TAMPEREEN TEKNILLINEN YLIOPISTO

KATRIINA KYMÄLÄINEN-MÄKELÄ

QT PIIRTO-OHJELMAN TOTEUTUSALUSTANA

DIPLOMITYÖ

Tarkastaja: professori Mikko Tiusanen

Tarkastaja ja aihe hyväksytty Tieto- ja sähkötekniikan tiedekuntaneuvoston kokouksessa 8.12.2009

ALKUSANAT

Tämä diplomityö on tehty Tampereen teknillisen yliopiston ohjelmistotekniikan laitokselle. Haluan kiittää työn ohjaajaa, professori Mikko Tiusasta kärsivällisyydestä sekä hyvistä ja rakentavista kommenteista työn suhteen. Kiitos myös Juha-Matti Vanhatuvalle kommenteista erityisesti Qt-luvun suhteen.

Lisäksi haluan erityisesti kiittää johtajaa hyvistä neuvoista ja kannustamisesta sekä äidin rinsessoja pusuista ja haleista.

Tampereella, 18.7.2011

Katriina Kymäläinen-Mäkelä

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Tietotekniikan koulutusohjelma
KYMÄLÄINEN-MÄKELÄ KATRIINA: Qt piirto-ohjelman toteutusalustana
Diplomityö, 45 sivua, 5 liitesivua.
Pääaine: Ohjelmistotuotanto
Työn tarkastaja: professori Mikko Tiusanen
Avainsanat: Qt, Graphics View Framework, piirto-alusta, XML, DTD, pelit, motivointi, Moodle

Opetuspelien vaikutuksesta oppimiseen on tehty useita tutkimuksia. Niiden mukaan seikkailupelityyppiset opetuspelit tukevat oppimista. Pelin aikana saadut positiiviset kokemukset kannustavat jatkamaan pelaamista ja motivaatio opittavaa asiaa kohtaan kasvaa. Hyvä opetuspeli noudattaa päätös-käytös-palaute-sykliä, jossa käyttäjän omat reaktiot aiheuttavat tapahtumia ja palautteita.

Tampereen teknillisellä yliopistolla kehitettiin vuonna 2007 ohjelmointikieli, jolla pystyy laatimaan Moodle-oppimisympäristöön opetuspelejä, *sokkeloita*. Ohjelmointikieli on XML-pohjainen ja se on jossain määrin vaikeasti omaksuttavana este sokkelopelin laatimiseen etenkin sellaisille, joille ohjelmoinnin opettelu ei muuten ole tarpeellista. Näistä lähtökohdista heräsi tarve graafiselle työkalulle, jonka avulla sokkelopelin XML-kuvaus voitaisiin laatia ilman, että XML-koodia tarvitsisi kirjoittaa. Käyttäjä voisi syöttää pelin tehtävät ja niiden vastausvaihtoehdot sekä piirtää sokkelopelin, jonka jälkeen ohjelma automaattisesti tekisi XML-kuvauksen. Sovellus toteutettiin osin Ohjelmistotekniikan laitoksen virtuaaliyliopistohankkeen alla.

Ohjelma toteutettiin käyttäen Trolltechin ja Nokian sovellusympäristöä Qt. Tämä on alustariippumaton ohjelmistojen ja graafisten käyttöliittymien ohjelmistokehys, joka sisältää C++-luokkakirjaston lisäksi ohjelmointiympäristön. Qt sisältää valmiita funktioita muun muassa grafiikan, animaatioiden ja XML:n käsittelyyn. Toteutuksessa käytettiin Qt:n omia työkaluja sovellusteknisissä ratkaisuissa, kuten Qt:n omaa grafiikkatyökalua Graphics View Framework:iä graafinpiirtoalustana.

Ohjelmalle asetettiin tavoitteita helppokäyttöisyyden, alustariippumattomuuden ja kattavuuden suhteen. Alustariippumattomuus saavutettiin pääosin, mutta helppokäyttöisyyden ja matalan käyttöottokynnyksen tavoitteet jäivät saavuttamatta. Ohjelman jatkokehitys onkin suotavaa.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Information Technology
KYMÄLÄINEN-MÄKELÄ KATRIINA: Qt as a Drawing Program Platfom
Master of Science Thesis, 45 pages, 5 appendix pages.
Major: Software engineering
Examiner: professor Mikko Tiusanen
Keywords: QT, Graphics View Framework, canvas, XML, DTD, games, motivation, Moodle

Impact of educational games for learning has been the focus of several studies. According to them, an adventure-type educational game can support learning. Positive experiences during the game will encourage playing to continue and the motivation to learn the subject matter increases. A good educational game follows the decision-behavior-feedback cycle, where the user's own reactions cause events and feedback.

In 2007, a programming language was developed at Tampere University of Technology to create educational games, mazes, into the Moodle learning environment. The programming language is XML-based, making it somewhat difficult to adopt, particularly those for whom learning programming is not otherwise necessary. As such it is a barrier to creating a maze. From this perspective, there was a need for an easy graphical tool that would allow creating the XML description without writing XML code. The user could enter the game problems and their responses and then draw the maze, after which the program would automatically create the XML description. Such an application was created partly under a virtual university project of the Department of Software Systems.

The program was created using the Qt application environment of Trolltech and Nokia. This is a platform-independent software and graphical user interface framework that contains a C++ class library in addition to the programming environment. Qt includes built-in functions for graphics, animations, and XML processing. Application development used Qt's own tools, such as Qt's own graphics tool Graphics View Framework.

The program was set goals for ease of use, platform independence, and coverage. Platform independence was basically achieved, but not ease of use and low adoption threshold were not. Further development of the program is, therefore, desirable.

SISÄLLYSLUETTELO

TERMIT JA LYHENTEET

Graafinen käyttöliittymä	Graphical User Interface (GUI); käyttöliittymän muoto, jossa käyttäjä kommunikoi järjestelmän kanssa käyttäen visuaalisia kontrolleja tekstimuotoisten sijaan.
Item	Qt:n grafiikkanäkymäkehyksessä käytetty kaksiulotteinen objekti, jota voidaan muokata.
Näkymä	View; Qt:n grafiikkanäkymäkehyksessä käytetty näkymä, joka visualisoi näyttämön sisällön
Näyttämö	Scene; Qt:n grafiikkanäkymäkehyksessä käytetty alusta, jolle objektit voidaan sijoittaa.
Qt	Alunperin Trolltechin, nykyään Nokian kehittämä graafinen sovelluskehys, joka tukee useaa eri käyttöjärjestelmää [8].
SDK	Software Development Kit; kokoelma kehitystyökaluja, joilla voidaan kehittää sovelluksia tiettyyn ympäristöön.
Signaali-lokero -mekanismi	Signal-slot mechanism; Qt:n tapa käsitellä tapahtumia kuten hiiren painalluksia.
Vimpain	Widget; graafisen käyttöliittymän elementti, josta voidaan käyttää myös nimeä kontrolli.

1 JOHDANTO

Ohjelmointikoodia voidaan kirjoittamisen lisäksi tuottaa myös piirtäen. Ohjelman piirtäminen saattaa helpottaa ohjelman rakenteen hahmottamista ja toiminnan ymmärtämistä. Jos ohjelman kirjoittamisen opettelemiselle ei ole välitöntä tarvetta, voidaan piirtämistä käyttää apuna tuottamaan ohjelmakoodia. Scratch-ohjelmointikieli [1] on kehitetty juuri näistä lähtökohdista. Helppokäyttöisenä, visuaalisena ja värikkäänä se on houkutteleva ja ohjelman tekeminen tapahtuu helposti ja vaivattomasti, ajanvietteenomaisesti.

Erilaisten pelien pelaaminen mielletään myös usein ajanvietteeksi. Opetuspelien tavoitteena on yleensä kuitenkin motivoida pelaaja oppimaan jotain haluttua asiaa ja soveltamaan sitä. Erilaisen pelien käyttämiseen opetuksen apuvälineenä on löydetty hyviä perusteita; erityisesti Internetissä pelattavat pelit ovat helposti pelattavissa ja niihin sisältyy tehokkaita oppimismalleja, kuten tekemällä oppiminen ja tavoitteiden asettelu [2].

Moodle on avoimen lähdekoodin oliopohjainen oppimisalusta. Sitä käytetään monilla kursseilla TTY:ssäkin. Opettajan näkökulmasta Moodle on keino luoda ja organisoida kursseja verkkoon, opiskelijan näkökulmasta se on verkko-oppimisympäristö. Moodle on vapaasti saatavissa ja ladattavissa Internetistä ja sitä voi räätälöidä erilaisilla laajennuksilla. Moodle toimii laajennuksen sovelluskehyksenä ja tarjoaa sille rajapinnan. Eräs tapa laajentaa Moodlea ovat aktiviteettimoduulit. Moodlessa aktiviteeteiksi kutsutaan erilaisia toimintoja, joita opettaja voi lisätä kurssille opiskelijoiden suoritettaviksi. Tällaisia aktiviteetteja ovat esimerkiksi kyselyt, oppitunnit tai keskustelupalstat. Tarkemmat kuvaukset eri aktiviteeteista ja niiden hyödyntämisestä Moodle-ympäristössä löytyvät Moodlen dokumentaatiosta. [3]

Eräs Moodlen laajennus on Tampereen teknillisellä yliopistolla vuonna 2007 kehitetty *aktiviteettimoduuli maze*. Sen avulla pystytään laatimaan opetuspelejä, *sokkelopelejä* [4]. Nämä pelit toimivat esimerkiksi kurssien harjoitustehtävinä. Pelien kuvaamista varten on laadittu XML-pohjainen ohjelmointikieli [5]. Sokkelopeli muodostuu tehtävistä, niihin liittyvistä vastausvaihtoehdoista ja tehtävien välisistä siirtymistä. Tehtävät ja siirtymät muodostavat suunnatun graafin. Tässä toteutuksessa pelin siirtymät riippuvat aina pelaajan edelliseen tehtävään antamasta vastauksesta ja kustakin vastauksesta voi seurata erilaisia vaihtoehtoisia siirtymiä. Peli voi haarautua puumaisesti, jopa palata takaisin samaan, mikä juuri tekee pelimaailmasta sokkelon.

Sokkelopelejä on ollut olemassa jo 1970-luvun loppupuolelta lähtien, esimerkiksi tekstiseikkailupeli Zork [6] kehitettiin vuosina 1977–79 ja julkaistiin vuonna 1980.

Moodle-ympäristössä sokkelopeleihin tarvittava XML-pohjainen merkkauskieli saattaa olla vaikeasti hahmotettavissa ja sen opettelu- sekä käyttöönottokynnykset voivat olla korkeat. Pelin laatijan työtä voitaisiin helpottaa helppokäyttöisellä graafisella työkalulla, jolla voitaisiin tuottaa sokkelopelin XML-kuvaus.

Tässä työssä kuvataan toteutettu sovellus XML-dokumenttien piirtelyä varten. Sovelluksen tulee helpottaa sokkelopelissä vaadittavan XML-dokumentin luomista. Piirto-ohjelman olisi oltava

- helppokäyttöinen,
- helposti opittavissa,
- alustariippumaton ja
- käyttöönottokynnykseltään matala.

Lisäksi sovelluksen tulee omata riittävä dokumentaatio. Pelin laatijan tulee pystyä kattavasti rakentamaan haluttu peli piirto-ohjelman avulla.

Piirto-ohjelman käyttäjän kannalta ohjelman helppokäyttöisyys tarkoittaa, että sovellus automatisoi XML-dokumentin tuottamisen mahdollisimman pitkälle. Käyttäjän tehtäväksi jää tehtävien ja vastausvaihtoehtojen syöttö sekä graafin piirtäminen. Alustariippumattomuus tarkoittaa, että piirto-ohjelma ei ole riippuvainen siitä ympäristöstä, jolle se on toteutettu, vaan se on siirrettävissä eri ympäristöihin. Piirto-ohjelman helppo opittavuus tarkoittaa, että sen käyttö on mahdollisimman helppoa ja suoraviivaista ja ohjelmaan liittyvä dokumentaatio on ajan tasalla ja oikeellista. Hyvällä suunnittelulla ja toteutuksella varmistetaan piirto-ohjelman matala käyttöönottokynnys.

Ohjelman toteuttamiseksi harkittiin erilaisia sovellusaluekieliä (*Domain Specific Language, DSL*), joita tyypillisesti käytetään häivyttämään sovelluksen käyttäjältä ohjelmistotekniset yksityiskohdat ja auttamaan käyttäjää keskittymään kohteena olevan sovellusalueen hahmottamiseen [7]. Sovellusaluekielet eivät ole mitenkään uusi keksintö, esimerkiksi raskasta tieteellistä laskentaa vaativissa tehtävissä käytetty FORTRAN on tässä mielessä sovellusaluekieli [8]. Sokkelopelin laatimiseen käytettävä merkkauskieli ei rajaa toteutusalustavaihtoehtoja, joten sovellusaluekielen kehittämiseen voi tässä tapauksessa työkalun valita melko vapaasti. Eräitä esimerkkejä näistä ovat Eclipse-ympäristön Generic Eclipse Modeling System (GEMS) [9], Microsoftin kehittämä meta-sovellusaluekieli ja kehitysympäristö DSL Tools [10] sekä Trolltechin kehittämä Qt [11].

DSL Tools on osa Microsoft Visual Studion [12] Software Development Kit:iä (SDK) ja se on vapaasti ladattavissa Microsoftin sivuilta. Sen käyttöönotto vaatii kuitenkin Visual Studiosta vähintään professional-tasoisen asennuksen. Käytännössä tämä tarkoittaa, että DSL Tools on maksullinen, mikä vähentää sen houkuttelevuutta.

Vuonna 2009 Microsoft ilmoitti, että se yhdistää DSL Toolsin ja toisen olemassa olevan sovelluskuvauskielen ja alkaa kehittämään näitä yhdessä. Uuden sovelluksen nimi on SQL Server Modeling Services. [13.] Koska odotettavissa oli suuria muutoksia DSL Toolsin toimintaan, hylättiin se toteutusvaihtoehtona.

Kehitysympäristö Eclipsen [9] sovellusaluekieli on Generic Eclipse Modeling System (GEMS). Se on ilmainen ja yhdistää Eclipsen metamallityökalut. Se on avointa lähdekoodia, joten periaatteessa jokainen pystyy halutessaan kehittämään sitä.

GEMSin työkaluista toteutusalustavaihtoehtona tutkittiin GMF:ää (Graphic Modeling Framework) [14]. Se kuitenkin todettiin hyvin raskaaksi käyttää, sillä sitä ajetaan Eclipsessä Eclipsen päällä. Lisäksi GMF osoittautui lisäksi poikkeuksellisen epävakaaksi alustaksi. Dokumentaatio oli hyvin puutteellista, ja huomionarvoista on, että GEMSiä ei juurikaan ole kehitetty vuoden 2008 jälkeen. Näiden vuoksi GEMS hylättiin toteutusvaihtoehtona.

Qt [15] on laaja C++-kehitysympäristö. Sillä voidaan tehdä monimutkaisiakin graafisia sovelluksia, jotka toimivat usealla eri toteutusalustalla (Windows [16], Unix [17], MacIntosh [18]). Käyttöliittymäkirjaston lisäksi Qt tarjoaa kirjastot tietoliikenneverkkojen, XML:n, SQL:n [19] ja OpenGL:n [20] käytölle. Qt on ilmainen ja sen saa ladata suoraan Internetistä [21]. Qt sisältää paljon valmiita komponentteja sekä valmiin kehitysympäristön, Qt Creatorin [22]. Niinpä Qt valittiin toteutusalustaksi harkituista vaihtoehdoista lupaavimpana. Tarkemmin sitä käydään läpi luvussa 5.

Luku 2 käy läpi tutkimustuloksia pelien käyttöön oppimisen ja motivoinnin apuvälineenä. Luku 3 esittää lyhyesti XML-kielen. Luku 4 käy läpi sokkelopelin ja siihen liittyvän XML-dokumentin. Luku 5 sisältää kuvauksen pelin toteutusalustasta, QT:sta ja luvussa 6 käydään läpi toteutettu sovellus. Luvussa 7 on työn arviointi sekä jatkokehitysajatukset. Luvussa 8 yhteenveto

2 PELIT JA OPPIMINEN

Tietokonepelit ovat ajanvietettä. Niiden parissa saatetaan viettää tuntikausia ilman, että ajan kulumista suuremmin edes huomaa. Tietokonepelien avulla ja niiden sisällä tapahtuvaa oppimista on tutkittu ja tutkijat ovat yrittäneet löytää mahdollisuuksia soveltaa tietokonepelien viehätystä myös opetukseen.

Esimerkiksi historian oppiminen sujuu tutkimuksen mukaan pelin kautta. Wisconsinin yliopistossa tehdyn tutkimuksessa tutkittiin pelien *Ages of Empires* ja *Ages of Mythology* yhdistämistä mytologiaa käsittelevien kirjojen lukemiseen ja peleihin sekä kirjallisuuteen liittyvien kuvien piirtämiseen ja tarinoiden kirjoittamiseen. Tutkimus osoitti, että pelien liittäminen osaksi oppimistapahtumaa auttoi yksityiskohtien sitomista todellisiin tapahtumiin. Lisäksi oppimista tapahtui vapaa-ajalla eikä ainoastaan koulussa. [23]

2.1 Tietokonepelit, oppiminen ja motivaatio

Motivaatio on tärkein asia oppimisessa. Motivaation kadotessa tai loppuessa myös oppiminen ja pelaaminen loppuvat.

Hyvin suunnitelluista ja toteutetuista tietokonepeleistä voidaan analysoida, miten motivaatiota kasvatetaan ja pidetään korkealla: ne antavat pelaajalle vihjeitä ja informaatiota oikeassa kohdassa, silloin kun he sitä tarvitsevat. Normaalissa oppimistilanteessa, esimerkiksi koulussa, tieto saatetaan esittää erillään asiayhteydestä tai oppijan tavoitteesta. Oppija saattaa myös sivuttaa tiedon turhana tai asiaankuulumattomana, mikäli se esitellään väärään aikaan tai erillään asiayhteydestä. Hyvin laaditut tietokonepelit myös vaikeutuvat pelin edetessä, jolloin ne asettavat koko ajan uusia haasteita. Kouluissa oppiminen tapahtuu usein heikoimman oppilaan mukaan, jolloin nopeammin etenevät oppilaat voivat turhautua. Hyvä tietokonepeli sopeutuu pelaajan vaikeustasoon ja muokkautuu sen mukaisesti. [23]

Hyvässä tietokonepelissä on usein mukana tekoälyä, joka oppii lisää, kun peliä pelataan, ja osaa soveltaa oppimaansa käytäntöön. Tällöin pelaaja kokee saavansa lisää haastetta, kun peli vaikeutuu ja onnistumisen elämyksiä, kun vaikeutumisesta huolimatta pystyy pelissä tehtäviä suorittamaan. Jos tämä oppimisominaisuus pelistä puuttuu, sitä ei välttämättä pelattaisi yhtä kauaa tai samalla intensiteetillä. Pelejä voisi tietysti helpottaa, mutta useimmat pelaajat eivät halua helppoja ja yksinkertaisia pelejä, vaan niissä pitää olla haastetta. Tässä piilee samalla opetusalalla laajalti vallitseva

ongelma: miten saada erityisesti nuoret ihmiset innostumaan jostakin vaikeasta ja haastavasta ja samalla nauttimaan siitä. [23]

Tietyt pelityypit soveltuvat oppimiseen toisia paremmin. Osana tutkimusta [24] ryhmä opiskelijoita pelasi neljää erityyppistä kaupallista tietokonepeliä. Tutkimus osoitti, että opiskelijat pitivät seikkailu- ja strategiapeleistä enemmän kuin ampumispeleistä. Tutkimus osoitti pelin tärkeimmiksi elementeiksi logiikan, ongelmanratkaisun ja visuaalisuuden. Näitä samoja elementtejä tarvitaan myös oppimisprosessissa. Kun ymmärretään opetustavoitteiden ja yllä mainittujen pelielementtien suhde, voidaan kehittää hyviä opetuspelejä, jotka sisältävät yllä mainitut komponentit. [24]

Yleensä ottaen oppiminen tapahtuu käytännön kautta. Esimerkiksi kemian tunnilla oppiminen tapahtuu parhaiten, jos oppilaat voivat olla, työskennellä ja ajatella kuin kemistit. Tätä tapahtuu myös pelaamisen yhteydessä; pelaaja on kuten pelin häntä edustava roolihahmonsa. Monipeli on yksi pelaamisen muoto, jota voidaan tehokkaasti yhdistää oppimisessa. Monipeli yhdistää pelaamiseen sosiaalisen ympäristön, jossa pelaajat voivat jakaa informaatiota keskenään. Tästä voidaan vetää yhteys jopa työelämään ja siellä tapahtuvaan oppimiseen. [23].

Oppiminen on tehokasta silloin, kun se tapahtuu leikin varjolla, eikä oppija edes välttämättä havaitse oppimista. Leikkimisen ja oppimisen välillä onkin osoitettu olevan yhteys. Tietokonepelit kannustavat oppimaan luovuuden ja kokeilun kautta, leikkien, ja pelit usein sisältävät ongelmia, joiden ratkaisu vaatii opitun asian omaksumista ja soveltamista. [24]

2.2 Pelien komponentit

Hyvän pelin ominaisuudet ovat olleet tutkijoiden mielenkiinnon kohteina, sillä niille on ollut hyvin vaikeaa löytää mitään yhteistä tekijää. Syitä tähän on useita, mutta todennäköisin syy on eri terminologia samoille asioille. Samaa tarkoittavat asiat on ilmaistu eri tavoin, joten asioita ei ole osattu yhdistää [25]. Pelin komponentit voidaan karkeasti jakaa kuuteen eri kategoriaan: fantasia, säännöt/päämäärät, mysteeri, hallinta, aistiärsykkeet ja haasteet. Jokaisesta pelistä voidaan tutkimuksen [25] mukaan osoittaa nämä elementit.

Fantasiaelementti pelissä tarkoittaa usein sitä, että pelin tapahtumat on sijoitettu kuvitteelliseen maailmaan, jolla ei ole vastinetta todellisessa maailmassa. Pelissä tapahtuvilla asioilla ei siis ole vastinetta reaalimaailmassa. Joidenkin tutkimusten mukaan fantasian avulla esitetty opetuksellinen materiaali edesauttaa mielenkiinnon heräämistä ja oppimista, esimerkiksi fysiikkaa voidaan opettaa ohjaamalla avaruusalusta maata ympäröivällä kiertoradalla.

Peliin on useimmiten määrätty tapahtumaympäristö ja aikaikkuna, jonka puitteissa peli täytyy suorittaa. Tällainen aikaikkuna kannustaa suorittamaan tehtävän annetussa ajassa. Pelin säännöt asettavat myös pelin päämäärän. Tutkimuksissa on havaittu

huomattava yhteys selkeän päämäärän, sääntöjen ja korkean motivaation välillä. Spesifi ja riittävän haastava määränpää kasvattaa suorituskykyä ja samalla motivaatiota. Sääntöjen täytyy kuitenkin olla joustavia esimerkiksi pelaajan omien valintojen, strategian ja muiden tekijöiden suhteen.

Tutkimuksen mukaan tärkein oppimista ohjaava tekijä on uteliaisuus. Sitä on kahta tyyppiä: aistinvarainen ja kognitiivinen eli tiedonhalu. Tutkijat ovat yhtä mieltä siitä, että uteliaisuus liittyy ihmisen luontaiseen haluun selvittää itselleen tuntemattomia asioita ja uusia tapahtumia. Pelissä tulee siis olla riittävä määrä uutta, mielenkiintoista opittavaa, mysteeriä.

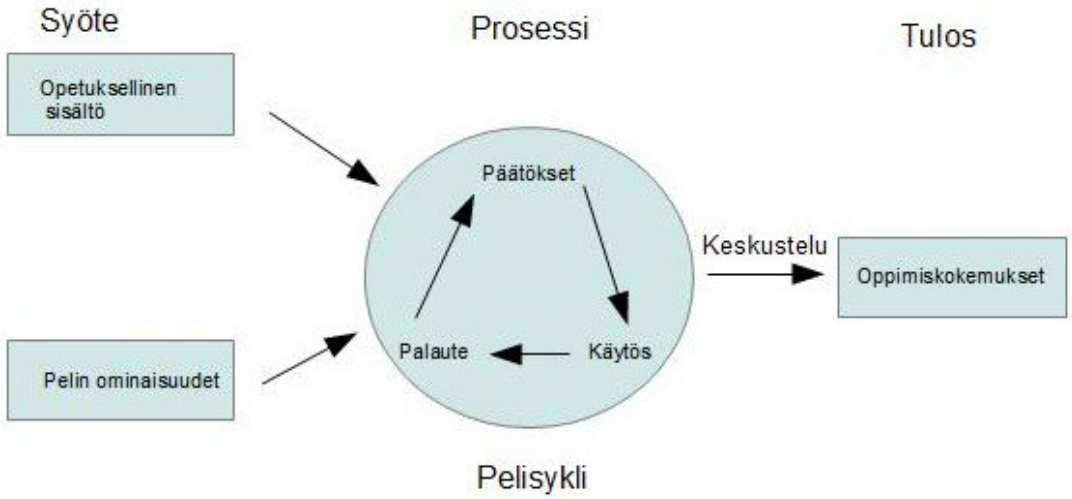
Hallinta liittyy ihmisen tarpeeseen hallita, ohjata tai käskeä jotakin. Kuitenkin tutkimukset, joissa on verrattu täysin käyttäjän hallinnassa olevia ja osittain tietokoneen hallussa olevia pelejä, ovat antaneet ristiriitaista tietoa siitä, kummalla tavalla saadaan parempi oppimistulos. Toisaalta, tutkimus, jossa verrattiin käyttäjän motivaatiota näissä peleissä osoitti, että käyttäjä haluaa itse hallita peliä ja sen tapahtumia.

Pelissä olevat aistiärsykkeet valmistavat pelaajaa kokemaan mahdollisesti samankaltaisia ärsykkeitä myös tosielämässä ja harjoittelemaan reaktioita niihin. Aistiärsykkeitä voivat olla erilaiset äänet, grafiikat ja muut, joko tutut tai tuntemattomat, huomiota herättävät asiat. Tutkimuksessa havaittiin grafiikan lisäämisen peliin parantavat pelin mielenkiintoa ja kasvattavan käyttäjän halua jatkaa pelaamista. [25]

2.3 Pelin oppimissykli

Opetusmalleissa on tapahtunut muutos perinteisestä, didaktisesta opetusmallista aktiiviseen oppijakeskeiseen malliin. Sen sijaan, että oppiminen tapahtuisi kuuntelemalla, se tapahtuu itse tekemällä. Interaktion mahdollistavat tekniikat antavat mahdollisuuden luoda oppimisympäristöjä, jotka aktiivisesti kannustavat oppijaa ongelmanratkaisuun. Empiiriset tutkimukset osoittavat myös, että pelit voivat olla tehokkaita työkaluja monimutkaisen asioiden ymmärtämiseen. [25]

Kuva 1 esittää oppimismallin, jota on käytetty onnistuneesti useissa opetuspeleissä. Oppimismalli koostuu kolmesta osasta; syöte, prosessi ja tulos. Syötteeseen sisältyy kaksi eri osa-aluetta, opetuksellinen sisältö ja pelin ominaisuudet. Nämä yhdistetään ja lopputuloksena syntyy prosessi, joka muodostaa varsinaisen pelisyklin. Syklissä toteutuu päätös-käytös-palaute-sykli. Päätös tarkoittaa käyttäjän päätöstä tehdä jokin toimenpide, päätös, pelin sisällä. Käytös on seurausta päätöksestä, jonka seurauksena syntyy puolestaan palaute. Palautteen perusteella käyttäjä tekee uusia päätöksiä. Aiemmat tapahtumat siis ohjaavat käyttäjän toimintaa. Pelisyklin jälkeen tapahtuu palautekeskustelu, josta syntyy tuloksena oppimiskokemuksia.



Kuva 1: Pelaamisessa tapahtuva oppimissykli. Mukaillen [25].

Pelin toteutus alkaa pelin suunnittelusta; se tulee laatia siten, että pelin luonnollinen tapahtumasykli tukee pelin opetuksellista sisältöä. Pelin aikana tapahtuvan päätös-käytös-palaute-prosessin lopputuloksena ovat oppimiskokemukset. Tämä painottaa pelin aikana tapahtuvaa oppimista, kun taas perinteinen opetus keskittyy enemmän ennen soveltamishetkeä tapahtuvaan opetukseen. Oppimismallissa suurin osa oppimisesta tapahtuu pelin aikana, sillä oppimista ohjaavat omat valinnat. [25]

Kuvan 1 syklissä on esitetty keskustelu (debriefing) pelisykliä ja oppimiskokemusta yhdistävänä tekijänä. Monen mielestä oppimistapahtuman jälkeen tapahtuva keskustelu on kriittisin osa pelikokemusta. Se on katsaus ja analyysi pelissä olleissa tapahtumista. Keskustelun tarkoitus on käydä läpi pelin tapahtumia, joita ei sellaisenaan voida suoraan soveltaa reaalimaailmaan. Peli voi sisältää elementtejä, kuten pisteiden laskeminen, joita ei suoranaisesti reaalimaailmassa esiinny. Erityisesti opetuspeleissä keskustelulla on tärkeä rooli, sillä tällä mahdollistetaan lopullisesti pelitapahtumien muokkaaminen oppimistapahtumiksi. Keskusteluun voi sisällyttää tapahtumakuvauksen, analyysin siitä, miksi niin tapahtui ja keskustelua virheistä ja korjaavista toimenpiteistä. Tutkimus painottaa, että tekemisen kautta oppiminen tarvitsee tukea. On epärealistista odottaa edes hyviltä oppijoilta sitä, että uusi tieto voitaisiin jäsentää, omaksua ja soveltaa ilman keskustelun tarjoamaa tukea. [25]

2.4 Opetuspelien edut

Pelit luovat oppimisympäristön, jossa käyttäjät voivat vapaasti ja ilman epäonnistumisen pelkoa kokeilla erilaisten tehtävien suorittamista. Pelit, joiden vaikeustaso kasvaa pelaamisen edetessä, antavat käyttäjälle mahdollisuuden kerätä luottamusta ja kehittää taitojaan. Tämä kasvattaa käyttäjän luottamusta omiin kykyihinsä myös tosielämän puolella, etenkin jos pelissä suoritettavat tehtävät ovat haastavia, stressaavia tai

vaarallisia. Henkilöt, joilla on luottamus omiin kykyihinsä, pystyvät kohtamaan tosielämän haasteita paremmin, kun pelimaailmassa opittuja ongelmanratkaisukeinoja voidaan soveltaa myös reaalimaailmassa. [25]

Tutkimus [25] osoittaa tehtäviin osallistumisen olevan se, johon yksilöt keskittyvät ja mikä saa heidät innostumaan aktiviteetistä. Tehtäviin osallistuminen muokkaa tehtävistä entistä mielenkiintoisempia ja mukaansatempaavampia. Tämänkaltainen peleissä koettava tehtäviin uppoutuminen voi johtua useista eri tekijöistä. Välittömät ja luonnolliset reaktiot tapahtumaan, aistihavainnot ja -ärsykkeet, häiriötekijät, jotka irrottavat käyttäjän fyysisestä ympäristöstään, ja tapahtuman realistisuus ovat olennaisia tekijöitä tehtävään uppoutumisessa. Tutkimus osoittaa oppimisen ja informaation omaksumisen parantuvan kognitiivisen sitoutumisen lisääntyessä.

Pelin aikana tehdyt päätökset vaikuttavat myöhempään käytökseen pelissä. Motivoituneet oppijat ovat valmiimpia osallistumaan aktiviteetteihin, he jatkavat innokkaammin ja keskittyvät kauemmin kuin vähemmän motivoituneet oppijat. Lyhyesti sanottuna, innostuneet oppijat ovat kiinnostuneempia ja osallistuvampia aktiviteetteihin ja he ovat myös sitoutuneempia suorittamaan aktiviteetin loppuun. Tämä on se kulmakivi, joka tekee tietokonepeleistä koukuttavia ja edustaa sitä, mitä pelien suunnittelijat hakevat takaa. Ne oppijat, jotka kokevat positiivisia asioita pelin aikana, sitoutuvat siihen, keskittyvät paremmin ja intensiivisemmin ja palaavat pelin pariin.

Palaute tai tietoisuus tuloksista on tärkeä asia ylläpidettäessä suorituskykyä ja motivaatiota. Tutkimukset osoittavat, että reaktio palautteeseen vaihtelee; joissakin tapauksissa palaute nosti suorituskykyä, toisissa tapauksissa laski. Kuitenkin palaute on yksiselitteisesti tärkeässä osassa.

Yksilölliset päätökset ja käytös sääntelevät palautteen ymmärtämistä. Jos palautteesta on tulkittavissa suorituskyvyn olleen jatkuvasti hyvin korkea, peli voidaan kokea liian helpoksi ja motivaatio pienenee. Yleensä palaute toteaa tämänhetkisen suorituskyvyn olevan keskimääräistä alempana, joka saattaa aiheuttaa motivaation laskun, mikäli käyttäjä kokee tämän virheellisenä tulkintana suorituskyvystään. Tällöin käyttäjä saattaa kokea tämän negatiivisena asiana ja lopettaa pelaamisen. Toinen yleinen reaktio on entistäkin tarmokkaampi, keskittyneempi ja määrätietoisempi suhtautuminen ongelmaan. Palaute on siis hyvin tärkeä osa oppimiskokemusta ja parhaassa tapauksessa se opastaa motivoitunutta käyttäjää keskittymään entistä paremmin tehtävään.[25]

Erityyppiset oppimiskokemukset jaetaan karkeasti kolmeen eri kategoriaan: taitoperustaiseen, kognitiiviseen ja affektiiviseen. Taitoihin perustuva oppimiskokemus keskittyy teknisiin tai motorisiin taitoihin ja niiden parantamiseen. Esimerkiksi taitolentäjät, jotka harjoittelivat simulaattorilla onnistuivat testilennoissaan huomattavasti paremmin kuin he, jotka eivät simulaattoriopetusta saaneet. Affektiivinen oppimiskokemus painottaa tunteita, esimerkiksi luottamusta, itsetuntoa ja asenteita ja näiden muuttamista tai vahvistamista. Pelejä voidaan käyttää esimerkiksi valistustyökaluna. Deklaratiivinen oppimiskokemus tarkoittaa tiedonhakua tehtävän suorittamista varten ja

usein myös tiedon soveltamista käytäntöön. Menetelmiin pohjautuva oppimiskokemus esimerkiksi edesauttaa oppijaa ymmärtämään teorian ja käytännön yhteyden demonstraation avulla. Strateginen oppimistulos taas vaatii opittujen asioiden omaksumista ja niiden liittämistä eri asiayhteyksiin.[25]

3 XML

World Wide Web Consortium (W3C) on kansainvälinen yritysten ja yhteisöjen yhteenliittymä, joka ylläpitää ja kehittää WWW:n standardeja eli *suosituksia*. Se on perustettu vuonna 1994 ja sen perustaja, Tim Bernes-Lee, toimii edelleen yhteenliittymän puheenjohtajana. W3C:llä on yli 400 jäsentä.[26]

Extensible Markup Language (XML) on W3C:n määrittelemä suositus tekstimuotoisten rakenteisten dokumenttien jäsentämiseen, merkkauskieli. Sitä käytetään erityisesti WWW-ympäristöissä ja paikoissa, joissa on paljon jäsennettävää, keskenään samankaltaista tietoa. XML on periaatteessa sekä ihmisen että tietokoneen luettavissa oleva formaatti. XML:n avulla voidaan erottaa itse tietosisältö käyttäjälle näkyvästä esitystavasta. XML:n kehitti W3C:n alainen XML-työryhmä vuonna 1996. [5]

XML-kielelle asetettiin tavoitteita:

- olla suoraviivainen ja helppokäyttöinen,
- tukea erilaisia sovelluksia,
- olla yhteensopiva SGML-kielen [27] kanssa ja
- syntaksin tulee olla muodollinen ja ytimekäs. [5]

XML-suosituksen lisäksi muita W3C:n julkaisemia suosituksia ovat esimerkiksi CSS [28], HTML [29] ja XHTML [30].

3.1 XML-dokumentti

XML on osajoukko HTML-kielen taustalla olevasta SGML-kielestä. Niin kuin HTML-kielessä, myös XML-dokumentin sisältämä tieto koodataan elementteihin (tag) ja attribuutteihin. Elementit merkitään alku- ja loppumerkeillä, jotka kumpikin laitetaan kulmasulkeiden (”<” ja ”>”) väliin. Attribuutit, jotka sisältävät kyseistä elementtiä kuvaavaa tietoa, kirjoitetaan elementin alkumerkin sisään sen nimen jälkeen. [5]

```
<?xml version="1.0"?>
<greeting>Hei, maailma!</greeting>
```

Yllä on yksinkertainen XML-esimerkki [5]: ensimmäinen rivi on niin sanottu XML-julistus (declaration). Se kertoo, että dokumentti on XML-muodossa ja että sen suositeltu versio (annettu attribuutin 'version' arvona) on tässä tapauksessa 1.0. Tämän jälkeen on yksi elementti nimeltä 'greeting', jonka sisältönä on teksti ”Hei, maailma!”

XML-dokumentti on rakenteeltaan hierarkkinen; elementit voivat sisältää toisia elementtejä, jotka voivat edelleen sisältää toisia elementtejä. XML:n yhteydessä käytetäänkin paljon samaa terminologiaa kuin puutietorakenteiden yhteydessä [5], kuten alla.

Jokaisella hyvin muodostetulla XML-dokumentilla on aina juurielementti, jonka jälkeläisiä kaikki sen sisältämät elementit ovat. Jokaisella elementillä – paitsi juurella – on yksi välitön edeltäjä, vanhempi. Mikäli kahdella elementillä on yhteinen vanhempi, ovat ne sisaruksia ja vanhemman välittömiä jälkeläisiä, lapsia; kaikki sisarukset ovat siis vanhemman alku- ja loppumerkkien välissä peräkkäin, ja vanhempi on *sisin* ne näin sisäänsä sulkeva elementti. Elementti, jolla ei ole lapsia, on lehti. [5]

3.2 DTD

XML ei määrittele, mitä elementtejä käytetään ja missä järjestyksessä, vaan se määrittelee ainoastaan näiden ulkoasun. Tässä mielessä se on vain merkkauskieliperhe. Tarkempi hierarkkinen rakenne, siis käytettävät elementit, näiden alkumerkkien attribuutit ja se, miten elementit voivat olla peräkkäin tai sisäkkäin, voidaan määritellä tähän tarkoitukseen kehitetyllä ja XML-suositukseen sisältyvällä Document Type Definition (DTD) -kielellä [31]. DTD voidaan kirjoittaa joko XML-dokumentin alkuun tai omaksi tekstidokumentikseen. Jälkimmäinen tapa on yleiskäyttöisyytensä vuoksi suositeltava.

DTD:ssä siis määritellään elementit ja niiden attribuutit sekä hierarkia:

```
<!ELEMENT choice    (show, response)>
<!ATTLIST choice      name        ID          #REQUIRED
                      breakskey   CDATA       #IMPLIED>
```

Tämä DTD määrittelee elementin *choice*. Sillä on kaksi lapsielementtiä, nimittäin *show* ja *response,* tässä järjestyksessä peräkkäin. Näiden lisäksi elementillä on kaksi pakollista attribuuttia, *name* ja *value.* Attribuutti *name* eroaa muista attribuuteista siten, että sen arvon tulee olla yksikäsitteinen kyseisen dokumentin sisällä. Tämän määrää käytetty tyyppi **ID**. Yksikäsitteisyys mahdollistaa elementtiin viittaamisen jostakin toisesta elementistä. Jotta näin voidaan tehdä, tulee sille DTD:ssä määritellä tyyppiä **IDREF** oleva attribuutti, viittaus. Tämän arvona tulee olla jonkin toisen, samassa dokumentissa määritellyn, tyyppiä **ID** olevan attribuutin arvo. Viimeisen attribuutin, *breakskey*, arvona voi olla mikä tahansa merkkijono (määre **CDATA**).

Kun dokumentin DTD sisältää tarvittavien elementtien ja attribuuttien määrittelyt sekä säännöt elementtien välisistä viittauksista, voidaan sen avulla tarkistaa XML-doku-

mentin rakenteen oikeellisuus ja viittausten eheys. Viittaus on eheä, mikäli sen kohteena oleva elementti on olemassa ja yksikäsitteinen.

4 SOKKELO

Sokkelo rakentuu XML-pohjaisen merkkauskielen ja siihen liittyvän DTD-dokumentin avulla. Tässä luvussa olevat esimerkit ovat liitteestä 2 löytyvästä esimerkkitiedostosta, joka kuvaa yksinkertaisen pelin. Merkkauskieli on selitetty tarkemmin viitteessä [4].

Juurielementti – *problems*

Hyvin muodostetussa XML-dokumentissa tulee olla yksi juurielementti, jonka lapsia tai jälkeläisiä kaikki muut elementit ovat. Tehtävätiedostossa tämä elementti on *problems*. Elementissä voidaan määritellä pelille kesto sekunteina, jonka rajoissa peli on suoritettava. Ajanlasku alkaa siitä hetkestä, kun pelaaja siirtyy ensimmäiseen tehtävään.

```
<problems xmlns="http://www.cs.tut.fi/~fyzix/" timelimit="60"
  <!---tähän muu peli-----></problems>
```

Yllä olevassa esimerkissä elementin *problems* sisään, siis alkumerkin jälkeen mutta ennen sen loppumerkkiä, määritetään peliin liittyvien elementtien nimiavaruus (XML NameSpace, xmlns [5]). Sitä käytetään erottamaan pelin elementit muualla määritellyistä samannimisistä elementeistä. Esimerkissä on määritelty pelin keston lisäksi myös WWW-osoite, josta voi hakea lisätietoa pelistä. Esimerkissä on lisäksi kommentti, joka aloitetaan kulmasululla, huutomerkillä ja kahdella tavuviivalla, ja päätetään kahdella tavuviivalla ja kulmasululla.

Asetukset – *settings*

Elementti *settings* määrittelee sokkelolle asetuksia. Näitä ovat logo ja kuvakkeet, jotka näytetään pelaajan vastattua oikein tai väärin.

```
<settings logo="fyzix-logo.png" correctimg="correct.gif"
  wrongimg="wrong.gif"/>
```

Tässä määritellään pelin logoksi *fyzix-logo.png*, oikeasta vastauksesta näytettäväksi kuvakkeeksi *correct.gif* ja väärästä näytettäväksi *wrong.gif*. Alkumerkin lopussa oleva kauttaviiva korvaa loppumerkin, kun elementin sisältö on tyhjä.

Parametrit – *param*

Parametrit ovat sukua ohjelmointikielen muuttujille, sillä kullakin on näkyvyysalue, joka riippuu siitä, missä kohtaa tiedostoa se on määritelty. Mikäli parametri määritellään dokumentin juuressa, on kyseessä globaali parametri, jota voi käyttää kaikissa tehtävissä. Tehtävän sisällä määritelty parametri näkyy vain kyseiseen tehtävään. Parametrilla tulee olla nimi, jonka ID-tyyppisenä attribuuttina tulee olla dokumentin sisällä yksikäsitteinen, ja arvo.

Parametrien arvot voivat olla numeerisia, merkkijonoja tai taulukoita. Arvoksi määritellään elementin 'param' sisältämä merkkijono, ja elementillä *use* sen arvoa voidaan käyttää halutussa kohtaa tekstiä. Jos käyttöyhteys vaatii numeerista arvoa, arvon on oltava tällainen.

Taulukoita käytetään esimerkiksi merkkijonojen satunnaistamiseen. Taulukkoon määritellään halutut merkkijonot, joista voidaan hakea satunnainen alkio sopivalla funktiolla.

```
<param name="g"
  description="Satunnaisen planeetan gravitaatiokiihtyvyys">
  <function name="random"/>(9.81, 100)
</param>

<param name="autot">
  <item>saab</item>
  <item>volvo</item>
</param>

<param name="kertoimet">
  <item>1.5</item>
  <item>10</item>
</param>
```

Yllä on määritelty peliin globaaleja parametreja. Ne näkyvät kaikkiin tehtäviin. Ensimmäinen parametri *g* on määritelty siten, että se saa satunnaisen arvon annetulta väliltä (9.81, 100) rajat mukaan lukien. Tällöin parametrin arvo vaihtelee pelaajakohtaisesti. Muut parametrit, *autot* ja *kertoimet*, ovat taulukoita. Niissä voi olla sisältönä merkkijono, kuten taulukossa *autot*, tai numeerisia arvoja, kuten taulukossa *kertoimet*.

Kehyskertomus – *framestory*

Kehyskertomuksen tarkoituksena on toimia pelin alkupisteenä ja pohjustaa sokkelon juoni pelaajalle. Kehyskertomukseen voi sisällyttää tekstin lisäksi kuvia ja videoita. Elementin attribuutteihin määritellään ne suunnat, joihin pelaaja voi siirtyä. Tämä elementti siis edustaa pelimaailman graafin juurta.

```
<framestory forward="teht1_i1">
  <figure>Hiittinen.jpg</figure>
   "...siellä kaiikilla oli ni-iin muukaavaa..."
</framestory>
```

Esimerkissä on kehyskertomukseen liitetty kuva, *Hiittinen.jpg* sekä tekstiä. Lisäksi kehyskertomukseen on liitetty suunta, joka tässä tapauksessa on eteenpäin tehtävän instanssiin eli ilmentymään nimeltä 'teht1_i1': *forward=”teht1_i1”*.

Pelin loppuminen – *success* ja *failure*

Kehyskertomuksen lisäksi juonen kannalta oleellisia ovat pelin päätepisteet. Elementeillä *success* ja *failure* voidaan määritellä, mitä pelaaja näkee pelatessaan pelin läpi onnistuneesti (*success*) tai epäonnistuessaan (*failure*).

```
<success name="onnistui">
  <figure>keto.jpg</figure>
  "...and they all lived happily ever after."
</success>
<failure name="poks">
  <figure>sieni.jpg</figure>
</failure>
```

Tässä esimerkissä elementin pelin onnistuneessa suorituksessa (*success*) näytetään kuva (*keto.jpg)* ja tekstiä. Epäonnistunut suoritus esittää kuvan *sieni.jpg*.

Tehtävä – *problem*

Tehtävänanto ja sen parametrit kirjoitetaan elementin *problem* sisään. Jokaiselle tehtävälle tulee antaa yksikäsitteinen nimi. Tehtävälle voidaan antaa myös aihepiiri, johon se kuuluu. Aihepiireihin voi liittää osoitteen Internet-sivulle (*Uniform Resource Locator,* URL [32]), josta voi etsiä materiaalia tehtävän ratkaisemiseen.

```
<problem topic="testi" name="teht1">
  <param name="pakko"/>
  <param name="mestarit">
    <item>Pablo Picasso</item>
    <item>Leonardo Da Vinci</item>
  </param>
  <param name="kysymys">
    <function name="array_rand"/>(<use   name="autot"/>)
  </param>
  <param name="kerroin">
    <function name="array_rand"/>(<use name="kertoimet"/>)
  </param>
  <param name="mestari">
    <function name="array_rand"/>(<use name="mestarit"/>)
  </param>
  <param name="mjono" readonly="true">
    -= <use name="kysymys"/> =-
  </param>
```

Tässä esimerkissä tehtävälle on annettu aihepiiri (*testi*) ja nimi (*teht1*). Sen lisäksi tehtävän yhteydessä on määritelty erilaisia parametreja samoin kuin globaalien parametrien yhteydessä. Tässä on määritelty tyhjä parametri (*pakko*), jolle on pakko asettaa arvo instanssin yhteydessä elementillä *set*. Parametri *mestarit* määritellään täysin samoin kuin globaali parametri.

Taulukkomuuttujia (*kysymys, kerroin* ja *mestarit*) käsitellään *array*-funktioilla. Parametrin *kysymys* yhteydessä tämä tarkoittaa, että funktio *array_rand* palauttaa satunnaisen alkion taulukosta *autot*. Taulukon *kerroin* yhteydessä funktio palauttaa satunnaisen alkion taulukosta *kertoimet* ja taulukon *mestari* yhteydessä palautetaan satunnainen alkio taulukosta *mestarit.*

Parametri voi olla myös merkkijonotyyppinen. Tehtävässä määritellään merkkijonoparametri *mjono,* jonka sisällä voidaan käyttää aiemmin määriteltyjä parametreja. Tämä tehdään toiminnolla *use*, kuten edelläkin. Tässä tehtävässä muuttujan *mjono* arvoksi asetetaan kysymys: *<use_name=”kysymys”/>*. Parametri *mjono* on asetettu readonly-tyyppiseksi (*readonly=”true”*), joten sitä ei voida tehtävän instanssissa muuttaa.

```
<problemstatement>
  Tämä piti asettaa instancessa: <use name="pakko"/>
    <use name="mestari"/> on muuten paras!
    Katso video:
      <video>tacoma.flv</video>
    Mikä näistä on <use name="kysymys"/>:n logo?
</problemstatement>
```

Varsinainen tehtävänanto alkaa kohdasta <*problemstatement*>. Tehtävässä on määritelty sisällöksi video (*tacoma.flv*) sekä tekstiä. Lisäksi aiemmin pakolliseksi määriteltyä parametria (*pakko*) on käytetty tässä (<*use name=”pakko”*>).

```
<choices showincorrect="2">
  <choice name="teht1_mjono" value="-1">
    <show type="text">
      <use name="mjono"/>
    </show>
    <response>Logo ei ole merkkijono.</response>
  </choice>
  <choice name="teht1_kuva" value="1">
    <show type="image">
      <figure><use name="kysymys"/>.jpg</figure>
    </show>
      <response>Oikein!</response>
  </choice>
  <choice name="teht1_kaava" value="0">
    <show type="expression">
       <expression>
         <use name="g"/> * <use name="kerroin"/>
       </expression>
     </show>
       <response>Mitähän mahdoit ajatella...</response>
  </choice>
</choices>
</problem>
```

Kun tehtävänanto on määritelty, annetaan tehtävään liittyvät vastausvaihdot. Niitä näytetään oikeiden lisäksi tehtävänannon yhteydessä kerrottu määrä, joka tässä tehtävässä on 2 (*choices showincorrect=”2”*). Jokaiselle vastausvaihtoehdolle annetaan nimi (*name*), arvo (*value*), vastauksen tyyppi (*type*) sekä reaktio vastaukseen (*response).* Vastauksen nimen tulee olla yksikäsitteinen, sillä siirtymät määritellään tehtävägraafia piirrettäessä nimen avulla. Siirtymät määritellään vasta, kun tehtävästä luodaan instanssi graafiin.

Vastausvaihtoehto voi olla tyypiltään *text, image* tai *expression*, kuten tässä olevat ovat. Jos vastausvaihto on tyyppiä *image*, voidaan sen nimessä käyttää (*use*) parametreja, joten vastaus voi vaihtua kysymystä vastaavaksi. Viimeisessä vastausvaihtoehdossa, lausekkeessa, on käytetty muuttujaa *g*, joka määriteltiin aiemmin ja se on kerrottu myös aiemmin määritellyllä muuttujalla *kerroin*.

Ilmentymä – *instance* ja reitit – *route*

Elementin *instance* avulla samaa tehtävää voi käyttää moneen kertaan samassa sokkelossa. Elementille nimetään tehtävä, josta ilmentymä luodaan, ja suunnat (*route*), joihin sen vastausvaihtoehdoista voi jatkaa. Myös ilmentymälle tulee määritellä yksi-

käsitteinen nimi. Ilmentymää luotaessa voidaan tiputtaa vastausvaihtoehtoja pois jättämällä vastausvaihtoehtoa vastaava elementti *route* määrittelemättä.

Ilmentymään on myös mahdollisuus määritellä paikallisia parametreja elementillä *set.* Tällä kerrotaan, minkä parametrin arvo halutaan korvata ja sen uusi arvo. Tehtävää määriteltäessä, yllä, voidaan vaikuttaa siihen, mitä parametreja ilmentymässä voidaan korvata.

```
<instance name="teht1_i1" problem="teht1">
  <set param="pakko">Jokin arvo</set>
  <set param="kerroin">
    <function name="array_rand"/>(<use name="kertoimet"/>) - 2
  </set>
  <set param="mestarit">
    <item>Michael Schumacher</item>
    <item>Mika Häkkinen</item>
  </set>
    <route choice="teht1_mjono" forward="teht1_i2"/>
    <route choice="teht1_kuva" forward="teht1_i2"/>
    <route choice="teht1_kaava" forward="teht1_i2"/>
</instance>
<instance name="teht1_i2" problem="teht1">
  <set param="pakko">Jokin toinen arvo</set>
    <route choice="teht1_mjono" left="teht1_i1" right="poks"/>
    <route choice="teht1_kuva" left="teht1_i1" right="onnistui"/>
    <route choice="teht1_kaava" left="teht1_i1" right="poks"/>
</instance>
```

Instansseille annetaan siis yksikäsitteinen nimi (*name*) ja tehtävä (*problem*), johon luotu instanssi liittyy. Samasta tehtävästä voidaan luoda useampi instanssi, niin kuin yllä olevassa esimerkissä on tehty. Eri instanssit erotetaan toisistaan niiden nimien perusteella, ensimmäisen instanssin nimeksi on annettu *teht1_i1* ja toiselle *teht1_i2.*

Instanssien yhteydessä täytyy määritellä aiemmin pakollisiksi asetetut parametrit. Yllä olevassa esimerkissä parametri *pakko* on aiemmin määritelty pakolliseksi, joten se määritellään nyt arvolla *Jokin arvo*. Tehtävässä määritellyn paikallisen taulukon voi instanssin määrittelyn yhteydessä määritellä uusiksi. Tässä tehtävässä taulukkoon *mestarit* on määritelty uusi sisältö. Parametrin *set* sisällä voidaan käyttää elementtejä *function* ja *use.* Tässä tehtävässä parametrin *kerroin* sisällä on käytetty funktiota *array_rand* ja taulukkoa *kertoimet*.

Instanssiin määritellään reitit (*route*). Suuntavaihtoja on viisi, eteenpäin, oikealle, vasemmalle, ylös tai alas. Yllä olevassa esimerkissä ensimmäisestä instanssista on määritelty kolme kulkusuuntaa, jotka riippuvat valitusta vastausvaihtoehdosta. Tässä esimerkissä vastausvaihtoehdosta *teht1_mjono* liikutaan eteenpäin tehtävään *teht1_i2.* Vastausvaihtoehdosta *teht1_kuva* siirrytään samoin eteenpäin tehtävään *teht1_i2*, kuten myös viimeisestä vastausvaihtoehdosta *teht1_kaava*.

Toinen instanssi määritellään samalla kuin ensimmäinen. Aiemmin pakolliseksi määritelty parametri (*pakko*) tulee tässäkin instanssissa määritellä. Nyt sen arvoksi

asetetaan *Jokin toinen arvo*. Reitit määritellään samoin avainsanalla *route*. Jokaiseen vastausvaihtoehtoon liitetään kaksi kulkusuuntaa, vasen (*left*) ja oikea (*right*). Vasen kulkusuunta johtaa kaikissa vaihtoehdoissa tehtävään *teht1_i1*. Oikealle menevä kulkusuunta johtaa ensimmäisessä vastausvaihtoehdossa (*teht1_mjono*) epäonnistuneeseen suoritukseen (*poks*), samoin kuin viimeisessä (*teht1_kaava*). Toinen vastausvaihtoehto johtaa onnistuneeseen suoritukseen (*onnistui*).

Kokonaisuudessaan tässä työssä käytetty DTD löytyy liitteestä 1. Kattavammin DTD:tä yleensä kuvaa [31].

5 QT

Trolltech esitteli Qt:n ensimmäisen kerran toukokuussa 2005, mutta kehitystyö oli alkanut jo vuonna 1991. Nykyään Qt:ta kehittää Nokia, joka osti Trolltechin vuonna 2008. Vuonna 2009 julkaistu versio 4.5 sisälsi ensimmäistä kertaa myös SDK:n. Tällä hetkellä viimeisin julkaistu versio, 4.7, sisältää noin 500 luokkaa ja yli 9000 funktiota.

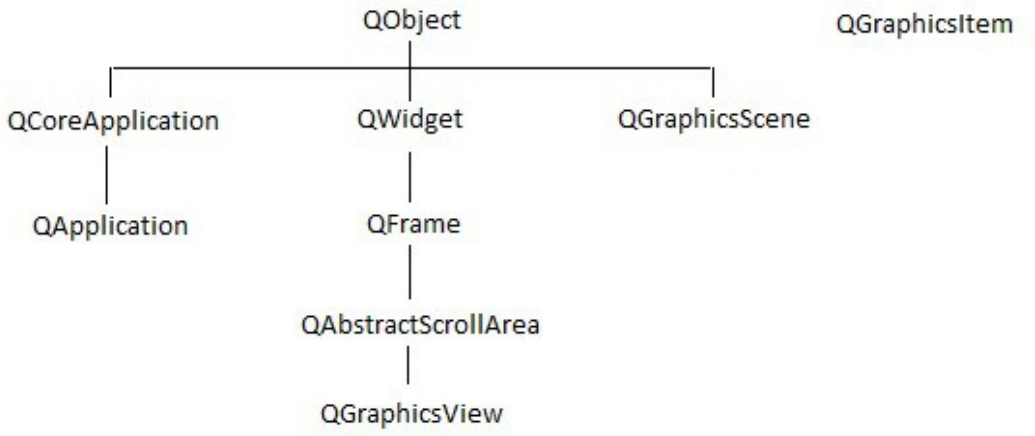
5.1 Qt:n luokkahierarkia

Qt on olio-orientoitunut. Se näkyy erityisesti sen luokkahierarkiassa, joka pohjautuu hyvin vahvasti periytymiseen. Periytymisen ansioista ohjelmoinnin työmäärä vähenee, sillä aiemmin tehtyjä luokkia voidaan käyttää hyödyksi vain tarpeelliset komponentit lisäämällä.

Qt-kirjaston ylimpänä rakenteena on QObject, josta siis periytetään lähes tulkoon kaikki muut luokat. Tämä koskee käyttöliittymävimpainten (widget, näistä lisää alla) lisäksi myös Qt-sovellusten suoritusta hallinnoivaa luokkaa QApplication, sekä sen kantaluokkaa QCoreApplication. Kolmantena esimerkkinä ovat luokat QGraphicsScene, QGraphicsView ja QGraphicsItem. Huomionarvoista on, että QGraphicsItem ei peri luokkaa QObject, sillä QGraphicsItem-olioiden ajatellaan olevan toiminnallisuudeltaan suhteellisen rajoittuneita, joten kaiken QObject-luokan toiminnallisuuden periyttäminen luokkaan QGraphicsItem ei ole tarkoituksenmukaista. [11]

Qt:ssa kaikkien vimpainten kantaluokkana toimii QWidget (Kuva 2). QWidget ei ole abstrakti luokka, vaan sitä voidaan käyttää myös säiliönä muille vimpaimille. Luokkahierarkian suunnittelun perusperiaatteena on ollut luokkien omistussuhteiden esittäminen luonnollisella tavalla. [11]

Luokka QCoreApplication tarjoaa tapahtumankäsittelysilmukan komentorivipohjaisille Qt-sovelluksille. QCoreApplication-olioita tulee olla tasan yksi, ja se käsittelee kaikki käyttöjärjestelmätasolta tulleet pyynnöt. Graafisella käyttöliittymällä varustetuille taas tulee käyttää luokasta QCoreApplication periytettyä luokkaa QApplication (Kuva 2).

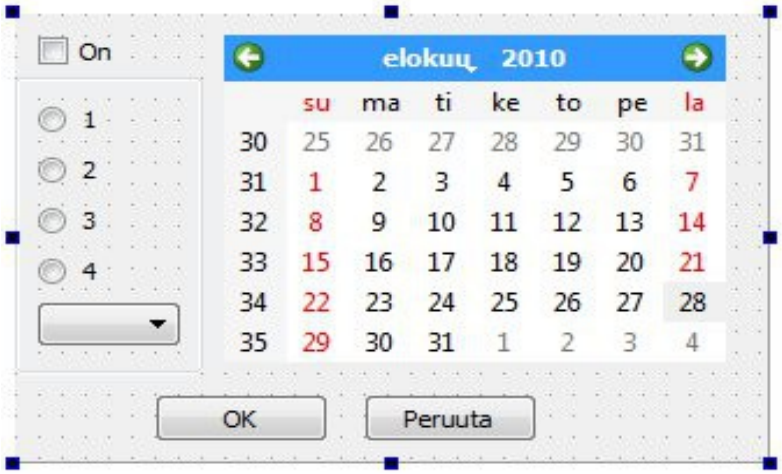


Kuva 2: Ote Qt:n luokkahierakista. Mukaillen [11]

5.2 Qt:n käyttöliittymäkirjasto

Käyttöliittymän peruselementteinä toimivat vimpaimet (widget). Ne ovat graafisen käyttöliittymän elementtejä, joiden avulla käyttäjä pystyy kommunikoimaan ohjelman kanssa. Esimerkiksi painikkeet tai erilaiset tekstilaatikot ovat vimpaimia. Qt:n käyttöliittymäkirjasto tarjoaa laajan valikoiman valmiita vimpaimia, mutta ohjelmoija voi luoda niitä myös itse.

Vimpaimet voidaan sijoittaa monella eri tavalla. Ne voivat toimia yksin itsenäisenä komponenttina tai ne voidaan asetella usean eri vimpaimen ryhmään. Ryhmässä olevien vimpainten ei tarvitse olla samanlaisia, esimerkiksi painike ja tekstilaatikko voivat kuulua samaan vimpainryhmään. Usein vimpaimet sijoitetaan ikkunaan. Alla olevassa esimerkissä (Kuva 3) neljä valintanappia muodostaa vimpainryhmänsä ja oikealla oleva kalenteri on oma itsenäinen vimpaimensa. Vimpaimet on sijoitettu ikkunaan (QWidget).

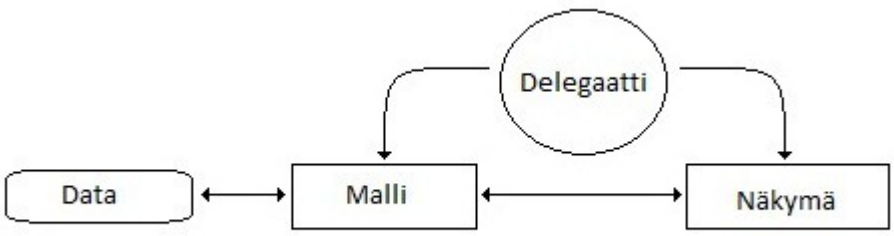


Kuva 3: Erilaisia vimpaimia sijoitettuna ikkunaan

5.3 MVC

Malli-näkymä-ohjain -arkkitehtuuri (model-view-controller, MVC) on alunperin ohjelmointikielessä Smalltalk [33] käytetty suunnittelumalli, jota käytetään käyttöliittymäsuunnitteluun yleisesti muuallakin [34]. Sen kolme peruskäsitettä ovat malli, joka toimii rajapintana dataan, näkymä, joka esittää tiedon käyttöliittymässä, ja ohjain, joka määrittelee, miten käyttöliittymä reagoi käyttäjän syötteisiin.

Qt käyttää tästä hieman poikkeavaa malli/näkymä (Model/View) -arkkitehtuuria, jossa ohjainta ei ole (Kuva 4). Tässä mallin ja näkymän käyttötarkoitus eroaa hieman MVC-arkkitehtuurin vastaavista, sillä malli/näkymä-arkkitehtuurissa näkymällä on MVC-mallin kontrollerille kuuluvia tehtäviä, kuten tapahtumankäsittely. Näiden lisäksi malli/näkymä -arkkitehtuurissa on delegaatti (delegate), joka toimii mallin ja näkymän välissä kuljettaen dataa. Se määrittelee yksittäisille mallin elementeille, miten ne

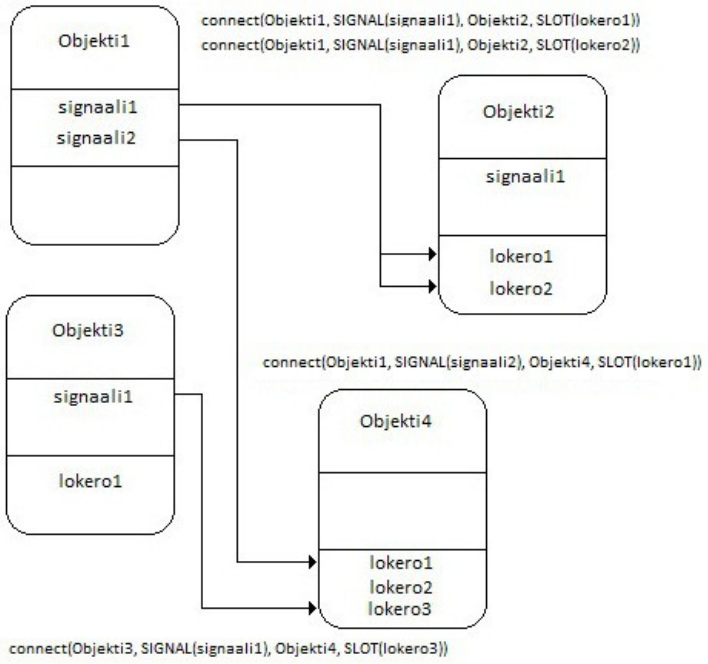


Kuva 4: Qt:n malli/näkymä-arkkitehtuuri. Mukaillen [11].

esitetään käyttöliittymässä. Delegaatti huolehtii myös yksittäisen itemin editoinnista. Yleensä käytetään oletusdelegaattia, mutta ohjelmoija voi myös toteuttaa oman delegaatin periyttämällä sen luokasta QAbstractItemDelegate [11].

Malli/näkymä -arkkitehtuurissa mallit tarjoavat näkymille ja delegaateille rajapinnan sisältöönsä. Tietoalkioiden käsittely on määritelty luokassa QAbstractItemModel, josta kaikki malli/näkymä -arkkitehtuurin mallit periytyvät. Näkymien kantaluokkana toimii QAbstractItemView. Qt:ssa on toteutettu valmiita näkymiä, kuten QListView, mutta ohjelmoija voi luokkaa QAbstractItemView periyttämällä tehdä myös oman näkymän.

5.4 Tapahtumankäsittely



Kuva 5: Qt:n signaali-lokero-mekanismi. Mukaillen [15].

Mallit, näkymät ja delegaatit kommunikoivat keskenään käyttäen signaali-lokero -mekanismia (signal-slot mechanism). Tämä on perinteistä callback- tekniikkaa [35] kehittyneempi ja tyyppiturvallisempi tapa toteuttaa objektien välinen kommunikointi. Menetelmässä tapahtumasta ilmoittava objekti lähettää signaalin, jonka seurauksena kutsutaan vastaanotettavan objektin signaaliin kytkettyä tapahtumankäsittelyfunktiota [11]. Esimerkiksi kuvassa 5 on objekteja liitettynä toisiinsa signaali-lokero-mekanismilla. Kuvan Objekti1:n signaali1 on liitetty Objekti2:n lokero1:een

käyttämällä funktiota connect(). Se ottaa parametreinaan signaalin lähettävän olion ja signaalin nimen SIGNAL-makron avulla sekä olion, jolle signaali lähetetään sekä lokeron nimen SLOT-makron avulla.

Signaali-lokero-mekanismi tarvitsee toimiakseen moc-esikääntäjän (meta object compiler), josta lisää alla. Sen tehtävänä on kääntää makron Q_OBJECT sisältävät luokat C++-kielelle. Moc-kääntäjän tuottama koodi linkitetään ja käännetään ohjelman implementoinnin yhteydessä varsinaiseen ohjelmakoodiin [15].

Lokerofunktiot (slot functions) toteutetaan ohjelmakoodissa täysin samoin kuin tavalliset jäsenfunktiot, sillä erolla, että ne on määritelty ohjelmakoodin otsikkotiedossa näkyvyysmääreen slots alle. Signaalit määritellään näkyvyysmääreen signals alle. Nämä molemmat voidaan laittaa joko julkiseen (public*),* suojattuun (protected*)* tai yksityiseen (private*)* rajapintaan. Tyyppiturvallisuutta taataan sillä, että määritellyn signaalin ja siihen liittyvän lokerofunktion parametrien tyyppien tulee aina täsmätä. Parametrien voi olla ylimääräisiä, sillä jos lähettävällä signaalilla on enemmän parametreja kuin lokerofunktiolla, ylimääräiset jätetään yksinkertaisesti käyttämättä. Lisäksi luokkien tulee periytyä joko suoraan tai epäsuorasti luokasta QObject, sillä Qt:n metaobjekti-järjestelmä käyttää luokassa QObject määriteltyjä rajapintoja, joiden toteutukset aliluokkaan luodaan makrolla Q_OBJECT.[15]

Huomionarvoista on, että lokerot ovat myös ihan tavallisia jäsenfunktioita eikä tällainen tiedä, mitä signaaleja siihen on kytketty. Samoin objekti ei tiedä, onko sen lähettämät signaalit kytketty johonkin lokeroon. Tämä mahdollistaa toisistaan riippumattomien komponenttien luomisen Qt:lla, sillä lokerot ja signaalit eivät ole toisistaan riippuvaisia ja samaan lokeroon voidaan kytkeä useampi signaali ja toisinpäin [15].

5.5 Meta-objekti-järjestelmä

Qt.n meta-objekti-järjestelmä perustuu:

- luokkaan QObject, joka tarjoaa kantaluokan niille luokille, jotka hyödyntävät järjestelmää,
- makroon Q_OBJECT, jota käytetään luokan sisällä ilmaisemaan meta-objektin käyttötarve, ja
- meta-objekti-kääntäjään (Meta Object complier, moc), joka tuottaa tarvittavan kooditiedoston.

Moc-kääntäjä lukee C++-lähdekooditiedostot. Jos se havaitsee luokan, jossa on makro Q_OBJECT, se tuottaa toisen C++-lähdekooditiedoston, jossa on toteutus makrolle. Tämä uusi lähdekooditiedosto on joko sisällytetty alkuperäiseen luokkaan (#include) tai se käännetään ja linkitetään luokan implementoinnin yhteydessä. [15]

Moc-kääntäjällä tuotettu lähdekooditiedosto sisältää muitakin ominaisuuksia kuin tapahtumankäsittelyn mahdollistamisen. Metakooditiedosto sisältää toiminnallisuutta

erilaisten funktioiden muodossa, ja esimerkiksi dynaamiset tyyppimuunnokset voidaan tehdä luokan QObject omalla funktiolla. Tämän etuna on, että se ei vaadi ajonaikaista tyyppi-informaatiota (run-time type information, RTTI) ja tyyppimuunnoksen voi tehdä yli kirjastorajojen. [15]

Luokasta QObject voidaan periyttää luokkia, jotka eivät käytä makroa Q_OBJECT. Tämä ei kuitenkaan ole suositeltavaa, sillä meta-objekti-järjestelmä tulkitsee tällaista luokkaa kuten sen lähintä esivanhempaa, joka sisältää metakoodia. Tällöin esimerkiksi funktio QMetaObject::className() ei palautaan kutsuvan luokan nimeä vaan sen esivanhemman nimen. Mikäli luokka moniperiytetään, Q_OBJECT tulkitsee ensimmäisen perittävän luokan olevan periytetty luokasta QObject. [15]

5.6 QT:n grafiikkanäkymäkehys

Grafiikkanäkymäkehys on Qt:n osa, joka tarjoaa alustan erilaisille kaksiulotteisille objekteille (items). Lisäksi siihen sisältyy näkymä (view) objektien näyttämiseen, suurentamiseen sekä pyörittämiseen. Sovelluskehys mahdollistaa näyttämöllä (scene) olevien objektien tapahtumanvälitysarkkitehtuurin, joihin kuuluu muun muassa kursorin tilan muuttumisen havaitseminen [36]. Yksinkertaistettuna voidaankin sanoa, että näyttämö toimii säiliönä erilaisille objekteille, joita katsotaan näkymän läpi. Grafiikkanäkymäkehyksen tapahtumanvälitysarkkitehtuurissa erilaiset näyttämölle tulevat tapahtumat välitetään suoraan siinä kohdassa olevalle objektille. Esimerkiksi hiiren näppäimen painallus objektin päällä voidaan välittää käsiteltäväksi kyseiselle objektille. Näyttämölle on toteutettu kuminauhavalitsin (rubberband selection), jolloin näyttämöltä voidaan valita hiiren avulla halutut objektit. Valittujen objektien käsittelyyn voidaan käyttää grafiikkanäkymäkehyksen omien luokkien metodeja. [36]

Grafiikkanäkymäkehys on toteutettu malli/näkymä -arkkitehtuuria käyttäen. Yhtä näyttämöä kohti voi olla useita näkymiä, ja näyttämöllä voi olla samanaikaisesti useita objekteja. [36]

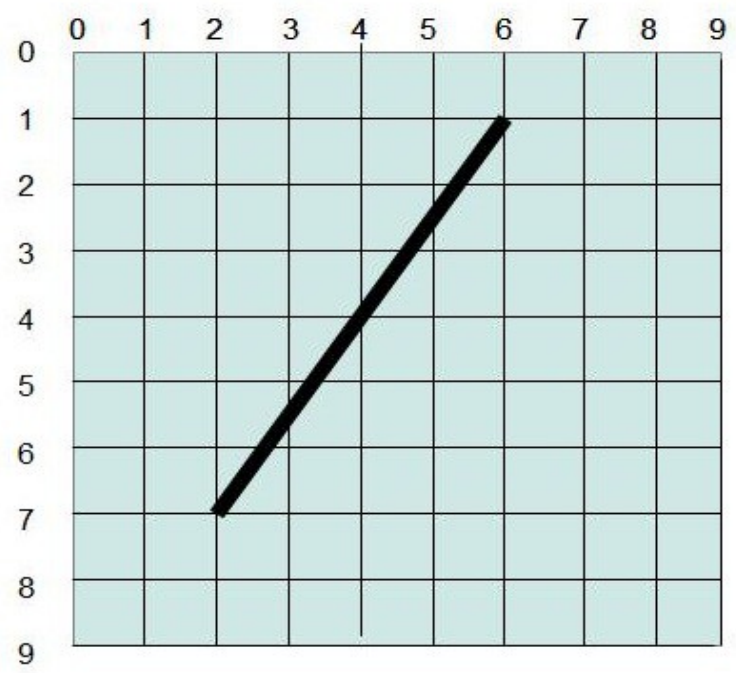
QGraphicsScene on näyttämö, jonka vastuulla grafiikkanäkymäkehyksessä ovat muun muassa eri objektien välinen kanssakäyminen sekä objektien tilat. Näyttämö toimii säiliönä objekteille. Niitä lisää näyttämölle funktio addItem(), joka joko lisää tai siirtää objektin lapsineen kutsuvalle näyttämölle, mikäli se on jo olemassa toisella näyttämöllä. Samalla omistusoikeus siirtyy uudelle näyttämölle.

QGraphicsView toteuttaa näkymän, joka mahdollistaa objektien näyttämisen. Samaan näyttämöön voi olla useita erilaisia näkymiä. Näkymä vastaanottaa näppäimistöltä ja hiireltä tulevat tapahtumat ja muuntaa ne näyttämötapahtumiksi, jotka se välittää näyttämölle. Koska näkymän ja näyttämön koordinaattijärjestelmät ovat erilaisia, näkymässä on olemassa myös keino muuntaa näyttämön koordinaatteja. Muunnos tehdään, kun näyttämötapahtumat välitetään näkymälle [36]. Koordinaatit tietotyyppinä esitellään tarkemmin kohdassa 5.7.

Kaikkien näyttämölle piirrettävien objektien kantaluokka on QGraphicsItem. Siihen on valmiiksi toteutettu yleisimpiä objekteja, kuten ympyrä ja neliö, mutta siitä voi periyttää myös oman objektin. Luokalle QGraphicsItem voidaan kirjoittaa myös erilaisia tapahtumankäsittelijöitä, esimerkiksi drag-and-dropin [36] toteuttamista varten.

5.7 Koordinaatit

Qt:n grafiikkanäkymäkehyksen koordinaattien käsittely perustuu karteesiseen koordinaattijärjestelmään. Objektin sijainti ja geometria esitetään kahdella koordinaattiparilla siten, että ensin ilmoitetaan objektin alkupisteen koordinaatit muodossa (x,y), jonka jälkeen ilmoitetaan loppupisteen koordinaatit samoin. Niinpä QLine(2,7,6,1) on suora, jonka alkupiste on (2,7) ja loppupiste(6,1). [37]



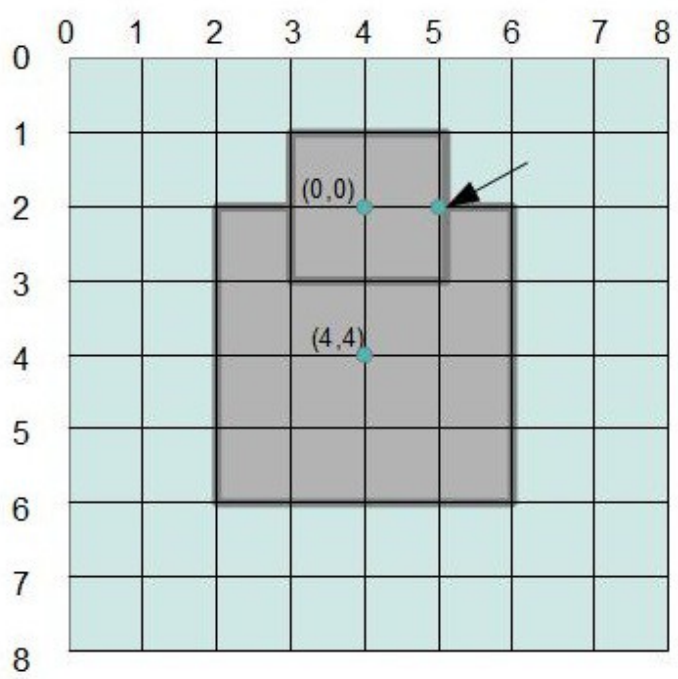
Kuva 6: QLine(2,7,1,6). Mukaillen [11].

Jokaisella grafiikkanäkymäkehyksen osalla, objektilla, näyttämöllä ja näkymällä, on omat koordinaatit, joiden käsittelyyn Qt tarjoaa erilaisia funktiota. Samat funktiot käyvät niin objektien, näyttämön tai näkymän koordinaattien muuntoon ja käsittelyyn. [37]

Jokaisella objektilla on omat, paikalliset koordinaattinsa. Ne ilmoittavat yleensä objektin keskipisteen. Muunneltujen objektien yhteydessä ainoastaan objektin paikallisista koordinaateista tulee huolehtia, sillä näyttämö ja näkymä muuntavat objektin koordinaatit omien koordinaattijärjestelmiensä mukaisiksi. Lapsi esitetään vanhemman

koordinaatteihin suhteutettuna siten, että jos lapsen keskipiste on kohdassa (10,0), sen koordinaatti (0,10) vastaa vanhemman koordinaattia (10,10). [36]

Alla oleva esimerkki selventää koordinaattijärjestelmää. Kuvaan on piirretty kaksi neliötä, joiden keskipisteet on esitetty niiden omissa koordinaateissa. Tällöin isomman neliön keskipiste on kohdassa (4,4) ja pienemmän neliön keskipiste on suuremman neliön koordinaatistossa (4,2). Nuolella osoitettu piste on pienemmän neliön koordinaatistossa piste (1,0) ja suuremman neliön koordinaatistossa (1,-2).



Kuva 7: Esimerkki koordinaattijärjestelmistä. Mukaillen [10].

Näyttämön oma koordinaattijärjestelmä tietää jokaisen siinä olevan objektin paikan, sekä mahdollistaa näyttämötapahtumien välittämisen objekteille. Objektin omien paikallisten koordinaattien lisäksi jokaisella objektilla on myös näyttämökoordinaatit. [36]

Näkymän koordinaattijärjestelmän erikoisuus on se, että se on suhteessa siihen alustaan, johon se on toteutettu. Vasen yläkulma on aina (0,0) ja oikea alakulma suurin mahdollinen koordinaatti kumpaankin suuntaan. Kaikki hiireltä tulevat tapahtumat sekä drag-and-drop -tapahtumat esitetään näkymäkoordinaatteina. [36]

Grafiikkanäkymäkehys tukee yleisimpiä muokkaamista helpottavia ominaisuuksia, kuten objektien pyörittämistä ja suurentamista. Lisäksi erilaiset kursorit ja vihjelaatikot on helppo toteuttaa. Näyttämön pystyy sellaisenaan tulostamaan ja siten tallentamaan käyttäjän toivomassa kuvaformaatissa.

Animaatiot pystyy toteuttamaan käyttämällä sovelluskehystä Animation. Näihin tarvittavat, yleensä mukaellut objektit periytetään suoraan luokasta QGraphicsObject.

Näyttämölle pystyy sulauttamaan mitä tahansa objekteja, olivat ne sitten yksinkertaisia kuten painikkeet (QPushButton) tai hyvinkin monimutkaisia (QTabWidget). Jopa pääikkunan (MainWindow) sulauttaminen on mahdollista.

Grafiikkanäkymäkehys ei ole mitenkään kevyt sovelluskehys. Siksi laitteistolla oletetaan olevan tarjota riittävän nopeita liukulukuoperaatioita, jotta objektien muunnokset sujuisivat riittävän nopeasti. [36]

5.8 Graafinen vimpain sekä asettelut (layout)

Luokka QGraphicsWidget toteutettiin Qt:n versiossa 4.4. Sen perustoiminnallisuus on suurin piirtein sama kuin luokan QWidget. QGraphicsWidget lisää alkuperäiseen luokkaan QWidget tuen muun muassa tapahtumankäsittelylle ja kirjasimille sekä ulkoasulle. Osa luokan QWidget funktioista on jätetty tarpeettomina pois. Yhteenvetona voisikin todeta, että luokka QGraphicsWidget on grafiikkanäkymäkehyksen yleiskäyttöinen vimpain. [36]

Luokka QGraphicsLayout on toteutettu erityisesti grafiikkanäkymäkehyksen tarpeisiin. Luokkien QGraphicsLayout ja QLayout suhde on samanlainen kuin yllä olevien luokkien QWidget ja QGraphicsWidget. Myös luokkaa QGraphicsLayout pystyy periyttämään tarpeen mukaisesti. [36]

Voidaankin todeta, että grafiikkanäkymäkehykseen on toteutettu funktiota ja luokkia, jotka erikoistavat jo olemassa olevia Qt:n luokkia. Lisäämällä näihin toimintoja ja jättämällä pois tarpeettomia grafiikan käsittelyn näkökulmasta, saadaan toteutettua tehokkaita työkaluja grafiikan käsittelyyn.

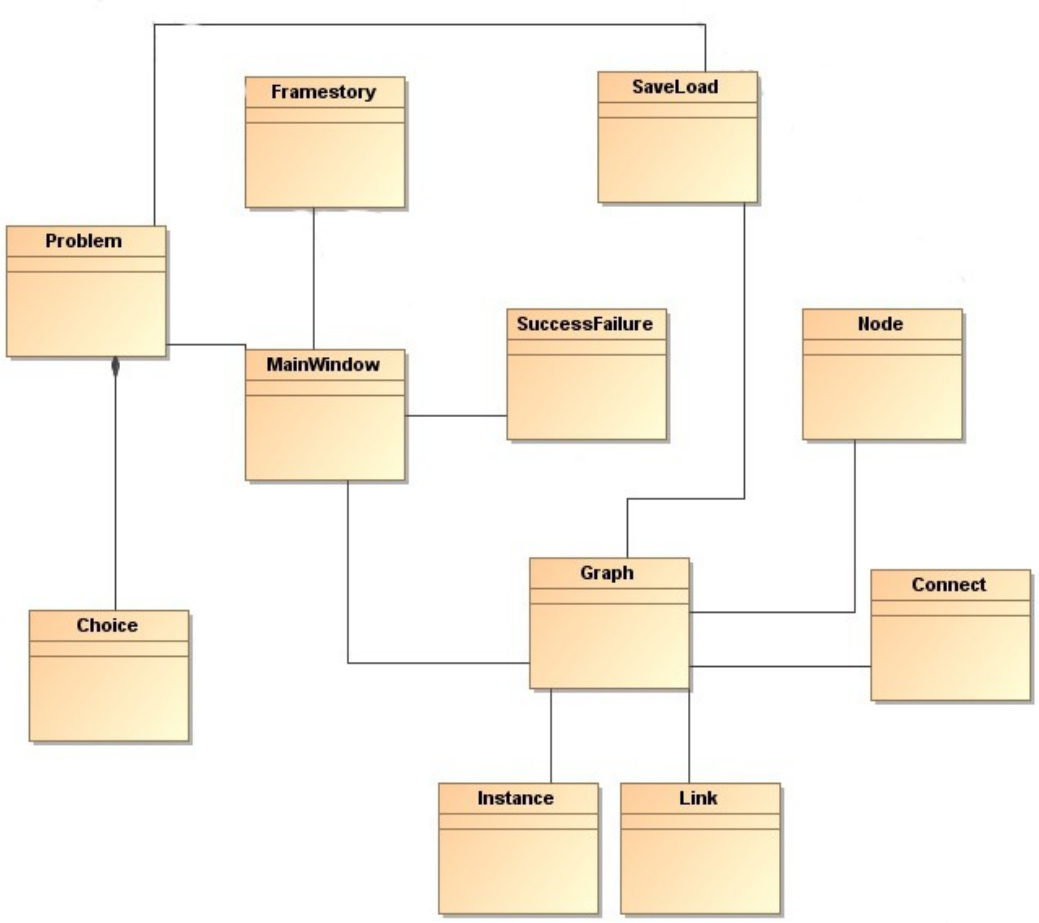
6 JÄRJESTELMÄN KUVAUS

Peligraafin piirtämiseen tarkoitettu ohjelma on suunniteltu käytettäväksi siten, että käyttäjä tuo kaikki haluamansa tehtävät ohjelmaan, piirtää sen jälkeen graafin ja tallentaa sen. Valmiista pelistä muodostuu .xml-tiedosto, jonka käyttäjä pakkaa .zip-pakettiin [38] yhdessä pelin muun datan (kuvat ja videot) kanssa. Jokainen tehtävä tallennetaan omaan .xml-tiedostoonsa, jolloin kaikki syötetyt tehtävät ovat käyttövalmiina myös toista käyttökertaa varten.

Käyttäjä syöttää tehtävät joko käyttäen ohjelman omaa toiminnallisuutta, tai kirjoittamalla tehtävän omaksi .xml-tiedostoksi. Ohjelman kannalta ei ole merkitystä, onko tiedosto kirjoitettu käsin vai ohjelman omaa toiminnallisuutta hyödyntämällä, kunhan tiedoston rakenne on oikea, DTD:n mukainen. Tehtävien välillä ei saa olla nimiristiriitoja, vaikka niillä olisikin eri aihepiiri.

Tehtävien syöttämisen jälkeen käyttäjä syöttää haluamansa kehystarinan sekä alku- ja loppusolmut, jonka jälkeen käyttäjä voi piirtää pelin graafin. Graafi kuvaa pelin siirtymät ja liittää tehtävät toisiinsa. Peli alkaa aina kehyskertomuksesta ja päättyy jompaankumpaan loppusolmuun.

Kun käyttäjä on saanut pelin valmiiksi, pelin data tallennetaan .zip-paketiin [38]. Tämän paketin käyttäjä lataa Moodleen luomalla kurssille sokkelo-aktiviteetin, joka asentaa pelin ja asettaa sen toimintakuntoon.

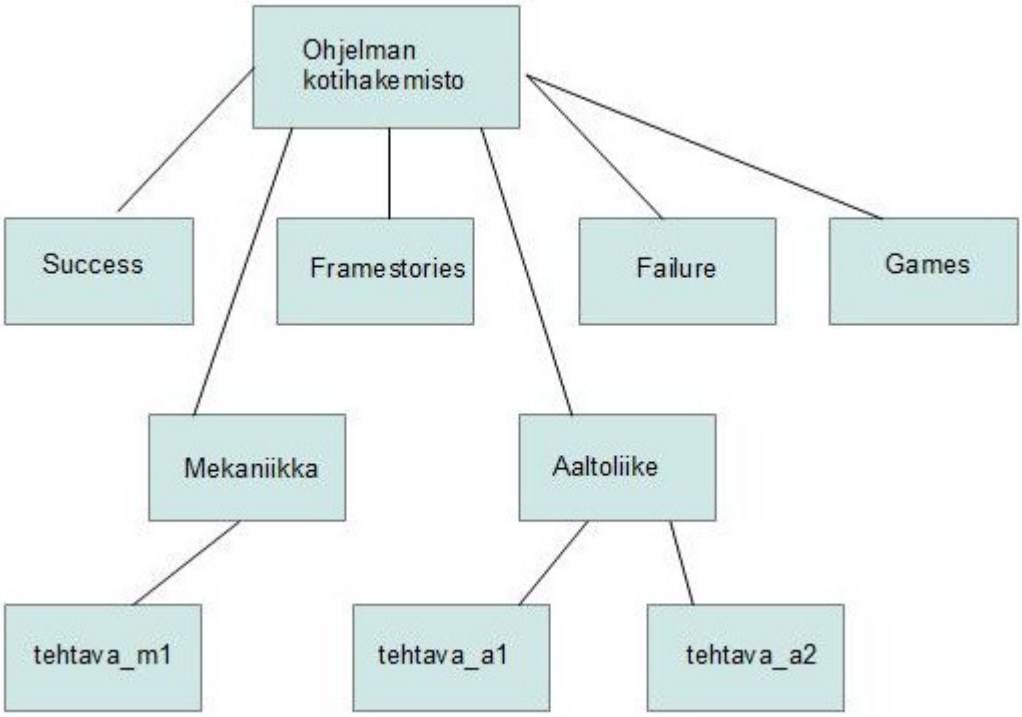


Kuva 8: Ohjelman rakenne UML-kaaviona

6.1 Toteutustekniset ratkaisut

Kuvassa 8 on kuvattu ohjelman rakenne UML-kaaviona. Ohjelman suunnittelussa pyrittiin yksinkertaisuuteen ja laajennettavuuteen. Sovelluksen eri ikkunat (uuden tehtävän syöttö, kehystarinan syöttö, vastausvaihtoehtojen syöttö sekä graafin piirto) toteutettiin jokainen omana luokkanaan. Toteutuksen alkuvaihteessa tämä vaikutti yksinkertaisimmalta ratkaisulta, joka olisi myös myöhemmin helposti ylläpidettävissä. Tehtävien syöttö haluttiin eriyttää graafin piirtämisestä, joten tehtävien syöttöön käytetään luokkia *Choice* ja *Problem* ja graafin piirrossa luokkia *Node* sekä *Link*. Graafin piirtämisen jälkeen sen tallentamisen yhteydessä jokaisesta piirtoalustan *Node*-oliosta luodaan *Instance*-olio, joka kaarineen tallennetaan XML-tiedostoon. Tallennukseen ja myöhemmin mahdollisesti toteutettavat lataustoimintoon liittyvät toiminnallisuudet on sijoitettu luokkaan *SaveLoad.* Kooditason toteutuksessa sekä dokumentoinnissa sovellettiin TTY:n Ohjelmistotekniikan laitoksen tyyliopasta [39].

Tehtävät tallennetaan aihepiireittäin kansioihin. Tästä muodostuu yksinkertainen tietokanta (Kuva 9). Tehtävien aihepiirin saa käyttäjä vapaasti määritellä. Selkeyden vuoksi aihepiiri kannattaa määritellä riittävän tarkasti. Aihepiirien, eli samalla kansioiden, määrää ei ole rajoitettu, samoin tehtäviä voi olla yhdessä aihepiirissä mielivaltainen määrä. Myös kehystarinoista (*framestories)* sekä loppusolmuista (*success* ja *failure*) on omat kansiot. Jokainen tehtävä tallennetaan omaan .xml-tiedostoon, johon tallentuu myös tehtävään liittyvät vastausvaihtoehdot.



Kuva 9: Ohjelman kansiorakenne

Jokaisella tehtävällä, alku- ja loppusolmulla kuten myös kehystarinalla tulee olla yksikäsitteinen nimi. Tällä vältetään mahdolliset konfliktit instansseja luotaessa.

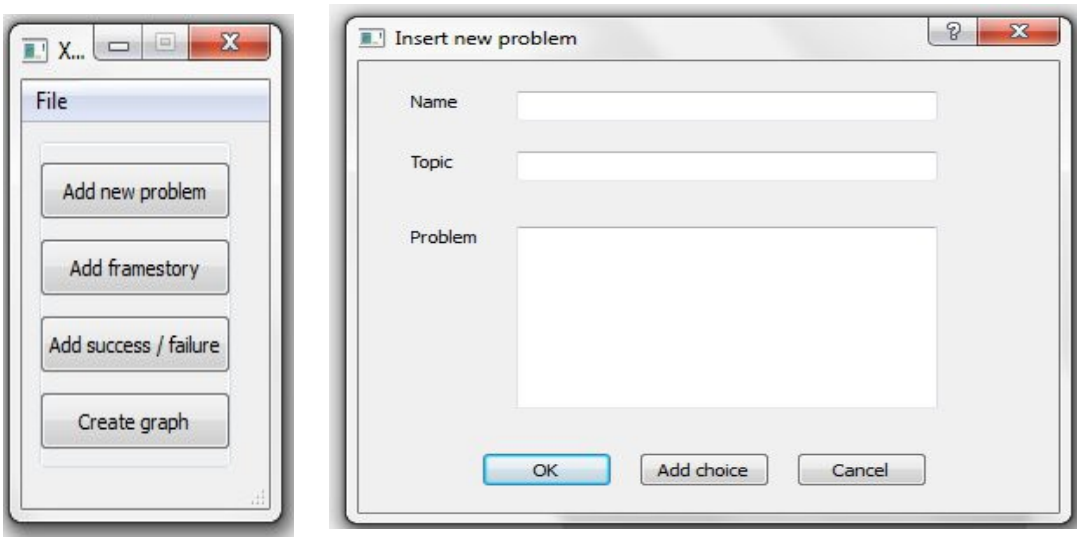
Koko sovelluksen pohjana toimii QApplication-olio, joka luodaan heti pääohjelmassa. Tämän jälkeen luodaan luokan QMainWindow olio MainWindow. Ohjelman suoritus siirretään Qt:n hallintaan pääohjelman viimeisessä lauseessa. Käytännössä tämä tarkoittaa siirtymistä silmukkaan, jossa odotetaan tapahtumia käyttöliittymästä.

```
int main( int argc, char *argv[])
{
    QApplication app(argc, argv);

    MainWindow *mainWin = new MainWindow();
    mainWin->resize(600,600);

    mainWin->show();
    return app.exec();
}
```

Muiden kuin pääikkunan alustana on luokka QDialog. Ikkunat ovat tilattomia (*non-modal*), eli pääikkunan valikoita pystyy valitsemaan, vaikka dialogi-ikkuna olisikin avoinna. Käyttäjä voi siis käyttää päävalikon toimintoja, vaikka esimerkiksi graafinpiirto-ikkuna olisi auki. Graafinpiirtoalustaksi on valittu QGraphicsScene sen monipuolisuuden ja laajennettavuuden takia. Näyttämöltä on helppo ottaa talteen solmujen sijaintipaikat ja tallentaa ne. Tätä ominaisuutta voidaan käyttää hyödyksi, jos jo kerran luotua graafia halutaan täydentää tai laajentaa ja piirretty graafi halutaan avata samanlaiseksi kuin mitä se tallennushetkellä oli.

6.2 Käyttöliittymä

Ohjelman kaikki käyttöliittymät ja niihin liittyvät vimpaimet on tehty Qt Creatorin [22] graafisella editorilla, jolla on helppo asetella eri komponentit käyttöliittymään. Qt Creator luo oman käyttöliittymäluokan, joka otetaan käyttöön kutsumalla rakentajassa käyttöliittymäluokan metodia setupUi(). Käyttöliittymäkomponentit voidaan luoda myös ilman graafista editoria, jolloin elementit toteuttava koodi kirjoitetaan suoraan toteutustiedostoon (.cc). Käyttöliittymäelementit liitetään tapahtumankäsittelijöihin funktiolla connect() joko koodissa tai käyttämällä Qt Creatorin graafista editoria.



Kuva 10: (a) Aloitusikkuna, (b) tehtävän syöttöikkuna

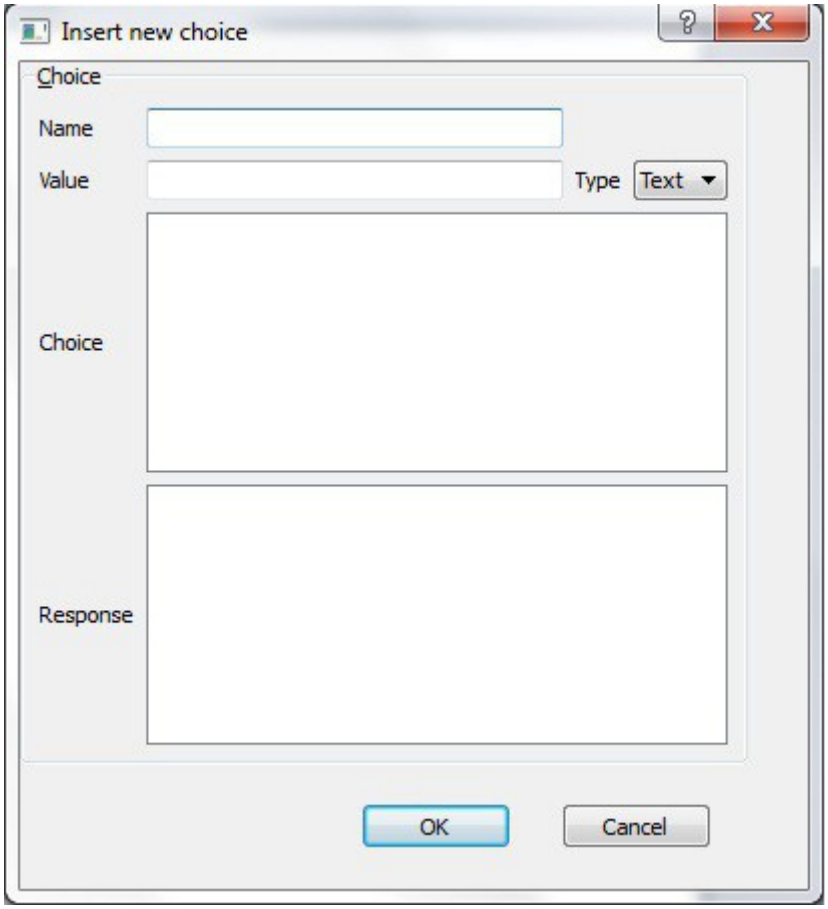
Ohjelma avautuu pääikkunaan (Kuva 10a), josta käyttäjä valitsee joko vaihtoehdon *Add new problem*, jos haluaa syöttää uuden tehtävän, tai *Create graph,* jos halutut tehtävät on jo syötetty. Graafissa tarvittavat onnistuneen ja epäonnistuneen suorituksen loppusolmut saa syötettyä valitsemalla *Add success/failure*. Peliin liittyvän taustatarinan voi kirjoittaa valitsemalla painikkeen *Add framestory*. Ohjelman saa suljettua valitsemalla ylävalikosta *File* → *Exit*. Uuden tehtävän syöttö järjestelmään voidaan tehdä ohjelman kautta, mutta mikään ei estä käyttämästä myös muuten luotuja XML-tiedostoja. Niiden täytyy tietysti vastata DTD:tä, jotta niitä voidaan myöhemmin käyttää graafissa eikä niissä saa määritellä samannimisiä elementtejä.

Kun käyttäjä haluaa syöttää uuden tehtävän, avautuu uuden tehtävän syöttöikkuna (Kuva 10b). Käyttäjä antaa uuden tehtävän nimen, aihepiirin sekä varsinaisen tehtävänannon, jotka tallennetaan DTD:n mukaisesti XML-tiedostoonsa. Kenttään *problem* käyttäjä syöttää varsinaisen tehtävänannon. Jokainen tehtävä tallennetaan omaan tiedostoon. Ikkunan voi sulkea painamalla painiketta *Cancel* ikkunan alalaidassa, jolloin ei tietoja tallenneta. Valitsemalla *OK* uusi tehtävä tallennetaan. Kaikki tiedot vaaditaan ennen tallentamista. Jos käyttäjä ei syötä uuden tehtävän nimeä, aihetta tai itse tehtävänantoa, antaa ohjelma virheilmoituksen, eikä tallentamista tehdä.

Uuteen tehtävään annetaan vastausvaihtoehdot painamalla painiketta *Add choice* sivun alalaidasta. Tämä avaa uuden ikkunan (Kuva 11). Jos uutta vastausvaihtoehtoa ei haluta antaa, voi ikkunan sulkea painamalla *Cancel*. Tällöin tehtävään ei tallennu uutta vastausvaihtoehtoa.

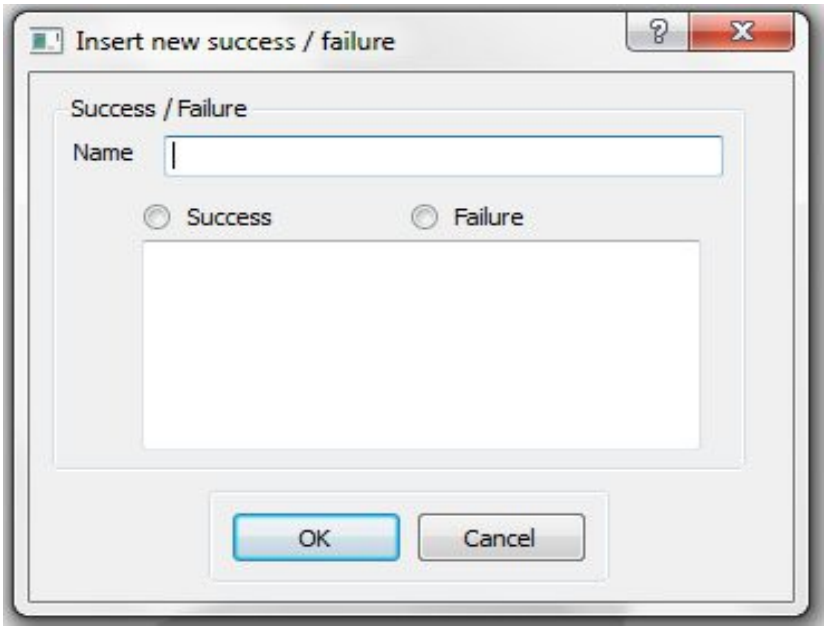
Käyttäjä syöttää vastausvaihtoehdon nimen kenttään *name*, arvon kenttään *value* ja valitsee tyypin alasvetovalikosta *type*. Vastausvaihtoehto kirjoitetaan kenttään *Choice* ja sen vaste kenttään *Response*. Vaste ei teoriassa ole pakollinen tieto, mutta pelaajan kannalta on miellyttävämpää, jos oikeasta ja väärästä vastauksesta saa spesifin vasteen. Erityisesti matemaattisia ongelmia sisältävän pelin pelaaminen on miellyttävämpää, jos vastauksesta tulee selkeä vaste. Lisäksi vasteen käyttöä tukevat myös luvussa 2 esitetyt tutkimukset. Vastausvaihtoehdon voi tallentaa valitsemalla *OK*, jolloin ikkuna sulkeutuu ja palataan takaisin uuden tehtävän syöttöikkunaan (Kuva 10b).

Tehtävät tallentuvat aihepiirteittäin siten, että jokainen aihepiiri sijaitsee omassa kansiossaan. Tällä pyritään siihen, että käyttäjän olisi helpompi löytää haluamansa tehtävä. Vastausvaihtoehtojen määrää ei ole rajoitettu. Vastausvaihtoehdon yhteyteen liitetään graafin piirtämisen yhteydessä suunta, jonne vastausvaihtoehdoista edetään. Kulkusuuntia on mahdollista esittää enintään viisi: ylös, alas, oikealle, vasemmalle ja eteenpäin.



Kuva 11: Vastausvaihtoehdon syöttöikkuna

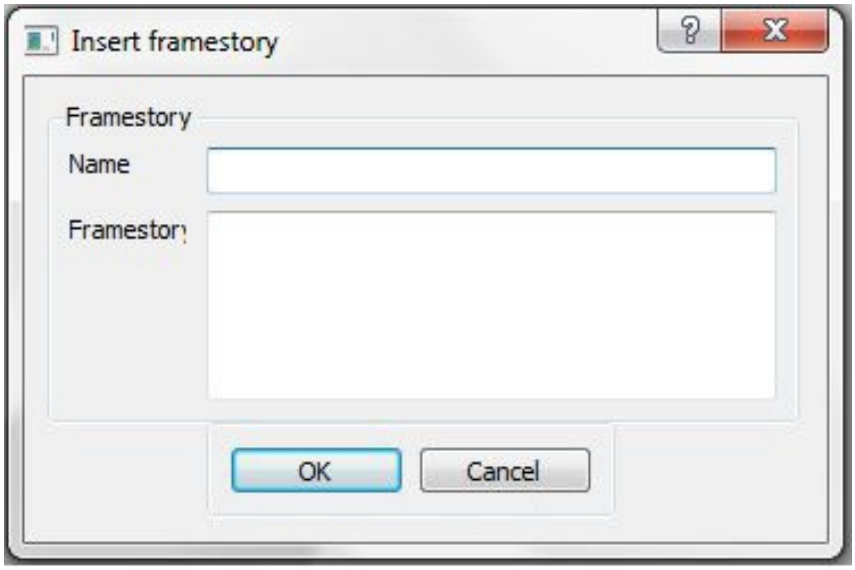
Vastausvaihtoehto voi olla tyypiltään joko teksti, kuva tai lauseke. Vastausvaihtoehdon tyyppi valitaan alasvetovalikosta kohdasta *type* (Kuva 11). Myös *response* voi sisältää tekstiä, kuvia tai lausekkeita. Nimettyjen kuvien ja videoiden on selvyyden vuoksi sijaittava samassa hakemistossa kuin itse tehtävä.. Tällöin kaikki halutut tehtävien osaset tulevat varmasti mukaan aikanaan laadittavaan .zip-pakettiin. Eri vastausvaihtoehdoille voi antaa pisteitä; oikeista positiivisia ja väärästä negatiivisia tai nollan. Pisteiden määritys tapahtuu yksinkertaisesti laittamalla vastausvaihtoehdon kohtaan *value* haluttu numeerinen arvo. Tyhjä arvo tarkoittaa nollaa. Loppusolmu (Kuva 12) eroaa tavallisesta tehtävästä siten, että siitä ei ole enää siirtymiä eteenpäin. Käyttäjä valitsee loppusolmulle kuvaavan nimen ja kirjoittaa sen kenttään *name* ja valitsee valintanappuloista haluamansa vaihtoehdon *(success* tai *failure*). Kirjoitettuaan loppusolmun tekstin tekstikenttään, käyttäjä valitsee painikkeen *OK*, jos haluaa tallentaa uuden loppusolmun ja *Cancel*, mikäli ei halua tallentaa. Kaikki tyypin *Success* loppusolmut tallennetaan kansioon *Success* ja tyypin *Failure* kansioon *Failure*.



Kuva 12: Uuden loppusolmun syöttöikkuna

Uuden kehystarinan syöttö (Kuva 13) ei juurikaan eroa uuden loppusolmun tai uuden tehtävän syöttämisestä. Käyttäjä kirjoittaa haluamansa kehystarinan ja valitsee *OK*, jos haluaa tallentaa sen, tai *Cancel*, jolloin uutta kehystarinaa ei tallenneta. Kaikki kehystarinat tallennetaan kansioon *Framestories*. Kehystarinan nimeä käytetään myös tallennettavan tiedoston nimenä. Ohjelma tuottaa aina sellaisen kehystarinan, josta pääsee eteen-

päin ensimmäiseen tehtävän instanssiin, josta peli varsinaisesti alkaa. Kehystarinan ja instanssin yhteys määritellään graafin piirtämisen yhteydessä.

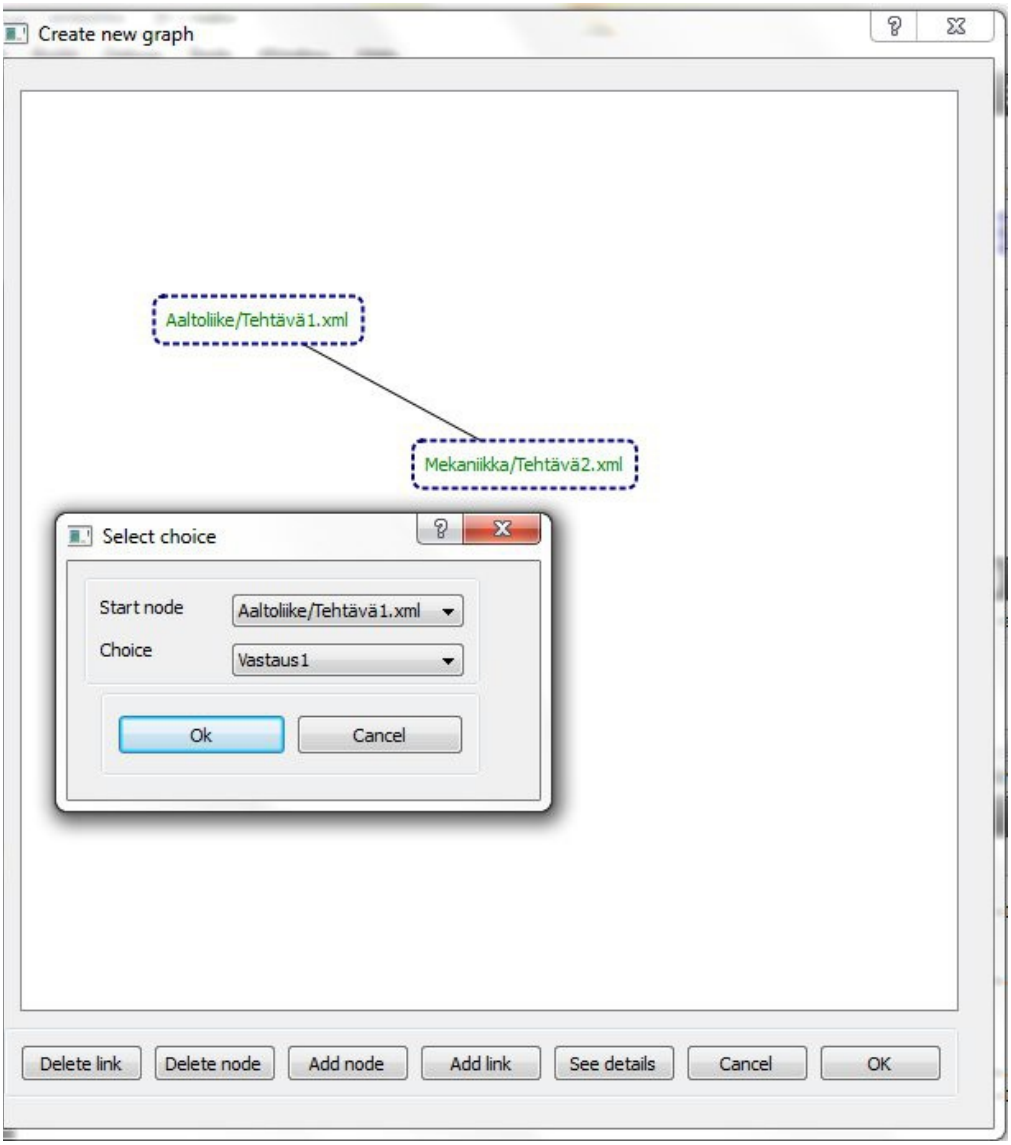


Kuva 13: Uuden kehystarinan syöttöikkuna

Kun käyttäjä on syöttänyt haluamansa tehtävät (joko samalla tai aiemmalla käyttökerralla), voi hän siirtyä piirtämään itse graafia. Se piirretään piirtoalustan avulla (Kuva 14).

Käyttäjä lisää tehtävän piirtoalustalle painamalla painiketta *Add node.* Tämän jälkeen avautuu dialogi, jossa käyttäjä voi tiedostovalikosta valita haluamansa tehtävän. Oletuksena tiedostovalikko avautuu käyttäjän kotihakemistoon.

Tehtävän valinta vahvistetaan painamalla *OK*-painiketta. Solmulle annetaan automaattisesti nimi, joka on muotoa aihepiiri / tehtävän nimi. Jos käyttäjä ei halua lisätä solmua, valitsemalla *Cancel* piirtoalustalle ei lisätä mitään.



Kuva 14: Solmujen yhdistäminen

Solmun saa poistettua siten, että valitsee sen ja painaa sen jälkeen painiketta *Delete Node*. Samalla solmun kaaret poistetaan.Kahden tehtävän välille saa yhteyden valitsemalla ne molemmat (joko kuminauhatoiminnolla tai *ctrl*-näppäimen kanssa klikkaamalla molempia solmuja) ja painamalla sen jälkeen painiketta *Add link*. Tämä avaa dialogin, jossa kysytään siirtymän suunta ja vastausvaihtoehto, joka siirtymään liittyy (Kuva 14). Samansuuntaisia siirtymiä voi solmussa olla vain yksi, joten ohjelma tulostaa virheilmoituksen, mikäli jo olemassa olevaa siirtymää yritetään tehdä uudestaan. Yhteyden saa poistettua siten, että valitsee sen, ja painaa sen jälkeen painiketta *Delete Link*. Yhteyden päässä olevat solmut pysyvät ennallaan.

Jos käyttäjä haluaa katsoa lisätietoja jostain tehtävästä, valitsee hän kyseisen solmun ja painaa painiketta *See details.* Tämän jälkeen avautuu ikkuna, jossa esitetään tehtävän nimi, aihepiiri, tehtävänanto ja käytössä olevat vastausvaihtoehdot. Lisätietoikkunan saa suljettua painamalla painiketta *OK*.

Kun käyttäjä on mielestään valmis sokkelon kanssa, hän valitsee painikkeen *OK* graafin luontinäytön alaosasta. Sovellus kysyy pelitiedoston nimen ja luo tälle nimelle XML-tiedoston, joka kuvaa yhtä peliä. Kaikki pelit tallennetaan kansioon *Games*.

7 ARVIOINTI

Tässä työssä oli tavoitteena saada aikaan piirto-ohjelma, jonka avulla pystyy laatimaan sokkelopeliin vaadittavan XML-dokumentin ilman, että käyttäjän tarvitsee kirjoittaa XML:ää. Piirto-ohjelmalle asetettiin erilaisia tavoitteita niin toteutuksen kuin käyttäjän näkökulmasta.

7.1 Tavoitteiden täyttyminen

Piirto-ohjelmalla pystyy toteuttamaan yksinkertaisen sokkelopelin ja tuottamaan peliin vaadittavan XML-kuvauksen, mutta kaikkia ominaisuuksia, joita piirto-ohjelmaan haluttiin, ei pystytty toteuttamaan. Erilaisia haasteita tuli kehittämisen yhteydessä eteen useita.

Sovellusalustaksi valittiin Qt, joka on edelleen suhteellisen nopeasti kehittyvä ympäristö. Alustan nopea kehitys saattaa aiheuttaa kehittäjälle vauhtisokeutta ja halua käyttää uusimpia ominaisuuksia. Tämä ei välttämättä ole aina tarkoituksenmukaista tai viisasta. Mikäli sovellusalustaksi valitaan vielä kehittyvä alusta, kannattaa pitäytyä siinä versiossa, jossa alun perin ohjelmaa alettiin kehittää. Myöhemmät versiot voivat tuoda lisää toiminnallisuutta, mutta toteutuksen kannalta on parempi tehdä ohjelma yhdellä versiolla valmiiksi ja vasta myöhemmin päivittää ohjelmisto uutta versiota vastaavaksi. Näin ohjelma toimii varmasti toteutusalustan tietyssä versiossa.

Työn alussa sovellukselle tärkeiksi todettujen kriteerien toteutumista on tarkasteltu taulukossa 1. Taulukon vasemmassa sarakkeessa on kriteeri, ja jokaista kriteeriä seuraa kolme vaihtoehtoa (+, 0 tai -), jotka ilmaisevat, saavutettiinko tavoite. Ensimmäinen vaihtoehto tarkoittaa että kriteeri toteutuu, keskimmäinen sitä, ettei kriteeriä voida vielä arvioida ja viimeinen, ettei kriteeri toteudu.

Taulukko 1: Kriteerien toteutuminen

Kriteeri	+ / 0 / -
helppokäyttöinen	-
helposti opittavissa	-
alustariippumaton	+
käyttöönottokynnykseltään matala	0

Sovelluksen toteutusalustaksi valittu Qt toteuttaa itsessään jo alustariippumattomuuden kriteerin. Sovellus toimii siten niin Windows-, MacIntosh- kuin Unix-ympäristöissä. Kriteereistä kaksi, helppokäyttöisyys ja helppo opittavuus jäävät saavuttamatta, sillä tällä hetkellä XML-tiedoston muokkaus vaatii XML-koodin kirjoittamista tekstieditorilla. Käyttöönottokynnyksen mataluutta ei tällä hetkellä voida arvioida, sillä sovellusta ei ole käyttäjätestattu, eikä käyttöönottokokemuksia ole. Sokkelopelin laatijan tulisi myös pystyä kattavasti rakentamaan haluttu peli sovelluksen avulla. Tavoitteeseen päästiin siinä mielessä, että piirto-ohjelman käyttäjän ei tarvitse huolehtia pelin teknisestä toteutuksesta ja merkkauskielen syntaksista muualla kuin syötettävien tekstien sisällä, mutta edelleen käyttäjän vastuulla on pelimaailman luominen ja tehtävien sekä niihin liittyvien vastausvaihtoehtojen syöttö. Nämä kuitenkin tallennetaan, joten kerran syötetty tehtävä on käytössä myös muissa peleissä.

7.2 Jatkokehitysajatukset

Tekoprosessin aikana heräsi useita jatkokehitysajatuksia. Näistä osa haluttiin toteuttaa samanaikaisesti ohjelman peruskehityksen kanssa. Tämä ei kuitenkaan osoittanut järkeväksi ajatukseksi, sillä tämä viivästytti ohjelman toteutusta entisestään.

Yksi kehitysidea on kehittää graafin piirtoa (Kuva 14, sivu 37) siten, että tehtävät voisi suoraan drag-and-drop -periaatetta käyttämällä liittää piirtoalustalle. Tällöin piirtoalustan yhteyteen tulisi liittää esimerkiksi hakemistopuu, josta tehtävät voitaisiin vetää.

Toinen kehitysidea liittyy peleihin: Nyt valmista sokkelopelin XML-tiedostoa ei pysty muokkaamaan piirto-ohjelman avulla, vaan muokkaus täytyisi tehdä suoraan tiedostoon. Jatkokehitysajatuksena onkin valmiin pelin avaaminen ja muokkaaminen. Esimerkiksi peliin voisi lisätä tehtäviä, kehystarinaa tai reittejä.

Tällä hetkellä graafi on kertakäyttöinen, eikä sitä voida laajentaa tai täydentää, vaan graafi täytyy piirtää uusiksi. Jatkokehitysajatuksena onkin graafin tallennusmahdollisuus sekä graafin muokkaaminen ja täydentäminen. Pelimaailman toteuttavat graafit voivat olla laajoja, ja mikäli tällainen tallennusmahdollisuus olisi, graafin voisi piirtää osissa valmiiksi. Samoin graafin osaa voisi käyttää hyödyksi toisessa graafissa.

Eräs kehitysidea liittyy tehtäviin ja niistä muodostuvaan tietokantaan. Nykyinen, kansiorakenteella toteutettu tietokanta ei ole riittävä, mikäli tehtäviä syötetään ohjelmaan useita kymmeniä, jopa satoja. Tällöin myös jonkinlainen hakutoiminto erilaisista tehtävistä olisi hyvä toteuttaa. Tämä kehitysidea vaatii kehittäjältä tietokanta-osaamista ja esimerkiksi tietokannan fyysinen sijoittaminen tulee pohtia.

Mikäli sovellusta aiotaan kehittää, yksi kehitysidea on tukea sovelluksen kieliversiointeja. Nyt sovellus on englanninkielinen, mutta se on suhteellisen helposti käännettävissä muille kielille Qt:n omilla työkaluilla.

8 YHTEENVETO

Tässä työssä oli tarkoituksena tukea pelien mahdollisuuksia opetuskäytössä toteuttamalla piirto-ohjelma. Tällä pystyttäisiin laatimaan sokkelopeliin vaadittava XML-dokumentti ilman, että käyttäjän tarvitsisi kirjoittaa XML:ää lainkaan. Sovellukselle asetettiin tavoitteita, joista osa jäi saavuttamatta.

Oppimisessa tärkein asia on motivaatio. Jos motivaatio loppuu, loppuu myös oppiminen. Motivaatiota voidaan kasvattaa ja pitää korkealla esimerkiksi opetuspelin avulla. Tutkimukset osoittavat, että opetuspelin avulla voidaan antaa käyttäjälle onnistumisen ja oppimisen elämyksiä sekä sitä kautta kasvattaa motivaatiota ja tukea oppimista. Tutkimukset osoittavat, että opetuspeliä voidaan käyttää opetustapahtuman tukena ja pelin avulla käyttäjä saattaa helpommin soveltaa oppimaansa kokemukseen, kun opittavaa asiaa pystyy kokeilemaan virtuaalimaailmassa. Opetuspeleissä on usein käytössä päätös-käytös-palaute -sykli. Vasta pelin jälkeinen palautekeskustelu muokkaa pelikokemukset oppimistapahtumiksi. Pelin jälkeen ja aikana saatu palaute saattaa parhaassa tapauksessa kannustaa käyttäjää entistä parempaan suoritukseen, mutta päinvastaisiakin havaintoja on. Jos käyttäjä kokee itse onnistuneensa hyvin, negatiivinen palaute huonosta suoriutumistasosta saattaa laskea motivaatiota ja tehdä oppimisesta haastavaa.

Sovelluksen toteutusalustaksi valittiin Qt sen kattavuuden ja laajan dokumentaation vuoksi. Työssä hyödynnettiin Qt:n valmiita funktiota mahdollisuuksien mukaan ja toteutettava sovellus toteuttaa Qt:n MV-mallia. Piirto-ohjelman käyttöliittymät toteutettiin Qt:n omalla kehitystyökalulla, Qt Creatorilla.

Sovellukselle asetetuista kriteereistä osa jäi saavuttamatta. Helppokäyttöisyyden ja helpon opittavuuden kriteerit jäivät saavuttamatta, sillä käyttäjän tulee edelleen kirjoittaa osa vaadituista tiedoista XML-kielellä. Käyttöönottokynnyksen mataluudesta ei voida vielä sanoa. Sovellus on alustariippumaton, sillä valittu toteutusalusta on alustariippumaton.

Sovellus ei kuitenkaan ole tällaisenaan riittävän kattava, sillä osa toiminnallisuudesta jätettiin toteuttamatta. Kehitystyön aikana tuli esiin erilaisia haasteita, joiden ratkaiseminen vaikutti väistämättä myös ohjelman kehitystyöhön. Sovelluksen avulla käyttäjä voi syöttää sokkelopeliin liittyvät tehtävät ja niiden vastausvaihtoehdot, piirtää sokkelopeliin liittyvän graafin sekä viimeistellä sen. Käyttäjän vastuulle jää tehtävien ja niihin liittyvien vastausvaihtoehtojen syöttö sekä graafin piirto.

Piirto-ohjelma vaatii jatkokehitystä. Sen jälkeen olisi hyvä kerätä käyttäjäpalautetta ja muokata ohjelmaa sen mukaisesti. Tällä hetkellä käyttäjäpalautetta ei kannata kerätä, sillä ohjelma on vielä sen verran keskeneräinen.

LÄHTEET

[1] Scratch [WWW]. [Viitattu 23.5.2011]. Saatavissa http://scratch.mit.edu/

[2] Mayo, M.J. Games for Science and Engineering Education. Communications of the ACM, Vol 50, 2007, Issue 7, s. 31–35

[3] Moodle.org: open-source community-based tools for learning [WWW]. [Viitattu 1.3.2011]. Saatavissa http://moodle.org/

[4] Nikula, V. XML-kuvaus ja toteutusalusta WWW-opetuspeleille. Diplomityö (TTY) 2007. 53s.

[5] Extensible Markup Language (XML) 1.0 (Fifth Edition) [WWW]. [Viitattu 4.2. 2010]. Saatavissa http://www.w3.org/TR/2008/REC-xml-20081126/

[6] Tekstiseikkailu Zork [WWW]. [Viitattu 12.6.2011]. Saatavissa http://fi. wikipedia.org/wiki/Zork

[7] Kelly, S. & Tolvanen, J.-P. Domain-specific modeling. Enabling .....full code generation. Hoboken NJ, USA, John Wiley & Sons, 2008. 427s.

[8] Bentley, J. Programming pearls: little languages. Communications of the ACM, Vol. 29, 1986, Issue 8, s. 711–721

[9] Generic Eclipse Modeling System [WWW]. [Viitattu 26.11.2010]. Saatavissa http://www.eclipse.org/gmt/gems/about.php

[10] Domain-Specific Development with Visual Studio DSL Tools [WWW] .[Viitattu 26.11.2010]. Saatavissa http://www.domainspecificdevelopment.com/

[11] Qt – A cross-platform application and UI framework [WWW]. [Viitattu 7.6. 2010]. Saatavissa http://qt.nokia.com/

[12] Microsoft Visual Studio [WWW] .[Viitattu 26.11.2010]. Saatavissa http://www. microsoft.com/visualstudio/fi-fi/

[13] Codename “Oslo” Repository Becomes SQL Server Modeling Services [WWW] [Viitattu 26.11.2010]. Saatavissa http://blogs.msdn.com/b/keith_short/archive/ 2009/11/23/codename-oslo-repository-becomes-sql-server-modeling-services.aspx

[14] Eclipse Graphic Modeling Framework [WWW]. [Viitattu 26.11.2010]. Saatavissa http://www.eclipse.org/articles/Article-Introducing-GMF/article.html

[15] Qt 4.6 Reference Documentation [WWW]. [Viitattu 4.2.2010] Saatavissa http:// doc.trolltech.com/4.6/

[16] Windows-kotisivu [WWW]. [Viitattu 18.5.2011. Saatavissa http://windows. microsoft.com/fi-fi/windows/home

[17] The Unix System [WWW]. [Viitattu 18.5.2011]. Saatavissa http://www.unix.org/

[18] Apple [WWW]. [Viitattu 18.5.2011]. Saatavissa http://www.apple.com/fi/

[19] Connolly, T. Database Systems 2nd. edition. Boston, Pearson Education Limited 1995, 2005, 1093s.

[20] OpenGL – The Industry of Standard for High Performance Graphics [WWW]. [Viitattu 30.6.2010]. Saatavissa http://www.opengl.org/

[21] Blanchette, J. & Summerfield, M. C++ GUI Programming with Qt 4, 2.painos, Massachusetts, Prentice Hall 2008, 752s.

[22] Qt Creator [WWW]. [Viitattu 26.11.2010]. Saatavissa http://qt.nokia.com/products/developer-tools/

[23] Gee, J. P.What Video Games Have to Teach Us About Learning and Literacy. ACM Computers in Entertainment, Vol. 1, No.1, October 2003

[24] Amory, A., Naicker, K., Vincent, J., Adams, C. The use of computer games as an educational tool: identification of appropriate game types and game elements. British Journal of Educational Technology, Vol.30, No.4, 1999, s. 311–321.

[25] Garris, R., Ahlers, R., Driskell, J.E. Games, motivation and learning: A research and practice model. Simulation and Gaming, Vol. 33, No. 4, December 2002, s.441–467

[26] World Web Consortium [WWW]. [Viitattu 4.1.2011]. Saatavissa http://www.w3.org/

[27] Cover Pages: Standard Generalized Markup Language (SGML) [WWW]. [Viitattu 4.1.2011]. Saatavissa http://xml.coverpages.org//sgml.html

[28] CSS Tutorial [WWW]. [Viitattu 12.6.2011]. Saatavissa http://www.w3schools.com/css/

[29] HTML tutorial [WWW]. [Viitattu [9.3.2011]. Saatavissa http://www.w3schools.com/html/default.asp

[30] XHTML Tutorial [WWW]. [Viitattu 12.6.2011]. Saatavissa http://www.w3schools.com/xhtml/

[31] DTD Tutorial [WWW]. [Viitattu 4.2.2010]. Saatavissa http://www.w3schools.com/dtd/

[32] What is URL? [WWW]. [Viitattu 12.6.2011]. Saatavissa http://www.webopedia.com/TERM/U/URL.html

[33] Smalltalk [WWW]. [Viitattu 23.5.2011]. Saatavissa http://www.smalltalk.org/main/

[34] Reenskaug, T. The original MVC reports. [WWW]. [Viitattu 5.5.2010]. Saatavissa http://home.ifi.uio.no/trygver/2007/MVC_Originals.pdf

[35] C++ Callback Solution [WWW]. [Viitattu 18.5.2011]. Saatavissa http://partow.net/programming/templatecallback/index.html

[36] Qt 4.6: The Graphics View Framework [WWW]. [Viitattu 17.7.2010]. Saatavissa http://doc.qt.nokia.com/4.6/graphicsview.html

[37] Qt 4.6: The Coordinate System [WWW]. [Viitattu 22.8.2010]. Saatavissa http://doc.qt.nokia.com/4.6/coordsys.html

[38] Info-ZIP [WWW]. [Viitattu 19.5.2011]. Saatavissa http://www.info-zip.org/Zip.html

[39] Tampereen teknillisen yliopiston ohjelmistotekniikan laitoksen tyyliopas [WWW]. Viitattu [4.3.2010]. Saatavissa http://www.cs.tut.fi/~oliot/kirja/tyyliopas/

LIITE 1 : DTD-SKEEMA

Tässä liitteessä on XML-kuvauksen sanaston määrittelevä DTD-skeema. Se määrittelee elementit ja niiden sallitun sisällön ja attribuutit. DTD-skeema on alunperin laadittu tämän diplomityön pohjana olleessa diplomityössä [4], jonka valmistumisen jälkeen DTD:tä jatkokehitettiin.

```
<!ELEMENT    problems      (settings?, param*, framestory, success, failure,
                            problem+, instance+) >
<!ATTLIST    problems      xmlns          CDATA      #IMPLIED
                           timelimit      NMTOKEN    #IMPLIED>

<!ELEMENT    settings      EMPTY>
<!ATTLIST    settings      logo           NMTOKEN    #IMPLIED
                           correctimg     NMTOKEN    #IMPLIED
                           wrongimg       NMTOKEN    #IMPLIED>

<!ELEMENT    param         (#PCDATA | function | use | item)*>
<!ATTLIST    param         name           ID         #REQUIRED
                           description    CDATA      #IMPLIED>

<!ELEMENT    item          (#PCDATA)*>

<!ELEMENT    framestory    (#PCDATA | figure | video | use) *>
<!ATTLIST    framestory    left           IDREF      #IMPLIED
                           forward        IDREF      #IMPLIED
                           right          IDREF      #IMPLIED
                           up             IDREF      #IMPLIED
                           down           IDREF      #IMPLIED>

<!ELEMENT    success       (#PCDATA | figure | video | use) *>
<!ATTLIST    success

<!ELEMENT    failure       (#PCDATA | figure | video | use)*>
<!ATTLIST    failure       name           ID         #REQUIRED>

<!ELEMENT    function      EMPTY>
<!ATTLIST    function      name           NMTOKEN    #REQUIRED>

<!ELEMENT    use           EMPTY>
<!ATTLIST    use           name           IDREF      #REQUIRED>

<!ELEMENT    problem       (param*, problemstatement, choices)>
<!ATTLIST    problem       name           ID         #REQUIRED
                           topic          NMTOKEN    #REQUIRED
                           key            CDATA      #IMPLIED
                           requiredkey    CDATA      #IMPLIED
```

```
<!ELEMENT    choices       (choice*)>
<!ATTLIST    choices       showincorrect  NMTOKEN    #REQUIRED>

<!ELEMENT    choice        (show, response) >
<!ATTLIST    choice        name           ID         #REQUIRED
                           value          NMTOKEN    #REQUIRED
                           breakskey      CDATA      #IMPLIED>

<!ELEMENT    show          (#PCDATA | param | expression | figure | function | use)*>
<!ATTLIST    show          type           (expression | image | text)      #REQUIRED>

<!ELEMENT    expression    (#PCDATA | function | use)*>

<!ELEMENT    response      (#PCDATA | function | use)*>

<!ELEMENT    instance      (set*, route+) >
<!ATTLIST    instance      name           ID         #REQUIRED
                           problem        IDREF      #REQUIRED
                           key            CDATA      #IMPLIED
                           requiredkey    CDATA      #IMPLIED
                           timelimit      NMTOKEN    #IMPLIED>

<!ELEMENT    set           (#PCDATA | function | use | item)*>
<!ATTLIST    set           param          IDREF      #REQUIRED>

<!ELEMENT    route         EMPTY>
<!ATTLIST    route         choice         IDREF      #REQUIRED
                           breakskey      CDATA      #IMPLIED
                           left           IDREF      #IMPLIED
                           forward        IDREF      #IMPLIED
                           right          IDREF      #IMPLIED
                           up             IDREF      #IMPLIED
                           down           IDREF      #IMPLIED>
```

LIITE 2: SOKKELOESIMERKKI

Tässä liitteessä on pieni sokkelopeliesimerkki. Se on alunperin toteutettu tämän diplomityön pohjana olleeseen diplomityöhön [4]. Sokkelon rakenne on kuvattu luvussa 4.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE problems SYSTEM "maze.dtd">

<problems xmlns="http://www.cs.tut.fi/~fyzix/" timelimit="60">
    <settings logo="fyzix-logo.png"
              correctimg="correct.gif"
              wrongimg="wrong.gif"/>

    <!-- Globaalit parametrit määritellään tässä. Ne näkyvät kaikkiin
         tehtäviin. -->
    <param name="g"
           description="Satunnaisen planeetan gravitaatiokiihtyvyys">
        <function name="random"/>(9.81, 100)
    </param>

    <!-- Taulukon määrittely. Taulukossa voi olla merkkijonoja... -->
    <param name="autot">
        <item>saab</item>
        <item>volvo</item>
    </param>

    <!-- ...tai numeerisia arvoja. -->
    <param name="kertoimet">
        <item>1.5</item>
        <item>10</item>
    </param>

    <!-- Huom. forward viittaa instanssin nimeen, ei tehtävän nimeen. -->
    <framestory forward="teht1_i1">
        <figure>Hiittinen.jpg</figure>

        "...siellä kaiikilla oli ni-iin muukaavaa..."
    </framestory>

    <!-- Peli voi loppua kahdella tavalla: -->
    <success name="onnistui">
        <figure>keto.jpg</figure>

        "...and they all lived happily ever after."
    </success>
    <failure name="poks">
        <figure>sieni.jpg</figure>
    </failure>

    <!-- Tehtävän määrittely. Huom. tästä ei muodostu tehtävää peliin ennen
         kuin siitä luodaan ilmentymä tiedoston lopussa määriteltävällä
         instance-elementillä. Siten on mahdollista luoda yhdestä tehtävästä
         useita instansseja. -->
    <problem topic="testi" name="teht1">

        <!-- Tyhjä parametrin määrittely tarkoittaa, että parametrin arvo on
             määriteltävä instanssin yhteydessä set-elementillä. -->
        <param name="pakko"/>

        <param name="mestarit">
            <item>Pablo Picasso</item>
            <item>Leonardo Da Vinci</item>
```

```
    </param>

    <!-- Taulukoita käsitellään array_funktioilla. Tässä array_rand
         palauttaa kysymys-parametrin arvoksi satunnaisen alkion
         taulukosta autot. -->
    <param name="kysymys">
        <function name="array_rand"/>(<use name="autot"/>)
    </param>

    <param name="kerroin">
        <function name="array_rand"/>(<use name="kertoimet"/>)
    </param>

    <param name="mestari">
        <function name="array_rand"/>(<use name="mestarit"/>)
    </param>

    <!-- Merkkijonoparametri, jonka sisällä voi käyttää (use) aiemmin
         määriteltyjä parametreja. Huom, tätä parametria ei voi
         instanssissa ylikirjoittaa, koska se on readonly. -->
    <param name="mjono" readonly="true">
        -= <use name="kysymys"/> =-
    </param>

    <problemstatement>
        Tämä piti asettaa instancessa: <use name="pakko"/>
        <use name="mestari"/> on muuten paras!

        Katso video:

        <video>tacoma.flv</video>

        Mikä näistä on <use name="kysymys"/>:n logo?
    </problemstatement>

    <!-- Huom. vastausvaihtoehdoille (choice) ei määritellä siirtymiä
         (left, right, ...) vaan yksikäsitteinen nimi. Siirtymät
         määritellään vasta luotaessa tehtävästä instanssi johonkin
         osaan tehtävägraafia. -->
    <choices showincorrect="2">
        <choice name="teht1_mjono" value="-1">
            <show type="text">
                <use name="mjono"/>
            </show>
            <response>
                Logo ei ole merkkijono.
            </response>
        </choice>

        <!-- Kuvan nimessä voi käyttää (use) parametreja, joten vastaus
             voi vaihtua kysymystä vastaavaksi. -->
        <choice name="teht1_kuva" value="1">
            <show type="image">
                <figure><use name="kysymys"/>.jpg</figure>
            </show>
            <response>
                Oikein!
            </response>
        </choice>

        <choice name="teht1_kaava" value="0">
            <show type="expression">
                <expression>
                    <use name="g"/> * <use name="kerroin"/>
                </expression>
            </show>
            <response>
                Mitähän mahdoit ajatella...
            </response>
        </choice>
    </choices>
</problem>

<!-- Edellä määritellystä tehtävästä luodaan instanssi. Pelimaailman
```

```
        rakenne muodostetaan siis näillä instance-elementeillä ja niiden
        sisään kirjoitettavilla route-elementeillä. Route viittaa aina
        yhteen vastausvaihtoehtoon (choice) ja määrittelee mihin suuntaan
        pelaaja voi mennä ja mihin suunta johtaa. Kohteeksi annetaan
        tehtävän instanssin nimi, ei tehtävän nimeä!
        Set-elementillä voidaan määritellä uusi arvo tehtävässä
        määritellylle parametrille. Globaaleja ei voi ylikirjoittaa. -->
    <instance name="teht1_i1" problem="teht1">
        <!-- Tämä parametri oli vaadittu määriteltäväksi täällä. -->
        <set param="pakko">Jokin arvo</set>
        <!-- Myös setin sisällä voi käyttää function ja use-elementtejä. -->
        <set param="kerroin">
            <function name="array_rand"/>(<use name="kertoimet"/>) - 2
        </set>
        <!-- Tehtävän paikallisen taulukonkin voi määritellä uusiksi. -->
        <set param="mestarit">
            <item>Michael Schumacher</item>
            <item>Mika Häkkinen</item>
        </set>
        <route choice="teht1_mjono" forward="teht1_i2"/>
        <route choice="teht1_kuva" forward="teht1_i2"/>
        <route choice="teht1_kaava" forward="teht1_i2"/>
    </instance>

    <!-- Tässä luodaan samasta tehtävästä toinen instanssi. -->
    <instance name="teht1_i2" problem="teht1">
        <set param="pakko">Jokin toinen arvo</set>
        <route choice="teht1_mjono" left="teht1_i1" right="poks"/>
        <route choice="teht1_kuva" left="teht1_i1" right="onnistui"/>
        <route choice="teht1_kaava" left="teht1_i1" right="poks"/>
    </instance>
</problems>
```