käytännössä kirjastot antavat käyttäjälle joukon uusia funktioita, joita tämä voi hyödyntää huolehtimatta niiden perimmäisten toimintatapojen ymmärtämisestä. Kirjastojen avulla voi tuottaa helposti esimerkiksi audioreaktiivista sisältöä tai ohjelmoida fyysisiä mikrokontrollereita. (FLOSS Manuals 2010.)

Kun kaikki Processingin tarjoamat mahdollisuudet yhdistetään, on helppo huomata, että Processingilla luodessa rajoitteena on todellakin vain mielikuvitus.

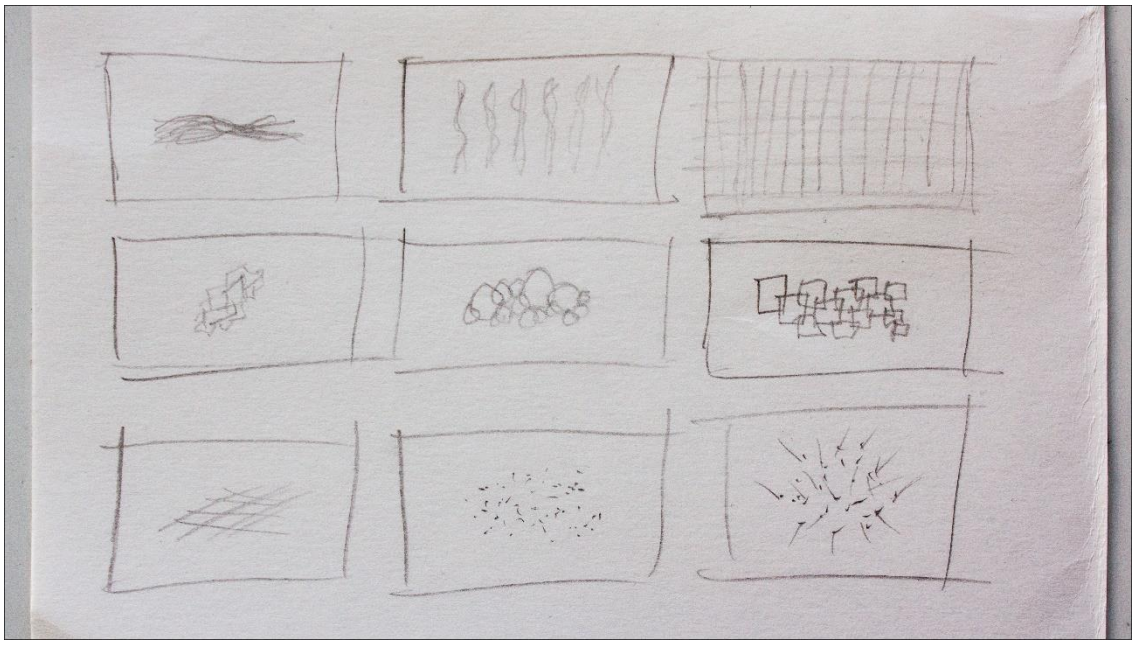
Vaikka ohjelmointi saattaakin siis tuntua kuvataiteilijalle ensi alkuun rajoittavalta työvälineeltä, on se todellisuudessa kenties kaikista vapauttavin.

4 TEOSOSUUS: OHJELMOIMALLA TUOTETUT ANIMAATIOT

4.1 Suunnitelma

Teososuuden ja koko opinnäytetyön tavoitteena oli tutkia, miten ohjelmoimalla voi tuottaa kuvia ja animaatioita. Koska animaatiot koostuvat yksittäisistä kuvista ja yksittäisten kuvien tuottaminen on animaatioita huomattavasti nopeampaa, lähdin liikkeelle nimenomaan kuvien tuottamisesta. Suunnitelmanani oli siis toteuttaa ensin yksittäisiä kuvia ja sitten vähitellen ohjelmointitaitojeni karttuessa kehittää niitä animaatioiksi asti.

Animaatioiden lopputuloksen suhteen tarkkaa suunnitelmaa ei ollut, vaan toimin itselleni tyypillisellä tavalla vapaasti kokeillen ja hetken mielijohteista uutta luoden. Koska omat teokseni ovat aina olleet melko abstrakteja ja ohjelmointi taiteellisena työvälineenä tukee nimenomaan nonfiguratiivista ilmaisua, oli kuitenkin alusta asti selvää, että myös nyt syntyvistä animaatioista tulisi abstrakteja. Vaikka erityistä suunnitelmaa lopputulosten suhteen ei ollutkaan, mielessäni oli kuitenkin jonkinlaisia minimalistisia aihioita, joiden pohjalta aioin lähteä liikkeelle. Kuvassa 7 on joitakin näistä suunnitelmista luonnosteltuna muistiin.

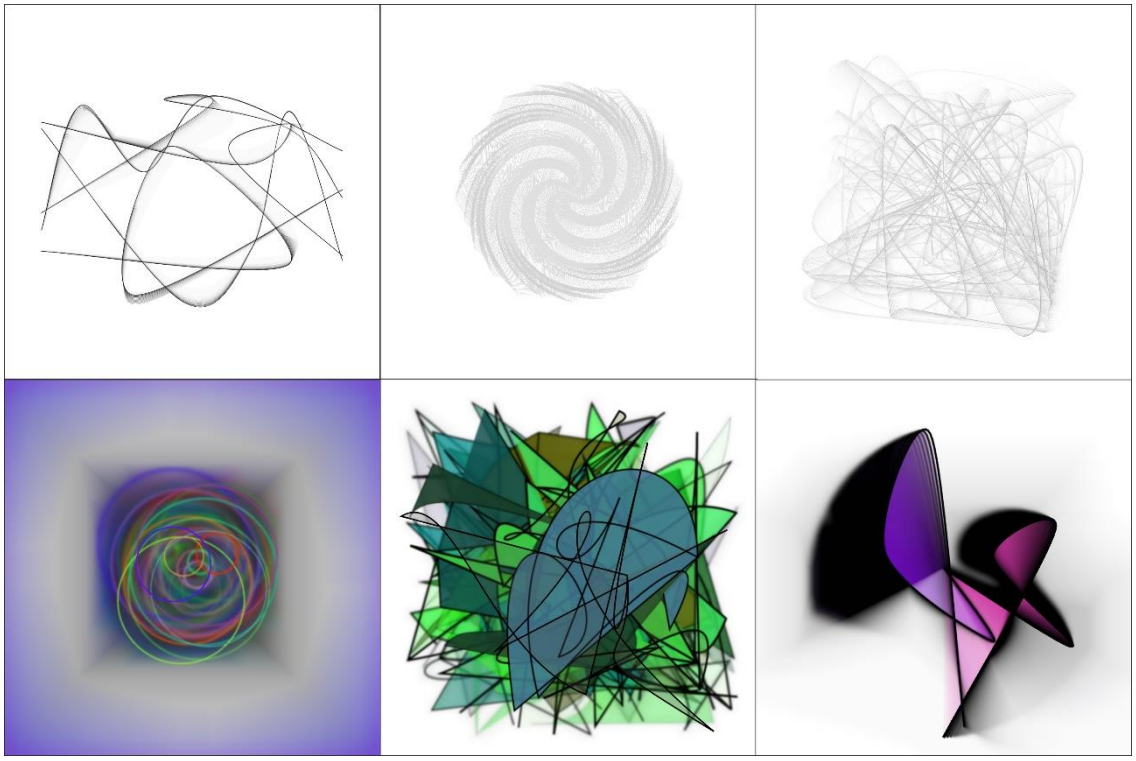


Kuva 7. Suunnitelmia luonnosteltuina

4.2 Toteutus

Animaatioiden toteuttaminen tuntui projektin alkuvaiheessa melko hankalalta. Vaikkakin yksinkertaisten piirtofunktioiden toimintaperiaatteet oli helppo ymmärtää ja onnistuin niiden avulla luomaan nopeasti yksittäisiä kuvia, varsinaisten animaatioiden luominen osoittautui syvempää ymmärrystä vaativaksi tehtäväksi. Jouduinkin käymään läpi ohjelmoinnin opiskeluun käyttämiäni materiaaleja useampaan kertaan ennen kuin aloin ymmärtää animaatioiden luomiseen tarvittavia ohjelmointirakenteita riittävän syvällisesti.

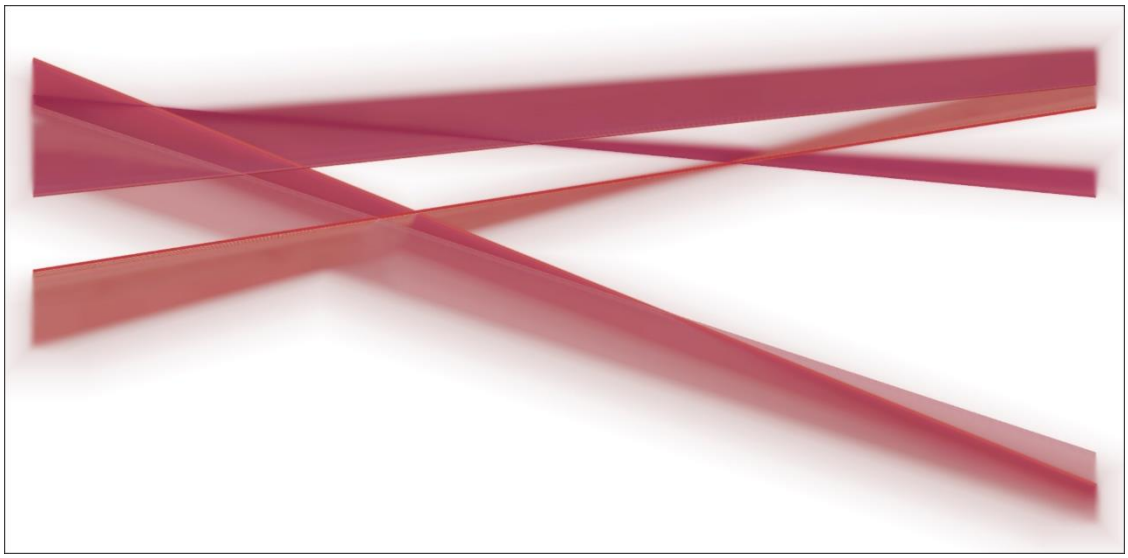
Lopulta opittuani animaatioiden tuottamiseen tarvittavat perusteet tapanani oli aloittaa työskentely jonkin visuaalisen aihion pohjalta ja ohjelmointikielen valmiisiin piirtofunktioihin perustuen. Kokeilin esimerkiksi tuottaa mahdollisimman yksinkertaisia animaatioita, joissa liikutin pelkkiä ympyröitä tai yksittäisiä viivoja. Päästessäni näin alkuun jatkoin kehittämistä ja aloin muokata animaatioista monimutkaisempia sekä visuaalisesti mielenkiintoisempia kokonaisuuksia. Jatkokehittely perustui yleensä hetkessä syntyneisiin ideoihin tai väärin kirjoitetun koodin synnyttämiin mielenkiintoisiin lopputuloksiin. Kuvaan 8 on kerätty kokoelma tässä vaiheessa syntyneistä luonnoksista.



Kuva 8. Kokoelma luonnoksista

Myöhemmässä vaiheessa työskentelyni lähtökohta muuttui visuaalisista ideoista hieman enemmän ohjelmointikielen logiikkaan pohjautuvaksi. Mitä pidemmälle työskentelyni siis eteni, sitä enemmän aloin ajatella mielessäni syntyviä ideoita spontaanisti ohjelmoinnillisista lähtökohdista. Tavallaan oma ideointiprosessini siis muuttui ainakin hetkellisesti puhtaasti visuaalisesta visuaalis-loogiseksi ajatteluksi. Aloin siis miettiä alitajuntaisesti, miten voisin "puhua ohjelmointikieltä" luovasti ja omalla tavallani.

Tekniseltä toteutukseltaan animaatioista suurin osa perustuu Processingin valmiisiin piirtofunktioihin, random-funktioon sekä olio-ohjelmointiin ja erilaisiin toistorakenteisiin. Esimerkiksi kuvan 9 animaatiossa olio-ohjelmoinnin avulla on ensin rakennettu viivan malli, joka sisältää yhden viivan ominaisuudet ja toiminnallisuuden. Ominaisuudet pitävät sisällään viivan sijainnin, nopeuden ja väriarvojen muuttujat, kun taas toiminnallisuus sisältää liikkeen ja ehtolauseista koostuvan törmäysten testauksen logiikan. Viivan mallin rakentamisen jälkeen mallista on luotu haluttu määrä kopioita ja lopulta for-loopin avulla piirretty nämä kopiot piirtotilaan. Random-funktiota on hyödynnetty viivojen sijainnin, nopeuden ja väriarvojen sattumanvaraiseen määrittämiseen. Kyseisen animaation sekä myös muiden teosten pohjalla olevat koodit kommentteineen on sijoitettu opinnäytetyön liitteestä 4.



Kuva 9. Olio-ohjelmointia hyödyntävä animaatio

Koska ohjelmointi antaa niin laajat mahdollisuudet toteuttaa periaatteessa mitä vain, halusin kokeilla perinteisten animaatiotekniikoiden lisäksi myös jotakin muuta tapaa tuottaa animaatiota. Kuvassa 10 on ruutukaappaus animaatiosta,

jossa liike on kylläkin luotu perinteisellä tekniikalla, mutta piirtoikkunaan ilmestyvät ”maalitahrat” valitsevat värinsä lukemalla ulkoista datatiedostoa. Tässä tapauksessa kyseessä on vapaasti verkosta ladattavissa oleva json-tiedosto, joka sisältää Suomen koronavirustartunnat järjestyksessä ja sairaanhoitopiireittäin. Animaatioon piirtyvät maalitahrat siis lukevat datatiedostosta kaikki Suomessa todetut koronavirustartunnat järjestyksessä ja valitsevat värinsä sen mukaan, missä sairaanhoitopiirissä tartunta on todettu. Keltaiset maalitahrat ovat tartuntoja Helsingin ja Uudenmaan sairaanhoitopiirissä, punaiset Pirkanmaalla ja niin edelleen. Vaikka kyseessä on hyvin yksinkertainen tekniikka, on kyseinen teos mielestäni kuvaava esimerkki siitä, kuinka ohjelmointia hyödyntäen on mahdollista havainnollistaa muuten vaikeasti hahmotettavia numeerisia kokonaisuuksia visuaalisessa muodossa. Myöskin kuvataiteilijan teoksiin on siis näin mahdollista saada uudenlaista syvyyttä, kun ilmeisen abstraktin teoksen pohjalle voikin kytkeä oikeasta maailmasta kerättyä dataa.

Kuva 10. Ruutukaappaus ulkoista datatiedostoa hyödyntävästä animaatiosta

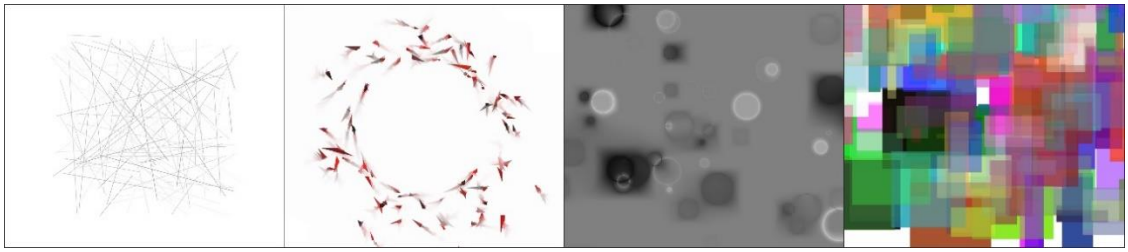
Videomuotoon Processingissa luodut animaatiot on tallennettu luvussa 3.2.2 kuvatun tekniikan avulla. Processingissa kirjoitetun animaatio-ohjelman käynnistyessä jokainen yksittäinen kuva on siis ensin tallennettu kovalevylle ja sen jälkeen kasattu kuvat yhteen After Effectsissä, josta ne on tallennettu videomuotoon. After Effectsin puolella on tehty myös pienimuotoista jälkikäsittelyä lisäämällä animaatioihin hienovaraisia kameran liikkeitä sekä

kuvapintaa muokkaavia efektejä. Myös värimaailmaa on muokattu. Alkuperäisenä ajatuksenani oli luoda animaatioiden taustalle myös jonkinlainen äänimaailma, mutta päädyin lopulta jättämään tämän pois, koska toteuttamani äänet olivat liian hallitsevia ja veivät huomiota pois varsinaisilta animaatioilta.

4.3 Lopputulos

Vaikkakaan en lähtenyt toteuttamaan mitään valmista suunnitelmaa, lopulliset animaationi vastaavat melko hyvin projektin alkuvaiheessa mieleeni muodostuneita ideoita. Animaatiot ovat siis minimalistisia ja ohjelmointilogiikaltaan yksinkertaisia ja täten myös niiden sisältämä liike on rauhallista ja toistuvaa. Toisin sanoen teokseni eivät sisällä yllätyksiä, vaan ovat jossain määrin jopa pitkäveteisiä. Tämä ei kuitenkaan ole huono asia, vaan vastaa nimenomaan alkuperäisiä ajatuksiani. Animaatioiden pituudet vaihtelevat noin 20:stä 30:een sekuntiin. Lopullisia animaatioita on yhteensä kymmenen.

Teoksistani on myös havaittavissa niiden rakentuminen yksittäisten viivojen sekä yksinkertaisten perusmuotojen varaan. Niin kuin kuvasta 11 käy ilmi, teokset koostuvat helposti erotettavissa olevista viivoista, ympyröistä, nelikulmioista ja kolmioista. Yksittäiset viivat ja perusmuodot ovat erotettavissa siitä syystä, että kirjoittamani ohjelmat perustuvat hyvin vahvasti Processingin valmiisiin piirtofunktioihin. Vähäisen ohjelmointikokemukseni takia myös väripinnat ja viivanpaksuudet ovat monissa teoksissani huomattavan tasalaatuisia ja tästä syystä melko yksitoikkoisia. Joistakin animaatioista voidaan niin ikään havaita niiden muistuttavan partikkeliefektejä. Tällöin animaatiot siis näyttävät koostuvan useista pienistä kappaleista, joiden liike syntyy osana suurempaa partikkelijärjestelmää.

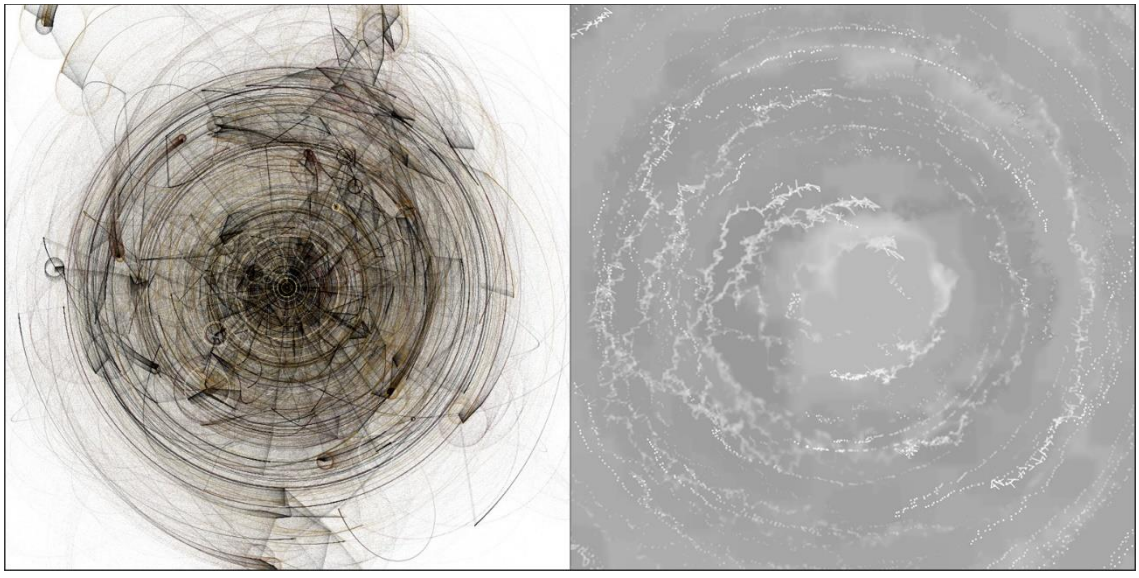


Kuva 11. Perusmuodoista koostuvia teoksia

Värimaailman puolesta teokseni eivät ole tarkkaan suunniteltuja, vaan niiden värit ovat syntyneet suurimmaksi osaksi sattumanvaraisesti random-funktiota hyödyntämällä. Teoksissa on luotettu sattumanvaraisuuteen, jotta tekeminen pysyisi mielenkiintoisempana ja syntyneet lopputulokset vaihtelevampina. Monissa animaatioissa värejä on kuitenkin rajoitettu jonkin verran rajaamalla värimaailma kokonaan mustavalkoiseksi tai random-funktion valitsemia RGB-arvoja pienimmiksi alueiksi.

4.4 Analyysi

Lopulliset animaationi onnistuivat visuaalisuutensa puolesta hyvin. Verrattaessa kuitenkin ammattilaisohjelmoijien tuottamiin teoksiin omista töistäni on nähtävissä vähäinen kokemukseni ohjelmoijana. Kuvassa 12 rinnakkain näkyvässä Jared Tarbellin (2004) teoksessa sekä omassa työssäni ero näkyy erityisesti töiden syvyydessä ja yksityiskohdissa. Tarbellin työssä voidaan selvästi erottaa useampia valöörejä sekä ennen kaikkea hyvin "tarkkaa piirustusjälkeä". Myös värien hallinta on Tarbellilla hienovaraisempaa ja kuva näyttää huomattavasti viimeistellymmältä kuin oma teokseni.



Kuva 12. Vasemmalla Jared Tarbellin (2004) teos Orbitals, oikealla oma animaationi

Lopullisten teosteni kiehtovin osa-alue on niiden liike. Vaikka animaationi eivät yllä logiikaltaan kovin korkealle tasolle, on niihin tuottamani liike sellaista, jota en olisi osannut tuottaa millään muulla tekniikalla. Vaikka olenkin siis aiemmin

toteuttanut lukuisia animaatioita muunlaisilla tekniikoilla, ohjelmoimalla saavutettu liike oli itselleni jotain täysin uutta. Uskoakseni kyse on siitä, että yksinkertaisellakin logiikalla animaatioiden liikkeeseen syntyy jonkinlaista älyä ja sattumanvaraisuutta, joka ei ole täysin ohjelmoijan hallussa. Havaintojeni perusteella näin tuotettu animaatio tuntuu jollain tavoin muuta animaatiota elävämmältä.

Yksinkertaisin keino kehittää teoksiani olisi oppia ohjelmoimaan vielä perusteellisemmin. Kattavammilla taidoilla voisin toteuttaa animaatioihin muun muassa vuorovaikutteisuutta, jonka avulla esimerkiksi toisensa kohtaavat objektit voisivat käynnistää jonkin uuden funktion. Paremmilla taidoilla pystyisin myös toteuttamaan monimutkaisempia 3d:tä hyödyntäviä animaatioita sekä vielä mielenkiintoisempaa liikettä.

Teoksillani ei ole varsinaista sanomaa tai sen kummempaa tavoitetta, vaan ne ovat puhtaasti oman taiteellisen kehittämistyöni seurauksena syntynyttä visuaalista materiaalia. Koska animaationi ovat melko hitaasti eteneviä, uskon että ne saattavat kuitenkin herättää katsojassa jonkinlaisia ajatuksia eivätkä toimi pelkästään passivoivana viihdykkeenä. Omissa mielikuvissani animaatiot olisivatkin ideaalitilanteessa vielä pidempiä ja ne olisi mahdollista esittää yksittäisinä teoksina esimerkiksi avoimessa galleriatilassa.

5 POHDINTA

Opinnäytetyöni päätutkimuskysymyksenä oli selvittää, miten ohjelmoimalla voi tuottaa kuvia ja animaatioita. Omaa kehittämistyötäni tarkasteltuani uskon, että kysymykseen on kaksi vastausta. Ensinnäkin ohjelmoimalla voi tuottaa kuvia ja animaatioita vähäisilläkin ohjelmointitaidoilla ja visuaalisista lähtökohdista. On ilmeistä, että tällainen tilanne on tyypillinen kuvataiteen kentältä ohjelmoinnin pariin siirtyvillä. Toisaalta taiteen tuottamista ohjelmoimalla voi lähestyä myös kattavammilla ohjelmointitaidoilla ja loogisesti haastavammista lähtökohdista. Tällöin mielenkiintoisiin lopputuloksiin on mahdollista päästä niukillakin visuaalisilla taidoilla. Vaikka kuvien ja animaatioiden tuottamista ohjelmoimalla voi siis lähestyä kahdesta eri suunnasta, on sanomattakin selvää, että kaikkein vaikuttavimmat teokset saavuttavat ne ohjelmoijat ja taiteilijat, jotka hallitsevat sekä ohjelmointikielen että kuvan rakentamisen.

Ohjelmoimalla luominen eroaa muista taiteen tekniikoista erityisesti yhden ominaispiirteensä takia. Kyseessä on tietenkin se omituisuus, että ohjelmoimalla luodessa taiteilija ei käytännössä näe tekemäänsä, vaan lopputuloksen näkee vasta kirjoitetun ohjelman käynnistyessä. Kokemusteni perusteella uskallan myös väittää, että ohjelmoimalla luominen on muita taiteen tekniikoita vähemmän itseilmaisullista toimintaa. Näin on siksi, että ohjelmointi on niin vahvasti logiikkaan perustuvaa, että se ei onnistu tunteella vaan järjellä. Toisaalta voidaan miettiä, miksi itseilmaisun pitäisi perustua nimenomaan tunteeseen. Ja onhan siinäkin jotain perin mielenkiintoista, että nimenomaan tunteellista itseilmaisua korostaneen abstraktin ekspressionismin suurmies Jackson Pollock on luonut tiedostamattaan teoksia, jotka nykytiedon valossa ovatkin täynnä loogista matematiikkaa. Itse arvelen, että järkeä ja tunnetta ei voidakaan koskaan kokonaan erottaa toisistaan, vaan tunne perustuu aina jossain määrin myös järkeen ja järki tunteeseen.

Milloin kuvataiteilijan sitten kannattaisi käyttää ohjelmointia työvälineenään? Havaintojeni perusteella uskon ohjelmoinnin sopivan erityisesti niille taiteilijoille, jotka ovat oppineet ohjelmoimaan muissa yhteyksissä, tai niille, joiden ajatuskulku toimii muutenkin loogisesti ja johdonmukaisesti. Koska ohjelmoimaan voi kuitenkin käytännössä oppia kuka vain, uskon ohjelmoinnin olevan

mahdollinen työväline periaatteessa myös kelle tahansa taiteilijalle. Toinen taiteilijaa mahdollisesti askarruttava kysymys on se, miksi käyttää ohjelmointia kuvien tai animaatioiden tuottamiseen, kun samantyyppisiä digitaalisia teoksia voi toteuttaa myös muilla taiteilijan käteen yleensä paremmin sopivilla ohjelmilla kuten Photosopilla, After Effectsillä tai erilaisilla 3d-ohjelmilla. Vastauksen ensimmäinen osa on, että todellisuudessa ohjelmoimalla tuotetut kuvat ja animaatiot ovat aina leimallisesti erilaisia kuin muilla ohjelmilla tai tekniikoilla tuotettu materiaali. Erottavia tekijöitä on vaikea kuvailla sanoin, mutta jollain tavalla ohjelmoimalla tuotettujen teosten taustalla piilevä loogisuus ja matematiikka mielestäni näkyvät. Toinen syy ohjelmoinnin käyttämiseen työvälineenä on se, että ohjelmoimalla pystyy luomaan sellaisia kuvia, joiden luominen muilla tekniikoilla ei olisi käytännössä mahdollista. Tästä esimerkkinä voidaan pitää muun muassa teoksia, jotka koostuvat tuhansista matemaattisten algoritmien mukaan piirretyistä viivoista.

Tärkein peruste ohjelmoinnin käyttämiseen työvälineenä liittyy kuitenkin ennen kaikkea taiteilijan vapauteen. Vaikka ohjelmointi saattaakin tuntua monelle kuvataiteilijalle nimenomaan taiteellista vapautta rajoittavalta työvälineeltä, tarjoaa se todellisuudessa juuri päinvastoin rajattomat mahdollisuudet luovuuden hyödyntämiseen. Näin on siksi, että hallitsemalla ohjelmoinnin tarpeeksi hyvin kuvataiteilija voi loppujen lopuksi ohjelmoida ja rakentaa itse omat työkalunsa. Näin kuvataiteilijasta tulee muiden tuottamista ohjelmista riippumaton ja hän voi hyödyntää käyttämänsä tietokoneen kapasiteetin täsmälleen haluamallaan tavalla. Kuvataiteilijan kannattaa siis käyttää työkalunaan ohjelmointia siksi, että hän on tällöin vapaampi toimimaan digitaalisesti. Toisaalta haittapuolena voidaan pitää sitä, että ohjelmoinnin oppiminen on pitkä prosessi ja realistisesti ajatellen omien työkalujen tuottamiseen tarvittava ohjelmointikokemus mitataan vuosissa. Tällöin kuvataiteilija joutuisi siis väistämättä käyttämään ison osan ajastaan varsinaisen taiteen tuottamisen sijasta ohjelmointiin sekä sen opiskeluun. Kenties suurimmalle osalle ohjelmointia hyödyntävistä taiteilijoista olisikin kannattavinta löytää omat työtavat jostain ohjelmien tuottamisen ja yksinkertaisen ohjelmoinnin väliltä. Tällöin kyseeseen voisi tulla taiteen tuottaminen esimerkiksi yhdistämällä omat ohjelmointitaidot sekä jo olemassa olevat ohjelmat ja lisäosat ja tuottaa näitä hyödyntämällä jotain omaa taitotasoa vastaavaa materiaalia.

Koska opinnäytetyön tarkoituksena oli kehittää omaa taiteellista työskentelyäni, ovat myös sen tulokset hyödyllisiä lähinnä itselleni. Itseni lisäksi uskon tutkimukseni kuitenkin olevan hyödyksi myös niille kuvataiteilijoille tai kuvataideopiskelijoille, jotka ovat kiinnostuneet ohjelmoinnista nimenomaan taiteellisena työvälineenä. Pidän tutkimukseni tuloksia myös odotetunlaisina ja ainakin henkilökohtaisella tasolla luotettavina. Mahdollisista jatkotutkimusideoista pidän mielenkiintoisimpana tunteen ja järjen suhteen tutkimista sekä laajemminkin taiteen sisältävän logiikan kartoittamista. Lopulliset animaatioteokseni ovat nähtävillä osoitteessa http://heikkiahvenainen.com/opinnaytetyo_heikki_ahvenainen.mp4. Teosten pohjalla olevat koodit on sijoitettu opinnäytetyön liitteisiin1-10.

LÄHTEET

Dufva, T. 2020. Digitaalisuuden ykköset ja nollat. Opetushallitus. Viitattu 20.4.2020 https://www.oph.fi/fi/koulutus-ja-tutkinnot/digitaalisuuden-ykkoset-ja-nollat.

FLOSS Manuals 2010. Processing Suomeksi. Viitattu 20.4.2020 http://write.flossmanuals.net/processing-suomeksi/johdanto/.

Galanter, P. 2003. What is Generative art? Complexity Theory as a Context for Art Theory. New York: New York University. Viitattu 15.2.2020 https://philipgalanter.com/downloads/ga2003_what_is_genart.pdf.

Gookin, D. 2014. Beginning Programming with C For Dummies. Hoboken: John Wiley & Sons, Inc. E-kirja. Viitattu 20.4.2020 https://luc.finna.fi/lapinamk/, Ebook Central.

Greenberg, I. 2007. Processing: Creative Coding and Computational Art. New York: Springer-Verlag New York, Inc.

Neal, B. 2018. A History of Creative Coding. Medium 18.8.2018. Viitattu 21.4.2020 https://medium.com/@laserpilot/a-history-of-creative-coding-8771524b9775.

Pearson, M. 2011. Generative art: A practical guide using processing. New York: Manning Publications Co.

Processing 2020a. Overview. A short introduction to the Processing software and projects from the community. Viitattu 20.4.2020 https://processing.org/overview/.

Processing 2020b. Environment (IDE). Viitattu 20.4.2020 https://processing.org/reference/environment/.

Reas, C & Fry, B. 2007. Processing: a programming handbook for visual designers and artists. Cambridge: The MIT Press.

Reunanen, M. 2010. Computer Demos—What Makes Them Tick? Aalto-yliopisto. Mediatekniikan laitos. Lisensiaatintutkimus. Viitattu 3.5.2020 https://aaltodoc.aalto.fi/bitstream/handle/123456789/3365/urn100199.pdf?sequence=1&isAllowed=y.

Rouse, M. 2016. Integrated development environment (IDE). TechTarget 3.6.2016. Viitattu 20.4.2020 https://searchsoftwarequality.techtarget.com/definition/integrated-development-environment.

Saariketo, M. 2015. Maailman rakentavat ne, jotka osaavat ohjelmoida? Verke 27.4.2015. Viitattu 20.4.2020 https://www.verke.org/blog/maailman-rakentavat-ne-jotka-osaavat-ohjelmoida/.

Salonen, K. 2013. Näkökulmia tutkimukselliseen ja toiminnalliseen opinnäytetyöhön. Turku: Turun ammattikorkeakoulu. Puheenvuoro. Viitattu 23.3.2020 http://julkaisut.turkuamk.fi/isbn9789522163738.pdf.

Shiffman, D. 2016. Tutorial: How to render Processing sketch as a movie. The Coding Train 9.2.2016. Viitattu 20.4.2020 https://www.youtube.com/watch?v=G2hI9XL6oyk.

Tarbell, J. 2004. Orbitals. Variation B. Complexification 2012. Viitattu 20.4.2020 http://www.complexification.net/gallery/machines/orbitals/indexB.php.

Wu, J. 2020. Getting Started With Creative Coding. When art meets programming and generates design. Medium 5.1.2020. Viitattu 20.4.2020 https://medium.com/better-programming/getting-started-with-creative-coding-16072ff7e778.

LIITTEET

Liite 1 1(5). Animaation 1 koodi

```
Ympyra[] pallot = new Ympyra[1165]; //Julistetaan lista, joka sisältää
maalitahrat piirtävien pallojen instanssit

//Luodaan muuttujat
int i;
int a;
float b;
float xspeed;
float f1, f2, f3, f4, f5, f6, f7, f8, f9, f10, f11, f12, f13, f14,
f15, f16, f17, f18, f19, f20, f21, f22, f23, f24;
float c;
String headline;

//Setup-funktio, jonka sisällä määritellään lähtötilanne
void setup() {
  size(1920, 1080); //Määritellään piirtoikkunan koko
  smooth(); //Määritellään piirtotilan "anti-aliasing-tilaksi smooth"
  background(255); //Määritellään taustaväri
  frameRate(30); //Määritellään framerate
  blendMode(REPLACE); //Määritellään sekoitussuhde

  //For-silmukka, jolla luodaan pallot
  for (int i=0; i<1165; i++) {
    pallot[i]=new Ympyra();
  }

  //Määritellään RGB-arvot random-funktiolla
  f1=random(255);
  f2=random(255);
  f3=random(255);
  f4=random(255);
  f5=random(255);
  f6=random(255);
  f7=random(255);
  f8=random(255);
  f9=random(255);
  f10=random(255);
  f11=random(255);
  f12=random(255);
  f13=random(255);
  f14=random(255);
  f15=random(255);
  f16=random(255);
  f17=random(255);
  f18=random(255);
  f19=random(255);
  f20=random(255);
  f21=random(255);
  f22=random(255);
  f23=random(255);
  f24=random(255);
}

//Draw-funktio, jonka sisällä kaikki toiminnallisuus tapahtuu
void draw() {

  //While-silmukka, jonka sisällä piirretään ja liikutetaan palloja
eli "maalitahroja"
  while (i<1165) {
```

Liite 1 2(5). Animaation 1 koodi

```
    fill(255, 5); //Määritellään täyttöväriksi valkoinen alpha-arvolla
5
    noStroke(); //Ei piirretä reunaviivoja
    rect(0, 0, 1920, 1080); //Piirretään lähes läpinäkyvä tausta
edellisen freimin päälle
    JSONObject json = loadJSONObject("corona.json"); //Ladataan
ulkoinen json-tiedosto
    //Ladataan ulkoisesta tiedostosta sairaanhoitopiirin nimi ja
tallennetaan se muuttujaan "headline"
    String headline =
json.getJSONArray("confirmed").getJSONObject(int(i)).getString("health
CareDistrict");

    if (headline.equals("Lappi")) { //Tarkistetaan, onko muuttujaan
tallennettu sairaanhoitopiiri "Lappi". Jos on, toteutetaan alla oleva
koodi.
      fill(f1, f2, f3, 125); //Määritellään täyttöväri
      pallot[i].piirra1(); //Piirretään pallo
    } else if (headline.equals("HUS")) { //Jos muuttujaan tallennettu
sairaanhoitopiiri ei ole "Lappi", tarkistetaan, onko se "HUS".
      fill(255, f6, f7, 255);
      pallot[i].piirra2();
    } else if (headline.equals("Kanta-Häme")) {
      fill(f9, f10, f11, 125);
      pallot[i].piirra3();
    } else if (headline.equals("Pirkanmaa")) {
      fill(f13, f14, f15, 125);
      pallot[i].piirra4();
    } else if (headline.equals("null")) { //Jos muuttujaan tallennettu
sairaanhoitopiiri ei ole "Lappi", tarkistetaan, onko se "null" eli ei
tiedossa.
      fill(f17, f18, f19, 125);
      pallot[i].piirra5();
    } else if (headline.equals("Varsinais-Suomi")) {
      fill(f21, f22, f23, 125);
      pallot[i].piirra6();
    }
    //Ohjelma osaa lukea vain suurinpien sairaanhoitopiirien
tartunnat. Jos tartunta on muusta piiristä, hypätään seuraavan
tartunnan tietoihin.
    i=i+1; //Lisätään muuttujaan i arvoon 1, jotta siirrytään
seuraavan tartunnan tietoihin
    checkkaa(); //Luetaan funktio checkkaa eli päivitetään kaikkien
pallojen uudet sijainnit + piirretään pallot
    filter(BLUR, random(1)); //"Blurrataan" kuvaa arvolla 0-1
    saveFrame("frames/korona_animaatio_001_001-####.png");
//Tallennetaan "freimit"
    break; // Hypätään koodissa eteenpäin eli while-silmukan alkuun
  }
}

//Funktio checkkaa
void checkkaa() {
  //For-silmukka, jolla päivitetään kaikkien pallojen uudet sijainnit
+ piirretään pallot
  for (a=0; a<i; a++) {
    pushMatrix(); //Otetaan käyttöön väliaikainen koordinaatisto

    translate(random(-2, 2), random(-2, 2)); //"Täräytetään"
koordinaatistoa muutaman pikselin verran
```

Liite 1 3(5). Animaation 1 koodi

```
    JSONObject json = loadJSONObject("corona.json"); //Sama kuin
edellä
    String headline =
json.getJSONArray("confirmed").getJSONObject(int(a)).getString("health
CareDistrict"); //Sama kuin edellä
    if (headline.equals("Lappi")) { //Sama kuin edellä
      fill(f1, f2, f3, 125); //Sama kuin edellä
      pallot[a].update1(); //Luetaan funktio update eli päivitetään
"Lappi-pallojen" pallojen uudet sijainnit + piirretään pallot
    }
    if (headline.equals("HUS")) {
      fill(255, f6, f7, 255);
      pallot[a].update2();
    }
    if (headline.equals("Kanta-Häme")) {
      fill(f9, f10, f11, 125);
      pallot[a].update3();
    }
    if (headline.equals("Pirkanmaa")) {
      fill(f13, f14, f15, 125);
      pallot[a].update4();
    }
    if (headline.equals("null")) {
      fill(f17, f18, f19, 125);
      pallot[a].update5();
    }
    if (headline.equals("Varsinais-Suomi")) {
      fill(f21, f22, f23, 125);
      pallot[a].update6();
    }
    popMatrix(); //Otetaan käyttöön alkuperäinen koordinaatisto
    translate(random(-.1, .1), random(-.1, .1)); //"Täräytetään"
koordinaatistoa minimaalisesti
  }
}

//Luodaan pallon malli eli olio
class Ympyra {
  //Nimetään muuttujat ja määritellään niiden tyyppi
  float x1, x2, x3, x4, x5, x6;
  float y1, y2, y3, y4, y5, y6;
  float a;
  float x1speed, x2speed, x3speed, x4speed, x5speed, x6speed;
  float y1speed, y2speed, y3speed, y4speed, y5speed, y6speed;

  //Määritellään pallon muuttujien lähtöarvot
  Ympyra() {
    //Pallojen alkuperäinen sijainti
    x1=random(1920);
    y1=random(1080);
    x2=random(1920);
    y2=random(1080);
    x3=random(1920);
    y3=random(1080);
    x4=random(1920);
    y4=random(1080);
    x5=random(1920);
    y5=random(1080);
    x6=random(1920);
```

Liite 1 4(5). Animaation 1 koodi

```
    y6=random(1080);
    //Pallojen koko random-funktiolla sattumanvaraiseksi
    a=random(30);
    //Pallojen nopeudet
    x1speed=random(-2, 2);
    x2speed=random(-2, 2);
    x3speed=random(-2, 2);
    x4speed=random(-2, 2);
    x5speed=random(-2, 2);
    x6speed=random(-2, 2);
    y1speed=random(-2, 2);
    y2speed=random(-2, 2);
    y3speed=random(-2, 2);
    y4speed=random(-2, 2);
    y5speed=random(-2, 2);
    y6speed=random(-2, 2);
  }

  //Piirretään pallot
  void piirra1() { //Funktio piirra1
    ellipse(x1, y1, a, a); //Piirretään "Lappi-pallo"
  }
  void piirra2() {
    ellipse(x2, y2, a, a);
  }
  void piirra3() {
    ellipse(x3, y3, a, a);
  }
  void piirra4() {
    ellipse(x4, y4, a, a);
  }
  void piirra5() {
    ellipse(x5, y5, a, a);
  }
  void piirra6() {
    ellipse(x6, y6, a, a);
  }

  //Päivitetään pallojen uusi sijainti + piirretään ne
  void update1() {
    ellipse(x1, y1, a, a); //Piirretään "Lappi-pallo" uudessa
sijainnissa
    x1=x1+x1speed; //Määritellään "Lappi-pallolle" uusi x-sijainti
    y1=y1+y1speed; //Määritellään "Lappi-pallolle" uusi y-sijainti
    if (x1>1920) { //Jos pallo osuu "ruudun reunaan", käännytään
takaisin
      x1speed=x1speed*-1;
    }
    if (y1>1080) { //Jos pallo osuu "ruudun ala- tai yläreunaan",
käännytään takaisin
      y1speed=y1speed*-1;
    }
  }
  void update2() {
    ellipse(x2, y2, a, a);
    x2=x2+x2speed;
    y2=y2+y2speed;
    if (x2>1920) {
      x2speed=x2speed*-1;
    }
```

Liite 1 5(5). Animaation 1 koodi

```
    if (y2>1080) {
      y2speed=y2speed*-1;
    }
  }
  void update3() {
    ellipse(x3, y3, a, a);
    x3=x3+x3speed;
    y3=y3+y3speed;
    if (x3>1920) {
      x3speed=x3speed*-1;
    }
    if (y3>1080) {
      y3speed=y3speed*-1;
    }
  }
  void update4() {
    ellipse(x4, y4, a, a);
    x4=x4+x4speed;
    y4=y4+y4speed;
    if (x4>1920) {
      x4speed=x4speed*-1;
    }
    if (y4>1080) {
      y4speed=y4speed*-1;
    }
  }
  void update5() {
    ellipse(x5, y5, a, a);
    x5=x5+x5speed;
    y5=y5+y5speed;
    if (x5>1920) {
      x5speed=x5speed*-1;
    }
    if (y5>1080) {
      y5speed=y5speed*-1;
    }
  }
  void update6() {
    ellipse(x6, y6, a, a);
    x6=x6+x6speed;
    y6=y6+y6speed;
    if (x6>1920) {
      x6speed=x6speed*-1;
    }
    if (y6>1080) {
      y6speed=y6speed*-1;
    }
  }
}
```

Liite 2 1(3). Animaation 2 koodi

```
KeskeltaViiva[] omat_KeskeltaViiva=new KeskeltaViiva[68];
//Julistetaan lista, joka sisältää viivojen instanssit
//Nimetään muuttujat ja määritellään niiden tyyppi
float a;

//Setup-funktio, jonka sisällä määritellään lähtötilanne
void setup() {
  size(3000, 2000, P3D); //Isompi piirtoikkuna, jotta jää tilaa
kamera-ajolle Afterissa
  smooth();
  frameRate(30);
  background(255);
  //For-silmukka, jolla luodaan viivat
  for (int i=0; i<omat_KeskeltaViiva.length; i++) {
    omat_KeskeltaViiva[i]=new KeskeltaViiva();
  }
  a=0; //Annetaan muuttujalla a arvoksi 0
}

//Draw-funktio, jonka sisällä kaikki toiminnallisuus tapahtuu
void draw() {
  translate(width/2, height/2); //Siirretään koordinaatiston
nollakohta piirtoikkunan keskipisteeseen
  rotate(radians(a)); //Käännetään koordinaatistoa muuttujan a arvon
verran
  a=a+.5; //Kasvatetaan muuttujan a arvoa
  noStroke(); // Ei piirretä reunaviivaa
  fill(255, 50); //Määritellään täyttöväri ja alpha-arvo
  rect(-1500, -1000, 3000, 2000); //Piirretään valkoinen nelikulmio
kuvan päälle, jotta vanhat viivat peittyvät
  //For-silmukka, jolla piirretään viivat
  for (int i=0; i<omat_KeskeltaViiva.length; i++) {
    omat_KeskeltaViiva[i].piirra(); //Luetaan funktio "piirra"
  }
  saveFrame("frames/KeskeltaViivat_viivat_anim_002_001_####.png");
//Tallennetaan "freimit"
}

//Luodaan viivan malli eli olio
class KeskeltaViiva {
  //Nimetään muuttujat ja määritellään niiden tyyppi
  float x2, x3, x4, x5, y2, y3, y4, y5;
  float x2speed, y2speed;
  float x3speed, y3speed;
  float x4speed, y4speed;
  float y5speed;
  float s, ss;

  //Määritellään viivan muuttujien lähtöarvot
  KeskeltaViiva() {
    x2=random(-400, 400);
    y2=random(-400, 400);
    x3=random(-400, 400);
    y3=random(-400, 400);
    x4=random(-400, 400);
    y4=random(-400, 400);
    x5=400;
    y5=random(100, 400);
    x2speed=random(-2, 2);
    y2speed=random(-2, 2);
```

Liite 2 2(3). Animaation 2 koodi

```
    x3speed=random(-2, 2);
    y3speed=random(-2, 2);
    x4speed=random(-2, 2);
    y4speed=random(-2, 2);
    y5speed=random(-2, 2);
    s=random(0, 2);
    ss=random(180, 255);
  }

  //Piirretään viivat hyödyntämällä beginShape-, endShape- ja
curveVertex-funktioita
  void piirra() {
    noFill(); //Ei täyttöväriä
    stroke(ss); //Viivan väri
    strokeWeight(s); //Viivanpaksuus
    beginShape(); //Aloita muoto
    curveVertex(0, 0); //Viivan alkupiste
    curveVertex(0, 0); //Viivan alkupiste
    curveVertex(x2, y2); //Viivan koordinaattipiste
    curveVertex(x3, y3); //Viivan koordinaattipiste
    curveVertex(x4, y4); //Viivan koordinaattipiste
    curveVertex(x5, y5); //Viivan loppupiste
    curveVertex(x5, y5); //Viivan loppupiste
    endShape(); //Lopeta muoto

    //Annetaan viivojen koordinaattipisteille uudet arvot ja
tarkastetaan, että viivat eivät kulkeude liian kauas keskipisteestä
    x2=x2+x2speed;
    if (x2>400) { //Jos ollaan 400:n pikselin päässä keskipisteestä,
käännytään takaisin
      x2speed=x2speed*-1; //Muutetaan nopeus sen vastaluvuksi
    }
    if (x2<-400) {
      x2speed=x2speed*-1;
    }
    y2=y2+y2speed;
    if (y2>400) {
      y2speed=y2speed*-1;
    }
    if (y2<-400) {
      y2speed=y2speed*-1;
    }
    x3=x3+x3speed;
    if (x3>400) {
      x3speed=x3speed*-1;
    }
    if (x3<-400) {
      x3speed=x3speed*-1;
    }
    y4=y4+y4speed;
    if (y4>400) {
      y4speed=y4speed*-1;
    }
    if (y4<-400) {
      y4speed=y4speed*-1;
    }
    y5=y5+y5speed;
    if (y5>400) {
      y5speed=y5speed*-1;
    }
```

Liite 2 3(3). Animaation 2 koodi

```
    y5=y5+y5speed;
    if (y5<-400) {
      y5speed=y5speed*-1;
    }
  }
}
```

Liite 3 1(3). Animaation 3 koodi

```
Kaareva[] omatKaaret=new Kaareva[3]; //Julistetaan lista, joka
sisältää viivojen instanssit(3 kpl)

//Setup-funktio, jonka sisällä määritellään lähtötilanne
void setup() {
  size(3000, 1500); //Suurempi piirtoikkuna, jotta Afterissa varaa
kamera-ajolle
  smooth();
  background(255);
  //For-silmukka, jolla luodaan viivat
  for (int i=0; i<omatKaaret.length; i++) {
    omatKaaret[i]=new Kaareva(); //Tallennetaan viivat "arrayhin" eli
listaan
  }
}

//Draw-funktio, jonka sisällä kaikki toiminnallisuus tapahtuu
void draw() {
  noStroke(); //Ei reunaviivaa
  fill(255, 25); //Täyttöväri valkoinen, alpha-arvo 25/255
  rect(0, 0, 3000, 1500); //Piirretään osittain läpinäkyvä valkoinen
nelikulmio kuvan päälle, jotta vanhat viivat peittyvät
  //For-silmukka, jolla piirretään viivat
  for (int i=0; i<omatKaaret.length; i++) {
    omatKaaret[i].piirra(); //Luetaan funktio "piirra"
  }
  saveFrame("frames/kaarevat_viivat_anim_001_001_####.png");
}

//Luodaan viivan malli eli olio
class Kaareva {
  float x1, x2, x3, x4, x5, y1, y2, y3, y4, y5;
  float x2speed, y2speed;
  float x3speed, y3speed;
  float x4speed, y4speed;
  float f1, f2, f3, f4;
  float s;
  float y1speed, y5speed;

  //Määritellään viivan muuttujien lähtöarvot
  Kaareva() {
    x1=100;
    y1=random(100, 1400);
    x5=2900;
    y5=random(100, 1400);
    x2=random(100, 2900);
    y2=random(100, 1400);
    x3=random(100, 2900);
    y3=random(100, 1400);
    x4=random(100, 2900);
    y4=random(100, 1400);
    x2speed=random(-6, 6);
    y2speed=random(-6, 6);
    x3speed=random(-6, 6);
    y3speed=random(-6, 6);
    x4speed=random(-6, 6);
    y4speed=random(-6, 6);
    y1speed=random(-6, 6);
    y5speed=random(-6, 6);
    f1=random(255);
```

Liite 3 2(3). Animaation 3 koodi

```
    f2=random(255);
    f3=random(255);
    f4=random(25);
    s=random(0, 10);
  }
  //Piirretään viivat käyttämällä beginShape-, curveVertex- ja
endShape-funktioita.
  void piirra() {
    fill(f1, f2, f3, f4);
    strokeWeight(s);
    beginShape(); //Aloitetaan muoto
    curveVertex(x1, y1); //Viivan alkupiste
    curveVertex(x1, y1); //Viivan alkupiste
    curveVertex(x2, y2); //Viivan koordinaattipiste
    curveVertex(x3, y3); //Viivan koordinaattipiste
    curveVertex(x4, y4); //Viivan koordinaattipiste
    curveVertex(x5, y5); //Viivan loppupiste
    curveVertex(x5, y5); //Viivan loppupiste
    endShape(); //Lopetetaan muoto

    //Annetaan viivojen koordinaattipisteille uudet arvot ja
tarkistetaan, että ne eivät kulkeude liian kauas sivuille tai ylös tai
alas
    //Jos koordinaattipiste on liian kaukana kuvan keskipisteestä,
muutetaan sen nopeus sen vastaluvuksi
    x2=x2+x2speed;
    if (x2>2900) {
      x2speed=x2speed*-1;
    }
    if (x2<100) {
      x2speed=x2speed*-1;
    }
    y2=y2+y2speed;
    if (y2>1400) {
      y2speed=y2speed*-1;
    }
    if (y2<100) {
      y2speed=y2speed*-1;
    }
    x3=x3+x3speed;
    if (x3>2900) {
      x3speed=x3speed*-1;
    }
    if (x3<100) {
      x3speed=x3speed*-1;
    }
    y3=y3+y3speed;
    if (y3>1400) {
      y3speed=y3speed*-1;
    }
    if (y3<100) {
      y3speed=y3speed*-1;
    }
    x4=x4+x4speed;
    if (x4>2900) {
      x4speed=x4speed*-1;
    }
    if (x4<100) {
      x4speed=x4speed*-1;
    }
```

Liite 3 3(3). Animaation 3 koodi

```
    y4=y4+y4speed;
    if (y4>1400) {
      y4speed=y4speed*-1;
    }
    if (y4<100) {
      y4speed=y4speed*-1;
    }
    //Annetaan myös viivojen alku- ja loppupisteille uudet arvot ja
tarkistetaan, että ne eivät kulkeude liian ylös tai alas
    //Jos alku- tai loppupiste on liian ylhäällä tai alhaalla,
muutetaan sen nopeus sen vastaluvuksi
    y1=y1+y1speed;
    if (y1>1400) {
      y1speed=y1speed*-1;
    }
    if (y1<100) {
      y1speed=y1speed*-1;
    }
    y5=y5+y5speed;
    if (y5>1400) {
      y5speed=y5speed*-1;
    }
    y5=y5+y5speed;
    if (y5<100) {
      y5speed=y5speed*-1;
    }
  }
}
```

Liite 4 1(2). Animaation 4 koodi

```
Viiva[] viivaArr= new Viiva[5]; //Julistetaan lista, joka sisältää
viivojen instanssit(5 kpl)
//Nimetään muuttujat ja määritellään niiden tyyppi
float a;

//Setup-funktio, jonka sisällä määritellään lähtötilanne
void setup() {
  size(1920, 1080); //Piirtoikkunan koko
  frameRate(30);
  background(255);
  smooth();
  //For-silmukka, jolla luodaan viivat
  for (int i=0; i<viivaArr.length; i++) {
    viivaArr[i]=new Viiva(); //Tallennetaan viivat "arrayhin" eli
listaan
  }
}

//Draw-funktio, jonka sisällä kaikki toiminnallisuus tapahtuu
void draw() {
  a=random(0, 10); //Annetaan a:lle arvo väliltä 0-10
  fill(255, a); //Täyttöväri valkoinen, alpha-arvo 0-10/255
  noStroke(); //Ei reunaviivaa
  //For-silmukka, jolla piirretään viivat
  for (int i=0; i<viivaArr.length; i++) {
    viivaArr[i].piirraViivat(); //Luetaan funktio ""piirraViivat"
    viivaArr[i].piirraViivat2(); //Luetaan funktio ""piirraViivat2"
  }
  saveFrame("frames/viivakuva_anim_003_#####.png"); //Tallennetaan
"freimit"
}

//Luodaan viivan malli eli olio
class Viiva {
  float y1;
  float y2;
  float yspeed1;
  float yspeed2;
  float s1;
  float s2;
  float s3;
  float w;

  //Määritellään viivan muuttujien lähtöarvot
  Viiva() {
    y1=random(50, 1030);
    y2=random(50, 1030);
    yspeed1=random(-1, 1);
    yspeed2=random(-1, 1);
    s1=random(160, 225); //Värin r-arvo 160-225
    s2=random(255); //Värin g-arvo 0-255
    s3=random(175, 255); //Värin b-arvo 175-255
    w=random(.2, 5);
  }

  //Piirretään viivat, jotka blurraantuvat
  void piirraViivat() {
    stroke(s1, s2, s3); //Viivan väri, RGB-arvo ei täysin
sattumanvarainen, vaan hieman rajattu
    strokeWeight(w); //Viivan paksuus
```

Liite 4 2(2). Animaation 4 koodi

```
    line(50, y1, 1870, y2); //Piirretään viiva
    filter(BLUR, random(-1, 1)); //"Blurrataan" kuvaa
    y1=y1+yspeed1; //Viivan vasemman reunan uusi y-koordinaatti
    y2=y2+yspeed2; //Viivan oikean reunan uusi y-koordinaatti
    //Jos viivan y-koordinaatti on suurempi kuin 1030 tai pinenempi
kuin 50, viivan nopeus muuttuu sen vastaluvuksi
    if (y1>1030) {
      yspeed1=yspeed1*-1;
    }
    if (y1<50) {
      yspeed1=yspeed1*-1;
    }
    if (y2>1030) {
      yspeed2=yspeed2*-1;
    }
    if (y2<50) {
      yspeed2=yspeed2*-1;
    }
  }
  //Piirretään päälle viivat, joiden väri vaihtelee hieman
  void piirraViivat2() {
    stroke(random(165, 235), random(60), random(90)); //"Freimeittäin"
vaihtelava täyttöväri
    strokeWeight(w); //Viivan paksuus
    line(50, y1, 1870, y2); //Piirretään viiva
    y1=y1+yspeed1; //Viivan vasemman reunan uusi y-koordinaatti
    y2=y2+yspeed2; //Viivan oikean reunan uusi y-koordinaatti
    //Jos viivan y-koordinaatti on suurempi kuin 1030 tai pinenempi
kuin 50, viivan nopeus muuttuu sen vastaluvuksi
    if (y1>1030) {
      yspeed1=yspeed1*-1;
    }
    if (y1<50) {
      yspeed1=yspeed1*-1;
    }
    if (y2>1030) {
      yspeed2=yspeed2*-1;
    }
    if (y2<50) {
      yspeed2=yspeed2*-1;
    }
  }
}
```

Liite 5 1(2). Animaation 5 koodi

```
Viiva[] omaviiva= new Viiva[100]; //Julistetaan lista, joka sisältää
viivan instanssit
float zoom; //Luodaan muuttuja, jota käytetään "kamera-ajoon"

//Setup-funktio, jonka sisällä määritellään lähtötilanne
void setup() {
  size(1920, 1080, P3D); //Määritellään piirtoikkunan koko
  background(255); //Määritellään taustaväri
  frameRate(30); //Määritellään framerate
  smooth(); //Määritellään piirtotilan "anti-aliasing-tilaksi smooth"
  //For-silmukka, jolla luodaan viivat
  for (int i=0; i<omaviiva.length; i++) {
    omaviiva[i]=new Viiva();
  }
}

//Draw-funktio, jonka sisällä kaikki toiminnallisuus tapahtuu
void draw() {
  translate(width/2, height/2, zoom); //Siirretään koordinaatiston
nollakohta piirtoikkunan keskipisteeseen ja zoomataan
  fill(255, 200); //Täyttöväri valkoinen, alpha-arvo 200/255
  noStroke(); //Ei reunaviivaa
  rectMode(CENTER); //Määritellään nelikulmioiden piirtotilaksi
"CENTER", jolloin niiden koordinaateilla viitataan niiden
keskipisteeseen
  rect(0, 0, 1920, 1080); //Peitetään edellinen freimi piirtämällä
valkoinen nelikulmio edellisen kuvan päälle
  //For-silmukka, jolla piirretään ja liikutetaan viivoja
  for (int i=0; i<omaviiva.length; i++) {
    omaviiva[i].display(); //Luetaan funktio "display"
    omaviiva[i].move(); //Luetaan funktio "move"
  }
  zoom+=1; //Lisätään muuttujan zoom arvoon 1
  saveFrame("framet/viivoja_anim_001_####.png"); //Tallennetaan
"freimit"
}

//Luodaan viivan malli eli olio
class Viiva {
  //Nimetään muuttujat ja määritellään niiden tyyppi
  float x1;
  float y1;
  float x2;
  float y2;
  float x1speed;
  float y1speed;
  float x2speed;
  float y2speed;
  float s;
  float c;
  float a;

  //Määritellään viivan muuttujien lähtöarvot
  Viiva () {
    x1=random(-200, 200); //Määritellään x1-muuttujan arvo random-
funktiolla, arvo on siis jotakin -200:n ja 200 väliltä
    y1=random(-200, 200);
    x2=random(-200, 200);
    y2=random(-200, 200);
    s=random(1, 2);
```

Liite 5 2(2). Animaation 5 koodi

```
    c=random(150, 255);
  }

  //Piirretään viiva
  void display() {
    strokeWeight(s); //Viivan paksuus
    stroke(c); //Viivan väri
    line(x1, y1, x2, y2); //Viivan koordinaatit
  }

  //Määritellään viivan liike
  void move() {
    //Lasketaan liikkeen nopeuden määrittäville muuttujille uusi arvo
hyödyntäen sini-, kosini- ja random-funktioita
    x1speed=sin(radians(a))*random(-1, 1)*cos(radians(a))*random(-1,
1);
    y1speed=cos(radians(a));
    x2speed=sin(radians(a))*random(-1, 1);
    y2speed=cos(radians(a));
    a=a+200; //Kasvatetaan muuttujaa a
    x1=x1+x1speed; //Viivan x1-koordinaatti saa uuden arvon
    y1=y1+y1speed; //Viivan y1-koordinaatti saa uuden arvon
    x2=x2+x2speed; //Viivan x2-koordinaatti saa uuden arvon
    y2=y2+y2speed; //Viivan y2-koordinaatti saa uuden arvon
  }
}
```

Liite 6. Animaation 6 koodi

```
Nelikulmio[] omatNelikulmiot=new Nelikulmio[150]; //Julistetaan lista,
joka sisältää nelikulmioiden instanssit

//Setup-funktio, jonka sisällä määritellään lähtötilanne
void setup() {
  size(1920, 1080); //Piirtoikkunan koko
  frameRate(30);
  smooth();
  background(255);
  //For-silmukka, jolla luodaan nelikulmiot
  for (int i=0; i<omatNelikulmiot.length; i++) {
    omatNelikulmiot[i]=new Nelikulmio();
  }
}

//Draw-funktio, jonka sisällä kaikki toiminnallisuus tapahtuu
void draw() {
  noStroke(); //Ei reunaviivaa
  fill(255, 25); //Täyttöväri valkoinen, alpha-arvo 25/255
  rect(0, 0, 1000, 1000); //Piirretään osittain läpinäkyvä valkoinen
nelikulmio kuvan päälle, jotta vanhat nelikulmiot peittyvät
  //For-silmukka, jolla piirretään nelikulmiot
  for (int i=0; i<omatNelikulmiot.length; i++) {
    omatNelikulmiot[i].piirra(); //Piirretään nelikulmio
  }
  saveFrame("frames/nelikulmiot_anim_001_####.png");
}

//Luodaan nelikulmion malli eli olio
class Nelikulmio {
  float x1, y1;
  float f1, f2, f3, f4;
  float w1, h1;
  float x1speed, y1speed;

  //Määritellään nelikulmion muuttujien lähtöarvot
  Nelikulmio() {
    w1=random(10, 400); //Nelikulmion leveys
    h1=random(10, 400); //Nelikulmion korkeus
    x1=random(100, 1820); //Nelikulmion x-koordinaatti
    y1=random(-100, 980); //Nelikulmion y-koordinaatti
    y1speed=random(0, .2); //Nelikulmioiden nopeus on jotain välillä
0-0.2 pikseliä/freimi
    f1=random(255);
    f2=random(255);
    f3=random(255);
    f4=random(25);
  }

  //Piirretään nelikulmio
  void piirra() {
    rectMode(CENTER); //Asetetaan nelikulmioiden piirtomodeksi
"CENTER" eli jatkossa niiden koordinaateilla viitataan niiden
keskipisteeseen
    fill(f1, f2, f3, f4); //Täyttöväri sattumanvarainen, alpha-arvo on
jotain välillä 0-25/255
    rect(x1, y1, w1, h1); //Piirretään nelikulmio
    y1=y1+y1speed; //Annetaan y1:lle uusi arvo
  }
}
```

Liite 7 1(2). Animaation 7 koodi

```
Ympyra[] ympyraArr= new Ympyra[100]; //Julistetaan lista, joka ympyrän
viivan instanssit(100kpl)

//Setup-funktio, jonka sisällä määritellään lähtötilanne
void setup() {
  size(2800, 1800); //Piirtoikkunan koko suurempi, jotta jää tilaa
kamera-ajolle Afterissa
  frameRate(30); //Määritellään framerate
  background(255); //Määritellään taustaväri
  smooth(); //Määritellään piirtotilan "anti-aliasing-tilaksi smooth"
  //For-silmukka, jolla luodaan ympyrät
  for (int i=0; i<ympyraArr.length; i++) {
    ympyraArr[i]=new Ympyra();
  }
}

//Draw-funktio, jonka sisällä kaikki toiminnallisuus tapahtuu
void draw() {
  fill(255, 1); //Täyttöväri valkoinen, alpha-arvo 1/255
  noStroke(); //Ei reunaviivaa
  rect(0,0,2800,1800); //
  filter(BLUR,5); //Blurrataan kuvaa arvolla 5
  //For-silmukka, jolla piirretään ja liikutetaan ympyröitä
  for (int i=0; i<ympyraArr.length; i++) {
    ympyraArr[i].piirraYmpyrat(); //Luetaan funktio "piirraYmpyrat"
  }
  saveFrame("framet/ympyrakuva_anim_002_#####.png"); //Tallennetaan
"freimit"
}

//Luodaan ympyrän malli eli olio
class Ympyra {
  //Nimetään muuttujat ja määritellään niiden tyyppi
  float x;
  float y;
  float a;
  float xspeed1;
  float yspeed1;
  float s1;
  float s2;
  float w;

  Ympyra() {
    //Määritellään ympyrän muuttujien lähtöarvot
    x=random(150, 2650);
    y=random(150, 1750);
    a=random(10,200);
    xspeed1=random(-.5, .5);
    yspeed1=random(-.5, .5);
    s1=random(245);
    s2=random(225);
    w=random(.2,5);
  }

  //Piirretään Ympyrä
  void piirraYmpyrat() {
    stroke(s1,s2); //Reunaviiva
    noFill(); //Ei täyttöväriä
    strokeWeight(w); //Viivan paksuus
    circle(x,y,a); //Piirretään ympyrä
```

Liite 7 2(2). Animaation 7 koodi

```
    x=x+xspeed1; // Ympyrän x-koordinaatti saa uuden arvon
    y=y+yspeed1; // Ympyrän y-koordinaatti saa uuden arvon

    //Jos ympyrän koordinaattipiste on liian kaukana kuvan
keskipisteestä, muutetaan sen nopeus sen vastaluvuksi
    if (x>1870) {
      xspeed1=xspeed1*-1;
    }
    if (x<50) {
      xspeed1=xspeed1*-1;
    }
    if (y>1870) {
      yspeed1=yspeed1*-1;
    }
    if (y<50) {
      yspeed1=yspeed1*-1;
    }
  }
}
```

Liite 8 1(2). Animaation 8 koodi

```
PyorivaKolmio[] PyorivatKolmiot=new PyorivaKolmio[80]; //Julistetaan
lista, joka sisältää kolmioiden instanssit
float a; //Luodaan muuttuja, jota käytetään koordinaatiston
pyörittämiseen

//Setup-funktio, jonka sisällä määritellään lähtötilanne
void setup() {
  size(2520, 1480); //Piirtoikkuna isompi, jotta varaa kamera-ajolle
Afterissa
  frameRate(30);
  background(255);
  //For-silmukka, jolla luodaan kolmiot
  for (int i=0; i<PyorivatKolmiot.length; i++) {
    PyorivatKolmiot[i]=new PyorivaKolmio();
  }
}

//Draw-funktio, jonka sisällä kaikki toiminnallisuus tapahtuu
void draw() {
  translate(width/2, height/2); //Siirretään koordinaatiston
nollakohta piirtoikkunan keskipisteeseen
  fill(255, 50); //Määritellään täyttöväriksi valkoinen alpha-arvolla
50/255
  noStroke(); //Ei reunaviivaa
  rect(-1260, -740, 2520, 1480); //Peitetään edelliset "freimit"
piirtämällä niiden päälle osittain läpinäkyviä nelikulmioita
  filter(BLUR, .6); //Blurrataan kuvaa arvolla 0-0.6
  //For-silmukka, jolla piirretään kolmiot
  for (int i=0; i<PyorivatKolmiot.length; i++) {
    PyorivatKolmiot[i].piirra(); //Luetaan funktio "piirra"
    rotate(a); //Pyöritetään koordinaatistoa muuttujan a arvon verran
    a=a+.000002; //Kasvatetaan muuttujan a arvoa
  }

  saveFrame("frames/nousevat_ja_laskevat_viivat_anim_001_####.png");
}

//Luodaan kolmion malli eli olio
class PyorivaKolmio {
  //Nimetään muuttujat ja määritellään niiden tyyppi
  float x1, y1;
  float a, b;
  float lastx, lasty;

  //Määritellään kolmion muuttujien lähtöarvot
  PyorivaKolmio() {
    x1=random(-500, 500);
    y1=0;
    a=random(-1, 1);
    b=random(-1, 1);
  }

  //Piirretään kolmio
  void piirra() {
    stroke(random(20, 245), random(20, 45), random(20, 45));
//Reunaviivan väri
    strokeWeight(random(20)); //Reunaviivan paksuus
    lastx=x1; //Annetaan lastx:lle x1:n edellinen arvo
    lasty=y1; //Annetaan lasty:lle y1:n edellinen arvo
    x1=x1+sin(a)*random(1, 15); //Lasketaan x1:lle uusi arvo
```

Liite 8 2(2). Animaation 8 koodi

```
    //Annetaan b:lle arvoksi -1 tai 1
    if (b>0) {
      b=1;
    } else {
      b=-1;
    }
    y1=y1+b; //Lasketaan y1:lle uusi arvo
    a=a+.1; //Kasvatetaan a:n arvoa
    triangle(x1, y1, lastx, lasty, x1+random(2), y1-random(2));
//Piirretään kolmio
  }
}
```

Liite 9 1(2). Animaation 9 koodi

```
Viivalive[] omat_Viivalive=new Viivalive[60]; //Julistetaan lista,
joka sisältää viivojen instanssit
//Nimetään muuttujat ja määritellään niiden tyyppi
float r=0;
float r1=0;

//Setup-funktio, jonka sisällä määritellään lähtötilanne
void setup() {
  size(3000, 1600); //Isompi piirtoikkuna, jotta jää tilaa kamera-
ajolle Afterissa
  background(255);
  //For-silmukka, jolla luodaan viivat
  for (int i=0; i<omat_Viivalive.length; i++) {
    omat_Viivalive[i]=new Viivalive();
  }
}

//Draw-funktio, jonka sisällä kaikki toiminnallisuus tapahtuu
void draw() {
  filter(BLUR, random(.8)); //"Blurrataan" kuvaa arvolla 0-0.8
  translate(width/2, height/2); //Siirretään koordinaatiston
nollakohta piirtoikkunan keskipisteeseen
  rotate(radians(r)); //Käännetään koordinaatistoa muuttujan r arvolla
  r+=.8; //Kasvatetaan muuttujaa r
  //For-silmukka, jolla piirretään viivat
  for (int i=0; i<omat_Viivalive.length; i++) {
    omat_Viivalive[i].piirra(); //Luetaan funktio "piirra"
  }
  saveFrame("frames/Viivalive_frame_####.jpg"); //Tallentaan "freimit"
}

//Luodaan viivan malli eli olio
class Viivalive {
  float x1, x2;
  float y1, y2;
  float lastx, lasty;
  float s1, s2;
  float a, b;
  float x3, y3;

  //Määritellään viivan muuttujien lähtöarvot
  Viivalive() {
    x1=random(-1300, 1300);
    y1=random(-800, 800);
    x2=random(-1300, 1300);
    y2=random(-800, 800);
    lastx=random(-1300, 1300);
    lasty=random(-800, 800);
    s1=random(5);
    s2=random(150, 255);
  }

  //Piirretään viivat
  void piirra() {
    strokeWeight(s1); //Viivan paksuus
    stroke(s2); //Viivan väri
    x1=x2; //Annetaan x1:lle x2:n edellinen arvo
    y1=y2; //Annetaan y1:lle y2:n edellinen arvo
    a=random(10); //Annetaan a:lle arvo väliltä 0-10
    if (a>5) { //Jos a on suurempi kuin 5, b=30
```

Liite 9 2(2). Animaation 9 koodi

```
      b=30;
    } else { //Muuten b=5
      b=5;
    }
    x2=x2 + random(-b, b); //Annetaan x2:lle uusi arvo eli lisätään
x2:n arvoon jokin luku -5:n ja 5:n väliltä
    y2=y2 + random(-10, 10); //Annetaan y2:lle uusi arvo eli lisätään
y2:n arvoon jokin luku -10:n ja 10:n väliltä
    line(x1, y1, x2, y2); //Piiretään viiva
    x3=x2+random(-20, 20); //Annetaan x3:lle uusi arvo eli lisätään
x2:n arvoon jokin luku -20:n ja 20:n väliltä
    y3=y2+random(-20, 20); //Annetaan y3:lle uusi arvo eli lisätään
y2:n arvoon jokin luku -20:n ja 20:n väliltä
    line(x1, y1, x3, y3); //Piiretään toinen viiva
    strokeWeight(5); //Viivan eli pisteen paksuus
    stroke(s2+50); //Viivan eli pisteen väri
    point(x2, y2); //Piirretään myös piste
    a=random(10); //Annetaan a:lle uusi arvo väliltä 0-10
    if (a>5) { //Jos a on suurempi kuin 5, b=20
      b=20;
    } else { //Muuten b=5
      b=5;
    }
    point(lastx, lasty); //Piirretään vielä yksittäinen piste
    lastx=lastx + random(-b, b); //Annetaan lastx:lle uusi arvo eli
lisätään lastx:n arvoon jokin luku -20:n ja 20:n väliltä
    lasty=lasty + random(-1, 1); //Annetaan lasty:lle uusi arvo eli
lisätään lasty:n arvoon jokin luku -1:n ja 1:n väliltä
  }
}
```

Liite 10. Animaation 10 koodi

```
//Nimetään muuttujat ja määritellään niiden tyyppi
float a=0;
float b=0;
float d=1;
float e=0;

//Setup-funktio, jonka sisällä määritellään lähtötilanne
void setup() {
  size(1920, 1080, P3D); ////Määritellään piirtoikkunan koko ja
"renderer"
  smooth();
  frameRate(30);
  background(255);
}

//Draw-funktio, jonka sisällä kaikki toiminnallisuus tapahtuu
void draw() {
  fill(random(255),2); //Täyttöväri valkoinen, alpha-arvo 2/255
  noStroke();
  rect(0,0,random(1920),random(1080)); //Piirretään eri kokoisia
nelikulmioita vanhan kuvan päälle
  fill(0);
  lights(); //3d-scenen valot päälle
  a=a+.12; //Kasvatetaan muuttujaa a
  b=b+1.2; //Kasvatetaan muuttujaa b
  translate(width/2, height/2, b); //Koordinaatisto keskelle
  rotateY(radians(a)); //Käännetään koordinaatistoa Y-
akselilla(näyttää kuin pallo kääntyisi)
  rotateX(radians(a)); //Käännetään koordinaatistoa X-
akselilla(näyttää kuin pallo kääntyisi)
  filter(BLUR,random(1)); //"Blurrataan" kuvaa arvolla 0-1 pikseliä

  float r=200;
  int total=100;
  //Kaksi for-silmukkaa, joiden avulla piirretään 3D-pallon
muodostavat pisteet. Kyseessä valmis koodinpätkä, jossa hyödynnetään
piitä ja trigonometrisia funktioita.
  for (int i=0; i<total; i++) {
    float lon=map(i, 0, total, -PI, PI);
    for (int j=0; j<total; j++) {
      float lat=map(j, 0, total, -HALF_PI, HALF_PI);
      float x=r*sin(lon)*cos(lat);
      float y=r*sin(lon)*sin(lat);
      float z=r*cos(lon);
      stroke(e); //Viivan eli pisteen väri
      strokeWeight(d); //Viivan eli pisteen paksuus
      point(x, y, z); //Piirretään piste
      d=d+random(-.0003,.0003); //Viivan eli pisteen paksuus vaihtelee
    }
    e=e+.006; //Viivan eli pisteiden väri muuttuu mustasta valkoiseen
  }
  saveFrame("framet/sphere_anim_001_####.png");
}
```