Ville Salminen

# Uuden sukupolven rakenteinen verkko-ohjelmointikieli

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Mediatekniikan koulutusohjelma

Insinöörityö

23.4.2013

Metropolia

| | |
|---|---|
| Tekijä<br>Otsikko<br><br>Sivumäärä<br>Aika | Ville Salminen<br>Uuden sukupolven rakenteinen verkko-ohjelmointikieli<br><br>47 sivua + 1 liitettä<br>23.4.2013 |
| Tutkinto | insinööri (AMK) |
| Koulutusohjelma | mediatekniikka |
| Suuntautumisvaihtoehto | digitaalinen media |
| Ohjaaja | yliopettaja Harri Airaksinen<br>yliopettaja Kari Aaltonen |

Insinöörityössä tutkittiin uuden sukupolven dynaamista verkko-ohjelmointikieltä, Google Dartia, jonka ensimmäinen versio julkaistiin vuoden 2011 lopussa. Tarkoitus oli tutkia uuden sukupolven ohjelmointikielen mahdollisuuksia menestyä nykyisillä markkinoilla. Tutkimuksen avulla haluttiin saada tietää, voitaisiinko JavaScript-ohjelmointikielen rinnalle tuoda toinen dynaamisesti toimiva ohjelmointikieli. Dartia ja JavaScriptia vertailtiin tarkoituksena selvittää ohjelmointikielten vahvuuksia ja heikkouksia.

JavaScriptin ja Dartin välisessä vertailussa havaittiin Dartin pyrkivän selkeyttämään koodin kirjoittamista ja JavaScriptin olevan vaativampi käyttää ja oppia. Dartin heikkouksiin lukeutui sen ikä ja tämänhetkinen toimivuus vain Googlen Chrome-selaimessa, vaikka yksi Dartin liitännäisistä, dart2js, pystyy muuntamaan Dart-koodin JavaScriptiksi, mikäli selain ei tue Dartia.

Työssä toteutettiin useita pieniä sovellusesimerkkejä Dartilla kirjoitettuna. Pohjana käytettiin CSS:llä ja HTML:llä luotuja verkkosivujen runkoja, joiden päälle sovellukset liitettiin tuomaan dynaamisuutta. Työssä luotiin sovelluksia, joissa tutkittiin, kuinka hyvin voidaan vaikuttaa DOM- ja Canvas-elementteihin, ja lisäksi tutkittiin luokan käyttöä. Työssä käytiin läpi datan varastoimiseen tarkoitettuja komentoja, List, Map ja JSON, ja tutkittiin palvelinpään toimintaa yksinkertaisella palvelin-asiakassovelluksella. Sovellukset toimivat hyvin. Lopuksi kirjoitettiin pelisovellus.

Insinöörityön tulosten perusteella Dartin on mahdollista vakiinnuttaa asemansa uuden sukupolven verkkoteknologioiden joukossa, jos useat eri selain- ja laitevalmistajat ovat valmiita liittämään sen osaksi tuotteitaan. Muussa tapauksessa Dart joko lakkaa olemasta tai se toimii yksin Chromessa, mutta muuntaa itsensä JavaScriptiksi tarvittaessa.

| | |
|---|---|
| Avainsanat | Dart, JavaScript, verkko-ohjelmointikieli |

| | |
|---|---|
| Author<br>Title<br><br>Number of Pages<br>Date | Ville Salminen<br>The new generation of structured web programming language<br><br>47 pages + 1 appendices<br>23 April 2013 |
| Degree | Bachelor of Engineering |
| Degree Programme | Media Technology |
| Specialisation option | Digital Media |
| Instructor | Harri Airaksinen, Principal Lecturer<br>Kari Aaltonen, Principal Lecturer |

The purpose of this thesis was to study a new generation dynamic web programming language called Google Dart, the first version of which was released in late 2011. I have also examined the possibilities for improvements in the Dart programming language. I wanted to know if it is possible to bring other dynamically working programming languages than JavaScript to market. Dart and JavaScript were compared to find out their strengths and weaknesses.

When comparing Dart and JavaScript it became clear that Dart is easier to write than JavaScript, as JavaScript felt old and clumsy. The weakness of Dart is its age and distribution and the fact that it is only working on Google Chrome. However, there is a plugin called dart2js, which on request will recode Dart-code to JavaScript.

In this thesis a number of small examples written with Dart were created. I used CSS and HTML to create basic webpages and Dart was merged as part of it. In applications also the following were examined: DOM- and Canvas elements. Data storing systems, such as List, Map and JSON were also examined. The client-server code was also studied to some extent and in the end “Collecting game” was created. Everything worked as intended.

There is a possibility that Dart could stabilize its position amongst the other programming languages. This is only possible if other browser- and device manufacturers are willing to integrate Dart as part of their products. Otherwise, Dart will slowly disappear or stay as part of Google Chrome and use dart2js plugin to change Dart to JavaScript.

| | |
|---|---|
| Keywords | Dart, JavaScript, Web programming language |

## Sisällys

## Lyhenteet

| | |
|---|---|
| ASP | Active Server Pages. Ohjelmointimenetelmä, jota hyödynnetään palvelinpuolella ja käytetään dynaamisen web-sisällön tuottamiseen. |
| CLR | Common Language Runtime. Virtuaalikone, joka hoitaa .NET-ohjelmien koodin tulkitsemisen. |
| CSS | Cascading Style Sheets. Verkkosivun tyylitiedosto, jossa voi määritellä kaiken fontteja, värejä ja koordinaatteja myöten. |
| Dart VM | Dart Virtual Machine eli Dartin virtuaalikone. Ohjelma, jolla voidaan jäjitellä oikeita tietokoneita. |
| DOM | Document Object Model. Alustariippumaton rajapinta, jonka avulla voidaan muokata HTML-sivun rakennetta, tyyliä ja sisältöjä. |
| FTP | File Transfer Protocol eli tiedonsiirtoprotokolla. Mahdollistaa tiedostojen helpon siirtämisen kahden tietokoneen välillä. |
| GML | Game Maker Language. GameMaker-ympäristön oma skriptauskieli. Luotu helpottamaan pelien ohjelmointia 2D-ympäristössä. |
| HTML | Hypertext Markup Language eli hypertekstin merkintäkieli. Voidaan määrittää hyperlinkit ja tekstin rakenne. |
| HTTP | Hypertext Transfer Protocol. Www-palvelimien ja -selainten käyttämä tiedonsiirtoprotokolla. |
| IP | Internetin protokolla. Yksilöi jokaisen verkkoon kytketyn tietokoneen. |
| JDK | Java Development Kit. Varta vasten Javan kehittämiseen tarkoitetut työkalut ja ympäristö. Lähestulkoon sama asia kuin SDK. |
| JSON | JavaScript Object Notation. Luotu helpottamaan monimutkaisten tietokantarakanteiden liikuttelua ja varastointia. |

| | |
|---|---|
| PHP | PHP: Hypertext Preprocessor. Voidaan luoda dynaamisia verkkosivuja ja käytetään palvelinympäristössä. |
| SDK | Software development kit. Sarja erilaisia työkaluja, joilla voidaan luoda haluttu sovellus. Esimerkiksi Unreal Engine. |
| UML | Unified Modeling Language. Graafinen mallinnuskieli järjestelmä- ja ohjelmistokehitystä varten. |
| WWW | World Wide Web eli maailmanlaajuinen verkko. Nykyinen standardi Internetissä. |

# 1 Johdanto

Insinöörityö tehdään Metropolia Ammattikorkeakoululle, ja sen tarkoituksena on selvittää uuden sukupolven verkko-ohjelmointikielten mahdollisuuksista selviytyä nykypäivän markkinoilla. Erityisessä tarkastelussa on viimeksi julkaistu, Googlen kehittämä Dart-ohjelmointikieli. Uusia verkko-ohjelmointikieliä ei julkaista yhtä usein kuin uusia työympäristöjä, jotka perustuvat johonkin olemassa olevaan ohjelmointikieleen. Pyrin myös selvittämään, tarvitaanko verkkokehityksessä enää uusia ohjelmointikieliä, ja samalla selvitän, onko Googlen julkistamalla Dartilla mahdollisuuksia menestyä kilpailussa.

Aluksi raportissa luodaan katsaus jo nykyään olemassa oleviin verkko-ohjelmointikieliin ja käydään läpi Dart-sovellusten kehittämiseen tarvittavia työkaluja. Perinteisesti verkkosivuja on toteutettu käyttämällä HTML-, CSS- ja JavaScript-ohjelmointikieliä. Dartin tarkoitus on tuoda JavaScriptin sijaan vaihtoehtoinen ohjelmointikieli dynaamisen sisällön kirjoittamiseen ja olla selkeämpi ympäristö sovellusten kehittämiseen.

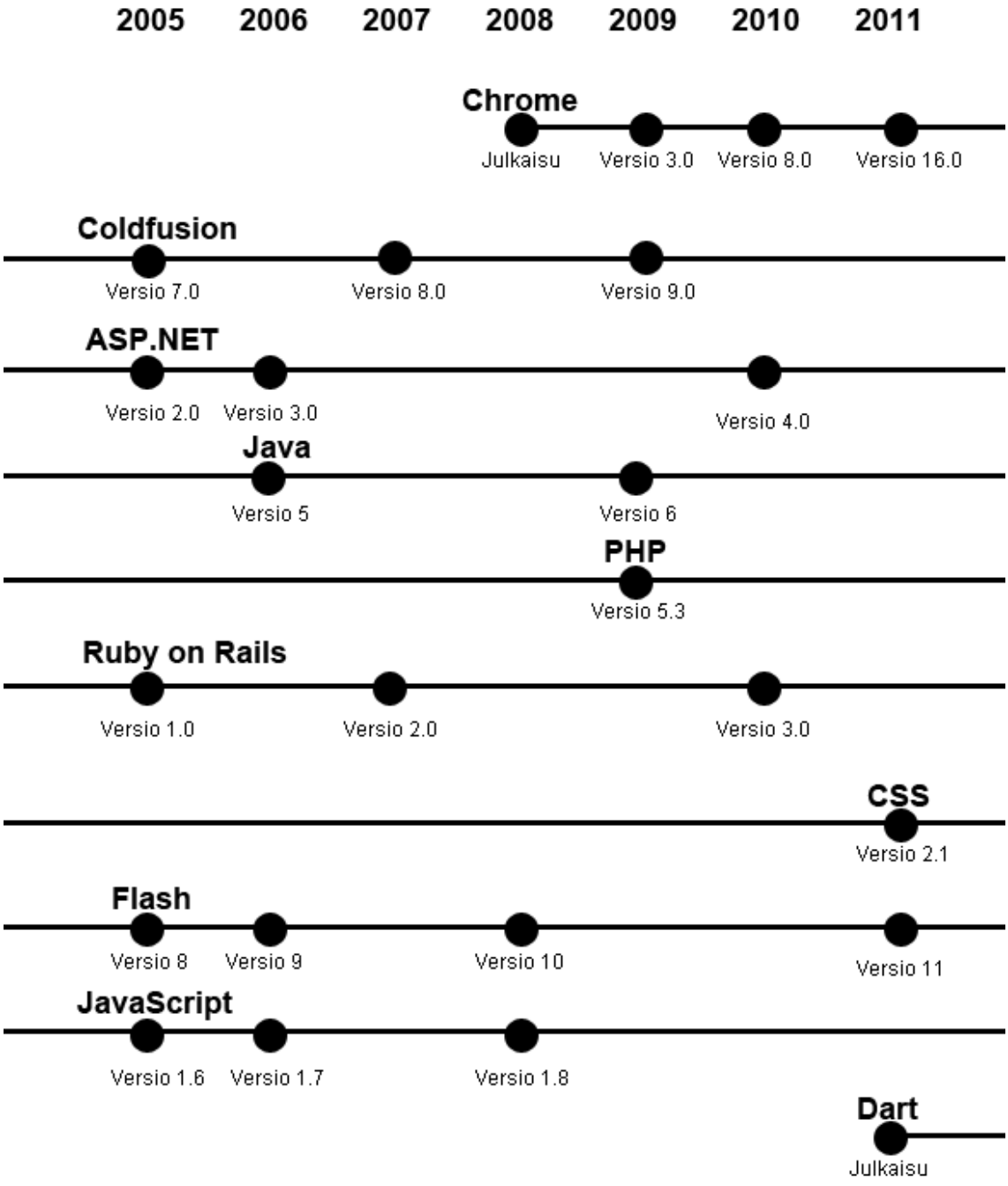
Työssä teen useita pieniä esimerkkisovelluksia hyödyntäen Dartia, ja nämä sovellukset sulautan yksinkertaisen verkkosivun pohjalle. Pyrin havainnollistamaan, miltä Dart-sovellukset ja valmis koodi voivat näyttää. Tutkin myös koodin rakennetta ja sen selkeyttä. Käyn läpi muutamia huomattavia eroja JavaScriptin ja Dartin välillä sekä tutkin kummankin kielen hyviä ja huonoja puolia. Lopuksi pohdin, kannattaisiko pidättäytyä edelleen JavaScriptissa vai pitäisikö Dart hyväksyä vakavasti otettavien verkko-ohjelmointikielten joukkoon tasavertaisena vaihtoehtona.

# 2 Modernien verkkoteknologioiden kehitys vuosina 2005–2011

Internetin kehitys on tuonut mukanaan uusia ohjelmointikieliä, ja nykyisillä markkinoilla onkin paljon vaihtoehtoja, mistä valita. Kuitenkin lähes kaikkien skriptauskielten ja ohjelmointiympäristöjen takana on jokin toinen, vanhempi ohjelmointikieli. Vaihtoehtojen lisääntyminen ei aina tarkoita kokonaisuudessaan uusia ohjelmointikieliä, vaan useita toiseen ohjelmointikieleen liittyviä tuotantoympäristöjä. Tässä tarkasteluun on otettu vuosien 2005–2011 välillä ilmestyneet uudet verkkokehitykseen suunnitellut ohjelmointikielet sekä vanhoja ohjelmointikieliä, joihin on tehty parannuksia.

Erilaisten verkko-ohjelmointikielien valikoima on runsas. Kaikkien mahdollisten sovellusten, työympäristöjen, kehitysalustojen, ohjelmointikielten ja selainten läpikäyminen olisi turhaa. Kuitenkin kattava otanta nykyaikaisempia ja käytetyimpiä verkko-ohjelmointikieliä antaa hyvän kuvan tämänhetkisestä tilanteesta ja markkinoilla vallitsevasta ideologiasta.

Selainkehitys on nopeutunut tämän vuosituhannen puolella, mutta kovin monta uutta selainta ei markkinoille ole tullut. Ainoa uusi tulokas on ollut Googlen kehittämä Chrome, joka julkaistiin vuonna 2008 [1]. Kuvassa 1 on esitetty vuodesta 2005 vuoteen 2011 aikajanana verkko-ohjelmoinnin kehitys ja vertailun vuoksi Google Chromen synty ja sen mahdollinen vaikutus ohjelmointikielten kehitykseen. Kaikkia mahdollisia verkko-ohjelmointikieliä en ole ottanut mukaan, sillä osa niistä on jo jäänyt kehityksestä jälkeen. Myös työkaluja, joilla voi tuottaa helpommin sovelluksia jollakin ohjelmointikielellä, on jätetty pois. Kuvan 1 aikajanalle on merkitty tärkeimmät muutokset kunkin ohjelmointikielen versioinnissa.




Kuva 1. Verkkoteknologioiden kehitys vuosina 2005–2011 (2; 3; 4; 5; 6; 7; 8; 9; 10; 11; 12; 13).

Kuvasta 1 voi huomata, että eri versioita ohjelmointikielistä on ilmestynyt yhä harvemmin viime vuosien aikana. Samoin on käynyt uusien ohjelmointikielien kohdalla. Tämä voi selittyä sen kautta, että kieli on saatu niin sanotusti ”valmiiksi” ja on keskitytty paikkaamaan virheitä ja tietoturvaa vaarantavia aukkoja. Myöskään uusia verkkoteknologioita ei ole syntynyt kovin monta viimeisen vuosikymmenen aikana. Google Chromen

nopea menestys markkinoilla ja kolme vuotta sen julkistamisen jälkeen ilmestynyt Dart-ohjelmointikieli tiukentavat kilpailua entisestään niin selainten kuin ohjelmointikielten välillä. Tässä tapauksessa kilpailu ei mielestäni ole pahasta, sillä sen tuoma paine pakottaa ohjelmointikieli- ja selainvalmistajat parantamaan tuotteitaan, tekemään niistä helpompia käyttää ja minimoimaan virheiden määrää.

Palvelinpuolen kehitys

Sanalla palvelin viitataan siihen sovellukseen tai tietokoneeseen, joka ylläpitää haluttua palvelua. Palvelimia voivat olla esimerkiksi verkko- tai pelipalvelimet. Palvelinpuolen ohjelmointikielissä pääpaino on siinä, mitä taustalla tapahtuu, eikä niinkään siinä, mitä käyttäjä näkee. Palvelinpuolen ohjelmointikielet voivat esimerkiksi hoitaa tietojen välittämistä palvelimelta toiselle tai olla yhteydessä tietokannan kanssa. Käyttäjä ei näe palvelinpuolen koodia, vaan asiat hoidetaan asiakaspuolen koodikielillä. Seuraavassa esitellään yleisimpiä palvelinpuolen ohjelmointikieliä ja työympäristöjä.

- ASP.NET – Microsoftin luoma ohjelmointiympäristö. Sen avulla voidaan rakentaa dynaamisia verkkosivuja ja -palveluja käyttäen HTML:ää, CSS:ää, JavaScriptiä ja palvelinpuolen skriptauskieliä. [14.]
- Coldfusion – Adobe Systemsin vuonna 2005 ostama verkkosovellusten kehittämisympäristö, jonka kehittivät Jeremy ja JJ Allaire vuonna 1995 [15].
- Java – Sun Microsystemsin (joka nykyään tunnetaan Oracle Corporationa) kehittämä laitteistoriippumaton oliopohjainen ohjelmointikieli. Java on käytössä niin mobiililaitteissa kuin verkkoympäristöissäkin. [16.]
- PHP – Laajalti käytetty vapaan lähdekoodin verkko-ohjelmointikieli, jota käytetään yleensä osana palvelinympäristöä. Mahdollistaa dynaamisten verkkosivujen luonnin. [17.]
- Ruby on Rails – Ruby-ohjelmointikieleen pohjautuva avoimen lähdekoodin sovelluskehys, jonka on kehittänyt David Heinemeier Hansson vuonna 2003 [18].

Asiakaspuolen kehitys

Asiakas on se tietokone tai ohjelma, joka ottaa yhteyttä palvelimeen, ja palvelin lähettää asiakkaalle vastauksen. Asiakaspuolen ohjelmoinnissa keskitytään siihen, miltä verkkosivut näyttävät käyttäjän silmin. Siinä määritellään sivuston rakenne, tyylitellään elementit ja lisätään mahdollinen dynaamisuus. Asiakaspuolen kehityksen tulisi olla

mahdollisimman selkeää, jotta käyttäjäkokemus oli mahdollisimman hyvä. Seuraavassa esitellään yleisimpiä asiakaspuolen kehityksessä käytettäviä ohjelmointikieliä ja kehitysympäristöjä.

- CSS – Tyylimääritelmä, joka toimii yleensä omana tiedostonaan ja jolla voidaan muokata verkkosivujen ulkomuotoa.
- HTML – Verkkosivujen määrittelyyn suunniteltu merkintäkieli. Koostuu tageista, joilla määritellään sivun sisältö.
- Flash – Adobe Systemsin luoma kehitysympäristö, jolla voidaan luoda monipuolisia multimediasovelluksia niin mobiilialustoille kuin verkkosivuille. Käyttää omaa Action Script 3.0 -skriptauskieltä. [19.]
- JavaScript – Verkkosivuille suunniteltu skriptauskieli dynaamisen toiminnallisuuden lisäämiseksi. Tunnettiin alkujaan LiveScriptinä, ja kehittäjänä toimi Brendan Eich. [20.]
- Silverlight – Microsoftin kehittämä Adobe Flashin kaltainen kehitysympäristö [21].
- Dart – Googlen kehittämä dynaaminen ohjelmointikieli, joka mahdollistaa sekä asiakas- että palvelinpuolen sovellusten kirjoittamisen [22, s. 2–3].

JavaScript ja Dart kuuluvat samaan luokkaan: molemmat ovat dynaamisia ohjelmointikieliä eikä niiden käyttö ole rajoittunut tietylle alueelle, kuten esimerkiksi Adoben Flashin, joka toimii vain omassa elementissään. JavaScript on tullut tunnetuksi yhtenä suosituimmista ja yleisimmistä ohjelmointikielistä lisätä verkkosivuille dynamiikkaa. Suosion myötä sen ympärille on rakentunut lukuisia lisäosia, joiden ansiosta sen käytöstä on tullut entistäkin helpompaa. Seuraavassa esitellään joitakin yleisimpiä JavaScriptin liitännäisiä.

- Ajax – Kehittäjänä Jesse James Garrett. Mahdollistaa uuden tiedon lataamisen ja esittämisen ilman, että koko sivua tarvitsee uudelleen hakea. Ei ole erillisesti ladattava liitännäinen, mutta teknisesti katsoen mullistava keksintö. [23.]
- jQuery – Kaikista suosituin JavaScriptin runko. Mahdollistaa DOM-elementtien helpon muuntamisen CSS:n avulla ja tarjoaa paremmat puitteet luoda käyttöliittymää. [23.]
- Prototype – Tarjoaa yksinkertaisen käyttöliittymän, jonka avulla voidaan kirjoittaa monia yleisimpiä komentoja helposti. Tuo myös luokat ja perimän. [23.]
- MooTools – Runko, joka mahdollistaa yleisten JavaScript-sovellusten teon. Sisältää myös efekti- ja animaatiofunktiot. [23.]

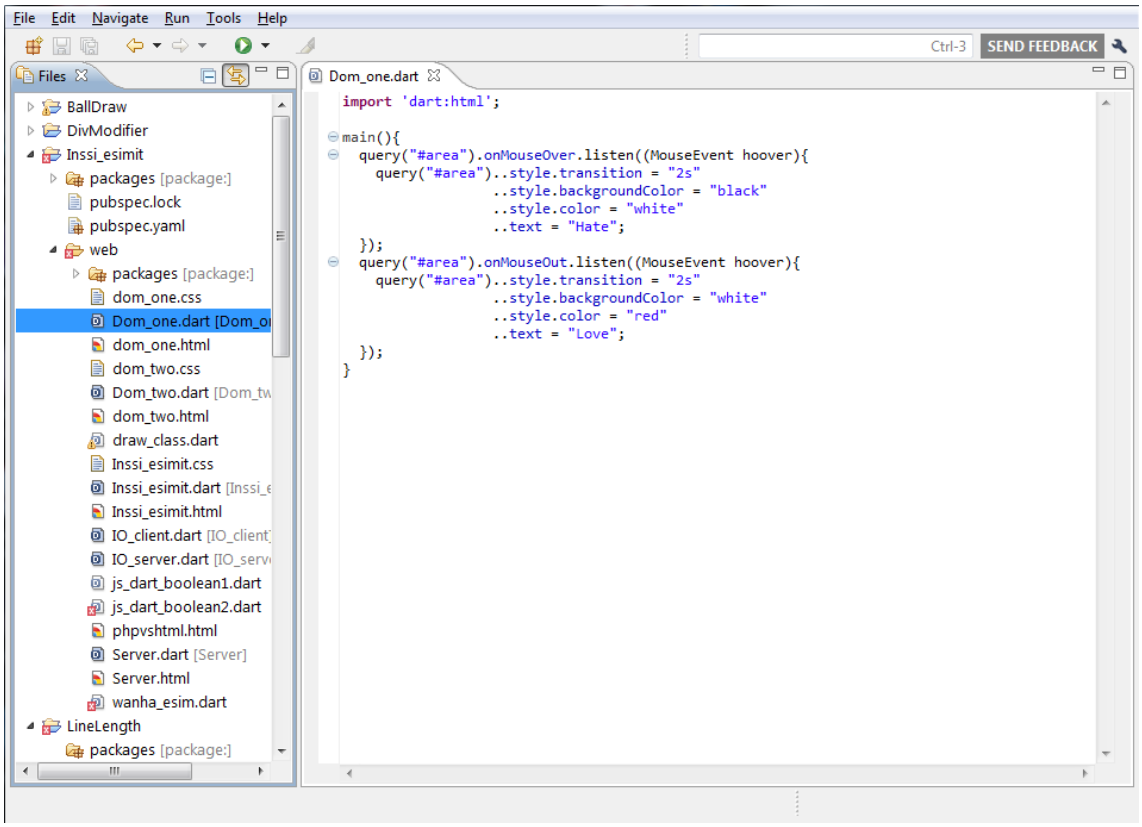
# 3 Rakenteinen selainohjelmointikieli Dart

## 3.1 Dart-verkko-ohjelmointikieli

Dart on Googlen julkaisema uusi verkko-ohjelmointikieli. Sen tarkoituksena on tehdä verkkosovellusten kirjoittamisesta helpompaa ja tarjota aloittelijaystävällinen ympäristö, mutta myös samalla tehdä sovellusten ylläpidosta ja päivittämisestä helpompaa kuin se tällä hetkellä on. [24.] Dart julkistettiin vuoden 2011 lokakuussa GOTO-konferenssissa [25].

Dartin on tarkoitus tuoda markkinoille haastaja JavaScriptille. JavaScript on kätevä kieli tehdä pieniä verkkosovelluksia, mutta isommissa sovelluksissa testaamisesta ja virheiden korjaamisesta tulee todella raskasta. Dartin on tarkoitus tuoda tähän ongelmaan ratkaisu. Googlen mukaan Dart on vapaan lähdekoodin hanke, jolla voi rakentaa moderniin verkkoon laajoja ja monimutkaisia sovelluksia, joilla on parempi suorituskyky. [24.] Googlessa työskentelevän Mark S. Millerin mukaan ”Dartin tavoite on korvata JavaScript täysin ja olla samalla verkkokehityksen Lingua franca avoimessa verkkoympäristössä.” [26.]

Dartissa on laaja valikoima erilaisia työkaluja ja ominaisuuksia. Sivulta https://www.dartlang.org/downloads.html saa ladattua viimeisimmän paketin, jonka mukana tulee paljon työkaluja, kuten Dart Editor, Pub, dart2js, Dart VM ja Dartium. Paketti on heti käyttövalmis, ja sen kaikki tiedostot tarvitsee vain purkaa haluttuun kohteeseen. Dart Editor on hyvin samankaltainen ulkoasultaan kuin Eclipse, joka on vapaan lähdekoodin sovelluskehitysympäristö. Dart Editor on samanlainen kehitysympäristö, jolla voidaan kirjoittaa, muokata ja suorittaa sovelluksia. Editori ei ole pelkästään Darttia varten, vaan sillä voi myös muokata muun muassa HTML:ää sekä CSS:ää. [27.] Kuvassa 2 on näkymä DartEditorista.




Kuva 2. Näkymä DartEditorista.

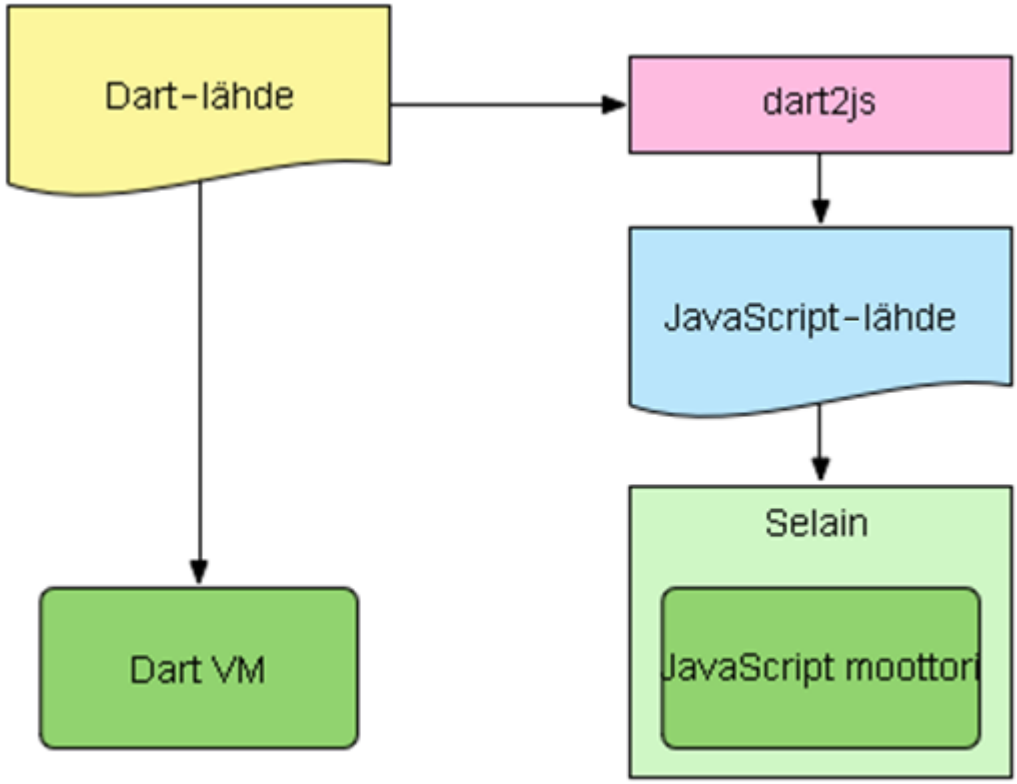
Kuvassa 2 on esillä työkalurivi ylhäällä, kansionäkymä vasemmalla ja työstettävä tiedosto oikealla. Jokainen edes vähän Eclipseä käyttänyt sisäistää DartEditorin toiminnan nopeasti, sillä ne muistuttavat paljon toisiaan niin ulkonäöllisesti kuin toiminnallisestikin.

Pub on Dartin paketinhallintatyökalu, jonka kautta käyttäjä voi hallita pakettejaan. Paketti voi sisältää kuvaa, ääntä ja dokumentointia, mutta pääpaino on hallita kirjastoja. Eri kirjastoilla on eri tarkoituksia riippuen siitä, mihin niiden ohjelmoija on ne tarkoittanut. Esimerkiksi jokin kirjasto voi auttaa ohjelmoijaa tarjoamalla helpon tavan soittaa ääntä ja jokin toinen kirjasto tarjoaa helpon tavan muodostaa yhteyden tietokantaan. Paketteja voi julkaista missä vain, mutta Dartin kehittäjät toivovat, että ne julkaistaisiin myös osoitteessa pub.dartlang.org. [28.] Pub-työkaluja pääsee käyttämään kuvassa 2 näkyvän DartEditorin työkaluriviltä kohdasta ”Tools”.

Dart2js on DartEditorin liitännäinen, joka muuntaa Dart-koodin JavaScriptiksi. Koska Google Chrome on ainoa selain, joka tukee Dartia, tällainen liitännäinen on hyvä olla olemassa. [29.] DartEditorin valikosta ”Tools” avautuu lista, josta löytyy vaihtoehto

”Generate JavaScript”. Tämä toimenpide käynnistää automaattisesti dart2js-liittännäisen ja tekee Dart-tiedostosta JavaScriptiä.

Dartium on Chromiumiin perustuva selain, jossa on valmiina Dart VM. Muun muassa Google Chrome -selain perustuu Chromiumin lähdekoodiin. Dartium on teknisesti vasta alkuvaiheessa, ja se saattaa sisältää suuria turvallisuusriskejä, joten ei ole suositeltavaa, että sitä käytetään muuhun kuin Dartin testaamiseen. [30.] Dart VM ymmärtää kirjoitetun Dart-koodin ja pystyy tuomaan sovelluksen käyttäjän näkyville. Ilman Dart VM:ää aktivoituu dart2js, kuten kuvasta 3 käy ilmi. Dart2js muuntaa Dartin JavaScriptiksi, ja näin selain pystyy tulkitsemaan koodia.




Kuva 3. Dartin toimintaperiaate selaimella avattaessa [31].

Dartin mukana tulee monia valmiita kirjastoja, joita voi hyödyntää sovelluksissa. Muutama yleisesti käytettävä on seuraavaksi esillä.

- Core library, jossa on kaikki yleisimmät datan rakenteet ja toiminnallisuudet. [27.]

- HTML library, jossa on liitännät HTML5 DOMiin. Tämä kirjasto kehittyy jatkuvasti HTML5-standardien kanssa. [27.]
- I/O library, jossa on palvelinpuolen tuki. Mahdollistaa tietojen kirjoittamisen ja lukemisen sekä HTTP-palvelimien ylläpidon. [27.]
- Isolate library, jonka avulla pyritään tekemään Dart-sovelluksista suojatumpia [27].
- JSON library, joka mahdollistaa tiedon pakkaamisen JSON-koodatuksi tiedoksi [27].
- Math library, joka sisältää yleisimmät matemaattiset toiminnallisuudet, kuten sin, cos, max, min ja pii [27].

Kathy Walrath ja Seth Ladd ovat kirjoittaneet lyhyen ja tiiviin teoksen ”What is Dart”, jossa esitellään Dartin tavoitteita ja perustellaan sen olemassaoloa. He kuuluvat Googlen kehityssuhteista vastaavaan tiimiin. Dart on kehitetty heidän mukaansa silmällä pitäen laajoja sovelluskokonaisuuksia. Lisäksi Dart on avoimen lähdekoodin projekti. Koodin kirjoittamisen tulisi laajoissakin sovelluksissa olla yhtä helppoa kuin pienissä. Dart taipuu helpommin paljon monimutkaisempien verkkosovellusten tekoon kuin esimerkiksi JavaScript. Pääpainona onkin ollut tehdä mahdollisimman selkeä ohjelmointikieli, jonka päivittäminen ja ylläpito eivät olisi aikaa vieviä. [22, s. 1.] Tietenkin se, onko koodi helppohoitoista vai ei, riippuu hyvin paljon ohjelmoijan tiedoista ja taidoista kirjoittaa muunneltavaa koodia, jota voi muokata helposti jälkeenpäin.

”What is Dart” -teoksen esimerkeistä Dart on muuttunut entistä yksinkertaisempaan suuntaan. Esimerkiksi DOM-elementteihin vaikutettaessa ei tarvita enää ”document.query” -alkuista aloitusta vaan pelkkä ”query” riittää. Dartilla koodaaminen ei rajoitu vain sen omaan editoriin, vaan millä tahansa tekstieditorilla voi kirjoittaa Dart-koodia. Ennakoiva tekstinsyöttö oli pitkään DartEditorin oma ominaisuus, mutta vuonna 2012 ilmestynyt Dart-liitännäinen mahdollistaa nyt Eclipsen käytön Dartin koodaamisessa [32]. Nykyisellään DartEditor on vielä erittäin huono antamaan käyttäjälle apua koodaamiseen, ja pahimmassa tapauksessa koodia saa kirjoittaa monta kymmentä riviä, ennen kuin yhtäkään vihjettä saa esille.

Jos haluaa kirjoittaa Dartilla sovelluksen ja julkaista sen verkossa, se kannattaa kääntää JavaScriptiksi. Toistaiseksi ainoastaan Google Chrome pystyy ymmärtämään Darttia. Dart on kuitenkin vasta nuori kieli, ja se tähtää aikaisessa vaiheessa isoille markkinoille. Sen menestyminen vaatii vielä paljon työtä. Alussa siinä oli vain kolme kirjastoa:

Core, HTML ja IO. Nykyään kirjastoja on reilusti yli kymmenen ja lisää on mitä todennäköisimmin tulossa siinä missä vanhoihin kirjastoihin lisätään uusia ominaisuuksia.

## 3.2 JavaScript ja Dart

Kaikki asiat, jotka on JavaScriptilla mahdollista toteuttaa, onnistuvat myös Dartilla jopa helpommin. Omien tutkimusteni perusteella Dart on ottanut paljon vaikutteita JavaScriptistä, mutta määrittelee tiettyjä asioita selkeämmin kuin JavaScript. Esimerkkikoodissa 1 on yksinkertainen if-lauseke, jossa on käytetty Booleania. Boolean ilmaisee, onko jokin tosi (true) vai epätosi (false).

```
***js_dart_boolean1.dart
var nimi = "Ville";
if (nimi){
  //Tämä tulostuu JavaScriptissä, mutta ei Dartissa
  print('Sinulla on nimi!');
}
```

Esimerkkikoodi 1. Booleanin käyttäytyminen Stringin kanssa.

Suoritettaessa koodia esimerkkikoodin 1 tapauksessa JavaScriptinä tulostuu teksti ”Sinulla on nimi!”. Sen sijaan Dart ei tunnista komentoa. Tämä johtuu siitä, että JavaScript katsoo nimen olevan non-null-objekti ja näin ollen yhtä kuin ”true”. JavaScript ei ymmärrä, että kyseessä on jokin muu kuin Boolean. Dart puolestaan katsoo tyypin, joka tässä tapauksessa on String, eikä haluttu Boolean. Koska if-lause ei täytä yhtäkään ehtoa, Dart ohittaa kohdan. [22, s. 18.]

Esimerkkikoodissa 2 on esitetty Booleanin käyttöä ykkösten (1) ja nollien (0) kanssa. Yleisesti on tiedossa, että joissain tapauksissa voidaan ajatella, että 0 = false ja 1 = true. Tämä ei kuitenkaan pidä aina Booleanin kanssa paikkaansa, kuten esimerkkikoodista 2 käy ilmi.

```
***js_dart_boolean2.dart
if (1){
    print('JavaScript tulostaa tämän, koska se luulee että 1 = true');
  }else{
    print('Dart tulostaa tämän production-tilassa.');
  }
```

Esimerkkikoodi 2. Booleanin käyttäytyminen ykkösen ja nollan kanssa.

Esimerkkikoodissa 2 JavaScript olettaa, että 1 on yhtä kuin true, eli jos if-lause on tosi, luetaan seuraava rivi. Tämä saattaa joissain tilanteissa aiheuttaa sekaannusta. Dart puolestaan vaatii, että Boolean on Boolean. 1 on numero, joten näin ollen lauseke ”if(1)” ei ole true eikä false, joten Dart ohittaa kohdan. Jos sovellusta suoritetaan Dartissa checked-tilassa, Dart antaa siitä virheilmoituksen kummankin esimerkin tapauksessa. [22, s. 18.] Checked-tilassa Dart on tarkkana tyypityksen kanssa ja ilmoittaa virheestä välittömästi estäen sovelluksen käynnistymisen.

JavaScript on kuitenkin laajemmin levinnyt ohjelmointikieli kuin Dart. Jokainen nykyaikainen markkinoilla oleva selain tukee JavaScriptiä, kun taas vain Google Chrome tukee Dartia. Jotta Dartista tulisi menestys, se vaatisi useiden eri selainvalmistajien mukaan tulemista. Tällä hetkellä se aika epätodennäköiseltä, sillä Microsoft on myös kehittämässä omaa ohjelmointikieltään, TypeScriptiä [6]. Tämänhetkiset ennusteet näyttävätkin siltä, että Dartista tulisi vain Google Chromessa toimiva ohjelmointikieli. Ongelmaa tämän ei kuitenkaan pitäisi muodostaa, sillä Dartin työkaluissa oleva dart2js-liitännäinen tarjoaa siihen ratkaisun.

Dartin vahvuudet

Christian Grobmeier ylistää blogissaan Dartin tuomaa selkeyttä ja luettelee kymmenen syytä, miksi Dart on paljon ”coolimpi” ohjelmointikieli kuin JavaScript [33]. Toisessa blogissa taas Peter-Paul Koch, mobiilialustastrategisti, konsultti ja ohjaaja, pitää itsestäänselvyytenä, että Dartilla ei ole mitään mahdollisuuksia menestyä ja se on näin ollen turha ohjelmointikieli. Grobmeier keskittyy käymään läpi asioita, joita Dart tekee paremmin kuin JavaScript, kun taas Koch keskittyy perustelemaan JavaScriptin paremmuutta sen levinneisyydellä [34].

JavaScript käyttää useita eri false-arvoja. Esimerkiksi false, null ja NaN voidaan kaikki luetella false-arvoiksi. Dart puolestaan käyttää vain yhtä false-arvoa, ja se on false. [33.] Ajatellaan tilanne, jossa on String arvolla ””, eli ei mitään. JavaScript tulkitsee tämän falseksi, vaikkei se välttämättä olisikaan tarkoitus. Väärällä tavalla aseteltu if-lause saattaa tämän ansiosta toimia oikein, vaikka se toimisikin väärin. Tämä puolestaan voi johtaa ohjelman kehittymisen myötä virheisiin, joita saa hakea pahimmillaan useita tunteja. Toisaalta, mikäli kyseessä on kokenut JavaScriptin koodaaja, tämä asia on varmaan tullut tutuksi eikä vaadi sen suurempaa huomiointia. Aloittelijoiden kannalta

tämä taas on huono asia, sillä he eivät välttämättä ymmärrä sitä ja kuluttavat paljon aikaa ongelman selvittämiseen.

Dartissa on mahdollisuus tyypittämiseen, mutta se ole pakollista. Tämä mahdollistaa tiettyjen arvojen tyypittämisen halutuksi ilman, että ne pitäisi ensin liittää johonkin arvoon. JavaScript hoitaa asian seuraavasti: Luodaan arvo X, joka on oletuksena undefined, eli ei mitään. Kun tälle laittaa esimerkiksi arvon 4, muuttuu X numeroksi, ja taas määrittelemällä arvo true muuttuu X Booleaniksi. Tätä samaa tekniikkaa voi hyödyntää myös Dartin kanssa. [33.] Tyypittämättömiä arvoja voidaan helposti vaihtaa tyypistä toiseen, toisin kuin tyypitettyjä. Esimerkiksi jos on arvo ”var X”, sille voidaan antaa arvo ”World”, joka on String. Myöhemmin jos halutaan, ilman erillisiä koodin muunnoksia voidaan antaa X:lle arvo 10 tai true. X:ää ei ole siis sidottu tiettyyn tyyppiin. Toinen mahdollinen tapa olisi antaa arvo ”String X”, jolloin X pystyisi käsittelemään vain tekstiä. Tyypittäminen jakaa mielipiteitä vahvasti. Jotkut ovat sitä vastaan, sillä se vaikeuttaa koodaamista, kun taas toiset pitävät sitä hyvänä asiana, sillä se tuo enemmän selkeyttä koodiin. Uskallan väittää, että riippumatta käyttäjästä tämä on vain hyvä asia. Sitä ei ole pakko käyttää, mutta se on silti hyvä olla olemassa.

JavaScriptin parissa saa työskennellä monen eri elementin kanssa. Rakenteiden kautta pitää valita haluttu elementti, tai esimerkiksi kuva, jota haluaa muokata. Ei voida vain valita elementtiä ja muokata sitä. JavaScriptissa onkin varsin kattavasti tähän tarkoitukseen luotuja komentoja. Dart puolestaan on tuonut tähän ratkaisun. Se sisältää kaksi komentoa: elem.query(’#foo’) ja elem.queryAll(’.foo’), joilla voi suoraan vaikuttaa haluttuun DOM:iin. [33.]

JavaScriptissä on myös joitakin outoja ominaisuuksia, kuten arrayn luominen. Esimerkiksi luodaan array ”var array1” ja annetaan sille arvo ”new Array(1,2,3);”. Tästä muodostuu array, jossa on 3 elementtiä: 1, 2 ja 3. Yksi JavaScriptin erikoisuuksista tulee esille, jos annetaankin arvona ”new Array(3);”. Tästä ei synny yhtä elementtiä arvolla 3, vaan kolme elementtiä arvoilla ”undefined”. Uusien arvojen luonnissa tulee myös olla tarkkana. Jos luo kaksi arvoa, esimerkiksi ”var student” ja ”teacher”, näistä kahdesta ”student” on paikallinen arvo mutta ”teacher” on taas globaali. Tässä piilee se vaara, että jos koodaaja on unohtanut ”var”-määritelmän, hän on luonut huomaamattaan globaalin arvon. Tämä saattaa aiheuttaa ongelmia, mikäli on olemassa toinen samanlainen arvo. [33.] Pienissä sovelluksissa tällainen ei ole vakavaa, mutta suurissa tuotannoissa pienistä virheistä voi koitua monen tunnin korjaustoimenpide.

Yhteenvetona Grobmeier sanoo, että hän ei näe mitään, mitä ei voisi tehdä Dartin avulla helpommin kuin JavaScriptillä. Ammatikseen JavaScriptiä koodaavat eivät välttämättä saa Dartista mitään uutta, mutta ei sitä välttämättä heille ole tarkoitettukaan. [33.] Omien tutkimusteni perusteella voin yhtyä Grobmeierin mielipiteeseen. Kun vertaillaan JavaSciprtiä ja Dartia, voidaan huomata, että Dart on selvästi selkeämpi. Tottuneelle koodaajalle JavaScript on todennäköisesti helpompaa luettavaa, mutta vasta-alkajan näkökulmasta Dart vaikuttaa paljon selkeämmältä. Vielä Dartilla ei voi tehdä mitään kovin suurta, teoriassa, sillä sen rakenne muuttuu jatkuvasti. Mikään este tämä ei ole, mutta jatkuvat lähdekoodin muutokset saattavat tehdä suuresta työstä hetkessä vaikeasti korjattavan.

Dartin heikkoudet

Dartia vastustavia koodareita löytyy internetistä lähes yhtä paljon kuin Dartia kannattavia. He ovat suurimmaksi osaksi kokeneita JavaScript-ammattilaisia, joista osa todennäköisesti käyttää sitä työkseen. Grobmeierin [33] mainitsemia hyviä puolia vastaan löytyi myös henkilöitä, jotka eivät pidä ajatuksesta, että vanhaan hyväksi havaittuun tapaan kirjoittaa koodia puututaan millään tavalla. Aika pitkälle on kyse siitä, miten henkilö on tottunut koodia kirjoittamaan ja mikä on hänelle luonnollisinta. Peter-Paul Kochin [34] artikkeli esittää, miksi hänen mielestään Dart on jo hävinnyt taistelussa JavaScriptiä vastaan. Ensinnäkin JavaScript on jo olemassa ja se on levinnyt miljardeihin laitteisiin. Se on vakiinnuttanut asemansa yhtenä verkkoteknologian standardeista. Tunnetusti verkosta on vaikea poistaa mitään uusien innovaatioiden tieltä. Esimerkiksi Adobe Flash, vanha ja äärimmäisen raskas teknologia, saa nykypäivän kannettavat tietokoneet ylikuumenemaan helposti, mutta sen poistaminen ei ole kuitenkaan helppoa. Jos tällainen teknologia vedettäisiin pois, se tarkoittaisi useiden tuhansien verkkosivujen toimimattomuutta. [34.] Vaikka Googlella on aikomus tuoda Dart markkinoille JavaScriptin korvaajana, en itsekään oikein usko sen tapahtuvan.

Kun luodaan uutta ohjelmointikieltä, se ei voi automaattisesti saavuttaa tiettyä asemaa verkkoteknologioiden kehityskaaressa. On mahdollista, että teknologia osoittautuu loppujen lopuksi täysin hyödyttömäksi ja se katoaa nopeasti. Dart yrittää tällä hetkellä löytää rakoa markkinoilta, mutta yksin se ei tule pärjäämään. Jotta sillä olisi realistisia mahdollisuuksia menestyä, pitäisi muidenkin selain- ja laitevalmistajien kuin Googlen ottaa Dart osaksi tuotteitaan. Muilla kuin Googlella ei vielä näytä olevan kiinnostusta

liittää Dartia osaksi kaikkia selaimia. Google Chromen markkinaosuus on noin 24 % eli noin yksi neljäsosa selainten markkinaosuuksista. [34.]

Koch [34] ihmettelee myös, miksei Google ole vielä julkistanut Dartia Androidille. Googlella on Kochin mukaan hallussaan noin 20 % puhelinmarkkinoista, ja hän ihmettelee, miksei mobiilipuolelle ole tehty mitään kehitystä. [34.] Toisaalta tämä on ihan ymmärrettävää, sillä Dart kehittyy jatkuvasti ja Google pyrkii pitämään Androidin mahdollisimman puhtaana, vaikkakin sen eri versiot ovat pirstaloittaneet tarkoitusta. Uusimmat älypuhelimet kyllä pystyvät hyödyntämään JavaScriptiä, joten uskoisin, että tämä tyhjiö täyttyy lähitulevaisuudessa.

# 4 Käyttöliittymän hallinta Dartissa

## 4.1 Dartin versionhallinta

Dart on uusi ohjelmointikieli. Se kehittyy koko ajan ja muuttuu nopeasti. Insinöörityötä tehdessäni sain lähes joka toinen viikko korjata koodeja uusien määritteiden myötä. Voi olla, että kun kaikki on valmista, yksikään esimerkeistä ei toimi ilman uudelleen ohjelmointia. Vaikka tämä kuulostaakin karulta, en pidä sitä huonona asiana. Moni uusi ominaisuus on selkeyttänyt kieltä entisestään ja tehnyt sen kirjoittamisesta helpompaa. Esimerkkikoodissa 3 on esiteltynä kaksi toimintoa, joista ensimmäinen ei toimi enää.

```
***OldVsNew.dart
//Vanhentunut
query("#canvas").on.mouseDown.add((MouseEvent e){
  //Haluttu toiminto
});

//Uusi
query("#canvas").onMouseDown.listen((MouseEvent e){
  //Haluttu toiminto
});
```

Esimerkkikoodi 3. Vanha ja uusi tapa suorittaa ”hiiren painallus” -toiminto

Vanhentuneesta tavasta esimerkkikoodissa 3 näkyy sen monimutkaisuus. Ensin pitää määritellä, mikä toimii toiminnon laukaisijana eli triggerinä. Se on esimerkin tapauksessa canvas-elementti. Tämän jälkeen halutaan tietää, mitä pitää tehdä, jotta toiminto käynnistyy. Ensin valitaan ”on”, josta pitää valita ”mouseDown” ja vielä lopuksi ”add”.

Uudistettu malli tuo selkeyttä ja yksinkertaisuutta. Ensin valitaan samoin kuin vanhentuneessa tavassa, mikä toimii triggerinä. Tämän jälkeen käyttäjän urakkaa on helpotettu valmiilla ”onMouseDown”-vaihtoehdolla ja valintojen listaa on karsittu niin paljon, että seuraavaksi luonnollinen valinta on ”listen”. Kuuntelija (listener) kuuntelee ja odottaa, milloin toiminto tapahtuu, ja alkaa sitten suorittaa koodia eteenpäin. Mielestäni ”add” ei puolestaan antanut tästä toiminnosta selkeää kuvaa. Käsitin, että se on lisäämistä, mutta en aina komennoilla halunnut lisätä mitään, joten se aiheutti pientä sekaannusta.

Valinta sen välillä, teenkö yhden suuren esimerkin vai muutaman pienemmän, ei ollut vaikea. Koska kieli on jatkuvasti kehittyvä, yksinkertaiset esimerkit ovat paljon helpompia korjata toimiviksi kuin suuret sovellukset. Tämän työn aikana olen joutunut moneen kertaan muuttamaan koodia, parhaimmillaan kaksi kertaa kuukaudessa. Suuren luokan sovellusta tässä vaiheessa ei vielä kannata tehdä. Kuitenkin pienet sovellukset antavat mielestäni lähes poikkeuksetta paremman kuvan ohjelmointikielen toimivuudesta. Lyhyet ja ytimekkäät esimerkit tietyn päämäärän saavuttamiseksi ovat oppimisen kannalta paljon parempia kuin suuren sovelluksen anatomian tutkiminen kokemattomana.

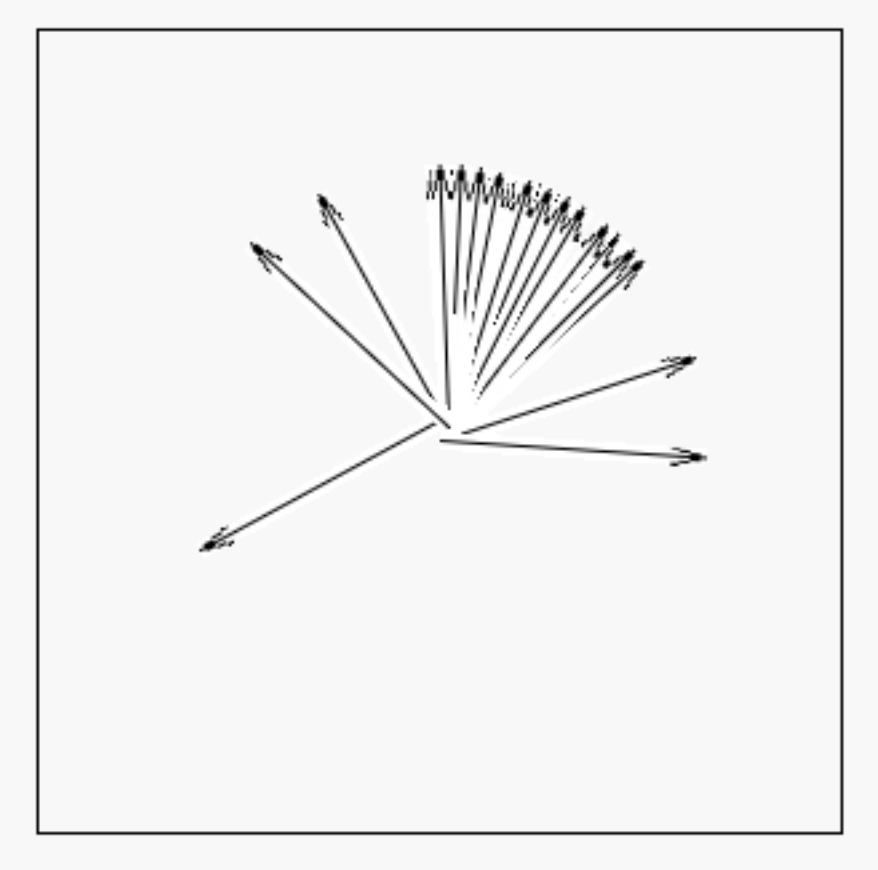
Dart SDK:n asentaminen on tehty todella helpoksi. Dartin kotisivuilla http://www.dartlang.org on suora linkki pakettiin, jossa on kaikki tarvittava. Dart ei kuitenkaan ole heti käyttövalmis, vaan vaatii taakseen JDK:n. Sen oikean version saa ladattua Oraclen omilta sivuilta [35]. Tarkkaa suositusta ladattavasta versiosta ei ole annettu, mutta Dartin kotisivun erään esimerkin mukaan suositeltavin versio olisi 1.7.0 [36] ja luonnollisesti kaikki sitä uudemmat versiot. Kun JDK on asennettu onnistuneesti, DartEditorin pitäisi käynnistyä ilman ongelmia. Mikäli ongelmia kuitenkin ilmenee, kannattaa ensimmäisenä katsoa apua Troubleshooting Dart Editor -sivulta [26]. Mikäli ongelma on jokin muu kuin sivulla manittu, suosittelen Dartin virallisia yhteisöjä, joiden luettelo löytyy osoitteesta http://www.dartlang.org/community/.

Dartin versionhallinta on osoitteessa http://pub.dartlang.org/. Se kattaa tällä hetkellä vain koodikirjastoja, joista suurin osa on ja tulee olemaan ulkopuolisten tahojen tekemiä. Se sisältää jo kattavan otannan erilaisia kirjastoja aina pienistä paketeista huomattavasti suurempiinkin. Jokaisella paketilla on oma versionhallintansa, ja näitä vanhempia versioita on mahdollista käyttää. Sivusto ei kuitenkaan sisällä esimerkiksi Dart SDK:ta, vaan aina uuden version ilmestyttyä vanhempi versio ilmoittaa, että sillä ei voi enää suorittaa sovelluksia, ellei viimeisintä versiota asenneta. Se pitää käydä manuaa-

lisesti hakemassa osoitteesta http://www.dartlang.org/downloads.html. Muutoksista virallisiin kirjastoihin voi tällä hetkellä lukea osoitteesta http://news.dartlang.org/.

## 4.2 Canvas - Tauluelementti

Ensimmäinen koodiesimerkki käsittelee canvas-elementtiä. Canvas tuli markkinoille vuonna 2004 Applen julkaisemana [37]. Se toimii yksinkertaisesti <canvas></canvas>-tageilla ja toimii kuin pala paperia: sille voidaan piirtää ja kirjoittaa mitä tahansa. Moni HTML5:sta hyödyntävistä peleistä on alkanut siirtyä canvaksen käyttöön, sillä se toimii taulumaisuutensa takia myös pelin ikkunana. Esimerkkikoodien 4 ja 5 pohjalta on kuva 4, jossa on nuoli. Nuoli on keskellä canvasta, ja se osoittaa hiiren suuntaan. Kun hiirellä painetaan canvasta, nuoli jättää siihen jälkensä. Se on mahdollista luomalla canvakseen kerrokset, ja siihen on monta tapaa. Kokeilin ensimmäiseksi tehdä useita canvas-elementtejä ja asettaa ne kaikki päällekkäin. Se toimi, sillä jokaisesta canvaksesta näkyi se, minkä halusinkin nähdä, mutta toiminnallisuuksien käyttö oli lähes mahdotonta. Esimerkin tapauksessa nuoli ei pyörinyt hiiren suuntaan vaan pysyi paikallaan. Päädyin kuitenkin tekemään yhden canvas-elementin, jonka sisälle tein kaksi kerrosta. Nämä kerrokset ovat koodissa ”etuCanvas” ja ”takaCanvas”. Esimerkkikoodin 2 tapauksessa ”etuCanvas” voisi ennemminkin olla ”mainCavas”, johon kaikki muut kerrokset voidaan liittää halutussa järjestyksessä. Tämä tapahtuu piirtämällä ”etuCanvas”in sisälle muut halutut kerrokset kuvina, eli tässä tapauksessa vain ”takaCanvas”. Luonnollisesti se, mikä ensin kirjoitetaan näkyville, näkyy kaikista taaimmaisimpana ja viimeisenä kirjoitettu näkyy ensimmäisenä. Kuva 4 esittää sovellusta toiminnassa.



Kuva 4. Canvas-esimerkki, jossa nuoli seuraa hiirtä ja jättää jäljen painettaessa.

Esimerkkikoodi 4 esittelee sovelluksessa tarvittavia arvoja. Riviltä 5 riville 17 luodaan kaikki tarvittava, jotta sovellus pystyy toimimaan. Esimerkkikoodissa 4 määritellään canvaksen koko, kuvan kääntökulma, hiiren X- ja Y-koordinaatit sekä kuvan paikaksi canvaksen keskusta. Rivellä 13 ja 14 luodaan canvas elementtinä ja sen kerrokset. Useita vaihtoehtoja tutkittuani päädyin lopulta siihen, että helpoin tapa tuoda kuva sovellukseen on käyttää koodin rivillä 17 esitettyä tapaa. Riviltä 21 riville 24 sovelluksen käynnistyessä määritellään, mihin canvakseen vaikutetaan. Tämän jälkeen canvaksen kokoa ilmaisevaa arvoa käyttämällä määritellään jokaisen canvaksen osan koko samaksi. Rivillä 33 käynnistetään animaatio, josta tarkemmin esimerkkikoodissa 5.

```
1   ***Canvas.dart (1/2)
2   import 'dart:html';
3   import 'dart:math';
4
5   const CANVAS_SIZE = 300; //Canvaksen koko
6   double rot=0.0;
7   num mouseX = 128;
8   num mouseY = 128;
9   num picX = CANVAS_SIZE/2;
10  num picY = CANVAS_SIZE/2;
11
12  //Luodaan Canvas-arvot
13  CanvasElement etuCanvas, takaCanvas;
14  CanvasRenderingContext2D etu, taka;
15
16  //Ladataan kuva. Yksi monesta mahdollisuudesta ladata kuva.
17  ImageElement img = new Element.html('<img src="images/pic1.jpg">');
18
19  void main() {
20  //Luodaan canvas ja sille eri kaksi eri kerrosta
21  etuCanvas = query("#canvas");
22  etu = etuCanvas.context2d;
23  takaCanvas = new Element.tag("canvas");
24  taka = takaCanvas.context2d;
25
26  //Määritellään Canvaksen ja sen eri kerrosten koko
27  etuCanvas..height = CANVAS_SIZE
28            ..width = CANVAS_SIZE;
29  takaCanvas..height = CANVAS_SIZE
30            ..width = CANVAS_SIZE;
31
32  //Pyöritetään animaatiota eli viisaria
33  window.requestAnimationFrame(animate);
```

Esimerkkikoodi 4. Canvas-elementti, jossa kuva pyörii hiiren mukaan. Sovelluksen arvot määriteltyinä (1/2).

Esimerkkikoodissa 5 on puolestaan toiminnallinen osa, kun esimerkkikoodi 4 oli vasta sovelluksen arvojen määrittelyä ja rakentamista. Riviltä 36 alkava koodi tarkkailee, liikkuuko hiiri canvaksen päällä, ja jos liikkuu, se päivittää X- ja Y-koordinaatit oikeiksi. Rivillä 42 alkava koodi tarkkailee, onko hiiren vasen painike painettu alas canvaksen päällä. Mikäli on, aktivoidaan komento ”drawImages();”, joka alkaa riviltä 68. Tämä koodi jättää canvakseen (takaCanvas) identtisen jäljen, joka ei lähde pois. Riviltä 48 alkaen käynnistyy animaatio, ja se sisältää kolme komentoa: laske kulma, piirrä nuoli ja aloita animaatiolooppi alusta. Rivillä 77 alkava komento laskee uuden kulman radiaaneissa käyttäen hyväksi kuvan X- ja Y-koordinaatteja verrattuna hiiren X- ja Y-koordinaatteihin. Tämän jälkeen suoritetaan riviltä 55 alkava koodi, joka rivin 68 tapaan poikkeaa siten, että se piirtää itsensä ”etuCanvakselle” eikä ”takaCanvakselle”. Lisäksi alussa oleva ”clearRect”-komento pyyhkii koko edellisen kuvan pois ennen uuden piirtämistä. Lopuksi palataan taas animointikehyksen alkuun.

```
***Canvas.dart (2/2)
34  //Kun hiiri liikkuu, sijaintitiedot päivittyvät ja näin kuva osaa suunnata hiirtä
35   kohti
36    query("#canvas").onMouseMove.listen((MouseEvent e){
37      mouseX = e.layerX;
38      mouseY = e.layerY;
39    });
40
41    //Klikatessa viisari tulostaa kuvan tismalleen oikeassa asennossa itsensä alle
42    query("#canvas").onMouseDown.listen((MouseEvent e){
43      drawImages();
44    });
45   }
46
47   //Pyöritetään animaatiota
48   void animate(num time){
49     calculateAngle();
50     images();
51     window.requestAnimationFrame(animate);
52   }
53
54   //Piirretään "heiluri" keskelle ruutua.
55   void images(){
56      etu..clearRect(0, 0, CANVAS_SIZE, CANVAS_SIZE)
57     ..drawImage(takaCanvas, 0, 0) //Liitetään Canvaksen muut kerrokset lataamalla
58    ne kuvina. Riippuen sijoituskohdasta, tämä voisi vaikka olla päälimmäisenä
59    ..save()
60    ..translate(picX, picY)
61    ..rotate(rot)
62    ..drawImage(img, 0, 0)
63    ..restore();
64   }
65
66   //Jätetään jälki canvakselle kun painetaan hiirtä
67   //Koodi on muuten sama, mutta nyt aluetta ei pyyhitä tyhjäksi joka välissä
68   void drawImages(){
69    taka..save()
70    ..translate(picX, picY)
71    ..rotate(rot)
72    ..drawImage(img, 0, 0)
73    ..restore();
74   }
75
76   //Lasketaan hiiren sijainti ja käännetään kuva osoittamaan hiirtä
77   void calculateAngle(){
78     var deltaY = mouseY-picY;
79     var deltaX = mouseX-picX;
80     rot = atan2(deltaY, deltaX); //Antaa suoraan radiaaneina
81   }
```

Esimerkkikoodi 5. Canvas-elementti. Sovelluksen toiminnallinen osa (2/2).

Esimerkeissä 4 ja 5 on paljon informaatiota pienessä tilassa. Ne eivät vain havainnollista, kuinka nuoli seuraa tarkasti hiiren liikkeitä tai kuinka canvakselle voi piirtää haluamansa kuvan, vaan ne näyttävät myös kaikki ne keinot, joilla on mahdollista suorittaa nämä toimenpiteet. Canvas on itsessään piirtoalusta, ja kaikki sille piirtyvät merkit ja kuvat jäävät siihen. Ellei Canvasta tyhjennetä jokaisella askelmalla, alkaa kuvio muistuttaa jumittunutta Windowsin virheilmoitusta, jota on raahattu pitkin näyttöä. Toinen tärkeä huomioitava on window.requestAnimationFrame-(animate). Tämä komento käynnistää animaation eli mahdollistaa kuvan jatkuvan liikkumisen hiiren mukaan ilman, että pitäisi jatkuvasti tehdä jokin tietty komento. Otetaan esimerkiksi peli, jossa

pelaaja haluaa liikkua ylös ja painaa siihen tarkoitukseen olevaa näppäintä. Ilman AnimationFramea joutuisi pelaaja painamaan joka kerta haluttua näppäintä, jotta saisi pelihahmonsa liikkumaan haluttuun suuntaan. Peliympäristö ei olisi eloisa, vaan heräisi eloon vain silloin, kun pelaaja painaisi näppäintä. Kaikki komennot AnimationFramen sisällä toistuvat peräjälkeen yhä uudestaan ilman, että käyttäjän erikseen täytyy asiaan puuttua.

### 4.3 Class - Luokka

Kun yhdistetään kahta Dart-tiedostoa, joista toinen on kutsuvana osapuolena, Dartilla on oma tapansa hoitaa tämä. Kutsuva tiedosto käyttää tuontiin komentoa ”part” ja kutsuttavan tiedoston nimeä. Erikoisuutena kutsuttavassa tiedostossa tulee olla viittaus ”part of” ja sen tiedoston nimi, joka kutsuu tätä tiedostoa. Esimerkin tapauksessa olen tehnyt hyvin pienen ja yksinkertaisen luokan, joka vaatii kaksi arvoa: ”name” ja ”title”. Nämä arvot lähetetään eteenpäin toiseen tiedostoon, joka liittää ne valmiin tekstin sekaan ja lähettää viestin takaisin. Erilaiset lisätiedostot ovat hyödyllisiä muun muassa funktioiden tallentamiseen ja kutsumiseen. Pääohjelmassa raskaatkin funktiot voidaan kutsua kätevästi yhdellä pienellä koodirivillä, ja näin pidetään koodi mahdollisimman selkeänä ja luettavana. Kaikkea ei ole pakko kirjoittaa pääohjelmaan. Pienten ohjelmien parissa saattaa tuntua turhalta luoda erillinen tiedosto yhdelle luokalle ja kutsua sitä sitten pääohjelmassa. Pienissä sovelluksissa se saattaa ollakin turhaa, mutta kun tehdään tuhansia rivejä koodia, optimointi ja selkeys ovat avainasemassa koodin kehittämisen kannalta. Kuva 5 esittää lopputulosta, joka kirjoitetaan käyttäjän näkyville.


Kuva 5. Luokka-esimerkin tulostus.

Esimerkkikoodissa 6 ensimmäisenä on pääohjelma, joka kutsuu toista tiedostoa. Toisena kutsuttava tiedosto, jota pääohjelma käyttää hyväkseen, avustaa viestin muodostamisessa.

```
***Class.dart
import 'dart:html';

part "draw_class.dart";

void main() {
  query("#text").text =
draw.message("Ville",
"opiskelija");
}
```

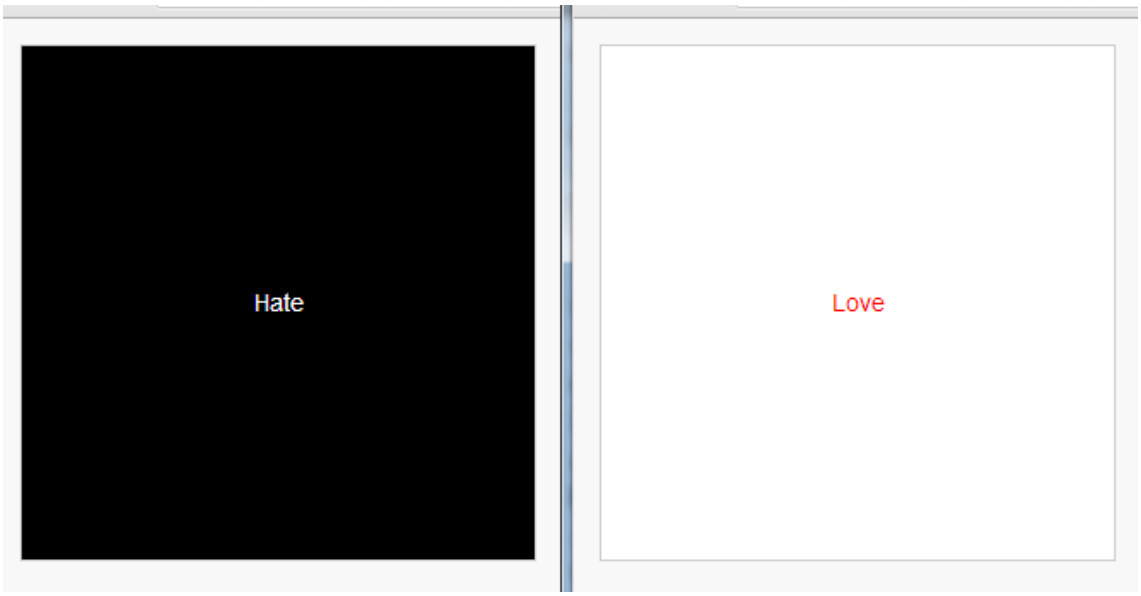
```
***draw_class.dart
part of inssi_esimit;

class draw{
  static message(String name, String title){
    String message = "Nimesi on $name ja asemasi
$title";
    return message;
  }
}
```

Esimerkkikoodi 6. Pääohjelma ja luokka.

Esimerkkikoodin 6 sovellus on hyvin yksinkertainen. Siinä valitaan muokattava DOM, tässä tapauksessa ”#text”, joka on tarkemmin määritetty HTML- ja CSS-tiedostossa. Elementille muutetaan tai annetaan teksti, jonka se tuo käyttäjän näkyville. Dartissa ei ole komentoa ”function”, joten sen sijaan käytetään komentoa ”class”. Tämä on yksinkertainen toiminto, joka tarvitsee kaksi syötettyä sanaa, muodostaa niistä lauseen ja lähettää sen takaisin. Esimerkin tapauksessa lähetän kaksi Stringiä, ”Ville” ja ”opiskelija”, draw-luokkaan. Draw-luokan sisällä on ”static message”, joka liittää saadut arvot oikeille paikoilleen. Kuvassa 5 käyttäjä ei osaa ollenkaan sanoa, kuinka monen tiedoston kautta teksti on kulkenut ennen esille tulostamista. Näin pienessä sovelluksessa staattisen arvon hyötyä ei välttämättä huomaa, mutta suuremmissa sovelluksissa se voi helpottaa työtä suuresti. Esimerkkinä voisi olla sovellus, jossa käyttäjä voi piirtää palloja, neliöitä ja kolmioita. Ohjelmassa on myös tusina muita toimintoja ja tarvitaan jokin helppo tapa saada luotua selkeä yhteys haluttuun luokkaan. Tässä tapauksessa voisi olla luokka ”draw”, joka kertoo ohjelmoijalle, että luokka sisältää piirtämiseen liittyviä kaavoja. Samaa draw-luokkaa hyödyntäen voidaan pisteen erotuksella syöttää halutut tiedot kaikille kolmelle palikalle. Se voisi olla draw.ball, draw.rectangle ja draw.triangle. Tästä käy hyvin esille, että ohjelmoija haluaa piirtää kuviot eikä halua tehdä mitään muuta sen lisäksi. Pelkät ball, rectangle ja triangle eivät kertoisi paljoa. Koodi ja luokat pysyvät hyvässä järjestyksessä tämän toiminnon avulla.

## 4.4 DOM – Yksittäinen elementti

DOM on alusta- ja kieliriippumaton järjestelmä, joka mahdollistaa sisällön, rakenteen ja tyylin dynaamisen muokkaamisen skriptien tai määriteltyjen ohjelmien avulla [38]. DOM-elementit ja niiden muokkaus ovat olleet pitkään JavaScriptin ehkä yleisin ja helpoin tapa. Esimerkki tällaisesta voi olla -elementti, jolle voidaan suorittaa animointi tai tehdä muutoksia ilman, että sivua tarvitsee ladata uudelleen. Sen sisältö voidaan haluttaessa kokonaan vaihtaa, taustaväriä muuttaa ja kokoa venyttää. Elementtejä voi myös luoda niin sanotusti ”tyhjästä” eli käyttämällä vaikkapa Dartia. DOM:sta olen tehnyt pari esimerkkiä tarkoituksenani näyttää, mihin kaikkeen voidaan vaikuttaa dynaamisella tasolla. Tarkastelussa ovat pelkästään -elementit, sillä niissä muutoksia on helppo tehdä ja ne näkyvät kaikista selvimmin. Kuva 6 esittää tilannetta molemmissa tapauksissa.




Kuva 6. DOM-esimerkin tulostukset, kun hiiri viedään elementin päälle ja pois.

Esimerkkikoodissa 7 olen valinnut yksittäisen div-elementin, joka muuttaa tyyliään ja sisältöään, kun hiiri viedään päälle ja tuodaan pois. Vaihto tapahtuu sulavasti värien osalta.

```
***Dom_one.dart
import 'dart:html';

main(){
  query("#area").onMouseOver.listen((MouseEvent hoover){
    query("#area")..style.transition = "2s"
                  ..style.backgroundColor = "black"
                  ..style.color = "white"
                  ..text = "Hate";
  });
  query("#area").onMouseOut.listen((MouseEvent hoover){
    query("#area")..style.transition = "2s"
                  ..style.backgroundColor = "white"
                  ..style.color = "red"
                  ..text = "Love";
  });
}
```

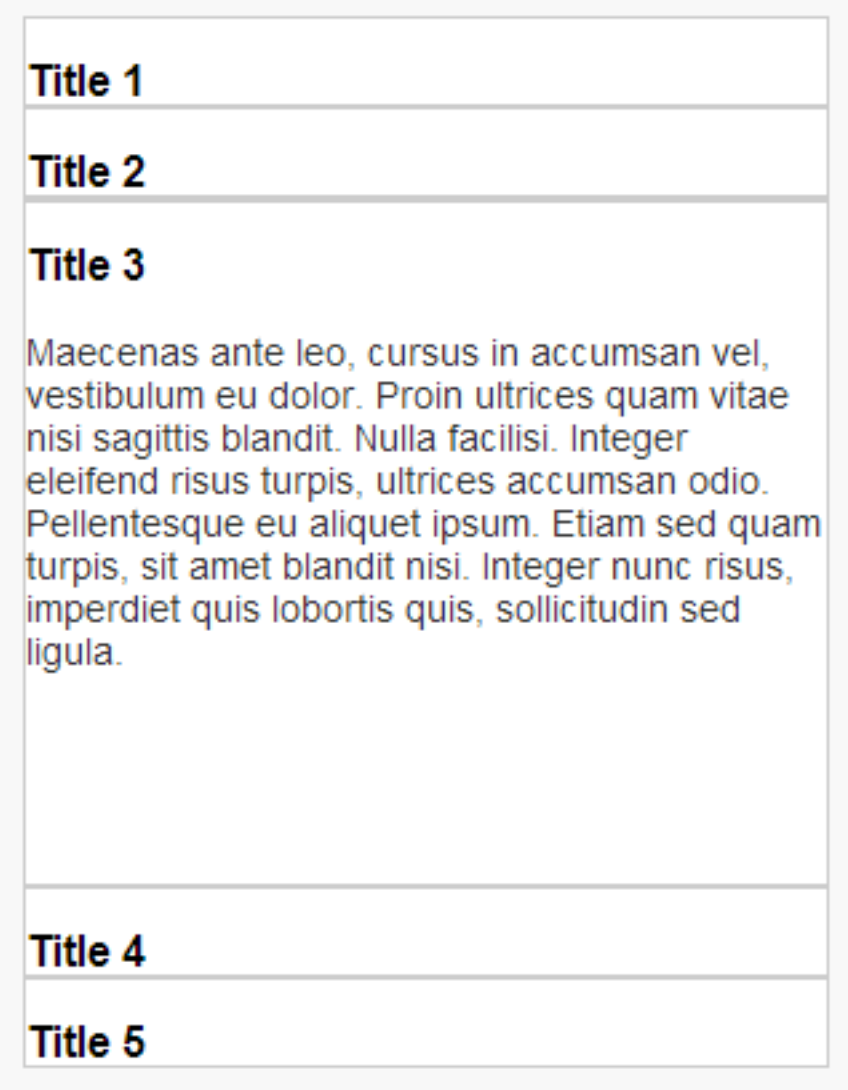
Esimerkkikoodi 7. DOM-elementin tyylin muokkaamista hiiren avulla.

Kuvassa 6 voidaan vasemmalla huomata vaikutus, kun hiiri viedään elementin päälle. Oikealla puolestaan hiiri on viety pois elementin päältä. Esimerkkikoodin 7 HTML-tiedostossa on määritelty -elementille ”ID=area”. Dartiin on kirjoitettu kaksi komentoa: ”onMouseOver” ja ”onMouseOut”. Ne tutkivat, onko hiiri päällä vai ei. Kun hiiri tuodaan päälle, muutetaan div:n tyyliä asettamalle sille ensin muuttumisaika. Tämä tarkoittaa sitä, kuinka nopeasti tyyli vaihtuu vanhasta uuteen. Esimerkin tapauksessa on valittu 2 sekuntia. Div:n taustaväri ja siinä olevan tekstin väritys vaihtuvat nyt sulavasti uuteen kahden sekunnin aikana. Myös teksti vaihtuu, mutta muutos ei tapahdu sulavasti vaan on heti voimassa, sillä koodissa käytetty ”transition” vaikuttaa vain tyylitiedostoon. Nyt kun hiiri viedään pois div:n päältä, aloitetaan sama prosessi, mutta eri väreillä ja eri tekstillä.

## 4.5 DOM – Monta elementtiä

Toisessa DOM-esimerkissä on viisi eri div-elementtiä päällekkäin. Jokaisessa niissä on näkyvillä vain otsikko, mutta kun niitä painaa, ne suurenevat alaspäin. Toiseen kertaan painettaessa ne palautuvat oletusarvoihinsa. Tämä on nykyään aika suosittu keino esittää uutisia tietyillä sivuilla. Ei tuoda kerralla kaikkea tekstiä käyttäjän näkyville, vaan käyttäjä voi omatoimisesti avata kiinnostavan uutisen. HTML-tiedostoon on luotu viisi div-elementtiä samalla ID:llä, ja niiden sisään on kirjoitettu tekstiä. Yleensä ne kaikki näkyisivät sivulla yhtenä tekstin jatkumona, mutta CSS:n avulla on määritelty kaikki div-elementit tiivistettyyn tilaan ja piilotettu ylivuotava teksti. Dartilla luotiin ohjelma, joka tutkii, onko div-elementti ”auki” vai ”kiinni,” ja näin se suorittaa halutun toiminnon hiiren

vasenta painiketta painettaessa. Kuvassa 7 näkyy yksi elementeistä tilassa ”auki”, kun muut ovat ”kiinni”.




Kuva 7. DOM-esimerkki, jossa voidaan avata ja sulkea elementtejä hiiren painalluksella.

Esimerkkikoodissa 8 näkyy, kuinka vähän koodia on loppujen lopuksi tarvittu esimerkin rakentamiseen.

```
***Dom_two.dart
import 'dart:html';
import 'dart:html';

int dummy = 32;

main(){
  var div = queryAll("#area");

  for(var i=0; i<div.length; i++)
  {
    div[i].onMouseDown.listen((Event e){
      //Animaatio - auki
      if (dummy == 32){
        dummy = 256;
        div[i]..style.transition = "0.5s"
              ..style.height = "${dummy}px";
      }
      //Animaatio - kiinni
      else if (dummy == 256){
        dummy = 32;
        div[i]..style.transition = "0.5s"
              ..style.height = "${dummy}px";
      }
    });
  }
}
```

Esimerkkikoodi 8. Usean DOM-elementin tyylin muokkaamista hiiren avulla.

Kuvan 7 tapauksessa elementtejä on viisi. Esimerkkikoodi 8 mahdollistaa kuitenkin elementtien määrän väliltä nolla ja ääretön. DOM-elementtejä varten Dartissa on kaksi komentoa. Esimerkkikoodissa 7 käytettiin komentoa ”query(”#area”)”, mutta tässä käytetään komentoa ”queryAll(”#area”)”. Erona on, että query vaikuttaa vain yhteen elementtiin, kun queryAll kerää kaikki samalla ID:llä olevat elementit. Hyvin helpolla for-silmukalla katsotaan, kuinka monta elementtiä samalla ID:llä on, ja tämän jälkeen pyöritetään silmukkaa lukumäärän verran läpi. Kun silmukka on löytänyt elementin, jota tässä tapauksessa on painettu, se tekee sille halutun toiminnon. Yksittäisten elementtien tyyliä voidaan muokata, vaikka niillä olisikin yhteinen ID. Aluksi kaikkien elementtien korkeus on säädetty 32 pikseliin, joka on määritelty tyylitiedostossa. Luodaan arvo ”dummy”, joka ilmaisee nykyisen korkeuden ja muuttuu korkeuden muutosten mukaan. Vaikka esimerkissä on vain yksi arvo, se toimii silti yksilöllisesti jokaisen elementin kohdalla. Ohjelma katsoo hiiren painalluksen jälkeen, onko dummy 32 vai 256, ja tekee muutokset tämän perusteella. Aika, joka kuluu korkeuden muuttamiseen, on 0,5 sekuntia, ja haluttu uusi korkeus on dummyn arvo. Dartissa aika-arvon voi ilmoittaa joko millisekunteina tai sekunteina, ja esimerkin tapauksessa ”0.5s” voitaisiin ilmoittaa ”500ms”. Koodi on niin tehokas, että sillä saisi luotua äärettömän moneen elementtiin toiminnallisuuden, kunhan niiden ID vain on sama.

# 5 Dart-sovelluksia

## 5.1 List, Map ja JSON – Datan varastointi

JSON on formaatti, joka on luotu vaihtamaan datan rakennetta ja kuljettamaan tietoa helposti. Se on ihmiselle helppolukuinen ja tietokoneelle sekä helppo purkaa että luoda. [39.] JSONia käytetään yleisesti List- ja Map-luokkien kanssa. List on Arrayn kaltainen luokka, johon voidaan tallentaa tietoa eri soluihin. Esimerkiksi vaikkapa numerojono yhdestä viiteen näyttäisi tältä: List numerot = [1,2,3,4,5]. Haluttu solu saadaan esille käyttämällä numerotunnistetta. Esimerkiksi: List numerot[3] on yhtä kuin 4. Map koostuu käytännössä useasta List-luokasta. Kun Mapiin kirjoitetaan arvoja, arvoille määritellään avainarvo. Esimerkiksi tallennettaessa henkilön rahamäärää Mapiin, sana ”raha” voi toimia avainarvona, jolla sovellus löytää rahamäärän Mapista: ”Map henkilo = {raha : 1337}” => ”var tulos = henkilo[’raha’] = 1337”.

JSONin käyttö Dartissa on helppoa. Siinä on kaksi funktiota, parse() ja stringify(), joista parse() hoitaa JSONin purkamisen ja stringify() puolestaan kääntää Mapin tai Listin Stringiksi. Käyttötarkoituksia voisi olla monia. Koska se mahdollistaa suuren listan pakkaamisen yhteen tekstitiedostoon, voidaan esimerkiksi tietokantaan tallentaa paljon tietoa halutun taulun yhteen soluun. Kaikissa tapauksissa tämä ei ole järkevää, ja onkin tärkeää pitää erillään tiettyjä elementtejä, joilla tietoa voidaan taas hakea. Seuraavissa esimerkissä esitellään List:n, Map:n ja JSONin käyttöä Dartissa ja sitä, mitä niillä voi tehdä yhdessä. Jokaisen esimerkkikoodin ”print” kohdan jälkeen on kommentteina esitettynä konsolitulostus. Esimerkkikoodissa 9 havainnollistetaan Listin käyttöä Dartin sisällä.

```
***List.dart (1/3)
import 'dart:json';

main() {
  //List
  List listData = new List(); //Luodaan uusi list
  listData = ['Hello','world'];
  listData.add('Jeps');
  print('${listData[0]}, ${listData[1]}, ${listData[2]}'); //Hello, world, Jeps
}
```

Esimerkkikoodi 9. Listin toimintaperiaate.

Ensin luodaan uusi, tyhjä List nimeltä ”listData”. listDataan lisätään halutut tiedot, mutta myöhemmässä vaiheessa tulee käyttää ”.add”-komentoa tai muuten jo olemassa olevat tiedot pyyhkiytyvät yli. Listin tietoja voi myös muuttaa käyttämällä esimerkkikoodin 9 tapauksessa listDatan solua numero 2. Kun Listiin lisätään informaatiota, se kasvaa automaattisesti. Ennen kuin on luotu solu, jonka löytää numerolla 2, ei voida hakea numeroa 2 ilman, että Dart ilmoittaa virheestä. Kuitenkin, jos on olemassa solu 2, siinä olevaa tietoa voidaan muuttaa käyttämällä komentoa ”listData[2] = ’Rabbit’;”. Esimerkkikoodissa 10 on esiteltynä Mapin toimintaa Dartin sisällä.

```
***List.dart (2/3)
  //Map
  Map mapData = new Map(); //Luodaan uusi
map
  mapData["animal"] = new List();
  mapData["animal"].add("Apina");
  mapData["animal"].add("Norsu");
  mapData["food"] = new List();
  mapData["food"].add("Pizza");
  mapData["food"].add("Kaali");
  print(mapData["animal"][1]); //Norsu
```

Esimerkkikoodi 10. Mapin toimintaperiaate.

Map on monen Listin kokoelma. Esimerkkikoodi 10 näyttää, kuinka ennen uuden tiedon lisäämistä tulee luoda Map ja sen sisälle List. Mapiin voi lisätä helposti useita Listejä, kuten esimerkistä käy ilmi. Tiedon lukeminen Mapista on askeleen monimutkaisempaa kuin Lististä. Siinä missä List-komennosta saa tietoa pelkän numeron avulla, Mapissa tarvitaan kaksi avainarvoa halutun arvon paikantamiseen. Esimerkiksi esimerkkikoodin 10 mukaan haluttaessa hakea arvo ”Pizza” tulee ensin tietää, että Pizza on ruokaa eli kuuluu avainarvon ”food” alle. Tämän jälkeen tulee tietää pizzan sijainti Listissä. Se on esimerkin mukaan 0, joten koodi on seuraava: ”Pizza = mapData[’food’][0]”. Mapin käytössä tulee olla tarkkana, sillä uuden listan luominen saattaa unohtua, mutta ohjelma toimii kaikesta huolimatta normaalisti. Esimerkiksi oletetaan, että on luotu List ”games” vaikka näin ei olekaan tehty. Kirjoitetaan seuraava pätkä koodia: ”mapData[”games”] = ’hei’;”. Dart tallentaa arvon ”hei” kohtaan ”games”. Tästä ei tule virheilmoitusta, mutta mikäli yrittää käyttää komentoa ”.add”, Dart ilmoittaa virheestä. Esimerkkikoodi 11 esittelee keinoja muuntaa Mapin ja Listin JSON-koodatuiksi sekä purkaa JSON takaisin Mapiksi ja Listiksi.

```
***List.dart (3/3)
//List -> String -> List
  String listToJson = stringify(listData); //Tehdään list:stä string
  print(listToJson); //["Hello","world","Jeps"]
  List jsonToList = parse(listToJson); //Muunnetaan strin list:ksi
  print(jsonToList); //[Hello, world, Jeps]
  print(jsonToList[1]); //world

  //Annetaan list suoraan Stringinä
  String list2Json = '["Mono","Poly"]'; // Syötetään jotain arvoja
  List parseList = parse(list2Json); //Tehdään stringistä list
  print(parseList[0]); //Mono


  //Map -> String -> Map
  String mapToJson = stringify(mapData); //Muutetaan Map Stringiksi
  print(mapToJson); //{"food":["Pizza","Kaali"],"animal":["Apina","Norsu"]}
  Map jsonToMap = parse(mapToJson);
  print(jsonToMap); //{food: [Pizza, Kaali], animal: [Apina, Norsu]}
  print(jsonToMap["food"][1]); //Kaali

  //Annetaan Map Stringinä
  String map2Json = '{"color":["Punainen","Keltainen"],"country":["Suomi","Norja"]}';
//Syötetään joitain arvoja
  Map parseMap = parse(map2Json);
  print(parseMap["country"][1]); // Norja

  //BONUS
  print(jsonToList[1][0]); //Ottaa "world" ensimmäisen kirjaimen, eli tulostaa "w"!
  print(mapData["animal"][1][2]); //Ottaa "Norsu" keskimmäisen kirjaimen, eli tulos-
taa "r"!
```

Esimerkkikoodi 11. JSONin toimintaperiaate.

Sen sijaan, että purettaisiin List tai Map useaan eri arvoon varastointia ja liikuttamista varten, JSON pystyy muuntamaan minkä tahansa List- tai Map-arvon yhteen Stringiin ja näin helpottamaan urakkaa. Esimerkkikoodissa 11 on listattu tapoja muuttaa List ja Map JSONiksi ja purkaa JSON takaisin Listiksi ja Mapiksi. Toimenpide on yksinkertainen muunnettaessa Listiä tai Mapia JSON Stringiksi, kuten esimerkistä näkyy. Takaisin muuntaminen om aivan yhtä helppoa, mutta tulee olla tarkkana, että kirjoittaa oikean tyypin, List tai Map. Esimerkissä havainnollistetaan myös, että JSON Stringin voi antaa suoraan ilman List- tai Map-tyyppejä kirjoittamalla sen esimerkiksi käsin. Koodi on silmin luettavissa ja muokattavissa ja JSON-parseri pystyy tulkitsemaan tekstin aivan yhtä hyvin. Esimerkkikoodin 11 neljä viimeistä riviä on omistettu ominaisuudelle, jolla halutusta sanasta voidaan vielä saada esille haluttu tai halutut kirjaimet. GML:ää hyödyntämällä tein testin, jossa pelaaja pystyy sijoittelemaan palikoita kentälle ja lataamaan ja tallentamaan tämän alueen. JSON osoittautui toimivaksi tavaksi tallentaa tietoa, ja tallennetun tiedon pystyisi nyt lähettämään palvelimelle muiden käyttäjien ladattavaksi.

## 5.2 IO – Palvelin ja asiakas

Palvelimella voidaan tarkoittaa tietokoneessa suoritettavaa palvelinohjelmistoa ja itse ohjelmistoa suorittavaa tietokonetta. Kaikkia tietokoneita tai sovelluksia, jotka käyttävät palvelinta, kutsutaan asiakkaiksi. Esimerkiksi julkisille verkkopalvelimille ei tarvita käyttäjien käyttäjätunnuksia tai salasanoja, vaan käyttäjän ottaessa yhteyttä palvelimeen palvelin vastaa pyyntöön kertaluonteisesti. Tämän jälkeen palvelin ei välttämättä tiedosta käyttäjän olemassaoloa. [40.] Alkuvuonna 2013 Dartin IO-kirjasto kirjoitettiin kokonaan uusiksi. Asyncin sijaan Dart käyttää nykyisin streamausteknologiaa. [41.] Esimerkkikoodit 12 ja 13 on otettu Dartin kotisivuilta, mutta niihin on tehty pieniä muutoksia ja lisätty kommentteja tuomaan selkeyttä [42]. Esimerkkikoodi 12 esittelee sovelluspalvelimen toimintaa.

```
***http://www.dartlang.org/docs/dart-up-and-running/contents/ch03.html#ch03-io-http
***IO_server.dart
import 'dart:io';

main() {
  //Asiakas esittää pyynnön palvelimelle. Tämä on paluuviesti.
  dartHandler(HttpRequest request) {
    print('Uusi pyyntö');
    request.response.addString('Täällä serveri. Kuuleeko client?');
    request.response.close();
  };

  //Liitetään palvelin tiettyyn osoitteeseen ja porttiin ja laitetaan
  //tietty polku minne se johtaa.
  HttpServer.bind('127.0.0.1', 8888).then((HttpServer server) {
    server.listen((request) {
      print("Saatiin pyyntö");
      //Mikäli asiakas yrittää saada tietoa seuraavasta polusta,
      //lähetetään hänelle viesti normaalisti.
      if (request.uri.path == '/languages/dart') {
        dartHandler(request);
      } else {
        //Mikäli asiakas yritti jotain muuta, heitetään virheilmoitusta.
        request.response.addString('Ei löytynyt');
        request.response.statusCode = HttpStatus.NOT_FOUND;
        request.response.close();
      }
    });
  });
}
```

Esimerkkikoodi 12. Yksinkertaisen palvelinpuolen koodi [42].

Esimerkkikoodissa 12 nähdään, että jos asiakkaalta tulee pyyntö oikeaan URL-osoitteeseen, tässä tapauksessa paikalliselle palvelimelle, lähetetään vastaus takaisin. Polku on määritelty ’/languages/dart’. Mikäli pyyntö osuu johonkin toiseen osoitteeseen, lähetetään viesti ”Ei löytynyt” sekä virheilmoitus ”404”.

Asiakkaan puolelta määritellään osoite, josta tietoa pyydetään. Osoitteena toimii tässäkin tapauksessa paikallinen palvelin. Tämän jälkeen lähetetään pyyntö eteenpäin. Riippumatta saadusta vastauksesta vastaus kirjoitetaan näkyviin ja suljetaan yhteys. Asiakaspuolen esimerkkikoodissa 13 asiakas lähettää palvelimelle pyynnön.

```
***http://www.dartlang.org/docs/dart-up-and-running/contents/ch03.html#ch03-io-http
***IO_client.dart
import 'dart:io';
import 'dart:uri';

main() {
  //Mistä osoitteesta haetaan tietoa
  var url = new Uri('http://127.0.0.1:8888/languages/dart');
  var httpClient = new HttpClient();
  //Lähetetään pyyntö haluttuun osoitteeseen
  httpClient.getUrl(url)
    .then((HttpClientRequest request) {
      print('Pyydetään');
      return request.close();
    })
    //Jos ja kun tulee vastaus
    .then((HttpClientResponse response) {
      print('Vastataan');
      response.transform(new StringDecoder()).toList().then((data) {
        //Kirjoitetaan viesti näkyviin
        var body = data.join('');
        print(body);
        httpClient.close();
      });
    });
}
```

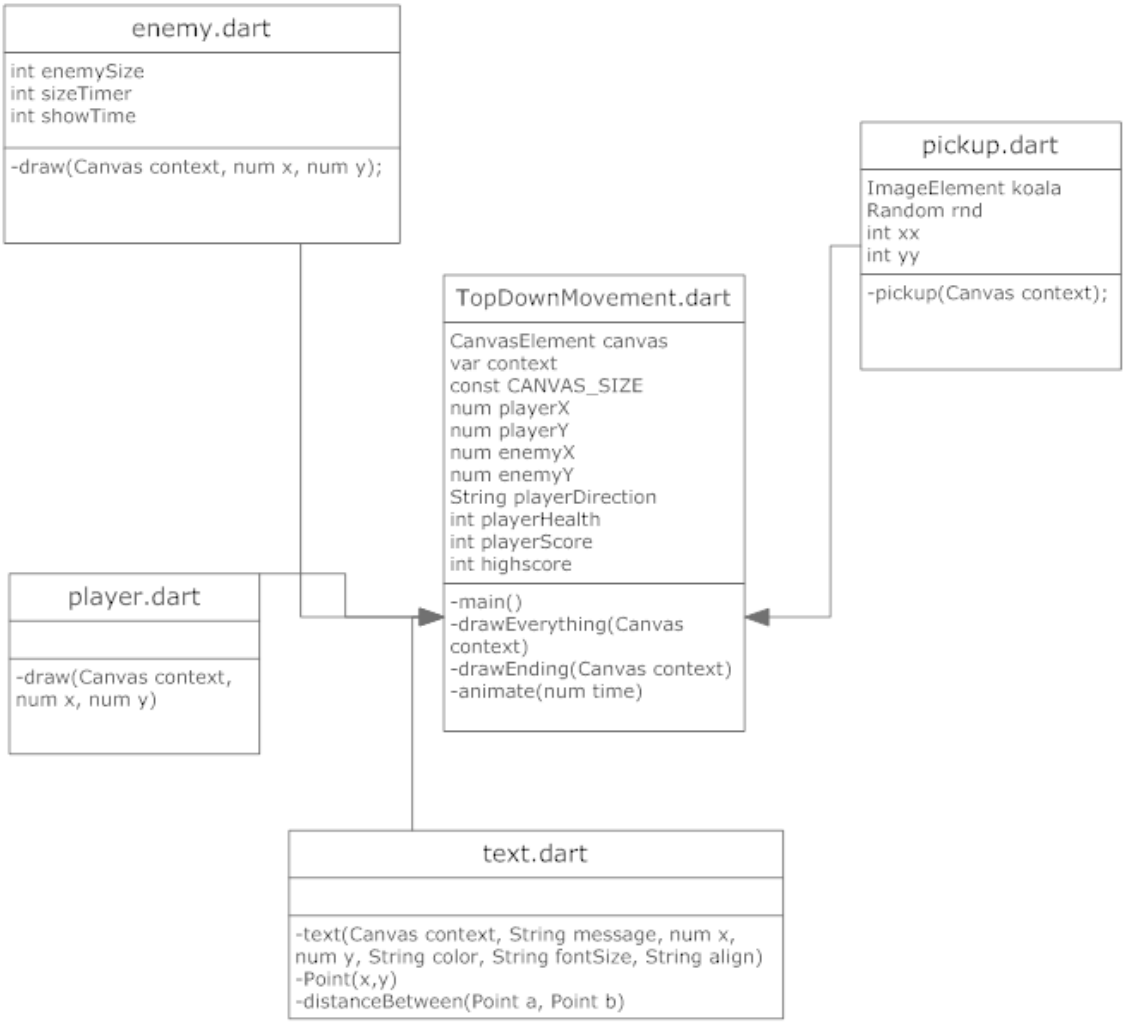
Esimerkkikoodi 13. Yksinkertaisen asiakaspuolen koodi [42].

Kuten esimerkkikoodista 13 huomataan, asiakas pyytää tietoja, jotka kohdistuvat tiettyyn osoitteeseen, ja vastauksesta riippumatta se tulostetaan käyttäjän näkyville. Asiakas- ja palvelinpuoli vaativat tavallista enemmän opettelua, jotta koodin kirjoittamisesta tulee sujuvaa. IO oli yksi ensimmäisistä Dartin kirjastoista dart:coren ja dart:html:n kanssa [5, s. 10–11]. Vielä toistaiseksi en odota näkeväni sitä PHP:n korvaajana palvelinpuolentekniikassa, mutta Dartin nopea kehittyminen saattaa mahdollistaa tämän jonakin päivänä.

## 5.3 Peli – Canvas, Class ja DOM

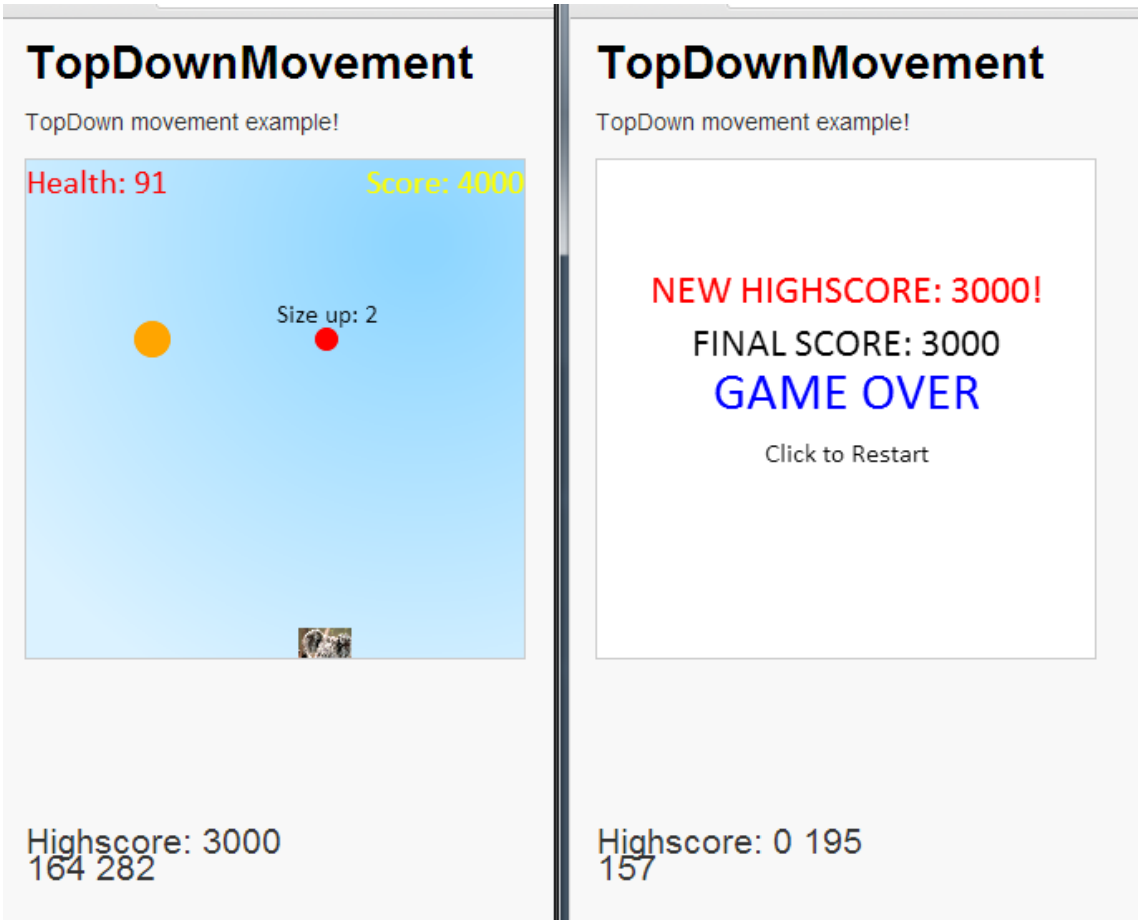
Yhdistelemällä esimerkkien 4, 5, 6 ja 7 aiheita loin pelin havainnollistaakseni monimutkaisen sovelluksen toimintaa. Pelissä on pelaajapallo ja vihollispallo. Pelaajan tehtävä

on kerätä neliöitä pitkin kenttää ja näin kasvattaa pisteitään. Jotta peli ei olisi liian helppo, vihollisen pallo kasvaa kolmen sekunnin välein isommaksi. Osuessaan viholliseen pelaaja ottaa vahinkoa -1 niin kauan, kuin on kosketuksissa vihollispallon kanssa. Kun vahinkopisteet ovat laskeneet nollaan, peli päättyy ja ruudulle ilmestyy lopputulos ja mahdollinen ilmoitus uudesta kärkituloksesta. Kärkitulos päivittyy pelin alla olevaan elementtiin. Pelin lähdekoodi on suuri, ja se on tämän raportin liitteenä 1. Kuvassa 8 on esitelty pelin toimintaa UML-kaaviokuvalla. Se esittää pääohjelmaa TopDownMovement.dartia, johon on liitettynä kaikki luokat. Luokat vaikuttavat pääohjelman komennossa ”drawEverything”, jossa piirtäminen ja laskeminen tehdään.




Kuva 8. UML-kaavio pelin toiminnasta.

Kuvassa 9 näkyy kaksi kuvankaappausta pelistä. Vasemmanpuoleinen esittää pelitilaa ja oikeanpuoleinen loppuruutua.




Kuva 9. Pelitilanne ja loppuruutu.

Kuvassa 9 näkyvä peli toimii AnimationFramen varassa. Pelaajan ei tarvitse erikseen antaa komentoa pitää palloaan liikkeessä, jotta vihollispallo saisi samalla käskyn liikkua. Kuvassa esiintyvät luvut peli-ikkunan alla, suoraan ”Highscore”-tekstin alapuolella, esittävät pistelaatikon X- ja Y-koordinaatteja canvaksella. Nämä luvut päivittyvät automaattisesti joka kerta, kun pelaajan hahmo saa koalan kiinni. Kuvan 8 UML-kaaviosta selviää, että kaikki tiedostot vaikuttavat suoraan pääohjelmaan.

## 6 Yhteenveto

Insinöörityön tarkoituksena oli tutkia uuden sukupolven verkko-ohjelmointikielten mahdollisuuksia nykyaikaisilla markkinoilla. Tarkkailussa oli viimeisin julkaistu verkko-ohjelmointikieli, Googlen kehittämä ja julkaisema Dart.

JavaScript on hallinnut markkinoita pitkään, ja se on ollut käytetyin ohjelmointikieli tuomaan dynamiikkaa verkkosivuille. Dartin tulo markkinoille tuo mukavaa vaihtelua, sillä koodaamisen aloittaminen on tehty mahdollisimman helpoksi ja yksinkertaiseksi. Se on tehty selkeäksi lukea, ja samalla sitä on todella helppo kirjoittaa. JavaScript tuntui Dartin jälkeen vanhalta ja turhan monimutkaiselta. JavaScriptin mielenkiintoiset tavat tulkita koodia verrattuna Dartin selkeään tulkintatapaan saivat Dartin tuntumaan enemmän omalta, kun ei tarvinnut ajatella liikaa, kuinka koodinsa muodostaa. Ainut puolustus JavaScriptille on sen levinneisyys. Dart on uusi ja koko ajan kehittyvä ohjelmointikieli, eikä se ole saanut vielä paljoa jalansijaa markkinoilta, mikä voi koitua sen kohtaloksi. Tällä hetkellä Dartin menestymisen suurin este ovat kaikki selain- ja laitevalmistajat paitsi Google, jonka Chrome-selaimessa Dart toimii. Jos yhteistyötä ei muiden kanssa saada muodostettua, Dartilla tulee olemaan lyhyt elämänkaari. Toisaalta, Googlella on rahaa ja se pystynee neuvottelemaan itselleen sopimuksia selain- ja laitevalmistajien kanssa, mutta se jää nähtäväksi.

Dartin SDK, joka sisältää editorin ja monta tarpeellista liitännäistä, on hyvin koottu paketti erilaisia ominaisuuksia. Se on helppo ladata Dartin kotisivuilta, purkaa koneelle ja käynnistää saman tien. Mikäli Eclipse-editori on tuttu, pääsee DartEditorin sisälle helposti.

Dart-sovellusten kirjoittaminen oli kokonaisuudessaan melko vaivatonta. Koodia on helppo lukea ja kirjoittaa. Verrattuna JavaScriptiin Dart-sovellukset ovat yleensä koodiltaan pienempiä sen optimoinnin takia. Moni komento, jota JavaScript käyttää muun muassa DOM-elementtien muuttamiseen, on Dartista karsittu pois ja vain välttämättömimmät jätetty jäljelle. Tämä selkeyttää koodaamista eikä pakota koodaajaa muistamaan liikaa erilaisia komentoja. Tämä puolestaan saattaa nopeuttaa kielen oppimista, ja sovellusten teko muuttuu helpommaksi.

Google kehittää Dartia edelleen aktiivisesti. Dartin kotisivuilla http://www.dartlan.org on linkkejä muille sivuille, kuten forumeille ja uutisiin. Uutisia Dartista ilmestyy lähes päivittäin. Uusia ominaisuuksia tai parannuksia on tulossa, joten omaa koodia saa olla jatkuvasti muuttamassa, mikäli muutos Dartin ydinkoodissa on suuri.

## Lähteet


1 Chrome Release Beta 02.149.29. 2008. Verkkodokumentti. Google. <http://googlechromereleases.blogspot.fi/2008/09/beta-release-0214929.html>. Luettu 3.4.2013.

2 Chrome Releases. 2013. Verkkodokumentti. Google. <http://googlechromereleases.blogspot.fi/>. Luettu 7.4.2013.

3 ColdFusion history. Verkkodokumentti. ColdFusiontimes. <http://www.coldfusiontimes.com/index.cfm?action=cftimes&view=history>. Luettu 7.4.2013.

4 Kumar, Anil. 2013. CLR C# and dot net version comparison chart. Verkkodokumentti. <http://www.c-sharpcorner.com/Blogs/11177/clr-C-Sharp-and-dot-net-version-comparison-chart.aspx>. Luettu 7.4.2013.

5 JSRs: Java Specification Requests. 2006. Verkkodokumentti. Oracle Corporation. <http://jcp.org/en/jsr/detail?id=244>. Luettu 7.4.2013.

6 JSRs: Java Specification Requests. 2009. Verkkodokumentti. Oracle Corporation. <http://jcp.org/en/jsr/detail?id=316>. Luettu 7.4.2013.

7 Unsupported Historical Releases. 2013. Verkkodokumentti. The PHP Group. <http://php.net/releases/index.php>. Luettu 7.4.2013.

8 Riding Rails: Releases. 2013. Verkkodokumentti. Ruby on Rails. <http://weblog.rubyonrails.org/releases/>. Luettu 7.4.2013.

9 Cascading Style Sheets Leve 2 Revision 1 (SS 2.1) Specification. 2011. Verkkodokumentti. W3C. <http://www.w3.org/TR/2011/PR-CSS2-20110412/>. Luettu 7.4.2013.

10 Flash Player Version History. 2010. Verkkodokumentti. Wavelength Media. <http://www.mediacollege.com/adobe/flash/player/version/>. Luettu 7.4.2013.

11 Release Notes | Flash Player 11, AIR 3. 2011. Verkkodokumentti. Adobe Systems. <http://helpx.adobe.com/x-productkb/multi/release-notes-flash-player-11.html>. Luettu 7.4.2013.

12 Resig, John. 2008. Versions of JavaScript. Verkkodokumentti. <http://ejohn.org/blog/versions-of-javascript/>. Luettu 7.4.2013.

13 Google Releases Dart Editor for Windows, Max OS X, and Linux. 2011. Verkkodokumentti. Google. <http://dartr.com/google-releases-dart-editor/>. Luettu 13.4.2013.

14 ASP.NET Tutorial. Verkkodokumentti. W3Schools. <http://www.w3schools.com/aspnet/default.asp>. Luettu 3.4.2013.

15 Using Tropo WebAPI with ColdFusion. Verkkodokumentti. Tropo. <http://blog.tropo.com/tag/coldfusion/>. 14.6.2012. Luettu 14.4.2013.

16 Oracle and Java | Technologies. 2013. Verkkodokumentti. Oracle Corporation. <http://www.oracle.com/us/technologies/java/overview/index.html>. Luettu 14.4.2013.

17 What is PHP. 2013. Verkkodokumentti. The PHP Group. <http://www.php.net/manual/en/intro-whatis.php>. Luettu 3.4.2013.

18 Ruby on Rails. 2013. Verkkodokumentti. Ruby on Rails. <http://rubyonrails.org/>. Luettu 3.4.2013.

19 Adobe Flash Player. 2013. Verkkodokumentti. Adobe Systems. <http://www.adobe.com/software/flash/about/>. Luettu 13.4.2013.

20 A Short History of JavaScript. 2013. Verkkodokumentti. W3C. <http://www.w3.org/community/webed/wiki/A_Short_History_of_JavaScript>. 27.6.2012. Luettu 12.2.2013.

21 About Silverlight. 2013. Verkkodokumentti. Microsoft. <http://www.microsoft.com/silverlight/what-is-silverlight/>. Luettu 18.4.2012.

22 Walrath, Kathy ja Ladd, Seth. 2012. Dart, Up and Running. Sebastopol, CA. O'Reilly Media.

23 JavaScript Libraries. 2013. Verkkodokumentti. W3Schools. <http://www.w3schools.com/js/js_libraries.asp>. Luettu 4.4.2013.

24 Walrath, Kathy ja Ladd, Seth. 2012. What is Dart? Sebastopol, CA. O'Reilly Media.

25 Opening Keynote: Dart. 2011. Verkkodokumentti. GOTO. <http://gotocon.com/aarhus-2011/presentation/Opening%20Keynote:%20Dart,%20a%20new%20programming%20language%20for%20structured%20web%20programming>. Luettu 5.2.2013.

26 Miller, Mark S. 2010. Future of JavaScript. Verkkodokumentti. <https://gist.github.com/paulmillr/1208618>. 16.11.2010. Luettu 5.2.2013.

27 Technical Overview. 2013. Verkkodokumentti. <http://www.dartlang.org/docs/technical-overview/>. Luettu 5.2.2013.

28 pub: The Dart Package Manager. 2013. Verkkodokumentti. Google. <http://www.dartlang.org/docs/dart-up-and-running/contents/ch04-tools-pub.html>. Luettu 28.3.2013.

29 dart2js: The Dart-to-JavaScript Compiler. 2013. Verkkodokumentti. Google. <http://www.dartlang.org/docs/dart-up-and-running/contents/ch04-tools-dart2js.html>. Luettu 28.3.2013.

30 Chromium with the Dart VM. 2013. Verkkodokumentti. Google. <http://www.dartlang.org/dartium/>. Luettu 27.3.2013.

31 Dartin toimintaperiaate selaimella avattaessa. Verkkodokumentti. <https://1-ps.googleusercontent.com/sx/s.dart-lang.appspot.com/www.dartlang.org/docs/technical-overview/images/source-flow.png.pagespeed.ce.zpPkjHzQO_.png>. Katsottu 27.3.2013.

32 Quitslund, Phil. 2012. Dart Plugin for Eclipse is Ready for Preview. Verkkodokumentti. <http://news.dartlang.org/2012/08/dart-plugin-for-eclipse-is-ready-for.html>. 13.8.2012. Luettu 2.4.2013.

33 Grobmeier, Christian. 2012. 10 reasons why Dart is cooler than JavaScript. Verkkodokumentti. <http://www.grobmeier.de/10-reasons-why-dart-is-cooler-than-javascript-03012012.html#.UTRgSVdd9ZN>. 3.1.2012. Luettu 4.3.2013.

34 Koch, Peter-Paul. 2012. Dart; or Why JavaScript has already won. Verkkodokumentti. <http://www.quirksmode.org/blog/archives/2011/10/dart_or_why_jav.html>. Luettu 4.3.2013.

35 Java SE Downloads. 2013. Verkkodokumentti. Oracle Corporation. <http://www.oracle.com/technetwork/java/javase/downloads/index.html>. Luettu 1.4.2013.

36 Troubleshooting Dart Editor. 2012. Verkkodokumentti. Google. <http://www.dartlang.org/docs/editor/troubleshoot.html>. Luettu 1.4.2013.

37 Hixie, Ian. 2004. Extending HTML. Verkkodokumentti. <http://ln.hixie.ch/?start=1089635050&count=1>. 7.12.2004. Luettu 27.3.2013.

38 Document Object Model (DOM). 2005. Verkkodokumentti. W3C. <http://www.w3.org/DOM/>. 19.1.2005. Luettu 1.4.2013.

39 Introducing JSON. Verkkodokumentti. JSON. <http://www.json.org/>. Luettu 27.3.2013.

40 Kuronen, Timo. Asiakas-palvelin –malli. Verkkodokumentti. <http://herkules.oulu.fi/isbn9514248635/html/node13.html>. Luettu 28.3.2013.

41 Ladd, Seth. 2013. I/O Library Now Uses Streams. Verkkodokumentti. <http://news.dartlang.org/2013/02/io-library-now-uses-streams.html>. 21.2.2013. Luettu 27.3.2013.

42 HTTP Clients and Servers. 2013. Verkkodokumentti. Google. <http://www.dartlang.org/docs/dart-up-and-running/contents/ch03.html#ch03-io-http>. Luettu 18.3.2013.

## Keräily-peli

Pääohjelman koodi on jaettu kahteen osaan. Tiedostonimi TopDownMovement.dart.

```
import 'dart:html';
import 'dart:math';

//Ladataan kaikki omat luokat
part 'player.dart';
part 'enemy.dart';
part 'pickup.dart';
part 'text.dart';

//Arvoja
CanvasElement canvas = query("#canvas"); //Piirtoalusta
var context = canvas.context2d;
const CANVAS_SIZE = 300; //Canvaksen koko
num playerX = 32;
num playerY = 32;
num enemyX = 256;
num enemyY = 256;
String playerDirection;
int playerHealth = 100;
int playerScore = 0;
int highscore = 0;

void main() {
  //sfx = new Sfx();
  drawEverything(context);
  window.requestAnimationFrame(animate); //Aloitetaan "animaatio"

  //Kun klikataan vasemmalla hiirellä...
  query("#canvas").onMouseDown.listen((MouseEvent e){
    //Jos pelaaja on kuollut
    if(playerHealth <= 0){
      //JOS tuli uusi highscore!
      if (playerScore > highscore) highscore = playerScore;
      //Resetoidaan peli ja palautetaan kaikki arvot takaisin alkutilanteeseen
      playerHealth = 100;playerX = 32;playerY = 32;enemyX = 256;enemyY =
256;playerScore = 0;
      xx = rnd.nextInt(300); // xx ja yy ovat pickup.dart:n omia arvoja
      yy = rnd.nextInt(300);
      playerDirection = null;
      //enemy.dart arvoja
      enemySize = 3; //Vihollisen säde
      sizeTimer = 60; //Sekunti
      showTime = 3; //3 sekuntia. Kun sizeTimer <= 0, niin showTime --;
    }
  });
}

//Piirretään peli
void drawEverything(CanvasRenderingContext2D context){
  //Nollataan ruutu
  context.clearRect(0, 0, CANVAS_SIZE, CANVAS_SIZE);
  //Piirretään se uudestaan!
  context.rect(0, 0, canvas.width, canvas.height);
  var grd = context.createRadialGradient(238, 50, 10, 238, 50, 300);
  grd.addColorStop(0, '#8ed6ff'); // light blue
  grd.addColorStop(1, '#dbf2ff'); // lighter blue
  context.fillStyle = grd;
  context.fill();

  //Piirretään canvaksen alle vähän tilastoja omaan elementtiinsä
  query('#text').text = 'Highscore: $highscore $xx $yy';
```

```
//Pelaajan liikkuminen. Käytetään "onPress" koska jatkuva pohjassa pitäminen aiheutti
toimivuusongelmia
  window.onKeyPress.listen((KeyboardEvent event){
    //print(event.charCode); Tällä voi helposti katsoa kutakin näppäintä vastaavat char-
Codet.
    //Liikkumisuunta
    if(event.charCode == 119) playerDirection = "up"; // W
    else if(event.charCode == 115) playerDirection = "down"; // S
    else if(event.charCode == 97) playerDirection = "left"; // A
    else if(event.charCode == 100) playerDirection = "right"; // D
  });

  //Katsotaan kulkusuunta sekä kentän rajat ja piirretään pelaaja
  if(playerDirection == "up" && playerY > 0) playerY -= 2; // W
  if(playerDirection == "down" && playerY < CANVAS_SIZE) playerY += 2; // S
  if(playerDirection == "left" && playerX > 0) playerX -= 2; // A
  if(playerDirection == "right" && playerX < CANVAS_SIZE) playerX += 2; // D
  Player.draw(context,playerX,playerY);

  //Piirretään vihollinen ja lasketaan sille seuraava paikka pelaajan x ja y suhteen
  if(playerX < enemyX){enemyX -= 1;}
  if(playerX > enemyX){enemyX += 1;}
  if(playerY < enemyY){enemyY -= 1;}
  if(playerY > enemyY){enemyY += 1;}
  Enemy.draw(context, enemyX, enemyY);

  //Osutaanko viholliseen? Jos osutaan, vähennetään pelaajan elämää.
  var playah = new Point(playerX,playerY);
  var enemyh = new Point(enemyX,enemyY);
  var distanceE = Point.distanceBetween(playah, enemyh);
  if (distanceE < enemySize+10) playerHealth -= 1;

  //Osutaanko koalaan?
  //HUOMAA! xx ja yy tulevat pickup.dart:sta!
  if (playerX >= xx && playerX <= xx+32 &&
      playerY >= yy && playerY <= yy+32){
    playerScore += 1000;
    xx = rnd.nextInt(300);
    yy = rnd.nextInt(300);
  }
  //Oma classi, johon syötetään seuraavat tiedot: Alusta, teksti, x, y, väri, koko ja
tasaus.
  Text.text(context, "Health: $playerHealth", 0, 20, "red", "20px", "left");
  Text.text(context, "Score: $playerScore", CANVAS_SIZE, 20, "yellow", "20px", "right");
  Text.text(context, "Size up: $showTime", enemyX,enemyY-enemySize-
4,"black","16px","center");
}

//Mikäli pelaaja on kuollut, piirretään tämä
void drawEnding(CanvasRenderingContext2D context){
  context.clearRect(0, 0, CANVAS_SIZE, CANVAS_SIZE);
  //Tuliko uusi kärkitulos?
  if (playerScore > highscore){
    Text.text(context,"NEW HIGHSCORE: $playerScore!", CANVAS_SIZE/2,CANVAS_SIZE/2-
64,"red","24px","center");
  }
  Text.text(context,"FINAL SCORE: $playerScore", CANVAS_SIZE/2,CANVAS_SIZE/2-
32,"black","24px","center");
  Text.text(context,"GAME OVER",CANVAS_SIZE/2,CANVAS_SIZE/2,"blue","32px","center");
  Text.text(context,"Click to Restart", CAN-
VAS_SIZE/2,CANVAS_SIZE/2+32,"black","16px","center");
}

//Animaatio rullaa.
void animate(num time){
  //Kaikki piirtäminen tapahtuu drawEverythingissä, jos pelaajalla on healthia jäljellä
  if(playerHealth > 0){
    //Jos pelaaja elossa
    drawEverything(context); //Piirretään kaikki paitsi koala.
    Pickup.pickup(context); //Piiretään koala. Kerää koalia -> saa pisteitä!
  }
  else{
    //Jos pelaaja kuollut
    drawEnding(context);
  }
  //Loopataan animaatiota
  window.requestAnimationFrame(animate);
}
```

Neljä luomaani luokkaa kommentteineen. Luokat toimivat pääohjelman kanssa.

/********/ enemy.dart /********/

```
//Liitetään osaksi pääohjelmaa
part of TopDownMovement;
//Omat arvot
int enemySize = 3;
int sizeTimer = 60;
int showTime = 3;
//Enemy-luokka ja sen piirto
class Enemy {
  static draw(CanvasRenderingContext2D context, num x, num y){
    //Kasvatateen vihollista pikkuhiljaa
    sizeTimer --;
    //Framet
    if (sizeTimer <= 0){
      sizeTimer = 60;
      showTime --;
    }
    //Sekunnit
    if (showTime <= 0){
      showTime = 3;
      enemySize += 1;
    }
    //Piiretään vihollinen, x ja y saadaan pääohjelmasta
    context.beginPath();
    context.lineWidth = 2;
    context.fillStyle = "red";
    context.strokeStyle = "red";
    context.arc(x, y, enemySize, 0, PI*2, false);
    context.fill();
    context.closePath();
    context.stroke();
  }
}
```

/********/ pickup.dart /********/

```
//Liitetään pääohjelmaan
part of TopDownMovement;

//Omia arvoja
//Kuva saadaan suoraan html-tiedostosta. Kuvan olen laittanut <canvas></canvas> tagien
SISÄÄN.
//Kuvan laittaminen ulkopuolelle piirtäisi sen kuin minkä tahansa normaalin kuvan verk-
kosivulle.
//Kuva on myös mahdollista ladata käyttämällä Dartin omia komentoja, kuten Canvas-
esimerkissä on tehty.
ImageElement koala = document.query('#koala');
Random rnd = new Random();
int xx = rnd.nextInt(300);
int yy = rnd.nextInt(300);

//Piiretään koala
class Pickup {
  static pickup(CanvasRenderingContext2D context){
    context.drawImage(koala, xx, yy, 32,32);
  }
}
```

/********/ player.dart /********/

```
//Liitetään pääohjelmaan
part of TopDownMovement;

//Piirretään pelaaja. x ja y tulevat pääohjelmasta
class Player {
  static draw(CanvasRenderingContext2D context, num x, num y){
    //Piirretään pelaaja
    context.beginPath();
    context.lineWidth = 2;
    context.fillStyle = "orange";
    context.strokeStyle = "orange";
    context.arc(x, y, 10, 0, PI*2, false);
    context.fill();
    context.closePath();
    context.stroke();
  }
}
```

/********/ text.dart /********/

```
//Liitetään pääohjelmaan
part of TopDownMovement;

//Teksti
//Tällä luokalla saa luotua monta erilaista ja monipuolista tekstiä omilla tyyleillään.
class Text {
  static text(CanvasRenderingContext2D context, String message, num x, num y, String col-
or, String fontSize, String align){
    context.font = "$fontSize Calibri";
    context.textAlign = align;
    context.fillStyle = color;
    context.fillText(message, x,y);
  }
}

//Etäisyys
//Kerätään kahden pisteen etäisyydet ja lasketaan niiden etäisyys.
class Point {
  num x;
  num y;
  Point(this.x,this.y);

  static num distanceBetween(Point a, Point b){
    var dx = a.x - b.x;
    var dy = a.y - b.y;
    return sqrt(dx*dx+dy*dy);
  }
}
```

Lopuksi osa html-tiedostoa, niin että ohjelman saa suoritettua.

```
<div id="container">
      <canvas id="canvas" width="300" height="300" class="center">
          <img id ="koala" src="images/Koala.jpg"/>
        </canvas>
      <p id="text"></p>
    </div>

    <script type="application/dart" src="TopDownMovement.dart"></script>
    <script src="packages/browser/dart.js"></script>
```