TURUN YLIOPISTO

Ohjelmoinnin oppiminen -

digitaaliset oppimisjärjestelmät oppimisen apuna

Tietojenkäsittelytiede
Tietotekniikan laitos, Teknillinen tiedekunta
Kandidaatintutkielma

Marika Parviainen
Syyskuu 2024

Kandidaatintutkielma
Tietotekniikan laitos, Teknillinen tiedekunta
Turun yliopisto

Tutkinto-ohjelma: Tietojenkäsittelytiede
Tekijä: Marika Parviainen
Otsikko: Ohjelmoinnin oppiminen - digitaaliset oppimisjärjestelmät oppimisen apuna
Sivumäärä: xx sivua, yy liitesivua
Päivämäärä: Syyskuu 2024

Ohjelmoinnin oppimisen haasteet ovat jo pitkään olleet didaktisen tietojenkäsittelytieteen keskeinen tutkimuskohde. Ohjelmistokehityksen ammattilaisista on jatkuvasti pulaa, ja monilla ihmisillä on vielä epämääräinen käsitys teknologian roolista arjessaan. Ohjelmoinnin osaaminen avaa tien paitsi tietokoneiden toimintalogiikan ymmärtämiseen, myös laajempaan käsitykseen siitä, miten teknologiavetoinen yhteiskuntamme toimii. Nämä ovat edellytyksenä uusien teknologioiden tehokkaalle soveltamiselle.

Ohjelmointi on monimutkainen taito, joka koostuu useista osa-alueista ja jossa harjoittelun merkitys on keskeinen. Interaktiiviset oppimisalustat tarjoavat arvokkaan tuen ohjelmoinnin opiskeluun, ja ne voivat merkittävästi rikastuttaa oppimiskokemusta. Kuten kaikkien pedagogisten työkalujen kohdalla, myös ohjelmoinnin oppimiseen tarkoitettujen järjestelmien käytössä tarvitaan opettajan asiantuntemusta – niin opetuksen suunnittelussa kuin oppimisprosessin ohjaamisessa.

Työn tutkimuskysymykset liittyvät digitaalisen oppimisen keskeisiin oppimista tukeviin elementteihin ja siihen onko niitä hyödynnetty kouluissa yleisesti käytetyissä järjestelmissä. Kirjallisuuskatsauksen tulosten perusteella todetaan, että visualisointi, automaattinen arviointi ja välitön palaute luovat perustan aktiivista oppimista tukeville ympäristöille, joita pelillistäminen tehokkaasti täydentää motivoivalla tavalla. Työssä tutkituissa järjestelmissä esiintyy kaikkia tutkimuksessa löydettyjä elementtejä. Tutkielma ja siinä esitellyt tulokset auttavat opettajia ohjelmointikursseilla käytettävän oppimisjärjestelmän valinnassa.

Asiasanat: ohjelmointi, oppiminen, opetus, visualisointi, arviointi, palaute, pelillistäminen

Sisällysluettelo

1 Johdanto

Nykyisessä opetussuunnitelmassa (Perusopetuksen opetussuunnitelman perusteet, 2014) ohjelmointia opetetaan peruskoulussa kaikilla luokka-asteilla, osana matematiikan ja osin käsityön opetusta. Ohjelmointia lähestytään toiminnallisesti ja alakoulussa käytetään visuaalisia ohjelmointikieliä. Yläkoulussa tavoitteena on tekstipohjaisen ohjelmointikielen käyttö matemaattisen ongelmanratkaisun apuna. Opetus- ja kulttuuriministeriön Uudet lukutaidot -hankkeessa (Uudet lukutaidot -kehittämisohjelma, 2022) kuvataan myös ohjelmointiosaamisen sisältöjä yksityiskohtaisemmin.

Ohjelmointitaidon osaamisen kautta tulevaisuuden aikuiset oppivat koulussa ymmärtämään miten jokapäiväinen teknologia toimii. On vaikea keksiä toimenkuvaa, jossa ei olisi etua teknologian paremmasta hallitsemisesta. Ohjelmoinnin oppimista oheen on viime vuosina noussut kysymys ohjelmoinnillisen ajattelun roolista.

Yliopistotason ohjelmointikursseja ovat piinanneet opiskelijoiden korkeina pysyvät keskeyttämisprosentit ja ohjelmoinnin oppimisen vaikeutta on tuskailtu tuhansissa tutkimuksissa. Onko ohjelmointitaidon opetuksen aikaistaminen ratkaisu korkeakouluopiskelijoiden puutteellisiin taitoihin – vai toistuvatko samat vaikeudet nyt peruskouluissa?

Oppimistuloksiin vaikuttaa monia tekijöitä – suuria, kuten oppilaiden ennakkotiedot ja motivaatio, sekä pienempiä, kuten aikatauluhaasteet. Näistä vain osaan opettajalla on mahdollisuus vaikuttaa. Oppimisympäristön valinta on merkittävä ratkaisu, joka pitää tehdä jo ennen kurssin alkua, eikä oppimisympäristöä voi kesken kurssin muuttaa. Opettaja voi usein itse vaikuttaa käytettävän oppimisympäristön valintaan. Jotta opettaja voisi valita kurssilleen parhaiten sopivan oppimisympäristön, hänen on tärkeää ymmärtää, miten eri oppimisympäristöt eroavat toisistaan ja mitä ominaisuuksia ne tarjoavat. Tämä kokonaiskuva auttaa tekemään harkittuja päätöksiä, jotka tukevat oppilaiden oppimista parhaalla mahdollisella tavalla.

1.1 Tutkimuskysymykset ja menetelmät

Tutkielman kontekstina ovat ohjelmoinnin oppimisen haasteet. Tutkielma voi osaltaan auttaa opettajaa valitsemaan yläkoulun tekstipohjaisen ohjelmoinnin opetuksen tueksi soveltuvan järjestelmän. Tutkielman tarkoitus tiivistyy tutkimuskysymyksiksi:

TK1 Mitä ohjelmoinnin oppimista tukevia menetelmiä on olemassa?

TK2 Missä laajuudessa ohjelmoinnin oppimista tukevia menetelmiä on käytetty tarkasteltavaksi valituissa järjestelmissä?

Tutkielma pyrkii vastaamaan ensimmäiseen kysymykseen kirjallisuuskatsauksen pohjalta. Tiedonhakuun on käytetty päälähteenä Turun yliopiston Volter-tietokantaa, josta tietoa on haettu avainsanojen avulla. Tulokset on rajattu vain suomen- ja englanninkielisiin artikkeleihin ja kirjoihin. Erityistä aikarajausta ei käytetty, mutta suosittiin viimeaikaisia ja usein siteerattuja lähteitä. Taulukossa 1 esitettyjen avainsanojen lisäksi hakua tarkennettiin osa-alueittain tutkielman edistyessä esiintulleiden käsitteiden avulla.

Taulukko 1 Kirjallisuuskatsauksessa käytetyt hakusanat

Programming	AND	Learning
Bloom taxonomy	AND/OR	Programming
Notional machine	AND/OR	Learning

Jälkimmäisen tutkimuskysymyksen tarkasteluun on valittu kolme yleisesti käytössä olevaa, suomenkielistä, ilmaista järjestelmää. Valinta on tehty kirjallisuuskatsauksessa saatuun mielikuvaan ja kollegoiden kanssa käytyihin keskusteluihin tukeutuen, mikä lienee tutkielman laajuuteen nähden riittävä perustelu.

1.2 Tutkielman rakenne

Tutkielma rakentuu viidestä luvusta. Ensimmäinen luku johdattelee aiheeseen ja esittelee tutkimuskysymykset ja -menetelmät sekä kuvailee tutkielman rakenteen. Toisessa luvussa kuvataan ohjelmoinnin oppimisen haasteita ja niihin esitettyjä ratkaisuja. Kolmannessa luvussa kuvataan tarkemmin kirjallisuuskatsauksessa löydetyt oppimista tehostavat menetelmät. Neljännessä luvussa esitellään ohjelmoinnin opetuksessa käytettyjä järjestelmiä. Luvut kaksi ja kolme liittyvät ensimmäiseen tutkimuskysymykseen ja luku neljä jälkimmäiseen tutkimuskysymykseen. Viidennessä luvussa kootaan yhteen tutkielman tulokset ja pohditaan tulosten merkitystä.

2 Ohjelmoinnin oppiminen

Tässä luvussa tarkastellaan ohjelmoinnin oppimista alkaen oppimisen teoriasta ja päätyen ohjelmoinnin apuna käytettäviin oppimisjärjestelmiin. Ohjelmoinnin oppiminen on tunnetusti vaikeaa, sillä ohjelmointi on monitahoinen taito. Oppimisen edistyessä opiskelijan käsitys koodin ja tietokoneen toiminnasta tarkentuu, ja opiskelija pystyy soveltamaan oppimaansa uusiin tilanteisiin. Opettajan tulee huomioida ohjelmoinnin oppimisprosessin erityispiirteet jo oppimistavoitteita asetettaessa ja oppimisympäristöä valitessa.

2.1 Tärkeää – mutta vaikeaa

Ohjelmointi on tärkeä osa tietojenkäsittelytieteiden opiskelua mutta opiskelijat kokevat sen oppimisen vaikeaksi (McCracken ym., 2001). Korkeakouluopintojen ensimmäisten ohjelmointikurssien oppimistulokset eivät ole hyviä, kertovat monet tutkimukset (Simon ym., 2006).

Ohjelmoinnin oppimista vaikeuttaa myös ohjelmoinnin luonne: ohjelmointitaito koostuu useista erilaisista osista. Hyvältä ohjelmoijalta vaaditaan käsitteiden, strategioiden ja käytännön taitojen yhdistämistä. Du Boulay (1989) esittää viisi osittain limittyvää ohjelmointitaidon osa-aluetta:

1. Yleinen perehtyminen: oppija ymmärtää ohjelman käsitteen ja tietää sen tarkoituksen.
2. Käsitteellinen järjestelmä: oppija ymmärtää järjestelmän toiminnan suhteessa ohjelmien suoritukseen.
3. Notaatio: oppija osaa käyttää kielen syntaksia ja kontrollirakenteita ohjelmien toteutuksessa.
4. Rakenteet: oppija kehittää tarvittavat taidot osaongelmien (ja näin ollen edelleen koko ongelman) ratkaisemiseksi.
5. Käytännöt: oppija kehittää tarvittavat tiedot ja taidot ohjelmien suunnittelua, toteutusta, testausta ja virheiden korjausta varten.

Noviisi oppii osataidot vähitellen harjoituksen myötä. Koska samanaikaisesti on käsiteltävä toisiinsa limittyviä taitoalueita, koetaan harjoittelu työlääksi. Ohjelmoinnin oppisessa ja opettamisessa työskennellään abstraktin tuotoksen, ohjelmakoodin kanssa. Noviisin on vaikea

ymmärtää, miksi jokin ohjelmakoodi toimii halutulla tavalla ja toinen koodi tuottaa väärän tuloksen. Pelkän toimivan koodin näkeminen ei vielä auta aloittelevaa ohjelmoijaa ymmärtämään miten ohjelma toimii, sillä monet oleelliset osat (ohjelmakirjastot, kääntäjän toiminta) jäävät piiloon. Oppimisprosessin ymmärtämiseksi ja oppijan tukemiseksi pitää ymmärtää mikä käsitys oppijalla on ohjelman toiminnasta.

2.2 Mielikuva tietokoneen ja ohjelman toiminnasta

Oppimisprosessin myötä opiskelija liittää uuden tiedon osaksi aiempia tietojaan. Ohjelmointia opiskellessaan opiskelija joutuu luomaan mielikuvan tietokoneen rakenteesta ja ohjelman suorituksesta. Mielikuvan puutteet aiheuttavat opiskelijoille tyypillisiä ohjelmointivirheitä. Esimerkiksi opiskelija saattaa olettaa, että tietokone tarkastalee käskyjoukkoa kokonaisuutena, sen sijaan että hän mallintaisi mielessään käskyjoukon peräkkäin suoritettavina toimintoina.

Du Boulay (1989) esittelee tietokonemallin käsitteen (engl. notional machine tai notational machine, NM) kuvaamaan sitä miten ohjelmoija ajattelee tietokoneen toimivan. Malli yhdistää ohjelmointikielen rakenteet ja tietokoneen todellisen toiminnan tilanteeseen sopivalla abstraktiotasolla. Eri tilanteissa ohjelmien ja tietokoneen toimintaa kuvataan eri tarkkuudella, mutta malleille on yhteistä ohjelmointikielen käsitteiden ja fyysisen tietokoneen tuominen yhteen. Alakoululaiselle yksinkertainen tietokoneen toiminnan malli voisi liittyä siihen että kone suorittaa käskyjä järjestyksessä. Myöhemmin, kun koululaiselle esitellään toisto, ehdollisuus ja aliohjelmien käyttö, oppilas ehkä täydentää malliaan ja mieltää että koneen muistissa on ohjelmakoodia, jota suoritetaan koodissa olevien ohjeiden mukaisessa järjestyksessä. Opiskelijan käsitys tarkentuu oppisen aikana ja samoin hänen mallinsa vastaa yhä tarkemmin tietokoneen todellista toimintaa. Yleisesti ottaen opiskelijoiden on vaikea mieltää ohjelmointikielen mahdollistaman tietokonemallin suhdetta fyysiseen tietokoneeseen (Satratzemi, Dagdilelis & Evageledis, 2001) vielä korkeakoulutasolla. Kuten Du Boulay (1989) kuvailee, vie pitkän aikaa oppia ohjelmakoodin ja sen kuvaaman toimintamekanismin yhteys. Opiskelijoiden on vaikea ymmärtää, että jokainen käsky suoritetaan edellisten käskyjen luomassa tilassa.

Konstruktivistinen oppimiskäsitys ja ohjelmoinnilliseen ajatteluun liittyvä tutkimus ovat tuoneet oppimisen metakognitiot ohjelmoinnin oppisen tutkimukseen. Sorva (2012) määrittelee että ohjelmoinnin osaamisen edellytyksenä on toimivan ja kattavan tietokonemallin omaksuminen. Tämä määritelmä laajentaa tietokonemallin roolia ymmärryksen kuvaajasta oppimistavoitteeksi.

Meyer & Land (2003) luonnehtivat mielikuvaa tietokoneen toiminnasta yhdeksi ohjelmoinnin oppimisen kynnyskysymykseksi. Suorittimen ja muistin tarkoituksen ja yhteistyön ymmärtäminen on tärkeää ohjelmoinnin oppimisen kannalta. Opiskelijoille tulee selvittää mitä muistissa tapahtuu ohjelman suorituksen aikana. Khalifen (2006) tutkimuksessa yksinkertaistetun, vuokaavioiden avulla estetyn tietokonemallin sisäistäminen paransi opiskelijoiden oppimistuloksia selvästi.

2.3 Oppimistavoitteet

Ennen opetusmenetelmien valintaa on valittava tilanteeseen sopiva oppimistavoite. Sen sijaan että nimetään kurssilla opittavat ohjelmointikielen rakenteet, voidaan oppismitavoitteita määritellä laajemmin osana ohjelmoinnin oppimisen jatkumoa. Yhtenä tavoitteena on esitetty sitä, että opiskelijalle kehittyy kuva tietokoneen toiminnasta ohjelmointikielen määrittämänä.

Toisaalta ongelmanratkaisua pidetään keskeisenä sekä ohjelmoinnin osaamisessa että sen opettelussa. McCrackenin ym. (2001) mukaan kaikille ohjelmoinnin peruskursseille on asetettavissa viisi selkeää oppimispäämäärää ongelmien ratkaisuun liittyen:

1. Kyky ongelman käsitteellistämiseen: oppijoiden on kyettävä erittelemään ongelman pääpiirteet ja mallintamaan ratkaisu annetussa abstraktiokehyksessä, tarkoittaen käytettävää ohjelmointikieltä, -ympäristöä sekä -tekniikkaa.

2. Ongelman jako osaongelmiin: oppijoiden on edelleen kyettävä jakamaan laajemmat ongelmat asiaankuuluviin osaongelmiin. Luonnollisena esimerkkinä toimii ohjelmien jakaminen funktioihin ja proseduureihin.

3. Osaratkaisujen muodostaminen osaongelmille: vaiheessa kolme oppija määrittelee ratkaisunsa konkreettisemmin tietorakenteiden, luokkien, funktioiden ja muiden valitun ympäristön tarjoamien ratkaisumahdollisuuksien kautta. Olennaista ratkaisun oikeellisuuden lisäksi on siis oikeiden keinojen käyttö sen muodostuksessa.

4. Ohjelman uudelleenlaatiminen: seuraavaksi oppija muodostaa osaratkaisuista toimivan ohjelman laatimalla algoritmin, joka toteuttaa halutun sekvenssin oikeassa järjestyksessä.

5. Evaluointi ja (tarvittaessa) toisto: viimeisessä vaiheessa oppija tarkastelee ratkaisunsa toimivuutta sekä syntaktisessa että loogisessa mielessä, ja tekee siihen tarvittaessa korjauksia.

Yhteenvetona voidaan todeta, että opetuksessa tulisi huomioida yhtäläisesti

- ohjelmointikielen rakenteiden hallinta
- tietokonemallin kehittyminen ja
- ongelmanratkaisutaitojen omaksuminen.

2.4 Oppimisen eteneminen

Opetusmenetelmien valinta on keskeinen tekijä siinä, miten määritellyt oppimistavoitteet saavutetaan. Bloomin taksonomian mukaan oppiminen etenee asteittain yksinkertaisemmista kognitiivisista taidoista kohti monimutkaisempia. Taksonomia tarjoaa hyödyllisen viitekehyksen oppimistavoitteiden asettamiselle ja niiden toteutumisen arvioimiselle. Se jakautuu kuuteen eri tasoon: muistiinpalauttaminen, ymmärtäminen, soveltaminen, analysointi, arviointi ja luominen. Jokainen taso edustaa syvempää ja monipuolisempaa oppimista. (Bloom ym., 1956)

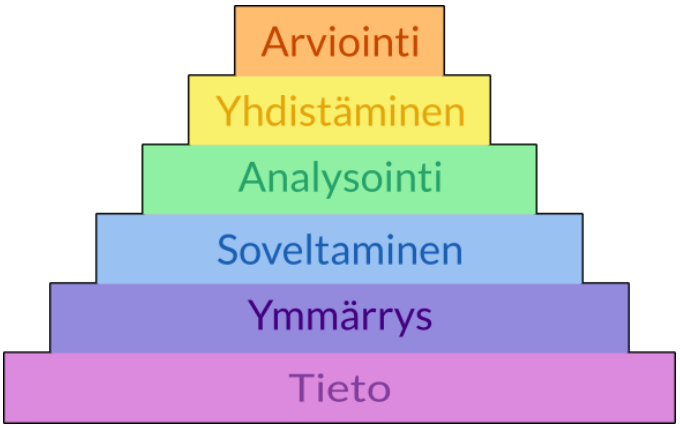
Perinteisesti ohjelmoinnin opettamisessa keskitytään muistiinpalauttamiseen ja ymmärtämiseen, mutta edistyneempiä tavoitteita, kuten analysointia, soveltamista ja uusien ratkaisujen luomista, voidaan tukea tehokkaammin oikeilla opetusmenetelmillä. **Tietokonemallin** (engl. notional machine) käsite liittyy erityisesti ohjelmointikielten ja -ympäristöjen käyttöön siinä, kuinka ne ohjaavat opiskelijaa ajattelemaan ohjelmakoodin rakennetta ja toimintaa. Tämä vaikuttaa suoraan siihen, miten opiskelijat etenevät Bloomin taksonomian tasoilla ohjelmoinnin oppimisessa.

Bloomin taksonomia jakaa mekanismin ja selityksen yhteyden osiin. Scott (2003) esittelee Bloomin taksonomian kategoriat ja niiden soveltamisen ohjelmoinnin oppimisen vaiheisiin seuraavasti:

1. **Tieto** (*Recall of Data, Knowledge*): Oppija muistaa ulkoa opetetut käsitteet kuten käskyt, avainsanat tai muuttujien tyypit.
2. **Ymmärrys** (*Comprehension*): Oppija pystyy seuraamaan opettajan selitystä ongelman ratkaisusta, ja ymmärtää, mitä ohjelma tekee ja mitkä ovat sen suorituksen vaikutukset esimerkiksi muuttujien arvoihin ja ohjelman tulostukseen.
3. **Soveltaminen** (*Application*): Oppija pystyy soveltamaan oppimaansa uusissa tilanteissa, esimerkiksi korvaamaan toisto- tai tietorakenteen toisella vastaavalla.

4. **Analysointi** (*Analysis*): Oppija pystyy jakamaan ongelman riittävän pieniin aliongelmiin, esimerkiksi jakamaan ohjelman aliohjelmiin.
5. **Yhdistäminen** (*Synthesis*): Oppija pystyy luomaan uusia ratkaisuja yhdistämällä aikaisemmin opittuja konsepteja keskenään, esimerkiksi muodostamaan toimivan luokan operaatioineen.
6. **Arviointi** (*Evaluation*): Oppija pystyy kriittisesti arvioimaan ratkaisun tai algoritmin laatua.

Alemmat tasot sisältyvät ylempiin tasoihin: kuudennelle tasolle päästäkseen oppijan on siis omaksuttava kaikki alemmat tasot.



Kuva 1 Bloomin taksonomia (Bloom, 1956)

2.5 Oppimisympäristö oppimisen tukena

Oppimistavoitteet ja opetusmenetelmät ohjaavat opettajaa työvälineenä käytettävän digitaalisen oppimisympäristön valinnassa. Oppimisympäristö kattaa monenlaisia ratkaisuja laajaa oppimisanalytiikkaa tukevista helppokäyttöisistä järjestelmistä yksinkertaiseen kääntäjästä ja koodiesimerkeistä koottuun opetusmateriaaliin. Opettajien mieltymykset ja opetustilanteet vaihtelevat ja paras ratkaisu on aina sellainen, jolla ei pelkästään saavuteta hyviä tuloksia, vaan johon myös sekä opettaja että oppilaat ovat tyytyväisiä.

Oppimisympäristöjen luokittelun ja ominaisuuksien kuvaamisen sijaan tässä nostetaan esille muutamia esimerkkejä, joista osaa tarkastellaan myöhemmin yksityiskohtaisemmin. Rajoitamme tarkastelua yläkouluikäisille soveltuvaan tekstipohjaiseen ohjelmointiin, ottamatta

kantaa käytettävään ohjelmointikieleen.

- Ohjelmoitavat laitteet (esim. Micro:bit)

  Laitteen ohjelmointi voi tuoda konkreettisuutta ohjelmoinnin oppimiseen. Hankinta- ja ylläpitokustannukset voivat nousta korkeiksi.

- Nettisivustot (esim. code.org)

  Opettajan on helppo ottaa käyttöön, mutta oppijoiden etenemistä voi olla vaikeaa seurata.

- Oppimisympäristöt (esim. ViLLE)

  Opettajalla on käytössään laaja oppimisanalytiikka ja oppilaiden on helppo aloittaa ohjelmoinnin opettelu. Tarvitsee korkeakoulutasolla yleensä oheen kehitysympäristön laajempia ohjelmointiprojekteja varten.

- Materiaali, jota käytetään yksinkertaisen netissä toimivan kääntäjän kanssa (esim. Jypeli)

  Käyttöönotto joustavaa, mutta käytön skaalaaminen laajempaan käyttöön vaatii resursseja.

- Harjoittelu ammattikäyttöön soveltuvassa kehitysympäristössä (esim. Eclipse)

  Ohjelmoinnin opettelun aloitukseen haastava ratkaisu, mutta pidemmälle ehtineet opiskelijat voivat tehdä harjoitustyöprojekteja.

3 Ohjelmoinnin oppimista tukevat menetelmät

Ohjelmoinnin oppimista voidaan tehostaa erilaisin menetelmin ja tutkimuksia aiheesta on julkaistu runsaasti. Osa menetelmistä on luonteeltaan opetuksellisia tekniikoita, joiden toteuttamisessa käytettävällä oppimisympäristöllä ja sen ominaisuuksilla ei ole suurta roolia – esimerkkeinä ongelmalähtöinen oppiminen ja pariohjelmointi. Ohjelmien visualisointi, automaattinen arviointi ja välitön palaute sekä pelillistäminen puolestaan ovat menetelmiä, jotka toteutetaan oppimisympäristön avulla. Tässä kappaleessa esitellään lyhyesti näiden menetelmien määritelmä, tarkoitus, luokitteluja ja hyötynäkökulmia sekä mainitaan menetelmän käyttöä koskevia tutkimuksia.

3.1 Visualisointi

Baeckerin (1988) mukaan visualisointi (engl. visualization) tarkoittaa näkemistä tai mielikuvan muodostamista jostakin ja ohjelman visualisointi on sen esittämistä graafisten elementtien avulla. Price ym. (1993) ja Stasko ym. (1997) määrittelevät visualisoinnin typografian, graafisen suunnittelun, animaation ja videon käyttämiseksi modernien käyttöliittymien ja tietokonegrafiikan avulla jotta käyttäjä ymmärtäisi ohjelmistoa paremmin ja käyttäisi sitä tehokkaammin. Ben-Ari (2001) sisällyttää visualisointiin kaiken ohjelman graafiseen esittämiseen liittyvän alkaen ohjelmakoodin sisennyksestä jatkuen ohjelman toiminnan esittämiseen animoinnin avulla. Algoritmien visualisointi eroaa ohjelman visualisoinnista. Napsin (1996) määritelmän mukaan algoritmien visualisoinnissa keskitytään esittämään mitä tietorakenteille tapahtuu ja varsinainen ohjelmakoodi voi jäädä tarkastelun ulkopuolelle. Visualisoinnin määritelmiä tarkastellessa tulee myös huomioida ero ohjelman visualisoinnin ja visuaalisen ohjelmoinnin välillä.

Visualisoinnin tarkoituksena on auttaa oppilasta tajuamaan mitä ohjelma tekee, miksi se tekee niin, miten se toimii ja mitä prosessin seurauksena tapahtuu. Visualisointijärjestelmä konkretisoi siis ohjelmien ja algoritmien suorituksen abstrakteja piirteitä. Oudshoorn ym. (1996) jakavat ohjelman visualisoinnin tarkoituksen mukaan toiminnan ymmärtämiseen, ohjelman virheiden etsimiseen ja suorituksen analysointiin tähtäävään visualisointiin.

Ohjelmistojen visualisointimenetelmät voidaan ryhmitellä erilaisin kriteerein. Visualisoinnin kohteen mukaan erotellaan algoritmien ja ohjelman visualisointimenetelmät (Naps ym., 1996). Ben-Ari (2001) jakaa visualisoinnin alempaan ja korkeampaan tasoon, ohjelmien visualisointi asettuu alemmalle tasolle ja algoritmien visualisointi korkeammalle. Ero on kuitenkin kapea ja useat ohjelmistot pystyvät visualisoimaan sekä suoritusta että tiedollista sisältöä. Myersin (1986) varhainen taksonomia luokittelee visualisoinnin nelikentän avulla. Akseleina ovat staattinen-dynaaminen ja koodi-data (taulukko 2).

Taulukko 2 Myersin (1986) visualisoinnin nelikenttäjaottelu.

	staattinen	**dynaaminen**
koodi	esim. koodin sisennys	esim. suoritettavan rivin korostaminen
data	esim. kaavio ohjelman tiloista	esim. ohjelman suoritusta visualisoiva järjestelmä

Hundhausenin ym. (2002) tutkimuksesta ilmeni, ettei visualisoinnista ole hyötyä, jos oppija ei aktiivisesti osallistu visualisointiin. Osittain mainitun tutkimuksen johdosta Naps ym. (2002) kehittivät sitoutumisen luokittelun (engl. the Engagement Taxonomy). Luokittelun mukaan opiskelijan sitoutuminen jaetaan kuuteen tasoon: ei visualisointia (no viewing), katsominen (viewing), vastaaminen (responding), muokkaaminen (changing), rakentaminen (constructing) ja esittäminen (presenting) (taulukko 3). Sitoutumisen luokittelua on testattu useissa tutkimuksissa (Laakso, 2010).

Taulukko 3 Sitouttamisen luokittelu (Naps ym., 2002).

	Taso	Kuvaus
Ei sitouttamista	Ei visualisointia	Materiaalia ei ole lainkaan visualisoitu.
Passiivinen sitouttaminen	Katsominen	Visualisointia voi katsoa
Aktiivinen sitouttaminen	Vastaaminen	Oppija vastaa visualisointiin liittyviin kysymyksiin
	Muokkaaminen	Oppija voi vaikuttaa visualisaatioon muokkaamalla sitä
	Rakentaminen	Oppija voi luoda omia visualisaatioita
	Esittäminen	Oppija esittelee visualisaatioita, keskustelee niistä ja saa palautetta

Useissa tutkimuksessa on verrattu eri tasojen tuomaa hyötyä oppimiseen (Laakso, 2010) ja on selvää, että aktiivisen sitouttamisen tasot tuovat parhaat oppimistulokset. Suoritetut tutkimukset ovat antaneet aihetta tarkentaa luokittelua ja Myller ym. (2009) ovat esittäneet laajennetun luokittelun, joka lisää

hallitun katsomisen, syötteen antamisen, muuttamisen ja arvioinnin tasot. Tämä laajennus ulottaa luokittelun koskemaan myös yhteistoiminnallista oppimista. Algoritmien visualisointia on tutkittu laajemmin kuin ohjelmasuorituksen visualisointia.

3.2 Automaattinen arviointi ja välitön palaute

Arviointi viittaa tässä ohjelmoinnin oppimiseen liittyvien suoritusten tai tehtävien vertaamista virheettömään malliin. Automaattisen arvioinnin (engl. automatic assessment) vastakohta on ihmisen suorittama arviointi. Isokin opiskelijajoukko voi automaattisen arvioinnin avulla saada välitönta palautetta (engl. immediate feedback).

Oppiseen vaadittavaa sitouttamista ei voi toteuttaa ilman palautetta. Kasvatustieteen näkökulmasta arvioinnin rooli oppimisessa ja oppimisen ohjaamisessa on keskeinen. Opiskelijat oppivat sen mikä arvostellaan. Arvosteltavien tehtävien määrän vähentäminen arviointiresurssien puutteen vuoksi heikentää oppimista. Automaattisesta arvioinnista on siis hyötyä sekä opettajille että opiskelijoille.

Ala-Mutka (2005) kokoaa väitöskirjassaan automaattisen arvioinnin menetelmiä ja työkaluja. Tyypillisesti arviointi tehdään vertaamalla oppilaan ratkaisua opettajan tuottamaan malliin. Muita lähestymistapoja ovat ohjelman tilan tai sen osan vertaaminen malliin ja yksityiskohtaisempi ohjelman testaus. Virheettömyyden lisäksi automaattisen arvioinnin kohteena voivat olla myös ohjelman tehokkuus, aikakompleksisuus tai esimerkiksi koodin tyyli. Automaattisen arvioinnin työkaluja voidaan myös käyttää usean opettajan arviointityylin yhtenäistämiseen. Oppilaan kannalta automaattisen arvioinnin avulla voi usein saada oppimisprosessia tukevaa välitöntä palautetta ennen tehtävän lopullista palautusta. Toisaalta vaikka automaattinen järjestelmä pystyykin osoittamaan oppilaalle hänen virheensä, se ei yleensä korjaamaan oppilaan virhekäsityksiä. Tehtävät täytyy suunnitella siten että opiskelija ei voi mahdollisesta virhekäsityksestä huolimatta päätyä oikeaan ratkaisuun. Plagiarismin torjumiseksi automaattisen arvioinnin järjestelmiin toteuttaa yksityiskohtaisia vertailujärjestelmiä. Yritys ja erehdys menetelmän käyttö on automaattisen arvioinnin järjestelmissä yleensä estetty esimerkiksi rajoittamalla tehtävän palautusten määrää, tai generoimalla uusi data tehtävään.

3.3 Pelillistäminen

2010-luvun jälkeen pelien ominaisuuksien käyttö muissakin kuin pelisovelluksissa on yleistynyt. Pelillistäminen (engl. game-based learning or gamification) liittyy laajaan pelien käytön käsitekenttään, mutta sen suhde kentän muihin käsitteisiin on ollut epäselvä. Deterding ym. (2011) tutkivat pelillistämisen syntyä ja ehdottavat määritelmäksi ”pelielementtien käyttöä muissa kuin peliyhteyksissä”.

Useimpien pelien keskeisin elementti on pelaajien välinen vuorovaikutus. Pelaajan täytyy aktiivisesti osallistua peliin menestyäkseen siinä. Shabanah ym. (2011) esittävät tietokonepelien pelaamisen uudeksi aktiivisen sitouttamisen muodoksi, jossa yhdistyvät kaikki Naps ym. (2002) sitouttamisen

luokittelun aktiivisen sitouttamisen tasot. Pelien avulla voidaan siis sitouttaa opiskelijoita oppimiseen ja siten tehostaa oppimista. Lisäksi pelillistämisen keinoin halutaan motivoida. Motivoinnin merkitystä ohjelmoinnin oppimiselle on selvitetty mm. Mannilan (2009) toimesta. Pelien välityksellä voidaan myös lisätä opiskelijoiden välistä yhteistyötä.

Pohtiessaan miten pelillistetyt sovellukset eroavat hyötypeleistä (en. serious game) Deterding ym. (2011) esittivät kysymyksen pelielementtien määrittelystä ja vastaavat kysymykseensä rajaamalla pelielementit elementeiksi, joita useimmissa (ei kuitenkaan välttämättä kaikissa) peleissä esiintyy. Taulukossa 4 on esitetty viisi tyypillisten pelielementtien tasoa.

Taulukko 4 Pelielementtien tasot (Deterding ym., 2011).

Taso	Kuvaus	Esimerkki
Pelirajapinnan suunnittelumallit	Yleiset vuorovaikutuskomponentit ja tiettyyn kontekstiin liittyvä ongelmanratkaisumalli, sisältäen prototyyppitoteutukset	Kunniamerkki, tulostaulu, taso
Pelin suunnittelumallit ja pelimekaniikka	Pelaamiseen liittyvät usein toistuvat pelisuunnittelun osat	Aikarajoitus, rajalliset resurssit, vuorot
Pelin suunnitteluperiaatteet ja heuristiikka	Ohjeet suunnitteluongelman tarkasteluun tai tietyn suunnitteluratkaisun analysointiin	Jatkuvakestoinen pelaaminen, selkeät päämäärät, erilaiset pelityylit
Pelimallit	Pelikomponenttien tai pelikokemuksen käsitteelliset mallit	MDA; haaste, fantasia, uteliaisuus; pelisuunnittelun rakenneosaset; CEGE
Pelin suunnittelumetodit	Pelisuunnittelukohtaiset käytännöt ja prosessit	Testaus pelaamalla, pelikeskeinen suunnittelu, arvotietoinen pelisuunnittelu

Hyötypelit ovat siis täysimittaisia pelejä, sitä vastoin pelillistetyt sovellukset ainoastaan käyttävät joitakin pelielementtejä. Pelien opetuskäytössä oppiminen voi tapahtua eri tavoin pelin kanssa vuorovaikutuksessa. Pelin toteuttaminen ohjelmoimalla, pulmapelin ratkaisun löytäminen ohjelmoimalla tai pelin pelaaminen ovat erilaisia tapoja. Li & Watson (2011) jakavat pelien käytön ensimmäisillä ohjelmointikursseilla (CS1) kolmeen kategoriaan – pelien kirjoittamiseen, pelien pelaamiseen ja niiden välillä sijaitsevaan visualisointiin pohjautuviin lähestymistapoihin.

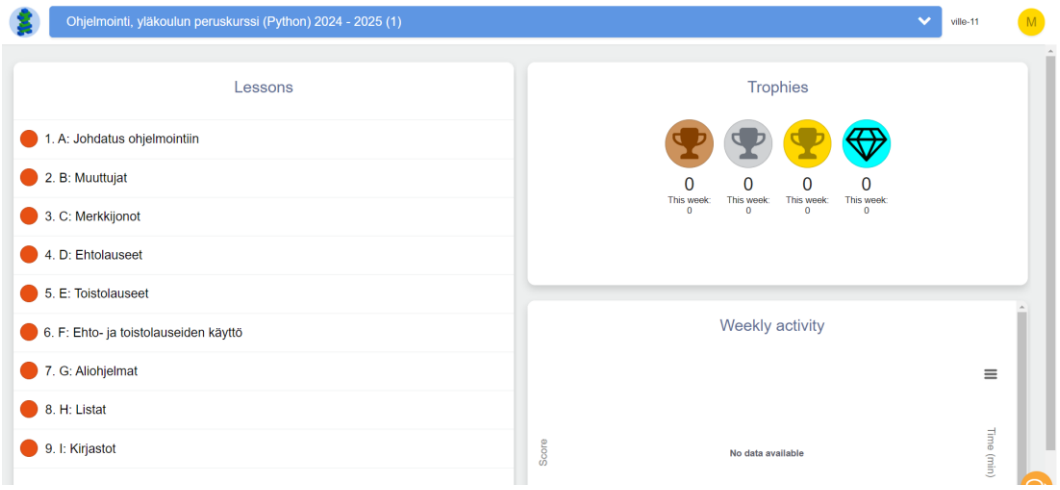
4 Järjestelmät

Tässä luvussa esitellään tarkemmin kolme suosittua, ilmaista ja suomenkielistä ratkaisua yläkouluun tekstipohjaisen ohjelmoinnin alkeiden oppimiseen. Järjestelmiin liittyviä tutkimusartikkeleista on kuvattu tarkemmin, mikäli niitä on ollut saatavilla. Järjestelmissä käytettyjä, oppimista tehostavia menetelmiä, on koottu Yhteenveto-luvussa esitettyyn taulukkoon.

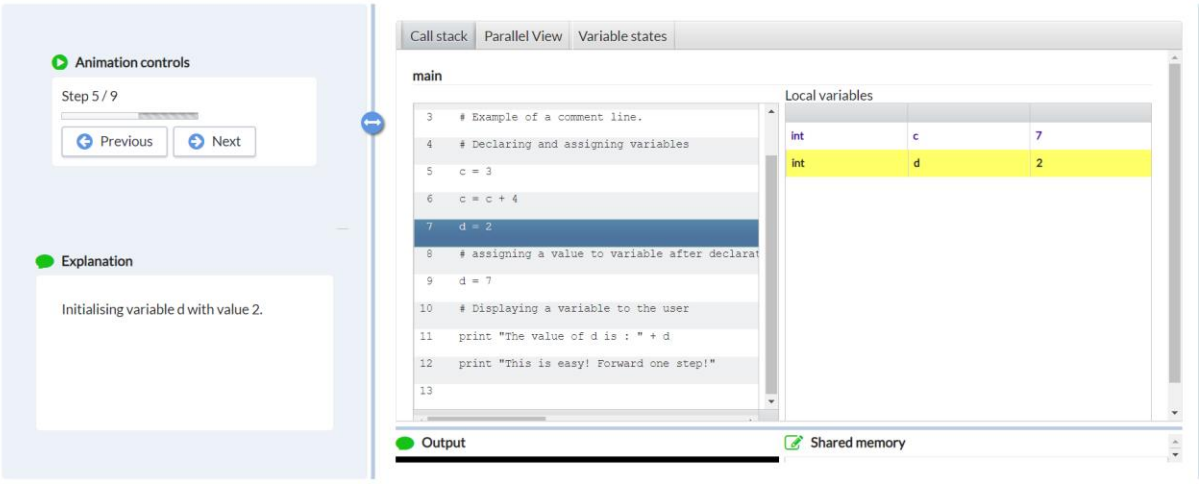
4.1 ViLLE

ViLLE (Laakso ym., 2018) on Turun yliopiston Oppimisanalytiikan instituutissa kehitetty oppimisalusta, joka tarjoaa opettajalle kattavan oppimisanalytiikan. ViLLE on alusta, johon sisällöntuottajatahot – Turun yliopisto tai yksittäiset käyttäjät – laativat kursseja käyttäen ympäristön editointityökaluja. Valmiit kurssit ovat kaikkien ViLLEn opettajakäyttäjien kopioitavissa ja muokattavissa.

ViLLEn on luotu useita kursseja ohjelmoinnin opetteluun, joista tässä tarkastellaan Oppimisanalytiikan instituutin laatimaa Python-ohjelmoinnin perusteiden kurssia: **Ohjelmointi, yläkoulun peruskurssi (Python) 2024 – 2025**. Kurssi koostuu ViLLEssä olevasta oppilaalle suunnatusta materiaalista ja sitä tukevista, Python-ohjelmointiympäristössä tehtävistä harjoitustöistä. Muiden ViLLE-kurssien tapaan materiaali on helppo ottaa käyttöön: opettaja kopioi itselleen muokattavan version kurssista ja lisää sinne opiskelijat. Materiaali on jaettu oppitunteihin, joita opettaja asettaa viikottain näkyville. Kurssin aikana oppilas tutustuu tekstipohjaiseen ohjelmointiin käyttäen Python-kieltä, ilman ennakkotietoja.

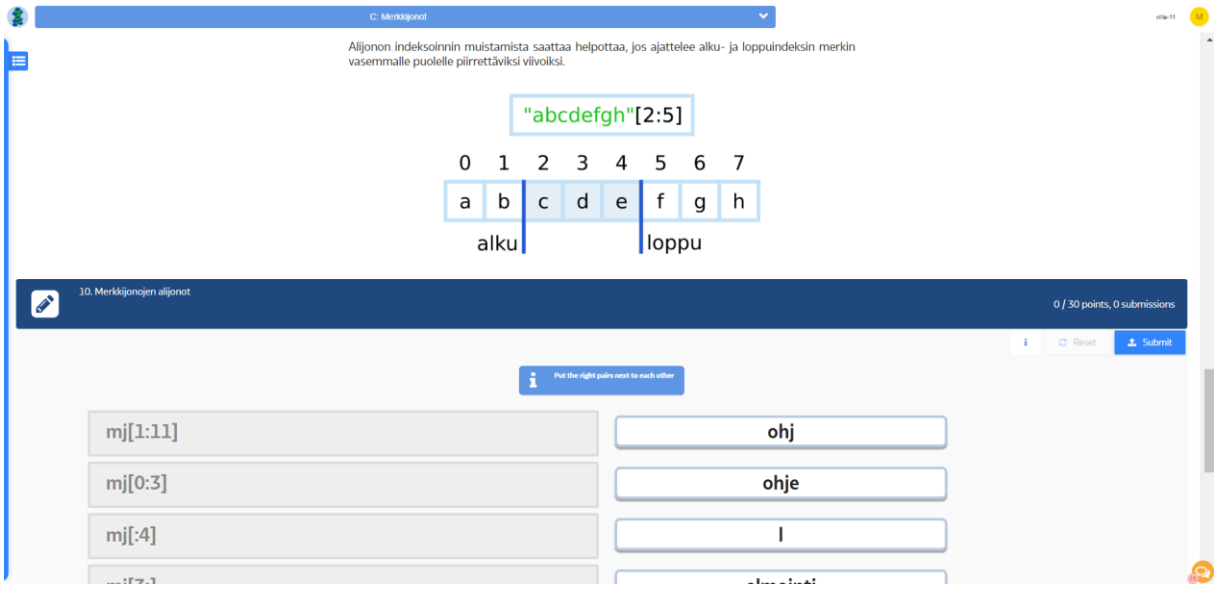


Kuva 2 Kurssin sisältö (kuva sivustolta https://ville.utu.fi).



Kuva 3 Esimerkki ViLLEn visualisointitehtävästä, jossa näkyy muuttujien arvojen muutokset ohjelman suorituksen aikana (kuva sivustolta https://ville.utu.fi).

Oppitunnin sisältö koostuu opetustekstistä ja siihen nivoutuvista, osaamista vahvistavista erityyppisistä tehtävistä. Tehtävät vaihtelevat pienistä ohjelmointihaasteista ja koodin visualisoinneista rakenteiden omaksumista selventäviin minipeleihin.



Kuva 4 Merkkijonot-oppitunnin alijonon indeksointiin liittyvä teksti ja sitä selventävä tehtävä (kuva sivustolta https://ville.utu.fi).

Oppilas kerää pisteitä vastatessaan oikein ja tavoittelee pokaalipalkintoa. Edistyminen kurssilla on selkeästi visualisoitu. ViLLEn oppimisanalytiikka antaa opettajalle kokonaiskuvan luokan

etenemisestä ja auttaa huomaamaan yksittäisen oppilaan tuen tarpeen niin varhain kuin mahdollista. ViLLEssä on myös suora viestintämahdollisuus opettajan ja oppilaiden välillä. ViLLEn lukio- ja korkeakoulutasolle suunnatuista ohjelmointikursseista on julkaistu useampia tutkimuksia, joissa on osoitettu positiivinen vaikutus oppimiseen (Kaila, 2018).

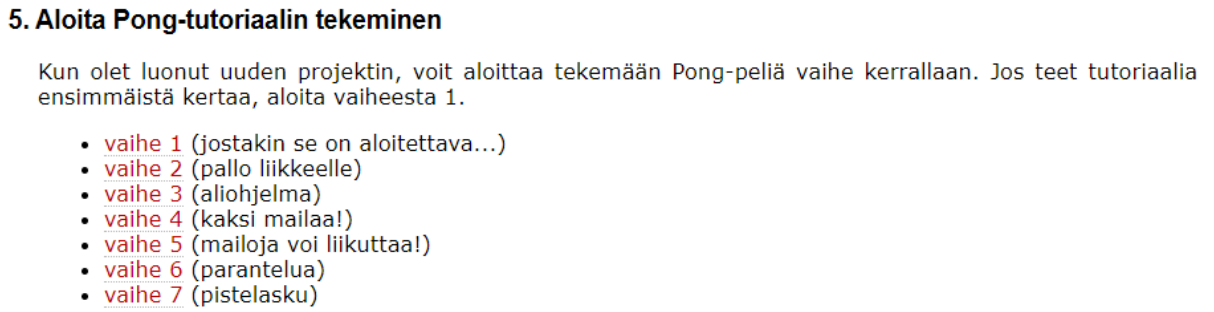
4.2 Jypeli

Jypeli on jo hieman vanhempi, mutta edelleen käytössä oleva Jyväskylän yliopistossa kehitetty avoimen lähdekoodin peliohjelmointia tukeva C#-kirjasto (Isomöttönen ym., 2011). Kirjastoa käyttäen toimivien pelien kehittäminen onnistuu aloittelevaltakin ohjelmoijalta ja ohjelmointiharrastuksessa pääsee samoilla työvälineillä varsin pitkälle. Jypelin avulla ohjelmoinnin oppimisessa korostuu itse tekeminen ja toimivan lopputuloksen tuoma motivaatio.

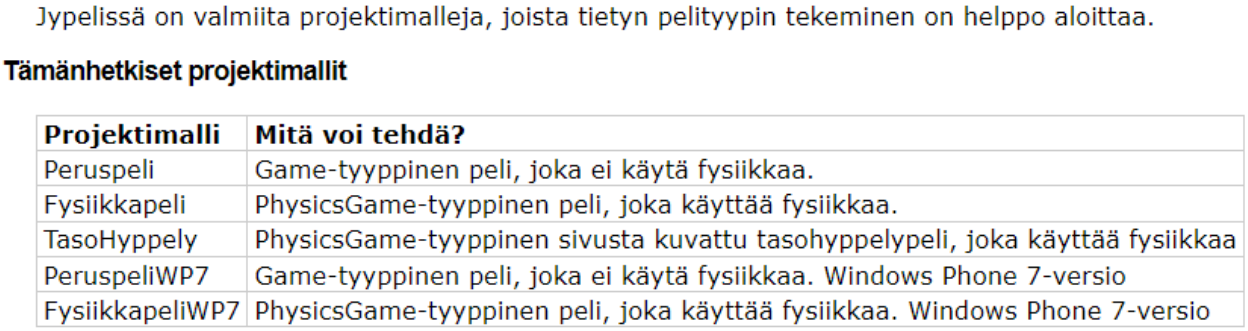
Kuva 5 Tukimateriaalissa ohjataan laatimaan Pong-peli Jypeli-kirjaston avulla (kuva sivustolta https://trac.cc.jyu.fi/projects/npo)

Kirjaston käyttö vaatii C#-ohjelmointiympäristön asentamisen oppilaan koneelle, mutta sen jälkeen ohjelmointiympäristössä on näkyvissä valmiita jypeli-projektimalleja, joiden avulla

oppilas pääsee alkuun. Ohjelmoinnin tueksi aloitteleva ohjelmoija tarvitsee tutoriaaleja ja dokumentaatiota. Useimmat yläkoulutoteutukset lienevät opettajajohtoisia, jolloin heikommatkin pysyvät mukana kopioimalla opettajan kirjoittamaa esimerkkikoodia. Pidemmälle edenneille oppilaille kirjasto tarjoaa tuen itsenäiseen etenemiseen ja oman luovuuden käyttämiseen.



Kuva 6 Pelin kehitys ohjataan vaihe vaiheelta (kuva sivustolta https://trac.cc.jyu.fi/projects/npo)



Kuva 7 Projektimalleja on erityyppisiä pelejä varten (kuva sivustolta https://trac.cc.jyu.fi/projects/npo)

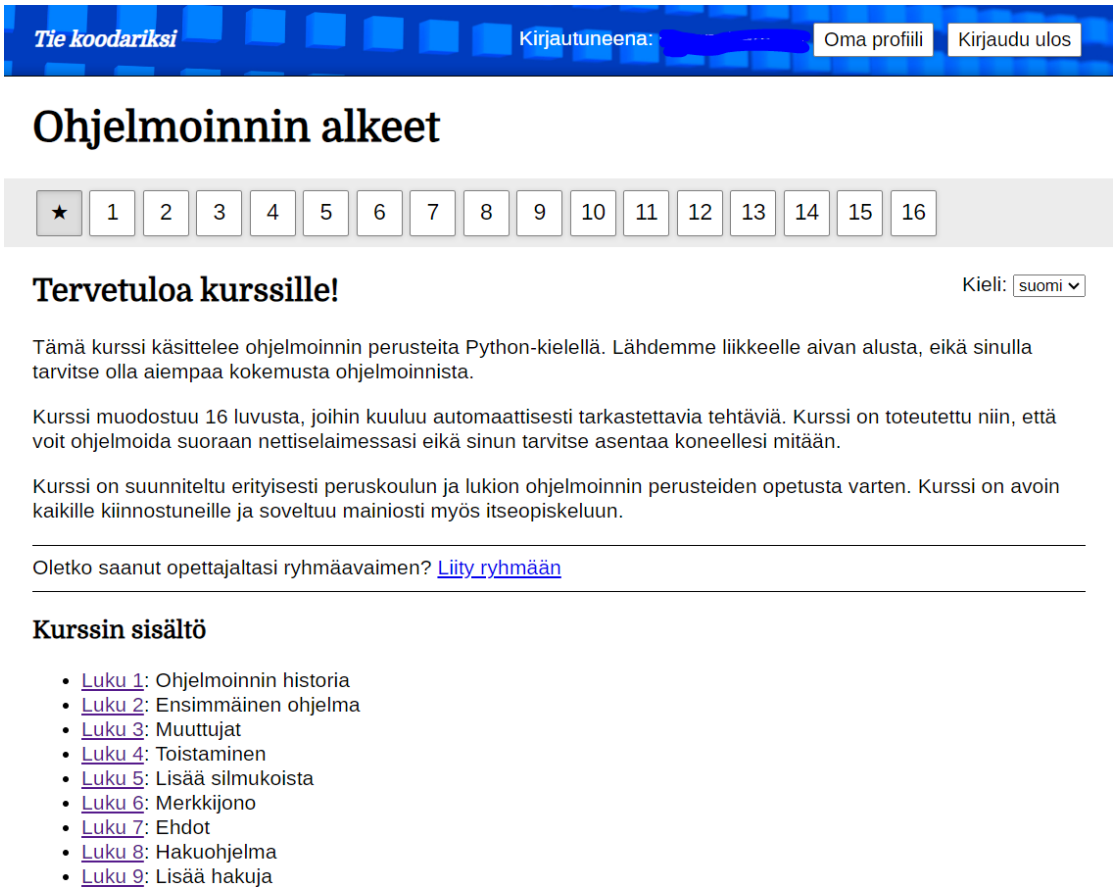
Jypeli keskittyy erityisesti 2D-pelien kehittämiseen, ja se sisältää monia valmiita ominaisuuksia, kuten pelaajan ohjattavia hahmoja, vihollisia, esteitä ja voimaesineitä. Pelinkehityksen tueksi ympäristö tarjoaa myös sisäänrakennetun fysiikkamoottorin, joka simuloi fysiikan lakeja, kuten painovoimaa ja törmäyksiä, helpottaen realistisen pelimaailman rakentamista. Käyttäjät voivat hyödyntää valmiita grafiikka- ja ääniresursseja, mutta halutessaan he voivat myös lisätä omia resurssejaan peliprojekteihin.

Ohjelmoinnin opettelu Jypelin avulla on yksinkertaista, sillä ympäristössä pelimekaniikan, kuten hahmojen liikkumisen ja erilaisten tapahtumien käsittelyn, koodaaminen on tehty helpoksi. Tämä mahdollistaa sen, että aloittelijat voivat keskittyä enemmän luovan sisällön

tuottamiseen, eikä heidän tarvitse huolehtia monimutkaisista teknisistä asioista. Jypelillä on myös kattava dokumentaatio ja runsaasti suomenkielisiä opetusmateriaaleja. Jypeli on ollut käytössä yliopisto-opiskelijoiden kursseilla, koululaisille suunnatuilla kesäkursseilla ja aktiivisten opettajien toimesta osana yläkoulun opetusta.

4.3 Tie koodariksi

Kolmas yläkouluissa käytössä oleva ilmainen suomenkielinen ratkaisu on tie.koodariksi.fi-sivusto ja sen Ohjelmoinnin alkeet -kurssi. Sivustoa ylläpitää projektiryhmä, jonka pääyhteistyökumppaneista ovat Helsingin yliopiston tietojenkäsittelytieteen Linkki-keskus sekä Maunulan yhteiskoulu ja Helsingin matematiikkalukio.



Kuva 8 Tie koodariksi -sivuston etusivu (kuva sivustolta https://tie.koodariksi.fi/alkeet/)

Sivuston automaattisesti arvioitujen ohjelmointitehtävien avulla oppilas omaksuu helposti tekstipohjaisen ohjelmoinnin alkeet Python-kielellä. Sivustolle on myös mahdollista saada

opettajatunnus, jonka avulla opettaja voi koota omista oppilaistaan ryhmiä, joiden tuloksia hän voi sivustolla tarkastella.

Sivustolla on tarjolla oppimateriaalin lisäksi kattavasti tehtäviä, jotka etenevät vaikeustasoltaan johdonmukaisesti. Käyttäjät oppivat ohjelmoinnin perusasioita, kuten muuttujia, ehtolauseita, silmukoita ja funktioita, mutta myös laajempia teemoja, kuten tiedon käsittelyä ja algoritmien suunnittelua.

Tehtävien rakenne on pedagogisesti suunniteltu siten, että ne tukevat uuden oppimista vähitellen. Aluksi tarjotaan yksinkertaisia harjoituksia, jossa käyttäjä opettelee tulostamaan tekstiä näytölle. Näitä seuraavat hieman haastavammat tehtävät, joissa tutustutaan ehtoihin ja toistolauseisiin, kuten ohjelmiin, jotka käsittelevät numeroiden listoja tai suorittavat laskutoimituksia käyttäjän antamien tietojen perusteella.

Kun oppilas etenee pidemmälle, tehtävät monimutkaistuvat ja niissä aletaan soveltaa enemmän algoritmista ajattelua ja rakenteellisempaa ohjelmointia. Esimerkiksi tietorakenteisiin liittyvät tehtävät, kuten listojen ja taulukoiden käyttö, vaativat jo syvempää ymmärrystä ohjelmoinnin logiikasta. Tehtäviin kuuluu myös paljon käytännön ongelmia, joissa ratkaisuja haetaan todellisista skenaarioista, kuten hakualgoritmeista ja lajittelusta.

Yksi keskeinen ominaisuus on automaattinen tehtävien tarkistaminen. Kun käyttäjä kirjoittaa ratkaisun tehtävään, sivusto tarkistaa automaattisesti koodin toimivuuden ja antaa palautetta virheistä tai onnistumisista. Tämä nopeuttaa oppimista, sillä käyttäjät saavat välitöntä palautetta tekemistään virheistä tai koodin onnistumisesta.

Kuva 9 Esimerkki sivuston tehtävästä (kuva sivustolta https://tie.koodariksi.fi/alkeet/)

4.4 Ohjelmoinnin oppimista tukevat menetelmät tutkituissa järjestelmissä

Kirjallisuuskatsauksen avulla löydetyt keskeiset ohjelmoinnin oppimista tukevat menetelmät visualisointi, automaattinen arviointi ja siihen kiinteästi liittyvä välitön palaute sekä motivoiva pelillisyys. Tutkittavat järjestelmät olivat tavoitteeltaan ja lähestymistavaltaan erilaisia, joten on selvää, että niiden menetelmäprofiilitkin ovat erilaiset. Järjestelmistä ViLLE oli monipuolisin ja siinä käytettiinkin kaikkia esiteltyjä menetelmiä. Osassa tehtävistä oli käytetty tehokkaita interaktiivisia visualisointeja. Jypeli-kirjaston avulla pelillisyys saa erilaisen roolin, sillä opiskelijat luovat sen avulla omia pelitoteutuksiaan. Tie koodariksi sivuston sisällöstä sai hyvän kuvan jo ilman kirjautumista, mutta kirjautuneille käyttäjille oli tarjolla lisäominaisuuksia.

Tulokset on koottu taulukkoon 5.

Taulukko 5 Ohjelmoinnin oppimista tehostavat menetelmät tutkituissa järjestelmissä.

	Visualisointi	Automaattinen arviointi	Välitön palaute	Pelillisyys
ViLLE	x	x	x	x
Jypeli				x
Tie koodariksi		x	x	

5 Yhteenveto

Tässä tutkielmassa peilattiin ohjelmoinnin oppimisen tavoitteita aiemmissa tutkimuksissa esiteltyyn tietokoneen toiminnan mentaaliseen malliin ja tarkasteltiin oppimisympäristöjä ohjelmoinnin oppimista tehostavien menetelmien kannalta.

Tutkimuskysymysten vastaukset:

TK1. Visualisointi, automaattinen arviointi ja välitön palaute yhdessä pelillistämisen kanssa ovat kirjallisuuden perusteella tehokkaita menetelmiä, joista on merkittävästi apua ohjelmoinnin oppimisessa.

TK2. Kaikissa tutkituissa järjestelmissä hyödynnettiin joitakin tutkimuksessa löydetyistä oppimista tehostavista menetelmistä. Turun yliopiston oppimisanalytiikan tutkimusinstituutin kehittämässä ViLLE-oppimisjärjestelmässä oli hyödynnetty kaikkia esitettyjä menetelmiä.

Tutkielmassa onnistuttiin vastaamaan molempiin kysymyksiin ja tutkimusmenetelmät olivat tarkoitukseen sopivat. Voidaan kuitenkin todeta, että suomenkielisiä, ilmaisia tekstipohjaista ohjelmointia tukevia järjestelmiä ei ole paljoa, ja olisikin ollut syytä laajentaa tarkastelua myös englanninkielisiin järjestelmiin, joita myöskin käytetään suomenkielisessä opetuksessa.

Tulevaisuudessa yhä isompi osa ihmisten elämästä tapahtuu digitaalisen maailman kontekstissa ja kyky ymmärtää ja taito muokata tätä ympäristöä lisää ihmisten vaikutusmahdollisuuksia. Näkemys siitä, että ohjelmointitaito on ihmisen perustaito, asettuu tähän kehykseen. Vaikka esimerkiksi peruskoulun opetussuunnitelmaan on lisätty ohjelmoinnin opetus tietoteknisten taitojen lisäksi, saattaa tavoite jäädä pinnalliseksi ohjelmointikielen syntaksin vilkaisuksi. Jotta nuoret oikeasti hyötyisivät ohjelmoinnin opetuksesta tulisi ohjelmoinnin avulla voida oppia ymmärtämään tietokoneen ja digitaalisen maailman takana olevan teknologian toimintaperiaatteita ja mahdollisuuksia.

Ohjelmoinnin oppimisen haasteet ovat laaja ongelmakenttä, mutta tutkimukseen valittu suppeampi näkökulma ja kysymyksenasettelu johdattivat alan perusteorian äärelle. Uskon että tutkimustulosten avulla tutkimuksen tarkoitus täyttyy ja opettajan on esiteltyjen tietojen avulla helpompi valita ohjelmoinnin opetukseen soveltuva järjestelmä.

Lähteet

Ala-Mutka, K. (2005). *Automatic assessment tools in learning and teaching programming* (Väitöskirja, Tampereen teknillinen yliopisto). Vol. 559

Baecker, R. (1989). Enhancing program readability and comprehensibility with tools for program visualization. *Proceedings. 11th International Conference on Software Engineering, Singapore, 1988*, 356-366

Bloom, B., Engelhart, M., Furst, E., Hill, W. & Krathwohl, D. (1956). Taxonomy of educational objectives: The classification of educational goals. *Vol. Handbook I: Cognitive domain*

du Boulay, J. (1989). Some difficulties of learning to program. *Studying the Novice Programmer,* 431-436

Deterding, S., Dixon, D., Khaled, R. & Nacke, L. (2011). From game design elements to gamefulness: defining "gamification". *Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments (MindTrek '11)*, 9–15

Hundhausen, C., Douglas, S. & Stasko, J. (2002). A Meta-study of Algorithm Visualization Effectiveness. *Journal of Visual Languages and Computing 13*, 259-290.

Kaila, E. (2018). *Utilizing educational technology in computer science and programming courses : theory and practice* (Väitöskirja, Turun yliopisto). TUCS Dissertations 230

Khalife, J. (2006). Threshold for the Introduction of Programming: Providing Learners with a Simple Computer Model. *Proc. PPIG*, 244-254.

Laakso, M. (2010). *Promoting programming learning: engagement, automatic assessment with immediate feedback in visualizations* (Väitöskirja, Turun yliopisto)

Laakso, M., Kaila, E. & Rajala, T. (2018). ViLLE – collaborative education tool: Designing and utilizing an exercise-based learning environment. *Education and Information Technologies, vol. 23, heinäkuu 2018*, 1655–1676

Isomöttönen V., Lakanen A. & Lappalainen V. (2011). K-12 Game Programming Course Concept Using Textual Programming. *SIGCSE '11: Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*.

Li F. & Watson C. (2011). Game-based concept visualization for learning programming. *Proceedings of the third international ACM workshop on Multimedia technologies for distance learning (MTDL '11)*, 37–42

Mannila, L. (2009), *Teaching mathematics and programming: new approaches with empirical evaluation* (Väitöskirja, Turun yliopisto).

McCracken, M., Almstrum, V., Diaz. D., Guzdial, M., Hagan. D., Kolikant, Y, Laxer, C., Thomas, L., Utting, I. & Wilusz, T. (2001). A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. *Working group reports from ITiCSE on Innovation and technology in computer science education (ITiCSE-WGR '01)*, 125–180

Meyer, J. & Land, R. (2003). Threshold concepts and troublesome knowledge: Linkages to ways of thinking and practising within the disciplines. *Improving Student Learning - Ten Years on*, 412-424.

Myers, B. (1986). Visual programming, programming by example, and program visualization: a taxonomy. *ACM sigchi bulletin 17 (4)*, 59–66

Myller, N., Bednarik, R., Sutinen, E. & Ben-Ari, M. (2009). Extending the engagement taxonomy: Software visualization and collaborative learning. *ACM Transactions on Computing Education. 9, 1*, artikkeli 7

Naps, T., Roßling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C., Korhonen, A., Malmi, L., McNally, M., Rodger, S. & Velazquez-Iturbide, J. (2002). Exploring the role of visualization and engagement in computer science education. *Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education 35, 2*, 131–152

Opetushallitus (2016). Perusopetuksen opetussuunnitelman perusteet 2014. 4. painos. url: https://www.oph.fi/fi/koulutus-ja-tutkinnot/perusopetuksen-opetussuunnitelman-perusteet.

Opetushallitus (2022). Uudet lukutaidot -kehittämisohjelma. url: https://eperusteet.opintopolku.fi/#/fi/digiosaaminen/8706410/tekstikappale/8709071

Oudshoorn M., Widjaja H. & Ellershaw S. (1996). *Aspects and taxonomy of program visualisation Software Visualisation*.

Price, B., Baecker, R. & Small, I. (1993). A Principled Taxonomy of Software Visualization. *Journal of Visual Languages and Computing 4(3)*, 211–266.

Satratzemi, M., Dagdilelis, V., & Evageledis, G. (2001). A system for program visualization and problem-solving path assessment of novice programmers. *Proceedings of the 6th Annual Conference on Innovation and Technology in Computer Science Education*, 137-140.

Scott, T. (2003). Bloom's taxonomy applied to testing in computer science classes. *Journal of Computing Sciences in Colleges, 19*, 267-274.

Shabanah S., (2011). Computer Games for Algorithm Learning. *Handbook of Research on Improving Learning and Motivation through Educational Games: Multidisciplinary Approaches*

Simon, B., Lister, R. & Fincher, S. (2006). Multi-institutional computer science educationresearch: A review of recent studies of no-vice understanding. *36th Annual Frontiers in Education Conference*, 12–17.

Sorva, J. (2012). *Visual program simulation in introductory programming education* (Väitöskirja, Aalto University).

Stasko, J., Badre, A. & Lewis, C. (1993). Do algorithm animations assist learning? An empirical study and analysis. *Proceedings of ACM INTERCHI'93 Conference on Human Factors in Computing Systems*, 61-66.