Ohjelmoinnin opetuspeli yläkouluikäisille

Oulun yliopisto
Tietojenkäsittelytieteiden laitos
Pro gradu -tutkielma
Harju Japi Aatu Armas
7.5.2015

Tiivistelmä

Tässä työssä tutkittiin, millaisella tietokonepelillä voitaisiin opettaa ohjelmoinnin peruskäsitteitä yläkouluikäisille nuorille. Kysymykseen vastattiin konstruktiivisella tutkimusotteella, jonka konstruktiona toteutettiin ohjelmointia opettava tietokonepeli, Koodilinna. Sen suunnittelu perustui kirjallisuuskatsauksessa löydettyihin viihteellisten ja opetuksellisten pelien suunnitteluperiaatteisiin sekä ohjelmoinnin opettamisen ja oppimisen periaatteisiin. Nämä periaatteet vastasivat osaltaan tutkimuskysymykseen siitä, millainen ohjelmointia opettava peli tulisi olla.

Koodilinnan opettavaisuutta, mieluisuutta, ja vaikutuksia ohjelmoinnin mielikuviin arvioitiin kvalitatiivisesti Oulun Pateniemen koulun kahdeksannen luokan oppilailla. Yhteensä kahdeksan oppilasta osallistui tutkimukseen, jossa he saivat pelata peliä. Ennen ja jälkeen pelaamista suoritettiin osallistujille kysely, jolla tiedusteltiin taustatietoja, mielipidettä ohjelmoinnista ja pelatusta pelistä, ja ohjelmoinnin peruskäsitteiden osaamista. Saatuja tuloksia verrattiin keskenään ja eroavaisuuksien avulla pystyttiin tarkentamaan vastausta kysymykseen pelin opettavaisuudesta.

Koodilinnan arviointi osoitti, että se opetti ohjelmoinnin peruskäsitteitä, mutta ei kyennyt tarjoamaan syvällistä ymmärrystä niistä. Tämän lisäksi havaittiin sen olevan mieluisa ja hyödyllisen oloinen osallistujille ohjelmoinnin opettelussa, ja nähtiin myönteisiä muutoksia jo ennalta olleessa myönteisessä kuvassa ohjelmointia kohtaan. Lopullinen vastaus tutkimuskysymykseen saatiin esitettyä ne Koodilinnan kehitykseen otettujen näkökulmien kautta, jotka olivat pelien motivaatiotekijät, viihteellisyys ja opettavaisuus, pelin yleinen suunnittelu, ja ohjelmoinnin pedagogia.

Tutkimus on relevantti siitä syystä, että ohjelmointia tullaan opettamaan peruskouluissa, joten on tärkeää tutkia erilaisia opetuskeinoja. Lisäksi Koodilinnan kaltaista erityisen interaktiivista, rikkaan pelimaailman ja pelimekaniikan sisältävää peliä ei ole aiemmin tehty, vaikka näitä osia sisältäviä pelejä on tehty. Tämä tutkimus osoittaa, että hyvinkin viihteellinen peli voi olla opettavainen, joten tämä antaa jatkotutkimukselle aihetta esimerkiksi opetuspelien viitekehyksen kehittämisen kannalta. Myös Koodilinnan kehitystä on mahdollista jatkaa tutkimuksessa saatujen parannusehdotusten mukaisesti.

Asiasanat
ohjelmointi, tietokonepelit, oppimispelit, oppiminen, nuoret, lapset

Alkusanat

Olen aina ollut kiinnostunut tietokoneista ja erityisesti tietokonepelit ja ohjelmointi ovat olleet minulle mieluisia aktiviteetteja jo pidemmän aikaa. Tutkielmani aiheeseen sain alustavaa inspiraatiota Projekti II –kurssilta, mutta tämänkin jälkeen aihetta mietiessäni kävin liudan mitä erilaisimpia ideoita läpi. Lopulta päädyin ajatukseen, että yhdistäisin intohimojani ja tutkisin aihetta, jossa voisin opettaa ohjelmointia pelin avulla, ja vieläpä käyttää sen toteuttamiseen ohjelmointitaitojani ja intoani peleihin. Asetelma tuntui täydelliseltä, mutta urakka ei olisi mitenkään edennyt niin hyvin ilman ystävääni, jonka seura ja moninainen tuki mahdollistivat jaksamiseni työn parissa. Kiitos sinulle näistä asioista.

Oulussa 7.5.2015

Aatu Harju

Sisällys

1. Johdanto

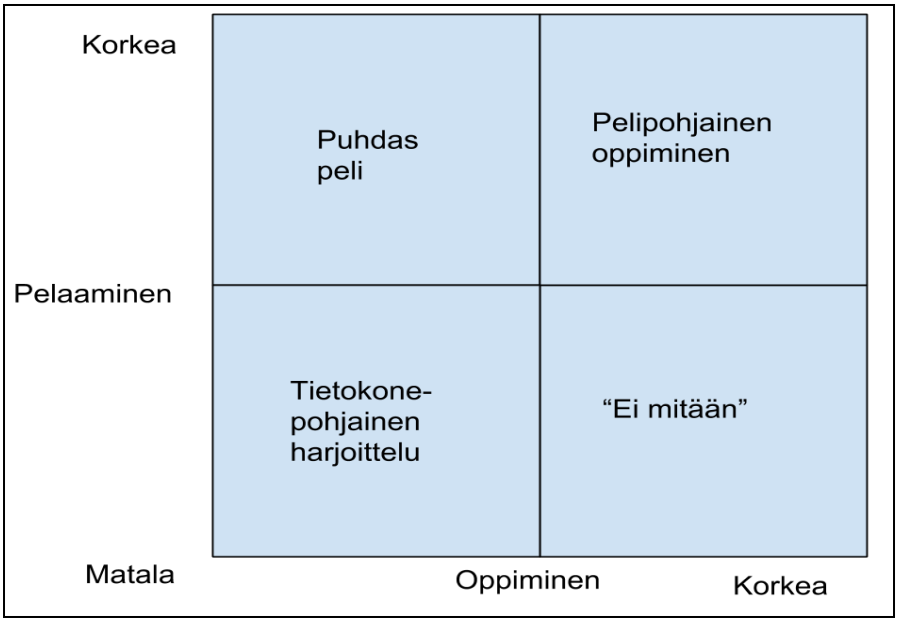
Sen lisäksi, että tietokonepelit ovat olleet kaupallisesti menestyineitä, myös opettajat ja ohjaajat ovat alkaneet huomioimaan ne (Garris, Ahlers ja Driskell, 2002). Prensky (2005) pohtii syytä siihen, miksi käyttää pelejä oikean maailman asioiden oppimiseen. Hänen mukaansa nuorten kasvaminen on muuttunut radikaalilla tavalla siihen suuntaan, että nykyinen sukupolvi käytännössä kasvaa tietokoneen äärellä. Hän myös lisää, että tällaisia uudenlaisia oppijoita tulee myös motivoida uusilla tavoilla, sillä heidän tapansa käsitellä tietoa on erilainen kuin heidän vanhemmillaan tai millään edeltäneellä sukupolvella. (Prensky, 2005.)

Oppilaan on vaikea oppia ratkaisemalla sellaisia ongelmia, joita hän ei koe itselleen merkityksellisiksi. Eräs ratkaisu motivaation puutteeseen on saada oppilas tekemään löydöksiä opeteltavasta asiasta ja tätä kautta motivoimaan itse itseään. (Simon, 2000.) Uudet teknologiat voivat rohkaista oppilaita ongelmanratkaisuun ja tämä pätee etenkin tietokonepeleihin siitä syystä, että pelaaminen on hyvin puoleensavetävä aktiviteetti. Tämän vuoksi opettajille ja muille koulutusalan ammattilaisille voisi olla suuri hyöty pelien yhdistämisestä opetustavoitteisiin. (Garris ja muut, 2002.)

Gee (2008) sanoo, että kouluissa opetus, jossa asiat tuodaan oppilaille suoraan esille sellaisenaan, on vallalla. Geen (2008) mukaan pelit, joissa oppiminen tapahtuu sen sijaan epäsuoraan jonkin toisen asian kautta, ovat kuitenkin vielä toissijainen vaihtoehto. Prenskyn (2005) mukaan pelit eivät myöskään vielä nykyään kykene korvaamaan koko opetusohjelmaa, vaan ovat osa laajempaa kokonaisuutta, vaikka ne ovat hiljalleen ottamassa yhä enemmän jalansijaa opetuksessa.

Pelit ja pelaaminen liittyvät olennaisesti ihmisen luonteeseen. Muiden eläinten lailla olemme pelanneet ja oppineet pelaamastamme jo esi-isiemme ajoista lähtien. (Crawford, 1984.) Pelit voidaan määritellä niin, että ne ovat tarkoin muusta maailmasta eristettyjä järjestelyjä, joita pelataan tietyssä ajassa ja paikassa, ja joita rajoittavat niille annetut säännöt. Pelaaminen itsessään voidaan määritellä niin, että se on aina vapaaehtoinen ja mieluisa aktiviteetti. (Caillois, 1961.) Crookall, Oxford ja Saunders (1987) erottelevat simulaation peleistä sanomalla, että pelit eivät simulaatioiden tavoin pyri mukailemaan todellisuutta mahdollisimman tarkasti ja niissä on omat sääntönsä ja strategiansa. Toisaalta Garris ja muut (2002) sanovat simulaation voivan sisältää pelillisiä elementtejä, kuten pisteenlaskun, joten näiden kahden ero ei välttämättä ole kovin selvä. He myös lisäävät, että jos pelit peilaavat liian todentuntuisesti oikeaa maailmaa, ne eivät ole pelaajalle enää peli.

Pelejä ja yleensä opetusohjelmia voidaan luokitella myös niiden opetuksellisuuden ja pelimäisyyden mukaan. Kuva 1 havainnollistaa Prenskyn (2005) kuvaamaa tietokonepohjaisten ohjelmistojen luokittelua opettavaisuuden ja pelimäisyyden näkökulmasta. Hänen mukaansa perinteinen tietokonepohjainen harjoittelu vastaa tilannetta, jossa pelaamisen elementti on vähissä ja samalla oppiminenkin. Tällaiset ohjelmat ovat Prenskyn (2005) mukaan käytännössä toistoharjoituksia. ”Ei mitään”-osio on mallin mukaan olematon tarjonnaltaan, sillä tietokoneohjelma, joka opettaa tehokkaasti ilman pelillistä sisältöä on käytännössä vain haave. Voidaan kuitenkin kyseenalaistaa kuinka ajankohtainen kyseinen tieto on, sillä 10 vuoden aikana on osioon voinut ilmestyä esimerkiksi pelillisyydestä vapaita sovelluksia. ”Puhtaiksi peleiksi” luokitellaan sellaiset pelit, joissa on vahva pelimäinen elementti, mutta ei opetuksellista sisältöä. Pelipohjainen oppiminen eroaa puhtaista peleistä tältä osaa, sillä niihin on tietoisesti upotettu opettavaisia elementtejä. (Prensky, 2005.)



Kuva 1. Opetusohjelmien luokittelu niiden pelimäisyyden ja oppimisen tehokkuuden kautta (Prensky, 2005).

Opetus ja kulttuuriministeriö on lisännyt ohjelmoinnin opettamisen peruskoulun opetussuunnitelmaan (Opetus- ja kulttuuriministeriö, 2014). Tämä johtaa ohjelmoinnin opetukseen tähtäävien sovellusten ja menetelmien kehittämispaineeseen. Pelit voisivat olla eräs keino tuoda ohjelmointi esille oppilaita stimuloivalla ja motivoivalla tavalla. Ohjelmointi voidaan määritellä toimintana, jossa kirjoitetaan tai valmistellaan tietokoneohjelmaa (Oxford English Dictionary, 2015). Tähän mennessä on toteutettu useita erityisiä helpotettuja ohjelmoinnin oppimista edistäviä ohjelmointikieliä- ja ympäristöjä, joista mahdollisesti mainittavimpia suosionsa puolesta on Scratch (Resnick ja muut, 2009). Useimpien ympäristöjen käyttäminen keskittyy enimmäkseen koululuokkamaiseen ympäristöön, mutta pelit voisivat olla myös sen ulkopuolella oleva keino ohjelmoinnin osaamisen edistämiseen.

Tämän tutkielman motivaatio lähti Oulun yliopiston tietojenkäsittelytieteiden laitoksen Projekti II -kurssilta, jonka aikana toteutettiin kenttien muokkausohjelma yläkoululaisten seikkailupeliä varten. Tällöin kiinnostus yleistä pelien opettavaisuutta kohtaan nousi ja henkilökohtaisen pohdinnan kautta tässä työssä päädyttiin tutkimaan ohjelmointipelin toteuttamista yläkoululaisille. Lisäksi vaikka ohjelmointipelejä on kehitetty lukuisia, on niiden ilmeinen käyttötarkoitus usein luokkahuoneessa, tai ne eivät hyödynnä täysipainotteisesti potentiaalia, jota peleillä voidaan ilmaista esimerkiksi pelimaailman, juonen tai pelimekaniikan suhteen. Tämän vuoksi haluttiin tutkia mikäli tätä potentiaalia voitaisiin vapauttaa opetuspeleissä. Tämän tutkimuksen aikana kehitetty peli nähdään sellaisena, joka sopii näiden aukkojen paikkaajaksi, ja erottautuu muista peleistä myös olemalla aidosti viihteellinen peli, joka on pohjimmiltaan opettavainen.

Tutkielmassa kysytty tutkimuskysymys muodostui seuraavanlaiseksi: Millaisella pelillä kannattaa opettaa yläkouluikäisille ohjelmoinnin peruskäsitteitä? Kysymykseen haettiin vastausta konstruktiivisella tutkimusotteella, jonka kautta kehitettiin ohjelmointia opettava Koodilinnaksi nimetty tietokonepeli. Koodilinnan kehittämiseen vaaditut suunnitteluperiaatteet kerättiin aiemmasta tutkimuksesta ja niiden avulla pystyttiin osittain vastaamaan tutkimuskysymykseen. Suunnitteluperiaatteita kerätessä otettiin huomioon sekä viihteellisyyden että opettavaisuuden näkökulmat. Myös pelin

ohjelmoinnin opetussisällön suunnitteluperiaatteet saatiin koostettua tarkastelemalla ohjelmoinnin oppimista ja opettamista käsittelevää aiempaa tutkimusta.

Lopulta siihen, kuinka hyvin Koodilinna opetti ohjelmointia, saatiin tietoa arvioimalla peliä suunnittelutieteellisesti konstruktiivisen tutkimusotteen (Hevner, March, Park ja Ram, 2004) mukaisesti. Arviointi suoritettiin kvalitatiivisesti tekemällä tutkimuskäynti Oulun Pateniemen koululle. Tutkimukseen osallistui kahdeksan kahdeksasluokkalaista oppilasta vastaamaan ohjelmointia ja pelaamista tiedusteleviin alku- ja jälkikyselyihin, joiden välissä pelattiin kehitettyä peliä. Aineiston analyysivaiheessa tarkasteltiin eroja kyselyiden vastauksissa ja pyrittiin saamaan tarkempi vastaus pelin varsinaisesta opettavaisuudesta.

Tutkimusaihe rajattiin niin, että Koodilinna suunniteltiin yläkouluikäisiä silmällä pitäen, jolloin päästiin keskittymään heidän tarpeisiinsa paremmin pelisuunnittelun osalta. Opetettavista ohjelmointikäsitteistä yritettiin rajata sellaiset, jotka olisivat luonteeltaan perustavaa laatua ohjelmoinnin oppimisen kannalta.

Tämä tutkielma koostuu pelejä käsittelevän aiemman kirjallisuuden, sekä ohjelmoinnin oppimista ja opettamista käsittelevän kirjallisuuden katsauksesta. Kirjallisuuslukujen jälkeen perustellaan tutkimusmenetelmän valinta ja esitellään konstruktiivinen tutkimusote ja sen arviointivaiheessa käytetty kvalitatiivinen tutkimusmenetelmä. Lisäksi kuvaillaan konstruktion toteutusprosessi, mitä seuraa lopuksi Koodilinnan kuvaus. Lopulta käydään läpi konstruktion arvioinnin tulokset ja keskustellaan tuloksista vastaten samalla tutkimuskysymykseen.

2. Pelien suunnittelu

Tässä luvussa käydään läpi olennaisia seikkoja, mitä tulee pelien suunnitteluun niiden viihteellisyyden ja opettavaisuuden kannalta. Tämä luku on jaettu seuraavasti: pelit ja motivaatio, pelien avulla oppiminen ja pelien suunnittelu. Pelien ja motivaation alaluvussa käsitellään motivaatiota ja sen merkitystä peleissä ja oppimisessa. Alaluvussa esitellään Kellerin (1987) ARCS-malli, joka sisältää erinäisiä strategioita motivaation parantamiseksi oppimisympäristössä ja konkreettisia keinoja saavuttaa nämä strategiat. Lopulta käydään läpi flow-teoria, joka kuvaa tietyn tehtävän suorittamisen aikana tapahtuvaa motivoitunutta tilaa, josta voi olla hyötyä oppimisesssa. Toisessa alaluvussa käy läpi oppimisen ja pelien yhteyttä toisiinsa, mitä tarkastellaan oppimisympäristöjen ominaisuuksien pohjalta ja kuinka saada nämä ominaisuudet läsnä peleihin. Lisäksi tehdään katsaus oppimiseen kokemuksien pohjalta pelien avulla ja siihen, kuinka oppimisaktiviteetin reflektointi voi parantaa oppimista. Kolmas alaluku esittelee Garrisin ja muiden (2002) kuvailemat yleiset pelien ulottuvuudet, joihin kuuluvat fantasia, säännöt ja tavoitteet, sensoriset virikkeet, haasteet, mysteeri, ja hallinta. Näihin ulottuvuuksiin liitetään myös tapoja saavuttaa ne peleissä.

Aineisto tätä teoriakatsausta varten hankittiin hakemalla aihealuetta koskevia keskeisiä artikkeleita muodostamalla systemaattisesti aiheeseen liittyviä hakusanoja ja hakemalla niiden avulla niistä asiasisällöltään oleellisia julkaisuja. Oleellisuutta punnittiin myös viittausten lukumäärän perusteella, ja aiemmin löydettyjen julkaisujen avulla haettiin muita julkaisuja lähdeviittauksia seuraamalla, tai niihin viittaavia artikkeleita hakemalla.

2.1 Pelit ja motivaatio

Motivaatio voidaan määritellä haluna tehdä tiettyä aktiviteettia ja motivoituneena ihminen haluaa uppoutua, nähdä vaivaa ja olla kauemmin tekemisissä asian kanssa (Garris ja muut, 2002). Motivaatio on tärkeä osa oppimista, sillä oppijan on luultavasti nähtävä vaivaa, jotta oppimista syntyisi. Myöskään perinteiset tavat motivoida oppilaita eivät toimi nykyään yhtä tehokkaasti kuin aiemmin. (Prensky, 2005.) Eräs syy tähän on Prenskyn (2005) mukaan rinnakkaisen tiedon tulva nykymaailmassa ja tästä juontuva nykynuorten rinnakkainen tiedon käsittely lineaarisen sijaan. Tämä ei taas vastaa luokkahuoneessa kohdattua ympäristöä, jossa tiedonlähde on useimmiten rajattu ainoastaan yhteen, eli useimmiten opettajaan.

Ulkoiset motivaattorit eivät välttämättä ole niin vahvoja kuin sisäiset, mutta kummallakin on osansa oppijan käytöksen kannalta (Garris ja muut, 2002). Itsemotivoituneet oppijat saavat sisäisen motivaationsa aktiviteetin mukavuudesta ja nautinnollisuudesta, kun taas ulkoinen motivaatio tulee siitä näkemyksestä, että aktiviteetti on tärkeää oppimisen kannalta (Deci, Vallerand, Pelletier ja Ryan, 1991).

Ihmiset ovat kiinnostuneita tietynlaisista peleistä omien motivationaalisten seikkojensa perusteella, mutta yksittäisen pelin valintaa sen sijaan sanelevat mielihyvää ohjaavat asiat, kuten pelin grafiikat, pelattavuus ja sensoriset ärsykkeet. Hyvällä pelattavuudella tarkoitetaan sopivaa tasapainoa pelissä etenemeisessä ja sen asettamassa kognitiivisessa kuormassa. Sensorisissa ärsykkeissä on kyse eri aisteja stimuloivista tekijöistä, kuten äänistä. Nämä ärsykkeet tukevat pelaajan kokemaa fantasian tunnetta pelissä, vaikkakaan eivät saisi olla keskeisessä asemassa sen luomisessa. (Crawford, 1984.)

Keller (1987) kuvailee motivaation elementtejä selittävän ARCS-mallin, jota käyttäen voidaan auttaa ymmärtämään motivaatiota opetuksessa. Tämä malli tunnistaa neljä

olennaista strategiaa motivoivan opetuksen saavuttamiseksi, jotka ovat huomiostrategia, relevanssistrategia, itsevarmuusstrategia ja tyytyväisyysstrategia. Huomiostrategiaa käyttämällä pyritään nostamaan ja pitämään yllä uteliaisuutta ja mielenkiintoa opetettavan asian sisältöä kohtaan. Tähän liittyy kolme osa-aluetta, jotka ovat oppijoiden mielenkiinnon saavuttaminen, johon voidaan päästä mm. huumorin avulla tai yllättävien tiedonpalasten kautta. Toinen osa huomiostrategiaa on uteliaisuuden stimuloiminen, joka voidaan saada aikaan luomalla ongelmatilanteita ja kyselemällä aiheeseen liittyviä kysymyksiä. Kolmas osa on huomion vaihtelevuus eri opetustapoja käyttämällä. (Keller, 1987.)

Toinen ARCS-mallin strategioista liittyy relevanssiin, joka on voimakas tekijä siinä, mitä asioita olemme kiinnostuneita oppimaan. Kellerin (1987) mukaan tarkkailemme tiedostamattamme oppiessamme sitä, kuinka opetettava aihe liittyy elämäämme ja jos löydämme merkityksellisen yhteyden näiden asioiden välillä, olemme motivoituneita oppimaan lisää. Relevanssissa on yleisesti kyse tietyn asian näkemisenä käyttökelpoisena elämämme halujen ja määränpäiden saavuttamisessa. Toinen osa strategiaa on oppijan motiivien saattaminen yhteen opeteltavien asioiden kanssa. Peleillä voidaan yhdistää samanaikaisesti esimerkiksi kilpailun- ja yhteistyöhaluisten henkilöiden motiiveja oppimiselle. Tuttuus on kolmas tekijä relevanssistrategiassa. Ihmiset ovat valmiimpia oppimaan asioista, joihin he jo etukäteen uskovat. Eräiksi tavoiksi tuttuuden tunteen saavuttamiseksi mainitaan opetusmateriaalin teksteissä nimien käyttäminen persoonapronominien käyttämisen sijaan sekä konkreettisten, tutuista asioista otettujen esimerkkien hyödyntäminen. (Keller, 1987.)

Itsevarmuusstrategia on kolmas Kellerin (1987) ARCS-mallin strategioista ja siihen kuuluvia onnistumisen halun ja epäonnistumisen pelon tunteita aliarvioidaan usein, sillä ihmiset voivat näyttää tuntemuksensa eri tilanteissa todellista neutraalimmin. Tämän vuoksi on tärkeää tarjota onnistumisen tunne mahdollisimman pian työpajojen tms. alussa ja antaa motivaatiota jatkaa annetun asian opettelua. Jatkamisen halu pätee hänen mukaansa vain jos haastetta on jatkossakin tarpeeksi, sillä liian vaikeat tehtävät saavat aikaan turhautumisen tunteen. Oppilaille tiedottaminen heiltä odotetuista asioista on eräs yksinkertaisimmista tavoista rakentaa itsevarmuutta. Toinen itsevarmuusstrategian osa on antaa oppilaiden onnistua, eli antaa tilaisuuksia siihen. Aloittelijoille on syytä antaa vuolaasti matalan tason haastetta runsaan palautteen kera, kun taas edistyneemmät tarvitsevat enemmän haastetta ja kilpailullisia elementtejä. Joka tapauksessa opetuksessa on liikuttava tarpeeksi nopeasti tylsistymisen välttämiseksi, vaikka liian nopea tahti voi aiheuttaa ahdistusta oppilaissa. Kolmas osa strategiaa on tarjota hallinnan tunnetta oppilaalle opetustilanteissa. Virheitä korjaavan palautteen antaminen jatkuvasti opetustilanteessa auttaa opeteltavassa asiassan orientoitumisessa ja tekee virheiden tekemisestä hyväksyttävämpää oppilaalle itselleen. (Keller, 1987.)

Kellerin (1987) ARCS-mallin viimeinen strategia on tyytyväisyyden kasvattaminen, joka voidaan saavuttaa tarjoamalla tilaisuuksia hyödyntää vasta opittua tietoa merkityksellisen tuntuisissa tehtävissä. Mikäli opeteltava asia on niin pitkä, ettei sitä voida heti hyödyntää käytännön tehtävissä, voidaan tarjota myönteistä tukea ja ulkoisia motivaattoreita, kuten fyysisiä palkintoja. Ihmiset vertailevat omaa suoritustaan muiden suorituksiin ja voivat olla tyytymättömiä itseensä, mikäli he eivät suoriudu yhtä hyvin kuin muut. Tätä lievittääkseen voidaan oppimisjakson alkuodotukset yrittää yhdenmukaistaa sen todellisten tulemien kanssa. (Keller, 1987.)

Motivaatioon liittyy myös flow-tila, joka esiintyy henkilön ollessa täysin uppoutuneena johonkin tehtävään. Tällöin henkilön ajan ja paikan taju on haihtunut, eikä hän sillä hetkellä välitä mistään muusta, vaan on täysin keskittynyt tehtäväänsä (Nakamura ja

Csikszentmihalyi, 2002). Flow'ta voidaan kuvailla optimaalisen suorituskyvyn tilaksi, jossa henkilön taidot kohtaavat täydellisellä tavalla kohdatut haasteet ja tällöin hän tuntee nauttivansa ja olevansa hallussa tilanteesta (Garris ja muut, 2002). Haasteen ollessa juuri sopiva, flow-tilassa olevan henkilön taidot tulevat jatkuvasti koetukselle. Mikäli haasteen taso on liian korkea, henkilö tulee levottomaksi, mutta tylsyydenkin tunne voi iskeä haasteen ollessa liian matalalla. Flow-tilassa toiminnalle annettu päämäärä on selvä ja sen saavuttamisen tiellä olevat esteet tuntuvat juuri sopivan haastavilta. (Nakamura ja Csikszentmihalyi, 2002.) Flow'n saavuttamiseksi erilaisten päämäärien ja palautteen on oltava selvällä tavalla annettuja (Garris ja muut, 2002).

Flow-teoria ja ARCS-malli ovat yhteneväisiä siten, että kumpikin niistä pyrkii ohjeistamaan sopivan haastetason, selvien tavoitteiden, keskittymisen ja selvän palautteen tarjoamisessa sekä opetuksen suunnittelussa. Flow-teoria avaa näin ollen sillan opetuksellisen sisällön ja motivaation suunnittelun välille. (Chan ja Ahern, 1999.) Flow'sta kuitenkin puuttuu reflektointi ja oppiminen aktiviteetin aikana. Vaikka omien taitojen käyttäminen olisikin flow'n aikana vedetty äärirajoille, ei niiden kehittäminen ole kuitenkaan välttämättä kovin tehokasta, sillä uusien toimintamallien kehittäminen annetuissa tehtävissä reflektoinnin seurauksena puuttuu lähes kokonaan. Vaikka uusia asioita voitaisiinkin tietokonepeleissä kohdata, ei niistä voida kuitenkaan oppia uutta ilman reflektointia. Oppimista voidaan kuitenkin parantaa flow-tilassa konkreettisten tavoitteiden asettamisella ja estää pelaajan eteneminen pelissä ennen kuin edelliset tavoitteet on saavutettu. (Paras & Bizzocchi, 2005.)

2.2 Pelien avulla oppiminen

Vanhemmat odottavat lähes oletusarvoisesti lasten leikkivän, sillä he ymmärtävät jollain tasolla pelien ja leikkien olevan hyväksi lapsille, sillä se on perustavaa laatua oleva tapa oppia. Pelit ovat alkuperäinen ja luonnollinen tapa opetukseen ja ne ajoittuvat ihmisen jokaiselle aikakaudelle. (Crawford, 1984.) Pelit eivät ole vain sattumalta hyvin lähellä oppimista, vaan nämä asiat ovat yhteenkietoutuneita jopa sillä tasolla, että oppimisteoriaa sovelletaan jopa tiedostamatta pelien suunnittelussa. (Gee, 2008). Pelien pelaaminen itsessään on opettavaista, vaikka se ei näkyisi siinä tietoisella tasolla. Pelaamiseen voi kuitenkin olla oppimista suurempia syitä, kuten esimerkiksi halu tutkia, fantasioida ja olla sosiaalisesti yhteydessä muiden kanssa. (Crawford, 1984.)

Hauskuuden salliminen oppimisen yhteydessä ei ole kielletty, ja kova työ ei ole hauskuudesta kiinni, sillä hauskakin peli voi olla vaativa ja sen hallitsemiseen voi joutua panostamaan runsaasti aikaa ja vaivaa, vaikkakaan pelaaja ei vaivan määrästä välittäisi kaiken hauskan keskellä. Oppiminen ei tunnu työltä, kun se on mukavaa. (Prensky, 2005.) Pelit liitetäänkin usein hauskanpitoon, se ei kuitenkaan ole opetuspelien perimmäinen idea (Kiili, 2005). Pelit tarjoavat meille kuitenkin formaalin ja strukturoidun tavan saada hauskuus mukaan oppimiseen, sillä ne motivoivat pelaajaa niissä asetettujen tavoitteiden saavuttamiseksi nähtyjen ponnistelujen kautta (Prensky, 2005).

Pelit voivat siis toimia opetuksessa ja pelit voivatkin täyttää oppimisympäristöjen perusvaatimukset (Kiili, 2005). On olemassa useita perusvaatimuksia oppimisympäristöille, joita ovat ympäristön motivoivuus, interaktiivisuus, palautteen rikkaus, ja selvät tavoitteet sekä keinot saavuttaa ne (Houser ja DeLoach, 1998). Oppimisympäristön tulisi myös tarjota jatkuvana virtana haastetta, joka ei ole liian helppoa oppilaan tylsistyttämiseksi, mutta ei ole liian vaikeatakaan, jotta mielenkiinto asiaa kohtaan säilyisi. Ympäristön on lisäksi tarjottava suora tapa olla tekemisissä sen kanssa, jolloin oppilas tuntee olevansa välittömässä yhteydessä ympäristön ja annetun tehtävän kanssa. Lopuksi ympäristön häiriöttömyys on eräs kriteeri, jotta oppilaat eivät

kokisi keskeytyksiä. (Houser ja DeLoach, 1998.) Oppimisympäristöt voivat edistää flow-tilan, ja näin oppimisenkin aikaansaamista pelimaailman suunnittelun kautta. Pelien osalta tämä tarkoittaa sitä, että pelimaailman on myötäiltävä kunkin oppilaan omaa osaamistasoa ja näin ollen tarjottava selkeät päämäärät ja välitöntä palautetta flow-tilan ylläpitämiseksi. (Paras & Bizzocchi, 2005.)

Pelit ovat turvallinen ympäristö oppia uutta (Kiili, 2005), sillä peleissä epäonnistuminen on paljon lievempää verrattuna oikeaan elämään, sillä ne antavat kokeilla omia teorioita ja keinoja ylittää este niin monesti kuin pelaaja haluaa, kun taas oikeassa maailmassa riskien ottaminen voi olla liian kallista (Gee, 2004).) Pelit myös antavat meille tavan kokea todellisuutta ilman todellisen maailman riskejä, sillä niissä voidaan komentaa armeijoita ilman oikeaa verilöylyä tai hallita talousimperiumia ilman vaaraa omalle pankkitilillemme (Crawford, 1984). Näin ollen oikeiden seuraamusten sijaan pelit antavat tehdä vapaasti omia päätelmiä pulmia kohdatessamme ja tarjoavat yrityksistämme rakentavaa palautetta rangaistusten sijaan (Prensky, 2005). Myös lasten keskuudessa peleissä esiintyvä kilpailullisuus on hyvin yleistä, motivoivaa ja mukavaa, mutta koulumaailmaan palatessa nämä asiat voivat olla aivan toisin (Gee, 2004).

Mitä tulee tutkimukseen opetuspelien tehokkuudesta, pelien uskotaan olevan motivationaalisesti ja opetuksellisesti tehokkaita, mutta näistä väitteitä ei ole saatu riittävästi todisteita tutkimuksessa (Papastergiou, 2009). Pelien opetuksellisuuden tutkimus sisältää aukkoja, johon tarvitaan lisää tutkimusta. Hänen mukaansa pelejä on tutkittu ennemmin motivaation silmin, mutta ei siitä näkökulmasta, kuinka hyvin ne sopivat opetussuunnitelmaan tai yleensä opettavat. Hän nostaa myös esille etenkin sen, että useissa tutkimuksissa keskitytään ajatukseen opetuksen korvaamisesta peleillä, ei niiden toimivuuteen muun opetussisällön tukijana. (Papastergiou, 2009.)

Randel, Morris, Wetzel ja Whitehill (1992) löysivät tutkiessaan pelejä koskevaa kirjallisuutta, että perinteisen luokkaopetuksen ja pelien välillä vallitsee epätietoisuus siitä, kumpi on tehokkaampi opetusmuoto. Heidän läpikäymissään tutkimuksissa 38 ei löytänyt eroja niiden väliltä, 27 suosi pelejä ja kolme olivat perinteisen opetuksen kannalla. Papastergioun (2009) mukaan iällä voi olla tekemistä oppimisen kanssa, sillä 4-8-vuotiaat lapset oppivat aidosti tiede- ja matematiikkapeleistä ja ovat lisäksi motivoituneita, mutta 8-12-vuotiaat eivät matematiikkapeliä pelattuaan ymmärtäneet pelissä esitettyjen matemaattisten konseptien takana olevia ideoita.

Papastergiou (2009) tutki 16–17-vuotiaiden lukiolaisten tietokoneen muistikäsitteiden oppimista ja oppimisen motivaatiota tietokonepelillä. Tutkimuksessa verrattiin opetuspeliä pelannutta ja ei-pelimäistä oppimisohjelmaa käyttänyttä ryhmää. Tulokset viittaavat siihen, että pelipohjainen oppiminen on tehokkaampaa sekä motivaation parantamisessa että tutkimuksessa opetettujen käsitteiden oppimisessa. Tutkimuksessa käytetty yksinkertainen peli innosti nuoria kovasti ja vielä pidemmälle viedyllä ja hienostuneemmalla pelillä voisi olla vieläkin mittavampi vaikutus. (Papastergiou, 2009.)

Nykyaikainen oppimisteoria pohjautuu siihen, että ihminen oppii kokemusten, ei abstraktien laskelmien tai yleistyksien kautta. Nämä kokemukset tallentuvat ihmisen pitkäaikaiseen muistiin, joka on ihmisen tarpeisiin nähden käytännössä rajaton. Muistiaan käyttäen henkilö voi käydä päässään simulaatioita vanhasta opitusta luodakseen ratkaisuja uusiin ongelmiin. (Gee, 2008.) Opetuspelit voivat tarjota sekä motivoivan, että suoria kokemuksia tarjoavan ympäristön (Kiili, 2005). Pelejä pelatessaan oppilaat voivat rakentaa tietämystään aktiivisesti kokemustensa avulla (Garris ja muut, 2002).

Ihmiset oppivat sellaisista kokemuksista, jotka tarjoavat palautteen välittömästi, jolloin oppija voi nähdä ja arvioida virheensä heti ja sen, missä omat odotukset eivät vastanneet todellisuutta. Palautetta pelaaja voi saada jatkuvasti pelin tilanteesta toiseen, mutta usein erityisesti pelin tason lopussa tai loppuvastuksen kohdalla, jolloin pelaajalla on tarve yhdistellä useita oppimiaan taitoja. (Gee, 2008.)

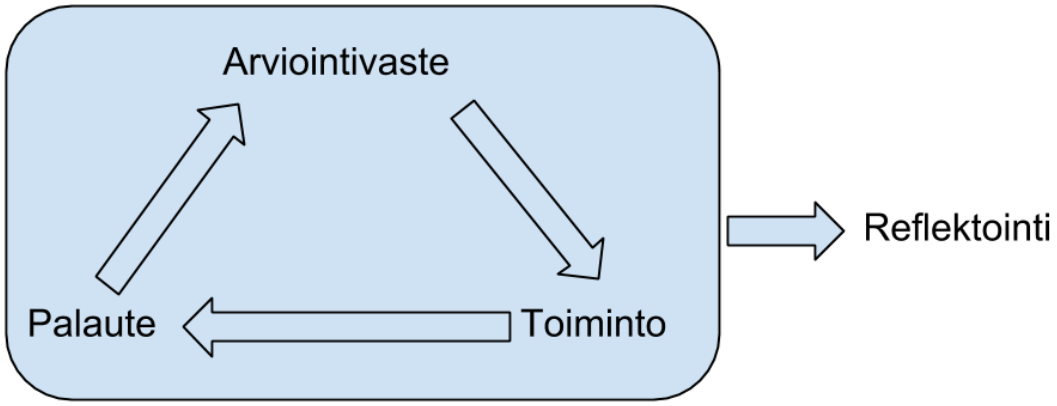
Kokemukset voivat olla kuitenkin käyttökelpoisia ongelmanratkaisussa vain, jos ne ovat strukturoituja, sisältävät tavoitteen ja ovat tulkittavissa ongelmatilanteen mukaisesti (Gee, 2008). Kokemukset tallentuvat niihin liittyvien tavoitteiden ja käytännön toimivuuden mukaan, ja oppiessaan kokemuksesta ihminen poimii niiden sisältämät opetukset ja tavat käyttää niitä parhaimmalla tavalla tulevaisuudessa. Pelit antavat runsaasti mahdollisuuksia soveltaa vanhojen kokemusten sisältämiä oppeja uusiin samankaltaisiin tilanteisiin, mikä on eräs ongelmien yleistettyjen ratkaisujen rakentamiseen vaadittu asia. (Gee, 2008.)

Tällainen tilannekohtainen ymmärtäminen on peleissä voimakkaasti läsnä (Gee, 2003). Tilannekohtainen ymmärtäminen voidaan määritellä sellaisena asiana, jossa opeteltavia asioita voidaan käyttää oikeissa konteksteissa ongelmanratkaisuun (Gee, 2004). Oppikirjamainen tapa, jossa lukijaa pommitetaan jatkuvasti uusilla asioilla ilman tietoa niiden soveltamisesta, on peleissä korjattavissa niin, että termit tuodaan niissä ilmi sitä mukaa kun niitä voidaan pelissä käyttää ongelmanratkaisuun (Gee, 2003).

Kokemuspohjaisen oppimisen pohjalta voidaan sanoa, että eräs hyvin suunnitellun pelin merkki on se, että se tarjoaa pelaajalle kokemuksia ensinnäkin sen mahdollistaman oppimisen hauskuuden, ja toisaalta pelin selättämisen halun avulla (Gee, 2008). On kuitenkin huomattava, että pelin sisällön on oltava yhteensopiva pelaajan kanssa, jotta oppimista voi ylipäätään tapahtua (Prensky, 2005).

Oppiminen on syklistä, jonka eräänä oleellisena osana on reflektointi (Kolb, 1984). Kyseisessä syklissä reflektointia seuraa uusien toimintamallien rakentaminen vanhasta opitusta, jonka jälkeen henkilö toimii uusien toimintamalliensa mukaan. Saatuaan palautteen toiminnasta hän voi jälleen reflektoida toimintaansa ja muuttaa toimintamallejaan. (Kolb, 1984.) Garris ja muut (2002) kuvaavat samankaltaisen syklin, joka on läsnä pelatessa, ja se on esitetty kuvassa 2. Kyseisessä syklissä pelin sisältämä opetuksellinen sisältö saa pelaajassa aikaan pelin arvosteluvasteen, joka voi hyvässä tapauksessa olla mielenkiinnon tai mielihyvän tunne. Tämä johtaa pelaajan tekemään jonkin toiminnon pelissä, josta peli antaa takaisin palautetta. Palautteen saaminen saa pelaajassa jälleen aikaan arviointivasteen. Kun pelisykli päättyy, voidaan pitää erillinen jälkipohdinta, jolloin lopullinen oppiminen voi tapahtua. (Garris ja muut, 2002.)

Kuva 2. Pelaamisen aikana esiintyvä arviointi-toiminto-palaute-sykli (Garris ja muut, 2002).



Ihminen oppii aktiivisen osallistumisen kautta ja pelin jälkeiset aktiviteetit, kuten jälkipohdinta, saavat aikaan tehokkaan ympäristön oppimiselle (Garris ja muut, 2002; Gee, 2008). Jälkipohdinnan aikana voidaan reflektoida pelissä tapahtuneita asioita ja pohtia mitä olisi voitu tehdä pelin aikana toisin, jotta jokin asia olisi onnistunut pelissä paremmin. On syytä korostaa, että reflektointi olisi suotavaa myös selostaa muille läsnäolijoille. (Gee, 2008.)

Jälkipohdinnan lisäksi pelaajan on pelaamisen aikana jatkuvasti tulkittava kokemaansa ja saatava selityksiä virheilleen ja vääristyneisiin odotuksiin. Pohdintaa voidaan pelaajassa nostattaa esimerkiksi hiljalleen vaikeutuvalla pelillä tai loppuvastuksilla. (Gee, 2008.) Pelien kuuluisi esittää ilmiöitä reflektoivaan sävyyn ja antaa pelaajan testata kehittämiään hypoteeseja, ei olla loputonta toistoharjoitusta ilman reflektiota (Kiili, 2005).

Reflektointiin vaikuttaa myös pelin sosiaalinen asetelma, jossa voi olla kyse sekä oikeassa maailmassa tunnetuista ihmisistä, mutta myös verkon yli käytävästä pelin keskustelusta ja pohdinnasta. Pelit voidaan erotella kahteen osaan, joista eräissä tulee mukana sosiaalinen asetelma ja muihin, joissa sitä ei tule. Oppilaat voivat oppia reflektoidessaan asiaa keskenään, mutta myös kuunnellessaan asiantuntijoiden tulkintoja tai selityksiä kokemastaan. (Gee, 2008.)

2.3 Pelien suunnittelu

Opetuspelit voivat viedä oppilaan maailmoihinsa harjoittelemaan sellaisiakin tehtäviä, jotka oikeassa maailmassa tuntuisivat tylsiltä (Prensky, 2005). Oppiminen peleissä ei ole luonteeltaan sellaista, että päästetään pelaajat valloilleen tekemään mitä tahtovat, vaan opetuspelien on ohjattava pelaajaa tarkoin. Ohjaus voi tapahtua pelisuunnittelun ja pelimaailman avulla, sekä pelissä annetun tiedon kautta. (Gee, 2008.)

Pelien suunnitteluun on panostettava tarkasti, jotta pelaaja päätyisi harjoittelemaan pelissä juuri oikeita asioita (Prensky, 2005). Vääriin asioihin keskittymistä on varottava myös peliympäristön monimutkaisuuden ja houkuttelevuuden osalta, jotka voivat olla harhauttavia tekijöitä pelaajan ajamisessa pois oppimisprosessista. Peleihin lisättyä maltillisen stimuloivaa ympäristöä kannattaa kuitenkin suosia siitä syystä, että ilman pelillisiä elementtejä opetusohjelma jää kuitenkin stimuloivan opetuspelin varjoon mitä tulee asioiden oppimiseen ja muistamiseen (Papastergiou, 2009).

Pelien yleisiä suunnitteluperiaatteita hyödyntäen peli saa sellaisen rakenteen, joka täyttää useita oppimisen edellytyksenä olevia ehtoja (Gee, 2008). Opetuspelejä suunniteltaessa on otettava huomioon pelimekaniikka ja oppimiseen liittyviä näkökulmia (Prensky, 2005). Pelin suunnitteluun vaikuttaa myös se, kuinka sitä tullaan käyttämään opetuksessa hyväksi. Hyvän oppimispelin vaatimus on pelien yleisen houkuttelevuuden yhdistäminen interaktiivisen oppimisprosessin kanssa. (Prensky, 2005.) Yhden mainitun ominaisuuden ollesssa kuitenkin liian vahva, peli sukeutuu joko puhtaiden viihdepelien joukkoon tai liian paljon tietokonepohjaista harjoittelua muistuttavaksi. Tästä syystä peli- ja oppimiselementit on sekoitettava oikeassa suhteessa. (Gee, 2008.)

Koska tunnetilat ovat läsnä pelejä pelatessa, voi niiden huomioiminen olla pelisuunnittelussa hyödyllistä. Garris ja muut (2002) käyvät läpi neljä tunnetilaa, jotka voivat esiintyä heidän kehittämänsä pelisyklin arviointivasteen aikana. Kyseiset tunteet ovat mielenkiinto, mielihyvä, tehtävään uppoutuminen ja itsevarmuus. Erityisesti mielenkiinto on heidän mukaansa voimakkaasti läsnä opetuspeleissä verrattuna tavanomaiseen luokkaopetukseen. (Garris ja muut, 2002.)

Tehtävään uppoutuminen voidaan määritellä useiden tekijöiden kautta, joita ovat mm. tunne siitä, kuinka välittömiä ja luonnollisia peliympäristössä tehdyt toiminnot pelaajan mielestä ovat, kuinka rikkaan aistillisen elämyksen peli tarjoaa, ja kuinka realistinen se on (Witmer ja Singer, 1994). Oppiminen ja vahvempi mieleenpainaminen vahvistuvat uppoutumisen myötä (Hannafin ja Hooper, 1993). Pelaamisen itsevarmuus saadaan aikaan sen ansiosta, että peleissä voidaan tehdä asioita ilman pelkoa ikävistä oikean maailman seuraamuksista (Garris ja muut, 2002).

On olemassa useita luokitteluja liittyen siihen, mitkä osat ovat pelien kannalta tärkeitä. Useat pelien suunnittelun ulottuvuudet ovat kirjallisuudessa sotkeutuneet niin, että useat tutkimukset puhuvat samoista asioista eri termeillä (Garris ja muut, 2002). Esimerkkinä eräs luokittelu (Kiili, 2005), jonka mukaan oleellisia opetuspelin suunnittelussa huomioon otettavia asioita ovat tarinankerronta, grafiikat ja äänet, ja pelin tasapainoisuus. Tarinankerronta yhdistää pelin haasteet suuremman mittakaavan ongelmaan ja mitä monimutkaisempi peli, sitä suuremmassa roolissa tarinankerronta on. Yksinkertaisessa pelissä ei tarvitse olla esimerkiksi kuin lyhyt teksti pelin alussa, joka asettaa pelin tarinan sisään. (Kiili, 2005.) Garris ja muut (2002) selvittivät kirjallisuudessa esiintyvien pelien ulottuvuuksien pohjalta yleistyksen siitä, mitkä ulottuvuudet esiintyvät niissä yleisesti. Heidän laatimaansa yleistävään luokitteluun kuuluvat fantasia, säännöt ja tavoitteet, sensoriset virikkeet, haasteet, mysteeri, ja hallinta.

Pelaaminen on oikeasta elämästä erossa oleva fantasiaa pursuava aktiviteetti, jolla ei ole todellisuudessa kirjaimellista vastinetta (Garris ja muut, 2002). Fantasialla voimme pyrkiä kutkuttamaan pelaajaa ja tarjoamaan peliin mielenkiintoisen asetelman. Fantasiaan kuuluuvat grafiikat, äänet, interaktio pelimaailman kanssa, ja kaikki muutkin asiat, joita pelaajat kokevat pelissä. (Hoonhout, Diederiks ja Stientstra, 2004.)

Pelit sisältävät mielikuvitusmaailman, jonka tapahtumilla ei ole vaikutusta oikeaan maailmaan ja kun pelaaja on pelin ääressä, sen ulkopuolisilla asioilla ei ole hänelle merkitystä (Hoonhout ja muut, 2004). Fantasia on tekijä, joka saa aikaan mielikuvia ja sosiaalisia tilanteita, jotka eivät oikeasti ole olemassa (Malone ja Lepper, 1987). Pelit edustavat subjektiivista todellisuutta, sillä ne ovat objektiivisessa mielessä epätodellisia, eli ne eivät luo oikeaan fyysiseen todellisuuteen pelihahmoja tai muitakaan edustamiaan asioita. Silti pelit ovat pelaajan mielestä subjektiivisesti tosia, joten ihmisen kyky fantasioida on oleellinen tekijä myös kyvyssä tehdä objektiivisesta asiasta subjektiivisesti todellisen. Näin ollen fantasia on asia, joka tekee pelistä pelaajan mielestä todentuntuisen. (Crawford, 1984.)

Pelit ovat todellisuuden alaluokka, sillä ne perustuvat aina oikeasta todellisuudesta otettuihin elementteihin. Ne, mitkä asiat todellisuudesta valitaan peliin mukaan tuovat fokuksen peliin. Liian todentuntuinen peli, joka kuvaa oikeaa maailmaa täydellisesti, on altis menettämään fokuksensa. Pelien ei siis tarvitse kuvata todellisuutta kovinkaan tarkasti, vaan sen sijaan todellisuuden osat voivat olla pelaajan kokeman fantasian uskottavuuden vahvistamisen roolissa. (Crawford, 1984.)

Fantasialla voi oppimisaktiviteettiin upotettuna olla oppilaiden mielenkiintoa ja oppimista vahvistava vaikutus (Cordova ja Lepper, 1996). Fantasiat voivat tarjota analogioita oikean maailman prosesseihin ja antavat pelaajan kokea asioita eri näkökulmista (Malone ja Lepper, 1987), ja uuden tiedon oppimiselle on suurempi valmius, kun se tarjotaan kuvitellussa ja mielenkiintoisessa kontekstissa (Garris ja muut, 2002). Opetusta varten tehdyt pelit eivät useinkaan kykene upottamaan pelaajaa subjektiiviseen fantasian maailmaansa, koska ne eivät sulauta pelimekaniikkaa opetussisältöä ja -kontekstia (De Castell ja Jenson, 2003).

Tämä liittyy siihen ongelmaan, että opetukselliset pelit kuorruttavat pelimaailman fantasialla (eksogeeninen fantasia), jolloin siitä tulee muusta pelistä irrallinen tai päälleliimattu (Rieber, 1996). Tällaisissa peleissä on yritetty parantaa oppimispuolta ulkoisesti, mutta oppimisen elementtejä ei ole sulautettu peliin ja sen mekanismeihin näin tyrehdyttäen pelaajan mahdollisuuden uppoutua peliin. Tästä eroten voidaan mainita pelit, joissa tämä sulauttaminen on tehty tarpeeksi hyvin, jolloin pelaaja ei kykene erottamaan peliä ja opetussisältöä toisistaan (endogeeninen fantasia). Opetussisältö tulee myös automaattisesti mielenkiintoisemmaksi, mikäli mukana tuleva endogeeninen fantasia kiehtoo pelaajaa. (Rieber, 1996.)

Pelaaminen tapahtuu niin kutsutun *taikakehän* (magic circle) sisällä, joka määrittelee rajoitetun pelimaailman. Taikakehän sisällä pelaajalla on kyky häivyttää kaikki epäusko ja ottaa vastaan pelimaailman tuomat asiat (Salen ja Zimmerman, 2003). Taikakehä tulisi ottaa suunnittelussa huomioon ja se, miten pelaajat pääsevät astumaan sen sisään (Paras & Bizzocchi, 2005). Taikakehään voidaan päästä sinne vapaaehtoisesti uppoutumalla ja luovuttamalla mielensä fantasiamaailmalle. Sinne päästyään pelaaja ei välitä ulkomaailman asioista, vaan keskittyy täysin pelimaailman tapahtumiin (Murray, 1997). Pelaajien on hyväksyttävä pelimaailman säännöt ja rajoitukset, mikä vaatii pelimaailmaan liittyvien epäuskojen hylkäämistä ja tästä pelaaja saa kuin vastalahjana mahdollisuuden hyödyntää pelimaailmaa täysipainotteisesti (De Castell ja Jenson, 2003).

Pelin säännöt rajoittavat pelaajaa, mutta kuvaavat samalla siihen asetettujen päämäärien rakennetta (Garris ja muut, 2002). Myös selkeät, tarkat ja samaan aikaan vaikeat tavoitteet pelissä saavat pelaajan toimintakykyisemmäksi (Locke ja Latham, 1990). Merkityksellisen oloiset peliympäristöt, joissa käy selkeästi ilmi pelin päämäärät ja niiden hierarkinen rakenne, voivat parantaa motivaatiota ja toimintakykyä (Garris ja muut, 2002). Peli vangitsee pelaajan huomion ja antaa motivaatiota, kun siinä asetetut tavoitteet ja palaute pelaajan nykyisestä tilasta ovat epäsopusoinnussa. Vaikka tietäisimme pelin säännöt, emme voi kuitenkaan tarkkaan tietää pelaajien tavasta pelata sitä. Säännöt ja tavoitteet eivät saa rajoittaa pelaajaa liikaa, joten pelissä tapahtuvien asioiden on kyettävä tapahtumaan esimerkiksi pelaajan pelityylin ja strategian mukaan. (Garris ja muut, 2002.)

Pelit tarjoavat oikeasta maailmasta poikkeavia aistimuksia, jotka tarjoavat mielihyvän tunnetta (Garris ja muut, 2002). Äänitehosteet, muuttuvat grafiikat sekä muunlaiset herätteet, jotka koetaan oudoiksi tai vieraiksi ovat huomiota herättäviä ja stimuloivia (Malone ja Lepper, 1987). Lisäksi animoidut grafiikat ovat oppitilanteissa motivaatiota herättäviä ja oppilaat palaavat herkästi niitä sisältäviin aktiviteetteihin (Rieber, 1996).

Konfliktit ovat keskeinen asia peleissä (Crawford, 1984). Peleissä olevat esteet ovat pelaajan lopullisen päämäärän saavuttamisen tiellä ja jos esteet ovat passiivisia, on kyseessä pelkkä pulmapeli, johon peruslogiikan löydyttyä pelaaja kyllästyy. Mikäli esteet ovat dynaamisia, eli niissä on jokin taho aktiivisesti estämässä pelaajaa, on kyseessä konflikti. (Crawford, 1984.)

Ihmiset kaipaavat sopivan tasoista haastetta, jonka vaikeusaste ei toisin sanoen ole liian helppoa tai vaikeaa (Malone ja Lepper, 1987). Pelin tulisi vaikeutua hiljalleen loppuun saakka, eikä tehdä yllättäviä piikkejä vaikeusasteessa (Kiili, 2005). Eräs tapa saavuttaa järkevä haastetaso on selkeiden päämäärien mainitseminen, vaikkakin niiden saavuttamisen tulee tuntua pelaajasta epävarmalta. Lisäksi pelissä annetun ja päämäärien saavuttamista helpottavan tiedon kuuluu olla sopivalla tavalla tulkinnanvaraista. (Garris ja muut, 2002.)

Mitä tulee pelin tarjoamiin ongelmiin, niissä tulisi ottaa huomioon pelajaan ja tehtävän lisäksi tehtävän suorittamiseen tarvittava artefakti (Kiili, 2005). Jos tehtävä tai artefakti on liian monimutkainen, flow-tilan saavuttaminen hankaloituu keskittymisen herpaantumisen myötä. Myös pelin huono käytettävyys, jossa pelaaja joutuu miettimään aktiviteetin kannalta epäolennaisia seikkoja, alentaa mahdollisuuksia flow-tilan syntymiseen. (Kiili, 2005.) Tehtävien ja päämäärien on oltava jollain tavalla merkityksellisiä ja pelin aktiviteettien liittäminen pelaajan itse arvostamiin taitoihin ja muihin asoihin on suotavaa. Kilpailullinen tai yhteistyömäinen henki pelissä voi saada päämäärät tuntumaan enemmän merkityksellisiltä. Pelissä on oltava useita päämääriä ja tasojen on oltava progressiivisesti vaikeutuvia, mutta pelaajan on kyettävä näkemään edistyksensä tietyn päämäärän saavuttamisessa. (Garris ja muut, 2002.)

Pelin tasojen edetessä pelaaja kohtaa kertaalleen sellaisia asioita, joita hänen eteensä on aiemmin tullut. Näin ollen pelaaja voi käyttää vanhaa opittua hyväkseen uusien ongelmien ratkaisemisessa. Tästä syystä pelin tasojen suunnittelu on olennainen asia oppimispeliä suunniteltaessa. (Gee, 2008.) Peli tulisi tasapainottaa niin, että yksittäinen strategia ei ole ylivoimaisesti muita tehokkaampi, vaan pelaajan aidon osaamistason tulisi olla määräävä tekijä pelissä etenemiselle. On painotettava, että pelissä pärjäävien pelaajien kehuminen ja palkitseminen pelin sisällä ei auta heikommin menestyviä, joten tällaisessa tilanteessa tulisi heikommin pelaavan peliä sovittaa hänen osaamistasoonsa. (Kiili, 2005.)

Peleillä on mahdollisuus tarjota pelaajalle ongelmia, jotka voivat olla joko hyvin tai huonosti jäsenneltyjä (Kiili, 2005). Hyvin jäsenneltyihin ongelmiin on selvä vastaus, kun taas huonosti jäsennellyt eivät välttämättä sisällä selvää tietoa ongelmasta ja lopullisesta ratkaisusta. Huonosti jäsennellyt eivät ole järjesteltyjä ongelmia huonompia, sillä ne ovat pelaajalle tarkoituksenmukaisempia niiden monipuolisempien ratkaisumahdollisuuksien vuoksi. Ongelmanratkaisu voidaan liittää tutkivaan oppimiseen ja tämän vuoksi pelit ovat hyviä opettamaan pelaajalle etenkin uusia sääntöjä ja ideoita löytämisen kautta. (Kiili, 2005.)

Mysteeri on ulkoinen osa peliä ja se sytyttää pelaajan uteliaisuuden, joka itsessään on tunne pelaajan sisällä. Uteliaisuus syntyy silloin, kun pelaajalla on tunne omien tietojensa riittämättömyydestä jotakin asiaa kohtaan. Aukko tiedoissa pitää pelaajan uteliaana, mutta se ei riitä pitämään häntä tyytyväisenä, mikäli tietoaukko on väärän kokoinen. Liian pieni aukko saa pelaajan ohittamaan saapuvan tiedonpalasen ja liian suuri aukko saa hänet sen sijaan hämmentymään sen monimutkaisuuden vuoksi. (Garris ja muut, 2002.)

Mysteerin saavuttamista helpottaa se, että pelien fantasiaelementit itsessään antavat kohdata pelissä epätavallisia asioita lisäten uteliaisuutta. Myös muita tekijöitä mysteerin rakentamisessa ovat mm. pelissä annetun tiedon epäyhteneväisyys, epätäydellisyys tai monimutkaisuus, pelin uutuusarvo, yllätykset, ja pelin oleminen ristiriidassa pelaajan odotusten kanssa. (Garris ja muut, 2002.)

Pelien hallintaulottuvuus voidaan toteuttaa joko niin, että itse opetusohjelma hallitsee kaikkia opetuksen puolia, tai että pelaajalle on annettu hallinta sen osalta. On todisteita oppilaan hallinnan antavan myönteisempiä reaktioita ja motivaatiota tietokoneen pitämään hallintaan verrattuna. Erityisesti pelit antavat oppilaalle hallinnan tunteen, jolloin he voivat itse valita strategian, pelin suunnan ja tehdä pelin kulkuun vaikuttavia päätöksiä. (Garris ja muut, 2002.)

2.4 Yhteenveto

Tässä pelejä koskevassa kirjallisuuskatsauksen osassa käytiin läpi opetuspelien suunnittelussa huomioon otettavia periaatteita. Pääpiirteittäin nämä periaatteet voidaan jakaa motivaation nostamiseen, pelien avulla oppimiseen, ja yleisiin peleihin sisällytettäviin ulottuvuuksiin.

Kellerin (1987) ARCS-malli kuvailee neljä strategiaa, jotka ehdottavat keinoja oppilaiden motivaation parantamiseksi. Huomiostrategialla saadaan aikaan uteliaisuuden ja mielenkiinnon tunne, mitä voitaisiin opetuspelissä nostaa kohderyhmälle sopivalla kielellä ja kontekstilla. Relevanssistrategiaa käytetään asian saattamiseksi merkityksselliseksi oppilaille, mikä olisi opetuspeleissä mahdollista saada aikaan tuomalla edellisen strategian tavoin kohderyhmälle jotakin heitä lähellä olevaa esimerkiksi populaarikulttuurista tai tekemällä peli yleensä tutun oloiseksi heille. Itsevarmuusstrategialla pyritään onnistumisen tunteen aikaansaamiseen, joka ohjelmointipelissä voidaan yrittää toteuttaa avoimilla ja runsailla tehtävillä ja itse kirjoitettavalla koodilla pelissä. Tätä auttaa myös tehtävien tekeminen sopivan tasoiseksi, jolloin pelaajat eivät koe turhautuvansa liian hankalien tehtävien edessä. Viimeinen mallin strategia, eli tyytyväisyysstrategia tähtää opitun tiedon hyödyntämisen kautta tyytyväisyyden tunteeseen. Tähän pelit sopivat hyvin, sillä normaalisti opitut asiat eivät välttämättä heti näy käytännössä, mutta esimerkiksi ohjelmointipelissä juuri opittuja asioita voidaan suunnitella hyödynnettäväksi heti niiden opettamisen jälkeen erilaisissa tehtävissä.

Motivaation kokemiseen liittyy myös flow-tila, jonka saavutettuaan henkilö keskittyy täysin tehtäväänsä ja on optimaalisen motivoituneessa ja suorituskykyisessä tilassa (Nakamura ja Csikszentmihalyi, 2002). Tähän päästää peleissä säätämällä vaikeusaste sopivaksi ja ottamalla ylimääräiset häiriötekijät pois, jotta pelaaja voi keskittyä tehtäväänsä täysin.

Pelit ovat oppimisen kannalta hyvä ympäristö (Kiili, 2005). Ne tuovat strukturoidulla tavalla hauskuuden oppimiseen (Prensky, 2005) ja ovat etenkin turvallinen ympäristö kokeilla erilaisia asioita (Kiili, 2005). Pelit voivat olla hyviä kokemuspohjaisen oppimisen kannalta, sillä niissä kokemukset voidaan liittää tiettyyn ongelmatilanteeseen, jolloin kokemusta voidaan käyttää jatkossakin uusien ongelmatilanteiden ratkaisemiseen. Ohjelmointipeliin tätä voidaan soveltaa niin, että kunkin ohjelmointikäsitteen kohdalla pohditaan, minkälainen ongelma kuvaisi mahdollisimman hyvin sitä. Toisin sanoen suunnitellaan sellainen olennainen ongelma, jonka ohjelmointikäsite ratkaisee luonnollisesti.

Oppimiseen kuuluu myös reflektointi, jolloin oppilas pohtii tekemäänsä sekä pelaamisen aikana että sen jälkeen erityisessä jälkipohdinnassa (Gee, 2008). Pelit antavat mahdollisuuden jatkuvaan pohdintaan pelin aikana, sillä tehtävästä toiseen pelaajat joutuvat reflektoimaan miksi aiemmin jokin asia meni hyvin tai huonosti, ja miten toimintaa voitaisiin muuttaa nykyisessä ongelmassa. Myös kenttien jälkeen on mahdollista laittaa yhteenveto kentässä tapahtuneista asioista ja palauttaa pelaajan mieleen mitä kentässä kuului oppia. Tällä tavoin pelien avulla oppimisessa ei tarvita opettajaa linkittämään pelissä esiintyneitä asioita opetettavaan asiaan, koska peli voi tehdä sen itse.

Oppimispeli tulisi suunnitella tarkoin, sillä ilman kunnollista ohjausta pelaaja voi ajautua tekemään oppimisen kannalta epäoleellisia toimintoja (Papastergiou, 2009; Prensky, 2005). Ehkä tästä syystä pelimekanismi tulisi suunnitella niin, että se on vahvasti sidottu

muuhun peliin, jolloin pelaaja ei karkaa tekemään asioita, joita hänen ei haluta tekevän. Toisaalta yleisiä suunnitteluperiaatteita noudattaen on mahdollista saada aikaan oppimista edistävä peli (Gee, 2008). Näitä periaatteita ovat Garris ja muut (2002) keränneet kirjallisuudesta ja koonneet niistä yhtenäisen luettelon pelien yleisistä ulottuvuuksista, joita ovat fantasia, säännöt, sensorinen stimulaatio, haaste, mysteeri ja hallinta. Näiden ulottuvuuksien avulla pelin voidaan olettaa saavan sellaisen ulkomuodon, joka muistuttaa enemmän peliä ja vähemmän opetusohjelmaa. Fantasian kohdalla on kuitenkin muistettava, että välttääksemme eksogeenisen fantasian (Rieber, 1996) tunnetta, pelin opetussisältö ja fantasia on suunniteltava toisiinsa yhteensopivaksi. Tätä voidaan todennäköisesti parantaa suunnittelemalla ne yhtä aikaa, jolloin kumpikin voidaan ottaa yhtäläisesti huomioon.

3. Ohjelmoinnin oppiminen

Ohjelmoinnin opettaminen ja oppiminen on vuosikymmeniä vanha tutkimusaihe ja tähän viittaa se, kuinka jo Mayer (1981) pyrki selvittämään miten opiskelijoille saataisiin opetettua ohjelmointia paremmin. Useissa tutkimuksissa sanotaan, kuinka ohjelmointi koetaan pulmalliseksi esimerkiksi yliopistojen kursseilla (Piteira ja Costa, 2012).

Tämän luvun ensimmäinen alaluku esittelee eroja ohjelmoinnin aloittelijoiden ja osaajien välillä. Sitä seuraavassa alaluvussa käydään läpi ohjelmoinnin oppimisen aikana usein esiintyviä vaikeuksia. Tämän jälkeen tarkastellaan ohjelmoinnin pedagogiikkaa oppimisen ja opettamisen näkökulmista omissa alaluvuissaan ja esitellään yleisiä ongelmia kummassakin ja keinoja parantaa niitä. Erityisesti ohjelmointiympäristön ominaisuudet otetaan oppimisessa ja opettamisessa huomioon kyseisissä alaluvuissa. Lopuksi esitellään nykyisiä relevantteja oppimissovelluksia, jotka valottavat ohjelmoinnin opetussovellusten nykytilaa.

3.1 Ohjelmoinnin aloittelijat ja osaajat

Winslow (1996) väittää, että vie noin kymmenen vuotta saada aloittelevasta ohjelmoijasta erityisosaajan tasoinen. Dreyfus, Dreyfus ja Athanasiou (1987) jakavat tämän kehityksen viiteen vaiheeseen, jotka ovat aloittelija, edistynyt aloittelija, pätevä, asiantuntija ja erityisosaaja. Se, mikä erityisesti erottaa aloittelijan edistyneemmästä ohjelmoijasta on Von Mayrhauserin ja Vansin (1995) mukaan se, että kokeneemmat jaottelevat tietoansa erikoistuneisiin tietämysskeemoihin sekä pitävät tietonsa järjestäytyneenä niiden alla piilevien algoritmien mukaisesti. Heidän mukaansa osaajat eivät lisäksi ajattele muistiinsa tallennettuja algoritmeja pinnallisella tasolla esimerkiksi ohjelmointikielen avulla.

Vasta-alkajilla on myös useita eroavaisuuksia osaajiin, joista eräs on vähäinen määrä erityisiä malleja, joita hyödyntäen voidaan pureutua tietynlaisiin ongelmiin. Osaajat kykenevät valitsemaan lähes tiedostamattomasti parhaimmat ratkaisumallit esitettyyn ongelmaan, joten heidän ei tarvitse ajatella niinkään ohjelman yksityiskohtia rivi riviltä aloittelijoiden tapaan, vaan he voivat hahmottaa ohjelman kokonaisuutena. (Lahtinen, Ala-Mutka ja Järvinen, 2005.) Aloittelijat ovat rajoittuneet korkean tason tietämykseen ohjelmoinnista, eivätkä he omaa laajaa valikoimaa mentaalimalleja tai osaa soveltaa tietojaan vielä parhaimmalla mahdollisella tavalla erilaisissa tilanteissa (Winslow, 1996).

Ohjelmointikykyjen karttuessa henkilö alkaa käyttää sekä yleisiä että erikoistuneita ongelmanratkaisutaitoja annettujen ongelmien analysoimiseksi ja ymmärtämiseksi. Osaajat antavat myös tilaa itselleen ymmärtää annettu ongelma monelta kantilta (Von Mayrhauser ja Vans, 1995.) Ohjelmointiosaajien piirteet eivät lopulta eroa merkittävästi muiden alojen osaajista. Useiden alojen asiantuntijat kykenevät näkemään ja sopeutumaan havaittuihin malleihin sekä pystyvät kehittämään tehokkaita strategioita ongelmien ratkaisemiseksi. (Robins, Rountree ja Rountree, 2003.)

3.2 Ohjelmoinnin vaikeus

Ohjelmointikielien rakenteiden, eli syntaksin, ymmärtämisessä on tiettyjä usein esiintyviä hankaluuksia. Esimerkiksi muuttujien luominen on vaikeammin ymmärrettävä asia kuin niiden muuttaminen tai testaaminen (Samurcay, 1989). Lisäksi yleisesti käytetty *for*-toistorakenne on erityisen hankala oppia, sillä aloittelijalla on vaikeuksia ymmärtää piilossa tapahtuvia toimintoja kyseisessä rakenteessa (Du Boulay, 1986).

Loppujen lopuksi ohjelmoinnin vaikeus juontuu mittavammin muista tekijöistä kuin syntaksin opettelusta (Winslow, 1996). Aloittelijoilla on eniten vaikeuksia ohjelmien datavirtojen ja ohjelman tarkoituksen ymmärtämisessä. Tämän jälkeen vaikein asia on kontrollivuon ymmärtäminen ja vähiten ongelmia tuotti varsinaisten ohjelmointikielen perusrakenteiden hallinta. (Corritore ja Wiedenbeck, 1991.)

Suunnittelemisen vähyys näkyy aloittelijoiden ohjelmoinnissa etenkin siitä syystä, että heillä on vaikeuksia yhdistää ohjelmointirakenteita kokonaiseksi ohjelmaksi. Vaikka aloittelija kykenisi ratkaisemaan annetun ongelman paperilla, sen muuntaminen koodiksi tuottaa vaikeuksia. (Winslow, 1996.) Lisäksi eräs puute on ohjelman testauksen vähyys, ja vain pienten paikallisten korjausten tekeminen koodiin sen sijaan, että aloittelija muokkaisi sitä kokonaisvaltaisemmalla otteella (Linn ja Dalbey, 1989).

Vaikka syntaksi olisikin opiskelijan hallinnassa, sen soveltaminen ongelmanratkaisuun näyttäisi olevan suurempi hankaluus (Ala-Mutka, 2004) sekä jo pelkästään eri ohjelmoinnissa käytettyjen lausekkeiden yhdisteleminen tuottaa useimmille vaikeuksia (Winslow, 1996). Silti opetusta tehdään usein niin, että opetetaan ohjelmointikielen rakenteet luennoiden siinä uskossa, että passiivinen vastaanottaminen voisi opettaa ohjelmoinnin taidon (Gomes ja Mendes, 2007).

3.3 Ohjelmoinnin oppiminen ja opettaminen

Ohjelmoinnin oppiminen pitää sisällään useita asioita, joita ovat mm. itse ohjelmointikielen ominaisuuksien ja syntaksin oppiminen, ohjelmien suunnittelun osaaminen ja luetun koodin ymmärtäminen (Ala-Mutka, 2004). Davies (1993) lajittelee ohjelmointiin liittyvän tiedon kahteen osaan. Hänen mukaansa on olemassa ohjelmointitietämystä, johon kuuluu ohjelmointirakenteiden toiminnan ymmärtäminen, ja ohjelmointistrategiat, joissa on kyse ohjelmointirakenteiden oikeanlaisesta käytöstä ja soveltamisesta tietyissä tilanteissa. Robins ja muut (2003) väittävät ohjelmoinnin oppikirjojen keskittyvän pääosin pelkkään ohjelmointitietämykseen. Lisäksi Ala-Mutkan (2004) mukaan oppilaille opetettaessa ohjelmointia kyseessä on usein oppikirjamainen kielen rakenteiden läpikäyminen, vaikka muitakin mainittuja seikkoja olisi hyvä tuoda esille.

Ohjelmoinnissa on toki tärkeää muistaa perusrakenteet, jotka voidaan oppia pelkästään ”hauki on kala”-tyyppisellä ulkoa muistamisella, mutta ohjelmoinnin varsinainen oppiminen on mahdollista vasta syvällisemmän ymmärtämisen kautta, joka saavutetaan ainoastaan itse harjoittelemalla, ei kirjoja lukemalla (Jenkins, 2002). Onkin näyttöä, että käytännön harjoitus on eräs tehokkaimmista tavoista opetella ohjelmointia (Tan, Ting ja Ling, 2009). Uusien ohjelmointikäsitteiden tuominen opetuksessa aiemmin opittujen käsitteiden kanssa tukee myös oppimista (Gomes ja Mendes, 2007).

Ohjelmointi on sekoitus useita taitoja, joista varsinaisen kielen osaaminen on siis vain yksi osa. Eräs toinen oleellinen osa on ongelmanratkaisu, jonka yleisessä osaamisessa näyttäisi Gomesin ja Mendesin (2007) mukaan olevan aukkoja. Vaikka heidän mukaansa taidossa näyttäisi olevan puutetta, väittävät he myös kyseisen taidon parantuvan ohjelmointitehtäviä ratkaisemalla.

Suurempi ongelma ongelmanratkaisua koskien on vaikeus suorittaa tehtävä, jonka alkutilanteessa on annettu ratkaistava ongelma ja lopullisen tuotoksen pitäisi olla sen ratkaiseva ohjelmakoodi (Ala-Mutka, 2004). Tällaisen tehtävän ratkaisuprosessiin kuuluu ongelman ymmärtäminen, ongelman ratkaisun määrittäminen, ratkaisun kirjoittaminen ohjelmakoodiksi ja ohjelman testaaminen sekä viankorjaus (Winslow, 1996).

Ongelmanratkaisuun liittyvä pulma voitaisiin ratkaista mainitulla opetusmenetelmällä, jossa uusia asioita tuodaan vanhojen tuttujen asioiden lomassa esille. Lisäksi ongelmien tulisi lähteä liikkeelle aivan perusteista ja ne vaikeutuisivat hiljalleen niin, että kunkin opiskelijan olisi mahdollista oppia ohjelmointia omassa tahdissa (Gomes ja Mendes, 2007.)

Ohjelmakoodi on usein suurimmalta osalta siinä määrin abstraktia, että opettelijan on vaikeaa hahmottaa mitä tietty rivi ohjelmassa tekee ja miten se vaikuttaa ohjelman suoritukseen ja sen tilaan (Ala-Mutka, 2004). Jos tietty koodipalanen voidaan liittää johonkin konkreettiseen tietokoneen ruudulla tapahtuvaan asiaan niin, että se visualisoi suoraan koodin suorittamaa toimintaa, voidaan koodin ymmärtämistä parantaa (Kuljis ja Baldwin, 2000). Visualisointikeinoja on kehitetty useilla eri keinoilla, kuten siihen erityisesti kehitetyistä työkaluista erilaisiin ohjelmointiympäristöihin, kuten Scratch (Resnick ja muut, 2009).

Naps ja muut (2002) jaottelevat visualisoinnin kuuteen tasoon: piilotettuun, katselemiseen, vastaamiseen, muuttamiseen, rakentamiseen, ja esittämiseen. Ensimmäisellä piilotetulla tasolla visualisointia ei ole lainkaan, ja vasta katselemistasolla käyttäjä voi nähdä ohjelman suorituksen askel askeleelta. Vastaamistasolla ohjelma esittää käyttäjälle kysymyksiä näytetystä ohjelmakoodista. Muuttamisella tarkoitetaan sitä, että käyttäjälle annetaan mahdollisuus muuttaa esitetyn ohjelman käyttäytymistä haluamillaan syötteillä. Rakentamistasolla käyttäjä saa rakentaa omia ohjelmiaan mahdollisesti tietyn tehtävän mukaisesti ja nähdä niiden suoritus visualisoituna. Esittämistasolla visualisointi voidaan esittää kokonaiselle yleisölle, jolloin sitä voidaan kommentoida kollektiivisesti. (Naps ja muut, 2002.)

Robins ja muut (2003) ehdottavat, että ohjelmoinnin opetuksen ei tulisi sisältää pelkästään ohjelmointikielen rakenteita vaan myös esitellä, kuinka niitä voidaan yhdistellä ongelmia ratkaistaessa. Tämän lisäksi he suosittelevat ohjelmien suunnitteluperiaatteiden esille tuomista ja pitävät erinäisten käsitteiden havainnollistamista tärkeänä, sillä ilman sitä oppilaat luovat omat ja usein väärät käsityksensä opetettavista asioista.

Tietokoneohjelmien ymmärrystä voidaan edesauttaa niin sanotulla *kuvailevalla koneella* (notional machine). Kuvaileva kone on kuin idealisoitu tietokoneen malli, joka on tehty mahdollisimman yksinkertaiseksi, ja sen avulla oppilaan tulisi kyetä tarkastelemaan tietokoneohjelman käyttäytymistä mahdollisimman selkeästi. (Du Boulay, O'Shea ja Monk, 1981.) Ohjelmoinnin oppimisympäristöön kuuluu useita ominaisuuksia, joihin kuuluu mainitun kuvailevan koneen lisäksi ohjelman kontrollivuon näkyväksi tekevä graafinen ympäristö, animoidut ohjelman muutokset ilman piilotettuja toimintoja, ja hiljattainen tukien ja rajoitusten vähentäminen (Spohrer ja Soloway, 1986).

Labroratoriossa tehdyt ohjelmointitehtävät ovat eräs hyvä tapa oppia ohjelmointia, sillä ne antavat oppilaille mahdollisuuden edetä omaan tahtiin. Lisäksi helposti saatavilla olevat tehtävät sekä graafinen ja animoitu ohjelman lopputulos voivat olla motivoivia tekijöitä. (Robins ja muut, 2003.) Usein itserakennetun ohjelman toimivuus on myönteinen rohkaisu jatkaa tehtävien tekoa (Linn ja Dalbey, 1989). Myös yhdessä muiden oppilaiden kanssa tehdyt ohjelmat voivat olla hyödyllisiä vertaistuen ansiosta (Van Gorp ja Grissom, 2001). Kuitenkin visuaalisesti dynaamisen ympäristön tulisi korostaa nimenomaan oppilaan tekemän ohjelmakoodin vaikutusta saavutettuihin animaatioihin ja muuhun grafiikkaan (Kurland, Pea, Clement ja Mawby, 1986).

Myös oppilaiden omassakin käyttäytymisessä on eroja siinä tilanteessa, kun he kohtaavat ohjelmoinnissa ongelmatilanteen. On olemassa kahdanlaisia oppilastyyppejä: pysähtyjät ja liikkujat (Perkins, Hancock, Hobbs, Martin ja Simmons, 1986). Ongelmallisessa tilanteessa pysähtyjät menettävät toivonsa asian ratkaisemiseksi, jolloin he lopettavat ja hylkäävät tekemisensä. Liikkujat yrittävät yhä uudelleen muuttaen ohjelmakoodia ja käyttävät virheistä tullutta palautetta hyväkseen. On olemassa myös *superliikkujien* tyyppi, jotka eivät ymmärrä ohjelmaa ja tekevät muutoksia satunnaisiin paikkoihin, ja lopulta pysähtyjien lailla he eivät etene tehtävässä mainittavasti. (Perkins ja muut, 1986.) Oppijatyyppeihin on mahdollista lisätä tehokkaat ja vähemmän tehokkaat aloittelijat. Tehokkaat oppivat asian nopeasti ilman voimakasta panostusta, mutta vähemmän tehokkaat vaativat Robinsin ja muiden mukaan jatkuvaa henkilökohtaista tukea (Robins ja muut, 2003).

Eräs ohjelmoinnin opettamisen pulma liittyy myös siihen, tulisiko aloittelijoille opettaa olio-orientoitunutta vai proseduraalista ohjelmointitapaa. Olio-ohjelmointi lisää kompleksisuutta sen yleisen oletuksen vastaisesti, että oliot olisivat luonnollinen tapa mallintaa oikean maailman ongelmia. Lisäksi olioiden tunnistaminen on usein hankalaa annetusta ongelmakuvauksesta (Robins ja muut, 2003.) Erityisesti sellaiset olio-ohjelmoinnin käsitteet, kuten virtuaalifunktiot ja funktioiden kuormitus, ovat erityisen vaikeita aloittelijoille (Milne ja Rowe, 2002). Proseduraalinen ja olio-orientoitunut ohjelmointi voivat korostaa tiettyjä asioita ja selventää niitä oppilaalle. Verratessa lyhyitä kummallakin ohjelmointityylillä toteutettuja ohjelmia, olio-ohjelmointi auttaa ymmärtämisessä hieman paremmin. Pitkällä koodilla toistettu koe tuotti sen tuloksen, että proseduraalisella tavalla tehdyt ohjelmat olivat helpommin ymmärrettäviä (Wiedenbeck, Ramalingam, Sarasamma ja Corritore, 1999.)

3.4 Kehitettyjä opetussovelluksia

Ohjelmoinnin oppimista varten on kehitetty lukuisia sovelluksia. Niiden tarjoamat ympäristöt antavat useita etuja verrattuna raa'an koodin kirjoittamiseen ammattilaiskäyttöön suunnatulla kehitysympäristöllä. Ne voivat antaa konkreettisen vasteen muuten abstraktille koodille, voivat poistaa syntaksin tuomat ohjelmoinnin mekaaniset hankaluudet ja antavat oppilaalle palautetta välittömästi koodin kirjoittamisen jälkeen (Maloney, Peppler, Kafai, Resnick ja Rusk, 2008.) Parhaimmillaan ympäristöt lievittävät myös muita ohjelmointiin liitettyjä ongelmia, joita on luonteeltaan mm. sosiaalisia, johon kuuluvat ohjelmoinnin nörttimäisyyden ja muista ihmisistä eristäytyneisyyden maine, sekä motivationaalisia, johon kuuluu esimerkiksi tunne ohjelmoinnin vaikeudesta (Gomes ja Mendes, 2007).

Koodin visualisointia varten on toteutettu useita sovelluksia, joista esimerkiksi Teaching Machine (Bruce-Lockhart ja Norvell, 2000) visualisoi ohjelmaa rivi riviltä näyttäen ohjelmassa tapahtuvat muutokset. PlanAni (Sajaniemi ja Kuittinen, 2003) näyttää miten ohjelman muuttujat toimivat. Elenbogen, Maxim ja McDonald (2002) kehittivät C++-kieltä varten visualisointiohjelman, jossa käyttäjät voivat muuttaa pieniä koodinpalasia ja nähdä muutosten vaikutukset ohjelman käyttäytymiseen.

Kehittyneemmistä oppimisympäristöistä esimerkkinä on Scratch (Resnick ja muut, 2009), jota voidaan käyttää suoraan verkkoselaimen avulla. Scratchin avulla voidaan graafisesti luoda animaatioita, pelejä ja muuta interaktiivista multimediaa. Scratchissa ohjelmointi tapahtuu vetämällä ja yhdistelemällä palapelimäisiä ohjelmointilausekkeita. Lausekkeet ovat samoja kuin tekstipohjaisessakin ohjelmoinnissa, mutta Scratchissa ei voi tapahtua syntaksivirheistä, sillä palaset eivät mene yhteen kuin tiettyjen muiden palasten kanssa. Scratchin kehityksessä on painotettu mahdollisuutta “värkätä” (tinker)

projektien kanssa niiden ollessa samalla tarkoituksellisia käyttäjilleen. Myös sosiaalinen osa ohjelmoinnista on otettu huomioon niin, että käyttäjät voivat kehittää vapaasti toisten luomuksia eteenpäin ja kommentoida muiden töitä. (Resnick ja muut, 2009.) On kuitenkin näyttöä, että esimerkiksi Scratch ei tue käyttäjiensä luottamusta omiin ohjelmointitaitoihin, eikä sitä mielletä ohjelmoinniksi verratessa sitä toiseen ympäristöön, Logoon, jossa helpotettua ohjelmointikieltä kirjoitetaan itse (Lewis ja muut, 2014).

Christian ja Mathrani (2014) tutkivat opiskelijoiden ohjelmoinnin oppimista pelien avulla. Mukaan otettiin ohjelmoinnin peruskäsitteet, joita ovat koodin sekvenssittäinen suorittamisen ymmärtäminen, ehtorakenteet, toistorakenteet, funktiot ja rekursio. Tutkimukseen osallistui opiskelijoita, jotka jaettiin kahteen ryhmään: ohjelmointia osaaviin ja vasta-alkajiin. He pelasivat Light Bot -nimistä ohjelmointipeliä, jossa esiintyvät tutkitut peruskäsitteet. Ennen pelaamista kummallekin ryhmälle tehtiin ohjelmoinnin osaamista mittaava testi, jossa annettuihin ohjelmakoodin pätkiin tuli määritellä niiden lopputulos.

Pelaamisen jälkeen opiskelijat täyttivät kyselyn, joka sisälsi sekä kvalitatiivisen että kvantitatiivisen osuuden. Kummassakin osuudessa kysyttiin opiskelijoiden mielikuvaa pelin hauskuuden elementeistä ja pelin hankalista osista. Vasta-alkajien ryhmälle esitettiin tämän lisäksi kysymyksiä myös varsinaisesta ohjelmoinnin osaamisesta pelin jälkeen. Kummatkin ryhmät antoivat myönteistä palautetta pelaamisesta. Kummankin ryhmän mielestä pelaaminen oli tehokas ja tukeva tapa oppia ohjelmointia, mikä tukee ajatusta, että pelaaminen voi olla hyödyllistä vasta-alkajille ja heille, joilla on jo kokemusta opetetusta aiheesta. Vasta-alkajien ryhmän oppiminen oli kuitenkin sekalaista, sillä 12 osallistujaa 20:stä vastasi oikein kyselyn ohjelmointikysymyksiin, ja 20 % oli sitä mieltä, että pelaaminen oli tylsää tai ei opettanut ohjelmointia. Myönteiset vastaukset antoivat tietää, että pelaaminen mm. oli hälventänyt ohjelmoinnin oppimisen pelkoa. (Christian ja Mathrani, 2014.)

Bromwich, Masoodian ja Rogers (2012) kehittivät Light Botin idean pohjalta pelin, jolla yritettiin tutkia 13–16-vuotiaiden ohjelmoinnin oppimista. Pelissä oli ajatuksena opettaa ohjelmoinnin peruskäsitteisiin kuuluvia toisto- ja ehtorakenteita, ja erona Light Botiin on se, että ohjelmointirakenteet tuodaan graafisesti eri tavalla esille. Pelissä ohjataan hahmoa kentästä toiseen ja yritetään saada se saavuttamaan kunkin kentän maali käyttäen annettuja liikkumiskäskyjä, joita voidaan ketjuttaa ja laittaa esiteltyjen ohjelmointirakenteiden sisään. Kaikki ohjelmointi tapahtuu visuaalisesti vetämällä rakenteita ja käskyjä omalle alueelleen, jossa niitä voidaan yhdistellä. Lopulta rakennettu "koodi" voidaan suorittaa ja katsoa kuinka hahmo suoriutuu. Tutkimuksessa osallistujat pelasivat peliä 45 minuuttia, mikä tutkijoiden mukaan oli riittävä pelin arvioimiseen. Tutkijat eivät käyttäneet videointia tai äänitystä, vaan kirjasivat ylös havaintojaan tutkimuksen aikana. Tutkijoiden mukaan osallistujat olivat koko tutkimuksen ajan kiinnostuneita pelistä, mikä tutkijoiden mukaan viittasi pelillisyyden olleen onnistunut. Palkkiojärjestelmä oli heidän havaintojensa mukaan erityisen motivoiva tekijä. Lopulta raportista ei käynyt ilmi, miten peli opetti ohjelmointikäsitteitä. (Bromwich ja muut, 2012.)

Toisessa tutkimuksessa (Papastergiou, 2009) tutkittiin 16–17-vuotiailla oppimispelin opetuksen tehokkuutta ja oppimisen motivaatiota. Tutkimuksessa kehitettiin kaksi keskenään vertailtua sovellusta, joista yksi opetti pelillisesti tietokoneen muistiin liittyviä asioita, ja toinen pyrki samaan ilman pelillisyyttä. Peli koostui kolmesta sokkelosta, joissa kussakin pelaajan oli selvitettävä ongelmia, jotka esitettiin hänelle kyllä-ei-kysymyksinä tai monivalintakysymyksinä. Kentät päästään läpi vastaamalla kuhunkin kysymykseen oikein sekä pääsemällä kenttiin asetettujen muunlaistenkin esteiden ohitse. Kysymyksiin

vastaamalla sai välittömän palautteen vastauksesta. Kentissä oli kohtia, joissa pelaaja sai lukea tietokoneen muistiin liittyvää oppimismateriaalia ja joihin kysymykset perustuivat. Tutkimuksessa käytettiin kahta kyselyä opetettujen asioiden oppimisen mittaamisessa. Alkukyselyssä tiedusteltiin osallistujan taustatietoja sekä etukäteistietoja liittyen tietokoneen muistiin. Jälkikyselyssä kysyttiin alkukyselyn tavoin muistiin liittyviä asioita sekä palautetta pelistä. Tulokset osoittivat, että pelaaminen on tehokkaampi tapa tavalliseen opetusohjelmaan verrattuna sekä opitun tiedon määrän että motivaation suhteen. (Papastergiou, 2009.)

3.5 Yhteenveto

Aloittevan ohjelmoijan eräs merkittävimmistä eroista kokeneisiin nähden on ongelmien ratkaisemista varten kehitettyjen ratkaisumallien määrä (Lahtinen ja muut, 2005) ja järjestäytyneempi tapa säilyttää tietoa muistissa. Ohjelmointipeli voitaisiin suunnitella niin, että siinä tarjotaan erilaisia ongelmia ja yleisiä ratkaisumalleja niihin. Toisaalta täysin aloittelijoille suunnattu peli ei voi sisältää kovin monimutkaisia ongelmia, joten ne tulisi liittää ohjelmoinnin peruskäsitteiden ympärille.

Vaikka ohjelmointikielten syntaksin oppiminen ei yleensä ole hankalin osa ohjelmoinnin oppimista (Winslow, 1996), on esimerkiksi toistorakenteiden ymmärtäminen hankalaa piilossa tapahtuvien asioiden vuoksi (Du Boulay, 1986). Tämän vuoksi toteutettavan ohjelmointipelin kuuluisi jollain tavalla visualisoida tai antaa palautetta pelaajan koodista ja sen suorittamisesta. Muitakin ongelmia esiintyy aloittelijoiden keskuudessa, ja ne liittyvät ohjelmien suunnittelun tai suuremman mittakaavan ymmärtämisen puuttumiseen (Winslow, 1996). Myös erityisesti ohjelmoinnin ongelmanratkaisu (Ala-Mutka, 2004) ja ohjelmointilausekkeiden yhdistely (Winslow, 1996) tuottavat ongelmia. Näitä hankaluuksia voitaisiin peleissä helpottaa aloittamalla yksinkertaisista ja maanläheisistä ongelmista, jotka on pelaajan helppo ymmärtää, ja tuomalla uusia asioita sopivalla tahdilla esille liitettynä vanhaan opittuun.

Ohjelmoinnin oppiminen voi olla syntaksin oppimista, ohjelmien suunnnittelemista ja luetun koodin ymmärtämistä (Ala-Mutka, 2004), ja ohjelmoinnin osaaminen voidaan jakaa ulkoa muistettavaan ohjelmointitietämykseen ja soveltavaan ohjelmointistrategiaan (Davies, 1993). Pelien avulla syntaksin opettelu ja soveltaminen voidaan sulavasti yhdistää keskenään niin, että käsitteet opetetaan oppilaalle soveltavien esimerkkien avulla. Toisaalta soveltaminen ei saa olla liian haastavaa aluksi, vaan jokainen käsite on tuotava mahdollisimman yksinkertaisella, vaikkakin käsitettä hyvin kuvaavalla tavalla esille. Peleissä eteneminen tarjoaa myös mahdollisuuden yhdistellä ja soveltaa aiemmin opittuja asioita uusiin asioihin. Peliin voidaan laittaa valmiiksi koodia, jota pelaajan on luettava ja ymmärrettävä, jotta hän voi muuttaa koodia esimerkiksi tehtävän läpäisyä varten.

Koodi voi olla aloittelijoille abstraktia ja ohjelman hahmottamisessa voi ilmetä hankaluuksia (Ala-Mutka, 2004), mutta koodin visualisoinnin avulla voidaan konkretisoida muuten abstraktin oloista koodia (Kuljis ja Baldwin, 2000). Ohjelmointipelin suunnittelussa tämä tarkoittaa visualisoinnin rakentamista sulavasti muuhun peliin, tai pelimäinen ulkomuoto voi saada siitä lomman. Peleissä voidaan käyttää esimerkiksi yleistä dialogia tai muuten pelimekaniikkaan, ja sitä kautta muuhun pelimaailmaan sopivaa tapaa visualisoinnissa. Visualisoinnin tasoja on useita lähtien olemattomasta visualisoinnista päätyen sellaiseen visualisointiin, johon kokonainen yleisö voi ottaa kantaa (Naps ja muut, 2002). Tätä jakoa tarkastellen ohjelmointipelissä olisi hyvä antaa pelaajan tehdä itse koodia, joten visualisointi päätynee toteutettavassa pelissä rakentamistasolle.

Myös kuvailevan koneen (Du Boulay, 1986) kehittäminen opetusta varten on suositeltavaa, sillä sen avulla oppilaat pääsevät tarkastelemaan ohjelmia yksinkertaistetun ohjelmointikielen avulla, ja ohjelmointipeliin voidaan suunnitella pelaajien osaamistason mukainen kuvaileva kone. Tässä tapauksessa täytyy lähteä ohjelmoinnin alkeista, missä auttaa Christianin ja Mathranin (2014) antama listaus ohjelmoinnin peruskäsitteistä. Robinsin ja muiden (2003) mainitsema ohjelmointitehtävien saatavuus ja visuaalinen ohjelmointiympäristö ovat myös peleille ominaisia, sillä ne ovat usein jollain tavalla graafisia ja tarjoavat tehtäviä jopa loputtoman määrän.

On väittelyä siitä, onko proseduraalinen vai olio-ohjelmointi parempi tapa ohjelmoinnin opettamisessa. Olio-ohjelmointi lisää kompleksisuutta (Robins ja muut, 2003) ja jotkut siihen liittyvät käsitteet voivat olla vaikeita (Milne ja Rowe, 2002). Kuitenkin lyhyet ohjelmat voivat olla helpompia ymmärtää olio-ohjelmoinnin kautta, kun taas pidempien ohjelmien kannalta proseduraalinen on parempi vaihtoehto (Wiedenbeck ja muut, 1999). Toisaalta proseduraalinen ohjelmointi pitää sisällään ohjelmoinnin peruskäsitteet, joita tarvitaan myös olio-ohjelmoinnissa. Tämän vuoksi aloittelijoille suunnatun ohjelmointipeli olisi suositeltavaa opettaa proseduraalista ohjelmointia, minkä jälkeen voidaan aloittaa olio-ohjelmoinnin perusteiden opettaminen.

4. Tutkimusmenetelmä

Tutkimusongelma toimii empiirisessä tutkimuksessa perustana, joka on mahdollista hajottaa tutkimuskysymyksiin (Hirsjärvi ja Hurme, 2008). Tämän tutkimuksen ongelmana on tutkia, millaisella pelillä kannattaa opettaa ohjelmoinnin peruskäsitteitä yläkouluikäisille. Tätä tutkittiin konstruktiivisella tutkimusotteella toteuttamalla näitä käsitteitä opettava tietokonepeli. Pelin suunnitteluperiaatteet kerättiin kirjallisuudesta ja pääteemoiksi niille muodostuivat motivaatiotekijät, pelien avulla oppiminen ja yleiset peleissä esiintyvät ulottuvuudet. Ohjelmoinnin piiriin kuuluneet suunnitteluperiaatteet sisälsivät keinoja parantaa ohjelmoinnin oppimista huomioimalla sekä oppimisen että opettamisen näkökulmat.

Konstruktion arvioinnissa käytettiin kvalitatiivista tutkimusmenetelmää, jossa hyödynnettiin kyselyitä, muistiinpanoja, tietokoneen ruudunkaappausta ja sanallista tiedonkeruumenetelmää. Kerätty tiedon määrä antoi viitteitä siitä, kykenikö kehitetty peli ohjelmoinnin peruskäsitteiden opettamiseen ollen samalla mieluisa ja ohjelmoinnin mielikuvia ylläpitävä. Näin saatiin tietää, kykenikö pelin taustalla ollut teoreettinen viitekehys ohjaamaan halutun kaltaisen pelin suunnittelua.

Tämä luku jakaantuu konstruktiivisen tutkimuksen kuvaukseen, jota käytettiin tämän tutkimuksen pääasiallisena tutkimusmenetelmänä. Sitä seuraavassa alaluvussa käydään läpi konstruktion toteutusprosessi, jossa painotetaan suunnitteluperiaatteiden ja erityisesti pelin pilotointien vaikutusta sen lopulliseen muotoon. Kolmannessa alaluvussa esitellään konstruktion arvioinnissa käytetty kvalitatiivinen tutkimusmenetelmä.

4.1 Konstruktiivinen tutkimus

Suunnittelutieteellinen, eli konstruktiivinen tutkimus perustuu tekniikan aloihin, joiden puitteissa tarvitaan usein ratkaisuja käytännön ongelmiin (Hevner ja muut, 2004). Siinä hyödynnetään teoriaa konkreettisten ratkaisujen saavuttamiseen (Järvinen ja Järvinen, 1996) paremmin kuin miten siihen asti on kyetty (Hevner ja muut, 2004). Konstruktiivisessa tutkimuksessa rakennetaan ja arvioidaan jokin innovaatio perustuen havaittuun ongelmaan, ja tämä kyseinen toteutettu innovaatio, eli artefakti, toimii samalla myös tutkimuskohteena (Hevner ja muut, 2004). Loppuun asti toteutetun artefaktin sijasta konstruktio voi olla myös esimerkiksi riittävän kattava suunnitelma artefaktin toteuttamiseksi tai sen prototyyppi (Järvinen ja Järvinen, 1996). Varsinainen tutkimuksen anti voi tulla joko itse artefaktista tai sen pohjalta toteutetun tutkimuksen tuloksista.

Konstruktiivisessa tutkimuksessa toteutettu innovaatio voi joko saavuttaa sille asetetut päämäärät, jäädä niiden alle tai innovaatio voi osoittaa muunlaista hyötyä kuin oli alun perin suunniteltu. Konstruktiivisessa tutkimuksessa toteutetun konstruktion lopputulos ei välttämättä ole tiedossa tutkimuksen alussa, vaan siinä tulisi asettaa tila, joka halutaan artefaktilla saavuttaa. (Järvinen ja Järvinen, 1996.) Tässä tutkielmassa artefaktin tavoitetilaksi asetettiin sellaisen ohjelmointipelin totetuttaminen, joka voisi opettaa yläkouluikäisille ohjelmoinnin peruskäsitteitä. Myös pelin oleminen yleisesti ottaen mieluisa pelaajalle, ja ohjelmoinnin mielikuvien pysyminen vähintään samalla tasolla pelaamisen jälkeen haluttiin asettaa toissijaisiksi tavoitetiloiksi. Syyksi tähän oli, että vaikka peli olisikin opettavainen, sen arvo opetuksessa luultavasti laskisi, jos se tekisi ohjelmoinnin mielikuvia kielteisemmäksi tai peliä olisi ikävä pelata.

Konstruktiivisessa tutkimuksessa tutkittava kohde on kuvailtava riittävän tarkasti (Hevner ja muut, 2004.). Tässä tutkielmassa käytettiin toteutetun ohjelmoinnin

opetuspelin kuvailuun kirjallisuudessa esitettyjä viihteellisten pelien ja opetuspelien suunnittelun, sekä ohjelmoinnin oppimisen ja opettamisen piiriin kuuluvia suunnitteluperiaatteita.

Toteutettu konstruktio tulee myös arvioida, joka voidaan tehdä jonkin artefaktiin liittyvän ominaisuuden perusteella. Konstruktiivisessa tutkimuksessa konstruktio rakentuu tyypillisesti kierros kierrokselta tutkimuksen aikana, ja arvionnin avulla voidaan saada jatkuvaa palautetta konstruktion parantamiseksi (Hevner ja muut, 2004.) Tässä tutkimuksessa käytettiin pilottitestejä konstruktion arvioimiseen ja tehtiin niiden pohjalta tarvittavat muutokset siihen. Tutkimustulosten toistettavuuden varmistamiseksi tutkimuskohde on rajattava ja tutkimuksen konteksti kuvailtava (Hevner ja muut, 2004). Tämän tutkimuksen tutkimuskohteeksi tuli sen aikana toteutettu ohjelmointipeli, joka arvioitiin yläkouluikäisillä käyttäen kvalitatiivisen tutkimuksen menetelmiä.

4.2 Konstruktion toteutusprosessi

Peli sai innoitusta Oulun yliopiston tietojenkäsittelytieteiden laitoksen Projekti II – kurssilta, jossa toteutettiin yläkoululaisille tehtyä seikkailupeliä varten oma kenttäeditori. Tällöin saatiin kosketusta siihen, millaisista peleistä kyseisen ikäiset nuoret voivat pitää. Tämän tutkielman alkaessa pelin yleistä ideaa alettiin muotoilla jo ennen pelin taustalla olleiden suunnitteluperiaatteiden hakua aiemmasta kirjallisuudesta. Ideaan tuli sitä mukaa muutoksia, kun suunnitteluperiaatteita kirjattiin. Loppujen lopuksi pelin toteutuksesta tuli iteratiivista, kuten Hevner ja muut (2004) sanovat tyypillisen konstruktion toteutuksen olevan.

Vaikka pelin toteutuksessa pidettiin alusta asti kiinni suunnitteluperiaatteista, jotka esitellään konstruktion kuvauksen yhteydessä luvussa 5, sen kehittymiseen vaikuttivat myös kaksi pilotointia, mikä on myös konstruktion jatkuvan arvioinnin toteutuksen piirre (Hevner ja muut, 2004). Ensimmäisen pilotoinnin suoritti 17-vuotias koehenkilö, jonka pelaamista seurattiin ja pelaamisen aikana koettuja vaikeuksia kirjattiin ylös.

Tämä pilottikokeilu osoitti, että pelin käyttöliittymä oli sekava. Tässä versiossa pelin käyttöliittynä oli jaettu kolmeen yhtä aikaa näkyvään osaan, joista ensimmäinen osa oli pelimaailmaan näkymä, jossa pelaaja liikkui ja oli interaktiossa sen kanssa. Toinen osa oli koodilaatikko, johon pelaaja kirjoitti koodia tehtävien ratkaisemista varten. Kolmas osa oli opetusmateriaalin näyttänyt näkymä. Kukin käyttöliittymän osa vaati pelaajan huomiota vuorotellen, mikä johti siihen, että peliin asetettua opetusmateriaalia ei luettu, vaan pelaaja keskittyi pelissä etenemiseen

Ensimmäistä pilotointia seurasi käyttöliittymän uudistaminen, jonka tuotoksena siinä oli kaksi yhtä aikaa näkyvää osaa. Ensimmäinen osa sisälsi yhä interaktiivisen pelimaailman näkymän, ja toiseen oli tässä versiossa yhdistetty sekä opetusmateriaali että tehtävät. Myös opetusmateriaalin interaktiivisuus koki muutoksen, sillä aiemmassa versiossa se koostui ainoastaan tekstistä. Pilotoinnin jälkeen siihen lisättiin koodilaatikoita, jotka sisälsivät kuhunkin opetettavaan käsitteeseen liittyviä koodiesimerkkejä. Pelaajille annettiin vapaus muokata ja suorittaa niiden sisältämää koodia, minkä toivottiin rohkaisevan heitä kokeilemaan ohjelmointia konkreettisemmin ja yhdistämään annetun teorian käytännön esimerkkiin.

Peliin tehtiin mainitut muutokset, minkä jälkeen suoritettiin toinen pilotointi, jossa oli osallisena 13-vuotias koehenkilö. Tälläkin kertaa huomattiin, että pelaaja ei lukenut opetusmateriaalia, vaan yritti ensimmäisen pilotoinnin tapaan vain edetä pelissä. Lisäksi

uudesta yhdistetystä tehtävien ja opetusmateriaalin näkymästä oli vaikea löytää haluttua tietoa.

Toisessa pilotoinnissa pilotoitiin myös alku- ja jälkikyselyitä, joiden täyttämisessä osallistujalla ei ilmennyt ongelmia. Niiden sivumääräistä pituutta haluttiin lyhentää laittamalla likert-kysymysten vastaukset vaakasuuntaisesti pystysuuntaisen sijaan. Lisäksi toinen pilotointi vaati yksinkertaistamaan käyttöliittymää vielä pidemmälle, minkä seurauksena käyttöliittymään tuli lopulta vain yksi näkymä. Pelaaja sai opetusmateriaalin ja tehtävät esille ruudulle ilmestyneessä ikkunassa. Muulloin ruudulla näkyi vain pelimaailma. Tämän arvioitiin olevan tarpeeksi sopiva tapa ohjata pelaajan huomiota pelimaailman, tehtävien ja opetusmateriaalin välillä. Lisäksi kaikki opetusmateriaali ei ollut alusta saakka pelaajan luettavissa, vaan se sijoitettiin pelaajien keräämiin ns. "kadonneisiin sivuihin", jotka olivat kentistä löytyviä paperinpalasia. Näin yritettiin varmistaa, että pelaajat todellakin kävisivät materiaalin läpi.

Lopullinen versio myös vähensi opetusmateriaalin määrää tekstin ja koodiesimerkkien osalta. Suurin väheneminen näiden osalta tapahtui "koodivinkeissä", joita näytettiin aina, kun pelaaja klikkasi jotakin koodiriviä tehtävässä tai opetusmateriaalissa. Niiden tehtävänä oli havainnollistaa, kuinka mikin koodirivi toimi, mutta pilotointien aikana koodivinkkejä ei luettu käytännössä lainkaan. Yleisestikin ottaen opetusmateriaalia tehtiin ytimekkäämmäksi ja itse kokeiltavan koodin rooli kasvoi. Ohjelmointitehtävien määrää lisättiin, sillä oppimisen ajateltiin tulevan juuri itsekseen kokeilemisen kautta. Näin ollen "kadonneet sivut" sisälsivät myös mainittujen osien lisäksi esiteltyyn oppimateriaaliin liittyneen tehtävän.

Myös kuvailevaa konetta yksinkertaistettiin aiemmista versioista. Ohjelmointiin yleisesti kuuluva ominaisuus, eli koodin kommentointi otettiin pois, sillä niiden uskottiin kummankin pilotoinnin perusteella hämäävän pelaajaa. Kommenttien tehtävänä on selostaa koodin toimintaa ilman sekaantumista koodin toimintaan, mutta pilotointiin osallistuneet ilmeisesti luulivat niiden olevan osa koodin toiminnallisuutta. Kommentit eivät lopullisessa versiossa näkyneet koodin sisällä, vaan tulivat esille, kun hiiren osoitin vietiin jonkin koodirivin päälle. Joitakin pienempiä yksinkertaistuksia tehtiin myös, joista eräs oli lausekkeen *var* poistaminen muuttujien luomisen aikana. Käytännössä uudistusten piti antaa tietä koodin ymmärtämiselle ilman häiritseviä tekijöitä.

4.3 Kvalitatiivinen tutkimus

Kvalitatiivinen tutkimus on luonteeltaan induktiivista, jossa yksittäisistä tapauksista yritetään saada jotakin ilmiötä yleisesti kuvaava selitys (Hirsjärvi ja Hurme, 2008). Laadullisessa tutkimuksessa on ajatuksena kuvailla ja ymmärtää ilmiötä sen sijaan, että siitä yritettäisiin tehdä päätelmiä tilastollisessa mielessä (Tuomi ja Sarajärvi, 2002). Laadullisen tutkimuksen periaatteena ei ole löytää ilmiötä selittävää lopullista totuutta, vaan pyrkiä käsittämään ihmisten toimintatapoja (Vilkka, 2005).

Aineiston keräämiseen laadullisessa tutkimuksessa käytetään mm. haastatteluita ja kyselyitä ja obvservointia. Haastattelussa tutkimukseen osallistuvat kertovat vastauksensa suullisessa muodossa, kun tass kyselyssä osallistujat vastaavat kysymyksiin annettuihin kaavakkeisiin. (Tuomi ja Sarajärvi, 2002.) Kyselyissä voidaan käyttää sekä suljettuja että avoimia kysymyksiä. Avoimilla kysymyksillä voidaan saada tallennettua osallistujien tunnetiloja paremmin monivalintakysymyksiin verrattuna (Hirsjärvi, Remes ja Sajavaara, 2002). Kyselyn etuna haastatteluun nähden on se, että tutkijan läsnäolo ei pääse niin herkästi vaikuttamaan osallistujien vastauksiin. Kysymysten koostamisvaiheessa on syytä huomata, että ne ovat kriittinen osa tutkimuksen

onnistumista. Kysymysten huono muotoilu on usein syynä tutkimusten tulosten virheisiin, ja kysely voidaan joutua uusimaan virheidensä vuoksi, mutta tätä riskiä voidaan vähentää pilotoimalla kyselyä etukäteen. (Valli, 2001.)

Mainitun kyselyiden edun vuoksi tässä tutkielmassa laadullinen tutkimus suoritettiin suurimmalta osalta kyselyin, mutta vastauksiin haettiin osallistujilta tarkennuksia myös sanallisesti pikahaastatteluita käyttäen, jossa käytiin kyselyssä olleita kysymyksiä läpi ja haettiin niihin osallistujilta mahdollisia lisäkommentteja. Lisäksi tutkimustilannetta observoitiin kirjoittaen ylös havaintoja liittyen konstruktion käyttöön, osallistujien reaktioihin ja kommentteihin. Varsinainen konstruktion käyttö haluttiin myös tallentaa ruudunkaappausohjelmalla, jolloin havainnointi voitiin keskittää tutkimustilanteessa paremmin osallistujiin. Tutkimukseen osallistui kahdeksan oppilasta Oulun Pateniemen koulun kahdeksannelta luokalta, joille lähetettiin ennalta tutkimuslupakaavake (liite A). Aineisto kerättiin kahdella kyselyllä (liitteet B ja C), jotka koostuivat taustatiedoista, ohjelmoinnin mielikuvakysymyksistä, ohjelmointitehtävistä ja pelin mielipidekysymyksistä. Tarkemmin arvioinnista on kerrottu luvussa 6.

5. Konstruktion kuvaus

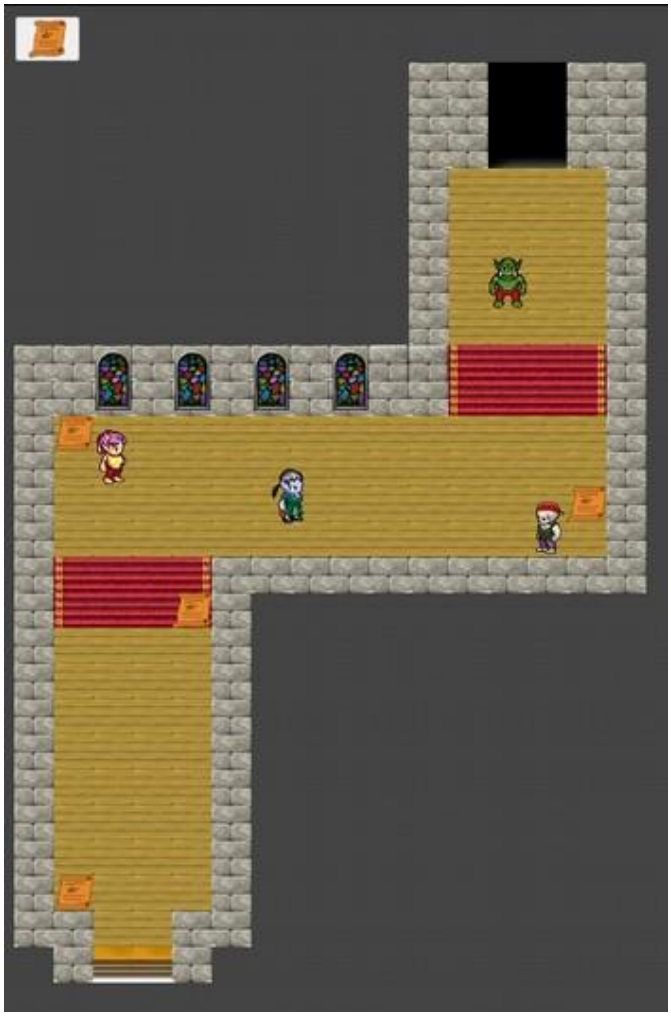
Tässä luvussa kuvataan tätä tutkielmaa varten toteutettu ohjelmointipeli. Toteutus perustuu kirjallisuudessa esiintyneisiin kuvauksiin opetuspelien ja viihteellisten pelien ominaisuuksista. Myös ohjelmointinäkökulma on otettu huomioon erityisesti pelin opetuksellista sisältöä toteuttaessa.

Tämä luku kuvaa niitä asioita, joita aiemmasta kirjallisuudesta on löytynyt peleihin ja ohjelmointiin liittyen, mutta sovellettuna toteutettuun konstruktioon suunnitteluperiaatteiden muodossa. Ensimmäinen alaluku keskittyy yleiseen pelin esittelyyn, mitä seuraa alaluku liittyen pelin ohjelmointisisältöön, kuten siinä esitettyihin ohjelmointikäsitteisiin ja niiden oppimisen helpottamiseksi tehtyihin ratkaisuihin. Sitä seuraavassa alaluvussa esitellään yleisemmin pelin opetuksellisia elementtejä ja sitä, kuinka pelimäisyys pyrittiin yhdistämään niiden kanssa. Sen jälkeisessä alaluvussa esitellään ARCS-mallin (Keller, 1987) strategiat ja flow-teorian soveltaminen pelin toteutuksessa. Viimeinen alaluku käsittelee Garrisin ja muiden (2002) pelien ulottuvuuksien soveltamista pelissä.

5.1 Koodilinnan yleinen kuvaus

Koodilinnaksi nimetty, tässä tutkimuksessa kehitty peli, haluttiin toteuttaa sellaiselle alustalle, joka olisi mahdollisimman laajalti tuettu, jolloin voitaisiin eliminoida yhteensopivuusonglmat sen arvioinnin aikana. Tämän vuoksi peli päätettiin kehittää verkkoselaimessa toimivaksi käyttäen JavaScript-ohjelmointikieltä. Toteutuksessa hyödynnettiin modernien verkkoselainten laajalti tukemaa *HTML5*-teknologiaa ja sen sisältämää grafiikan ja pelien toteuttamiseen käytettyä *canvas*-elementtiä.

Peliä voidaan luonnehtia roolipelimäiseksi fantasiaseikkailuksi, jossa pelaaja voi vapaasti liikkua kentissä ja olla tekemisissä pelimaailman kanssa. Kuva 3 esittää pelin perusnäkymää pelin kolmannesta kentästä. Tytöksi valittu pelihahmo on keskellä kenttää pelin eräiden muiden hahmojen ympäröimänä. Kentästä pääsee oletettavasti ulos ylhäällä olevan oviaukon kautta, mutta vasta sen jälkeen, kun vihreän örkkihahmon antama tehtävä on suoritettu. Eri puolilla näkyy opetusmateriaalia sisältäviä vaaleanruskeita “kadonneita sivuja”, joita pelaaja voi kerätä ja näin saada esille kunkin kentän opetettavaan teemaan liittyvän ohjelmointikäsitteen.



Kuva 3. Pelin perusnäkymä.

Ohjelmointikäsitteet valittiin tukeutuen osittain Christianin ja Mathranin (2014) tutkimuksessa listattuihin ohjelmoinnin peruskäsitteisiin, joista peliin valittiin ehtorakenteet, toistorakenteet ja funktiot. Näiden lisäksi haluttiin siis myös opettaa yleisesti käytetyt muuttujatyypit, eli kirjaimet, kokonaisluvut, desimaalit ja listat. Christian ja Mathrani tarjosivat viitteen, mitä aloittelijoille suunnatut ohjelmointikäsitteet ovat, mutta sen nähtiin olevan vajaavainen juuri muuttujien puuttumisen vuoksi. Käytännössä pelin haluttiin opettavan sellaiset ohjelmointikäsitteet, joiden avulla oppilas voisi ohjelmoida sovelluksia myös pelin ulkopuolella, ja muuttujat ovat ehdottomasti tätä varten tarvittava ohjelmointikäsite. Sekvenssittäisen koodin suorituksen ymmärtäminen oli Christian ja Mathranin (2004) ohjelmointikäsite, mutta tämä tulee selväksi Koodilinnan pelaajalle ilman erillistä opetusta. Lisäksi heidän mainitsemansa rekursio ei sopinut vaikeutensa vuoksi opetettavien ohjelmointikäsitteiden joukkoon.

Kenttiä suunniteltiin peliin seitsemän, joissa kussakin käydään läpi joko täysin uusi ohjelmointikäsite tai lisätään aiemmin opetettuun jotakin uutta. Kentissä on aina sivuhahmoja pelaajan hallitseman hahmon lisäksi. Jotkut sivuhahmoista antavat tietoa maailmasta ja pelin juonesta, ja jotkut antavat päätehtävän, joka liittyy kunkin kentän ohjelmointikäsitteeseen. Päätehtäviä on kussakin kentässä useita ja ne on päästävä läpi ennen kuin pelaaja voi edetä seuraavaan kenttään. “Kadonneista sivuista” löytyy teoreettisen opetusmateriaalin lisäksi sivutehtävä, jotka liittyvät suoraan esiteltyyn teoriaan. Näin ollen pelissä on tehtäviä likipitäen 40, joista noin 20 on sivutehtäviä.

Esimerkki pelissä esiintyvästä päätehtävästä on kolmannessa kentässä olevan sivuhahmon ongelma, joka liittyy kadonneeseen taikajauhetta sisältäneeseen pussiin. Pelaajan täytyy tehdä uutta taikajauhetta hahmolle kirjoittamalla koodi, jossa on jauheen ainesten määrä. Tehtävänannossa on kerrottu sanallisesti kuinka paljon kutakin ainesta tulee, joten pelaajan on itse sovellettava sekä kyseisessä että aiemmissa kentissä opittuja muuttujien ominaisuuksia, jotta ainesten määrä olisi oikein. Tehtävästä saa palkinnoksi kyvyn ymmärtää kentän oviaukolla olevan vihreän örkkihahmon puhetta, mikä mahdollistaa kentän toisen päätehtävän suorittamisen.

Tehtävien ja opetusmateriaalin yhteydessä ilmestyy pelaajan ruudulle koodilaatikko, johon täytyy koko ratkaisu kirjoittaa itse tai jo valmiina olevaa koodia voi muokata ja kokeilla. Tätä koodia pelaaja voi muokata haluamallaan tavalla. Kun pelaaja on mielestään saanut tehtävän ratkaisevan koodin kirjoitettua, voi hän suorittaa sen ja nähdä tuloksen.

Kuvassa 4 on esimerkki eräästä opetusmateriaalisivusta. Opetusteksti pyrittiin saamaan mahdollisimman lyhyeksi ja ytimekkääksi. Koodilaatikossa annettiin esimerkki kustakin ohjelmointikäsitteestä ja siihen pelaajat saivat tehdä muutoksia ja katsoa, miten ne vaikuttivat annettuun palautteeseen. Oikeassa alakulmassa olevan nuolen klikkaaminen vei opetusmateriaaliin liittyvään tehtävään.

Kuva 4. Kolmannesta kentästä löytyvä opetusmateriaalisivu.

5.2 Koodilinnan ohjelmointisisältö

Koodilinnassa käydään läpi ohjelmoinnin perusteisiin lukeutuvia käsitteitä. Kunkin uuden käsitteen tapauksessa lähdetään liikkeelle peruskoodiesimerkistä, jota pelaajan tarvitsee muokata vain vähän läpäistääkseen tehtävän. Seuraavat tehtävät menevät yhä syvemmälle konseptiin ja lopulta pelaaja saa itse kirjoittaa koko ratkaisun tehtävään. Pelin kolme ensimmäistä kenttää käy läpi kirjain-, kokonaisluku ja desimaalityyppiset muuttujat sekä niiden laskutoimitustapoja. Neljäs kenttä sisältää *if*-ehtorakenteen, jonka alle kuuluvat *else if*- ja *else*-rakenteet. Viidennessä kentässä otetaan esille *while*-toistorakenne, kuudennessa kentässä listatyyppiset muuttujat ja niiden perustoiminnot, kuten elementtien lisääminen ja poistaminen. Lisäksi opetellaan käymään listoja läpi *while*-rakennetta käyttäen. Seitsemäs kenttä esittelee funktioiden eri käyttötapoja, jotka ovat niiden peruskäyttö, parametrit ja palautusarvo.

Muut kuin ohjelmointikäsitteisiin liittyvät pelin suunnitteluperiaatteet ovat näkyvillä taulukossa 1. Kenttien tehtävät pyrittiin suunnittelemaan niin, että ne liittyisivät luontaisesti opetettavaan käsitteeseen. Esimerkiksi *while*-toistorakennetta ensi kertaa esiteltäessä annetaan esimerkkinä kultarahojen nostaminen aarrearkusta. Koska rahojen nostaminen yksitellen ei onnistu, on käytettävä toistorakennetta, joka nostaa kolikot toistamalla erityistä kolikon poimintakäskyä arkussa olevien kolikoiden lukumäärän verran. Pelissä edetessä tuli lähes luonnollisesti yhdisteltyä aiemmin opittuja käsitteitä, joten tämän suunnitteluperiaatteen toteuttaminen ei vaatinut suuria ponnisteluja.

Taulukko 1. Pelissä esiintyvät ohjelmoinnin oppimisen ja opettamisen suunnitteluperiaatteet.

Suunnitteluperiaate	**Pelissä ilmenevä ominaisuus**
Ohjelmointirakenteiden yhdistely (Robins ja muut, 2003)	Tehtävissä yhdistellään aiemmin opittuja käsitteitä.
Ohjelmointikäsitteiden havainnollistaminen (Robins ja muut, 2003)	Tehtävät ja koodiesimerkit toteutetaan niin, että ne sopivat luonnollisella tavalla niiden käyttämiin ohjelmointikäsitteisiin.
Ohjelmoinnin oppimisympäristö: graafisuus (Kurland ja muut, 1986; Spohrer ja Soloway, 1986), kontrollivuon tekeminen näkyväksi, yksinkertainen kuvaileva kone, animoidut ohjelman muutokset, hiljattainen tuen ja rajoitusten väheneminen.	Luodaan graafinen pelimaailma, johon ohjelmoimalla voidaan vaikuttaa. Ei esitellä jokaisen käsitteen kaikkia asioita, vaan niistä oleellisimmat. Annetaan suorittaa koodi milloin vain ja saada palaute.
Ohjelmointitietämys ja -strategiat (Robins ja muut, 2003)	Havainnollistavat koodiesimerkit, soveltavat tehtävät
Ongelmanratkaisuprosessin (Winslow, 1996) mahdollistaminen (ongelman ymmärtäminen, ratkaisun määrittely, kirjoittaminen, testaaminen, viankorjaus)	Soveltavat tehtävät, pelihahmojen palaute koodin suorituksesta
Koodin hahmottamisen vaikeus sen abstraktin luonteen vuoksi (Ala-Mutka, 2004)	Vinkit, jotka ilmestyvät osoittaessa jotakin koodiriviä; pelihahmojen palaute koodin suorituksesta; havainnollistavat esimerkit
Proseduraalinen ohjelmointi vai olio-ohjelmointi (Robins ja muut, 2003)	Opetetaan proseduraalisen ohjelmoinnin käsitteitä, sillä ne toimivat lähtökohtana mahdollisesti myöhemmin opetettavassa olio-ohjelmoinnissa.

Luonnollisella ohjelmointikäsitteiden havainnollistamisella on linkki kokemukselliseen oppimiseen, jossa pelaaja mahdollisesti muodostaa kokemuksen pelaamansa pohjalta ja pystyy muistamaan sekä tulevaisuudessa hyödyntämään ohjelmointirakennetta kyseisen kokemukseen ja siihen liittyneen ongelman avulla (Gee, 2008). Pyrkimyksellä luonnolliseen tapaan esitellä käsitteet tehtävissä on myös yhteys Robinsin ja muiden (2003) mainitsemaan käsitteen havainnollistamisen tärkeyteen.

Pelissä yritettiin saada aikaan Winslow'n (1996) kuvaaman ongelmanratkaisuprosessin käyttäminen ongelman ymmärtämisen, ratkaisun määrittämisen ja kirjoittamisen osalta. Koodin testaamista edistää rohkaisu kokeilla ja muuttaa koodia omalla tavalla. Pelaajalle annetaan myös vinkkejä muuttaa koodia tietyllä tavalla ja nähdä mitä sen johdosta tapahtuu. Viankorjausta varten ei peliin erityisesti suunniteltu mitään mekanismia, vaikka pelihahmojen antama palaute koodin vioista saattaa täyttää tämän toiminnon.

Koodin suorittaminen saa aikaan palautteen pelin hahmoilta, mikä antaa tehtävän ratkaisuun lisää neuvoja tai kertoo tehtävän menneen läpi. Tämä on eräs tapa saada pelaaja reflektoimaan tekemisiään (Gee, 2008). Esimerkiksi tehtävässä, jossa pelaaja siivoaa noidan pölyisen varaston, hän saa palautetta, mitä esineitä siellä on vielä jäljellä, jos kaikkea ei tullut koodin suorituksella siivottua.

Koodin hahmotuksen parantamiseksi sijoitettiin tehtävien koodeihin runsaasti niitä selittäviä kommentteja. Tiettyä riviä varten tehty kommentti voitiin saada esille liikuttamalla hiiren osoitin halutun koodirivin päälle. Hahmotuksen parantamiseksi toteutettiin kuhunkin koodiesimerkkiin mahdollisimman kattava palautemekanismi. Pelaajan tehdessä jonkin asian väärin, tuli siitä kuhunkin tilanteeseen räätälöity uniikki virheilmoitus ja vinkki mitä tulisi tehdä, jotta koodi toimisi.

Mitä tulee pelin ohjelmoinnin oppimisympäristöön (Kurland ja muut, 1986; Spohrer ja Soloway, 1986), peliin on liitetty kuvaileva kone niin, että pelissä esiteltyjen käsitteiden kaikkia puolia ei otettu huomioon. Esimerkiksi *while*-toistorakanteen lisäksi ei esitelty *for*-toistorakennetta, joka tekee käytännössä saman asian kuin *while*, mutta on Du Boulayn (1986) mukaan hankala oppia. Pääasia pelissä on saada aikaan ymmärrys siitä, että ohjelmoinnissa on olemassa toistorakenne ja miten sitä voidaan hyödyntää. Kaikkien yksinkertaistettujen rakenteiden osalta käytiin pohdinta mitä niiden osalta voitaisiin jättää pois, jotta ymmärrys kyseisen rakenteen toiminnasta ei jäisi oleellisella tavalla vajavaiseksi.

5.3 Koodilinnan opetuksellisuus

Pelissä on ollut eräänä keskeisenä ajatuksena tuoda yhteen pelimäisyys ja opetussisältö oikeassa suhteessa, jotta peli ei tuntuisi liian puhtaalta peliltä vieden pois huomion oppimiselta eikä myöskään tuntuisi liian opetukselliselta vieden pois ajatuksen vapaaehtoisesta pelaamisesta (Prensky, 2005). Sopivaan pelimäisyyden ja opetuksellisuuden tasapainoon pyrittiin tekemällä pelihahmosta maailmassa vapaasti liikkuva, ja että pelaaja voi tutkia maailmaa rauhassa.

Taulukosta 2 ilmenee mitkä ominaisuudet pelissä vastasivat kutakin haluttua opetuksellista asiaa. Opetuksellisuuden lisäämiseksi kenttien alussa selostetaan mitä uusia ohjelmointikäsitteitä kentässä on ja kehotetaan pelaajaa keräämään niihin liittyvät “kadonneet sivut”, jotka ovat kentistä kerättäviä opetusmateriaalin palasia. Kenttien jälkeen näytetään jälkipohdintana yhteenveto asioista, joita kentän läpäisemiseen vaadittiin. Kentän jälkeisellä yhteenvedolla yritettiin siis saada pelaaja reflektoimaan tekemäänsä ja oppimaansa (Garris ja muut, 2002; Gee, 2008), sekä pyrittiin parantamaan kokemuspohjaista oppimista pelissä. Pelimäisyyttä edistettiin selostamalla juonta sekä alku- että lopputeksteissä.

Taulukko 2. Pelissä käytetyt peleistä oppimisen suunnitteluperiaatteet.

Suunnitteluperiaate	**Pelissä ilmenevä ominaisuus**
Reflektointi: palaute (Garris ja muut, 2002), konkreettiset tavoitteet, sekä pelaajan etenemisen estäminen ennen nykyisen tavoitteen saavuttamista (Paras & Bizzocchi, 2005), jälkipohdinta (Garris ja muut, 2002; Gee, 2008), annetaan pelaajan kirjoittaa omaa koodia (Kiili, 2005), hiljalleen vaikeutuvat kentät (Gee, 2008)	Peli antama palautetetta koodia suorittaessa. Tehdään tehtävien tavoite selväksi dialogissa eikä päästetä pelaaja seuraavaan kenttään ennen kun kaikki tehtävät on suoritettu. Näytetään kentän lopuksi yhteenveto kentän asioista ja tapahtumista. Laitetaan peliin koodilaatikoita esimerkkien ja tehtävien yhteyteen, joihin pelaaja voi kirjoittaa koodia ja suorittaa sen. Vaikeutetaan peliä uusilla ohjelmointikäsitteillä.
Kokemuspohjainen oppiminen: Strukturoidut, tavoitteelliset ja ongelmatilanteen mukaan tulkittavat kokemukset (Gee, 2008)	Luodaan ohjelmointiaiheille niitä vastaavat sopivat ongelmat, kuten kultarahojen ottaminen aarrearkusta toistorakennetta hyödyntäen.
Kokemuspohjainen oppiminen: Tarjotaan palautetta (Gee, 2008)	Näytetään graafisesti tai dialogissa, toimiko pelaajan tekemä ratkaisu ongelmaan. Annetaan palautetta kunkin kentän jälkeen.

Eräs toinen reflektoinnin mahdollistava keino pelissä oli se, että peliin sisällytettiin konkreettiset tavoitteet sille, mitä pelaajan kuuluu tehdä, ennen kuin voi edetä pelissä seuraavalle tasolle. Pelaajan annettiin kirjoittaa pelissä omaa koodia esimerkeissä ja tehtävissä, ja tehtiin pelistä hiljalleen vaikeutuva niin, että tuotiin esille uusia ohjelmointikäsitteitä jokaisessa kentässä.

Kellerin (1987) ARCS-mallia pyrittiin soveltamaan peliä suunniteltaessa ja sen sisältämät strategiat sekä niiden soveltaminen pelissä ovat näkyvillä taulukossa 3. Mallin huomiostrategian osasiksi valittiin humoristiset dialogit hahmojen puheessa ja käytettiin epämuodollista kieltä pelin dialogissa kauttaaltaan. Relevanssistrategiaa sovellettiin niin, että peli pyrittiin saamaan muistuttamaan muita pelejä, joita tutkimuksen kohderyhmä olisi mahdollisesti aiemminkin pelannut. Tähän tavoitteeseen hakeuduttiin toteuttamalla samankaltainen pelimekanismi ja ulkoasu mihin kohderyhmä oli tottunut. Pelin graafinen teema ja pelituntuma tehtiin lähentelemään muita fantasiamaailmoihin sijoittuvia, kuten viihdeyhtiö Nintendon lukuisille alustoille kehitettyjä Pokémon-pelejä (Pokémon Omega Ruby and Pokémon Alpha Sapphire, 2015). Tällä tavalla päästiin mahdollisesti lähentymään kohderyhmää niin, että heille tarjoutuisi parempi mahdollisuus tuntea peli relevantiksi heille itselleen. Merkityksellisyyden tunnetta yritettiin nostattaa myös tuomalla esille viitteitä kohderyhmän suosimista asioista, joita voi löytää mm. verkkomedioista. Kuten sanottu, pelissä käytetty kieli on epämuodollista, jolla pyrittiin karistamaan opetuksellinen sävy. Myös opetukselliset esimerkit tehtiin niin, että ne tuntuisivat hauskoilta ja olisivat lähellä kohderyhmän kiinnostuksen aiheita. Mahdollista sukupuolikohtaista relevanssia yritettiin kohentaa antamalla pelaajien valita pelihahmonsa sukupuoli pelin alussa.

Taulukko 3. Pelissä käytetyt suunnitteluperiaatteet motivaation osalta.

Suunnitteluperiaate	**Pelissä ilmenevä ominaisuus**
Huomiostrategia (Keller, 1987)	Humoristiset dialogit, informaali kieli, pelin ongelmat yleensä
Relevanssistrategia (Keller, 1987)	Populaarikulttuurin viitteitä dialogissa, grafiikoissa, hahmoissa; mahdollisuus valita päähahmon sukupuoli; pelin tuntuma tuttu kohderyhmälle
Itsevarmuusstrategia (Keller, 1987): Koodia selittävät vinkit, annetaan kirjoittaa koodia vapaasti, tehdään ongelmat sopivan haastaviksi, ja annetaan pelaajan itse keksiä ratkaisu tehtäviin (Linn ja Dalbey, 1989)	Näytetään tiettyä koodiriviä selittävä laatikko, kun pelaaja vie osoittimen sen päälle. Lisätään peliin koodilaatikoita esimerkkeihin ja tehtäviin, joihin pelaaja voi kirjoittaa koodia. Pitäydytään esitellyissä aiheissa, jolloin ei pakoteta pelaajaa soveltamaan ratkaisuja aiheiden ulkopuolelta. Tehdään ongelmat mahdollisimman avoimiksi, joihin ratkaisu kuuluu tehdä itse.
Tyytyväisyysstrategia (Keller, 1987)	Esitellään uusi asia ja annetaan siihen liittyvä ongelma. Hyödynnetään vanhaa opittua uusissa tehtävissä, kuten muuttujien käyttöä ehtorakenteiden kanssa.
Flow-tila (Nakamura ja Csikszentmihalyi, 2002): Sopiva ja hiljalleen vaikeutuva haastavuus kentissä (Garris ja muut, 2002)	Vaikeutetaan peliä uusilla ohjelmointikäsitteillä.

Itsevarmuusstrategiaa sovellettiin peliin tekemällä tutustumisesta ohjelmointiin mahdollisimman helppoa lähtemällä liikkeelle aivan perusteista kunkin ohjelmointikäsitteen kohdalla. Pelissä annetun tuen määrän tulisi hiljalleen vähentyä vanhojen ohjelmointikäsitteiden osalta (Spohrer ja Soloway, 1986) ja tämä tulee esille pelissä niin, että vain uusimpien käsitteiden käyttöä neuvotaan opetusmateriaalissa tarkemmin. Ohjelmointikäsitteitä opettavia tekstejä havainnollistetaan käytännön koodiesimerkein ja tehtäviä pelaaja sai ratkaista omalla tavallaan itse, mikä on eräs itsevarmuutta lisäävä tekijä (Linn ja Dalbey, 1989). Itsevarmuutta pyrittiin lisäämään myös sillä, että pelaajan liikuttaessa hiiren osoittimen lähes minkä tahansa pelissä annetun valmiin koodirivin päälle, saa hän tarkemman selityksen kyseisen koodirivin toiminnasta. Eri pelihahmot antavat ohjeita tehtävien ratkaisuun joko heille puhuttaessa, tai tehtävän koodia suoritettaessa. Lopuksi itsevarmuutta pyrittiin kannattelemaan pitämällä tehtävien vaikeusaste kohtuullisena. Pelin sopiva vaikeusaste ja hiljattainen vaikeutuminen ovat Garrisin ja muiden (2002) mukaan myös flow-tilaa edistävä tekijä, joten vaikeuden säätämisellä pyrittiin saamaan aikaan lisäksi tämä vaikutus.

Tyytyväisyysstrategian soveltamiseksi esiteltiin kentän alussa uusi käsite ja annettiin siihen liittyvä tehtävä, jolloin pelaaja pääsee heti kokeilemaan oppimaansa. Edellisten kenttien käsitteitä yritettiin silti pitää näytillä koko pelin ajan uusissa tehtävissä, jolloin mahdollisesti ajatus käsitteiden yhdistelystä koodissa tulee myös selville pelaajille. Esimerkiksi muuttujia käytettiin ehtorakenteiden sisällä senkin kentän jälkeen, kun muuttujat oli jo varsinaisesti opetusmateriaalin osalta käyty läpi. Tämä ei ole pelkästään eräs haasteen lisäämiseksi mainittu keino (Gee, 2008), vaan Robins ja muut (2003) mainitsevat yhdistelyn myös eräänä ohjelmoinnin opettamisen seikkana.

Flow-tilan saavuttamiseen pyrittiin lähinnä sopivalla vaikeusasteella, jonka oli tarkoitus vaikeutua hiljalleen lähinnä uusien ohjelmointikäsitteiden esittelyn myötä. Sopivaan vaikeustasoon pyrittiin käyttämällä aiemmin esiteltyjä ohjelmointikäsitteitä seuraavissa kentissä (Gee, 2008). Myös yksinkertaistettu kuvaileva kone (Du Boulay ja muut, 1981) oli tärkeässä roolissa haasteellisuuden säätämisessä, minkä tukena myös pelille tehdyt pilotoinnit toimivat.

5.4 Koodilinnan yleiset pelien ulottuvuudet

Taulukossa 4 nähdään pelien yleiset ulottuvuudet (Garris ja muut, 2002) ja niihin kuuluvat pelissä ilmenevät ominaisuudet. Fantasian tunnetta pyrittiin nostamaan useita keinoja käyttäen, kuten hyödyntämällä piirrettyjä ja miellyttävän oloisia grafiikoita. Peliin kehitettiin myös tarina, joka tässä tapauksessa on se, että päähahmo yrittää päästä linnaan pahan valtiaan juhliin saadakseen tietää mitä hän juonii juhliensa varjolla. Pelissä esiintyvä ohjelmointi ja koodi on selitetty pelissä taikana, jolla pelaaja voi tehdä erilaisia asioita pelimaailmassa.

Taulukko 4. Yleiset pelien ulottuvuudet (Garris ja muut, 2002) ja niiden toteutus pelissä.

Suunnitteluperiaate	**Pelissä ilmenevä ominaisuus**
Fantasia (Garris ja muut, 2002)	Käytetään piirrettyjä ja uskottavia grafiikoita, kehitetään peliin tarina, interaktio sivuhahmojen kanssa, yleinen dialogi.
Säännöt (Garris ja muut, 2002)	Selkeä mekaniikka ongelmien ratkaisemiseen; ei rajoiteta tapoja ratkaista ongelmia, mutta ohjataan kussakin kentässä opetettuun keinoon
Haaste (Garris ja muut, 2002): varotaan tekemästä tehtävistä monimutkaisia (Kiili, 2005), käytetään vanhoja käsitteitä tulevaisuudessa (Gee, 2008), tehdään pelin käytettävyydestä hyvä (Kiili, 2005)	Annetaan selvä kuvaus ongelmasta ja sen ratkaisutavasta, mutta jätetään loppu pelaajan pohdittavaksi. Varotaan tehtävien monumutkaisuutta välttämättä liian monen ohjelmointikäsitteen soveltamista yhtä aikaa. Hyödynnetään ongelmissa edeltäneissä kentissä esiteltyjä ohjelmointikäsitteitä. Pidetään käyttöliittymä ja pelin mekanismi yksinkertaisena.
Sensorinen stimulaatio (Garris ja muut, 2002): vetoavat grafiikat ja ja animaatiot (Garris ja muut, 2002; Malone ja Lepper, 1987)	Käytetään miellyttäviä grafiikoita pelissä ja sen käyttöliittymässä, animoidaan hahmojen liike, ja lisätään efektejä kenttien väliin sekä tehtävien erikoistapahtumiin
Mysteeri (Garris ja muut, 2002)	Keskiaikainen pelimaailma, mainitut fantasian osat
Hallinta (Garris ja muut, 2002)	Annetaan pelaajan liikutella pelihahmoa vapaasti, olla interaktiossa pelimaailman kanssa ja ratkaista ongelmia omalla tavalla.

Pelissä esiintyy pelaajan ohjaaman päähahmon lisäksi sivuhahmoja, joille pelaaja voi puhua ja jotka koodivinkkien lisäksi antavat tietoa pelimaailmasta. Nämä fantasiaan kuuluvat pelin ominaisuudet toteuttivat samalla myös mysteerin peliulottuvuutta. Fantasiaan pääosin kuuluva graafinen ulkoasu ulottuu myös sensorisen stimulaation peliulottuvuuteen erinäisissä tehosteissa, joita pelissä esiintyy kenttien vaihdoksissa, tehtävien pääsemisessä läpi ja opetusmateriaalia sisältävien “kadonneiden sivujen” löytämisen yhteydessä. Pelaajan annettiin liikutella hahmoaan vapaasti ja olla

interaktiossa pelimaailman sivuhahmojen kanssa, mikä toteutti ajatusta pelin hallintaulottuvuudesta.

Pelissä tehtävät eivät usein pidä sisällään kovin montaa oikeaa tai järkevää vastausta. Tehtävät vaativat soveltamista, sillä tehtävänanto on annettu tekstimuodossa. Tehtävät on kuitenkin tehty tuntumaan avoimilta ja pelaajaa ohjataan luonnollisesti käyttämään samassa kentässä esiteltyjä käsitteitä tehtävän ratkaisemiseksi. Näin pelaajalle ei tuputeta opetettua tapaa, millä on tavoiteltu pelien sääntöulottuvuutta, jonka mukaan pelin ei tulisi rajoittaa tapoja ratkaista ongelmia (Garris ja muut, 2002).

Haasteulottuvuutta tavoiteltiin antamalla tehtävistä selkeät kuvaukset, mutta niiden ratkaisu jätettiin ohjelmointivinkeistä huolimatta pelaajan pohdittavaksi. Esimerkiksi eräässä tehtävässä ongelman kerrotaan olevan se, ettei linnassa järjestettäviin juhliin mahdu enempää vieraita. Kentässä opastetaan muuttujien käytössä, joten pelaaja voi päätellä, että juhlien vierasmäärää voi säätää muuttujasta niin, että hahmo mahtuu juhliin. Haastetta säädettiin välttämällä liiallista tehtävien monimutkaisuutta (Kiili, 2005) antamalla tarpeeksi ohjeita tehtävän suorittamiseksi ja pitämällä vaikeustaso sopivana.

6. Koodilinnan arviointi

Tässä luvussa esitellään Koodilinnan kvalitatiivinen arviointi. Ensimmäisessä alaluuvssa esitellään konstruktion arvioinnin toteutus, eli suunnitelma, miten kyseisen tilaisuuden oli määrä edetä yleisten järjestelyiden osalta. Erityisesti perustellaan tutkimuksessa käytettyjen kyselyiden sisältö kirjallisuuden avulla. Tätä seuraa alaluvut, joissa esitellään tutkimukseen osallistuneiden vastaukset taustatietojen, ohjelmointitehtävien, ohjelmoinnin mielikuvien, ja pelin mielipiteiden osalta.

6.1 Arvioinnin toteutus

Koodilinnan arviointi päätettiin toteuttaa Oulun Pateniemen koululla, jossa kohdejoukoksi valittiin kahdeksan oppilasta kahdeksannelta luokalta. Valinta tehtiin antamalla luokan opettajalle kehotus valita yleensä tietokoneista kiinnostuneita oppilaita mukaan. Kahta viikkoa ennen tutkimusta oli oppilaiden kotiin lähetetty tutkimuslupakaavake (liite A), jonka avulla oppilaat ja heidän vanhempansa saivat antaa suostumuksensa tutkimukseen osallistumiseen. Tutkimuksessa oppilaat vastasivat alkukyselyyn (liite B) ja jälkikyselyyn (liite C), joiden välissä he pelasivat tutkimusta varten kehitettyä ohjelmointipeliä yksinään.

Koko arviointi tehtiin koululla yhdellä käyntikerralla, jolloin oppilaat jaettiin kahteen ryhmään (A ja B), joissa kummassakin oli neljä oppilasta. A-ryhmässä oli kolme poikaa ja yksi tyttö, B-ryhmässä kaksi poikaa ja kaksi tyttöä. Syy ryhmäjakoon oli tilanteen hallinta, sillä kahdeksan oppilaan ohjaaminen olisi voinut osoittautua liian haastavaksi, ja neljän oppilaan ryhmien arvioitiin olevan sopiva määrä otettavaksi kerralla tutkimukseen. Ryhmille annettiin samat aktiviteetit tehtäväksi ja kummallekin ryhmälle oli varattu aikaa kaksi tuntia, jolloin koko arvioinnin oli määrä kestää yhteensä neljä tuntia. Arviointitilaisuutta ohjaamassa ja valvomassa oli kaksi henkilöä, joista yksi oli tämän tutkielman kirjoittaja ja toinen tämän opiskelijaystävä. Valvojien pääasialliseksi tehtäväksi muodostui ohjeistaminen aktiviteettien suorittamisessa.

Ensimmäiseksi ryhmille annettiin täytettäväksi alkukysely (liite B), jonka avulla kerättiin osallistujien taustatietoja, kuten ikä, sekä kiinnostusta tietokoneita ja pelejä kohtaan. Lisäksi kysyttiin, mitä mieltä osallistujat olivat ohjelmoinnista, millä haluttiin selvittää alkutilaa ohjelmoinnin mielikuvista. Sen jälkeinen osuus koostui kymmenestä ohjelmointitehtävästä, jotka olivat niistä aiheista, joita konstruktiolla pyrittiin opettamaan. Ohjelmointitehtävillä kartoitettiin osallistujien ohjelmoinnin osaamista, jotta kyettäisiin jälkikäteen tietämään, opettiko Koodilinna ohjelmoinnin peruskäsitteitä. Ensimmäiseen kyselyyn käytettiin odotettua enemmän aikaa, sillä osallistujat pohtivat ohjelmointitehtäviä yllättävän pitkään. Näin saatiin kummankin ryhmän tapauksessa kyselyn täyttämiseen käytettyä kokonaisuudessaan optimistisen pitkäksi ajateltu 15 minuutin aika.

Alkukyselyn jälkeen osallistujien annettiin pelata Koodilinnaa, mikä tallennettiin ruudunkaappausohjelmalla, jolloin pelaamista voitiin tarkastella jälkikäteen. Kumpikin tutkija tarkkaili kerrallaan kahden osallistujan pelaamista ja kirjoitti muistiinpanoja liittyen pelaamiseen. Tarkalleen ottaen haluttiin tietää, kuinka hyvin osallistujat pystyivät etenemään pelissä, missä mahdollisia hankaluuksia ilmeni, ja kuinka hyvin peliin upotettua oppimismateriaalia luettiin. Ryhmä A sai pelata peliä tunnin ajan, mutta B-ryhmän tapauksessa aikaa saatiin käyttöön enemmän johtuen koulun oppituntien ajoituksesta. He saivat aikaa pelaamiseen noin tunnin ja 20 minuuttia, eli todellisuudessa B-ryhmän arviointitilaisuus kesti noin kaksi tuntia ja 20 minuuttia.

Pelaamisen jälkeen täytettiin jälkikysely (liite C), jossa kysyttiin samat kysymykset kuin alkukyselyn ohjelmoinnin kiinnostusta tiedustelevassa osassa. Tätä seurasivat alkukyselyä vastaavat ohjelmointitehtävät, joita annettiin tehdä vain siihen saakka, johon he pelissä olivat pääseeet. Esimerkiksi osallistujan päästessä pelissä ehtorakenteita käsittelevään kenttään saakka, annettiin hänen tehdä kyselyssä tehtävät vain ehtorakenteisiin saakka. Ohjelmoinnin mielikuvakysymyksien ja ohjelmointitehtävien vastauksia verrattiin myöhemmin keskenään ja haettiin vastausta sille, olivatko ohjelmoinnin mielikuvat pysyneet vähintään samalla tasolla ja erityisesti oliko peli opettanut ohjelmointikäsitteitä. Kahdeksan osallistujaa pääsi tekemään neljä ensimmäistä tehtävää, kuusi osallistujaa pääsi tehtävään viisi ja kolme osallistujaa tehtävään kuusi. Etenemisvauhti yllätti, sillä pilotointien perusteella osallistujien odotettiin ehtivän vain neljättä tehtävää edustavaan kenttään pelissä. Ohjelmointitehtäviä seurasi vielä osio, jossa tiedusteltiin mielipidettä pelistä sen mukaansatempaavuuden, grafiikoiden, opettavaisuuden ja muiden tekijöiden osalta. Kyselyn tueksi osallistujilta pyrittiin lopuksi tiedustelemaan sanallisesti tarkennuksia ajatuksiinsa ohjelmoinnista, Koodilinnasta ja oppimastaan. Näitä kysymyksiä kysyttiin siksi, että Koodilinnan erääksi tavoitetilaksi oli asetettu sen oleminen mieluisa pelaajille.

Syy sille, miksi tutkimusta ei haluttu suorittaa kokonaan haastatteluna, oli siinä, että vastaajien antamat vastaukset liittyen arvioitavaan konstruktioon olisivat voineet olla haastattelevan tutkijan vuoksi kaunisteltuja. Vastaajia rohkaistiin kertomaan mielipiteensä mahdollisimman rehelliseti sen vuoksi, että kielteisetkin mielipiteet ovat yhtä arvokkaita kuin myönteiset, eivätkä vihastuta konstruktion kehittäjää. Rohkaisulla ja rennolla ilmapiirilla ajateltiin olevan myönteinen vaikutus yläkouluikäisten osallistujien vastausten rehellisyyteen.

Käytetyn kyselyn jako alku- ja jälkikyselyyn perustuu Papastergioun (2009) tutkimukseen, ja varsinainen syy tälle oli se, että kyseisessä tutkimuksessa toteutettiin samankaltainen peli kuin tässä tutkimuksessa, ja myös siinä verrattiin oppimista kahden kyselyn avulla. Papastergioun tutkimuksen kyselyissä tiedusteltiin perustietoja, tietokoneen muistikäsitteiden osaamista ja mielipidettä pelistä. Perustiedoiksi tässä tutkimuksessa valittiin ikä, sukupuoli, nimi ja kokemus pelaamisesta ja ajankäyttö niiden parissa, ja kokemus ohjelmoinnista. Mielipidettä pelistä loppukyselyssä tiedusteltiin lähes samoin kysymyksin kuin Papastergioun tutkimuksessa.

Kyselyyn haluttiin liittää osa, joka tiedustelisi mielikuvia ohjelmoinnista ja tämän osan päätettiin jakautuvan avoimiin kysymyksiin ohjelmoinnin helppoudesta ja vaikeudesta, sekä likert-kysymyksiin halusta opetella ohjelmointia itse, koulussa ja pelien avulla. Näiden kysymysten nähtiin riittävän joko myönteisen tai kielteisen ohjelmoinnin mielikuvan saattamisesta tutkijan tietoon.

Ajatus ohjelmoinnin osaamisen tiedustelusta kyselyn avulla voidaan perustella sekä Papastergioun (2009) että Christianin ja Mathranin (2014) tutkimuksilla, joissa tiedusteltiin tutkittavan asian osaamista kyselyin. Tämän tutkimuksen kyselyiden ohjelmointitehtävissä mukailtiin erityisesti Christianin ja Mathranin käyttämää tapaa mitata ohjelmoinnin osaamista kyselyssä. Heidän tutkimuksessaan annettiin kyselyn tehtävissä ohjelmakoodi, jonka lopputulos tuli päätellä ja oikea vastaus rasitettiin annetuista vaihtoehdoista. Tässä tutkimuksessa käytettiin suurelta osaa Christianin ja Mathranin (2014) tutkimuksen kyselyn tapaisia kysymyksiä, mutta muuttujiin liittyvät kysymykset olivat epäkäytännöllisiä tiedustella “rasti ruutuun”-periaatteella kysymysten yksinkertaisuuden vuoksi. Sen sijaan kyselyyn vastaajien haluttiin kirjoittavan vastauksensa ohjelmakoodina, mikä oli likipitäen yhtä tehokas ja nopea tapa testata muuttujien osaamista kuin vastauksen rastittaminen.

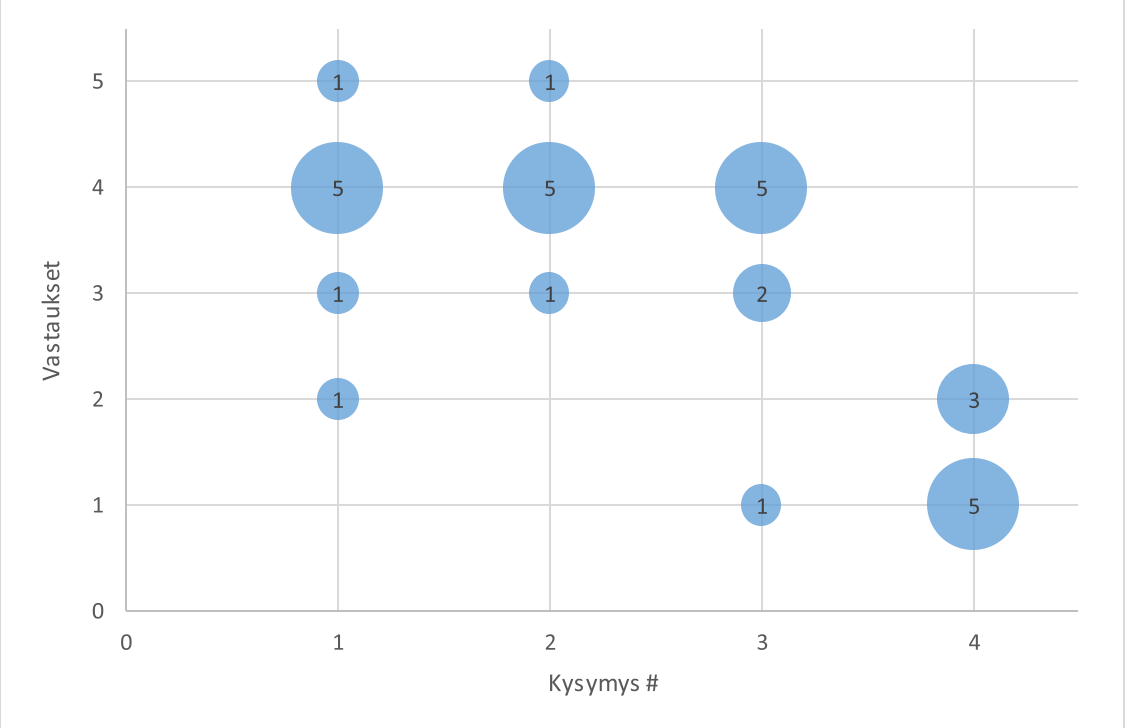
6.2 Taustatietojen vastaukset

Alkukyselyn taustatieto-osuudessa osallistujilta kysyttiin iän lisäksi kiinnostusta tai aktiivisuutta tietokoneiden, pelien ja ohjelmoinnin osalta. Nämä kysymykset on esitetty taulukossa 5. Kysymysten määrä oli kartoittaa kuinka kokeneita tietokoneiden käyttäjiä osallistujat olivat, ja kuinka usein heillä oli tapana pelata tietokonepelejä. Lisäksi kysyttiin arviota osallistujien ohjelmoinnin osaamisesta.

Taulukko 5. Alkukyselyn taustatieto-osuuden kysymykset.

Kysymys #	**Kysymysteksti**
1	Kuinka kiinnostunut olet tietokoneista? (1 = En yhtään, 5 = Todella paljon)
2	Kuinka paljon pidät peleistä? (1 = En yhtään, 5 = Todella paljon)
3	Kuinka usein pelaat pelejä tietokoneella, kännykällä tai nettiselaimella? (1 = En koskaan, 4 = Päivittäin)
4	Kuinka paljon sinulla on kokemusta ohjelmoimisesta tietokoneella? (1 = Ei yhtään, 5 = Todella paljon)

Vastaukset kiinnostusta tiedusteleviin kysymyksiin on esitelty kuvassa 5. Osallistujien keski-ikä oli noin 14 vuotta ja he olivat melko kiinnostuneita tietokoneista. Erityisesti osallistujat olivat kiinnostuneita peleistä, mikä oli verrattain samansuuruista kuin tietokoneisiin kohdistunut kiinnostus. Osallistujat myös pelasivat pelejä useita kertoja viikossa tai päivittäin. Kiinnostus pelejä kohtaan oli myös huomattavissa taustatietojen avoimessa kysymyksessä, jossa kysyttiin, mitä pelejä osallistujat olivat viime aikoina pelanneet. Suurin osa vastaajista antoi useita esimerkkejä peleistä, joiden parissa he ovat viettäneet aikaansa.



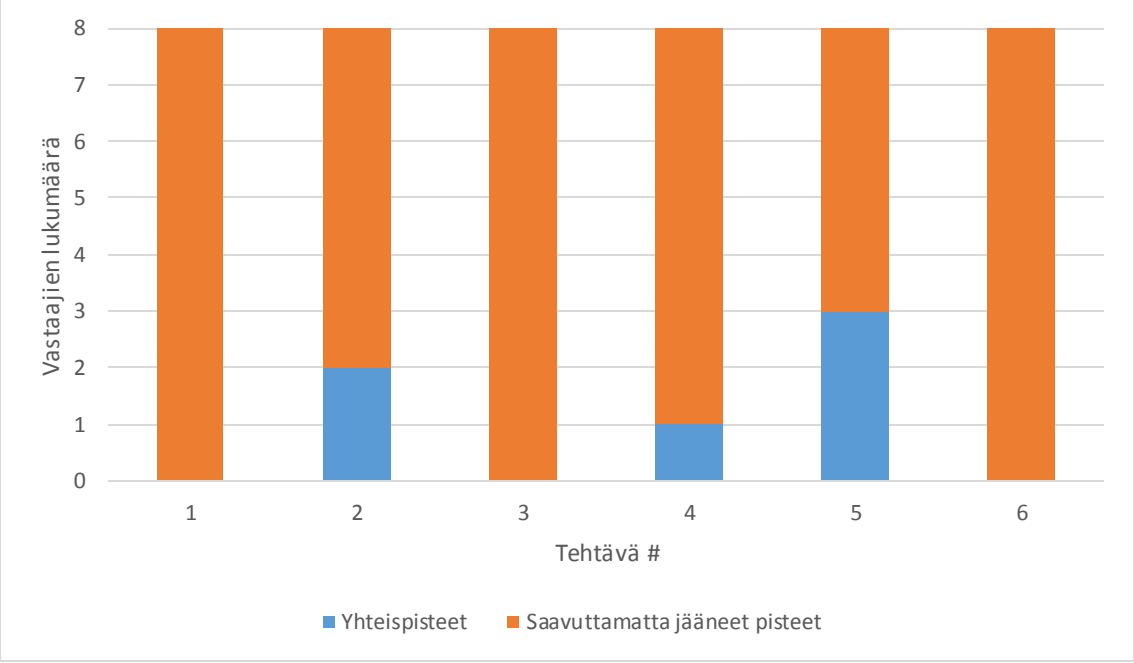
Kuva 5. Alkukyselyn taustatietokysymysten vastausten jakauma. Kysymykset on esitetty taulukossa 5.

Osallistujilla ei ollut paljoa kokemusta ohjelmoinnista. Viisi vastaajaa kertoi, etteivät osanneet ohjelmointia lainkaan. Kolme vastaajaa ilmoitti osaavansa jonkin verran, jolloin he antoivat vastaukseksi 2. Yksi vastaaja tarkensi sanallisessa tiedustelussa, että hänellä on vähän kokemusta Lua-skriptauskielestä. Toinen vastaaja ilmoitti, että hänellä oli jonkinlainen käsitys ohjelmoinnista vanhempansa ansiosta.

Erityisiä huomioita voidaan nostaa esille yhden vastaajan kiinnostuksesta tietokoneisiin, jota tiedustelevaan kysymykseen hän antoi vastauksen 2. Tällä ei silti ollut vaikutusta muihin peleihin tai ohjelmointiin liittyviin vastauksiin, sillä kyseinen osallistuja listasi erityisen runsaasti pelaamiaan pelejä ja pelasi niitä useita kertoja viikossa. Osallistujien joukosta yksi kertoi, ettei pelaa pelejä lainkaan, mutta oli kuitenkin neutraalisti kiinnostunut peleistä vastauksella 3. Sanallisessa vastauksessa hän kertoi pelaavansa Hill Climb Racing –peliä sekä ”joitain satunnaisia pelejä, kuten miinaharava, spiderpasianssi tms.” Osallistuja oli melko kiinnostunut ohjelmoinnista vastauksella 4, ja kertoi osaavansa vähän ohjelmointia vastauksella 2.

6.3 Ohjelmointitehtävien tulokset

Alku- ja jälkikyselyiden ohjelmointitehtävien pisteytyksellä mitattiin kuinka peli opetti siinä esitettyjä ohjelmointikäsitteitä. Kyselyiden ohjelmointitehtävien pisteytys tehtiin niin, että mikäli vastauksesta ilmeni tehtävän ajatuksen täysi ymmärrys, sai vastauksesta yhden pisteen. Jos tehtävästä ilmeni osittainen ymmärrys, sai vastauksesta puolikkaan pisteen. Kuvat 6 ja 7 esittävät osallistujien saamaa yhteispistemäärää tehtävissä ennen ja jälkeen pelaamisen. Koska osallistujia oli kahdeksan, oli maksimipistemäärä kunkin tehtävän kohdalla kahdeksan.

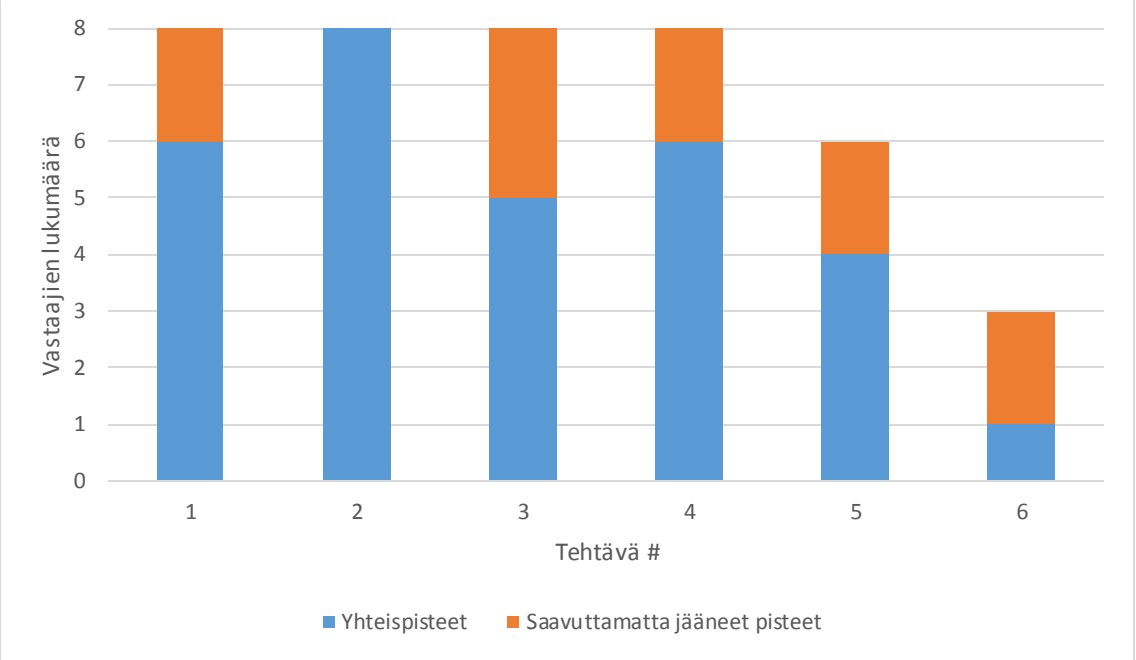


Kuva 6. Alkukyselyn kunkin ohjelmointitehtävän vastauksen saavutetut yhteispisteet.

Osallistujat vastasivat alkukyselyssä kaikkiin kymmeneen annettuun ohjelmointitehtävään. Jälkikyselyssä vastattiin siihen tehtävään saakka, joka vastasi heidän saavuttamaansa kenttää pelissä. Nopeimmat osallistujat pääsivät pelissä tehtävää kuusi vastaavaan kenttään. Ennen pelaamista osallistujien menestys tehtävissä oli heikkoa, sillä vain tehtävistä kaksi, neljä ja viisi saatiin parhaimmassakin tapauksessa alle puolet maksimipistemäärästä. Yhteensä pisteitä kerättiin alkukyselyn

ohjelmointitehtävissä kuusi mahdollisesta 48:sta, mikä vastaa 13 %:n pistesaldoa maksimista. Loppukyselyssä tehtävissä menestyminen parani huomattavasti, sillä saavutettu pistemäärä nousi 30:een maksimipistemäärän ollessa 41, joten yhteensä 73 % pisteistä saatiin kerättyä.

Tarkasteltaessa kuvan 6 pistemäärää ja verrattaessa sitä osallistujien ilmoittamaan keskimääräiseen ohjelmoinnin osaamiseen ennen pelaamista, on mahdollista todeta osallistujien yleisen arvion ohjelmoinnin osaamisesta pitävän paikkansa. Kun tarkastellaan yksittäisten vastaajien ohjelmoinnin osaamisen arviota, yksi vastauksen 1 ja toinen vastauksen 2 antanut osallistuja osasi vastata kahteen ohjelmointitehtävään oikein. Toisaalta kaksi vastauksen 2 antanutta osallistujaa sai kaikista tehtävistä nolla pistettä. Hienoisella ohjelmoinnin osaamisella etukäteen ei siis ollut vaikutusta siihen, kuinka paljon osallistujat saivat pisteitä alkukyselyssä.



Kuva 7. Osallistujien saavuttama yhteispisteytys jälkikyselyn ohjelmointitehtävissä. Viidennen ja kuudennen tehtävän maksimipistemäärä ei ole kahdeksan, sillä kaikki osallistujat eivät päässeet tekemään kyseisiä tehtäviä.

Jälkikyselyn ohjelmointitehtävät olivat identtiset alkukyselyyn verrattuna. Ensimmäiset kolme tehtävää keskittyivät muuttujiin ohjelmoinnissa. Ensimmäisessä tehtävässä kuului tehdä merkkijonotyyppinen, toisessa lukutyyppinen ja kolmannessa desimaalityyppinen muuttuja. Tarkasteltaessa jälkikyselyn ensimmäisen tehtävän vääriä vastauksia huomattiin, että yleinen virhe merkkijonomuuttujia tehdessä on unohtaa lainausmerkit muuttujan arvon ympäriltä. Desimaalitehtävässä ideana oli muistaa laittaa desimaalipilkun sijasta desimaalipiste, mikä on ohjelman toiminnan kannalta elintärkeää.

Kuvan 8 vasen puoli havainnollistaa yleisesti havaittuja vääriä vastauksia ensimmäisen ja kolmannen tehtävän osalta ja oikealla puolella näkyvät niiden oikeat vastaukset. Erityisenä huomiona mainittakoon, että kaikki osallistujat osasivat kuitenkin vastata toiseen kysymykseen oikein, mikä viittää siihen, että osallistujat ymmärsivät, miten muuttujia tehdään ja miten niille annetaan jokin arvo. Voidaan siis väittää, että osallistujat ymmärsivät, mitä muuttujilla tehdään, mutta niiden syntaktinen osaaminen jäi osittain puolitiehen.

elain = tiikeri piirakoitaSyoty = 2,3	elain = “tiikeri” piirakoitaSyoty = 2.3

Kuva 8. Yleisiä jälkikyselyn ohjelmointitehtävissä esiintyneitä virheitä vasemmalla. Oikealla näkyvät oikeat vastaukset.

Neljännessä tehtävässä testattiin ohjelmoinnissa esiintyvien muuttujien avulla tehtävien laskutoimitusten osaamista. Tehtävässä kuului hyödyntää kahta ennalta annettua muuttujaa, jotka havainnollistivat rahan määrää lompakossa ja tilillä. Osallistujien kuului tehdä kolmas muuttuja, joka kuvasi rahan yhteismäärää. Sen arvoksi oli määrä antaa kahden annetun muuttujan summa miinus 15. Suurimmat virheet tämän tehtävän osalta ilmenivät jälkikyselyssä siinä, että tehtävänannosta huolimatta jotkut osallistujat olivat laskeneet muuttujat yhteen päässä ja asettaneet uuden muuttujan arvoksi saamansa tuloksen. Jotkut olivat unohtaneet tehdä pyydetyn vähennyslaskun, jolloin tehtävästä annettiin puolikas piste.

Viidenteen ja kuudenteen tehtävään eivät kaikki osallistujat vastanneet, sillä he eivät ehtineet päästä pelissä niiden aiheita käsitteleviin kenttiin. Viides tehtävä, johon vastasi kuusi osallistujaa, koski ehtorakennetta. Kuudenteen tehtävään vastasi kolme osallistujaa, ja sen aiheena oli *while*-toistorakenne.

Ennen pelaamista kerätty pistesaldo viidennen tehtävän kohdalla juontuu ilmeisesti siitä, että ehtorakennetta on helppo lukea sellaisenaankin, vaikka ei ohjelmointia osaisikaan. Toisin sanoen vastaus on pääteltävissä, mikäli tehtävään keskittyy tarpeeksi. Viidenteen tehtävään oli kolme vastannut alussa oikein, joista kaksi oli saanut vastata tehtävään myös pelaamisen jälkeen, jolloin tehtävä meni myös oikein. Mainittakoon myös, että osallistujilla oli vaikeuksia ymmärtää ehtorakenteita, kun tarkastellaan tallennettua ruudunkaappausmateriaalia. Pelaajat eivät yleisesti osanneet kirjoittaa rakennetta kokonaan loppuun jättäen pois esimerkiksi siinä vaaditut kaarisulkeet. Tähän suhteutettuna viidennen tehtävän pistemäärä on kuitenkin hyvä, kun vertaa sitä esimerkiksi kolmannessa tehtävässä hankittuihin pisteisiin ja enimmäispistemäärään. Ehkä tähän vaikutti juuri se, että ehtorakenteita voi olla helppo tulkita, vaikka ei niitä täysin ymmärtäisikään.

Kuudenteen tehtävään saatiin jälkikyselyssä vastauksia vain kolme, joista yksi oli oikein. Voidaan päätellä, että väärät vastaukset johtuvat siitä, että osallistujat olivat juuri päässeet pelissä tehtävän aihetta käsittelevään kenttään, kun aika loppui. He eivät siis ehtineet kunnolla perehtyä *while*-rakenteen toimintaan.

Jälkikyselyssä saatujen pisteiden määrällä voi olla yhteyttä ilmoitettuun ohjelmoinnin osaamiseen. Yksi vastauksen 2 tähän kysymykseen antanut osallistuja sai keskimääräisen pistetuloksen, mutta kaksi muuta kyseisen vastauksen antanutta saivat kumpikin parhaimmat pisteet tehtävistä.

Loppukyselyssä haluttiin tietää kuinka vaikeita kyselyn ohjelmointitehtävät olivat olleet. Tätä kysyttiin yhdellä likert-kysymyksellä: ”Olivatko edelliset tehtävät helppoja vai vaikeita?” Kysymyksen asteikko oli 1-5, jossa vastaus 1 tarkoitti ”todella vaikeaa” ja 5 ”todella helppoa. Yksi osallistuja ilmoitti vastauksella 5, että tehtävät olivat todella helppoja. Kyseinen osallistuja oli aiemmin kertonut osaavansa Lua-skriptausta ja muutenkin ollut myönteinen ohjelmointia kohtaan. Kolme vastaajaa vastasi tehtävien olleen melko helppoja vastauksella 4, ja kaksi osallistujaa antoi vastauksen 3. Tästä voidaan päätellä, että kyselyn ohjelmointitehtävät olivat keskimäärin helpon oloisia, sillä

vain yksi vastaaja antoi vastaukseksi 2. Vastaajien arvio tehtävien helppoudesta meni tehtävien osallistujakohtaisia pisteytyksiä tarkastellessa käsi kädessä, joten oma arvio ohjelmoinnin osaamisesta oli ilmeisesti tässä tapauksessa tarkka. Vahva arvio omista taidoista voi myös viitata itsevarmaan suhtautumiseen ohjelmoinnista.

Tämän lisäksi kysyttiin kahdella avoimella kysymyksellä tehtävien helppoja ja vaikeita puolia. Mainittu Lua-skriptauskielen osaaja kertoi, että tehtävien ollessa suomenkielisiä totutun englannin sijasta, sen vuoksi hän ”ei tajunnut niitä”. Kolme muuta osallistujaa kertoi tehtävien olleen hankalia. Helpoiksi puoliksi mainittiin muuttujien luominen, asioiden helppo opittavuus ja tuttuus matematiikan tunneilta. Jotkut osallistujat ilmeisesti ajattelivat kysymyksen koskevan Koodilinnan tehtäviä, koska joiltakin osallistujilta oli tullut vastauksissaan viittauksia peliin. Tämän vuoksi voidaan epäillä likert-kysymyksen ja avoimien kysymysten vastausten oikeellisuutta.

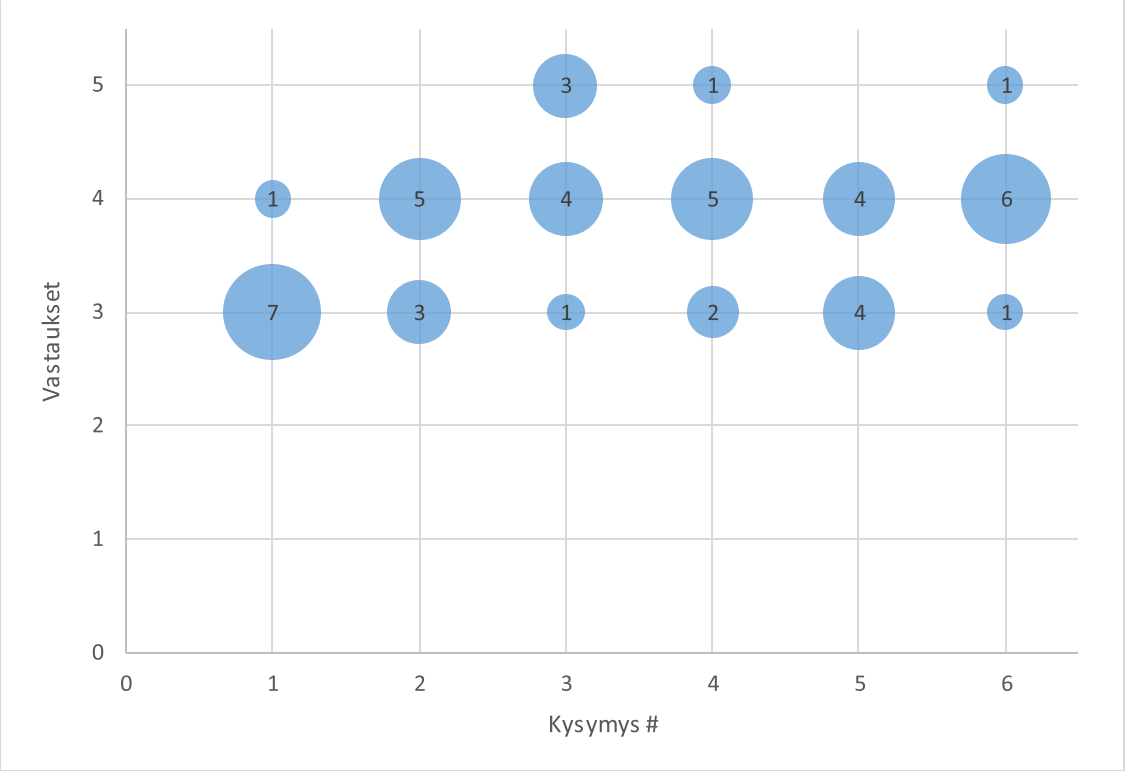
6.4 Vastaukset ohjelmoinnin mielikuvista

Mielikuvia ohjelmoinnista tiedusteltiin kuudella kysymyksellä, jotka on esitetty taulukossa 6. Kysymyksillä pyrittiin saamaan kuva osallistujien kokemasta ohjelmoinnin vaikeudesta, sen hyvistä ja huonoista puolista, ja halusta opetella sitä eri ympäristöissä. Näiden oletettiin riittävän yleisen mielikuvan saamiseen ohjelmoinnista, jolloin voitiin arvioida oliko osallistujien mielikuva muuttunut vai pysynyt samana.

Taulukko 6. Kyselyissä käytetyt ohjelmoinnin mielikuvakysymykset.

Kysymys #	**Kysymysteksti**
1	Onko ohjelmointi sinusta vaikeaa vai helppoa? (1= Todella vaikeaa, 5 = Todella helppoa)
2	Onko ohjelmointi mielestäsi muodikasta tai "siistiä"? (1 = Ei yhtään, 5 = Todella siistiä)
3	Haluaisitko opetella ohjelmointia KOULUSSA? (1 = En yhtään, 5 = Todella paljon)
4	Haluaisitko opetella ohjelmointia ITSE? (1 = En yhtään, 5 = Todella paljon)
5	Olisitko kiinnostunut ohjelmoimaan omia ohjelmia? (1 = En yhtään, 5 = Todella paljon)
6	Haluaisitko opetella ohjelmointia pelejä pelaamalla? (1 = En yhtään, 5 = Todella paljon)

Kuva 9 esittää kysymysten vastausten jakaumaa. Kaikki osallistujat yhtä lukuun lukuun ottamatta vastasivat ohjelmoinnin haasteellisuutta tiedustelevaan kysymykseen 3, mikä voi tarkoittaa neutraalin suhtautumisen lisäksi sitä, ettei osallistujilla ollut vahvaa mielipidettä asiasta. Yksi osallistuja vastasi ohjelmoinnin olevan melko helppoa vastauksella 4, ja kyseinen osallistuja oli taustatiedoissa arvioinut ohjelmoinnin osaamisensa vastauksella 2.



Kuva 9. Alkukyselyn ohjelmoinnin mielikuvakysymysten vastausten jakauma. Kysymykset on esitetty taulukossa 6.

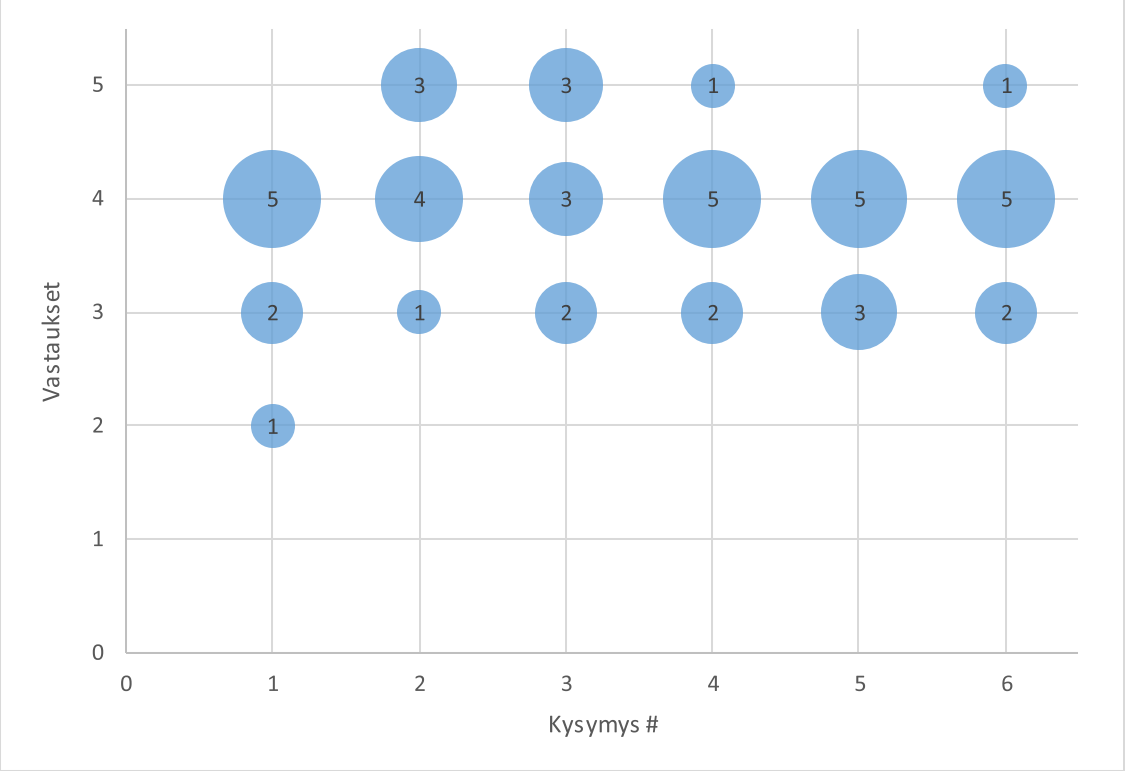
Ohjelmointi käsitettiin yleisesti ottaen muodikkaaksi tai ”siistiksi”, sillä kolme vastaajaa antoi vastaukseksi 3, muut viisi vastasivat sitä korkeammalla arvolla. Osallistujat olivat keskimäärin melko halukkaita opettelemaan ohjelmointia koulussa. Myös kotona ohjelmoinnin opettelemiseen oli kiinnostusta, vaikka peleillä kyseinen kiinnostus oli suurempi. Aiemmin mainittu osallistuja, joka arvioi ohjelmoinnin melko helpoksi, ilmoitti ainoana halunsa opetella ohjelmointia peleillä vastauksella 3. Vaikka osallistujat olivatkin halukkaita opettelemaan ohjelmointia, jäi varsinaisesti omien ohjelmien tekemisen kiinnostus keskimäärin näitä pienemmäksi.

Alkukyselyssä tiedusteltiin osallistujilta hyviä ja huonoja puolia ohjelmoinnista. Hyvin usein hyväksi puoleksi ilmoitettiin, että ohjelmoimalla voi tehdä haluamiaan sovelluksia, pelejä tai ”juttuja”. Useat osallistujat myös vastasivat ohjelmoinnista olevan hyötyä yhteiskunnassa, missä ”kaikki toimivat tietokoneilla” tai se ”liittyy vahvasti nykypäivään ja tulevaisuuteen”. Yksi vastaaja ei osannut mainita hyviä puolia ohjelmoinnista. Ohjelmoinnin huonoksi puoleksi mainittiin usein sen tuntuminen hankalalta tai oppimisen vaikeus. Kolme vastaajaa joko ei osannut mainita huonoja puolia, tai sanoivat niiden puuttuvan ohjelmoinnista. Ajan kuluminen ohjelmoinnin parissa nähtiin kahden vastaajan silmissä eräänä huonona puolena, ja yksi osallistuja sanoi, että ”ohjelmointiin käytetään joskus vanhoja ohjelmia, mikä voi tehdä siitä bugisen tai haavoittuvan”.

Kuva 10 esittää ohjelmoinnin mielikuvakysymysten vastausten jakauman pelaamisen jälkeen. Ohjelmoinnin vaikeutta tiedustelevan kysymyksen vastaukset olivat kaikilla osallistujilla nousseet paitsi yhdellä, jolla vastaus oli laskenut 3:sta 2:een. Ohjelmoinnin ”siisteys” oli myös kaikilla yhtä vastaajaa lukuun ottamatta noussut.

Kiinnostus ohjelmoinnin opettelua kohtaan pysyi vahvana, vaikka koulussa opettelun halu oli noin puolella osallistujista laskenut, ja toisella puolella noussut yhden pykälän verran. Myös halu opetella ohjelmointia itse tai pelejä pelaamalla pysyi vakaana.

Aiemmin mainittu ohjelmointia osannut ja siihen myönteisesti suhtautunut osallistuja piti kiinni vastauksistaan ohjelmoinnin opettelun halun suhteen, mutta laski haluaan opetella ohjelmointia koulussa yhdellä pykälällä. Yksi osallistuja pudotti haluaan pelata ohjelmointipelejä kahdella pykälällä 3:een. Kyseisellä osallistujalla myös muut ohjelmointihalut olivat laskeneet yhdeen asteen verran, paitsi itseopettelun osalta.



Kuva 10. Loppukyselyn ohjelmoinnin mielikuvakysymysten vastausten jakauma. Kysymykset on esitetty taulukossa 6.

Avoimissa kysymyksissä ohjelmoinnin hyviksi puoliksi kaksi osallistujaa mainitsi, että siinä voi tehdä ”juttuja” ja haluamiaan ohjelmia. Kolme muuta osallistujaa kertoivat uusien asioiden ja ”koodien” oppimisen sekä niiden käsittelyn olevan hyvä asia ohjelmoinnissa. Muiksi asioiksi kaksi osallistujaa mainitsi hauskuuden, yksi hyödyllisyyden, ja yksi asioiden nopeuttamisen ohjelmien avulla. Pelaamisen jälkeen huonoihin puoliin kuului kolmen osallistujan mukaan yhä se, että ohjelmointi oli hankalaa. Eräs osallistuja antoi vastauksensa yhteydessä myönteisen pilkahduksen ilmoittamalla, että ”opettelemalla oppii”. Huonojen puolien puuttuminen ilmoitettiin kahden osallistujan myötä erääksi ohjelmoinnin "huonoksi puoleksi". Kaksi osallistujaa kertoi huonona asiana, että pienenkin virhe voi tehdä ohjelmasta toimimattoman. Samoin kaksi muuta osallistujaa kertoi ohjelmoinnin vievän paljon aikaa, vaikkakin tässäkin näkyi myönteinen kipinä toisen ilmoittaessa tämän asian, että ”voi innostua liikaa ja olla vaan koneella koko ajan”.

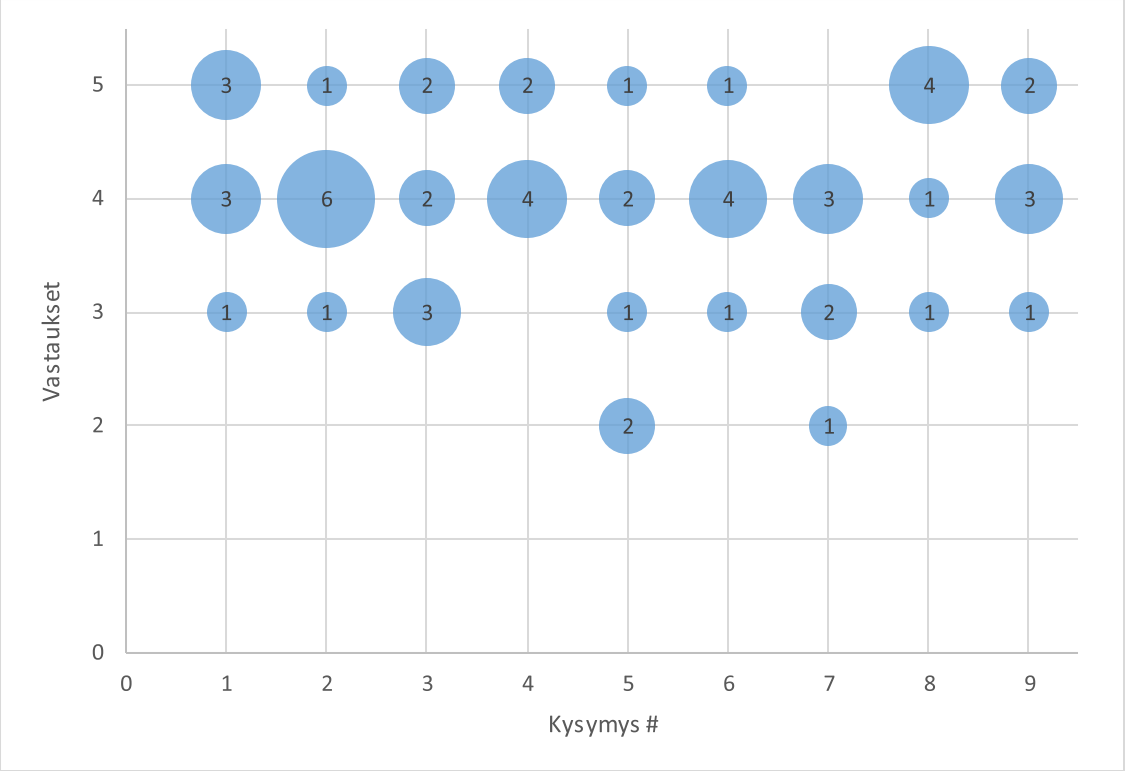
6.5 Osallistujien mielipide Koodilinnasta

Taulukosta 7 nähdään Koodilinnan mielipidettä tiedustelleet kysymykset loppukyselyssä. Kysymysten voidaan sanoa jakautuvan pelin yleiseen hauskuuteen ja käytettävyyteen viiden ensimmäisen kysymyksen osalta. Neljä viimeistä kysymystä tarkoitettiin selvittämään osallistujien mielipidettä pelin opettavaisuudesta. Näin ollen kysymykset jakautuivat pelimäiseen ja opettavaiseen osaan samalla tavoin kuin Koodilinnan toteutus.

Taulukko 7. Loppykyselyssä käytetyt pelin mielipidekysymykset.

Kysymys #	Kysymysteksti
1	Oliko pelaamasi peli kiinnostava? (1= Ei yhtään, 5 = Todella kiinnostava)
2	Oliko peli mukava? (1 = Ei yhtään, 5 = Todella mukava)
3	Oliko peli mukaansatempaava? (1 = Ei yhtään, 5 = Todella mukaansatempaava)
4	Oliko pelin idea helppo ymmärtää? (1= Ei yhtään, 5 = Todella helppo)
5	Oliko pelissä hyvät grafiikat? (1 = Ei yhtään, 5 = Todella hyvät)
6	Oliko ohjelmointia helppo ymmärtää pelin avulla? (1 = Todella vaikea, 5 = Todella helppo)
7	Oliko pelin tehtäviä helppo ratkaista? (1 = Todella vaikea, 5 = Todella helppo)
8	Tuntuiko peli hyöydylliseltä? (1 = Ei yhtään, 5 = Todella hyödylliseltä)
9	Tuntuiko peli edistävän ohjelmoinnin oppimistasi? (1 = Ei yhtään, 5 = Todella paljon)

Kuva 11 havainnollistaa osallistujien pelin mielipidekysymysten vastausten jakaumaa. Peli oli melko kiinnostava, mihin liittyy myös samoihin lukemiin yltävä pelin koettu mukavuus. Mukaansatempaavuus sai näistä kysymyksistä heikoimman vastaanoton. Osallistujien ymmärrys pelin ideasta sai melko hyvän tai paremman tuloksen, eli he ymmärsivät, mitä pelissä kuului tehdä. Tätä tukee myös ruudunkaappausmateriaali, mistä huomaa yleisesti pelaajien pääsevän nopeasti tekemään haluttuja asioita pelissä, eli tutkimaan maailmaa, puhumaan hahmoille ja ratkaisemaan tehtäviä. Vaikka pelaajat keräsivät tämän lisäksi opetusmateriaalisivuja kentistä, eivät he osanneet navigoida tekemään niiden sisältämiä sivutehtäviä itse, vaan tähän täytyi usein opastaa. Ohjelmointia oli osallistujien mielestä kuitenkin melko helppo ymmärtää, vaikka tehtävien ratkaiseminen oli keskimäärin tätä vaikeampaa. Peli tuntui myös lähes todella hyödylliseltä ja tuntui edistävän ohjelmoinnin oppimista melko paljon. Pelin grafiikat arvioitiin jo pilotointivaiheessa sekä tässä saatujen tulosten mukaan hieman keskimääräistä heikommaksi.



Kuva 11. Jälkikyselyn pelin mielipideosan vastausten jakauma. Kysymykset on esitetty taulukossa 7.

Osallistujat kommentoivat vielä jälkikäteen niin, että parannettavaa on löydettyjen toiminnallisten vikojen (bugien) osalta ja virheilmoitukset pitäisi saada selvemmäksi etenkin siltä osin, että peli kertoisi paremmin millä koodirivillä pelaajan virhe tuli. Selvästi yleinen vaikeus pelissä oppia oli jo mainittu *else if* -rakenne. Kuitenkin peli ilmoitettiin hyväksi ja motivoivaksi, ja ainakin osa osallistujista ilmoitti olevansa kiinnostuneita ohjelmoinnista pelin ansiosta.

Suullisilla pikahaastatteluilla haettiin tarkennuksia kyselyiden kysymyksiin ja erityisesti mielipiteitä pelistä. Usea osallistuja kertoi, että ehtorakenteita oli vaikeinta ymmärtää ja tähän saatiin tarkennus siltä osin, että niiden toimintaa ei selitetty pelissä kunnolla. Toisaalta peliä kehuttiin siltä osin, että se selosti hyvin miten kaikki tehdään. Yksi maininta tuli toistorakenteen olemisesta vaikein asia, ja toinen mainitsi muuttujien lainausmerkkien olevan tärkeä asia muistaa. Myös huomautus koodin virheiden huomaamisesta tuli esille.

Pelin parannusehdotukseksi tuli bugien parantaminen ja eräänä heikkoutena pidettiin useaan otteeseen sitä, ettei pelin antamia virheilmoituksia aina osannut jäljittää oikealle koodiriville esimerkeissä tai tehtävissä. Pelin jälkeen osallistujat ilmoittivat olevansa kiinnostuneita ohjelmoinnista.

6.6 Yhteenveto

Konstruktion arvioinnissa otettiin huomioon kolme asiaa, joista ohjelmointikäsitteiden oppiminen oli tärkein. Koska pelaamisen kuuluisi olla Caillois'n (1961) mukaan mieluisa aktiviteetti, myös tätä puolta tiedusteltiin. Lisäksi ohjelmoinnin mielikuvan muutosta haluttiin arvioida, sillä haluttiin tietää vaikuttaako peli kielteisesti opeteltavaan asiaan, mitä ei nähtäisi suotuisana tuloksena.

Kyselyiden ohjelmointitehtävissä nähtiin selvästi parannusta oikeiden vastausten määrissä, joten on pääteltävissä, että peli kykeni jollain tasolla selventämään joitakin ohjelmointikäsitteitä. Tulosta heikensi kuitenkin erinäiset epäselvät kohdat pelin opetusmateriaalissa, joiden parantaminen voisi mahdollisesti tukea käsitteiden vahvempaa ymmärtämistä. Muuttujien ymmärtäminen oli selvästi vahvin osa-alue, vaikka syntaksi kirjain- tai desimaalimuuttujissa ei jäänyt kaikilla osallistujilla mieleen. Laskutoimitusten osaaminen oli toinen vahva alue, jota seurasi ehtorakenteiden ja lopulta toistorakenteen ymmärtäminen.

Vaikka kuva ohjelmoinnista oli ilmeisen myönteinen ennen pelaamista, nousi näiden vastausten keskiarvo pelaamisen jälkeen. Kielteisinä asioina pidettiin virheiden suurta merkitystä ohjelman toimintaan ja ohjelmoinnin vaatimaa aikaa. Osallistujien keskimääräinen halu opetella ohjelmointia koulussa laski, mutta itseopettelun halu kasvoi. Yhden osallistujan vastauksen laskeminen laski keskimääräistä halua opetella ohjelmointia pelien avulla. Muiden osallistujien tapauksessa vastaukset pysyivät samoina tai lievästi myönteisempinä. Osa osallistujista ilmoitti olevansa motivoitunut ohjelmoinnista pelaamisen jälkeen, joten voidaan sanoa, että yleisesti ottaen halu opetella ohjelmointia kasvoi.

Peli nähtiin melko kiinnostavana ja mukavana, ja sen mukaansatempaavuus arvioitiin myös keskimäärin korkeaksi. Pelin grafiikat eivät yltäneet näiden vastausten tasolle. Osallistujat ymmärsivät pelin idean, vaikka sen sisältämiä tehtäviä ei ollut muiden vastausten keskiarvoihin nähden helppo ratkaista. Peli nähtiin vahvasti hyödyllisenä ja sellaisena, joka edistää ohjelmoinnin oppimista.

7. Pohdinta

Tässä luvussa pohditaan Koodilinnaa tutkimuksen tuloksia. Ensimmäisessä alaluvussa vastataan esitettyyn tutkimuskysymykseen pohtien tärkeimpiä tässä tutkimuksessa havaittuja asioita ohjelmointipelien suunnittelua koskien. Tämän jälkeisessä alaluvussa pohditaan Koodilinnan arvioinnin tuloksia, mitä seuraa alaluku sen suunnittelun aikana nousseista ajatuksista, puutteista ja jatkokehitysideoista.

7.1 Koodilinnan suunnittelu

Tämän tutkielman tutkimuskysymys oli, *millaisella pelillä kannattaa opettaa yläkouluikäisille ohjelmoinnin peruskäsitteitä*. Kysymykseen haettiin vastausta konstruktiivisella tutkimusotteella, jonka yhteydessä toteutettiin ohjelmointia opettava tietokonepeli, Koodilinna. Pelin haluttiin uppoavan Prenskyn (2005) luokittelussa pelipohjaiseen oppimiseen, missä nähdään onnistuneen. Pelimäisyyden ja opettavaisuuden yhdistäminen voi vapauttaa sen pelaamisen pelkässä luokkahuoneessa ja siihen tuotu interaktiivinen pelimaailma, dialogi, juoni ja pelimekaniikka ovat eräitä osia, jotka erottavat sen muista ohjelmointipeleistä. Koodilinnan voidaan ajatella olevan kuin perinteinen viihteellinen, mutta aidosti opettavainen tietokonepeli, ja tämä on toinen oleellinen erottava tekijä muista ohjelmointipeleistä. Pelin suunnittelu perustui aiemmasta tutkimuksesta kerättyihin suunnitteluperiaatteisiin, jotka voidaan tämän tutkielman löydöksien mukaan karkeasti jaotella pelien motivaatiotekijöihin, niiden opettavaisuuteen ja pelimäisyyteen, niiden yleisiin ulottuvuuksiin, ja ohjelmoinnin pedagogisiin näkökulmiin.

Pelin ulkoasulla ei todennäköisesti ole väliä, vaan muut pelimäiset elementit, etenkin pelimekanismi ja sen ongelmat ovat erityisessä asemassa. On tärkeää saada pelaaja jollain tavalla kiinnostumaan pelistä ja jatkamaan pelaamista jopa pelin loppuun saakka. Pelin kiinnostavuuteen vaikuttaa sen varsinainen pelimäisyys, johon Garris ja muut (2002) antavat ohjeita pelien ulottuvuuksien muodossa. Kyseiset ulottuvuudet eivät kuitenkaan anna tarkemmin ohjeita pelin toiminnallisen sisällön, eli pelimekaniikan ja pelin haasteiden suunnittelemiseen muuten kuin pinnallisesti pelaajan hallinnan, haasteen ja sääntöjen puolesta. Koska tehtävät ja niihin liitettävä opetuspelille ominainen opettavanen sisältö on suuressa roolissa pelin sisällössä, tulisi sen muun viihteellisen sisällön lisäksi olla mielenkiintoista. Ehkä pelin tehtävien sisältö ja pelimekanismi on jopa korostetussa asemassa muihin pelielementteihin verrattuna niiden tärkeyden vuoksi. Siksi niiden mieluisaksi mieluisaksi pelaajalle on tärkeää, ja Koodilinnan tapauksessa tähän pyrittiin pelaajan hallinnan tunteen ja omien strategioiden käyttämisen tarjoamisella. Tämän vuoksi liian yksinkertainen tapa ratkaista tehtäviä voi olla haitallista pelin mielenkiintoisuudelle, ja mitä kauemmaksi yksinkertaisesta ratkaisumallista tehtävissä päästään, sitä kiintoisampi niistä tulee, sillä peli siirtyy pois Crawfordin (1984) mainitsemasta pulmapelimäisestä olemuksesta. Tästä syystä voidaankin kritisoida Papastergioun (2009) tutkimaa tietokoneen muistikäsitteitä opettavaa peliä, sillä siinä tehtävät olivat pelkästään monivalintakysymyksiä. Toisaalta hänen havaintojensa mukaan pelaaminen oli jo niinkin yksinkertaisen pelin tapauksessa jännittävää osallistujille, vaikka toisaalta koulun penkillä pelin käyttäminen opiskeluun voi olla erikoinen ja mielenkiintoinen kokemus niin kauan, kuin sen uutuusarvo jaksaa viehättää.

Pulmapelimäinen opetuspelin olemus voi myös johtaa oppilaiden passivoitumiseen, eli he ottavat herkästi asioita vastaan ilman Kolbin (1984) mainitsemaa oppimisen kannalta tärkeää reflektointia. Mitä tulee juuri ohjelmointia tai muita aihetta opettavaan

tietokonepeliin, pelaajalle tulisi antaa vapauksia käyttää omia strategioitaan sen läpäisemisessä, eli esimerkiksi Koodilinnan tapauksessa antaa mahdollisuus kirjoittaa omaa koodia annetun ongelman mukaan ja testata omia hypoteeseja olematta jatkuvaa toistoa, mikä on Kiilin (2005) suositus opetuspeleille. Omien strategioiden käyttäminen mahdollistaa myös sen, että peli jaksaa kiinnostaa useampia ihmisiä, koska heitä ei pakoteta tiettyyn ongelmanratkaisumuottiin.

Opetuksellisuus ei saisi tuntua kuin se olisi erikseen lisätty siihen. Endogeenisen fantasian (Rieber, 1996) tavoin opetusmateriaali on upotettava pelin kontekstiin, eli juoneen ja pelimaailmaan, ja pelimekanismiin sulavasti. Koodilinnassa voidaan huomata, että kenttien läpäisemiseksi tehdyt päätehtävät oli sulautettu paremmin kuin ohjelmointikäsitteitä ja koodiesimerkkejä esittelevät ”kadonneet sivut”, joita pelaajat saivat kerätä kentistä. Päätehtävät oli pyritty sitomaan juoneen tehokkaasti esimerkiksi ensimmäisessä kentässä, jossa pelajaan täytyi ohjelmoida muuttuja ja asettaa sen arvoksi linnaan päästävä tunnussana. Kyseisen kentän kadonneissa sivuissa oli toisaalta muuttujakäsitettä esittelevässä koodiesimerkissä kyse muuttujan *taistelulaji* tekemisestä, mikä olisi voitu sulauttaa peliin paremmin esimerkiksi jollakin keskiaikaan liittyvällä muuttujalla.

Opetuksellisuus ja pelimäisyys ovat siis toisiinsa yhteydessä. Opetuspelissä oppimisen kuuluu tapahtua pelimekanismin ja ongelmien kautta, jotka on toisaalta voitu liittää muihin pelielementteihin. Mikäli pelimekanismi ja ongelmat on toteutettu sopivalla tavalla, muut pelielementit voivat hyvinkin jäädä pelissä toissijaiseksi pelin hauskuuden tai opettavaisuuden kärsimättä. Muita pelillisiä elementtejä ovat mainitut Garrisin ja muiden (2002) peliulottuvuudet, joita voidaan lisätä resurssien mukaisesti. Toisaalta päätös niiden käytöstä täytyy osin tulla jo harkitusti oikeassa vaiheessa projektia, sillä esimerkiksi fantasiaa liittäessä peliin liian myöhään, voi se tuntua mainitusti päälleliimatulta. Opetuksellisuus päätettiin Koodilinnassa liittää tehtäviin, mitä voidaan pitää yleisenä suosituksena opetuspelien toteutuksessa, jotta oppimisaktiviteetit olisivat samalla pelin keskeisiä aktiviteetteja.

ARCS-mallin sisältämien motivaatiostrategioiden liittäminen opetuspeliin voi Kellerin (1987) mukaan parantaa motivoivan oppimisen saavuttamista. Koodilinnan toteutuksessa malli oli hyödyllinen, vaikka strategioiden toteuttamiseksi pelin täytyi olla jo viety lähes pelattavaksi asti. Mallin strategiat toimivat pitkälti kuin kuorrutteena pelille, jotka saivat sen ehkä tuntumaan kiintoisammalta pelaajien mielestä. Toisin sanoen strategiat ovat siinä määrin joustavia, että niiden toteutusta ainakin Koodilinnan tapauksessa pystyi muuttamaan esimerkiksi pilotointien jälkeen, kun tehtäviä ja opetusmateriaalia jouduttiin uudistamaan. Voitaneen yleistää strategioiden merkitystä pelissä niin, että ne tekivät pelin dialogista mielenkiintoisemman ja paremmin ”virtaavan” pelissä esiintyneiden tehtävien ja ongelmien puolesta.

Lopulta voidaan sanoa, että Koodilinnan yleinen suunnittelu täytti lähes automaattisesti kerättyjä suunnitteluperiaatteita, mikä on yhteydessä Geen (2008) sanomaan, että yleinen pelisuunnittelu täyttää oppimisympäristön ehtoja. Tämä tarkoitti käytännössä sitä, että suunnitteluperiaatteiden soveltamista ei tarvinnut tehdä väkisin, vaan niiden lisääminen oli ennemmin kuin palapelin täyttämistä. Suunnitteluperiaatteiden on voinut sanoa viitoittavan toteutusprosessia sen sijaan, että ne olisivat sanelleet sille tiukan linjan. Kenties eniten Koodilinnaa suunniteltaessa pohdittiin sitä, mitkä ohjelmointikäsitteet tulisi liittää peliin. Christianin ja Mathranin (2014) mainitsemat koodin sekvenssittäinen suorittamisen ymmärtäminen, ehtorakenteet, toistorakenteet, funktiot ja rekursio viitoittivat tietä, mutta rekursio ja sekvenssittäinen suorittaminen eivät olleet arvion mukaan opetukseen sopivia asioita. Koodin sekvensimäisyyden hahmottamisen ajateltiin

tulevan ilmi koodiesimerkeissä ja rekursion taas ajateltiin olevan ulkona peruskäsitteistä, joihin miellettiin ensisijaisesti vain perussyntaksi ja niiden kevyt soveltaminen. Pilotointi ja Du Boulayn ja muiden (1981) kuvaileva kone olivat tärkeässä asemassa opetettavien ohjelmointikäsitteiden harkinnassa. Aivan aloittelijoille tulisi opettaa Koodilinnassa opetetut ohjelmointikäsitteet, eli muuttujat, ehtorakenteet, toistorakenne ja funktiot. Tämä toimii lähtökohtana muille syventäville käsitteille, joita voidaan löytää mainutuista käsitteistä tai esimerkiksi olio-ohjelmoinnin ja kehittyneemmän soveltamisen, kuten Christian ja Mathranin (2014) mainitseman rekursion saralta.

Tärkeintä on ottaa huomioon oppilaiden osaamistaso ja sovittaa siihen valittavat ohjelmointirakenteet, ja sovittaa samoin annetut ongelmat heidän osaamiseensa. Ei olisi siis mielekästä lähteä aivan ohjelmoinnin alkumetreillä opettelemaan esimerkiksi rekursiota, vaan lähteä mieluummin hiljalleen johdattelemaan haasteellisempiin ongelmiin opetettujen käsitteiden avulla. Näin voidaan myös kyseenalaistaa, onko olemassa varsinaisia hankaluuksia ongelmanratkaisuprosessin ymmärtämisessä, kuten Ala-Mutka (2004) väittää, vai löytyykö vika opetuksesta ja liian nopeasta tahdista käydä käsitteet läpi. Voidaan kuitenkin sanoa, että ongelmanratkaisusta tulee vaikeaa, jos sitä ei ohjelmoinnin opetuksessa koskaan tehdä, vaan esimerkiksi käydään ohjelmointikäsitteet vain mekaanisesti läpi. Peleissä toisaalta ongelmien liittäminen opetettaviin käsitteisiin on lähes luontevaa niiden suunnittelun kannalta, joten tältä osin pelit voivat olla vahvemmassa asemassa luennoivaan luokkaopetukseen verrattuna.

Pelkän syntaktisen osaamisen lisäksi pelissä siis kuuluu havainnollistaa ohjelmointirakenteiden käyttöä erilaisissa tilanteissa ja saada pelaajat itse yrittämään esiteltyjä käyttötapoja, mikä juuri on eräs tapa saada ongelmanratkaisua esille pelissä. Koodilinnassa annetut koodiesimerkit ja tehtävät liittyivät toisiinsa niin, että esimerkeistä pystyi soveltamaan, vaikkakaan ei suoraan kopioimaan, tehtävien vastauksen. Tehtävien ja itse kirjoitettavan koodin runsaus nähtiin eräänä tärkeänä tekijänä ohjelmoinnin opettamisessa, missä arvion mukaan onnistuttiin, sillä tehtäviä oli pelissä lopulta noin 40 ottamatta huomioon koodiesimerkkejä. Tehtävissä haluttiin myös soveltaa vanhaa opittua uuteen, mikä oli myös Gomesin ja Mendesin (2007) suositus. Koodin suorituksesta annettavaan palautteeseen tulisi myös panostaa, jotta pelaajat osaisivat ymmärtää, milloin koodi on oikein ja milloin jokin siinä on vialla. Palaute voi peleissä olla Koodilinnan tavoin dialogissa annettua, tai koodi voi suoraan muuttaa pelimaailmaa graafisesti jollain tavalla, mikä voisi olla kiehtovampi keino pelaajaa ajatellen. Pelaajien on myös annettava tehdä omia muutoksia koodiin ja nähtävä eri näkökulmista miten jokin koodi toimii. Jos pelaajalle ei anneta mahdollisuuksia tyydyttää ohjelmointiin liittyviä epäselvyyksiä koodia kokeilemalla, on vaarana, että hän muodostaa opetettavien asioiden toiminnasta oman käsityksensä, kuten Robins ja muut (2003) varoittavat.

Käytettävyys viihteellisissä ja opettavaisissa peleissä on erityisen tärkeää sen vuoksi, että pelaaja voi keskittyä pelin tarjoamaan hauskuuteen ja opetussisältöön, eikä miettimään, miten peliä käytetään. Etenkin oppimispeleissä pelaajan huomiota tulisi yrittää keskittää oikein, jotta hän tekisi pelin aikana haluttuja oppimisaktiviteetteja, eikä sen ulkopuolisia turhia toimintoja. Tähän tähdättiin Koodilinnassa yksinkertaistamalla pelin käyttöliittymää ja parantamalla pelattavuutta.

Pelin jako motivaatiotekijöihin, opettavaisuuteen ja pelimäisyyteen, yleisiin pelien ulottuvuuksiin, ja ohjelmoinnin pedagogiikan näkökulmiin ei lokeroinut niiden sisältämiä suunnitteluperiaatteita, vaan ne olivat keskenään yhteydessä. Tätä jakoa ja sen suunnitteluperiaatteita on mahdollista käyttää myös muissa ohjelmointia opettavissa peleissä, oli se suunnattu joko yläkouluikäisille tai jollekin muulle ikäryhmälle, mutta

tarkemmat pelin ominaisuudet suunnitteluperiaatteiden osalta tulee harkita tapauskohtaisesti.

7.2 Koodilinnan arvioinnin tulokset

Koodilinna arvioinnista saatujen tulosten perusteella toteutettu peli opettaa ohjelmointia. Tätä tukee alku- ja jälkikyselyiden ohjelmointitehtävien tulokset, joiden saatu keskimääräinen pistemäärä oli pelaamisen jälkeen huomattavasti suurempi kuin pelaamista ennen saatu tulos. Tuloksista kävi kuitenkin ilmi, että peli ei kyennyt tarjoamaan täyttä ymmärrystä eri käsitteistä, sillä joidenkin kohdalla näkyi puutteita osaamisessa. Esimerkiksi kirjaintyyppisen muuttujan arvoa kirjoittaessaan osallistujat unohtivat usein lainausmerkit ja *else if* -ehtorakennetta ei käsitetty kunnolla. Etenkin ehtorakenteen ymmärryksen puutteen syyksi pääteltiin pelin opetusmateriaalin puutteet. Eräs toinen lovi Koodilinnassa huomattiin siinä, että osallistujat eivät huomanneet tehdä kenttiin ripoteltujen opetusmateriaalien tehtäviä, niiden esille ottaminen vaati pienen nuolen painamista opetusmateriaalisivuista. Näiden sivutehtävien ideana oli valmentaa pelaajaa opetettavien ohjelmointikäsitteiden käytössä ja pelin päätehtävien läpäisemisessä. Sivutehtävien löytämiseen piti osallistujia usein erikseen ohjata. Yleisesti ottaen pelin pelattavuus oli kuitenkin siinä määrin hyvä, että pelaaminen meni eteenpäin omalla painollaan.

Ohjelmoinnin oppimisen lisäksi arvioinnissa tiedusteltiin osallistujien mielipidettä pelistä. Yleisesti ottaen peli koettiin mukavaksi, hyödylliseksi ja hyväksi oppimisen kannalta. Grafiikat ja mukaansatempaavuus jäivätkin muita tiedusteltuja ominaisuuksia alemmaksi. Erityisesti mukaansatempaavuuteen on voinut vaikuttaa se, että vaikka osallistujat vaikuttivat olevan uppoutuneina peliin, ajoittaiset pelin hankalat kohdat särkivät tätä tunnetta. Peli saattoi olla joiltakin osin jopa turhauttava, mikä saattoi johtua jälleen siitä, että osallistujien mukaan pelin opetusmateriaalin tekstit eivät olleet kauttaaltaan kattavia. Tämän vuoksi myös asioiden ymmärtämiseen jäi aukkoja etenkin ohjelmoinnin ehtorakenteissa, joten pelin tehtävät eivät keskimäärin olleet kovin helppoja ratkaista. Tätä myötäilee uppoutumiseen vaadittava sopiva vaikeusaste (Garris ja muut, 2002) ja tehokkaan oppimisympäristön vaatimukset (Houser ja DeLoach, 1998), joiden mukaan liian vaikea haaste turhauttaa pelaajat.

Koodilinnaa arvioitiin myös siltä kantilta, mitä osallistujat olivat mieltä ohjelmoinnista pelaamisen jälkeen. Peli sai osallistujat kiinnostumaan ohjelmoinnista, vaikka koulussa ohjelmoinnin oppimisen halu laski. Vaikka ajatukset ohjelmointia kohtaan olivat myönteisiä jo ennen pelaamista, nähtiin tämän suhteen myönteinen muutos pelin ansiosta. Kirjallisia vastauksia tarkastellessa huomattiin, että ohjelmoinnin hankaluutta ilmaisevia vastauksia oli yhtä paljon. Toisaalta ohjelmoinnin vaikeutta tiedustelevan likert-kysymyksen mukaan ohjelmointi tuntui osallistujien mielestä helpommalta pelaamisen jälkeen, joten kokonaisuudessaan mielikuva ohjelmoinnin vaikeudesta näyttää hieman lievenneen. Lopulta ohjelmointiin liittynyt vaikeuden tunne väheni, vaikka itse peli koettiinkin osittain hankalaksi. Keskimääräinen halu opetella ohjelmointia pelien avulla kuitenkin laski, mikä voidaan nähdä ristiriitana verratessa tätä muihin myönteisiin tuloksiin. Tämän todettiin kuitenkin johtuvan yhden osallistujan vastauksen laskusta kahdella pykälällä, kun muilla osallistujilla vastaukset pysyivät samoina tai nousivat lievästi. Täten voidaan kysyä, onko syynä aineiston pienuus, vai voitaisiinko sama todeta laajemmalla aineistolla.

Yhteenvetona voidaan sanoa, että Koodilinna opetti ohjelmoinnin peruskäsitteitä, vaikka syvällinen ymmärrys jäi niistä vielä puuttumaan. Peli motivoi osallistujia opettelemaan ohjelmointia tai paransi mielikuvaa ohjelmoinnista. Tämän lisäksi itse peli koettiin

miellyttäväksi ja hyväksi ohjelmoinnin opettelun kannalta. Tulokset ovat yhteneväisiä Papastergioun (2009) tutkimuksen kanssa, jonka mukaan opetuspelit ovat motivoivia, vaikka taustalla voi olla myös pelien uutuusarvo, joka voitaisiin eliminoida pelaamalla pelejä opetustarkoituksessa useammin.

Eräs osallistuja mainitsi omaavansa kokemusta Lua-skriptauskielestä, ja vastasi tätä myötä alkukyselyssä ohjelmoinnin osaamiseksi 2, eli tunsi tietävänsä jotakin ohjelmoinnista. Hän myös vastasi ohjelmoinnin olevan helpompaa ja vastasi yleensäkin myönteiseen sävyyn ohjelmointia tiedusteleviin kysymyksiin. Kyseinen osallistuja oli erityisen optimistinen jälkikyselyn tehtävien tekemisen jälkeen, sillä hän vastasi niiden vaikeutta tiedustelevaan kysymykseen 5, eli ne tuntuivat todella helpoilta. Tästä huolimatta kyseinen osallistuja sai keskimääräisen tuloksen ohjelmointitehtävistä jälkikyselyssä. Toisaalta osallistuja ilmoitti sanallisissa vastauksissa annettujen ohjelmointitehtävien vaikeaksi puoleksi suomen kielen käyttämisen niissä totutun englannin kielen sijaan.

Mitä tulee Perkinsin ja muiden (1986) oppilastyyppeihin, arviointiin osallistuneet eivät olleet tyypiltään pysähtyjiä, sillä yksikään heistä ei luovuttanut hankalaksi ilmenneen ongelman edessä. Ennemmin voidaan sanoa, että osallistujat olivat liikkujien ja superliikkujien väliltä, sillä he reagoivat asianmukaisesti pelin antamaan palautteeseen ja ratkaisivat sen avulla pelin ongelmia. Mikäli osallistujat jumiutuivat johonkin kohtaan pelissä, vasta tällöin osa alkoi kokeilla nopeammin eri tapoja päästä tehtävä läpi. Tällöin oli tarvetta ohjata osallistujia ottamaan mallia koodiesimerkeistä, joista usein löytyi suora malli tehtävän ratkaisemiseksi. Robinsin ja muiden (2003) tehokkaiden ja vähemmän tehokkaiden aloittelijoiden jako näkyi osallistujissa niin, että osa osallistujista kykeni etenemään pelissä hyvinkin oma-aloitteisesti, ja joitakin heistä täytyi tukea enemmän ja ohjata juuri pelin koodiesimerkkien lukemiseen.

7.3 Koodilinnan suunnitteluperiaatteet

Pelin suunnittelun eräs tärkeä periaate oli tasapainottaa viihteellinen ja opettavainen sisältö Prenskyn (2005) suosituksen mukaisesti. Näiden kahden sopivaaa suhdetta pyrittiin saamaan pelissä aikaan hahmon vapalla liikkuvuudella ja interaktiolla maailman kanssa, ja tehtävissä annettiin pelaajille vapaat kädet niiden ratkaisemiseksi. Tässä onnistutiin arvion mukaan hyvin, sillä pelaajien täytyi keskittyä juuri opetukselliseen osaan peliä siinä edetäkseen, mutta opetuksellinen materiaali oli myös tehty pelimäiseksi. Tehtävät itsessään olivat pelimäisiä, mutta sisälsivät aina opetusmateriaalissa esiteltyjä käsitteitä. Opetusmateriaalia ei myöskään haluttu ”liimata” muun pelin päälle, joten sen upottamista harkittiin tarkkaan, jotta se sopisi pelin tilanteeseen juonen osalta. Tästä syystä pelimekaniikkaa suunniteltiin alusta lähtien muun pelimaailman kanssa, mikä osoittautui tehokkaaksi keinoksi näiden kahden sulauttamiseen keskenään. Tehtävistä päätehtävät saatiin erityisen hyvin asetettua pelimaailman kontekstiin, mutta ”kadonneista sivuista” löytyneet koodiesimerkit ja sivutehtävät olisi voinut upottaa paremmin. Toisaalta näissä tapauksissa peliin haluttiin tuoda hauskuutta ja tuttuuden tunnetta pelaajien omasta maailmasta tuotujen esimerkkien avulla, joten niiden ollessa sivuroolissa myöskään niiden oleminen ulkona pelin ydinkontekstista ei tuntunut haittaavan.

Motivaation puolesta Koodilinnan toteutuksessa päätettiin käyttää ARCS-mallia (Keller, 1987) ja flow-teoriaa, sillä ne antoivat konkreettisia esimerkkejä kuinka parantaa oppimista Koodilinnassa. Mitä tulee flow-teoriaan, peliä arvioidessa havaittiin, että osan aikaa osallistujat olivat uppoutuneita peliin, mutta turhautuminen pelin joidenkin hankalien kohtien kanssa sai osallistujat levottomammaksi, kuten voidaan ennustaa

Garrisin ja muiden (2002) maininnasta flow-tilan ja pelin sopivan vaikeusasteen yhteydestä. Vaikeusasteen lisäksi opetusmateriaalin puutteet aiheuttivat hankaluuksia edetä pelissä, mikä luultavasti aiheutti saman turhautumisen tunteen. Alun perin flow-teorian hyödyntäminen tarkoitti pelin käytettävyyden säätämistä sellaiselle tasolle, että pelaajat voisivat keskittyä pelaamiseen eikä pelin käyttämiseen. Myös vaikeusaste haluttiin saada sopivalle tasolle, ja tässä oli tärkeässä roolissa opetettavien ohjelmointikäsitteiden rajaaminen ja niiden tuominen luontevalla tavalla esille. Pilotointi osoitti, että lyhyet opetustekstit ja runsas pelaajan itse muokattavissa oleva koodisisältö toimisi parhaiten uusien asioiden oppimisessa, mikä on yhteydessä Robinsin ja muiden (2003) mainintaan runsaiden laboratoriotehtävien merkityksestä ohjelmoinnin oppimiselle. Koodilinnan sivutehtävät ja koodiesimerkit luotiin pilotoinnin jälkeen, mutta ongelmaksi jäi, että päätehtävät olivat vielä vanhaa perua. Päätehtävät saatiin säädettyä sopivalle vaikeustasolle vertailemalla niitä sivutehtäviin, mikä osoittautui vaivalloiseksi. Tulevaisuudessa pilotointi olisi suotavaa tehdä aikaisemmassa vaiheessa ja käyttää esimerkiksi paperille tehtyjä kevyempiä prototyyppejä.

Garrisin ja muiden (2002) kuvailemat pelien ulottuvuudet olivat kuin pelin hahmotelma, joiden avulla muut suunnitteluperiaatteet toteutettiin. Näitä ulottuvuuksia tarkastellessa voidaan sanoa, että pelin toteutus lähti liikkeelle pelimekaniikasta, johon yritettiin saada ideoita muista peleistä, jonka päälle koottiin muita ulottuvuuksia. Koodilinnan toteutusta voidaan verrata venäläiseen nukkeen, joka kunkin ideoidun kerroksen jälkeen lähes luonnollisesti vei uuden kerroksen ideointiin. Näin tultiin siihen tulokseen, että pelimekanismi on tärkeimmässä asemassa pelien suunnittelua, sillä sen päälle on aina mahdollista lisätä kaikki muut pelin ominaisuudet. Pelien ulottuvuudet antoivat runsaasti artistista vapautta erityisesti fantasiaulottuvuuden kohdalla. Suurimpana fantasian suunnittelua ohjanneena tekijänä oli eksogeeninen fantasia (Rieber, 1996), eli päälleliimattu fantasia, jonka tunnetta yritettiin välttää lähtemällä sunnittelemaan yhtä aikaa sekä opetuksellista että pelimäistä konstruktion olemusta.

Fantasian piiriin sisällytetty taikakehä (Salen ja Zimmerman, 2003) ja sen sisään astuminen otettiin huomioon niin, että fantasiasta pyrittiin tekemään relevantti pelaajille ARCS-mallin (Keller, 1987) huomio- ja relevanssistrategioiden avulla. Yleisesti ottaen pelin fantasiaelementtien todesta otettavuuden tai ”sulatettavuuden” huomioiminen oli tärkeä osa taikakehään pääsyn suunnittelua. Erityisen haastavaa oli ideoida yläkouluikäisille sopivia viitteitä, eikä sen onnistumisesta ole tietoa, sillä ei voitu olla varmoja kuinka itselleen merkittäväksi osallistujat näkivät humoristiseksi ja viehättäväksi tähdätyn dialogin. Suunnittelussa voitaisiinkin osallistaa sen ikäryhmän edustajia, jolle peli tehdään, mikä vaatii suunnitteluprosessin muuttamista ja vie enemmän aikaa. Jos tutkimuksen resurssit olisivat riittäneet, tämä olisi voinut olla eräs vaihtoehto Koodilinnankin tapauksessa.

Sopiva haaste oli eräs hankalasti arvioitava ominaisuus, sillä tehtävien ja esimerkkien suunnittelussa täytyi aina muistaa kohderyhmän ohjelmoinnin osaamistaso. Haasteen säätämiseen kuului etenkin yksinkertaistetun kuvailevan koneen (Du Boulay ja muut, 1981) saaminen sopivalle tasolle, missä ennen arviointia tehdyt pilotoinnit auttoivat. Opetussisällöstä löytyneet puutteet olivat lopulta todennäköisesti suuressa osassa pelin koettua vaikeutta, vaikka käytettävyydenkin pulmat olivat omassa osassa ruokkimassa pelin hankaluuden tunnetta. Ohjelmointikäsitteiden yleinen valinta peliin ja niidenkin osalta opetettavien asioiden karsinta kuitenkin onnistuivat, sillä osallistujat saivat jonkinlaisen ymmärryksen niiden toiminnasta.

Koodilinnassa ilmenevät peleistä oppimisen ominaisuudet, kuten havainnollistavat esimerkit ja soveltavat tehtävät, yhdistyvät kokemuspohjaisen oppimisen (Gee, 2008)

kanssa. Pelissä olleet esimerkit ja tehtävät suunniteltiin sellaisiksi, että ne voisivat tarjota kokemuksia, joita pelaajat voisivat käyttää jatkossa hyödykseen ongelmanratkaisussa. Tämä oli päätehtävien suunnittelun aikana tärkein tavoite, mikä saatiin arvion mukaan onnistumaan hyvin, sillä kukin ohjelmointikäsite saatiin liitettyä johonkin sitä hyvin kuvailevaan ongelmatilanteeseen. Esimerkiksi piirakoiden lukumäärää saatettiin kuvailla murtoluvuilla, tai toistorakennetta kolikoiden nostamisella kirstusta. Oppimiseen kuului myös pelin antama palaute, mitä annettiin koodin suorituksen yhteydessä ja kenttien lopussa. Palautteen pelättiin olevan liian yksipuolista pelkkien puhekuplien vuoksi, mutta ne osoittautuivat kuitenkin tehokkaaksi tavaksi tuoda yhteen sekä palaute että koodin visualisointi.

Vaikka peli kokikin useita radikaaleja uudistuksia käytettävyytensä puolesta, joitakin epävarmoja tekijöitä jäi pelissä elämään arviointiin saakka. Ensinnäkin peliin haluttiin saada aikaan tuntuma "oikeasta" pelistä sen sijaan, että peli olisi tuntunut koulumaiselta opetukselta. Oikean pelin tunnelmaa yritettiin tuoda esimerkiksi fantasian, pelimekaniikan ja pelaajalle annetun vapauden kautta. Opetusmateriaalin upotus peliin oli myös eräs käytetty keino tähän ja pääasiallinen epävarmuus kumpusikin juuri siitä, että upottamista ei olisi tehty tarpeeksi hyvin, jolloin materiaali tuntuisi päälleliimatulta. Ei ole varmuutta, kuinka hyvin tässä onnistuttiin, vaikka peli saikin yleisesti ottaen hyvän arvion osallistujilta.

Toisena huolenaiheena oli ohjelmoinnin visualisointi, sillä pelissä tehtävien “visuaalisena” palautteena tuli pelkkä teksti. Vaikka pelin visualisointi olikin Napsin ja muiden (2002) visualisointitasoja tarkastellessaa viidennellä tasolla kuudesta, eli rakentamistasolla, tekstin ajateltiin kuitenkin olevan riittämätön varsinaiseksi visualisoinniksi. Tämän takia arvioitiin, että kyseisen kaltainen peli soveltuisi paremmin kokeneemmille ohjelmoijille, vanhemmille oppilaille, kuten lukioikäisille, tai erityisesti tällaisesta palautteesta pitäville. Pelko visualisoinnin puutteesta ei ilmeisesti pitänyt paikkaansa, sillä osallistujat lukivat palautetekstejä ja tekivät niiden mukaan muutoksia tekemäänsä koodiin. Pelin jatkokehityksessä voitaisiin kuitenkin panostaa vielä rikkaampaan visualisointiin, jossa itse pelimaailma voisi muuttua suoraan pelaajan koodin ansiosta.

Pelin selvä heikkous oli osittain puutteelliset opetustekstit siltä osin, että ne eivät arviointiin osallistuneiden mukaan selittäneet jokaista aihetta tarpeeksi ymmärrettävästi. Tämä päti erityisesti ohjelmoinnissa esiintyviin ehtorakenteisiin, joita ei osallistujien palautteen mukaan ollut kunnolla selvennetty. Ehkä ehtorakenteisiin kuuluva alarakenne *else if* olisi voitu jättää pelistä pois tai jättää se myöhemmäksi. Tämä on eräs varteenotettava seikka pelin jatkokehitystä ajatellen. Paikallisiin opetusmateriaalin puutteisiin liittyi myös pelin nopea tahti esitellä ohjelmointikäsitteitä, jolloin yksittäisten käsitteiden vahvaan ymmärtämiseen ei annettu kovinkaan montaa mahdollisuutta. Peliä voitaisiin pidentää jatkokehityksessä niin, että pelaaja voisi keskittyä yhteen aiheeseen pidemmän aikaa. Näin peliä kohtaan koettua vaikeutta voitaisiin lieventää lievemmällä vaikeuskäyrällä, jolloin pelaajat voisivat myös uppoutua peliin tehokkaammin (Garris ja muut, 2002).

Lisäksi käytettävyyspulmat kaihersivat peliä joissakin tapauksissa. Kenttiin asetettuihin opetusmateriaaleihin ei aina osattu navigoida ilman neuvontaa. Se, että pelaajat eivät päässeet tutustumaan opetettaviin käsitteisiin teki pelin kenttien läpäisystä huomattavasti vaikeampaa ja turhauttavaa, joten oppimisen ja peliin uppoutumisen kannalta käytettävyys näyttäisi olevan tärkeä seikka oppimispeleissä, mikä on yhteneväinen Kiilin (2005) maininnan kanssa.

Pelin sisältämät tehtävät olivat sellaisia, että niiden luonnolliseen ratkaisemiseen ei lopulta ollut useita järkeviä vaihtoehtoja, vaikka tehtävissä pitikin soveltaa ja saada aikaan tekstipohjaisesta tehtävänannosta oikea vastaus. Siispä pelin jälleenpelattavuus on matala. Uusi opetuksellinen elementti voitaisiin pystyä lisäämään peliin sitä kautta, että annetaan pelaajille työkalut ohjelmoida peliin omia kenttiä, jolloin oppiminen jatkuu pelin ulkopuolellakin. Itse tehdyt kentät siirtyisivät lähemmäksi Scratchin käyttämisen (Resnick ja muut, 2009) kaltaista aktiviteettia, jolloin peli saisi uusia ulottuvuuksia jakamisen ja luovuuden muodoissa.

8. Päätäntä

Tässä tutkielmassa tutkittiin, millaisella pelillä kannattaa opettaa yläkouluikäisille ohjelmoinnin peruskäsitteitä. Kysymykseen vastattiin konstruktiivisella tutkimusotteella, jossa toteutettiin ohjelmoinnin Koodilinnaksi nimetty opetuspeli perustuen aiemmasta tutkimuksesta kerättyihin suunnitteluperiaatteisiin. Koodilinna arvioitiin kvalitatiivisesti tekemällä tutkimuskäynti Oulun Pateniemen koululle, missä eräältä kahdeksannelta luokalta osallistui tutkimukseen kahdeksan oppilasta. Arviointikriteereiksi asetettiin pelin opettavaisuus, mieluisuus, ja ohjelmoinnin mielikuvien myönteisyyden pitäminen vähintään samalla tasolla kuin ennen pelaamista. Arvioinnin aikana täytettiin alkukysely, jolla tiedusteltiin aiempaa kokemusta ja mielipidettä tietokoneista, peleistä ja ohjelmoinnista. Kysely sisälsi myös ohjelmointitehtäviä, joihin osallistujat yrittivät vastata osaamisensa mukaan. Kyselyn jälkeen pelattiin peliä, mitä seurasi loppukysely. Loppukysely sisälsi samoja ohjelmoinnin mielipidettä tiedustelevia kysymyksiä ja identtiset ohjelmointitehtävät kuin alkukyselyssä. Lisäksi loppukyselyssä tiedusteltiin osallistujien mielipidettä pelin eri ominaisuuksista, kuten hauskuudesta, grafiikoista ja opettavaisuudesta. Verratessa alku- ja loppukyselyiden vastauksia keskenään, tultiin siihen tulokseen, että peli oli kyennyt opettamaan ohjelmoinnin peruskäsitteitä, vaikka syvä ymmärrys oli jäänyt puuttumaan. Peli koettiin mieluisaksi, hyödylliseksi ja opettavaiseksi. Lisäksi jo ennalta myönteiset ohjelmointia kohtaan koetut mielikuvat muuttuivat myönteisemmäksi.

Vastaus tutkimuskysymykseen saatiin tarttumalla Koodilinnan toteutuksessa käytettyihin suunnitteluperiaatteisiin, jotka jaettiin yleisellä tasolla pelien motivaatiotekijöihin, opettavaisuuteen ja pelimäisyyteen, yleisiin pelien ulottuvuuksiin, ja ohjelmoinnin pedagogisiin näkökulmiin. Pelimekanismi ja tehtävät ovat pelissä tärkeässä roolissa ja vasta tämän jälkeen tulevat muut elementit, kuten grafiikat ja hienosäikeinen juoni. Pelin tehtävien täytyy olla sellaisia luonteeltaan, että niihin ei voi vastata aina samalla kaavalla pulmapelimäiseen tyyliin, vaan sen sijaan pelaajan on annettava käyttää omia strategioitaan tehtäviä tehdessään. Opetuksellisuutta ei saa lisätä peleihin tarramaisesti, vaan se on sulautettava niihin käyttämällä opetusmateriaalissa hyödyksi pelimaailman teemoja. Motivaatiokeinoja opetuspeliin voidaan lisätä myös jälkikäteen, vaikka yleiset peliulottuvuudet ja opetusmateriaali tulisikin niistä poiketen sulauttaa mahdollisimman pian peliä suunniteltaessa. Pelaajan keskittyminen pelaamiseen ja erityisesti oppimista edistäviin aktiviteetteihin täytyy varmistaa suunnittelemalla pelin käytettävyys, käyttöliittymä ja pelattavuus. Ohjelmoinnin opetussisältöä suunniteltaessa voidaan lähteä liikkeelle hahmottelemalla kuvaileva kone, jonka puitteissa pohditaan, mitkä osat ohjelmoinnista kuuluisi ottaa mukaan opetukseen. Pelien ei tule opettaa kuitenkaan pelkkiä ohjelmointirakenteita, vaan niiden keskinäistä käyttöä ja ongelmanratkaisua annetusta tehtävänannosta. Pelaajan tulisi saada kirjoittaa runsaasti omaa koodia ja soveltaa vanhaa opittua uusiin asioihin. Pelin palautteen koodin suorituksesta täytyy tukea pelaajan ongelmanratkaisua ja virheiden löytämistä omasta koodista.

Lopulta Koodilinnan arvioitiin olevan Prenskyn (2005) opetuksellisten ja viihteellisten pelien luokittelussa pelipohjaiseen oppimiseen perustuva peli. Tämän ajateltiin mahdollistavan tehokkaammin sen pelaamisen luokkahuoneen ulkopuolella ilman opettajan valvontaa. Pelin nähtiin eroavan muista ohjelmoinnin opetuspeleistä interaktiivinen pelimaailman, dialogin, juonen ja pelimekaniikan yhdistelmänsä kautta. Näin ollen Koodilinna voidaan nähdä esimerkkinä siitä, että hyvinkin viihteelliseltä tuntuva peli voi olla opettavainen.

Tämän tutkielman merkitys on erityisesti se, että se laajentaa vielä tällä hetkellä suppeaa valikoimaa ohjelmoinnin opetuspeleissä ja osoittaa, millaisia pelejä opetuspelit voivat yleensä olla. Myönteiset tulokset ohjelmoinnin oppimisen kannalta viittilöivät myös siihen suuntaan, että opetusta pelien avulla voitaisiin yrittää myös muiden aiheiden osalta, sillä tässä tutkielmassa esitetyt suunnitteluperiaatteet ovat sovellettavissa ohjelmoinnin ulkopuolelle. Tutkimuksen relevanssi tulee erityisesti siitä, että ohjelmoinnin opetus on tekemässä tuloaan peruskouluun (Opetus- ja kulttuuriministeriö, 2014). Vaikka täysin viihteelliset pelit pysyvät osana elämäämme, ehkä pelaaminen aletaan nähdä myös sellaisena aktivitettina, jolla on kyky opettaa ohjelmoinnin lisäksi mitä moninaisimpia asioita. Ehkä pelejä voidaan ajatella muunakin kuin silmittömän väkivallan tai mieltä tylsyttävän viihteen lähteenä.

Etenkin koulutusalan ammattilaisille tämä tutkimus antaa ehdotuksia siitä, että pelit voisivat olla eräs keino täydentää opetusta mieluisalla tavalla ja saada heidät oppimaan aktiivisen osallistumisen ja reflektoinnin kautta. Tutkimus hyödyttää muuta tutkimusta olemalla vahvistuksena opetuspelien tehokkuudesta. Muille tutkijoille tutkimuksesta on hyötyä niin, että se tarjoaa valmiin kehyksen suunnitteluperiaatteista, joita voidaan hyödyntää muussa opetuspelien tutkimuksessa.

Tutkimuksen rajoituksena oli aineiston pieni koko ja arviointitilaisuuteen varattu lyhyt aika, sillä osallistujat eivät ehtineet päästä peliä läpi. Lisäksi pelissä ilmenneet puutteet saattoivat vaikuttaa tutkimuksen tuloksiin etenkin vaikeuden osalta. Jatkotutkimuksen kannalta pelin käytettävyys ja toiminnalliset viat tulisi korjata. Pelin opetusmateriaalia tulisi selkeyttää ja pelin pituutta lisätä, jolloin pelaajat voisivat keskittyä kuhunkin opetettavaan asiaan syvällisemmin. Lisäksi omien kenttien lisäysmahdollisuutta peliin ehdotettiin, jolloin peli voisi olla opettavainen myös oman luovuuden kautta. Tutkimus on relevantti siksi, että ohjelmoinnin opetus ollaan sisällyttämässä peruskoulujen opetussuunnitelmaan (Opetus- ja kulttuuriministeriö, 2014). Tutkimuksen tarjoamat suunnitteluperiaatteet voisivat olla jatkossa hyödyksi pelien toteutusprosessia tai kattavampaa viitekehystä kehittävässä tutkimuksessa. Tutkimus rohkaisee myös yleensä opetuspelien tutkimusta muidenkin opetusaiheiden osalta

Jatkotutkimuksen kannalta seuraava askel voisi olla yleistävämpi malli opetuspelien toteutusprosessista tai systemaattisempi opetuspelien viitekehys. Tutkimuksessa olisi syytä osallistaa suurempi joukko oppilaita mukaan pidempään tutkimukseen. Ehkä pelien vieminen luokkahuoneeseen ja liittäminen se osaksi opetusta voisi olla hyödyllistä sen osoittamisessa, että niillä voitaisiin opettaa ohjelmointia pidemmälläkin aikajänteellä. Tämä paikkaisi myös puutetta nykyisessä tutkimuksessa mitä tulee opetuspelien tehokkuuteen verrattuna perinteiseen opetukseen. Mikäli tässä tutkielmassa toteutettua ohjelmointipeliä haluttaisiin käyttää jatkossa, olisi sen käytettävyyteen liittyvät pulmat ja opetusmateriaalin osittaiset puutteet suotavaa korjata. Lisäksi pelin pidentäminen, ja pelaajien omien kenttien tekeminen ovat mahdollisia jatkokehitysehdotuksia pelin syvemmän opettavaisuuden kannalta.

Lähteet

Ala-Mutka, K. (2004). Problems in learning and teaching programming. *Codewitz Needs Analysis*.

Bromwich, K., Masoodian, M., & Rogers, B. (2012, July). Crossing the game threshold: A system for teaching basic programming constructs. *Proceedings of the 13th International Conference of the NZ Chapter of the ACM's Special Interest Group on Human-Computer Interaction* (pp. 56-63). ACM.

Bruce-Lockhart, M. P., & Norvell, T. S. (2000). Lifting the hood of the computer: program animation with the Teaching Machine. In *Canadian Conference on Electrical and Computer Engineering, 2000* (Vol. 2, pp. 831-835). IEEE.

Caillois, R. (1961). *Man, play, and games*. University of Illinois Press.

Chan, T. S., & Ahern, T. C. (1999). Targeting motivation-adapting flow theory to instructional design. *Journal of Educational computing research*, *21*(2), 151-164.

Christian, S., & Mathrani, A. (2014). Play2Learn: A Case of Game Based Learning Approach in ICT Education. *Proceedings of the 25th Australasian Conference on Information Systems*. ACIS.

Cordova, D. I., & Lepper, M. R. (1996). Intrinsic motivation and the process of learning: Beneficial effects of contextualization, personalization, and choice. *Journal of educational psychology*, *88*(4), 715.

Corritore, C. L., & Wiedenbeck, S. (1991). What do novices learn during program comprehension?. *International Journal of Human - Computer Interaction*, *3*(2), 199-222.

Crawford, C. (1984). *The art of computer game design*. Berkeley, Calif.: Osborne/McGraw-Hill.

Crookall, D., Oxford, R., & Saunders, D. (1987). Towards a reconceptualization of simulation: From representation to reality. *Simulation/Games for Learning, 17*(4), 147-171.

Davies, S. P. (1993). Models and theories of programming strategy. *International Journal of Man-Machine Studies*, *39*(2), 237-267.

De Castell, S., & Jenson, J. (2003). OP- ED Serious play. *Journal of Curriculum Studies*, *35*(6), 649-665.

Deci, E. L., Vallerand, R. J., Pelletier, L. G., & Ryan, R. M. (1991). Motivation and education: The self-determination perspective. *Educational psychologist*, *26*(3-4), 325-346.

Dreyfus, H. L., Dreyfus, S. E., & Athanasiou, T. (1987). *Mind over machine: The power of human intuition and expertise in the era of the computer*. Free Press.

Du Boulay, B., O'Shea, T., & Monk, J. (1981). The black box inside the glass box: presenting computing concepts to novices. *International Journal of Man-Machine Studies*, *14*(3), 237-249.

Du Boulay, B. D. (1986). Some difficulties of learning to program. *Journal of Educational Computing Research*, *2*(1), 57-73.

Elenbogen, B. S., Maxim, B. R., & Mcdonald, C. (2000). Yet, more Web exercises for learning C. *ACM SIGCSE Bulletin, 32*(1), 290-294.

Garris, R., Ahlers, R., & Driskell, J. E. (2002). Games, motivation, and learning: A research and practice model. *Simulation & gaming*, *33*(4), 441-467.

Gee, J. P. (2003). What Video Games Have To Teach Us About Learning And Literacy. *Computers in Entertainment, 1*(1), 20-20.

Gee, J. P. (2004). *Situated language and learning: A critique of traditional schooling*. Psychology Press.

Gee, J. P. (2008). Learning and games. *The ecology of games: Connecting youth, games, and learning*, *3*, 21-40.

Gomes, A., & Mendes, A. J. (2007, September). Learning to program-difficulties and solutions. *International Conference on Engineering Education–ICEE* (Vol. 2007).

Hannafin, M. J., & Hooper, S. R. (1993). Learning principles. *Instructional message design: Principles from the behavioral and cognitive sciences*, *2*, 191-227.

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design science in information systems research. *MIS quarterly*, *28*(1), 75-105.

Hirsjärvi, S., & Hurme, H. (2008). *Tutkimushaastattelu: teemahaastattelun teoria ja käytäntö*. Gaudeamus Helsinki University Press.

Hirsjärvi, S., Remes, P., & Sajavaara, P. (2009). Tutki ja kirjoita (15. uud. p. ed.). Helsinki: Tammi.

Hoonhout, J., Diederiks, E., & Stienstra, M. (2004). Designing fun, and test it too. *Usability Professionals' Association*.

Houser, R., & DeLoach, S. (1998). Learning from Games: Seven Principles of Effective Design. *Journal of the Society for Technical Communication*, *45*(3), 319-29.

Jenkins, T. (2002, August). On the difficulty of learning to program. *Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences* (Vol. 4, pp. 53-58).

Järvinen, P., & Järvinen, A. (1996). *Tutkimustyön metodeista*. Tampere: Opinpaja.

Keller, J. M. (1987). Development and use of the ARCS model of instructional design. *Journal of instructional development*, *10*(3), 2-10.

Kiili, K. (2005). Digital game-based learning: Towards an experiential gaming model. *The Internet and higher education*, *8*(1), 13-24.

Kolb, D. (1984). Experiential learning: Experience as the source of learning and development. Englewood Cliffs, N.J.: Prentice-Hall.

Kuljis, J., & P Baldwin, L. (2000). Visualisation techniques for learning and teaching programming. *CIT. Journal of computing and information technology*, *8*(4), 285-291.

Kurland, D. M., Pea, R. D., Clement, C., & Mawby, R. (1986). A study of the development of programming ability and thinking skills in high school students. *Journal of Educational Computing Research*, *2*(4), 429-458.

Lahtinen, E., Ala-Mutka, K., & Järvinen, H. M. (2005). A study of the difficulties of novice programmers. *ACM SIGCSE Bulletin*, *37*(3), 14-18.

Lewis, C., Esper, S., Bhattacharyya, V., Fa-Kaji, N., Dominguez, N., & Schlesinger, A. (2014). Children's perceptions of what counts as a programming language. *Journal of Computing Sciences in Colleges*, *29*(4), 123-133.

Linn, M. C., & Dalbey, J. (1989). Cognitive consequences of programming instruction. In E. Soloway & J. Spohrer (Eds.), *Studying the Novice Programmer* (pp. 57-81). Hillsdale, N.J.: Lawrence Erlbaum Associates.

Locke, E. A., & Latham, G. P. (1990). *A theory of goal setting & task performance*. Englewood Cliffs, N.J.: Prentice Hall.

Malone, T. W., & Lepper, M. R. (1987). Making learning fun: A taxonomy of intrinsic motivations for learning. *Aptitude, learning, and instruction*, *3*(1987), 223-253.

Malone, T. W., & Lepper, M. R. (1987). Making learning fun: A taxonomy of intrinsic motivations for learning. In R. E. Snow & M. J. Farr (Eds.), *Aptitude, learning, and instruction* (Vol. 3, pp. 223-253). Hillsdale, N.J.: Erlbaum.

Maloney, J. H., Peppler, K., Kafai, Y., Resnick, M., & Rusk, N. (2008). Programming by choice: urban youth learning programming with scratch. *ACM SIGCSE Bulletin*, *40*(1), 367-371.

Mayer, R. E. (1981). The psychology of how novices learn computer programming. *ACM Computing Surveys (CSUR)*, *13*(1), 121-141.

Milne, I., & Rowe, G. (2002). Difficulties in learning and teaching programming—views of students and tutors. *Education and Information technologies*, *7*(1), 55-66.

Murray, J. (1998). *Hamlet on the holodeck: The future of narrative in cyberspace*. Cambridge, Massachusetts: MIT Press.

Nakamura, J., & Csikszentmihalyi, M. (2002). The concept of flow. *Handbook of positive psychology*, 89-105.

Naps, T. L., Rößling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C., ... & Velázquez-Iturbide, J. Á. (2002). Exploring the role of visualization and engagement in computer science education. *ACM SIGCSE Bulletin*, *35*(2), 131-152.

Opetus- ja kulttuuriministeriö. (2014, 21. tammikuuta). Haettu 10.12.2014 osoitteesta http://www.minedu.fi/OPM/Tiedotteet/2014/01/Koodauskoulu.html

Papastergiou, M. (2009). Digital game-based learning in high school computer science education: Impact on educational effectiveness and student motivation. *Computers & Education*, *52*(1), 1-12.

Paras. B. & Bizzocchi, J. (2005). Game, motivation and effective learning: An integrated model for educational game design. *Proceedings of the Digital Games Research Association 2005 Conference: Changing Views - Worlds in Play*. Digital Games Research Association.

Perkins, D. N., Hancock, C., Hobbs, R., Martin, F., & Simmons, R. (1986). Conditions of learning in novice programmers. *Journal of Educational Computing Research*, *2*(1), 37-55.

Piteira, M., & Costa, C. (2012, June). Computer programming and novice programmers. *Proceedings of the Workshop on Information Systems and Design of Communication* (pp. 51-53). ACM.

Pokémon Omega Ruby and Pokémon Alpha Sapphire. (2015). Haettu 23.3.2015 osoitteesta http://www.pokemon.com/us/pokemon-video-games/pokemon-omega-ruby-and-pokemon-alpha-sapphire/

Prensky, M. (2005). Computer games and learning: Digital game-based learning. *Handbook of computer game studies*, *18*, 97-122.

Programming. (2015). *Oxford English Dictionary*. Haettu 6.4.2015 osoitteesta http://www.oed.com/view/Entry/152232

Randel, J. M., Morris, B. A., Wetzel, C. D., & Whitehill, B. V. (1992). The effectiveness of games for educational purposes: A review of recent research. *Simulation & gaming*, *23*(3), 261-276.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N.,f Eastmond, E., Brennan, K., ... & Kafai, Y. (2009). Scratch: programming for all. *Communications of the ACM*, *52*(11), 60-67.

Rieber, L. P. (1996). Seriously considering play: Designing interactive learning environments based on the blending of microworlds, simulations, and games. *Educational technology research and development*, *44*(2), 43-58.

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer Science Education*, *13*(2), 137-172.

Sajaniemi, J., & Kuittinen, M. (2003, June). Program animation based on the roles of variables. *Proceedings of the 2003 ACM symposium on Software visualization* (pp. 7-ff). ACM.

Salen, K., & Zimmerman, E. (2003). *Rules of play: Game design fundamentals*. Cambridge, Massachusetts: MIT Press.

Samurcay, R. (1989). Cognitive consequences of programming instruction. In E. Soloway & J. Spohrer (Eds.), *Studying the Novice Programmer* (pp. 161-178). Hillsdale, N.J.: Lawrence Erlbaum Associates.

Simon, H. A. (2000). Observations on the sciences of science learning. *Journal of Applied Developmental Psychology*, *21*(1), 115-121.

Spohrer, J. C., & Soloway, E. (1986). Novice mistakes: Are the folk wisdoms correct?. Communications of the ACM, 29(7), 624-632.

Tan, P. H., Ting, C. Y., & Ling, S. W. (2009, November). Learning difficulties in programming courses: undergraduates' perspective and perception. *International Conference on Computer Technology and Development, 2009*. (Vol. 1, pp. 42-46). IEEE.

Tuomi, J., & Sarajärvi, A. (2002). *Laadullinen tutkimus ja sisällönanalyysi*. Tammi.

Van Gorp, M. J., & Grissom, S. (2001). An empirical evaluation of using constructive classroom activities to teach introductory programming. *Computer Science Education*, *11*(3), 247-260.

Valli, R. (2001). Kyselylomaketutkimus. *Teoksessa J. Aaltola & R. Valli (toim.) Ikkunoita tutkimusmetodeihin*, *1*, 100-112.

Vilkka, H. (2005). *Tutki ja kehitä*. Helsinki: Tammi.

Von Mayrhauser, A., & Vans, A. M. (1995). Program comprehension during software maintenance and evolution. *Computer*, *28*(8), 44-55.

Wiedenbeck, S., Ramalingam, V., Sarasamma, S., & Corritore, C. (1999). A comparison of the comprehension of object-oriented and procedural programs by novice programmers. *Interacting with Computers*, *11*(3), 255-282.

Winslow, L. E. (1996). Programming pedagogy—a psychological overview. *ACM SIGCSE Bulletin*, *28*(3), 17-22.

Witmer, B. G., & Singer, M. J. (1994). Measuring immersion in virtual environments. *US Army Res. Inst., Alexandria, VA, Tech. Rep*, *1014*.

Liite A: Tutkimuslupakaavake

HYVÄT OPPILAIDEN HUOLTAJAT

Gradu

Hei, opiskelen tietojenkäsittelytieteiden laitoksella Oulun yliopistossa ja teen pro gradu -tutkielmaa nuorten ohjelmoinnin oppimisesta tietokonepelien avulla. Tavoitteenani on tutkia kuinka pelaamalla voitaisiin opettaa ohjelmoinnin peruskäsitteet ja kuinka ohjelmointiin liittyvä motivaatio, mielikuvat ja koettu vaikeus voivat muuttua pelien avulla. Tulen tekemään tutkimuskäynnin Pateniemen koululla maaliskuun aikana, ja haen yläkouluikäisiä osallistujia tutkimukseen.

Tutkimus

Tutkimuskäynnin aikana täytetään esikysely, jonka avulla tiedustellaan iän ja sukupuolen lisäksi taustatietoja tietokoneella pelaamisesta ja ohjelmoinnista. Tämän jälkeen pelataan tutkielmaa varten kehitettyä tietokonepeliä. Pelaamisen jälkeen täytetään loppukysely, jonka tarkoituksena on saada selville onko tapahtunut muutoksia tutkittavissa asioissa, eli ohjelmoinnin peruskäsitteiden osaamisessa ja niiden soveltamisessa, sekä muutoksissa ohjelmoinnin motivaatiossa, mielikuvissa ja koetussa vaikeudessa. Oppilaita ei arvostella tutkimuksen aikana, eivätkä saadut tulokset vaikuta kouluarvosanoihin.

Tutkimuksessa pelattava peli on kehitetty yläkoululaisille nuorille, eikä se sisällä kyseiselle ikäluokalle sopimattomaksi katsottuja teemoja.

Lupa

Tutkimuskäynti tullaan tekemään maaliskuussa ja pyydän teiltä lapsenne huoltajana lupaa siihen, että lapsenne voi osallistua kyseiseen tutkimukseen. Tässä tarkoituksessa pyydämme teitä oheisen kaavakkeen ja palauttamaan sen oppilaan mukana luokanopettajalle/ryhmänohjaajalle perjantaihin 27.2.2015 mennessä. Voitte milloin tahansa peruuttaa antamanne suostumuksen ilmoittamalla tästä tutkimuksen toteuttajalle tai koulun henkilökunnalle. Toivon myös, että keskustelette lapsenne kanssa asiasta ja annatte hänen itse päättää lopullisesti haluaako hän osallistua tutkimukseen, vaikka huoltajana antaisittekin suostumuksenne.

Kiittäen,
Aatu Harju

Lisätietoja kääntöpuolella ⇒>

- - - - ✂ -

Rastita ja palauta opettajalle tai ryhmänohjaajalle perjantaihin 27.2.2015 mennessä.

Huoltaja täyttää:	**Oppilas täyttää:**
☐ Lapsemme saa	☐ Olen halukas
☐ Lapsemme ei saa osallistua tutkimukseen	☐ En ole halukas osallistumaan tutkimukseen
Huoltajan allekirjoitus ja nimen selvennys:	Oppilaan allekirjoitus ja nimen selvennys:
_____	_____

Tutkimusaineiston käyttö, suojaaminen ja säilytys

Oulun yliopiston tietojenkäsittelytieteiden laitos vastaa tässä tutkimuksessa syntyvän empiirisen tutkimusaineiston säilytyksestä, käytöstä ja suojaamisesta. Tutkimusaineistoa tullaan hyödyntämään yksinomaan tieteellisessä tutkimuksessa (mukaan lukien opinnäytetyöt) ja opetuksessa, eikä sitä käytetä kaupallisiin tarkoituksiin. Aineiston keruussa voidaan käyttää hyödyksi videokuvausta, ja kaikki aineisto käsitellään luottamuksellisesti ja raportoidaan anonyymisti, joten lopullisissa aineiston pohjalta tehdyissä tuotoksissa ei osallistujia voida tunnistaa. Tutkimusaineisto voidaan myös arkistoida sellaisenaan pitempiaikaiseen käyttöön tutkimus- ja opetustyötä varten. Tällöin tieto arkistoidaan Oulun yliopiston tieto- ja sähkötekniikan tiedekunnan tietojenkäsittelytieteiden yksikössä.

Tutkimuksen toteuttaa Aatu Harju (<yhteystiedot poistettu>). Tutkielmaa ohjaavat professori Netta Iivari, FT (<yhteystiedot poistettu>) ja yliopistonlehtori Tonja Molin-Juustila (<yhteystiedot poistettu>). Ohjaajat vastaavat mielellään mahdollisiin tutkimusta koskeviin kysymyksiinne.

Liite B. Alkukysely

1. Taustatiedot

Tässä kysellään hieman kokemustasi pelien ja ohjelmoinnin parissa.

Nimi: _____

Ikä: _____

Sukupuoli:

□ Tyttö

□ Poika

Kuinka kiinnostunut olet tietokoneista?

En yhtään				Todella paljon
1	**2**	**3**	**4**	**5**

Kuinka paljon pidät peleistä?

En yhtään				Todella paljon
1	**2**	**3**	**4**	**5**

Kuinka usein pelaat pelejä tietokoneella, kännykällä tai nettiselaimella?

□ En koskaan

□ Useita kertoja kuukaudessa

□ Useita kertoja viikossa

□ Päivittäin

Mitä pelejä olet viime aikoina pelannut?

Kuinka paljon sinulla on kokemusta ohjelmoimisesta tietokoneella?

En yhtään				Todella paljon
1	2	3	4	5

Onko ohjelmointi sinusta vaikeaa vai helppoa?

Todella vaikeaa		En osaa sanoa		Todella helppoa
1	**2**	**3**	**4**	**5**

Onko ohjelmointi mielestäsi muodikasta tai "siistiä"?

Ei yhtään		En osaa sanoa		Todella siistiä
1	**2**	**3**	**4**	**5**

Mitä hyvää ohjelmoinnissa mielestäsi on?

Mitä huonoa ohjelmoinnissa mielestäsi on?

Haluaisitko opetella ohjelmointia KOULUSSA?

En yhtään		En osaa sanoa		Todella paljon
1	**2**	**3**	**4**	**5**

Haluaisitko opetella ohjelmointia ITSE?

En yhtään		En osaa sanoa		Todella paljon
1	**2**	**3**	**4**	**5**

Olisitko kiinnostunut ohjelmoimaan omia ohjelmia?

En yhtään		En osaa sanoa		Todella paljon
1	**2**	**3**	**4**	**5**

Haluaisitko opetella ohjelmointia pelejä pelaamalla?

En yhtään		En osaa sanoa		Todella paljon
1	**2**	**3**	**4**	**5**

Muista, että tämä ei ole koe, eivätkä vastauksesi vaikuta mitenkään kouluarvosanoihisi, eikä vastauksia näe edes opettajanne.

2. Ohjelmointi

Tässä katsotaan mitä tiedät ennalta ohjelmoinnista. Ei haittaa, jos et osaa vastata jokaiseen kysymykseen. Siinä tapauksessa voit rastittaa "En osaa vastata"-laatikon.

Tehtävä 1

Kirjoita haluamasi niminen muuttuja, jolle annat arvon **tiikeri**

□ En osaa vastata

Tehtävä 2

Kirjoita haluamasi niminen muuttuja, jolle annat arvoksi jonkin luvun

□ En osaa vastata

Tehtävä 3

Kirjoita alle koodi, jossa teet muuttujan nimeltään **piirakoitaSyoty.** Anna muuttujalle arvoksi jokin haluamasi desimaaliluku.

□ En osaa vastata

Tehtävä 4

Tee uusi muuttuja nimeltään **rahaaYhteensa**, jonka arvoksi tulee koodissa valmiina olevien muuttujien summa ja vähennät siitä vielä 15. Tee kaikki koodina, älä laske laskuja itse!

```
rahaaLompakossa  = 25

rahaaTililla  = 160
```

□ En osaa vastata

Tehtävä 5

Rastita mikä on muuttujan **katsomo** arvo koodin lopussa

□ 300

□ 0

□ “Aitio C”

□ 100

□ “Katsomo K”

□ “Alue G”

```
lipunHinta = 300


if (lipunHinta == 0)
{
        katsomo = “Aitio C”
}


else if (lipunHinta < 100)
{
        katsomo = “Katsomo K”
}


else
{
        katsomo = “Alue G”
}
```

□ En osaa vastata

Tehtävä 6

Rastita mikä on muuttujan **lautojaKasassa** arvo koodin lopussa

□ 0

□ 1

□ 2

□ 4

□ 7

```
lautojaKasassa = 0
nippuja = 0

while (lautojaKasassa < 7)
{
        lautojaKasassa = lautojaKasassa + 1

        if (lautojaKasassa == 4)
        {
                nippuja = nippuja + 1
        }
}
```

□ En osaa vastata

Tehtävä 7

Kirjoita alle koodi, jossa teet listan nimeltään **ostoslista**, josta löytyy kaksi keksimääsi ostoksen nimeä.

□ En osaa vastata

Tehtävä 8

Rastita millä seuraavista saisit muuttujan **happo** arvoksi “suolahappo”

□ happo = happolista

□ happo = happolista[]

□ happo = happolista[0]

□ happo = happolista[1]

□ happo = happolista.push(“suolahappo”)

```
happolista  = ["rikkihappo",  "suolahappo",  "typpihappo"]
```

□ En osaa vastata

Tehtävä 9

Rastita mitkä alla olevat väittämät pitävät paikkansa.

□ Muuttujan **ruokalaskuri** arvo on lopussa 4

□ Muuttujan **ruokalaskuri** arvo kuuluisi alussa olla 1

□ Muuttujan **valmistetutRuuat** arvo on lopussa 3

□ Koodi käy listan **ruokaLista** läpi

□ Muuttuja **ruoka** saa jossain vaiheessa arvokseen "lihapullat"

□ While käy niin kauan kuin muuttuja **ruokalaskuri** on suurempi kuin 4

□ Muuttujan **ruoka** arvo on lopussa "lohikeitto"

□ Muuttuja **ruoka** saa jossain vaiheessa arvokseen "maksalaatikko"

```
ruokaLista= ["lasagne", "pizza", "maksalaatikko", "lohikeitto"]


ruokalaskuri = 0
valmistetutRuuat = 0


while (ruokalaskuri < 4)
{
        ruoka = ruokaLista[ruokalaskuri]
        valmistetutRuuat = valmistetutRuuat + 1


        ruokalaskuri = ruokalaskuri + 1
}
```

□ En osaa vastata

Tehtävä 10

Rastita mikä on muuttujan **katsottava** arvo koodin lopussa

□ “Miten se toimii: vasara”

□ 15

□ video

□ “Smokahontasin ystävät”

□ “katsoVideo(15)”

□ 10

□ 12

□ “Söpöimmät kissanpennut”

□ return

□ “kellonaika”

```
function katsoVideo(kellonaika)
{
        if (kellonaika == 10)
        {
                video = “Smokahontasin ystävät”
        }
        else if (kellonaika > 12)
        {
                video = “Söpöimmät kissanpennut”
        }
        else
        {
                video = “Miten se toimii: vasara”
        }
        return video
```

```
}

katsottava = katsoVideo(15)
```

□ En osaa vastata

Liite C: Jälkikysely

Tämä ei ole koe, eivätkä vastauksesi vaikuta mitenkään kouluarvosanoihisi, eikä vastauksia näe edes opettajanne.

1. Ajatukset ohjelmoinnista

Tässä kysellään hieman kokemustasi pelien ja ohjelmoinnin parissa.

Onko ohjelmointi sinusta vaikeaa vai helppoa?

Todella vaikeaa		En osaa sanoa		Todella helppoa
1	**2**	**3**	**4**	**5**

Onko ohjelmointi mielestäsi muodikasta tai "siistiä"?

Ei yhtään		En osaa sanoa		Todella siistiä
1	**2**	**3**	**4**	**5**

Mitä hyvää ohjelmoinnissa mielestäsi on?

Mitä huonoa ohjelmoinnissa mielestäsi on?

Haluaisitko opetella ohjelmointia KOULUSSA?

En yhtään En osaa sanoa Todella paljon

1 2 3 4 5

Haluaisitko opetella ohjelmointia ITSE?

En yhtään En osaa sanoa Todella paljon

1 2 3 4 5

Olisitko kiinnostunut ohjelmoimaan omia ohjelmia?

En yhtään En osaa sanoa Todella paljon

1 2 3 4 5

Haluaisitko opetella ohjelmointia pelejä pelaamalla?

En yhtään En osaa sanoa Todella paljon

1 2 3 4 5

Muista, että tämä ei ole koe, eivätkä vastauksesi vaikuta mitenkään kouluarvosanoihisi, eikä vastauksia näe edes opettajanne.

2. Ohjelmointi

Tässä katsotaan mitä tiedät ennalta ohjelmoinnista. Ei haittaa, jos et osaa vastata jokaiseen kysymykseen. Siinä tapauksessa voit rastittaa "En osaa vastata"-laatikon.

Tehtävä 1

Kirjoita haluamasi niminen muuttuja, jolle annat arvon **tiikeri**

□ En osaa vastata

Tehtävä 2

Kirjoita haluamasi niminen muuttuja, jolle annat arvoksi jonkin luvun

□ En osaa vastata

Tehtävä 3

Kirjoita alle koodi, jossa teet muuttujan nimeltään **piirakoitaSyoty.** Anna muuttujalle arvoksi jokin haluamasi desimaaliluku.

□ En osaa vastata

Tehtävä 4

Tee uusi muuttuja nimeltään **rahaaYhteensa**, jonka arvoksi tulee koodissa valmiina olevien muuttujien summa ja vähennät siitä vielä 15. Tee kaikki koodina, älä laske laskuja itse!

```
rahaaLompakossa  = 25

rahaaTililla  = 160
```

□ En osaa vastata

Tehtävä 5

Rastita mikä on muuttujan **katsomo** arvo koodin lopussa

□ 300

□ 0

□ “Aitio C”

□ 100

□ “Katsomo K”

□ “Alue G”

```
lipunHinta = 300


if (lipunHinta == 0)
{
        katsomo = “Aitio C”
}


else if (lipunHinta < 100)
{
        katsomo = “Katsomo K”
}


else
{
        katsomo = “Alue G”
}
```

□ En osaa vastata

Tehtävä 6

Rastita mikä on muuttujan **lautojaKasassa** arvo koodin lopussa

□ 0

□ 1

□ 2

□ 4

□ 7

```
lautojaKasassa = 0
nippuja = 0

while (lautojaKasassa < 7)
{
        lautojaKasassa = lautojaKasassa + 1

        if (lautojaKasassa == 4)
        {
                nippuja = nippuja + 1
        }
}
```

□ En osaa vastata

Tehtävä 7

Kirjoita alle koodi, jossa teet listan nimeltään **ostoslista**, josta löytyy kaksi keksimääsi ostoksen nimeä.

□ En osaa vastata

Tehtävä 8

Rastita millä seuraavista saisit muuttujan **happo** arvoksi “suolahappo”

□ happo = happolista

□ happo = happolista[]

□ happo = happolista[0]

□ happo = happolista[1]

□ happo = happolista.push(“suolahappo”)

```
happolista  = ["rikkihappo",  "suolahappo",  "typpihappo"]
```

□ En osaa vastata

Tehtävä 9

Rastita mitkä alla olevat väittämät pitävät paikkansa.

□ Muuttujan **ruokalaskuri** arvo on lopussa 4

□ Muuttujan **ruokalaskuri** arvo kuuluisi alussa olla 1

□ Muuttujan **valmistetutRuuat** arvo on lopussa 3

□ Koodi käy listan **ruokaLista** läpi

□ Muuttuja **ruoka** saa jossain vaiheessa arvokseen "lihapullat"

□ While käy niin kauan kuin muuttuja **ruokalaskuri** on suurempi kuin 4

□ Muuttujan **ruoka** arvo on lopussa "lohikeitto"

□ Muuttuja **ruoka** saa jossain vaiheessa arvokseen "maksalaatikko"

```
ruokaLista= ["lasagne", "pizza", "maksalaatikko", "lohikeitto"]


ruokalaskuri = 0
valmistetutRuuat = 0


while (ruokalaskuri < 4)
{
        ruoka = ruokaLista[ruokalaskuri]
        valmistetutRuuat = valmistetutRuuat + 1


        ruokalaskuri = ruokalaskuri + 1
}
```

□ En osaa vastata

Tehtävä 10

Rastita mikä on muuttujan **katsottava** arvo koodin lopussa

□ “Miten se toimii: vasara”

□ 15

□ video

□ “Smokahontasin ystävät”

□ “katsoVideo(15)”

□ 10

□ 12

□ “Söpöimmät kissanpennut”

□ return

□ “kellonaika”

```
function katsoVideo(kellonaika)
{
        if (kellonaika == 10)
        {
                video = “Smokahontasin ystävät”
        }
        else if (kellonaika > 12)
        {
                video = “Söpöimmät kissanpennut”
        }
        else
        {
                video = “Miten se toimii: vasara”
        }
        return video
```

```
}

katsottava = katsoVideo(15)
```

□ En osaa vastata

3. Ajatukset tehtävistä ja pelistä

Tässä kysellään ajatuksiasi ohjelmoinnista ja pelaamastasi pelistä.

Olivatko edelliset tehtävät helppoja vai vaikeita?

Todella vaikeita En osaa sanoa Todella helppoja

1 2 3 4 5

Mikä oli helppoa edellisissä ohjelmointitehtävissä?

Mikä oli vaikeaa edellisissä ohjelmointitehtävissä?

Oliko pelaamasi peli kiinnostava?

Ei yhtään En osaa sanoa Todella kiinnostava

1 2 3 4 5

Oliko peli mukava?

Ei yhtään En osaa sanoa Todella mukava

1 2 3 4 5

Oliko peli mukaansatempaava?

Ei yhtään		En osaa sanoa		Todella mukaansatempaava
1	**2**	**3**	**4**	**5**

Oliko pelin idea helppo ymmärtää?

Todella vaikea		En osaa sanoa		Todella helppo
1	**2**	**3**	**4**	**5**

Oliko pelissä hyvät grafiikat?

Ei yhtään		En osaa sanoa		Todella hyvät
1	**2**	**3**	**4**	**5**

Oliko ohjelmointia helppo ymmärtää pelin avulla?

Todella vaikea		En osaa sanoa		Todella helppo
1	**2**	**3**	**4**	**5**

Oliko pelin tehtäviä helppo ratkaista?

Todella vaikea		En osaa sanoa		Todella helppo
1	**2**	**3**	**4**	**5**

Tuntuiko peli hyödylliseltä?

Ei yhtään		En osaa sanoa		Todella hyödylliseltä
1	**2**	**3**	**4**	**5**

Tuntuiko peli edistävän ohjelmoinnin oppimistasi?

Ei yhtään		En osaa sanoa		Todella paljon
1	**2**	**3**	**4**	**5**