Pinja Turunen

Lapset ja ohjelmointi: lapsille tarkoitetut ohjelmointiympäristöt ja ohjelmoinnillisen ajattelun kehittyminen

Tietotekniikan kandidaatintutkielma

30. huhtikuuta 2019

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Pinja Turunen

Yhteystiedot: pinkartu@student.jyu.fi

Ohjaaja: Antti-Jussi Lakanen

Työn nimi: Lapset ja ohjelmointi: lapsille tarkoitetut ohjelmointiympäristöt ja ohjelmoinnillisen ajattelun kehittyminen

Title in English: Children and Programming: Programming environments for children and the development of computational thinking

Työ: Kandidaatintutkielma

Sivumäärä: 24+0

Tiivistelmä: Lapsia varten on tehty erilaisia ohjelmointiympäristöjä, jotka auttavat heitä oppimaan kaikille hyödyllistä ohjelmoinnillista ajattelua. Ohjelmoinnilliseen ajatteluun kuuluu erilaisia ohjelmoinnin konsepteja ja sitä voidaan hyödyntää esimerkiksi ongelmanratkaisuun. Lapsille tarkoitettuja ohjelmointiympäristöjä voidaan jakaa erilaisiin luokkiin. Tässä tutkielmassa lapsille tarkoitettuja ohjelmointiympäristöjä on jaettu ensin sen mukaan, onko niiden päätavoitteena opettaa ohjelmointia vai onko ohjelmoinnin oppiminen sivutuote jonkin muun tavoitteen edistämiselle. Näistä päätavoitteelliset eli opettavat järjestelmät jaetaan eteenpäin ohjelmakoodin päämuodon mukaan tekstuaalisiin, visuaalisiin, tekstuaalisgraafisiin, esittäviin, konkreettisiin sekä valikollisiin ja arvollisiin ohjelmointiympäristöihin. Sivutavoitteellisissa eli voimaannuttavissa ohjelmointiympäristöissä sen sijaan hyödynnetään monimuotoisempia tapoja harjoitella ohjelmointiin liittyviä konsepteja joko käyttämällä vaihtoehtoisia tapoja ohjelmoida tai parantelemalla muita aktiviteetteja ohjelmoinnin avulla.

Avainsanat: lapset, ohjelmointi, ohjelmointiympäristö, ohjelmoinnillinen ajattelu

Abstract: Different kinds of programming environments have been made for children that help them learn computational thinking, which is a useful skill for everyone. Computational thinking consists of different programming concepts and it is used, for example, in problem solving. Programming environments for children can be categorized into different groups. In this paper, programming environments for children have first been separated based on

whether learning to program is the main goal or is learning to program a side product of advancing another goal. Those with learning to program as their main goal, which are called teaching systems, are further categorized based on the main form of coding into textual, visual, textual-graphical, tangible, menu-based and value-based programming environments. Those with learning to program as their secondary goal, which are called empowering systems, use more diverse ways to practice programming concepts by either using alternative ways of programming or enhancing other activities with programming.

Keywords: children, programming, programming environment, computational thinking

Sisältö

1 Johdanto

Tämän tutkielman aiheena ovat lapset ja ohjelmointi. Teknologian merkityksen kasvaessa vuosi vuodelta niin ohjelmointitaidon osaamisen kuin ylipäätään siihen liittyvän ohjelmoinnillisen ajattelun voidaan uskoa kasvattavan merkitystään sen mukana. Pysyäksemme teknologian kehityksen tahdissa voisi näiden taitojen opettamisen yhteisesti peruskoulussa nähdä olevan tärkeää, jotta mahdollisimman moni osaisi ohjelmoida ja hallitsisi ohjelmoinnillisen ajattelun konsepteja.

Jotta lapsille voitaisiin opettaa ohjelmointia, tulee kuitenkin ottaa huomioon ja ymmärtää minkälaiset tekijät ohjelmointikielissä ja -ympäristöissä auttavat lapsia oppimaan niin ohjelmointia kuin ohjelmoinnillista ajattelua sekä motivoitumaan näiden opiskeluun. Tässä tutkielmassa käydään läpi miten lapsille suunnattuja ohjelmointiympäristöjä voidaan jakaa antaen esimerkkejä olemassaolevista ohjelmointiympäristöistä sekä käydään läpi niille yleisiä ominaisuuksia.

Lapsiin ja ohjelmointiin liittyvää tieteellistä kirjallisuutta löytyy laidasta laitaan. Aiheesta on kirjoitettu hyvin monista erilaisista näkökulmista ja aiheen eri puoliin liittyen. Hyödynsin kirjallisuuskatsauksessani lähteenä esimerkiksi erilaisista yksittäisistä lapsille tarkoitetuista ohjelmointikielistä ja -ympäristöistä sekä niiden jaotteluista kirjoitettuja artikkeleita. Tarkastelin myös kirjallisuutta liittyen lapsille ohjelmoinnin opetukseen sekä ohjelmoinnillisen ajattelun kehittymiseen.

Luvussa 2 käydään läpi lasten ja ohjelmoinnin aiheeseen olennaisesti liittyvä käsite ohjelmoinnillinen ajattelu, mitä se tarkoittaa, minkälaisia konsepteja siihen liittyy ja minkä takia sen osaaminen voidaan nähdään hyödyllisenä taitona kaikille. Luvussa tuodaan esiin myös käsitteeseen liittyvää epäyhdenmukaisuutta sekä kritiikkiä sen laajaan opettamiseen kohdistuen.

Luvussa 3 käydään läpi useita esimerkkejä siitä, minkälaisia lapsille suunnattuja ohjelmointiympäristöjä on olemassa. Luvussa pohditaan miten näitä erilaisia ohjelmointiympäristöjä voidaan luokitella antaen samalla esimerkkejä luokkiin sopivista ohjelmointiympäristöistä ja niiden ominaisuuksista. Lopuksi yhdistellään esimerkeistä korostuneita ajatuksia sekä kirjal-

lisuudesta löytyvää tietoa lapsille suunnatuissa ohjelmointiympäristöissä usein esiintyvistä yleisistä ominaisuuksista.

Luku 4 on tutkielman yhteenveto, jossa käydään tiiviisti läpi ohjelmoinnillisen ajattelun käsitteen merkitys, tutkielmassa käytetty lapsille tarkoitettujen ohjelmointiympäristöjen jako sekä näille ympäristöille yleisiä ominaisuuksia.

2 Ohjelmoinnillinen ajattelu

Ohjelmoinnin oppimiseen kuuluu olennaisesti ohjelmoinnillisen tai laskennallisen ajattelun käsite, jota kutsutaan englanniksi nimellä *computational thinking*, sekä sen oppiminen (Kafai ja Burke 2014, s. 23). Wing (2006, s. 33) pitää ohjelmoinnillista ajattelua hyödyllisenä taitona kaikille. Ohjelmoinnillisen ajattelun tarkkaan merkitykseen liittyy kuitenkin sekavuutta siitä tehtyjen erilaisten määritelmien takia (Grover ja Pea 2013, s. 38). Kuitenkin siihen voidaan katsoa liittyvän erilaiset ohjelmointiin liittyvät konseptit, kuten seuraavat määritelmät antavat ymmärtää:

Wingin (2006, s. 35) mukaan ohjelmoinnillisen ajattelun käsitteellä tarkoitetaan ongelmanratkaisuun käytettävää ajattelutapaa, johon kuuluu niin matemaattista kuin insinööriajattelua, sekä abstraktioajattelua. Siihen liittyy myös muun muassa taito ymmärtää ihmisten käytöstä sekä kyky ajatella virheitä ja niistä selviämistä (Wing 2006, s. 33–34).

Rose, Habgood ja Jay (2017, s. 298–299) ovat tutkineet ohjelmoinnilliseen ajatteluun liittyviä konsepteja ja huomanneet, että heidän käyttämissään lähteissä esiintyy paljon samankaltaisia konsepteja, mutta myös useita eroavaisuuksia löytyy. Näiden perusteella he (2017, s. 299) kokosivat, että ohjelmoinnilliseen ajatteluun kuuluvat esimerkiksi virheiden havaitseminen ja korjaaminen, erilaisten rakenteiden tunnistaminen ja käyttäminen, abstrahointi ja yleistäminen, algoritmit eli moniaskeliset toimintaohjeet sekä samanaikaisuus.

Ohjelmoinnillisen ajattelun oppimiseen käytettävät työkalut, kuten ohjelmointiympäristöt, auttavat oppimaan vähintäänkin osaa ohjelmoinnilliseen ajatteluun liittyvistä konsepteista (Grover ja Pea 2013, s. 41). Wing (2006, s. 33) sekä Grover ja Pea (2013, s. 40) pitävät tärkeänä, että kaikki osaisivat ohjelmoinnillista ajattelua. Ohjelmoinnillisen ajattelun laajemman osaamisen saavuttamiseksi voidaan hyödyntää esimerkiksi sen opettamista kaikille peruskoulussa.

Grover ja Pea (2013, s. 40) tuovat esiin myös vastapuolen ohjelmoinnillisen ajattelun opettamisen tuomisesta koulujen opetussuunnitelmiin erittelemällä annettua kritiikkiä. Ohjelmoinnillisen ajattelun opettaminen kaikille oppilaille ei vaikuta tarpeelliselta kaikkien mielestä, ja toisaalta osa näkee ohjelmoinnillisen ajattelun olevan tarpeeksi samanlaista esimerkiksi

matemaattisen ajattelun kanssa, jonka takia sitä ei tarvitsisi opettaa erikseen (Grover ja Pea 2013, s. 40).

Riippumatta siitä, haluavatko kaikki hyödyntää ammatillisesti ohjelmointia, sen konseptien ymmärtäminen yleisellä tasolla vaikuttaa hyödylliselle taidolle vähintäänkin ongelmanratkaisussa. Ohjelmoinnillisen ajattelun opettaminen ja sen mahdolliset hyödyt kaipaavat kuitenkin lisää tutkimusta.

3 Lapsille suunnatut ohjelmointiympäristöt

Lapsia varten on suunniteltu erilaisia heitä oppimaan innostavia ja auttavia ohjelmointikieliä ja -ympäristöjä. Käsitteet ohjelmointikieli ja ohjelmointiympäristö kulkevat usein käsi kädessä etenkin lapsille suunnatuissa tuotteissa, sillä monesti tiettyä ohjelmointikieltä on tarkoitus ohjelmoida nimenomaan sille tarkoitetussa ohjelmointiympäristössä. Tällöin sekä ohjelmointikieltä että sille tarkoitettua ympäristöä usein kutsutaan samalla nimellä. Tässä tutkielmassa käytän pääasiassa käsitettä ohjelmointiympäristö, vaikka jokin asia voikin liittyä ohjelmointiympäristöä oleellisemmin keskeiseen ohjelmointikieleen.

Kirjallisuudessa ohjelmointiympäristöjä on jaoteltu eri tavoin esimerkiksi sen perusteella, miten niitä käytetään tai minkälaisiin asioihin niitä luodessa on keskitytty. Kelleher ja Pausch (2005, s. 84) luokittelevat aloittelijoille tarkoitettuja ympäristöjä sen perusteella, onko niiden pää- vai sivutavoitteena ohjelmointi. Tämä liittyy olennaisesti lapsille tarkoitettujen ohjelmointiympäristöjen aiheeseen, sillä myös ohjelmointia opettelevat lapset kuuluvat ohjelmoinnin aloittelijoihin.

Yksittäisistä ohjelmointiympäristöistä kertovien artikkelien perusteella voidaan nähdä, että ohjelmointiympäristöjä kuvailtaessa niissä usein mainitaan ohjelmakoodin luomisen päämuoto. Näistä yleisimpiä ovat tekstuaaliset ja visuaaliset ohjelmointiympäristöt, mutta näiden yhdistelmää, eli tekstuaalisgraafisia ohjelmointiympäristöjä, esiintyy myös vaihtelevin nimityksin. Ohjelmakoodin luomisen päämuodon jaotteluun voidaan nähdä kuuluvan myös konkreettisen ohjelmoinnin ympäristöt.

Louca ja Zacharia (2008, s. 290–291) luokittelevat nuorille suunnattuihin tietokonepohjaisiin ohjelmointiympäristöihin tekstuaalisen, animoidun, kolmiulotteisen, visuaalisen ja graafisen ohjelmoinnin ympäristöt. Tämä jaottelu on melko lähellä edellistä muutamia eroavaisuuksia lukuunottamatta.

Kelleher ja Pausch (2005, s. 127) kuvaavat aloittelijoille suunnattujen järjestelmien ominaisuuksia, joista yksi kategoria on ohjelmien luomisen tapa. He mainitsevat tähän kuuluvan kirjoittamisen, graafisten tai fyysisten objektien yhdistelemisen, toimintatavan esittämisen, valikot tai arvojen syöttämisen (Kelleher ja Pausch 2005, s. 127). Yhdistellen tätä jaotte-

lua edellisiin voidaan kutsua ohjelmointiympäristöjä, joissa ohjelmia luodaan kirjoittamalla tekstuaalisiksi, graafisia objekteja yhdistäviä visuaalisiksi ja fyysisiä objekteja yhdistäviä konkreettisiksi ohjelmointiympäristöiksi.

Nämä erilaiset jaottelut sopivat mielestäni yhteen niin, että päätavoitteellisen ohjelmoinnin ympäristöihin, eli opettaviin järjestelmiin (Kelleher ja Pausch 2005, s. 84), kuuluvat yllä mainitut eri ohjelmien luomisen muodon luokat. Toisaalta sivutavoitteelliset ohjelmointiympäristöt, eli voimaannuttavat järjestelmät, pitävät sisällään muunlaisia ohjelmointia sisältäviä, mutta ei siihen perinteisellä tavalla pääasiallisesti painottuvia ympäristöjä, jotka voidaan jakaa vaihtoehtoisia ohjelmoinnin tapoja hyödyntäviin ympäristöihin sekä ohjelmoinnilla paranneltuihin aktiviteetteihin (Kelleher ja Pausch 2005, s. 109–125).

3.1 Ohjelmoinnin oppiminen ympäristössä päätavoitteena

Kelleher ja Pausch (2005, s. 84) kutsuvat ohjelmointiympäristöjä, joissa ohjelmointi on päätavoitteena, nimellä opettavat järjestelmät (engl. *teaching systems*). Jos ympäristön päätavoitteena on ohjelmointi, se tarkoittaa, että sitä käytettäessä tavoitellaan pääasiallisesti ohjelmoinnin oppimista (Kelleher ja Pausch 2005, s. 84). Kelleherin ja Pauschin (2005, s. 84) mukaan opettavissa järjestelmissä on tärkeää, että ohjelmoinnin aloittaminen olisi helppoa, mutta myös tavoitellaan taidon yleistettävyyttä perinteisempiin ohjelmointiympäristöihin, jotka eivät ole suunniteltu aloittelijoita varten.

Kelleher ja Pausch (2005, s. 84) jakavat opettavia järjestelmiä sen perusteella, mihin niiden suunnittelussa on keskitytty. Ensin he tekevät jaon ohjelmoinnissa käytettävien mekanismien ja opetuksen tuen prioriteetin suhteen (Kelleher ja Pausch 2005, s. 85–86). Opetuksen tuen ryhmän he jakavat eteenpäin sosiaalisen oppimisen ja motivoivan kontekstin edistämisen priorisointiin (Kelleher ja Pausch 2005, s. 106–107). Suurin osa opettavista järjestelmistä painottuu kuitenkin ohjelmoinnissa käytettyihin mekanismeihin, joiden perusteella Kelleher ja Pausch (2005, s. 86–105) jakavat näitä ohjelman ilmaisun, rakenteen ja ohjelman ajamisen perusteella.

Yksittäisistä lapsille suunnatuista opettavista ohjelmointiympäristöistä tehdyissä artikkeleissa korostuu Kelleherin ja Pauschin perusteellisemman jaottelun sijaan ajatus siitä, että jako

ohjelmakoodin päämuodon perusteella on oleellinen ero opettavien järjestelmien välillä. Jako ohjelmakoodin päämuodon perusteella voidaan kuitenkin nähdä sopivan yhteen Kelleherin ja Pauschin (2005, s. 127) esittämän ohjelmointiympäristöjen ominaisuuden, ohjelmien luontitavan, kategorian alle. Seuraavaksi käyn tämän perusteella tehtävän jaottelun tekstuaalisiin, visuaalisiin, tekstuaalisgraafisiin, esittäviin, valikollisiin ja arvollisiin, sekä konkreettisiin ohjelmointiympäristöihin.

3.1.1 Tekstuaaliset ohjelmointiympäristöt

Tekstuaaliset ohjelmointiympäristöt ovat ohjelmointiympäristöjä, joissa ohjelmia luodaan kirjoittamalla ohjelmakoodia tekstimuodossa (Kelleher ja Pausch 2005, s. 127). Yksi esimerkki aloittelijoille tarkoitetuista tekstuaalisista ohjelmointikielistä on Processing:

Tsukamoto ym. (2015, s. 3) pitävät tekstuaalisen ohjelmointikielen opettelua hyvänä harjoitteluna todellisiin sovelluksiin. He (2015, s. 3) hyödynsivät Processing-nimistä tekstuaalista ohjelmointikieltä. Processingissa on kuitenkin mukana myös visuaalisuutta ohjelman tulosteen muodossa (Tsukamoto ym. 2015, s. 3), joten siinäkin hyödynnetään lapsille suunnatuille ohjelmointiympäristöille tyypillistä visuaalisuutta omalla tavallansa.

Vaikka tekstuaaliset ohjelmointiympäristöt ovat kaikista ohjelmointiympäristöistä yleisimpiä, lapsille suunnatuissa ohjelmointiympäristöissä tekstuaalisia ohjelmointiympäristöjä esiintyy vähemmän kuin visuaalisia ohjelmointiympäristöjä.

3.1.2 Visuaaliset ohjelmointiympäristöt

Visuaaliset ohjelmointiympäristöt ovat järjestelmiä, joissa hyödynnetään pääasiallisesti hiiren käyttöä ohjelmointiin, sillä niille on tyypillistä esimerkiksi elementtien yhdistäminen toisiinsa. Ohjelmakoodia luodaan pääasiallisesti ohjelmakoodia edustavia elementtejä raahaamalla kirjoittamisen sijaan (Kelleher ja Pausch 2005, s. 127). Visuaalisista ohjelmointiympäristöistä esimerkkeinä toimivat Kodu, Scratch ja Alice 2:

Maclaurinin (2011, s. 241) luoma Kodu-ohjelmointikieli ja siihen integroituna kuuluva 3D-peliohjelmointiympäristö ovat luova työkalu nimenomaan pelien tekemiseen. Siinä tuleekin

mukana editorit sekä fysiikat ynnä muut valmiiksi, jotta lapset voisivat keskittyä itse pelien tekemiseen (Maclaurin 2011, s. 241–242).

Scratch on hyvin tunnettu visuaalinen ohjelmointikieli ja -ympäristö, jossa ohjelmointi tapahtuu erilaisia palikoita yhdistäen (Sáez-López, Román-González ja Vázquez-Cano 2016, s. 4). Scratchin käyttö pyritty saamaan intuitiiviseksi (Sáez-López, Román-González ja Vázquez-Cano 2016, s. 4) ja helpoksi, jotta oppiminen voisi tapahtua kokeilun kautta ilman aiempaa ohjelmointikokemusta (Maloney ym. 2010, s. 3). Scratch-ympäristössä on esillä palikoista koostuvan ohjelmakoodin osio ”Script”, osio visuaalisille elementeille eli ”Sprite” sekä ”Stage”, joka toimii näyttämönä siitä mitä ohjelmassa tapahtuu (Lee 2011, s. 27–28).

Alice on helpoksi ja kiinnostavaksi luotu työkalu kolmiulotteisten ympäristöjen ja animaatioiden tekemiseen (Cooper, Dann ja Pausch 2000, s. 3–4), jonka pohjalta on kehitetty Alice 2 (Kelleher ym. 2002, s. 1). Alicen ensimmäisessä versiossa kirjoitettiin ohjelmakoodia (Cooper, Dann ja Pausch 2000, s. 3), mutta Alice 2 kehitettiin Alicesta saadun palautteen myötä käyttämään kirjoittamisen sijaan elementtien raahamista, joka estää syntaksivirheitä muodostumasta (Kelleher ym. 2002, s. 1). Kirjoittamisesta luopumisen takia Alice 2 voidaan nähdä kuuluvan visuaalisiin ohjelmointiympäristöihin.

Louca ja Zacharia pitävät visuaalisia ja graafisia ohjelmointiympäristöjä erillisinä, mutta esimerkiksi Cross ym. (2013, s. 1–7) kutsuvat Scratchia visuaaliseksi ja graafiseksi ohjelmointiympäristöksi. Näiden kahden käsitteen käyttö toistensa vastineena vaikuttaa kirjallisuudessa yleisemmälle kuin niiden pitäminen erillisinä, joten tässä tutkielmassa olen päättänyt lukea molemmat osaksi visuaalisten ohjelmointiympäristöjen kategoriaa.

3.1.3 Tekstuaalisgraafiset ohjelmointiympäristöt

Tekstuaalisgraafiset ohjelmointiympäristöt ovat suurelta osin samanlaisia kuin graafisten objektien yhdistelyä käyttävät visuaaliset ohjelmointiympäristöt, mutta niissä olennaista on myös mahdollisuus tarkastella ohjelmakoodia tekstimuodossa, kuten tekstuaalisissa ohjelmointiympäristöissä. Näin niissä yhdistyy molemmissa esiintyviä ominaisuuksia. Tekstuaalisgraafisista ohjelmointiympäristöistä esimerkkeinä toimivat Flip ja BrickLayer:

Flip-ohjelmointikieli on kaksimuotoinen: vaikka kieli on visuaalinen koostuessaan yhdis-

teltävistä graafisista palikoista, kuten esimerkiksi Scratchissa, tämän lisäksi ohjelmakoodi kirjoittuu esille myös luonnolliselle kielelle palikoita käsiteltäessä (Howland ja Good 2015, s. 229).

Ennen Flipiä Cheung ym. (2009, s. 276) ovat luoneet BrickLayer-ohjelmointiympäristön, jossa on Flipin tapaan käytössä sekä tekstimuotoinen ohjelmakoodi että graafiset ohjelmakoodipalikat. Cheung ym. (2009, s. 276) käyttävät kahdenlaista nimitystä nämä kaksi asiaa yhdistävistä ohjelmointiympäristöistä: ”tekstuaalisgraafinen hybridiympäristö” (engl. *textual-graphical hybrid environment*) sekä ”tekstiparanneltu graafinen ohjelmointiympäristö” (engl. *text-enhanced graphical programming environment*).

3.1.4 Esittävät ohjelmointiympäristöt

Kelleherin ja Pauschin (2005, s. 127) mukaan on olemassa opettavia järjestelmiä, joissa ohjelmia luodaan esittämällä miten niiden tulee toimia. Tällaisia järjestelmiä voidaan kutsua esittäviksi ohjelmointiympäristöiksi.

Esimerkkinä esittävistä ohjelmointiympäristöistä toimii ToonTalk, jonka avulla ohjelmointia voi oppia hauskalla tavalla pelimäisessä ohjelmointiympäristössä ohjaten omaa ohjelmoijahahmoaan ja olemalla vuorovaikutuksessa muiden pelimaailman olioiden kanssa (Kahn 1996, s. 3–4). ToonTalk on luotu hyvin ilmaisuvoimaiseksi sekä helpoksi oppia käyttämään itsenäisesti (Kahn 1996, s. 3–4).

3.1.5 Valikolliset ja arvolliset ohjelmointiympäristöt

Opettavia ohjelmointiympäristöjä, joissa ohjelmia luodaan joko valikkojen tai arvojen syöttämisen kautta, voidaan kutsua valikollisiksi ja arvollisiksi ohjelmointiympäristöiksi (Kelleher ja Pausch 2005, s. 127). LEGOsheets on yksi tähän kategoriaan sopivista ohjelmointiympäristöistä. LEGOsheets on sääntöihin pohjautuva ohjelmointiympäristö, jossa käyttäjä luo ohjelmia asettamalla eri arvoja, kuten alkuarvoja tai prioriteetteja, erilaisissa sääntöjen muokkausikkunoissa (Gindling ym. 1995, s. 1–4).

3.1.6 Konkreettiset ohjelmointiympäristöt

Muiden opettavien ohjelmointiympäristöjen rinnalle on luotu myös mahdollisuuksia konkreettiseen ohjelmointiin. Konkreettisessa ohjelmoinnissa käytetään ohjelmakoodin luomiseen fyysisiä palikoita (Kelleher ja Pausch 2005, s. 127), jotka voidaan jakaa elektronisiin ja ei-elektronisiin palikoihin (Wang, Zhang ja Wang 2011, s. 127). Tämä jako vaikuttaa siihen, miten ohjelmakoodi viedään palikoista eteenpäin ajettavaksi (Wang, Zhang ja Wang 2011, s. 127).

Palikoiden määrän ja käytössä olevan tilan rajoitukset huomioiden konkreettinen ohjelmointi on kuitenkin usein hyvin rajoitettua ohjelmakoodin pituuden suhteen. Tämä antaa ymmärtää, että konkreettisen ohjelmoinnin avulla voi harjoitella ohjelmointia, mutta laajempien ohjelmien toteuttaminen voi käytännössä olla haastavaa. Tämän takia konkreettiset ohjelmointiympäristöt voidaan nähdä myös osana voimaannuttavia järjestelmiä.

Esimerkkeinä konkreettisista ohjelmointiympäristöistä toimivat seuraavaksi kuvailtavat AlgoBlock, Tern sekä PROTEAS-sarja:

Suzuki ja Kato (1993, s. 1–2) tavoittelivat tapaa ohjelmoida yhdessä, ilman että hiiri ja näppäimistö rajoittavat yhteistyön tekemistä vuorottelun takia. Siksi he (1993, s. 2) loivat AlgoBlockin, jossa fyysisiä palikoita käyttäen ohjataan virtuaalista sukellusvenettä.

Tern-ohjelmointiympäristö hyödyntää puisia palikoita konkreettiseen ohjelmointiin (Horn ja Jacob 2007, s. 1966). Hornin ja Jacobin (2007, s. 1966) mukaan Ternissä palikat luetaan tietokoneelle ei-elektronisesti kameraa hyödyntäen.

Sapounidis, Demetriadis ja Stamelos (2015, s. 225–228) hyödynsivät tutkimuksessaan PROTEAS-sarjaa, jonka mukana tulevilla konkreettiseen ohjelmointiin tarkoitetuilla palikoilla (T-ProRob-alasysteemi) voidaan antaa toimintaohjeita robotille. Ohjelmakoodin lukeminen palikoista tapahtuu elektronisesti joko kaapelin tai Bluetoothin avulla (Sapounidis, Demetriadis ja Stamelos 2015, s. 228). Konkreettisen ohjelmointijärjestelmän rinnalle on tehty myös visuaalinen ohjelmointiympäristö saavuttamaan sama asia ilman fyysisiä palikoita (Sapounidis, Demetriadis ja Stamelos 2015, s. 228).

3.2 Ohjelmoinnin oppiminen ympäristössä sivutavoitteena

Kelleher ja Pausch (2005, s. 84, 109) kutsuvat sellaisia ohjelmointiympäristöjä, joissa ohjelmointi on sivutavoitteena, voimaannuttaviksi järjestelmiksi (engl. *empowering systems*). Jos ympäristön sivutavoitteena on ohjelmointi, tällöin tavoitellaan jotain muuta, jossa tarvitaan myös ohjelmointia (Kelleher ja Pausch 2005, s. 84). Ohjelmoinnin oppiminen tapahtuu siis sivutuotteena päätavoitteelle. Tällaisiksi voidaan luokitella esimerkiksi ohjelmointipelit, joissa lapsi pääasiallisesti pelaa peliä, mutta voi samalla oppia ohjelmointiin liittyviä konsepteja.

Kelleherin ja Pauschin (2005, s. 109–125) mukaan voimaannuttaviin järjestelmiin kuuluvat sellaiset ohjelmointiympäristöt, joissa käytetään perinteisestä poikkeavia tapoja ohjelmointiin tai muita aktiviteetteja parannellaan ohjelmoinnin avulla. Ne tarjoavat vaihtoehtoisia tapoja luoda ohjelmia rajoittamatta niitä opettavissa järjestelmissä olevalla tavoitteella taidon yleistettävyydestä perinteiseen ohjelmointiin (Kelleher ja Pausch 2005, s. 84, 109). Näin voimaannuttavat järjestelmät voivat olla huomattavasti opettavia järjestelmiä monimuotoisempia, kun samankaltaisuutta perinteisten ohjelmointiympäristöjen kanssa ei tarvitse tavoitella. Kuitenkin voimaannuttavien järjestelmien ominaisuuksia voi löytyä myös lapsille tarkoitetuista opettavista järjestelmistä, joten ohjelmointiympäristöjen jakaminen opettaviin ja voimaannuttaviin järjestelmiin ei ole täysin yksiselitteistä.

3.2.1 Vaihtoehtoiset tavat ohjelmoida

Voimaannuttavien järjestelmien taustalla on ajatus siitä, että ohjelmakoodin ymmärtäminen ja sen luominen vastaamaan omia tavoitteita voi olla haastavaa (Kelleher ja Pausch 2005, s. 109). Qian ja Lehman (2017, s. 17) uskovat lasten ja muiden ohjelmoinnin aloittelijoiden aiemman tietämyksen olevan merkittävässä roolissa ohjelmointiin liittyvissä vaikeuksissa. Ongelmia esiintyy usein liittyen syntaksiin, konsepteihin ja strategioihin (Qian ja Lehman 2017, s. 17).

Ohjelmakoodiin liittyvistä vaikeuksista irti päästäkseen voimaannuttavat järjestelmät voivat hyödyntää erilaisia tapoja luoda ohjelmia kuin kirjoittamalla varsinaista ohjelmakoodia (Kelleher ja Pausch 2005, s. 109). Kelleher ja Pausch (2005, s. 109–112) tuovat esiin kolme eri-

laista tapaa tähän: 1) käyttäjä näyttää ohjelmalle mitä tulee tehdä, 2) käyttäjä määrää tilanteet missä toimia ja minkälaisen lopputuloksen hän haluaa tai 3) käyttäjä valitsee mitä ohjelma tekee.

Voimaannuttavissa järjestelmissä voidaan myös tavoitella ohjelmointikielten parantamista aloittelijoita ajatellen. Kelleher ja Pausch (2005, s. 113) mainitsevat mahdollisiksi parantamista kaipaaviksi tekijöiksi kielen ymmärrettävyyden, vuorovaikutuksen ja ympäristön integraation:

Kielen ymmärrettävyyttä paranneltaessa pyrkimyksenä on valita mahdollisimman ymmärrettäviä termejä ja tapoja ilmaista asioita, jotta ohjelmointiympäristön käyttö olisi kielellisestä näkökulmasta mahdollisimman selkeää (Kelleher ja Pausch 2005, s. 113). Vuorovaikutuksen parantamisella tavoitellaan, että ohjelmien tekeminen olisi helpompaa hyödyntäen erilaisia tapoja, jotka pyrkivät virheiden vähyyteen (Kelleher ja Pausch 2005, s. 116). Viimeinen Kelleherin ja Pauschin (2005, s. 113) mainitsemista asioista on ympäristön integraation parantaminen, jossa pyrkimyksenä on yhdistää ohjelman luomis- sekä ajoympäristö niin, ettei ohjelmaa tarvitse erikseen ajaa. Vastaavia ominaisuuksia voidaan kuitenkin nähdä myös lapsille suunnatuissa opettavissa järjestelmissä.

3.2.2 Aktiviteettien parantaminen ohjelmoinnilla

Voimaannuttaviin järjestelmiin luetaan mukaan myös aktiviteetit, joita on paranneltu ohjelmoinnin avulla. Näihin luetaan niin viihde- kuin opetuskäyttöön tarkoitetut aktiviteetit (Kelleher ja Pausch 2005, s. 122–125).

Ohjelmointipelit toimivat yhtenä esimerkkinä voimaannuttaviin järjestelmiin kuuluvista aktiviteeteista, joita on paranneltu ohjelmoinnin avulla. Lapset voivat oppia ohjelmointia ja siihen liittyviä konsepteja pelien pelaamisen kautta. Pelit voivat auttaa heitä motivaation ylläpidossa, ja samalla he voivat oppia uutta. Lapsia varten on luotu erilaisia pelejä, joita pelaamalla he voivat oppia ohjelmointia tai ohjelmoinnillista ajattelua, kuten esimerkiksi PlayLOGO 3D sekä CTArcade:

PlayLOGO 3D on ympäristö, jonka avulla voidaan harjoitella LOGO-komentoja ja –syntaksia kahden vastakkaisen pelaajan ongelmanratkaisupelin avulla (Paliokas, Arapidis ja Mpimpit-

sos 2011, s. 25–26).

CTArcade on ohjelmoinnillisen ajattelun kehittämiseen tarkoitettu opetuspeliympäristö (Lee ym. 2014, s. 26–27). Lee ym. (2014, s. 28) mukaan siihen kuuluu muun muassa ristinollapeli, jossa lapsi opettaa omaa hahmoaan pelaamaan peliä, antaen sille sääntöjä noudatettavaksi. Lopulta lapsi voi myös pelata omaa hahmoaan vastaan (Lee ym. 2014, s. 28).

3.3 Yleisiä ominaisuuksia

Ohjelmointikielissä ja -ympäristöissä, jotka ovat lapsille suunnattuja, on ominaisuuksia, jotka esiintyvät monissa niissä. Kuitenkin eroavaisuuksiakin löytyy, eikä kaikkia näistä ominaisuuksista löydy jokaisesta lapsille tarkoitetusta ohjelmointiympäristöstä.

Kuten yllä esiteltyjen ohjelmointikielten ja -ympäristöjen kuvauksista tulee ilmi, monet lapsille suunnatuista ohjelmointiympäristöistä ovat visuaalispainotteisia. Tämä näkyy useinmiten ohjelmakoodin esityksen muodossa, mutta myös muita tapoja visuaaliseen esitykseen on, kuten tulosteen visuaalisuus Processing nimisessä tekstuaalisessa ohjelmointiympäristössä.

Päätavoitteelliseen ohjelmointiin tarkoitetut ympäristöt, eli opettavat järjestelmät, usein sisältävät erilaisia graafisia elementtejä, kuten ohjelmakoodipalikoita, joita yhdistellen itse koodia kirjoitetaan. Nämä voivat olla myös fyysisiä palikoita konkreettisessa ohjelmoinnissa. Tekstuaalisia ohjelmointiympäristöjä hyödynnetään myös lapsille ohjelmoinnin opettamisessa, jotta lapset oppivat minkälaista ohjelmointi yleensä on (Tsukamoto ym. 2015, s. 2). Tekstuaalisgraafisissa ohjelmointiympäristöissä ohjelmakoodi on graafisten palikoiden lisäksi esitetty myös tekstuaalisena. Cheung ym. (2009, s. 276) mukaan tekstuaalisen ja graafisen puolen yhdistäminen auttaa välivaiheessa täysin visuaalisesta pelkästään tekstuaaliseen ohjelmointikieleen. Näiden lisäksi on olemassa myös esittäviä ohjelmointiympäristöjä, joissa käyttäjä luo ohjelmia esittämällä ohjelman haluttua toimintaa, sekä valikollisia ja arvollisia ohjelmointiympäristöjä, joissa joko valikkojen tai arvojen syöttämisen kautta luodaan ohjelmia.

Opettavien järjestelmien lisäksi on luotu myös ohjelmointiympäristöjä, joissa ohjelmointi on sivutavoitteena. Näitä kutsutaan voimaannuttaviksi järjestelmiksi. Ne voivat olla opet-

tavia järjestelmiä monimuotoisempia toteutuksessaan, sillä ne ovat vähemmän rajoittuneita ohjelmointitaidon yleistettävyyden suhteen. Näihin voidaan ajatella kuuluvan esimerkiksi ohjelmointipelit, joissa lapsen odotetaan innostuvan itse pelaamisesta ja oppivan sen oheella ohjelmointia tai ohjelmoinnillista ajattelua. Voimaannuttavissa järjestelmissä voidaan hyödyntää vaihtoehtoisia tapoja ohjelmointiin, kuten käyttää erilaisia tapoja ilmaista ohjelman toimintaa kuin ohjelmakoodi tai parannella kielen ymmärrettävyyttä, vuorovaikutusta tai ympäristön integraatiota, tai parannella muita aktiviteetteja ohjelmoinnilla. Kuitenkin näitä erilaisia voimaannuttavien järjestelmien ominaisuuksia voi myös löytyä opettaviin järjestelmiin luokiteltavista ohjelmointiympäristöistä, joten raja näiden välillä ei ole täysin selkeä.

Lapsille tarkoitetuissa ohjelmointiympäristöissä käyttäjän ei usein tarvitse erikseen kääntää ohjelmakoodia, vaan ohjelmakoodi ajetaan välittömästi (Fessakis, Gouli ja Mavroudi 2013, s. 88), kuten esimerkiksi Scratchissa (Maloney ym. 2010, s. 4). Tämä on myös yksi Kelleherin ja Pauschin mainitsemista voimaannuttaviin järjestelmiin liittyvistä ominaisuuksista. Myös syntaksin yksinkertaisuus ja syntaksivirheiden estäminen, esimerkiksi palikoiden muodon avulla (Maloney ym. 2010, s. 7) tai hyödyntäen valikoita (Kelleher ym. 2002, s. 1), ovat melko yleisiä ominaisuuksia (Fessakis, Gouli ja Mavroudi 2013, s. 88).

Lapsille suunnatuissa ohjelmointiympäristöissä korostuu ajatus siitä, että yksinkertaisten ohjelmien tekeminen itsenäisen kokeilun kautta olisi mahdollisimman helppoa, mikä on oleellista myös voimaannuttavissa järjestelmissä, mutta ohjelmointikielen ilmaisuvoima riittää kuitenkin myös monimutkaisempiinkin asioihin, jotta kokemuksen karttuessa raja ei tulisi niin nopeasti vastaan (Grover ja Pea 2013, s. 40).

4 Yhteenveto

Ohjelmoinnillinen ajattelu on kaikille hyödyllinen taito, jota voi hyödyntää muun muassa ongelmanratkaisuun. Ohjelmoinnillisen ajattelun käsite ei ole yksiselitteinen, mutta yleisesti siihen ajatellaan kuuluvan ohjelmoinnissa käytettäviä konsepteja, kuten esimerkiksi erilaisten rakenteiden tunnistaminen ja hyödyntäminen, algoritmit, abstraktio ja niin edelleen. Väestöllisesti laajan osaamisen takaamiseksi ohjelmoinnillisen ajattelun opettaminen peruskoulussa voidaan nähdä hyödyllisenä, mutta siitä on myös esitetty kritiikkiä vedoten laajan osaamisen tarpeettomuuteen ja ohjelmoinnillisen ajattelun samankaltaisuuteen muun kehittyvän ajattelun kanssa. Laajan opetuksen mahdollisten hyötyjen kartottamiseksi tarvitaan lisää tutkimusta.

Ohjelmointiympäristöjä hyödyntäen lapset voivat oppia ohjelmoinnillista ajattelua. Lapsille tarkoitettuja ohjelmointiympäristöjä on luokiteltu kirjallisuudessa eri tavoin. Tässä tekstissä lapsille suunnattuja ohjelmointiympäristöjä on luokiteltu ensin sen perusteella, onko ohjelmoinnin oppiminen ympäristössä pää- vai sivutavoitteena. Tämän jälkeen päätavoitteelliset ympäristöt, eli opettavat järjestelmät, on jaettu eteenpäin ohjelmakoodin päämuodon perusteella tekstuaalisiin, visuaalisiin, tekstuaalisgraafisiin, esittäviin, valikollisiin ja arvollisiin sekä konkreettisiin ohjelmointiympäristöihin. Sivutavoitteellisiin ympäristöihin, eli voimaannuttaviin järjestelmiin, luokitellaan ohjelmointiympäristöjä, joissa hyödynnetään vaihtoehtoisia tapoja ohjelmointiin tai parannellaan muita aktiviteetteja ohjelmoinnilla.

Opettaviin järjestelmiin kuuluvissa tekstuaalisissa ohjelmointiympäristöissä ohjelmakoodia kirjoitetaan tekstimuodossa. Sen sijaan visuaalisissa ohjelmointiympäristöissä ohjelmakoodi esitetään hyödyntäen visuaalisia elementtejä, kuten ohjelmakoodipalikoita, joita yhdistämällä ohjelmakoodia kirjoitetaan. Tekstuaalisgraafiset ohjelmointiympäristöt ovat välimuoto visuaalisten ja tekstuaalisten ohjelmointiympäristöjen välillä, sillä niissä ohjelmakoodia voidaan kirjoittaa niin tekstimuodossa kuin palikoita yhdistäen. Esittävissä ohjelmointiympäristöissä käyttäjä luo ohjelmia esittämällä ohjelman toimintaa. Valikollisissa ja arvollisissa ohjelmointiympäristöissä ohjelmia luodaan valikkojen tai arvojen syöttämisen avulla. Näiden lisäksi on luotu visuaalisten ohjelmointiympäristöjen kaltaisia konkreettisia ohjelmointiympäristöjä, joissa fyysisiä palikoita yhdistäen luodaan ohjelmakoodia.

Lapsille suunnattuja ohjelmointiympäristöjä on erilaisia, mutta niissä usein korostuu samanlaisia ominaisuuksia. Niissä on tyypillisesti visuaalisia elementtejä esimerkiksi ohjelmakoodipalikoiden tai ohjelman tulosteen muodossa. Usein ohjelmaa ei tarvitse erikseen ajaa, vaan ohjelma on koko ajan käynnissä samassa ympäristössä, sekä syntaksia on yksinkertaistettu. Syntaksivirheitä on voitu jopa estää esimerkiksi palikoiden muotojen avulla, jolloin toisiinsa sopimattomia koodin paloja ei pysty liittämään toisiinsa. Lapsille suunnatuissa ohjelmointiympäristöissä ohjelmien luominen on pyritty saamaan mahdollisimman helpoksi, mutta ilmaisuvoima on pyritty nostamaan korkealle, jotta kokeneempikaan käyttäjä ei koe niitä liian rajoittuneiksi.

Lähteet

Cheung, Joey C.Y., Grace Ngai, Stephen C.F. Chan ja Winnie W.Y. Lau. 2009. “Filling the Gap in Programming Instruction: A Text-enhanced Graphical Programming Environment for Junior High Students”. Teoksessa *Proceedings of the 40th ACM Technical Symposium on Computer Science Education,* 276–280. SIGCSE ’09. Chattanooga, TN, USA: ACM. ISBN: 978-1-60558-183-5. doi:10.1145/1508865.1508968.

Cooper, Stephen, Wanda Dann ja Randy Pausch. 2000. “Alice: a 3-D tool for introductory programming concepts”. *Journal of Computing Sciences in Colleges* 15 (5): 107–116. http://cse.unl.edu/~scooper/alice/ccscne00.PDF.

Cross, J., C. Bartley, E. Hamner ja I. Nourbakhsh. 2013. “A visual robot-programming environment for multidisciplinary education”. Teoksessa *2013 IEEE International Conference on Robotics and Automation,* 445–452. doi:10.1109/ICRA.2013.6630613.

Fessakis, G., E. Gouli ja E. Mavroudi. 2013. “Problem solving by 5–6 years old kindergarten children in a computer programming environment: A case study”. *Computers & Education* 63:87–97. ISSN: 0360-1315. doi:10.1016/j.compedu.2012.11.016.

Gindling, J., A. Ioannidou, J. Loh, O. Lokkebo ja A. Repenning. 1995. “LEGOsheets: a rule-based programming, simulation and manipulation environment for the LEGO Programmable Brick”. Teoksessa *Proceedings of Symposium on Visual Languages,* 172–179. doi:10.1109/VL.1995.520806.

Grover, Shuchi, ja Roy Pea. 2013. “Computational Thinking in K–12: A Review of the State of the Field”. *Educational Researcher* 42 (1): 38–43. doi:10.3102/0013189X12463051.

Horn, Michael S., ja Robert J. K. Jacob. 2007. “Tangible Programming in the Classroom with Tern”. Teoksessa *CHI ’07 Extended Abstracts on Human Factors in Computing Systems,* 1965–1970. CHI EA ’07. San Jose, CA, USA: ACM. ISBN: 978-1-59593-642-4. doi:10.1145/1240866.1240933.

Howland, Kate, ja Judith Good. 2015. “Learning to communicate computationally with Flip: A bi-modal programming language for game creation”. *Computers & Education* 80:224–240. ISSN: 0360-1315. doi:10.1016/j.compedu.2014.08.014.

Kafai, Yasmin, ja Quinn Burke. 2014. *Connected code : why children need to learn programming*. Cambridge, Ma.: The MIT Press.

Kahn, Ken. 1996. “ToonTalkTM—An Animated Programming Environment for Children”. *Journal of Visual Languages & Computing* 7 (2): 197–217. ISSN: 1045-926X. doi:10.1006/jvlc.1996.0011.

Kelleher, Caitlin, Dennis Cosgrove, David Culyba, Clifton Forlines, Jason Pratt ja Randy Pausch. 2002. “Alice2: programming without syntax errors”. Teoksessa *User Interface Software and Technology*. Volyymi 2. 2. Citeseer. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.60.4640&rep=rep1&type=pdf.

Kelleher, Caitlin, ja Randy Pausch. 2005. “Lowering the Barriers to Programming: A Taxonomy of Programming Environments and Languages for Novice Programmers”. *ACM Comput. Surv.* (New York, NY, USA) 37, numero 2 (): 83–137. ISSN: 0360-0300. doi:10.1145/1089733.1089734.

Lee, Tak Yeon, Matthew Louis Mauriello, June Ahn ja Benjamin B. Bederson. 2014. “CTArcade: Computational thinking with games in school age children”. *International Journal of Child-Computer Interaction* 2 (1): 26–33. ISSN: 2212-8689. https://doi.org/10.1016/j.ijcci.2014.06.003.

Lee, Young-Jin. 2011. “Scratch: Multimedia Programming Environment for Young Gifted Learners”. *Gifted Child Today* 34 (2): 26–31. doi:10.1177/107621751103400208.

Louca, Loucas T., ja Zacharia C. Zacharia. 2008. “The Use of Computerbased Programming Environments as Computer Modelling Tools in Early Science Education: The cases of textual and graphical program languages”. *International Journal of Science Education* 30 (3): 287–323. doi:10.1080/09500690601188620.

Maclaurin, Matthew. 2011. “The Design of Kodu: A Tiny Visual Programming Language for Children on the Xbox 360”. *SIGPLAN Not.* (New York, NY, USA) 46, numero 1 (): 241–246. ISSN: 0362-1340. doi:10.1145/1925844.1926413.

Maloney, John, Mitchel Resnick, Natalie Rusk, Brian Silverman ja Evelyn Eastmond. 2010. “The Scratch Programming Language and Environment”. *Trans. Comput. Educ.* (New York, NY, USA) 10, numero 4 (): 16:1–16:15. ISSN: 1946-6226. doi:10.1145/1868358.1868363.

Paliokas, I., C. Arapidis ja M. Mpimpitsos. 2011. “PlayLOGO 3D: A 3D Interactive Video Game for Early Programming Education: Let LOGO Be a Game”. Teoksessa *2011 Third International Conference on Games and Virtual Worlds for Serious Applications,* 24–31. doi:10.1109/VS-GAMES.2011.10.

Qian, Yizhou, ja James Lehman. 2017. “Students’ Misconceptions and Other Difficulties in Introductory Programming: A Literature Review”. *ACM Trans. Comput. Educ.* (New York, NY, USA) 18, numero 1 (): 1:1–1:24. ISSN: 1946-6226. doi:10.1145/3077618.

Rose, Simon P., M. P. J. Habgood ja Tim Jay. 2017. “An Exploration of the Role of Visual Programming Tools in the Development of Young Children’s Computational Thinking”. *Electronic Journal of E-Learning* 15, numero 4 (): 297–309. http://www.ejel.org/volume15/issue4/p297.

Sáez-López, José-Manuel, Marcos Román-González ja Esteban Vázquez-Cano. 2016. “Visual programming languages integrated across the curriculum in elementary school: A two year case study using “Scratch” in five schools”. *Computers & Education* 97:129–141. ISSN: 0360-1315. https://doi.org/10.1016/j.compedu.2016.03.003.

Sapounidis, Theodosios, Stavros Demetriadis ja Ioannis Stamelos. 2015. “Evaluating Children Performance with Graphical and Tangible Robot Programming Tools”. *Personal Ubiquitous Comput.* (London, UK, UK) 19, numero 1 (): 225–237. ISSN: 1617-4909. doi:10.1007/s00779-014-0774-3.

Suzuki, Hideyuki, ja Hiroshi Kato. 1993. “AlgoBlock: a tangible programming language, a tool for collaborative learning”. Teoksessa *Proceedings of 4th European Logo Conference,* 297–303. https://www.researchgate.net/profile/Hideyuki_Suzuki5/publication/242383829_Algoblock_a_tangible_programming_language_a_tool_for_collaborative_learning/links/575f5c8e08ae414b8e5496e3/Algoblock-a-tangible-programming-language-a-tool-for-collaborative-learning.pdf.

Tsukamoto, H., Y. Takemura, H. Nagumo, I. Ikeda, A. Monden ja K. Matsumoto. 2015. “Programming education for primary school children using a textual programming language”. Teoksessa *2015 IEEE Frontiers in Education Conference (FIE),* 1–7. doi:10.1109/FIE.2015.7344187.

Wang, Danli, Cheng Zhang ja Hongan Wang. 2011. “T-Maze: A Tangible Programming Tool for Children”. Teoksessa *Proceedings of the 10th International Conference on Interaction Design and Children,* 127–135. IDC ’11. Ann Arbor, Michigan: ACM. ISBN: 978-1-4503-0751-2. doi:10.1145/1999030.1999045.

Wing, Jeannette M. 2006. “Computational Thinking”. *Commun. ACM* (New York, NY, USA) 49, numero 3 (): 33–35. ISSN: 0001-0782. doi:10.1145/1118178.1118215.