Opinnäytetyö (AMK)

Tietojenkäsittely

Sähköisen liiketoiminnan järjestelmät

2015

Tommy Johansson

OPPIMISPELIN SUUNNITTELU JA TOTEUTUS

– case Turun Yliopiston ViLLE-oppimisympäristö

TURUN AMMATTIKORKEAKOULU
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tietojenkäsittely | Sähköisen liiketoiminnan järjestelmät

Kevät 2015 | 40+10

Päivi Killström

Tommy Johansson

OPPIMISPELIN SUUNNITTELU JA TOTEUTUS– CASE TURUN YLIOPISTON VILLE-OPPIMISYMPÄRISTÖ

Tämän opinnäytetyön kirjallisessa osuudessa käydään läpi suunnittelu ja toteutusvaiheet Turun Yliopiston ViLLE-oppimisympäristöön kehitetystä Kuplamatikka-oppimispelistä. Teknologiaa ja pelejä hyödynnetään yhä enemmän nykypäivän erilaisissa opetustilanteissa niin työelämässä kuin kouluissa. Siksi on hyvä tietää mitä pelien hyödyntämisellä voidaan saavuttaa sekä miten se parantaa tai tukee oppimista. Tämä opinnäytetyö toteutui ViLLEen Turun Yliopiston pyynnöstä kehittää alakouluikäisille uusi matematiikan oppimispeli.

Opinnäytetyön oppimispeli toteutettiin projektina Turun Yliopiston informaatioteknologian laitoksen ViLLEen syksyllä 2014. Oppimispelin tarkoituksena on opettaa ja tukea alakouluikäisten matematiikan peruslaskutoimituksien oppimista. Peli toteutettiin Phaser-pelimoottoria hyödyntäen selainpohjaiseen ympäristöön, jossa ViLLE toimii. Ohjelmointikielenä käytettiin HTML-standardeista HTML5:sta, joka on HTML5-standardin, CSS3:sen ja JavaScriptin yhdistelmä.

Projekti toteutui onnistuneesti ja oppimispeli integroidaan ViLLEen kevään 2015 aikana.

ASIASANAT:

oppimispelit, ViLLE, HTML5, JavaScript, matematiikka

BACHELOR´S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Business Information Technology | e-Business System

Spring 2015 | 40+10

Päivi Killström

Tommy Johansson

DESIGNING AND DEVELOPING EDUCATIONL GAMES
– CASE COLLABORATIVE EDUCATIONAL TOOL VILLE IN UNIVERSITY OF TURKU

This thesis deals with designing and developing an educational game which is implemented to the collaborative educational tool ViLLE for University of Turku. Nowadays technology and games are spread and used widely in different educational situations both in working life and schools. That is why it is good to know what are the purpose and goals of using games as a part of education. This thesis was done to ViLLE as a request from University of Turku to develop an educational mathematic game for children in primary school.

The educational game discussed in this thesis was engineered as a project to ViLLE in autumn 2014. The purpose of the game is to teach and support children in primary school to learn basic arithmetic. The game was developed by using Phaser game engine. HTML5-standard, CSS3 and JavaScript were used to develop Bubblemath.

The project was carried out successfully and the educational game is will be released into ViLLE during the spring 2015.

KEYWORDS:

educational games, HTML5, JavaScript, mathematic, arithmetic

SISÄLTÖ

LIITTEET

KUVAT

KUVIOT

TAULUKOT

1 JOHDANTO

Pelit ovat yksi vanhimmista ja kunnianarvoisimmista oppimisen motivoinnin keinoista. Historiaa tutkimalla pelien ja leikkien on havaittu toimineen aina niin ihmiskunnan kuin luonnonkin tapana opettaa. Niiden yksi perimmäisistä motivaatioista on aina kautta aikojen ollut opettaminen, vaikka ajan saatossa ympärille on kasvanut kymmeniä uusia motivaatioita. (Crawford, 1984.) Jo kirjoituksessaan Valtio, Platon kuvailee pelien ja oppimisen yhteyttä suositellen pelien käyttöä lasten opettamisessa. Hän on mm. todennut aikoinaan, että ”henkilöstä voidaan saada selville enemmän tunnin mittaisessa pelissä, kuin vuoden mittaisessa keskustelussa”. Nykypäivän alati kehittyvä teknologia on saanut tietokoneet ja videopelit tuntumaan yhä enemmän viihdeteollisuudelta, mutta on silti ymmärrettävä sen suuri vaikutusvalta myös oppimisen työkaluna. Videopelejä ja simulaatioita hyödynnetään mm. valtion virastoissa, kuten armeijassa, sairaaloissa tai kouluissa, sekä eri yhtiöiden ja yritysten työkaluna koulutuksissa ja opetuksissa. (Michael & Chen 2005, xv.) Tästä syystä on hyvä pohtia tietotekniikan ja digitaalisten pelien hyödyntämistä oppimisen työkaluina.

Tämän opinnäytetyö käsittelee oppimispelin suunnittelua ja toteutusta. Opinnäytetyön toimeksiantajana oli Turun Yliopiston informaatioteknologian laitoksen tutkimusryhmä, joka vastaa ViLLE-oppimisympäristön (ville.cs.utu.fi) kehityksestä. Tilaajan antaman idean pohjalta luotiin matemaattinen oppimispeli, joka myöhemmin integroidaan osaksi ViLLEä. Kuplamatikka on ViLLEn kautta pelattava selainpohjainen oppimispeli, joka on toteutettu HTML5/Javascript ohjelmointikielellä. Tarkoituksena oli luoda uusi peruslaskutoimituksia harjoittava matemaattinen peli, jota hyödynnetään alustavasti alakouluikäisten matematiikan tunneilla.

Opinnäytetyön yleisessä osiossa kerrotaan mitä oppimispeleillä tarkoitetaan ja annetaan niin ulkomainen kuin kotimainenkin esimerkki oppimispelistä sekä kerrotaan lyhyesti ViLLEstä. Suunnittelu-luvussa paneudutaan pelin suunnittelulle tärkeisiin elementteihin, kuten pelin suunnitelmaan (eng. high concept) sekä pelin eri kehitysalustojen ja pelimoottoreiden vertailuun. Toteutus-luku koostuu oppi-

mispelin toteutuksen kannalta oleellisista asioista sekä Kuplamatikan pelimekaniikan ongelmista ja niiden ratkaisutavoista. Lopuksi annetaan pohdintoja ja yhteenveto opinnäytetyöstä.

2 OPPIMISPELEISTÄ YLEISESTI

Oppimispelit esiteltiin ensi kertaa ”hyötypelit”-nimikkeellä (eng. serious games) 1970-luvulla Clark Abtin kirjassa Serious Games. Termillä tarkoitettiin pelejä, joilla oli tarkka ja huolellisesti mietitty opetuksellinen tarkoitus eikä niitä ole tarkoitus pelata huvin vuoksi. 1980-luvulta lähtien, viihdeteollisuuden kasvaessa, pelien ja oppimisen yhdistelmää on alettu kutsua ”edutainmentiksi”, joka myöhemmin muotoutui pelipohjaiseksi oppimiseksi (eng. game-based learning) (Egenfeldt-Nielsen ym 2011, 9-10). Nykyään hyötypelit käsittelevät suurempaa kokonaisuutta, joka voidaan jakaa viiteen alalajiin; edutainment, mainospelit, markkinointipelit, poliittiset pelit sekä harjoitus- ja simulaatiopelit. (Alvarez ym. 2007.) Sittemmin alalajeihin on lisätty vielä erikseen oppimispelit.

Tässä luvussa kerrotaan lyhyesti oppimispelien hyödyistä sekä kahdesta erilaisesta oppimispelistä, DragoBox:sta ja Ekapelistä. Lopuksi esitellään Turun Yliopistossa kehitettävää ViLLE-oppimisympäristöä, johon on integroitu useita oppimispelejä mukaan lukien opinnäytetyön Kuplamatikka - peli.

2.1 Oppimispelien hyödyntäminen opetuksessa

Pelaaminen ja leikkiminen mielletään usein oleelliseksi osaksi lapsen sosiaalista ja tiedollista, eli kognitiivista kehitystä. Lapset oppivat pelien kautta kanssakäymistä muiden lasten kanssa, luovuutta, itsensä kehittämistä sekä sääntöjen noudattamista. Pelien avulla pystytään edesauttamaan myös abstraktin ajattelutavan sekä symboliikan kehitystä. (Mayer, 2013.) Pelejä voidaan luonnehtia kuuden ominaisuuden mukaan, jotka yhdistettynä pitävät yllä pelaajan mielenkiintoa. Nämä ovat säännöt; kilpailu, haastavuus, vastakkainasettelu tai konflikti; tavoite tai tehtävät; vuorovaikutus; lopputulos ja palaute sekä tarina tai esitys. Peli itsessään voi olla niin lautapeli, korttipeli, videopeli kuin simulaatio, mallien rakentaminen, roolipeli kuin mikä tahansa muu aktiviteetti, jossa edellä mainitut ominaisuudet toteutuvat.(Mitchell & Savill-Smith 2004, 3.)

Marc Prensky (2005) määrittelee kaksi pääsyytä pelien hyödyntämiselle opetuksessa; opiskelijoiden radikaali muuttuminen sekä heidän motivointi uudella tavalla. Ensimmäisen väitteensä Prensky perustelee digitaalisessa ja teknologisessa ympäristössä kasvamisen muuttaneen nykynuorten tapaa vastaanottaa ja ajatella informaatiota. Toisen syyn Prensky perustelee vaikeudella ylläpitää motivaatiota. Nykyteknologia ja etenkin videopelit ovat mahdollistaneet opiskelijoiden motivoinnin uudella tavalla. Kaikki niin sanottu formaali opetus, luokkahuoneista etäopetukseen, kärsii samasta ongelmasta – opiskelijoiden motivoinnin vaikeudesta läpi tuntien, kurssien tai lukukausien. Prensky painottaa etenkin motivoinnin tärkeyttä oppimisessa.

Monet oppimista pelien avulla tutkineet ovat luoneet erilaisia listauksia hyvän oppimispelin piirteistä. Osa näistä elementeistä on samoja Mitchell ja Savill-Smithin kuuden mielenkiintoa ylläpitävän ominaisuuden kanssa, kuten säännöt ja tavoitteet, palautteen anto haastavuus ja kilpailu. Kirjassaan Digital Game-Based Learning (2001) Prensky esittelee oman listansa oppimisen eri tyypeistä ja niille mahdollisista pelityyleistä (taulukko 1). Näistä pelityypeistä monet saattaisivat toimia paremmin leikkeinä kuin digitaalisina peleinä.

Taulukko 1. Oppimisen eri tyypit ja niihin liittyvät mahdolliset pelityylit.(Prensky, 2001).

"Content"	Examples	Learning activities	Possible Game Styles
Facts	Laws, policies, product specifications	questions memorization association drill	game show competitions flashcard type games mnemonics action, sports games
Skills	Interviewing, teaching, selling, running a machine, project management	Imitation Feedback coaching continuous practice increasing challenge	Persistent state games Role-play games Adventure games Detective games
Judgment	Management decisions, timing, ethics, hiring	Reviewing cases asking questions making choices (practice) feedback coaching	Role play games Detective games Multiplayer interaction Adventure games Strategy games
Behaviors	Supervision, self-control, setting examples	Imitation Feedback coaching practice	Role playing games
Theories	Marketing rationales, how people learn	Logic Experimentation questioning	Open ended simulation games Building games Constructing games Reality testing games
Reasoning	Strategic and tactical thinking, quality analysis	problems examples	Puzzles
Process	Auditing, strategy creation	System analysis and deconstruction Practice	Strategy games Adventure games
Procedures	Assembly, bank teller, legal	imitation practice	Timed games Reflex games
Creativity	Invention, Product design	play	Puzzles Invention games
Language	Acronyms, foreign languages, business or professional jargon	Imitation Continuous practice immersion	Role playing games Reflex games Flashcard games
Systems	Health care, markets, refineries	Understanding principles Graduated tasks Playing in microworlds	Simulation games
Observation	Moods, morale, inefficiencies, problems	Observing Feedback	Concentration games Adventure games
Communication	Appropriate language, timing, involvement	Imitation Practice	Role playing games Reflex games

Oppimispeliä mietittäessä on hyvä tutkia ja pohtia minkä tyyppistä oppimista halutaan tapahtuvan sekä kartoittaa, aiheesta riippumatta, millaista oppimista mistäkin oppimateriaalista on jo olemassa.(Prensky, 2005.) Kuten kuvasta näkyy, esimerkiksi faktaan perustuvaan oppimiseen löytyy useampia pelityylejä ja oppimisen aktiviteetteja. Opetettavia asioita voivat olla esimerkiksi politiikka, laki sekä

tuotteen määrittely tai, kuten tässä opinnäytetyössä, peruslaskutoimitukset, joihin liittyvät matemaattiset ongelmat.

Oppimispeliä pohtiessa, suunnitellessa, rakentaessa tai testatessa on hyvä muistaa neljä oppimispelin periaatetta, jotka kaikkien oppimispelien tulisi täyttää. Prensky määrittelee nämä seuraavasti:

1. Pelin viihdyttävyys – viihdyttävä oppimispeli vetää luokseen ja opettaa pelaajia myös kohdeyleisön ulkopuolelta

2. Pelaaja vai opiskelija? – hyvin tehty peli saa kohdeyleisön tuntemaan itsensä pelaajaksi opiskelijan tai harjoittelijan sijaan poistaen perinteisen oppimistilanteen ennakkoluulot.

3. Pelikokemus – unohtumaton ja hyvä pelikokemus saa pelaajan helpommin palaamaan pelin äärelle sekä suosittelemaan sitä muille.

4. Kehittyminen - hyvin toteutettu oppimispeli auttaa pelaajaansa kehittymään oppimateriaalin aihealueella merkittävän nopeasti mitä enemmän pelaaja kuluttaa peliin aikaa. Pelin tulisi myös rohkaista miettimään opittuja asioita.

Jotta peli voidaan laskea oppimispeliksi, tulisi periaatteiden järjestyksen pysyä muuttumattomana – ensin hauskuus, sitten oppiminen. Monilla oppimispeleillä hauskuus löytyy periaatteiden listalta, mutta hyvin paljon alempana. Tällöin pelit saattavat osoittautua usein jonkun henkilön omaksi teoriaksi tavasta opettaa tai vaihtoehtoisesti köyhäksi simulaatioksi kauniilla kuorrutuksella.(Prensky, 2005.)

2.2 Palkittu ulkomaalainen esimerkkipeli: DragonBox Algebra

DragonBox Algebra +5 on ranskalaisen opettajan, Jean-Baptiste Huynhin, ja We-WantToKnow -tiimin ensimmäinen matemaattinen oppimispeli ja se on äänestetty International Mobile Gaming Awards:lla maailmanlaajuisesti yhdeksi parhaimmista hyötypeleistä (eng. serious games). Sen julkaisu herätti useat tutkijat tutkimaan lisää pelillisen oppimisen hyötyjä.

DragonBox pelisarjan tarkoituksena on opettaa kenet tahansa ratkaisemaan algebran yhtälöitä vaistonvaraisen ja motivoivan pelin avulla. Pelaajan tavoitteena on muokata hahmoa kunnes se on yhtälön muuttujan tavoin eristetty pelilaudan toiselle puolelle. Mielenkiintoisen pelistä tekee pelaajan huomion kiinnittäminen pois matemaattisen pelin pelaamisen ajatuksesta ja sen korvaaminen ‘oman lohikäärmeen kasvattamisesta’. Näin pelaaja ei välttämättä ymmärrä opettelevan algebran perusteita vaikka oppisi sen perusteita, kuten yhtälön tasapainottamisen. Pelin edetessä hahmot muuttuvat vähitellen numeroiksi ja muuttujiksi kunnes pelaaja ratkoo yhtälöitä normaaliin tapaan. Yhtälöiden laskennan säännöt opitaan kokeilun avulla. Tehokas muuttujien muokkaaminen sekä yhtälön ratkaiseminen yhä harvemmalla siirrolla antaa pelaajalle suuremman tähtimäärän pelatusta tasosta.

WeWantToKnow hyödynsi osaamistaan kehittämällä jatko-osan menestyksekkäälle DragonBox Algebra 5+:lle. Siinä missä Algebra 5+ tarkoitus oli opettaa viisi vuotiaita ja sitä vanhempia ratkomaan yhtälöitä, algebra 12+ ymmärtämään algebran hankalempien, ensimmäisen ja toisen asteen yhtälöiden ratkomista. Sen yhtälöt sisältävät mm. murtolukuja, sulkeita ja parametrien luontia. WeWantToKnow tarjoaa myös perinteistä oppimateriaalia sekä kynä- ja paperitehtäviä, jotta pelaaja voi olla varma riittävästä oppimisestaan ja osaamisestaan perinteisten yhtälötehtävien ratkaisemisessa. Tätä varten, poikkeuksena algebra 5+:n, algebra 12+ antaa mahdollisuuden harjoitella jokaista operaattoria myös yksittäin, jotta pelaaja olisi valmis kynä- ja paperitehtävien yhtälöihin.

DragonBox on hyvä esimerkki oppimispelistä, joka on kehitetty tukemaan oppimista viihdyttävällä tavalla. Oppimispelit tarjoavat hyödyllisen lisän etenkin oppimisvaikeuksista kärsivien oppilaiden opettajille. Vaikka oppimispelit eivät soveltuisikaan täysin kaikkien opetussuunnitelman aihealueiden opetukseen, voidaan sitä käyttää laajasti joko aihealueen täydentämisessä tai kokonaan yksittäisen aihealueen opettamiseen. Peleihin rakennettava suunnitelmallinen sisältö sekä looginen oppimisen polku helpottaa niin oppimisen motivoinnissa, sosiaalisten taitojen kehittämisessä kuin opiskelijalle sopivan oppimistavan löytämi-

sessä.(Mannila ym. 2007, 71-73.) DragonBox:n pelillinen maailma auttaa oppilaita unohtamaan mahdolliset negatiiviset oppimista jarruttavat ennakkoluulot matematiikkaa kohtaan.(Ashcraft, 2002.) Pelaajan odotetaan viihtyvän pelin äärellä samalla sisäistäen matemaattisia laskukaavoja omalla kokeilunhalun sekä loogisen päättelykyvyn avulla. Osalle toimintatapa sopii, toisille taas ei, aivan kuten mikä tahansa muu opetusmenetelmä. Peli soveltuu myös hyvin ylöspäin eriyttäväksi materiaaliksi taitaville oppilaille.

2.3 Kotimainen esimerkkipeli: Ekapeli - Matikka – Graphogame Math

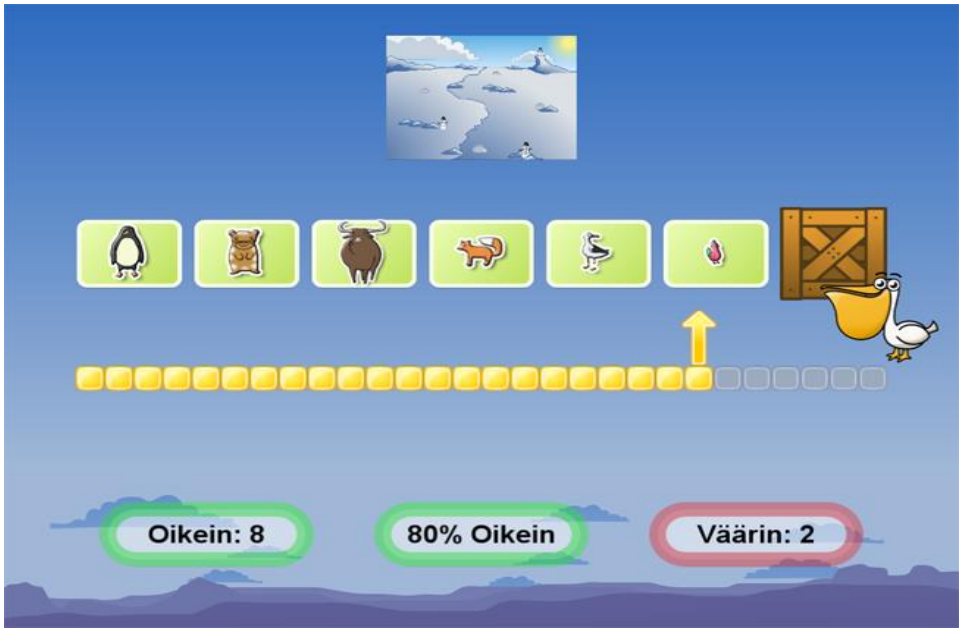
Toisena hyvänä esimerkkinä nykyteknologian ja pelien hyödyntämisestä opetuksessa on Niilo Mäki Instituutin (NMI) kehittämä Ekapeli – oppimisympäristö, jonka käyttö on hyvin laajaa suomalaisten esi- ja alakouluikäisten parissa. Ekapeli on professori Heikki Lyytisen, Jyväskylän yliopiston ja Niilo Mäki Instituutin työryhmän kehittämä oppimispeli tietokoneelle ja Android-mobiililaitteille. Sen ensimmäisenä ajatuksena oli harjoittaa lukutaidon perusteita, mutta sittemmin Ekapelistä on laadittu useampia eri versioita. Ekapelin pelit ovat ilmaisia, mutta kaikkien ominaisuuksien käyttäminen ja täyden hyödyn saaminen vaatii ilmaisen rekisteröitymisen. Rekisteröitynyt käyttäjä pystyy ylläpitämään pelituloksiaan ja tasoaan verkkoyhteyden avulla missä tahansa kun taas rekisteröimättömän pelaajan tiedot tallentuvat vain yhdelle koneelle. Tässä luvussa perehdytään matematiikan harjoittelemiseen suunnattuun Ekapelin versioon.

Ekapeli-Matikka on ”tietokonepeli, joka harjoittaa yksi-yhteen vastaavuutta, vertailua, järjestämistä, lukusanan, -määrän ja numerosymbolin vastaavuutta sekä lukujono- ja yhteenlaskutaitoja”.(Lukimat, 2014a.) Se on suunnattu matematiikan oppimisvaikeuksista kärsiville esi- ja alakouluopetusikäisille oppilaille, joskin siitä hyötyvät kaikenlaiset oppijat.

Pelissä valitaan ääniohjeiden perusteella oikea vastaus annettujen vaihtoehtojen joukosta. Oikeita vastauksia voi olla yksi, kaksi tai kolme ja vastausvaihtoehtoja pelaajan osaamisen mukaan eli peliympäristö on adaptiivinen. Palautejärjes-

telmä auttaa tukemaan pelaajan ratkaisua ja oikean ratkaisun löytämistä on helpotettu vaiheittaisilla vinkeillä, mikäli pelaajan on etsittävä useampi oikea vastaus. Lisäksi peli edesauttaa lapsen ymmärrystä korostamalla ja vahvistamalla oikein tehtyjä valintoja ja toistaen esimerkiksi lukujonotehtävässä muodostuneita lukujonoja.

Eräs Ekapelin tärkeä ominaisuus on lapsen palkitseminen hyvästä suorituksesta. Käytännössä tämä tarkoittaa erilaisten tarrojen keräämistä pelin eri tasoilta. Kun pelattu taso on ohi, tuodaan ruudulle tasolla saavutettu tarramäärä sen mukaan kuinka hyvin pelaaja suoriutui tasosta (kuva 2). Kuten kuvasta huomaa, pelaajalle kerrotaan kuinka monta tehtävää hän laski oikein tai väärin sekä kuinka monta prosenttia vastuksista oli oikein. Tämän perusteella pelaajalle avautuu tietty määrä tarroja, joista hän valitsee yhden ja pääsee sijoittamaan sen omaan tarrakirjaansa.



Kuva 1. Esimerkkipelaaja on onnistunut laskemaan 80 % laskuista oikein.(Lukimat, 2014a).

Ekapeli on hyvä esimerkki pelistä, jonka tarkoituksena on tehdä riskilasten oppimisesta helpompaa ja tehokkaampaa etenkin luku- ja kirjoitustaidoissa. Esimerkiksi erityisopetuksessa hyödynnetty Ekapeli auttoi lukihäiriöisiä oppilaita saavut-

tamaan sekä ohittamaan vertailuryhmänä olleet normaalilukutaitoiset oppilaat.(Saine, 2010.) Ekapelin suosio on kasvanut myös kotioloissa sen ilmaisen saatavuuden takia. Tällä hetkellä Ekapeliin kirjautuu päivittäin keskimäärin 6000 pelaajaa ja rekisteröityneitä lasten pelitunnuksia on lähes 270 000.(Lukimat, 2014b.)

2.4 ViLLE-oppimisympäristö

ViLLE on Turun yliopiston Informaatioteknologian laitoksella kehitetty oppimisympäristö, joka sisältää useita erilaisia oppimiseen kehitettyjä tehtävätyyppejä. ViLLEn avulla opettajat voivat tehdä yhteistyötä niin oppimateriaalin, tuntisuunnitelmien kuin tehtävien jakamisen, kommentoimisen sekä arvioimisen suhteen. ViLLE kerää tietoa oppilaiden suoriutumisesta ja oppimisesta, joiden avulla opettajat voivat kehittää opetusta ja oppimistulosten arviointia. ViLLEn avulla arviointia voidaan helposti muuttaa jatkuvaksi arvioinniksi parantaen opettajien ja opiskelijoiden vuorovaikutusta. Näinollen ViLLE työkaluna sekä tukee oppilaiden arvioimista että antaa mahdollisuuden oman opetuksen reflektoimiseen ja kehittämiseen.(Laakso, 2010.) ViLLEssä on tällä hetkellä noin 800 opettajaa ja 12 000 opiskelijaa ja sen käyttöä on laajennettu myös ulkomaille. Lukion- ja peruskoulunopettajat pääsevät käyttäjäksi ViLLE-koulutuksen kautta ja korkeakouluopettajat voivat anoa tunnuksiaan ViLLEn kotisivuilta.(ViLLE, 2014.) Yleisesti ottaen ViLLEä käytetään alakouluista yliopistoon.

Tässä opinnäytetyössä luotu Kuplamatikka-peli integroidaan osaksi ViLLE-järjestelmää. ViLLEä käyttämällä on saatu hyviä tuloksia alakoulun matematiikan opetuksessa. Tutkimukset osoittavat oppimispelien innostavan ja motivoivan oppilaita harjoittelemaan matematiikan tehtäviä sähköisesti.(Kurvinen ym, 2014.) Kurvinen toteaakin matematiikan opetuksessa ViLLEn tarjoavan ”perinteiseen oppikirjaan verrattuna huomattavan määrän etuja. Tietokone on väsymätön harjoittelukumppani ja pystyy muodostamaan suuren määrän tehtäviä annetuilla alkuehdoilla, joten tekeminen ei lopu kesken ja tehtäväsarjat muuttuvat pelikerrasta toi-

seen. Samoja laskuja on mahdollista harjoitella eri tavoin esitettyinä ja havainnollistettuna sekä hieman erilaisin vastausmekanismein, jolloin rutiinilaskuihin tulee enemmän vaihtelua”.(Kurvinen, 2014.)

3 OPPIMISPELIN SUUNNITTELU JA VALMISTELU

Pelisuunnittelija Doug Churchin lyhyt määritelmä suunnittelusta on, että ”suunnittelu on peli; ilman sitä pelit sisältäisivät vain dataa, mutta ei lainkaan kokemusta.” Tämä toteamus on osoittautunut todeksi sillä useimmiten pelit kehittyvät ideasta, joka halutaan toteuttaa. Tästä syystä pelin kehitykselle oleellisin askel on hyvän suunnitelman laatiminen, jossa kartoitetaan mm. niin syy ja seuraus-suhteet kuin pelityyli sekä tuleva asiakaskunta.(Mitchell 2012,43.)

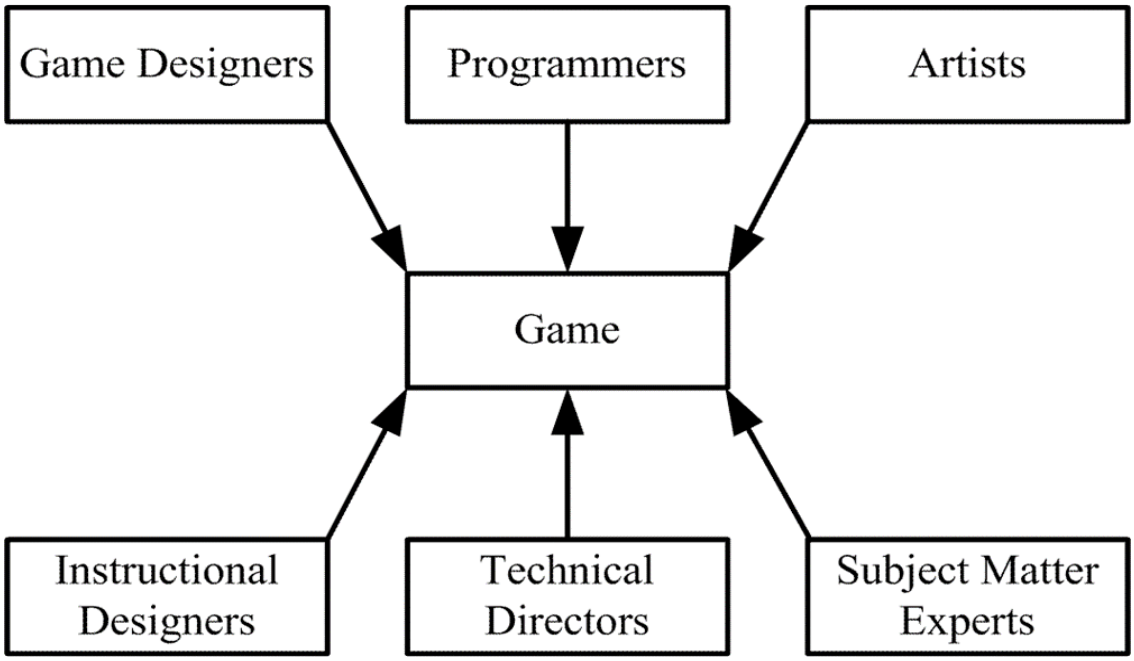
Tässä luvussa käsitellään oppimispelin suunnitteluvaihetta ja vertaillaan vaihtoehtoisia pelinkehitysalustoja sekä pelimoottoreita. Lopuksi kerrotaan oppimispelin tason mekaniikasta ja rakenteesta suunnitteluvaiheessa.

3.1 Pelin suunnittelu

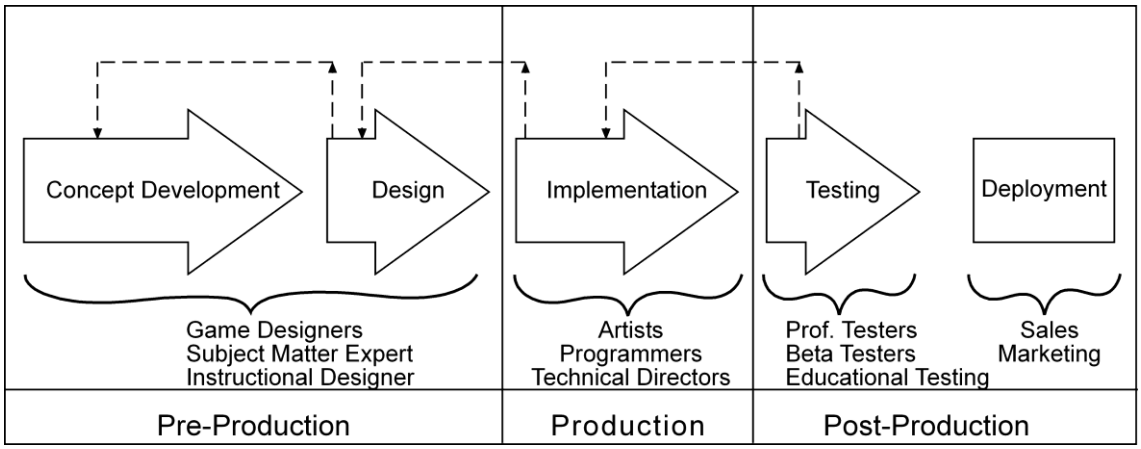
Pelin suunnittelulla tarkoitetaan prosessia, jonka pohjalta peli rakennetaan. Pelien suunnitteluvaiheessa luodaan peli-idean suunnitelma (eng. high concept), jossa kartoitetaan idean kehitys, pelin motivaatiot, ominaisuuksien ja kohderyhmän määritykset, pelin tarina tai juoni sekä pelin yleisilme. Pelin suunnitteluvaiheessa ideoimiseen (eng. brainstorm) osallistuvat suunnittelijoiden lisäksi sovelluskehittäjät, graafikot, tarinan suunnittelijat (eng. loremaster) ja projektin johtohahmot kuten tiimin vetäjät ja projektipäälliköt. Oppimispeleissä suunnitteluun osallistuu vielä opettamiseen erikoistuneita henkilöitä, joita ovat esimerkiksi opettajat, lehtorit tai opetusalan professorit.

Eräänlaisessa mallissa opetuspelin toteutukseen osallistuva työryhmä koostuu niin sovelluskehittäjistä, graafikoista, pelisuunnittelijoista, teknisistä ohjaajista kuin alan asiantuntijoista sekä ohjeistuksen suunnittelijoista (kuvio 1). Itse toteutuksen prosessi on jaettu kolmeen osaan; alku-, pää- ja jälkituotanto (kuvio 2). Alkutuotantovaiheeseen osallistuu pelin suunnittelijat sekä aihealueen eri asiantuntijat. Tämä vaihe on niin sanottu suunnitteluvaihe, jossa pelin konsepti eli rakenne luodaan ja suunnitellaan. Tuotantovaiheessa kehitykseen osallistuvat

graafikot, sovelluskehittäjät ja tekniset ohjaajat, jotka pyrkivät toteuttamaan ja sisällyttämään suunnitteluvaiheessa määritetyt elementit peliin ja vastaamaan niihin tehdyistä muutoksista. Viimeisessä vaiheessa eli jälkituotannossa pelin kehitykseen osallistuvat erilaiset testaajat, jotka parhaansa mukaan pyrkivät löytämään muutosta vaativat tai virheelliset elementit valmiista pelistä. Viihdeteollisuuden peleissä tätä vaihetta kutsutaan usein Alpha- ja Beta-testaukseksi. Kun vaadittavat testaukset ja niitä seuranneet muutokset on tehty, peli voidaan julkaista myynti- ja markkinointitiimien toimesta.(E-Games, 2007.)



Kuvio 1. Opetuspelin kehitykseen osallistuvan työryhmän havainnollistaminen.(E-Games, 2007).



Kuvio 2. Opetuspelin kehitysoprosessi.(E-Games, 2007).

ViLLEen integroitavassa opetuspelissä suunnitteluprosessi toteutui lähestulkoon samalla kaavalla. Pienemmässä mittakaavassa toteutettu peli mahdollisti työryhmän tehtävien jakamisen kehittäjän, asiantuntijan ja teknisen ohjaajan kesken. Alkutuotannossa mm. pelin teemaksi asetettiin aritmeettisten taitojen harjoittaminen ja pelimekaniikka kehitettiin. Pelin varsinainen työstäminen alkoi tuotantovaiheessa, jossa yhteistyö tiivistyi teknisen ohjaajan kanssa ja prosessin etenemisessä ketteristä menetelmistä luontevaksi työtavaksi osoittautui Scrum. Kehitysprosessin kaaviosta poiketen testaus toteutettiin jatkuvana kehityksen ohella, mutta lopullinen testaaminen jää käyttäjien varaan. Näin saadaan omalta osalta parannettua lopputulosta ennen pelin käyttöönottoa. Jälkituotantovaiheeseen jäi ainoastaan pelin viimeistely, niin sanotun visuaalisen ilmeen parantelu, sekä valmiin pelin sisällyttäminen osaksi ViLLEä. Pelin julkaisun jälkeen pienemmät korjaukset voidaan tehdä kevyillä päivityksillä.

3.2 Kehitysalustojen vertailu ja alustan valinta

ViLLEen integroitavaa peliä suunnitellessa tärkein elementti oli kehitysalustan valinta eli millä alustalla peliä tullaan pelaamaan. Koska ViLLE toimii selainpohjaisessa ympäristössä, muodosti se pelin alustan valinnalle selkeitä rajoitteita. Pelin tulisi toimia selainpohjaisesti, mutta paras vaihtoehto sen toteuttamiseksi oli vielä epäselvää.

Selainpohjaisia pelinkehitysalustoja on useita. Tällä hetkellä käytetyin ohjelmointikieli selainpohjaisissa peleissä on “HTML5”, joka on terminä viittaa virallisesti HTML-standardiin. Tässä konseptissa HTML5:llä tarkoitetaan HTML5-standardin, css3:sen ja JavasScriptin yhdistelmään. JavaScript itsesään on olio-orientoitunut kieli, joka on rakenteeltaan hyvin erilainen Javaan tai C#:iin verrattuna. Siinä kaikki rakenteet ovat olioita muuttujista funktioihin. HTML5:n ohella suosiota on kasvattanut erinäiset itsenäiset ohjelmistot, jotka helpottavat omilla työkaluil-

laan selainpohjaisten pelien luontia. Näistä tunnetuimpia ovat Unity3D, joka mahdollistaa pelien tekemisen niin työpöytä-, konsoli-, mobiili- kuin selainversionakin (Unity, 2015) sekä Flash formaatit.

Asetettaessa vastakkain, sekä HTML5 että Unity taipuvat moneksi pelinkehityksessä. Lähtökohtaisesti HTML5 kehitysympäristön pystyttäminen on helppoa ja suoraviivaista ohjelmointia aiemmin internetympäristöön tehneille kehittäjille. HTML5 tarvitsee testiympäristöön pääsyn paikalliseen palvelimeen (eng. localhost), joskin pelkän HTML-tiedoston pystyy avaamaan selaimella tarkastelua varten. Siinä missä HTML5 tukeutuu selainten päälle rakennettuihin HTML:ää ja JavaScriptiä hyödyntäviin kirjastoihin (eng. framework), Unity on itsenäinen kehitysalusta sisäänrakennetulla editorilla (eng. editor) sekä yhtenäisellä kehitysympäristöllä.(Develop, 2014.)

Uusimilla selaimilla käytettävät JavaScript-kielen kirjastot ovat tehokkaita ja mahdollistavat näyttävienkin pelien kehittämisen ilman liitännäisiä (eng. plugins). Yleistyvän WebGL-rajapinnan tekniikka hyödyntää suoraan näytönohjainta ja ainoana vaatimuksen on moderni selain. Riippumattomuus erillisistä liitännäisistä voidaan laskea HTML5 hyväksi ominaisuudeksi, vaikka se rajoittaa omalta osin tavallisten alustojen selaintukea.

Monet tämänhetkiset kirjastot ovat avointa lähdekoodia, joka mahdollistaa käyttäjille tarkastelun ja muokkaamisen. Tämä ominaisuus voidaan tulkita sekä hyvänä että huonona puolena. Koodien avoin julkaisu ja helppo muokattavuus tuovat monipuolisuutta, mutta lisäävät samalla virheiden ja koodin rikkinäisyyden riskiä. Huonoina puolina ovat HTML5:n riippuvuus sen niin kutsutusta pääohjelmointikielestä, JavaScriptistä. Siinä puutteellisia ovat pelikehitykselle tyypilliset piirteet, kuten olio-ohjelmoinnissa käytettävä perintä, rajapintojen hyödyntäminen sekä jäsenten käsitteleminen julkisen, yksityisen tai suojatun nimiavaruuden (eng. namespace) kautta. Samoin erilaisten lisäominaisuuksien, kuten videon, äänen tai animaation, sisällyttäminen ei ole suoraviivaista selainten eri standardeista johtuen.

Unityssä hyvänä puolena ovat sen visuaaliset muokkaus-työkalut, joita käyttäjä voi laajentaa omien tarpeidensa mukaan. Unityn vahva kommuuni auttaa löytämään parhaat ratkaisut niin lisäominaisuuksiin kuin liitännäisiin. Sen kolme ohjelmointikieltä, C#, UnityScript, joka toimii kuten JavaScript ja Boo, tuovat omat haasteensa ja etunsa laajan käytettävyyden myötä, joskin näistä kolmesta Boo on vähiten käytetty. Unity tukee runsaasti eri tiedostoformaatteja ja alustoja, joka helpottaa eri elementtien lisäämistä ja edesauttaa optimaalisen formaatin kääntämistä halutulle alustalle. Tämä antaa mahdollisuuden pelin julkaisemisen useammalle alustalle. Ilmaiset lisenssit kattavat suurimman osan Unityn tarjoamista ominaisuuksista, mutta maksettu lisenssi jokaista alustaa kohden saa kokemattomammankin ohjelmoijan harkitsemaan pelimoottorin päivittämistä.(Unity, 2015.)

Unityn suurimmaksi haitaksi koituu yhteistyön tekeminen kehittäjätiimin välillä tai jopa sen puuttuminen. Unityllä, kuten usein suurimmilla ohjelmointityökaluilla, on oma maksullinen tiimien välistä yhteistyötä edistävä palvelintuote, jonka puute saattaa koitua yhteistyön kohtaloksi. Paras ratkaisu on ulkoisten versionhallintajärjestelmien käyttäminen tiedostojen jakamisessa ja versioinnissa, mutta osaa Unityn binääritiedostoista ei silti pystytä tunnistamaan ja siirtämään ulkoisen versionhallinnan avulla. Toinen ongelma Unityssä on sen suorituskyky. Vastikään Unity on aloittanut parantamaan suorituskykyä mm. mobiililaitteissa. Aikaisemmin se käytti ainoastaan yhtä säiettä ja ylimääräisten ydinten hyödyntäminen oli lähes olematonta. Kolmas suurimmista haitoista on Unityn lähdekoodi, joka ei ole avoin millään tavalla pelinkehittäjille, omistipa sitten maksullisen version tai ei. Toisin sanoen, mikäli pelinkehityksessä kohtaa Unityn toimivuuden kannalta ongelman, on odotettava että se korjataan tulevissa Unityn julkaisuversioissa, joissa pahimmassa tapauksessa saattaa kulua useita kuukausia. Tämä asettaa rajoituksia pelimoottorin laajentamiselle itsessään.(Developer Economics, 2014.) Viimeinen ongelma, johon Unityllä pelinkehittäminen ViLLEen pysähtyy, on Unityn liitännäinen, joka vaaditaan Unityllä kehitettyjä pelejä varten. Yksityisessä käytössä liitännäinen ei tuota ongelmaa ja yksi asennuskerta riittää kattamaan kaikki selaimella pelattavat Unityn pelit. Laajemmassa käytössä olevat, kuten kouluissa hyödynnettävät, tai toisen ohjelman sisäänrakennetut pelit vaativat liitännäisen

lisäksi ylimääräistä ohjelmointia toimivuuden takaamiseksi. Kouluissa suurin ongelma on rajatut ympäristöt ja asennusoikeudet, jotka estävät liitännäisten asentamisen.

Vertailun tuloksena voidaan todeta, että toimeksiantajalle ja ViLLEen sopiva alusta pelinkehitykselle on HTML5. Laaja pelimoottorien valikoima ja useat JavaScript-kirjastot edesauttavat hyvän ja optimaalisen pelinkehitysympäristön luomisen sekä ylläpitämisen.

3.3 Pelimoottorien vertailu ja pelimoottorin valinta

Pelimoottorilla (eng. game engine) tarkoitetaan videopelien ohjelmistokehystä, jonka päälle ohjelmoijat rakentavat pelejä. Pelimoottori on videopelien sydän ja sen tärkeyttä peleissä voi verrata auton moottorin tärkeyteen autolla ajamisessa. Se sisältää useimmat pelien tarvitsemat rajapinnat ja moduulit, kuten pelin piirtämisen 2D- tai 3D-grafiikalle, fysiikkamallinnuksen, syötteen hyödyntämisen, tekoälyn sekä audiojärjestelmän, jotta kehittäjät voivat keskittyä pelin ainutlaatuisuuden yksityiskohtiin.(UBM Tech, 2014.) ”Pelimoottorit tarjoavat visuaalisen ohjelmoinnin paketteja sekä uudelleen käytettäviä ohjelmiston elementtejä, joita tyypillisesti tarjotaan sisäänrakennetuissa alustoissa aktivoiden tehokkaan, tieto-ohjatun pelinkehityksen”.(Janalta Interactive Inc.) Pelimoottorin ominaisuudet ja käyttömahdollisuudet on dokumentoitu ohjelmointirajapinnan dokumenttiin (eng. API documents), jossa pelimoottorin kaikki funktiot selityksineen ovat listattuina. Usein pelimoottorien kehittäjänä toimii joko suurempi pelitalo tai pienempi, usein avoimen lähdekoodin, tiimi, jotka edesauttavat pelimoottorien jatkuvaa kehittämistä ja parantamista. Näin pelimoottorien versiot pysyvät useimmiten aina ajan tasalla, joka taas mahdollistaa tietotekniikan nopeassa ja vaihtelevassa virrassa mukana pysymisen.

Suunnitellessa oppimispeliä ja sen toteuttamista, yksi tärkeimmistä elementeistä on pelin toteutukseen sopivan pelimoottorin valitseminen. Pelin suunnitteluvaiheessa on hyvä vertailla eri pelimoottoreita ja käydä läpi niiden heikkoudet ja

vahvuudet. Monet pelimoottorit tarjoavat useita eri ominaisuuksia pelien ohjelmoimiseen, mikä tekee sopivan pelimoottorin löytämisen vaikeaksi. Pelimoottorin tärkeimpiä ominaisuuksia, joita sen useimmiten kuuluu tarjota, ovat erilaisten mallien tai kuvien lataaminen, näyttäminen ja animointi, törmäyksen tunnistus eri olioiden välillä, fysiikka, syötteen hyödyntäminen (eng. input), graafiset käyttöliittymäelementit sekä tekoäly. Nämä ominaisuudet luovat itse pelimoottorin, jonka pohjalta pelin kehittäjien on helpompi toteuttaa ainutlaatuinen kokonaisuus.

Tärkeimpiä ehtoja, joita tässä opinnäytetyössä hyödynnettävän pelimoottorin on täytettävä, ovat hyvä dokumentointi niin käyttömahdollisuuksista kuin rajapinnasta, riittävä ohjeistus ja esimerkit, käyttöönoton helppous ja julkaisemisen lisenssit, joiden tässä tapauksessa on oltava vapaasti käytettävät. Pelimoottorin toiminnan puolelta tärkeitä ominaisuuksia on ensisijaisesti selaintuki sekä fysiikkamoottori. Näitä ehtoja ja ominaisuuksia silmällä pitäen laadittiin listaus seitsemästä eri pelimoottorista, joita vertaillaan keskenään (liite 1). Osiot, joihin eri kappaleissa syvennytään, on merkitty eri värikoodeilla liitteeseen rakenteen hahmottamisen parantamiseksi

Dokumentaatio (vihreä)

Pelimoottorien dokumentaatio on yksi tärkeimmistä kohdista pelimoottorin valinnassa. Monen pelimoottorin dokumentointi vaihtelee suuresti ja niiden laajuus riippuu täysin pelimoottorin käyttötarkoituksista sekä tarjolla olevista ominaisuuksista. Dokumentoinneista sanotaan mm. että ”dokumentoinnin tarkoituksena ei ole listata kaikkia pelimoottorin funktioita vaan tutustuttaa käyttäjä pelimoottorin eri osa-alueisiin ja näin ollen auttaa käyttäjää pääsemään eteenpäin”.(Crafty 2013.) Harva pelimoottoreista pohjustaa dokumentoinnin pelkästään ohjelmointirajapinnan dokumenttiin, jossa kaikki funktiot ja niiden toiminta ovat listattuna, sen sijaan että pelimoottorin toimivuuksista ja käytettävyydestä olisi kerrottu vielä erillisessä dokumentissa. Esimerkiksi Pixi.js, Crafty ja CreateJS eivät sisällä erillistä dokumentaatiota pelimoottorin ominaisuuksista vaan rajoittuvat rajapinnan

dokumentointi kun taas Phaser, CanvasEngine, COCO S2D ja PlayCanvas sisältävät joko käyttäjälle suunnatun ohjekirjan pelimoottoriin ja sen hyödyntämiseen liittyen tai yhdistelmän sekä ohjekirjasta että rajapinnan dokumentaatiosta. Tärkeintä pelimoottorien kohdalla on että funktionaalinen toiminta on pystytty dokumentoimaan ja selittämään käyttäjälle niin että sen avulla voidaan helposti löytää ja hyödyntää pelimoottorin ominaisuuksia ja käyttää niitä ilman komplikaatioita. Käyttäjäystävällisyys ja riittävä dokumentointi takaa pelimoottorin käyttäjäkunnan. On hyvä muistaa että mitä laajempi dokumentointi on, sitä enemmän varsinaisesta pelimoottorista saa irti.

Aloitusoppaat ja esimerkit (keltainen)

Yksi pelimoottorien hyödyntämisessä ja käyttöönotossa huomioitava tärkeä asia on saatavilla olevat esimerkit ja käyttöoppaat eli tutorialit. Nämä liittyvät myös käyttäjäystävällisyyteen ja takaavat pelimoottorin paremman ymmärrettävyyden ja kokonaisvaltaisen hyödyntämisen. Vertailukohteiden kohdalla tärkein esimerkki ja opas on ehdottomasti niin sanottu ”Näin pääset alkuun” –ohje, jossa annetaan moottorin käyttöönotosta yksinkertainen esimerkki. Tämän avulla käyttäjä pääsee heti kiinni pelimoottorin toimivuuteen ja oppii kuinka esimerkiksi moottorin alustus-funktiota kutsutaan ja käytetään. Laajin esimerkkien käyttö ja hyödyntäminen näkyi Phaserillä, jonka esimerkki-sivustolta löytyy satoja eri toimintaa, ominaisuuksia ja käytettävyyttä kuvaavia esimerkkejä. Jokainen pelimoottori toimii eri tavalla, joten jokaisen pelimoottorin kohdalla on selvitettävä sen toimintaperiaate. Hyvänä esimerkkinä toimii PlayCanvas, joka hyödyntää myös videoesimerkkejä muiden esimerkkien ohella ja julkaisee niitä omilla sivuillaan. Tämän etuna on että käyttäjä näkee konkreettisesti miten funktioita tai ominaisuuksia käytetään helpottaen ainakin omalla kohdalla oppimista. Muiden vertailukohteiden sivuilta ei löydy suoraan ohjeistusvideoita, mutta monet käyttäjät ovat tehneet niitä itse.

Ominaisuudet (oranssi)

Pelimoottorien ominaisuuksien tarjonta on yleisesti hyvin laaja, sillä harvoin pelimoottoreita luodaan ja kehitetään vain yhtä pelityyppiä tai tarkoitusta varten. Vertailukohteiden tarjoamat ominaisuudet eivät poikkeaa toisistaan kovinkaan paljon, vaikka osa pelimoottoreista käyttää hyväkseen muita kirjastoja täydentämään omaansa. Pelimoottorien ominaisuuksia vertaillessa on hyvä muistaa omat tarpeet ja vaatimukset peliä varten. Jos kevyt pelimoottori täyttää käyttäjän tarpeet, ei kannata valita raskainta pelimoottoria, vaikka se olisi arvosteluissa paras. Pelimoottori on paras vain silloin, kun itse pystyy hyödyntämään kaikkia sen tarjoamia ominaisuuksia. Pelimoottorin ominaisuuksista yksi tärkeimmistä on selaintuki. Melkein kaikki vertailukohteet tukevat moderneja selaimia mikä käytännössä tarkoittaa, että Canvas ja WebGL toimivat piirrettäessä. Vertailukohteista ainoastaan PlayCanvas ja Phaser sisältävät integroidun fysiikkamoottorin ja näistä kahdesta Phaser hyödyntää jopa kolmea fysiikkamoottoria. Muut vertailukohteet eivät sisällä fysiikkamoottoria, mutta tukevat tämän päivän käytetyimpiä ulkopuolisia fysiikkamoottoreita, kuten Box2D, Chipmunk tai PhysicsJS, erinomaisesti. Graafisesti pelimoottorit on suunniteltu pääasiassa 2D-peleille, joskin PlayCanvas, COCO S2D ja CreateJS tukevat myös 3D-grafiikkaa. Sprite-grafiikka on laajasti tuettuna kaikilla pelimoottoreilla, joka mahdollistaa animoinnin ja se on laajasti käytössä lähes kaikissa 2D-peleissä. Opetuspelissä animointi tehdään pääasiassa Sprite-grafiikalla ja sen hyödyntämisellä voi tuoda nostalgisen vivahteen pelimaailmaan etenkin kun 80-luvun pikselimäinen peligrafiikka kasvattaa jälleen suosiota. Äänet ovat myös oleellinen osa pelimaailmaa ja suurin osa pelimoottoreista hyödyntää sisäänrakennettuja ääniominaisuuksia, kuten HTML5 audiota tai audiomoottoria. Muutamat vertailunkohteista käyttävät erillistä äänikirjastoa audion toistamiseen kuten esimerkiksi CreateJS ja sen oma audion kirjasto SoundJS sekä CanvasEngine ja sen tukema SoundManager 2 –kirjasto. Oppimispelissä äänimailmaa on hyvä hyödyntää sen luoman mielenkiinnon ja tunnelman vuoksi. Etenkin tunnelmien luonti musiikin avulla peliin saa pelaajan helpommin uppoutumaan ja keskittymään itse pelaamiseen.

Asennus ja käyttöönotto (sininen)

Hyvin moni pelimoottori on luotu avoimen lähdekoodin periaatteella. Käytännössä se tarkoittaa käyttäjän mahdollisuutta muokata ohjelman lähdekoodia omiin tarpeisiinsa ja käyttää sitä vapaasti omien tarkoitusten mukaan niin kopioimiseen kuin levitykseen.(COSS, 2014.) Avoimen lähdekoodin määrittelyssä myös lisenssit ovat tärkeässä osassa, siksi kaikki vertailukohteet toimivat MIT (Massachusetts Institute of Technology)-lisenssien alaisuudessa, joka tarkoittaa vapaata ohjelmistolisenssiä. Sen käyttäjällä on vapaat oikeudet muokata, kopioida ja käyttää teosta. MIT-lisenssi sallii myös teoksen käytön kaupallisissa suljetun lähdekoodin ohjelmistoissa.(Sofokus, 2014.) Kaikki vertailukohteet ovat avoimena lähdekoodina toteutettuja ja hyödyntävät GitHub-versionhallintaa, josta kuka tahansa saa hakea pelimoottorin lähdekoodit sekä viimeisimmän julkaistun version ja hyödyntää niitä omassa projektissaan. Näin käyttäjät pystyvät vaikuttamaan omaan pelimoottoriin ja edesauttamaan niiden kehitystä. GitHubia hyödynnetään laajasti nykypäivän avoimien lähdekoodien ohjelmistoissa luoden ohjelmien ympärille laajan kehittäjäkunnan, joka auttaa ohjelman parantamisessa sekä vikojen etsinnässä ja korjauksessa. Useampi pelimoottori tarjoaa myös suoraa linkkiä kotisivuilta tarvittavien tiedostojen lataamiseksi. Tämä helpottaa suunnattomasti asennusta ja pelimoottorin kirjaston käyttöönottoa, kun kirjaston viimeisintä versiota ei tarvitse etsiä erikseen. Varsinaisessa käyttöönotossa pelimoottorit hyödyntävät aloitusoppaita runsaasti. Monet vertailukohteista ohjeistavat lisäämään JavaScript -tiedostot omaan projektikirjastoon ja tarvittavan lähdeviittauksen html tiedostoon, jonka jälkeen pelimoottori on jo käyttövalmis. Vertailukohteista PlayCanvas ja COCO S2D sisältävät myös omat työkalut pelimoottorin käyttöönottamiseksi kuten PlayCanvas Designer tai CodeIDE ja CocoStudio, joiden avulla pelimoottorin hyödyntäminen ja pelin teko on helppoa. Käyttöönotossa ja asennuksessa avustavat myös laajat foorumi-yhteisöt. Vertailukohteista ainoastaan CanvasEnginellä ei ole virallisia foorumeita, mutta muut vertailukohteet ylläpitävät yhteisöjä, joista osa on hyvin aktiivisessa käytössä.

Pelimoottorin valinnassa käyttötarkoitukseltaan parhaaksi vaihtoehdoksi matemaattiselle oppimispelille valikoitui Photon Storm Ltd:n kehittämä Phaser.js,

jonka avulla HTML5- ja JavaScript -pelien luonti sekä työpöydälle että mobiililaitteille luonnistuu helposti ja vaivattomasti. Phaserin oma kirjasto on rakennettu Pixi.js kirjaston pohjalta, joka mahdollistaa laajan ominaisuuksien kirjaston sisältäen mm. kolme sisäänrakennettua fysiikkamoottoria, Pixi.js:n tuoman automaattisen renderöinnin sekä kolme erilaista animointi- kirjastoa.

Vaikuttavinta Phaserissä oli sen helppo käyttöönotto sekä laaja esimerkki-sivusto, jonka avulla ohjelmoija pääsee nopeasti alkuun. Phaseristä on myös julkaistu ohjekirjoja avaamaan kehittäjille Phaserin haastavimpia ominaisuuksia. Laajat ja aktiiviset foorumit tarjoavat paljon apua ja takaavat ettei pelin kehitys jää puolitiehen suuremmankin ongelman vastaan tullessa. Foorumeille lisättyihin kysymyksiin saatetaan vastata jopa samana päivänä, joka auttaa ohjelmoijaa pääsemään nopeasti eteenpäin. Lisenssinä Phaserillä on MIT, joka mahdollistaa pelin vapaan julkaisun, myynnin sekä markkinoinnin. Photon Storm haluaa myös edesauttaa pelien kehitystä tarjoamalla mahdollisuuden mainostaa Phaserillä tehtyjä pelejä sen omilla nettisivuilla. Tämä tuo huomattavasti lisää arvoa pelin kehitykselle.

3.4 Pelimekaniikan suunnittelu

Pelimekaniikalla tarkoitetaan yleisesti pelin sisäisiä sääntöjä, jotka määrittävät pelin toiminnan. Säännöt pitävät pelin sisäisen rakenteen muuttumattomana, vaikka pelin ulkoiset rakenteet, kuten teema tai tarina, saattaa muuttua. Tuhansia vuosia vanhan pelin mekaniikka voidaan helposti tuoda nykypäivään niin, että sen rakenne ja säännöt säilyvät kuten esimerkiksi Shakissa, jonka historia kantautuu aina 500jkr asti. Shakin tapauksessa pelin tarinalla tai pelimuodolla ei ole ollut merkitystä pelin sääntöjen kannalta. Säännöt määräävät edelleen pelin kulun ja sen, miten mikäkin nappula Shakkilaudalla reagoi pelikenttään ja muihin nappuloihin.(ShakkiNet, 2007.)

Kaikkien pelien säännöt jakavat tietynlaiset yleiset tunnuspiirteet, joita pelimekaniikat seuraavat. Näitä ovat mm. pelaajan rajoittaminen, täsmällisyys ja johdonmukaisuus, sääntöjen jakaminen kaikkien kesken, muokattavuus, sitoutuminen

sekä toistettavuus. Näitä sääntöjen periaatteita harvemmin noudatetaan sanatarkasti, mutta ne luetaan yhteisiksi piirteiksi pelisääntöjen muodollisesta näkökulmasta.(Salen & Zimmerman 2003, 118-125.)

Oppimispelin säännöt mukailevat näitä tunnuspiirteitä Kuplamatikka-pelin prosessikaaviossa (liite 2). Pelin tason alkaessa tulostetaan palikoita määrätty määrä pelikentälle, jonka jälkeen siirrytään tason varsinaisen silmukkaan, kuten liitteestä voidaan havaita. Silmukassa tehdään tarkistuksia esimerkiksi palikoiden määrästä pelikentällä sekä onko pelaajalle esitettyyn kysymykseen vastattu oikein vai väärin. Kun pelaaja on kerännyt tarpeeksi pisteitä tai pelikenttä on täyttynyt riittävästi palikoista, siirrytään silmukasta ulos. Säännöt rajoittavat pelaajan pelaamaan siis tason kerrallaan ja eteneminen on riippuvainen pelaajan suoriutumisesta. Epäonnistuessa tason suoriutumisesta, pelaajalle annetaan mahdollisuus yrittää tasoa uudestaan. Mikäli pelaaja on suoriutunut tasosta, pääsee hän jatkamaan seuraavalle tasolle. Näin pelille luodaan riittävästi toistettavuutta eikä pelaaja joudu tyytymään vain yhteen pelikertaan tai –kokemukseen.

Pelin ympärille voidaan luoda lähes minkälainen teema tai tarina tahansa, mutta tason prosessikaavion pelimekaniikkaa noudattamalla pelin virallinen identiteetti ei koskaan muutu. Tästä hyvänä esimerkkinä toimii suomalaisen pelitalon, Rovion, kehittämä Angry Birds, jonka pelisarjasta löytyy jo yli kymmenen erilaista peliä. Näissä peleissä varsinainen pelimekaniikka ei ole muuttunut mitenkään vaikka pelin teema on vaihtunut lähes puolen vuoden välein. Peli etenee silti samoja sääntöjä noudattaen – lintuja ammutaan porsaita päin keräten samalla mahdollisimman paljon pisteitä. Pelin säännöt eivät varsinaisesti tuo peliin lisäarvoa tai paranna pelaajan kokemusta, mutta ilman sääntöjä se ei voisi olla peli.

4 OPPIMISPELIN TOTEUTUS

Oppimispelin suunnitteluvaiheen jälkeen ryhdyttiin työstämään itse peliä. Tavoitteena oli luoda yksinkertainen peruslaskutoimituksia harjoittava oppimispeli, jossa pelikentälle laskeutuvien kuplien ja lukujen avulla ratkotaan erilaisia matemaattisia lausekkeita. Suunnitteluvaiheesta toteutukseen päätyneitä ominaisuuksia olivat mm. kahden kuplan yhdistäminen, haamukuplan luonti klikatusta kuplasta sekä kuplan liikuttaminen eli raahaa ja pudota-ominaisuus. Pelin ideaan ja ViLLEstä tuotaviin asetuksiin perehdytään myöhemmin tässä luvussa.

Tässä luvussa kerrotaan ohjelmistotuotannon menetelmistä ja niiden hyödyntämisestä Kuplamatikka-pelin sekä suunnitteluvaiheessa että toteutusvaiheessa. Lisäksi luvussa paneudutaan pelin tason mekaniikan toteutusvaiheen muutoksiin ja kuvaillaan pelin kehittämisessä kohdattuja ongelmia ja niiden ratkaisuja.

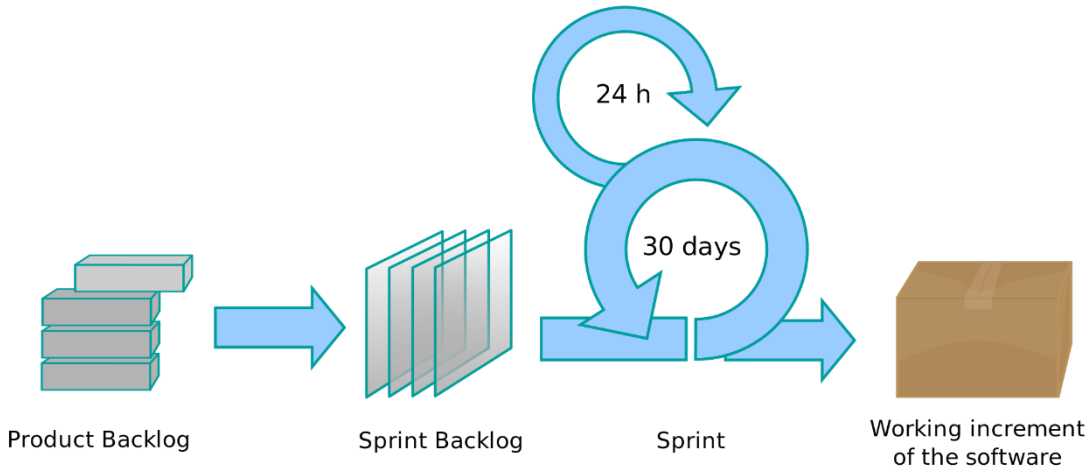
4.1 Ohjelmistotuotannon menetelmät ja niiden hyödyntäminen

Ohjelmistotuotannolla tarkoitetaan menetelmiä, joissa tuotetaan tai muutetaan tietokonepohjaisia ohjelmistoja. Se kattaa laajasti niin prosessinhallinnan kuin menetelmät, joita hyödynnetään tietokoneohjelmien valmistamisessa.(Haikala & Märijärvi 2004, 16.) Ohjelmistotuotannossa kehitys jaetaan perinteisesti eri osiin, mitä kutsutaan vaihejakomalliksi. Tätä mallia toteuttavat menetelmät voidaan jakaa yleisesti kahteen pääryhmään, suunnitelmaohjautuviin ja ketteriin ohjelmistokehityksen menetelmiin. Suunnitelmaohjautuvissa menetelmissä (eng. plan-driven methods, structural methods) pääpaino on tarkassa määrittelyssä ja suunnittelussa sekä dokumentoinnissa, jotka pyritään pitämään muuttumattomina alusta loppuun. Ketterissä menetelmissä (eng. agile methods) taas pääpaino on projektissa mukana olevien ihmisten välisessä kommunikaatiossa sekä sitä kautta tapahtuvassa tiedonhankinnassa. Ketterille menetelmille ominaista on edetä vaihe tai ominaisuus kerrallaan suuremmasta kokonaisuudesta, josta ei

välttämättä ole kovin tarkkaa kuvaa. Ketterissä menetelmissä muutokset pystytään hyväksymään helpommin kun taas suunnitelmaohjautuvissa menetelmissä muutoksia alkuperäiseen on vaikeaa tehdä.(Huttunen, 2006.)

Oppimispelin toteutuksessa parhaimmaksi ohjelmiston työstämistavaksi valikoituivat ketterät menetelmät sen helpon ja mukautuvan työstämistavan takia. Ketterässä menetelmässä keskeisimpiä periaatteita on muutosten hyväksyminen, tiivis yhteistyö, toimiva sovellus, suora keskustelu tai palautteen anto, luottamus, tekninen onnistuminen, yksinkertaisuus, tasainen työtahti sekä itsetarkastelu. Nämä periaatteet eivät kuitenkaan toteudu systemaattisesti kaikissa ketterissä menetelmissä vaan ne jakautuvat tilanteen mukaan, riippuen monesta eri tekijästä.(Highsmith ym. 2002, 15-18.)

Oppimispelin toteuttamiselle ei asetettu varsinaista aikataulua, joten se toteutettiin soveltaen ohjelmistoprojekteille ominaista ketterien menetelmien Scrum -mallia. Scrum on ohjelmistokehityksen prosessi pienikokoisille tiimeille, jonka periaatteena on tuottaa tehokkaasti tulosta lyhyessäkin ajassa (kuvio 3). Scrum -mallille yleistä on lyhyt suunnitteluvaihe, jossa projektille luodaan rakenne ja nimitetään siitä vastaava arkkitehti. Arkkitehti määrittelee projektin vision sen rakenteen mukaan, jota pyritään toteuttamaan läpi projektin kontrolloimalla ja kommunikoimalla kehittäjien kanssa. Projektin toteuttamisvaiheessa tiimit, työskentelevät lyhyissä kehitysjaksoissa, sprinteissä, luoden vähitellen valmiin tuotteen. Sprintit kestävät useimmiten kahdesta neljään viikkoa. Toteutusvaihetta seuraa päätösjakso, joka päättää tuotteen kehityksen. Tiimit ylläpitävät tilauskantaa (eng. backlog), jolla seurataan ja havainnollistetaan projektin eri tehtäviä sekä seurataan niiden edistymistä tiimin aktiivisuuden mukaan. Sprinttien aikana tehtävälistaa päivitetään ja tehtävät järjestetään uudelleen prioriteettien mukaan. Tiimin ulkopuolelta tuleviin muutoksiin ei tehtävälistauksessa oteta kantaa. Ihanteellisinta Scrum -menetelmän käyttö on silloin, kun projektin vaatimuksia ei voida etukäteen määritellä ja kaoottiset tilanteet on etukäteen ennakoitavissa projektin kehityskaareen nähden.(Rising & Janoff 2000, 30.)



Kuvio 3. Scrum -menetelmän havainnollistaminen.(Wikipedia, 2015.)

Oppimispelin toteutuksessa ketterän menetelmän periaatteista pystyttiin hyödyntämään sovellettuna kaikkia ja Scrumia toteutettiin yksinkertaistettuna versiona. Viikoittaisissa sprintin välisissä palavereissa, joita pidettiin sekä kasvotusten että puhelimitse, keskusteltiin lyhyesti viikon aikana suoritetuista tehtävistä sekä tulevista muutoksista. Tahti pysyi tasaisena alusta loppuun asti ja tiivis yhteistyö toimeksiantajan kanssa helpotti muutosten hyväksymistä sekä projektin teknistä toteutumista. Jatkuva työtulosten esittäminen ja toimeksiantajan positiivinen palaute työn edistymisestä paransi omalta osaltaan luottamusta toimeksiantajan ja kehittäjän välillä. Pelin yksinkertaisuus mahdollisti toimivan sovelluksen alusta alkaen helpottaen ominaisuuksien testausta ja projektin pitämistä kasassa. Loppua kohden projektista pystyttiin karsimaan selkeästi ylimääräiset ominaisuudet pois parantaen työn tulosta ja työtahtia entisestään luoden pelistä version, joka sittemmin olisi helppo sisällyttää osaksi ViLLEä.

4.2 Pelimekaniikan rakennemuutokset toteutusessa

Oppimispelin toteutusvaiheessa pelin rakennetta ja pelimekaniikkaa jouduttiin muuttamaan vaadittujen ominaisuuksien ja niiden tuomien haasteiden mukaan.

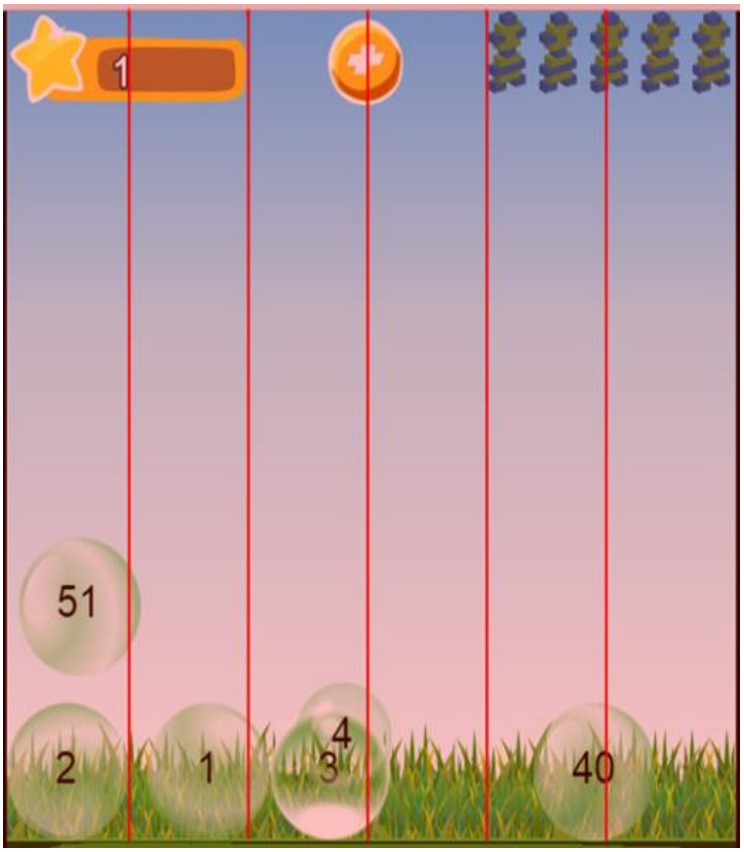
Alkuperäisessä prosessikaaviossa tason mekaniikka oli hyvin pelkistetty ja yksinkertainen. Pelin tarkoituksena oli päästä eteenpäin taso kerrallaan vaikeustason noustessa lineaarisesti. Toteutusvaiheessa muotoutunut prosessikaavio antaa tarkemman kuvan pelin kulusta ja sen eri elementeistä (Liite 3).

Liitteen prosessikaavion avulla havaitaan mm. ennen pelin alkua haettavat asetukset, jotka ovat joko opettajan ViLLEen syöttämiä tai oletusasetuksia. Asetusten saaminen ViLLEstä poikkeaa aikaisemmasta suunnitelmasta siten, että ylimääräistä ja erillistä asetus-sivua ei tarvinnut enää ohjelmoida. Valmiit asetukset helpottavat vähentäen prosessoitavan koodin määrää. Tämä nopeuttaa pelin itsensä toimivuutta ja muistin käyttöä sekä päästää pelin alkaessa pelaajan suoraan pelaamaan ilman erillisten asetusten säätämistä kohdilleen.

Pelin aloitustapa pystyttiin toteuttamaan alkuperäisen suunnitelman mukaan. Pelin alkaessa lähtölaskennan aikana pelikentälle pudotetaan ensimmäiset aloituskuplat sekä näytetään pelaajalle ensimmäinen matemaattinen tehtävä (kuva 2). Tämän jälkeen siirrytään pelisilmukkaan, jossa edetään useamman sen mukaan mitä pelaaja pelikentällä tekee

Kuva 2. Pelikenttä pelin alkaessa.

Pelisilmukassa tarkistetaan ensimmäisenä pelikentässä olemassa olevien kuplien määrä. Mikäli pelikenttä on ei ole vielä täysi, pudotetaan asetuksissa määrätyn aikavälein kentälle uusi kupla satunnaisesti kuuden eri sarakkeen väliltä. Sarakkeilla estetään kuplien päällekkäisyys niiden luontivaiheessa (kuva 3).

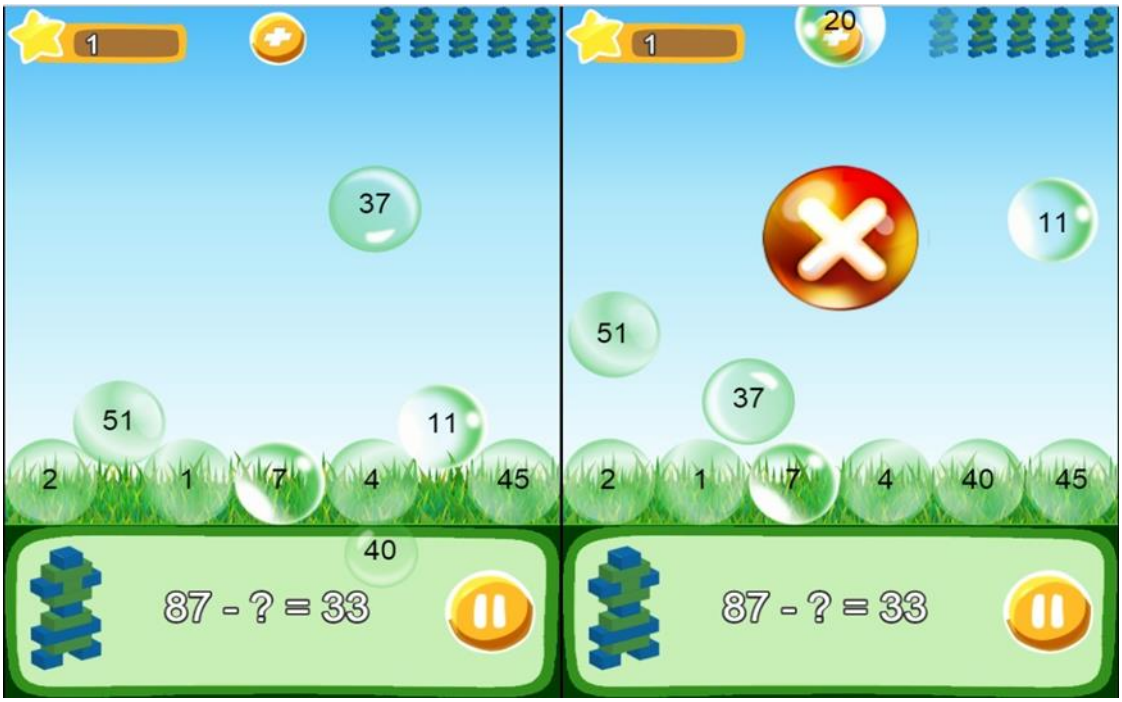


Kuva 3. Kuplien putoamissarakkeiden havainnollistaminen.

Pelikentällä pelaaja pystyy joko liikuttamaan tai jakamaan kuplia hiiren vasemmalla näppäimellä. Jakaessa peli tuo kentälle kaksi uutta kuplaa kun taas liikuttamalla kaksi kuplaa päällekkäin peli yhdistää kaksi kuplaa yhdeksi. Hiiren näppäimen pohjaan klikkaus luo pelikentälle liikutettavan haamukuplan, joka kuvastaa klikatun kuplan varjoa (kuva 4).

Kuva 4. Päällekkäiset kuplat yhdistyvät.

Siirrettäessä haamukuplaa tehtävälaatikon päälle peli yrittää ratkaista tehtävää kuplaan asetetulla luvulla. Mikäli ratkaisu on oikein, näytetään pelaajalle matemaattinen tehtävä kokonaisuudessaan oikealla vastauksella ja siirrytään seuraavaan tehtävään, jos pisteitä ei ole kerätty riittävästi. Jos taas ratkaisu oli väärin, vähennetään pelaajalta yksi elämä ja palataan pelisilmukassa pelikentän kuplien määrän tarkistukseen, jolloin silmukka alkaa jälleen alusta (kuva 5). Käytännössä koko pelisilmukan rakenne muuttui pelin toteutusvaiheessa. Pisteiden saaminen ja laskuun vastaaminen olivat suunnittelussa päätettyjä elementtejä, jotka jäivät prosessikaavioon muuttumattomina. Pelimekaniikkaan lisätyt kuplien liikuttaminen ja niiden jakaminen toivat lisää interaktiivisuutta ja haasteellisuutta alkuperäiseen toteutukseen. Laskut ja vastaukset kerätään ViLLEen, joka hoitaa lopullisen pisteytyksen ja tallentamisen. Tämä karsii pois esimerkiksi haaste-peleille tyypilliset Top-listaukset.



Kuva 5. Väärän haamukuplan siirto tehtävälaatikon päälle antaa väärän vastauksen.

Pelikentän ja tason lopetus tapahtuu, kun pelikenttä on täyttynyt kuplista, pelaaja on menettänyt kaikki elämänsä tai vaadittava pistemäärä on saavutettu (kuva 6). Aikaisemman prosessikaavion mukaan pelin loppuessa pelaajalle annettiin mahdollisuus hävitessä pelata taso uudelleen tai voittaessa jatkaa seuraavalle tasolle. ViLLEn takia mahdollisuus poistettiin sillä udelleen pelattavuus määritetään ViLLEn kautta. Samoin prosessikaaviossa kuvatut tason suoriutumistaulukon elementit tulevat ViLLEstä, joten ne pudotettiin kehitysvaiheessa pois.

Kuva 6. Havainnollistava kuva pelin häviämisestä ja voittamisesta.

Suurin syy muutosten syntymiseen oli lähtökohtaisesti projektin läpivientiin valittu ohjelmistokehityksen menetelmä. Scrum -mallin tavoin viikoittaisissa palavereissa projektin toteutukseen lisättiin ominaisuuksia sen mukaan miten projektin eteneminen antoi mahdollisuuksia pelin parantamiseksi. Perinteisen kaavan sijaan, jossa ohjelman ominaisuudet päätetään yleensä ennen projektin varsinaista aloittamista, ketterät menetelmät antavat mahdollisuuden ohjelman ominaisuuksien määrittelemiseen toteutusvaiheessa. Tällöin osa alkuperäisisitä määrityksistä saattaa muuttua tai pudota pois tarpeen mukaan. Tämä parantaa myös omalta osin projektin lopputulosta, kun hankalat tai tarpeettomat ominaisuudet voidaan tiputtaa pois tai vaihtaa yksinkertaisempiin ratkaisuihin.

4.3 Pelimekaniikan ongelmat ja ratkaisut

Oppimispelin tarkoituksena on auttaa alakouluikäisiä matematiikan harjoittelussa ViLLEn kautta. Peli-idean taustalla on Learning Yard Incorporationin kehittämä lapsille suunnattu MathCritters-peli, jossa ratkaistaan plus-, miinus-, kerto- ja ja-

kolaskuja viidessä matemaattisessa maailmassa tai rajoittamattomassa pelikentässä yksin tai ystävän kanssa. MathCritters:n peli-ideaa mukaillen ja hieman muuttaen aikaan saatiin Kuplamatikka. Pelissä lasketaan pelikentälle ilmestyvien kuplien avulla pelin alareunassa näkyvän laskulaatikon tehtäviä. Laskut voidaan ratkaista yhdistelemällä kuplien lukuja ja viemällä vastauskuplat laskulaatikon laskun päälle.

Suurimpia ongelmia tuotti pelissä, hyvän pelimoottorin löytämisen lisäksi, halutujen ominaisuuksien toteuttaminen. Näistä haastavimmat olivat kuplien luominen, niin sanottu raahaa ja pudota (eng. drag & drop) -ominaisuus sekä pelin jatkuva ominaisuuksien testaus ja asetusten mukauttaminen.

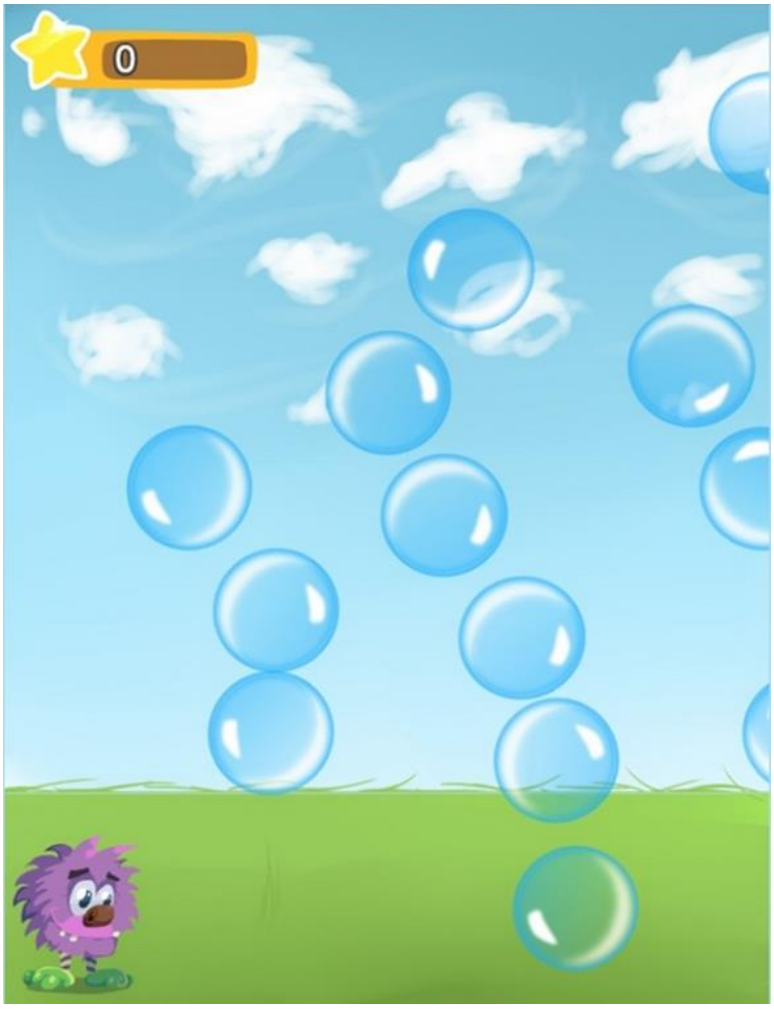
4.3.1 Kuplien luominen

Kuplien luonnin ja putoamisen pohjakaava syntyi HTML5 pelien ohjelmoijan, Andrzej Mazurin, blogin ”Getting Started With Phaser: Building ’Monster Wants Candy’” avulla ja blogi itsessään toimii hyvänä esimerkkinä pelien ohjelmoimisesta Phaserillä. Mazur käy blogissaan läpi yksityiskohtaisesti kuinka rakentaa yksinkertainen peli putoavilla objekteilla, hiiren klikkaus-ominaisuuksilla sekä pisteiden keräämisellä. Tätä logiikkaa soveltaen peli loi alussa yksittäisiä kuplamaisia objekteja, kuplia, pelikentälle pudottaen luonnollisesti ne ruudun ylälaidasta fysiikkamoottorin gravitaatiota hyödyntäen. Esimerkin mukaisesti kuplan sivuttainen putoamispiste määräytyi satunnaislogiikkaa hyödyntäen. Korkeus määräytyi kuplan koon mukaan, joka vähennettiin korkeuden nollapisteestä. Muita kuplalle määrättäviä ominaisuuksia oli mm. fysiikan asettaminen kuplan vartalolle, kuplan animointi, syötteiden hyväksyminen objektissa, kentän rajojen törmäyksen asettaminen sekä mahdolliset objektin tapahtumat erinäisissä tilanteissa (eng. object events) (kuva 7).

```
Bubble.item = {
    spawnBubble: function(game){
        // calculate drop position (from 0 to game width) on the x axis
        var dropPos = Math.floor(Math.random()*Bubble.GAME_WIDTH+100);
        // define the offset for every bubble
            //var dropOffset = [-27,-36,-36,-38,-48];
        // randomize bubble type
            //var bubbleType = Math.floor(Math.random()*5);
        // create new bubble: game.add.sprite(dropPos,dropOffset[bubbleType], 'bubble');
        var bubble = game.add.sprite(dropPos, -98, 'bubble');
        // add new animation frame
        bubble.animations.add('anim', [0], 10, true);
        // play the newly created animation
        bubble.animations.play('anim');
        // enable bubble body for physic engine
        game.physics.enable(bubble, Phaser.Physics.ARCADE);
        // enable bubble to be clicked/tapped
        bubble.inputEnabled = true;
        // add event listener to click/tap
        bubble.events.onInputDown.add(this.clickBubble, this);
        // add gravity to bubble
        bubble.body.gravity.y = 200;
        // set the anchor (for rotation, position etc) to the middle of the bubble
        bubble.anchor.setTo(0.5, 0.5);
        // set the random rotation value
        bubble.rotateMe = (Math.random()*4)-2;
        // add bubble to the group
        game._bubbleGroup.add(bubble);
    },
```

Kuva 7. Kuplan luonnin funktion ensimmäinen versio.

Kuplan luonti-funktion ensimmäisessä versiossa funktion sisäänotettavia parametrejä oli vain peli-objekti, jota käytettiin mm. sprite-grafiikan luonnissa sekä fysiikan asettamisessa objektille. Kuplat tottelivat pelikentän rajojen fysiikkaa osittain, mutta kuplien yhteentörmäyksen logiikka ei ollut vielä toimiva. Tästä syystä, kuplien kasautuessa, ne työnsivät toisensa ulos pelikentän rajoista (kuva 8).



Kuva 8. Fysiikan toiminta pelin kehityksen alkuvaiheessa.

Phaserin toimintaa vähitellen sisäistäen kuplien logiikka alkoi muotoutua paremmaksi ja esimerkiksi kuplien törmäys (eng. collision) kehittyi. Pelikentän fyysisten rajojen muotoutuessa kuplien fyysiset rajat saatiin näkymään ja toimimaan yhdessä pelikentän kanssa. Kuplan vartalolle asetettiin mm. ympyrän muotoinen törmäyskehä, jonka avulla kuplat mukailevat realistisesti yhteentörmäystä niin pelikentän rajojen kuin muiden kuplien kanssa (kuva 9).

Kuva 9. Kuplien törmäyskehää havainnollistava kuva. Kehä on 50% kuplan koosta.

Peliin luotavien kuplien logiikka muotoutui viimeiseensä vasta pelin toteutuksen loppuvaiheessa, jolloin kuplien logiikkaa ei tarvinnut enää muuttaa vastaamaan pelin muun logiikan tarpeita. P2.js fysiikkamottoria hyödyntäen kuplien liikkuminen pystytiin rajoittamaan pelikentän sisäpuolelle. Tässä tapauksessa pelikentän pituus ja leveys olivat molemmat 600, vaikka peli kokonaisuudessaan oli asetettu kokoon 600x800. Pelikentän yläpuolelle jätettiin näkymätön alue, jonne kuplat ilmestyvät. Näin kuplille ei tarvinnut tehdä erillistä animaatiota vaan niiden ilmestyminen saatiin piiloitettua pelaajalta.

Objektien putoamislogiikkaa luodessa tärkeintä on ottaa huomioon sekä objektin fyysiset ominaisuudet että korrelointi niin pelikentän kuin muiden pelissä käytettävien objektien kanssa. Samoin kuplien toimintatapaa on hyvä verrata pelikentän fyysisiin rajoihin ja siihen mitä tapahtuu kun esimerkiksi pelikenttä täyttyy kuplista ja fyysiset objektit alkavat puskea toisiaan eri suuntiin (kuva 10).

Kuva 10. Fyysisten objektien toimintaa havainnollistava kuva pelikentän täyttyessä.

Kun fyysiset ominaisuudet on määritelty ja pohdittu etukäteen, objektin ohjelmoinnin toteuttaminen on huomattavasti helpompaa. Pelin eri objekteille ja elementeille on hyvä miettiä etukäteen erilaisia sääntöjä, joiden mukaan ne toimivat

pelikentällä, kuten luotavien objektien määrä tai niiden tapahtumat eri toimintojen aikana. Sääntöjen määrittely edesauttaa myös eri objektien ja pelikentän suhteiden hahmottamisessa, jolloin ohjelmoitaessa on helpompaa löytää ja soveltaa funktioita omien tarpeiden mukaan.

4.3.2 Drag & Drop

Raahaa ja pudota –ominaisuus oli yksi haastavimmista toteutettavista ominaisuuksista pelin kehityksessä. Alkuperäisen suunnitelman mukaan kuplat yhdistettiin siirtämällä kaksi kuplaa päällekkäin. Kuplan siirtämisessä oleellisinta oli huomioida hiiren toiminnot sekä fyysisen objektin suhde muihin objekteihin ja niiden törmäyksen laskeminen. Suurin apu oikeanlaisen funktion luomiseen löytyi Phaserin omista esimerkeistä, jossa mm. fysiikkamoottorille löytyi esimerkkejä törmäyksen havainnollistamisesta aina fyysisen objektin liikuttamiseen. Näitä hyödyntämällä onnistuttiin luomaan funktio, joka ratkaisi sekä objektin liikuttamisen että tarkisti kahden objektin päällekkäisyyden yhdistämisvaiheessa.

Ensimmäinen askel raahaa ja pudota –ominaisuuden toimimisessa oli fyysisen objektin liikkeelle saaminen. Pelin alussa hiirelle annettiin kolme funktionaalista toimintoa; klikkaus, irrotus ja liikkuminen. Nämä funktiot toteutuivat aina kun hiirtä klikattiin, liikuteltiin tai päästettiin irti. Klikkauksen funktio käytti hyväksi yksinkertaisuudessaan hiiren ja objektin vartaloita vertaillen niiden sijainteja sekä yhdistäen hiiren väliaikaisesti objektin vartaloon. Tällä tavoin liikkumisen funktion avulla pystyttiin muuttamaan fyysisen objektin paikkaa pelikentällä ja klikkauksen lopussa irrottamaan hiiri fyysisestä objektista palauttaen objektin pelikentälle.

Toisessa vaiheessa ominaisuus yhdistettiin yhden funktion alle, joka hyödynsi vain kupla-objektia. Tässä vaiheessa ominaisuus oli suunniteltu uudelleen luomaan klikkauksen yhteydessä alkuperäisestä kuplasta haamun, jota liikutettiin ympäri kenttää todellisen kuplan sijaan. Haamukuplasta tehtiin sprite-objekti ilman fyysistä vartaloa, joka näinollen voisi mennä päällekkäin muiden fyysisten objektien kanssa. Objektien päällekkäisyys tarkastettiin siten raahauksen loputtua ja kuplat yhdistettiin, mikäli ne olivat päällekkäin. Kun päällekkäisyys toteutui,

haamukupla ja alkuperäinen kupla tuhottiin funktion avulla jättäen jäljelle vain yhdistetyn kuplan. Jos haamukuplasta päästettiin irti ja objektien päällekkäisyys ei toteutunut fyysinen objekti palautettiin alkuperäisiin koordinaatteihin. Irti päästettäessä haamukupla tuhottiin ja alkuperäinen kupla palautettiin samoihin koordinaatteihin (kuva 11).

Kuva 11. Liikuteltavan haamukuplan havainnollistaminen.

Viimeisessä vaiheessa raahaa ja pudota –ominaisuuden kehitystä tarkastusten määrää lisättiin ja kuplien jakaminen kahdeksi lisättiin uutena ominaisuutena. Kuplien jakamisen myötä tarkistuksia oli paranneltava ongelmien välttymiseksi. Päällekkäisten objektien tarkistukseen lisättiin vastauksen tarkistaminen, mikäli

raahattu kupla pudotetaan vastauslaatikon päälle. Tarkistuksista tärkein on päällekkäisten objektien etäisyyksien laskenta, jotta kuplien yhdistäminen on mahdollisimman täsmällinen. Tähän tarkoitukseen käytettiin Phaserin omaa funktiota lähimmän kuplan löytämiseksi. Tätä hyödyntämällä saadaan tarkka arvo kahden objektin välille ja näinollen yhdistettyä haamukupla ja sitä lähinnä oleva kupla.

Oleellisin asia raahaa ja pudota –ominaisuuden kannalta on sen toteuttaminen mahdollisimman yksinkertaisella tavalla, vaikka tarkistuksia joutuisikin funktion sisälle luomaan useamman. Ominaisuudelle tärkeimmät toiminnot ovat aloitus- ja lopetusfunktio, joiden avulla pystytään kontrolloimaan niin raahauksen alkua kuin loppua saaden objektin fyysinen vartalo tottelemaan sen hetkistä osoitinta.

4.3.3 Pelin asetukset ja niiden integrointi

Pelin testauksen yhteydessä tärkeäksi elementiksi nousi myös asetusten huomioiminen pelinkehityksessä. Peliin tuotavat erilliset asetukset lisäsivät omalta osaltaan testauksen merkitystä sillä ViLLEen sisällytettynä peliin annetaan asetukset erillisenä objektina. Jotta kaikki vaadittavat asetukset saataisiin toimimaan, oli pelille luotava oletusasetukset. Niitä käytetään silloin, kun peliä ei pelata suoraan ViLLEn kautta.

Oletusasetuksia suunnitellessa kerättiin ylös tiedot ja elementit, jotka käyttäjä asettaa ViLLEstä ennen pelin alkua. Näitä ovat kuplien yhdistämisen operaattori, lista lausekkeista, pelaajan elämien määrä, lausekkeen piilotettua osaa indikoiva luku, kuplien putoamisen aika sekä negatiivisten lukujen sallittavuus. Oletusasetusten kokonaisuus tuodaan yhtenä objektina peli-funktioon, jossa sitä pilkotaan aina tarpeen mukaan, kuten lausekkeen näyttämisessä tai kuplien sisäisten lukujen luomisessa.

4.4 Pelin testaaminen

Ohjelmiston testaus on yksi tärkeimmistä prosesseista ohjelmiston kehityksessä. Sen tarkoituksena on taata, että ohjelmisto ja sen koodi on vakaa sekä estää

koodia tekemästä mitään mitä sen ei ole tarkoitus tehdä. Ohjelman tulisi olla ennalta arvattava ja johdonmukainen, yllättämättä käyttäjäänsä.(Mayers ym. 2011, 2.) Pelinkehityksessä testauksen tärkeys korostuu entistäkin enemmän sillä useimmiten peliä kehitettäessä monimutkainen rakenne ja koodi rikkovat väistämättä pelin ominaisuuksia. Pelin testauksen määrä onkin siten lähes suoraan verrannollinen pelin menestymiseen.(Boespflug & Schultz 2005, 39-40.)

Perinteisessä pelinkehityksessä ohjelmoijat eivät täysin testaa omia pelejään. Useimmiten kehittäjillä ei ole aikaa testauksen suorittamiseen vaan pääpaino pysyy ominaisuuksien ja kokonaisuuden ohjelmoimisessa ja testauksen tekee siihen erikseen nimetty työryhmä tai henkilöt.(Boespflug & Schultz 2005, 151.) Kuplamatikan kohdalla oli tähän periaatteeseen tehtävä poikkeus resurssien ja aikarajan takia. Ohjelmointivirheiden (eng. bug) etsiminen sekä ohjelmoinnin ohella suoritettava testaus (eng. debug) ovat olennainen osa ohjelmointiprosessia. Tästä tavasta ei poikettu kuplamatikan kohdalla, mutta erillistä valmiin kokonaisuuden testausta ei missään vaiheessa tehty. Samoin versionhallinnan ylläpito oli helpompi toteuttaa jatkuvan testauksen myötä ja toimivat ominaisuudet työnnettiin versionhallinnan säilytyspaikkaan. Tässä projektissa se oli GitHub-nettipalvelimella. Virheen ilmentyessä työnalla olevan ominaisuuden kohdalla, oli versionhallinnasta helppo tutkia koodimuutoksia ja ottaa ylös mikä toiminto tai koodin osa sai pelin rikkoutumaan.

Testausmenetelmistä Kuplamatikassa toteutettiin pelitestausta (eng. play testing). Se poikkeaa täysin kaikista perinteisistä pelikehityksen testauksen tavoista. Useimmat testausmenetelmät pyrkivät vastaamaan kysymykseen; toimiiko peli kun taas pelitestaus pyrkii vastaamaan kysymykseen toimiiko peli hyvin. Pelitestauksen kysymykseen haetaan perinteistä poiketen merkitykseltään enemmän arvostelevaa kantaa kuin faktoihin perustuvaa vastausta. Tästä syystä sen on myös testauksen muodoista vaikein.(Boespflug & Schultz 2005, 296.)

Testauksessa parhaana työkaluna toimi tietenkin itse selain, jossa peliä loppujen lopuksi pelataan. Nykypäivän moderneissa selaimissa pystytään helposti näkemään sivun lähdekoodi ja useimmiten selaimissa on sisäänrakennettu testaus-

toiminto. Kuplamatikan testauksessa käytettiin Googlen tarjoamaa Chrome-selainta, jonka GCWI:n (Google Chrome Web Inspector) avulla pystytään tarkastelemaan mm. sivustolla toimivia koodeja, kuten JavaScript, jolla Kuplamatikka on kirjoitettu. Tätä testauksen sisäistä toimintaa kutsutaan viralliselta nimeltään ”White box”-testaamiseksi. Se antaa testaajalle mahdollisuuden käyttää lähdekoodia eri tavalla kuin loppukäyttäjät, jotka normaalisti eivät edes pääse käsiksi ohjelman lähdekoodiin. Tämä tapa on normaalille testaajalle hyvin vaikea lähestymistapa testauksessa, mutta sitä käytetään usein juuri tilanteissa, jossa testauksen tekee ohjelmoija itse.(Boespflug & Schultz 2005, 153.)

5 POHDINTA JA YHTEENVETO

Pelinkehitys tai ohjelmistokehitys on usein monimutkainen prosessi, johon sisältyy monta, useimmiten pitkää ja haastavaa, vaihetta. Tämän opinnäytetyön tavoitteena oli tehdä ViLLE-oppimisympäristöön alakouluikäisten peruslaskutaitoja harjoittava ja tukeva oppimispeli, Kuplamatikka. Projektin lopputuloksena aikaan saatiin toimeksiantajaa miellyttävä oppimispeli, joka sisällytetään ja otetaan käyttöön ViLLEssä kevään 2015 aikana.

Kuplamatikka toteutui suunnitellussa aikamääreessä 2014 loppusyksystä. Annettu aikaraja pystyttiin jakamaan hyvin sekä suunnitteluvaiheen että toteutusvaiheen kesken ja ketterien menetelmien Scrum-tyyliä hyödyntäen tuloksia saatiin nopeasti. Suuremmilta kompastuskiviltä pystyttiin välttymään tiiviillä yhteistyöllä toimeksiantajan kanssa ja hyvän suunnitteluprosessin myötä, jotka molemmat kasvattivat luottamusta projektin toteutumisesta.

Kuplamatikka-projektin hyviä puolia oli sen tuoma kokemus pelinkehityksestä ja ohjelmointikielestä. Suunnitteluvaihe toteutui projektin kannalta hyvin jättäen tilaa ominaisuuksien muokattavuudelle työstämisvaiheessakin. Peliä suunnitellessa oleellisinta oli löytää selaimelle ja toteutukselle sopiva pelimoottori, joka vastasi pelille asettamia tarpeita, sekä pelimekaniikan hiominen toimivaksi kokonaisuudeksi.

Toteutusvaiheessa oleellisinta oli yksinkertaisuus sekä projektille asetettuihin tavoitteisiin pääseminen. Projektin ominaisuuksien ohjelmointi ja aikaisempi kokemattomuus käytetystä pelimoottorista toivat sopivaa haasteellisuutta. Vielä toteutusvaiheessakin pohdittiin pelin eri ominaisuuksista, olivatko ne oleellisia pelin tavoitteiden saavuttamiseksi. Esimerkiksi ominaisuudet, kuten kuplien klikkaus tai raahaaminen, jouduttiin miettimään uudelleen. Aikarajalliselle projektille ominaisuudet on hyvä suunnitella niin, että niiden ohjelmointi on nopeaa ja vaivatonta. Kokemattomuus taas lisäsi haasteellisuutta ongelmatilanteissa, mutta laaja vastauksien etsiminen eri lähteistä auttoi etenemisessä. Ratkaisujen löytämisessä on kuitenkin varottava liiallisen ajan käyttöä. Ohjelmoinnin tärkeäksi osaksi

osoittautui myös jatkuvan testauksen ylläpito. Vaikka kyseessä olisi suurempi pelinkehitys-projekti, testauksen ylläpitäminen edesauttaa kokonaisuuden hallitsemisessa ja virheettömämpään lopputulokseen.

Itselleni koko pelinkehitysprosessista ja -projektista jäi miellyttävä kokemus. Phaserin avulla sain runsaasti kokemusta ohjelmoinnista ja JavaScript-ympäristö tuli hyvin tutuksi. Uskon ongelmanratkaisukykyni kehittyneen matkan varrella. Tulevaisuutta ajatellen oppimispelien kehityksessä on hyvä panostaa suunnitteluvaiheeseen ja miettiä aina pelin kokonaisuutta. Tulevissa pelinkehitysprojekteissa tulen mitä todennäköisemmin hyödyntämään oppimaani tietoa JavaScriptistä sekä käyttämään vastaisuudessakin Phaseriä pelimoottorina.

LÄHTEET

Ashcraft, M. H. 2002. Math anxiety: Personal, educational, and cognitive consequences. Current Directions in Psychological Science, Vol. 11, No. 5, 181-185 Viitattu 09.02.2015 http://www.thinkingahead.com.au/Documents/math_anxiety-consequences.pdf

Alvarez, J.; Rampnoux, O.; Jessel, J. P. & Methel, G. 2007. Serious Game: Just a question of posture. Artificial & Ambient Intelligence, AISB, 7, 420-423. Viitattu 04.02.2015 http://www.researchgate.net/publication/260517048_Serious_Game_just_a_question_of_posture

Boespflug, K. & Schultz, C. P. 2005. Game Testing All in One. USA: Course Technology.

COSS 2014. Avoin lähdekoodi. Viitattu 23.10.2014 www.coss.fi > Tietoa avoimuudesta > Avoin Lähdekoodi.

Crafty 2011-2013. Documentation for Crafty.js. Viitattu 23.10.2014 www.craftyjs.com > Documentation.

Crawford, C. 1984. The art of computer game design. Viitattu 03.02.2015 http://www.vic20.vaxxine.com/wiki/images/9/96/Art_of_Game_Design.pdf

Develop. 2014. Develop Live: Unity vs HTML5. Viitattu 20.01.2015 http://www.develop-online.net/news/develop-live-unity-vs-html5/0198301

Developer Economics. 2014. Top Game Development Tools: Pros and Cons. Viitattu 20.01.2015 http://www.developereconomics.com/top-game-development-tools-pros-cons/

E-Games. 2007. Game Development Process. Viitattu 20.01.2015 http://www.e-games.tech.purdue.edu/GameDevProcess.asp

Egenfeldt-Nielsen, S.; Meyer, B. & Sørensen, B. H. 2011. Serious Games in Education: A Global Perspective. Tanska: Aarhus University Press

Haikala, I. & Märijärvi, J. 2004. Ohjelmistotuotanto. Helsinki: Talentum.

Highsmith, J.; Cockburn, A.; Paulk, M. C.; Manzo, J.; McMahon, P. E.; Bowers, P. & Sleve, G. 2002. Agile Software Development. Crosstalk, Vol.15, No. 10. Viitattu 23.01.2015 http://agilesweden.com/doc/oct02.pdf

Huttunen, J. 2006. Ketterän ohjelmistokehitysmenetelmän määrittely, vertailu ja käyttäjä-kysely. Diplomityö. Sähkö- ja tietoliikennetekniikan osasto, Viestintätekniikka. Helsinki: Helsingin Teknillinen Korkeakoulu. Viitattu 26.01.2015 http://lib.tkk.fi/Dipl/2007/urn007665.pdf

Janalta Interactive Inc. 2010-2014. Engine. Viitattu 30.10.2014 www.techopedia.com > Dictionary > Tags > Software

Kurvinen, E. 2014. Pelinomainen sähköinen matematiikan opetus ja solmukohtien automaattinen tunnistaminen alkuopetuksessa. Opinnäytetyö. Informaatioteknologian laitos, Tietojenkäsittelytieteet. Turku: Turun Yliopisto. Viitattu 19.1.2015

Laakso, M.-J. 2010. Promoting Programming Learning. Engagement, Automatic Assessment with Immediate Feedback in Visualizations. TUCS Dissertations no 131

Lukimat. 2014a. Ekapeli-Matikka. Viitattu 16.1.2015 http://www.lukimat.fi/ > Matematiikka > Materiaalit > Tietokoneohjelmat > Ekapeli-Matikka

Lukimat. 2014b. Vuonna 2014 Ekapelillä keskimäärin 6000 päivittäistä pelaajaa. Viitattu 19.1.2015 http://www.lukimat.fi/ > Uutiset

Mannila, B.; Hämäläinen, R. & Oksanen, K. 2007. Pelaa ja opi: räätälöityjä verkkopelejä ammatilliseen oppimiseen. Jyväskylän yliopisto, Koulutuksen tutkimuslaitos. Viitattu 01.02.2015 https://jyx.jyu.fi/dspace/bitstream/handle/123456789/37477/978-951-39-3191-9.pdf?sequence=1#page=70

Mayer, B. 2013. Game-based Learning. Viitattu 30.01.2015 http://css-kti.tugraz.at/research/cssarchive/courses/TeLearn/SS05/Presentations/Game-Based_Learning.pdf

Michael, D. R. & Chen, S. L. 2005. Serious Games: Games That Educate, Train, and Inform. USA: Course Technology.

Mitchell, A. & Savill-Smith, C. 2004. The use of computer and video games for learning. A review of the literature. UK: LSDA. Viitattu 30.01.2015 http://simventure.co.uk/sites/default/files/Gaming_Research.pdf

Mitchell, B. L. 2012. Game Design Essentials. USA: Sybex.

Prensky, M. 2001. Digital Game-Based Learning. USA: McGraw-Hill.

Prensky, M. 2005. Computer games and learning: Digital game-based learning. Handbook of computer game studies, 18, 97-122 Viitattu 30.01.2015 http://www.itu.dk/people/jrbe/DMOK/Artikler/Computer%20games%20and%20learning%202006.pdf

Rising, L. & Janoff, N. S. 2000. The Scrum Software Development Process for Small Teams. IEEE SOFTWARE, Vol. 17, No. 4, 26-32. Viitattu 23.01.2015 http://faculty.salisbury.edu/~xswang/Research/Papers/SERelated/scrum/s4026.pdf

Salen, K. & Zimmerman, E. 2004. Rules of Play: Game Design Fundamentals. USA: The MIT Press

ShakkiNet. 2007.Shakkipelin säännöt. Viitattu 21.01.2015 http://www.shakki.net/ > Shakkipeli > Säännöt

Sofokus 2000-2014. Avoin lähdekoodi – Tiedätkö mitä rajoituksia se asettaa ohjelmistosi käytölle?. Viitattu 23.10.2014 www.sofokus.com > Blogi > Avoin lähdekoodi

UBM Tech 2014. Game Career Guide – What is a Game Engine? Viitattu 30.10.2014 www.gamecareerguide.com > Features > What is a Game Engine?

Unity. 2015. What Is Unity? Viitattu 20.01.2015 http://unity3d.com/pages/what-is-unity

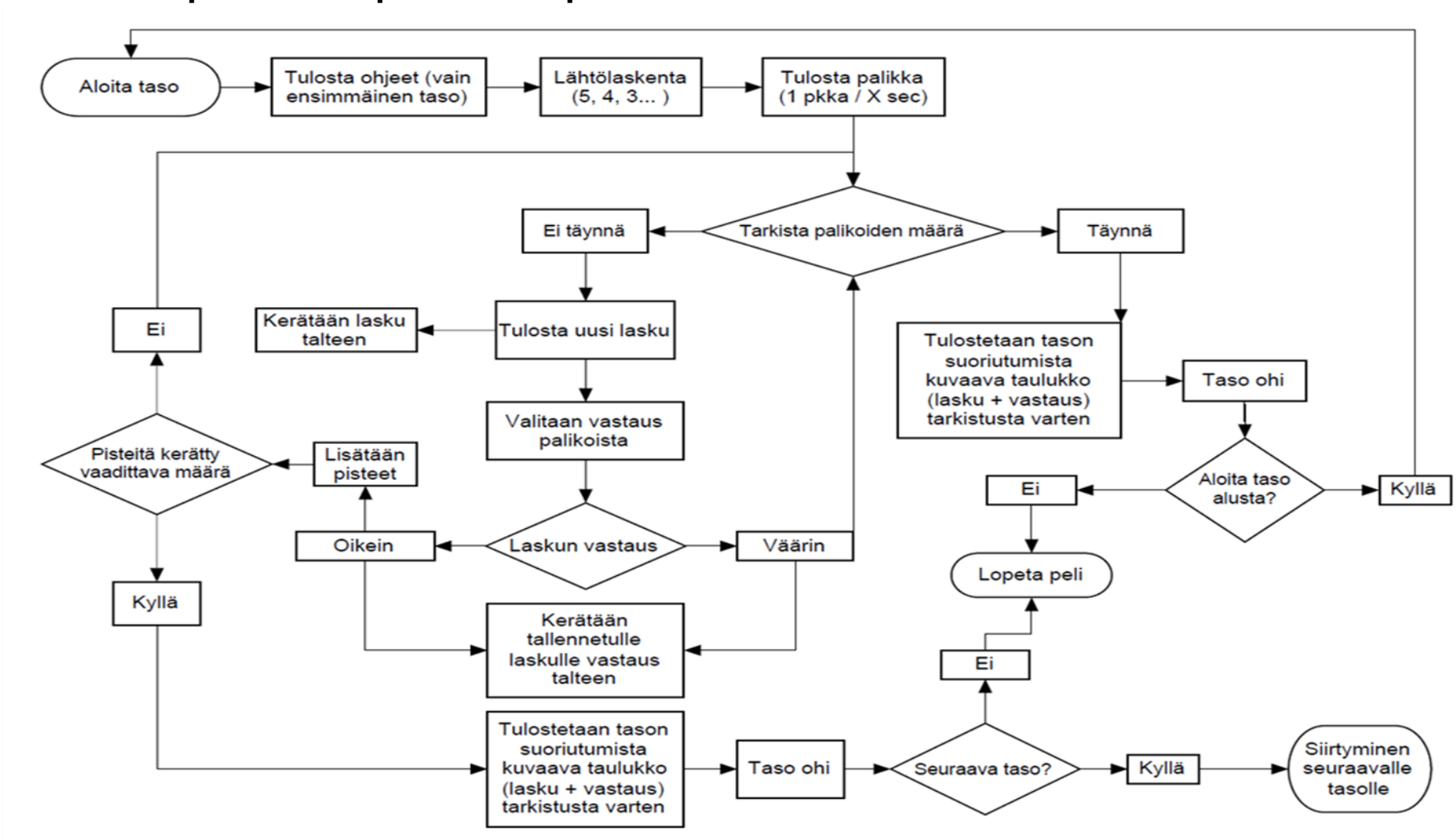
ViLLE. 2014. Usein kysyttyjä kysymyksiä – Yleistä ViLLEstä. Viitattu 19.1.2015. https://ville.cs.utu.fi/faq/faq_fi.html#yleista1

Wikipedia. 2015. Scrum (software development). Viitattu 08.02.2015 http://en.wikipedia.org/wiki/Scrum_(software_development)

HTML5/ JavaScript-pelimoottorien vertailu-taulukko

	Crafty	Phaser	Pixi.js	PlayCanvas	CreateJS/EaselJS	CanvasEngine	COCOS2d-js
Dokumentointi	API	Online doc	API	Käyttöopas ja API	API	Online doc ja API	Online doc ja API
Kattavavuus	Funktiot listattuna	Ei tarkaa funktioiden määrittelyä	Funktiot listattuna	Ominaisuuksien määrittelyjä, API:ssa funktiot listattuna	Funktiot listattuna	Ominaisuuksien määrittelyjä, API funktiot listattuna	Ominaisuuksien määrittelyjä, API funktiot listattuna
Aloitusoppaita	Nettisivut	GitHub	Nettisivut	Youtube-kanava ja nettisivut	Nettisivut	Nettisivut	Nettisivut
Esimerkkejä	GitHub	GitHub ja nettisivut	GitHub ja nettisivut	Youtube-kanava ja nettisivut	GitHub ja nettisivut	GitHub ja nettisivut	GitHub ja nettisivut
Ominaisuudet							
Selaintuki	kaikki modernit selaimet, jopa IE 9	Modernit selaimet, jotka tukevat Canvasia ja WebGL	Modernit selaimet, jotka tukevat Canvasia ja WebGL	Modernit selaimet, jotka tukevat Canvasia ja WebGL	Modernit selaimet, jotka tukevat Canvasia ja WebGL	Modernit selaimet, jotka tukevat Canvasia ja WebGL	kaikki modernit selaimet
HTML5/Canvas/WebGL	Canavas tai DOM	WebGL/Canvas	WebGL/Canvas	WebGL/Canvas	Canvas	Canvas	HTML5
Kattavavuus	Useita eri ominaisuuksia	Useita eri ominaisuuksia, käyttää mm. Pixi.js integroituna	Useita eri ominaisuuksia	Useita eri ominaisuuksia	Useita eri ominaisuuksia	Useita eri ominaisuuksia	Useita eri ominaisuuksia
Fysiikka	Ei sisällä omaa fysiikkamoottoria, mutta tukee fysiikkamoottoreita, kuten Box2D	Integroituna kolme fysiikkamoottoria: Arcade Physics, Ninja Physics & p2.js	Täysi tuki PhysicsJS kanssa.	Sisäänrakennettu fysiikkamoottori	Ei sisäänrakennettua fysiikkamoottoria. Tukee tiedettävästi ainakin Box2D:tä	Ei sisäänrakennettua fysiikkamoottoria.	Ei sisäänrakennettua fysiikkamoottoria, Box2D ja Chipmunk tuki
Alusta	Soveltuu niin työpöytäversiolle kuin mobiililaitteellekin	Suunniteltu niin työpöytäversiolle kuin mobiililaitteellekin (android, IOS)	Tukee kaikkia alustoja (mobiili, työpöytä..)	Suunniteltu niin työpöytäversiolle kuin mobiililaitteellekin (android, IOS)	Selaimia tukevat alustat (käytettävyys vaihtelee alustasta riippuen)	Selaimia tukevat alustat (käytettävyys vaihtelee alustasta riippuen)	Suunniteltu niin työpöytäversiolle kuin mobiililaitteellekin (android, IOS)
2D/3D	2D	2D	2D	Sekä 2D että 3D	Sekä 2D että 3D	2D	Sekä 2D että 3D
Sprite	Sisäänrakennettu SpriteMap tuki	Sisäänrakennettu Sprite-tuki	SpriteSheet -tuki	Rendering Sprites lisäosa sprite.js:n avulla	SpriteSheet -tuki (SpriteSheetUtils, SpriteSheetBuilder)	SpriteSheet -tuki	
Animaatio	Tween ja spriteAnimation	Sprite Sheets, Texture Packer ja Flash	Texture Packer ja Atlas formaatit	Sisäänrakennettu käyttöönotto animaatiodatan avulla.	CreateJS oma TweenJS-kirjasto	Sprite animaatio	Spine editor ja Dragon Bones - flash editor
Audio/Sound	Sisäänrakennettu äänio minaisuus. HTML5 audio, cross-browseria varten MP3, Ogg ja Wav	Web audio ja legacy HTML audio	Ei tue ääntä itsessään. Käytettävä äänikirjastoja (Howler.js, SoundJS..)	Tukee kaikkia ääniformaatteja joita selaimet tukevat. Sisäänrakenenttu Audiosource komponentti	CreateJS oma SoundJS-kirjasto	HTML5 audio api tai SoundManager 2	Sisäänrakennettu audioengine
Asennettavuus	Suora latauslinkki nettisivuilta, GitHubin repository	GitHubin repository	GitHubin repository	PlayCanvasi Designer, GitHubin repository	Suora latauslinkki nettisivuilta, GitHubin repository, CDN:n kautta hostattuna	Suora latauslinkki nettisivuilta, GitHubin repository	Suora latauslinkki nettisivuilta, CodeIDE, COCO Studio
Käyttöönotto	Tarvittavat js tiedostot ja DOM:n lisäys.	Lokaali web serveri, editori ja Phaserin uusin versio	Tarvittavat js tiedostot ja DOM:n lisäys.	PlayCanvasi Designer tai tarvittavat js tiedostot ja DOM:n lisäys.	Tarvittavat js tiedostot ja DOM:n lisäys.	Tarvittavat js tiedostot ja DOM:n lisäys.	CodeIDE:n helppo integrointi, CocoStudio tai tarvittavat js tiedostot ja DOM:n lisäys.
Foorumi	Julkiset foorumit	Julkiset foorumit	Julkiset foorumit	Julkiset foorumit	Julkiset foorumit	ei virallisia foorumeita	Julkiset foorumit
Lisenssit	MIT tai GPL lisenssit	MIT lisenssit	MIT lisenssit	MIT lisenssit	MIT lisenssit	MIT lisenssit	MIT lisenssit

Kuplamatikka-pelin tason prosessikaavio suunnittelusvaiheessa



Kuplamatikka-pelin tason prosessikaavio toteutusvaiheessa