Juha Salorinne

Boxle-Ongelmanratkaisupelin ohjelmointi

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikan koulutusohjelma

Insinöörityö

11.4.2014

Metropolia

Tekijä Otsikko Sivumäärä Aika	Juha Salorinne Boxle-Ongelmanratkaisupelin ohjelmointi 25 sivua 11.4.2014
Tutkinto	insinööri (AMK)
Koulutusohjelma	tietotekniikka
Suuntautumisvaihtoehto	ohjelmistotekniikka
Ohjaaja	projektijohtaja Juha Huhtakallio

Insinöörityön aiheena on pelinluomisprosessi, jonka tarkoituksena on toimia lisäopetuksena koulussa opitun lisäksi.

Tavoitteena oli suunnitella ja toteuttaa peli, jonka tekeminen on sopivan haastavaa. Suunnitelman pohjana toimivat asiat, jotka ovat tuttuja oppitunneilta ja vasta myöhemmin lisätään uusia opeteltavia asioita, jotta projektissa pääsisi hyvään alkuun.

Peli suunniteltiin vuoropohjaiseksi ja toteutettiin C++ ohjelmointikielellä. Toteutuksessa käytettiin lisäksi SFML-kirjastoja sekä Github varastoa version hallintaan.

Projektin aikana pelille suoritettiin kaksi julkista testausta, joihin osallistui satunnaisesti valittu 15 pelaajan joukko. Tämän lisäksi koodia ja peliä testattiin itsenäisesti aina, kun uutta koodia oli tehty.

Tuloksena syntyi toimiva ja pelattava peli, joka täytti suunnitteluvaiheessa asetetut tavoitteet. Pelissä on kuitenkin vielä paljon työtä ollakseen täysin valmis tuote.

Projektin aikana opittiin paljon uutta suunnittelun ja toteutuksen saralla. Suunnittelussa on hyvä rauhassa miettiä asioita, jotta koodi on hyvää ja jatkon kannalta helposti laajennettavaa. Toteutuksen puolelta opittiin, miten kiire voi saada aikaiseksi huonon koodin tuottamista.

Avainsanat	peliohjelmointi, ongelmanratkaisupeli, C++

Author Title Number of Pages Date	Juha Salorinne Programming of Boxle puzzle solving game 25 pages 11 April 2014
Degree	Bachelor of Engineering
Degree Programme	Information Engineering
Specialisation option	Software Engineering
Instructor	Juha Huhtakallio, Project Manager

The topic of this thesis is the process of game creation. The goal was to plan and execute a game so that the game creation process would be challenging enough. A starting point for the plan were basic skills in C++, and at a later stage new and unfamiliar things were included in the form of player input and graphics. This was to enable a good start on the project.

The game was designed to be a turn-based puzzle game, and it was programmed in C++. Additionally, SFML libraries were used to create the game window, and handle player input and Github were used in version control.

During the project there were two public tests where fifteen randomly chosen players did the testing. In the second test one bug was found and it was fixed. In addition, the code and game were tested by one person whenever any new code was written.

As a result, a functioning and playable game was made that fulfilled the goals set in the designing phase. However, the game still needs a lot of work to be completely done.

The project gave a lot of new information about the planning and execution of games. In the planning phase it is good to think about things with time so that the code is good and easy to expand in the future. As far as execution is concerned, being in a hurry can result in bad code so enough time should be reserved for that as well.

In conclusion, many new skills and working methods were acquired that will be useful in future programming projects.

Keywords	game programming, puzzle game, C++

Metropolia

Sisällys

Lyhenteet

IDE	Integrated development environment. Yleinen nimitys eri ohjelmoinnin kehitysympäristöille.
SFML	Simple and Fast Multimedia Library. Kokoelma ohjelmointiin käytettyjä kirjastoja, joilla on helppoa toteuttaa multimediaa sisältäviä ohjelmia.
XML	Extensible Markup Language. HTML:ää muistuttava merkintäkieli, jonka ideana on muotoilla ohjelmakieli koneelle ja ihmiselle luettavaan muotoon.

1 Johdanto

Tämän insinöörityön aiheena on katsaus pelinluomisprosessiin aina suunnittelusta valmiiseen peliin asti. Tarkoituksena on oppia uusia asioita, joita työn edistyessä tulee vastaan.

Ensimmäisessä osiossa tarkastellaan pelin suunnitteluprosessia ja pelin eri osa-alueita sekä niihin liittyviä päätöksiä.

Toisessa osiossa käydään läpi suurimpia pelin toteutuksessa tarvittavia ja käytettyjä työkaluja. Lisäksi tarkastellaan pelin eri kehitysversioita. Pelin testaus käydään myös lyhyesti läpi.

Kolmannessa osiossa pohditaan pelin tulevaisuutta. Mitä kaikkea peli vielä tarvitsee tullakseen myyntikelpoiseksi tuotteeksi ja miten peliä voisi sitten markkinoida ja myydä?

2 Pelin suunnittelu

2.1 Lajityypin valinta

Pelin lajityypiksi valikoitui puzzle eli ongelmanratkaisupeli. Puzzle-peleissä yleensä tarkoituksena on ratkaista jokin ongelma loogisella päättelyllä läpäistäkseen kentän tai päästäkseen pelissä eteenpäin. Tähän projektiin tämä lajityyppi valikoitui oman mielenkiinnon lisäksi myös yleisesti ottaen tämäntyyppisten pelien toiminta vuoropohjaisesti, joka pitää erilaiset asiat yksinkertaisempina läpi projektin.

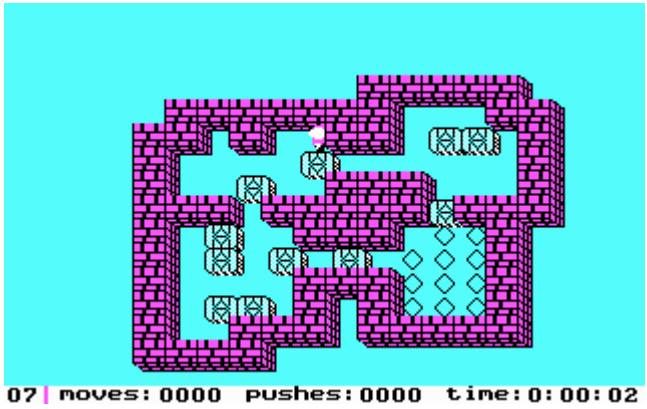
Lajityyppinä ongelmanratkaisupelit eivät ole sieltä suosituimmasta päästä, sillä pääpaino peleissä on jonkin ongelman ratkaiseminen yleensä loogisella päättelyllä, joten pelaaja saattaa käyttää enemmän aikaa ongelman ratkaisemiseen mielessään kuin itse pelaamiseen. Pääsääntöisesti ongelmanratkaisupeleissä on tarkoituksena joko suoraan tai epäsuorasti käsitellä erilaisia muotoja, värejä tai kuvioita päämääränä jokin ennalta määritelty lopputulos.

Erilaisista peli-ideoista juuri tämä sopi parhaiten itsenäisesti toteutettavaksi peliksi, sillä muut ideat ovat joko liian suuritöisiä tai sopivat paremmin pelattavaksi jollain muulla alustalla kuin tietokoneella.

2.2 Peli-idean suunnittelu

Suunnittelun kriteereinä oli oma osaamistaso. Tarkoituksena oli kehittää peli, jonka saisi hyvään alkuun ja sen jälkeen vaatisi uusien asioiden opettelua jatkoa varten. Lähtökohdaksi valikoitui laatikoiden työntelyyn perustuva ongelmanratkaisupeli. Yleisesti ottaen laatikontyöntelyä vaativien ongelmanratkaisupelejä kutsutaan Sokobaneiksi.

Tämä juontaan juurensa ensimmäiseen tämän tyyliseen julkaistuun peliin nimeltä Sokoban (Kuva 1). Peli ilmestyi 1982 Japanissa ja sen nimi tarkoitti varastomiestä. [1.] Pelissä on tarkoituksena työntää laatikoita ennalta määrättyihin paikkoihin ja kentän voittaa saamalla kaikki laatikot niihin.



Kuva 1. Sokobanin pelinäkymä

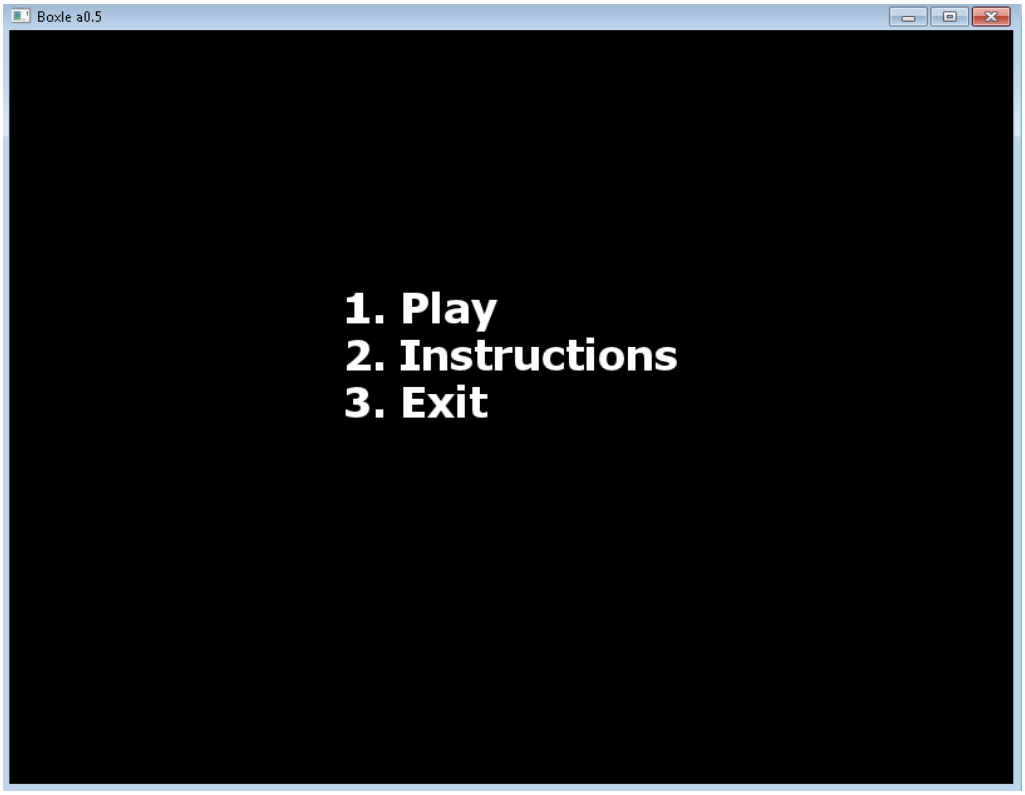
Tämän kaltaisia pelejä on satoja, eikä kaikki erityyppiset ole entuudestaan tuttuja, mutta erottuakseen tunnetuimmista, peli toimii kolmessa ulottuvuudessa. Normaalin x- ja y-akselien suuntaisen liikkumisen lisäksi myös z-akselilla voi liikkua.

Pelin nimi Boxle tulee yhdistämällä sanat Box ja Puzzle, eli laatikko ja ongelma, sillä siitähän pelissä on kyse.

Peli-idea syntyi tässä tapauksessa siltä pohjalta, mihin omat taidot tuntuvat riittävän tällä hetkellä. Peliä ei suunniteltu ja demottu täysin valmiiksi, vaan alettiin toteuttamaan ideaa ja katsomaan miten se toimisi, ja tarpeen vaatiessa muokkaamaan sitä.

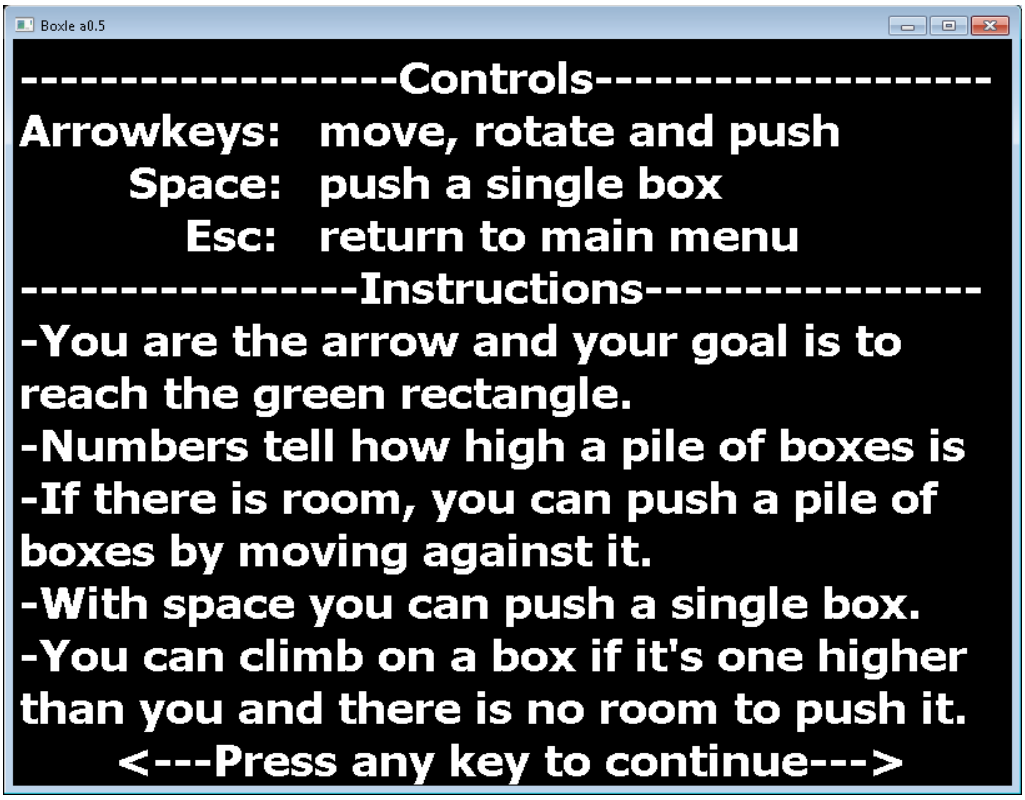
2.3 Pelin toiminta

Pelin päävalikko on yksinkertainen ja siellä navigointi tapahtuu numeronäppäimillä (Kuva 2).



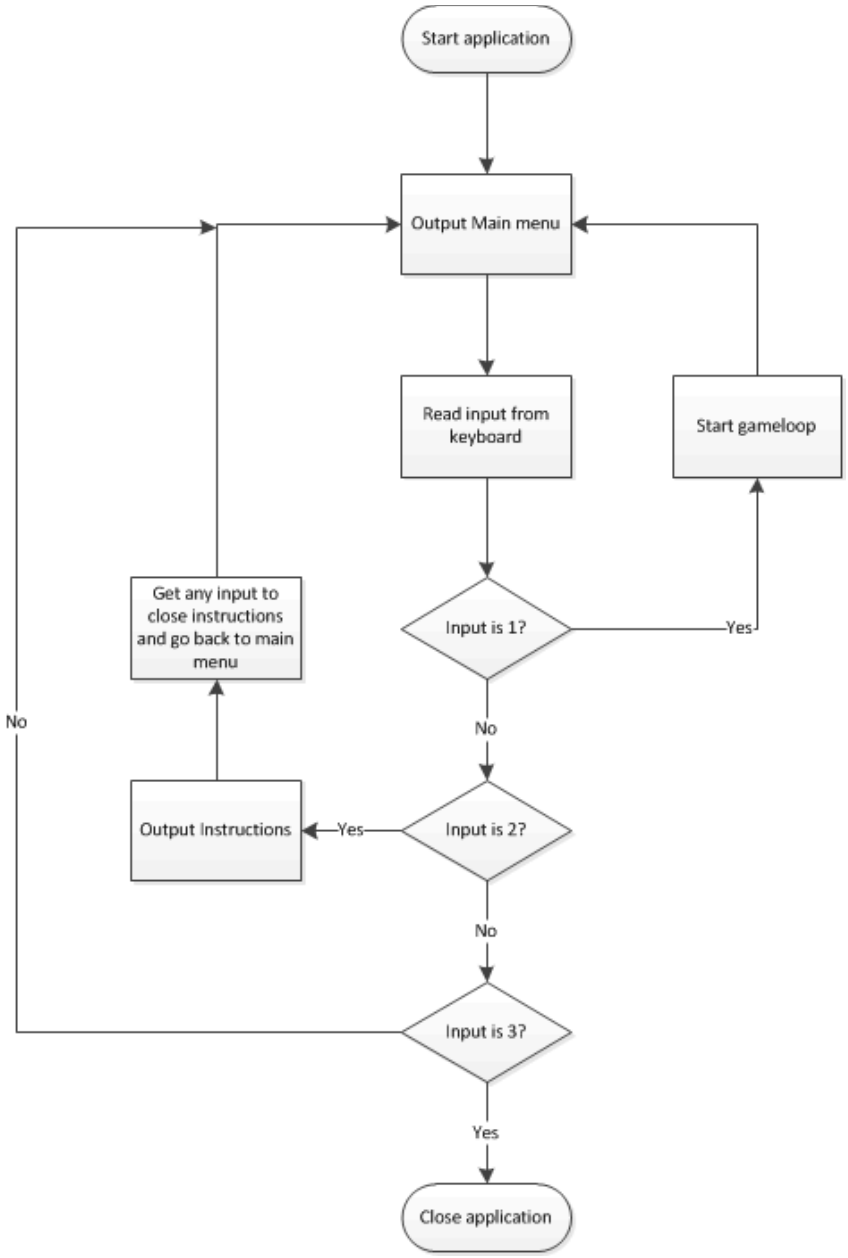
Kuva 2. Pelin päävalikko.

Näppäimellä 1 peli alkaa alusta, jos peli on juuri käynnistetty, muutoin pelaaminen jatkuu viimeksi ladatusta kentästä.



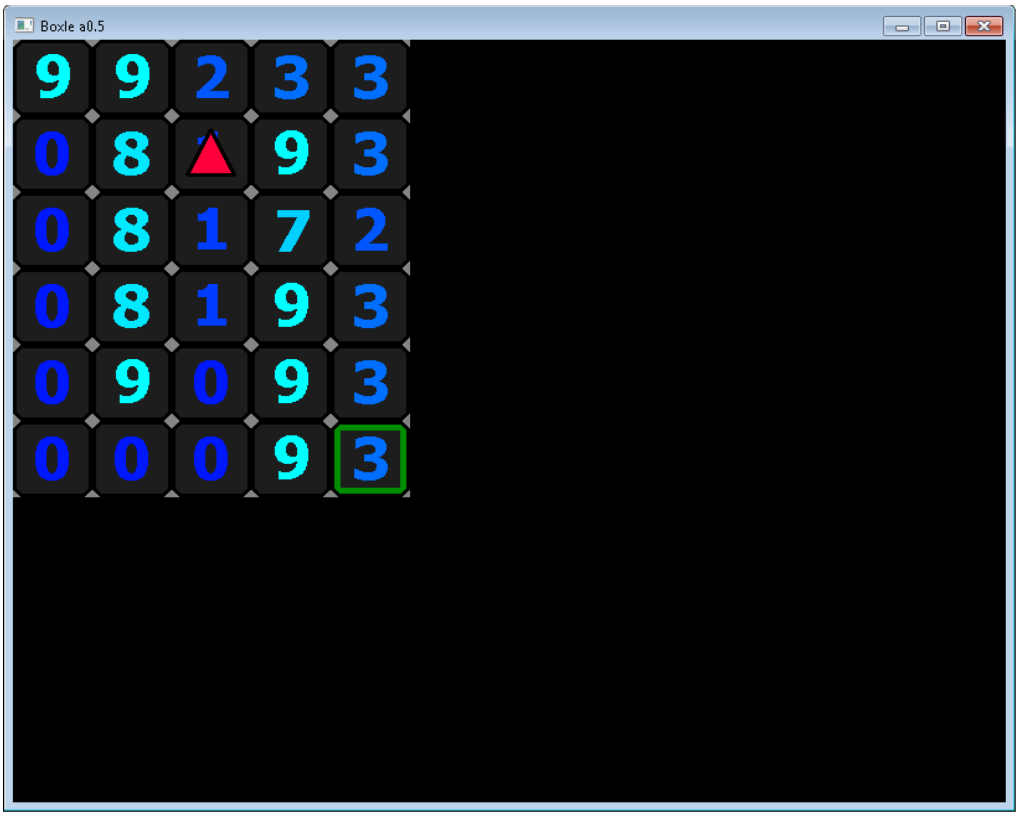
Kuva 3. Pelin ohjeruutu.

Näppäimellä 2 avautuu kuvan 2 ohjeruutu, joka selittää pelin toiminnot ja idean. Testipalautteen mukaan ohjeet ovat selvät, mutta jatkokysymyksillä selvisi, että kuvallinen ohje voisi toimia paremmin. Kuvallinen ohjeruutu selventävillä esimerkeillä on suunnitteilla pelin jatkokehityksessä. Näppäin 3 sulkee ohjelman (kuva 3).



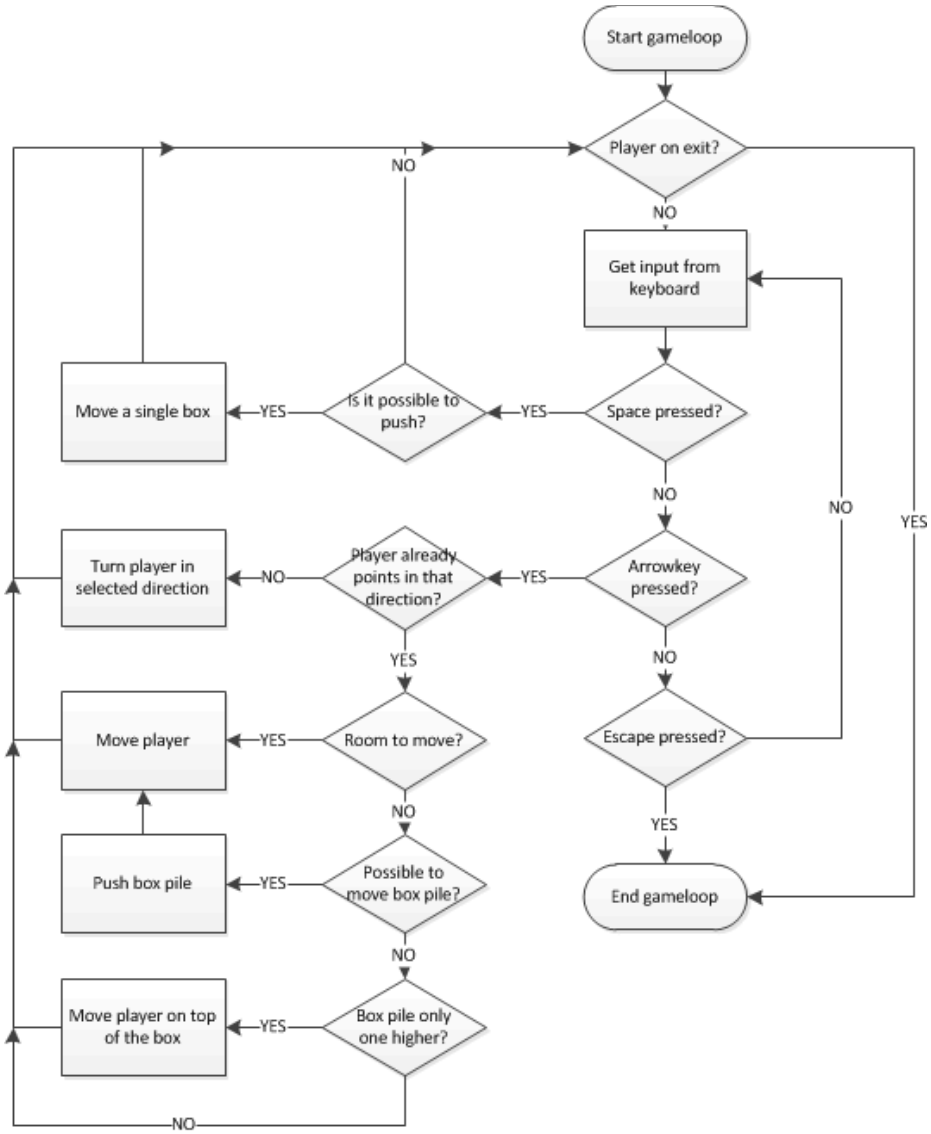
Kuva 4. Pelin päävalikko vuokaaviona.

Päävalikosta siis käynnistetään itse peli, joka ensimmäisellä käynnistyskerralla lataa pelin mukana tulevan kenttäpaketin ensimmäisen kentän. Kentän läpäistyä ladataan paketin seuraava kenttä, kunnes kaikki kentät on suoritettu. Jos päävalikkoon palaa kesken pelin, niin pelaamista pääsee jatkamaan samasta kentästä valitsemalla pelin pelaamisen painamalla 1 (kuva 4).



Kuva 5. Pelinäkymä.

Itse pelissä pelaaja ohjaa nuolella kuvattua hahmoa ja tavoitteena on laatikoita työntelemällä päästä uloskäynnille, joka on merkitty vihreäreunaisella laatikolla. Numerot kentällä edustavat laatikkopinojen korkeuksia. Kävelemällä päin laatikkopinoa pelaaja voi työntää sitä, jos sen takana on tilaa. Pelaajan tulee siis hahmottaa kulloinenkin oma sijainti korkeudessa suhteessa ympäristöönsä. Laatikoiden päälle voi myös kiivetä, jos kävelee päin pinoa, joka on vain yhden korkeammalla kuin pelaaja ja sen takana ei ole tilaa työntämiseen. Lisäksi välilyönnillä voi työntää yksittäisen laatikon pinosta sen sijaan, että työntäisi koko pinoa (kuva 5).



Kuva 6. Pelin toiminta vuokaaviona.

Kentän lopetusehtona on siis se, että pelaaja on uloskäynnin päällä, ja niin kauan kuin asia ei näin ole, pyörii peli (kuva 6).

Tämän suunnitelman mukaisesti peli siis tehtiin ja useiden testien jälkeen on löytynyt vielä pelin toiminnallisuudesta asioita, joita tullaan jatkossa muuttamaan.

2.4 Graafinen ulkonäkö

Graafiselta ulkoasultaan peli on alusta asti suunniteltu näyttämään abstraktilta esitykseltä, eli vaikka pelissä työnnetään laatikoita, eivät ne ulkonäöltään näytä sellaisilta. Laatikkopinoja ja niiden korkeuksia kuvastetaan numerosymboleilla. Numeroiden väritys myös vaalenee, kun numero suurenee. Tällä on pyritty selventämään pinojen korkeuseroja. Värimaailmaksi on valittu tummahko sävymaailma. Pelaajaa kuvastava punainen nuoli sekä uloskäyntiä kuvastava vihreä-reunuksinen laatikko erottuvat selvästi muusta pelin sisällöstä.

3 Pelin toteutus

3.1 Käytetyt työkalut ja menetelmät

Seuraavassa listataan tärkeimmät projektissa käytetyt työkalut ja tietolähteet.

Code::Blocks ja C++

Code::Blocks on ilmainen avoimen lähdekoodin IDE (Integrated development environment eli kehitysympäristö) monelle eri alustalle. Ohjelma tukee C-, C++- ja Fortran-kieliä. Sen lisäksi, että ohjelma itsessään on monipuolinen, voi sitä myös helposti laajentaa liitännäisillä. [2.]

Projektin toteutuskieleksi valikoitui C++ sen tarjoamien valmiiden toimintojen takia ja lisäksi se tuntui parhaimmalta vaihtoehdolta. Pelkkä C olisi myös toiminut, mutta siinä olisi joutunut itse toteuttamaan monia asioita, jotka C++:ssa on jo valmiina. Jos pelistä olisi tehty Android-versio mobiileille alustoille, niin Java saattaisi olla parempi kieli.

SFML

Simple and Fast Multimedia Library on kokoelma kirjastoja, jotka tarjoavat helppoja tapoja toteuttaa erilaisia multimediaa sisältäviä ohjelmia. Kirjastoissa on system-, window-, graphics-, audio- ja network-moduulit. System-moduuli sisältää ajan ja säikeiden käsittelyä. Window-moduuli sisältää ikkunoiden luonnin ja käsittelyn,

tapahtumat, näppäimistön, hiiren ja peliohjaimen käsittelyn sekä OpenGL:n käytön. Graphics-moduuli sisältää kaksiulotteisten asioiden piirron, kuvahahmot ja tekstuurit, tekstit ja fontit, muodot, erilaista kuvan ja muodon manipulointia, erikoisefektejä sekä kameran ohjausta näkymillä. Audio-moduuli sisältää äänen ja musiikin soiton, äänen nauhoittamisen, omat audio lähteet sekä kolmiulotteisen tila-audion. Network-moduuli sisältää kaiken tarpeellisen, jotta ohjelmat voisivat keskustella verkon välityksellä keskenään. [3.]

Projektissa käytettiin SFML 2.1-versiota. Projektissa ei käytetty kaikkia tarjolla olevia moduuleita, mutta jatkokehitystä ajatellen ne tarjoavat hyvän pohjan pelin täysin valmiiksi saattamiselle.

SFML-opaste sivuilla olevien esimerkkien ja dokumentaatioiden lisäksi Youtubesta löytyy SFML 2.0-opastevideosarja, sekä C++-videoita CodingMadeEasy-nimimerkillä toimivan Peter Henryn kanavalla. Videoissa käydään SFML 2.0:n eri ominaisuuksia perinpohjaisesti läpi yksinkertaisten esimerkkien kanssa. SFML esimerkkeinä käytettiin muun muassa erilaisia peleihin soveltuvia toiminnallisuuksia, joita pystyi soveltamaan tässä projektissa. C++-videoilta oli hyvä tarkastaa ja kerrata perus C++ toiminnallisuuksien käyttöä ja toimintaa.

Github

Github on yksi monista Git-alustoista. Git on kehitetty hajautettuun versionhallintaan sekä lähdekoodin hallinnointiin. Pelin versiohallintaan ja lähdekoodin säilytykseen käytin Githubia, johon Metropolian opiskelijana saa yksityisen varaston. Versiohallinnan ja koodin säilytyksen lisäksi Github mahdollistaa myös muiden ihmisten osallistumisen projektiin ja selkeän verkkokäyttöliittymän, jolla näkee helposti erot eri tiedostojen versioiden välillä. Verkkokäyttöliittymän lisäksi on mahdollista myös ladata koneelle pieni ohjelma ja määritellä kansio, joka linkitetään omaan varastoon.

Ohjelma pitää kirjaa valitun kansion sisällä olevista tiedostoista (yleensä kooditiedostot) ja vertaa niitä varastossa oleviin. Jos eroja löytyy, niin ohjelma listaa ne, ja napin painalluksella uudet ja muutetut kooditiedostot voi lähettää varastoon. Tämä tapahtuu useimmiten sen jälkeen, kun jotain on koodissa muutettu ja uudempi versio tallennettu. Jos myöhemmin tulee tarve muuttaa koodia aikaisempaan versioon, voi sen tehdä napin painalluksella. Git mahdollistaa myös koodiprojektin haaroittamisen, jolloin eri

haaroissa voi tehdä vaikka erilaisia kokeiluita ilman, että niiden päivittäminen varastoon muuttaisi päähaarassa olevaa koodia.

Problem solving with C++ kirja

Kirjalähdemateriaalina käytössä oli Problem solving with C++, 5 painos. Kirjan 5. painos on vuodelta 2005, joten jotkin kirjassa käytetyt ohjelmointikieleen liittyvät termit ovat vähemmän tunnettuja esimerkiksi ”free store” jota nykyisin kutsutaan kasaksi ”heap”. Nykyisin kirjasta on saatavilla 8 painos. Kirja tuli käyttöön sellaisissa suunnittelutilanteissa, joissa ei ollut nettiä tarjolla juuri sillä hetkellä.

3.2 Kehitysversiot

Seuraavaksi käydään läpi pelin toteutuminen osa osalta. Projektin alkupuolella asiat suunniteltiin huolellisesti ja ylensä niiden toteutuksessa otettiin huomioon myös jatkossa jo tiedossa olevat muutokset, jotta myöhemmin ei tarvitsisi koodia muokata paljoa. Ohjelma on paljolti jaettu pienempiin moduuleihin. Tämä mahdollisti keskittymisen aina kulloinkin tekeillä olevaan asiaan sekä sen, että myöhemmin oli helppo korjata koodia, jos jokin ei toiminut kuten piti.

Ensimmäinen versio

Pelin ensimmäisessä pelattavassa versiossa käytettiin kahta taulukkoa. Ensimmäinen taulukko sisälsi kentän tiedot ja oli tyyppiä Int. (Int tarkoittaa ohjelmointikielessä jotakin muuttujaa, joka voi sisältää numeroita) Toinen taulukko oli tyyppiä Char (Char-tyypin muuttuja taas sisältää merkkejä). Kaikki laatikoiden siirtelyt tapahtuivat ensimmäisessä taulukossa, jolloin taulukon solujen muokkaukset pystyi tekemään helposti lisäämällä ja vähentämällä numeroita, jotka symboloivat laatikkopinojen korkeuksia.

Pelaajahahmolle ja uloskäynnille oli omat luokat, joiden sisällä pidettiin kirjaa kummankin sijainnista. Kun muutokset kenttään oli tehty, Int-taulukon tiedot siirrettiin Char-taulukkoon tekemällä tyyppimuunnos numerosta merkiksi (Int → Char), jonka jälkeen pelaajaa ja uloskäyntiä kuvaavat merkit lisättiin niiden koordinaatteja vastaaviin soluihin. Lopuksi Char-taulukon sisältö tulostettiin komentoriville. Numeroilla siis esitettiin laatikkopinojen korkeudet, uloskäynti oli merkki 'E' ja pelaajaa kuvasti pelaajan

osoitussuunnasta riippuen jokin seuraavista merkeistä: '<', 'A', '>' tai 'V'. Koska merkit peittivät allansa olevan solun, tulostettiin ruudulle myös lisätietona pelaajan ja uloskäynnin korkeus. Pelin tässä vaiheessa tarkoitus oli saada toimiva runko, jolla testata toiminnallisuuksia, joten pelaajan ohjaaminen tapahtui syöttämällä jokin numeroista 2, 4, 6 tai 8 ja painamalla enter. Numerot tulevat näppäimistön oikeassa laidassa sijaitsevien numeronäppäimien muodostamasta ristikosta, jossa 8 on ylös, 6 oikealle, 2 alas ja 4 vasemmalle.

Ensimmäisessä versiossa itse koodi ei ollut haasteellista tai uutta, mutta pelilogiikan toteutus ja toimimaan saanti vaati työtä. Tässä vaiheessa projektia tarkoituksena oli päästä nopeasti testaamaan ja saamaan pelilogiikka toimimaan tarkoitetulla tavalla, joten kaikki tarkastukset siitä, meneekö laatikko tai pelaaja ulos kentältä, hoidettiin kahden solun paksuisilla seinillä.

Toinen versio

Perusrungon ollessa kunnossa seuraava vaihe oli reaaliaikainen komentojen lukeminen sekä grafiikka. Näppäimistön painallusten lukeminen tapahtuu käyttäen SFML-kirjastosta löytyvää tapahtumaa eli eventiä sekä pollEvent-funktiota, jolle syötetään tapahtuma (koodiesimerkki 1).

```
77          sf::Event event;
78          while(gameloop){
79              while(window.pollEvent(event)){

88                  if(event.type == sf::Event::KeyPressed){
89                      if(event.key.code == sf::Keyboard::Num1){
90                          selection = 1;
91                      }
92                      else if(event.key.code == sf::Keyboard::Num2){
93                          selection = 2;
94                      }
95                      else if(event.key.code == sf::Keyboard::Num3){
96                          selection = 3;
```

Koodiesimerkki 1. Päävalikossa käytettävien näppäinten lukeminen.

Tapahtumatyyppejä on erilaisia, mutta tässä projektissa käytettiin vain näppäinpainalluksia. Näppäimiä lukiessa oli myös mahdollista valita, onko toisto päällä vai ei. Toistolla tarkoitetaan tässä tilannetta, jossa käyttäjä pitää yhtä nappia pohjassa pitkään. Jos toiston poisti, niin pitkä painallus luetaan vain yhdeksi painallukseksi,

kuten tässä pelissä. Jos toisto taas on päällä, niin painettu näppäin luetaan uudestaan niin kauan kuin se pysyy painettuna. Reaaliaikaisen liikkumisen toimivaksi saamisen johdosta vanha liikkumistapa poistettiin.

Pelin grafiikka sijaitsee yhdessä tiedostossa, jota kutsutaan myös tekstuuriatlakseksi. Yhden tiedoston käyttäminen monen pienen sijaan on tehokkaampaa tämän tyyppisessä pelissä, joka koostuu soluista. Tietokoneen muistiin ladataan siis yksi kuva, josta sitten haetaan koordinaattien perusteella eri kohdista pienempiä palasia. Yleistä tekstuuriatlaksien sisältämillä pienemmillä kuvilla on niiden koon sitominen kahden potensseihin [4.] Tässä pelissä pienet kuvat ovat kooltaan 64 x 64 pikseliä.

```
33  void drawMap(sf::Texture& tilemap, sf::Sprite& tiles, sf::RenderWindow& window,Level level){
34      int i , j;
35      for(j = 0 ; j < level.getWidth() ; j++){
36          for(i = 0; i < level.getHeight() ; i++){
37              tiles.setTextureRect(sf::IntRect(0, (TILESIZE*level.y[i][j]), TILESIZE, TILESIZE));
38              tiles.setTexture(tilemap);
39              tiles.setPosition(i*TILESIZE, j*TILESIZE);
40              window.draw(tiles);
41          }
42      }
43  }
```

Koodiesimerkki 2. Pelikartan piirtäminen.

Pelinäkymän piirtäminen tapahtuu kahdessa for-silmukassa. Sisemmässä silmukassa haetaan ensin piirrettävän grafiikan kohta isommasta kuvasta käyttäen pelitaulukon tietoja. Isossa kuvassa pelin laatikot sijaitsevat allekkain, joten x-akselin arvoksi on asetettu 0, y-akselin arvoksi asetetaan kentässä olevan laatikkopinon arvo kertaa TILESIZE-vakio, joka on aikaisemmin määritelty luvuksi 64. Näillä kahdella arvolla kerrotaan aloituspiste isossa kuvassa. Tämän jälkeen piirrettävän kuvan kooksi kerrotaan kaksi kertaa TILESIZE. Lopputuloksena saadaan siis 64 x 64 pikseliä oleva kuva, joka haetaan pelikentän tietojen mukaiselta korkeudelta. Piirretävän kuvan koordinaatit määräytyy for-silmukoiden kierrosten arvoilla, joita käytetään tiles.setPosition-kohdassa. Ensimmäisellä kierroksella kuva piirretään alkamaan siis kohdasta 0 * 64, 0 * 64. Lopuksi window.draw piirtää kuvan ruudulle (koodiesimerkki 2).

Kun pelaajan liikuttaminen ja grafiikan saaminen näytölle saatiin toimimaan, oli aika siivota koodista pois ylimääräisiä rivinvaihtoja ja lisätä rajoittimet sille, ettei pelaaja voi

kävellä ulos kentältä tai työntää laatikoita ulos. Tämän siivouksen yhteydessä poistettiin kentän reunoilla olevat kahden solun paksuiset seinät. Koodista poistettiin myös vaihe, jossa numeroita sisältävä taulu muutettiin merkkimuotoiseksi ja merkkimuotoinen taulukko, sillä niitä ei enää tarvittu.

Kolmas versio

Pelin perusrungon ollessa nyt valmis oli aika tehdä useita kenttiä. Tätä varten staattinen kaksiulotteinen taulukko piti vaihtaa vektoreihin, jotta kentät voisivat olla eri kokoisia. Tämä vaihe vaati huomattavasti työtä, mutta onneksi kaksiulotteinen vektoritaulukko toimii samalla logiikalla kuin normaali staattinen taulukko, joten aivan kaikkea ei tarvinnut uudelleen ohjelmoida. Suurin ero staattiseen kaksiulotteiseen taulukkoon oli se, että kaksiulotteinen vektoritaulukko on itseasiassa vektori, jossa on vektoreita. Ensimmäinen testi uudella systeemillä opetti sen, että se mikä staattisella taulukolla ei aiheuttanut ongelmia, niin aiheutti niitä välittömästi vektoritaulukolla. Aikaisemmin tehdyt rajoittimet eivät olleet aivan täydellisiä, sillä niissä oli mahdollista pelaajan toimista riippuen käydä lukemassa tietoja taulukon ulkopuolelta, mikä ei vanhassa versiossa ilmennyt mitenkään useasta testikerrasta huolimatta. Uudelleen kirjoitettujen rajoitusten jälkeen tämä ongelma oli korjattu. Tähän asti testikenttä oli suoraan koodiin kirjoitettuna, mutta nyt tarkoitus oli ladata kenttätiedot tiedostosta. Kentän tietojen tallennusmuodoksi tuli XML-tyylinen malli, jossa eri asioiden tiedot on nimettyjen tagien välissä. (kuva 8).

```
<Name>kuusi</Name>
<Height>7</Height>
<Width>6</Width>
<Player>0 0 0</Player>
<Exit>0 5 3</Exit>
<Map>0 2 0 2 0 3 2 0 2 0 2 0 0 2 0 2 0 2 2 0 2 0 2 0 0 2 0 2 0 2 2 0 2 0 2 0 0 2 0 2 0 1 </Map>
```

Kuva 7. Esimerkki pelin kenttädatan tallennusmuodosta.

Alkuperäisenä ajatuksena oli se, että yhdessä tiedostossa olisi useita kenttiä, joiden tiedot haettaisiin nimen perusteella. Kentän nimen jälkeen tulee kentän koko, pelaajan ja uloskäynnin koordinaattien x, y ja z koordinaatit ja viimeisenä itse kenttä.

Kentän tietojen kirjoitus tiedostoon sujui ongelmitta. Ongelmia tuli vasta, kun oli aika lukea tiedostosta tiedot kenttä-olioon. Tagien välissä sijaitsevan tiedon hakeminen onnistui helposti aikaisemmin koulun ohjelmointikurssilla tehdyn tehtävän avulla. Mutta

ongelmia tuli numerolistojen kanssa. Tarkoituksena oli saada luettua yksittäiset numerot, joiden pohjalta luotaisiin pelaajan, uloskäynnin ja kentän tiedot. Ongelma oli se, että käytetty valmis koodi luki kaiken tagien välissä olevan yhdeksi stringiksi eli merkkijonoksi. Lopulta tämäkin ratkesi stringstreamia käyttämällä, jonka avulla sai merkkijonon purettua yksittäisiksi merkeiksi. Tämän jälkeen koodista poistettiin kovakoodattu testikenttä.

Neljäs versio

Testikentän lataamisen tiedostosta peliin toimivaksi saamisen jälkeen tavoitteena oli tehdä kymmenen kentän karttapaketti. Valitettavasti kentän tekeminen kirjoittamalla numeroita tiedostoon ei ollut kovin sujuvaa tai mukavaa, joten kenttäpaketin kooksi tuli seitsemän kenttää. Paketin ensimmäinen kenttä erosi testikentästä paitsi sisältönsä myös koon suhteen. Tämä paljasti ohjelmointivirheen, jonka takia pelaajan ja uloskäynnin tiedot eivät latautuneet oikein. Ongelma oli nopeasti ratkaistu välittämällä referenssit pelaajan ja uloskäynnin olioista muutamalle funktiolle käyttämällä &-merkkiä funktioiden argumenteissa.

Projektin saaminen tähän vaiheeseen oli vienyt niin paljon aikaa, että toimivan kenttäkierron tekemisessä käytettiin huonoa ohjelmointikäytäntöä kovakoodaamalla pitkä if - else-lohko, jossa laskurin avulla määriteltiin seuraavaksi ladattava kenttä. Jokainen kentä sijaitsee myös omalla tiedostollaan, koska ei ollut aikaa selvittää, miten yhden tiedoston paketin saisi toimimaan. Lopuksi vielä yksinkertainen päävalikko sekä ohjesivu ja peli oli tarpeeksi valmis tätä projektia varten. Testautuksessa löytyi vielä yksi virhe, joka liittyi laatikoiden työntelyä rajoittaviin määreisiin ja joka oli nopea korjata.

3.3 Pelin testaus

Peliä testattiin kahdella eri tavalla. Pienemmät välitestaukset eri muutosten ja versioiden välillä suoritettiin itsenäisesti. Isommissa testauksissa peliä jaettiin testattavaksi muille. Testihenkilöt valittiin satunnaisesti niiden ihmisten joukosta, jotka kulloisellakin hetkellä olivat tavoitettavissa. Suurin osa testaajista pelaa myös muita pelejä. Projektin aikana suoritettiin kaksi isompaa testausta ja kummassakin testaajia oli 15.

Yleiset ohjeet testaukseen oli läpäistä kentät, koettaa työntää laatikoita pois kenttäalueelta liikkumalla niitä päin sekä työntämällä välilyönnillä yksittäisiä laatikoita. Myös pelaajahahmoa tuli yrittää liikuttaa pois kentältä.

Ensimmäinen isompi testaus tapahtui, kun grafiikat ja ohjaus oli saatu toimimaan testikentällä. Tämän testin tarkoituksena oli selvittää, toimiiko peli myös muilla koneilla. Testaajille selitettiin myös pelin toiminta sanallisesti, sillä tässä vaiheessa ohjeita ei voinut nähdä pelissä. Pelin toimimisen ja itse peli-idean testaamisen lisäksi pelaajia ohjeistettiin yrittämään rikkoa peliä kokeilemalla erilaisia asioita. Tällä testauskerralla ei ongelmia ilmennyt.

Toinen isompi testaus oli kattavamampi peliin lisättyjen ominaisuuksien takia. Tällä kertaa pelin toimintaa ja ideaa ei selitetty, sillä pelistä löytyi ohjesivu. Osa testaajista oli samoja kuin ensimmäisellä kerralla, joten heille pelin toiminta oli entuudestaan tuttu. Toinen osa taas oli uusia, ja he joutuivat pärjäämään pelkästään ohjesivun perusteella. Kummatkin ryhmät onnistuivat läpäisemään pelin seitsemän kenttää, ja vain yksi löysi toiminnallisen ohjelmointivirheen, joka ei kuitenkaan estänyt kenttien läpäisemistä. Testaajia pyydettiin kertomaan mielipiteitä itse pelistä ja myös ohjesivun selkeydestä. Suurin osa oli tyytyväisiä ohjesivuun, mutta muutama olisi kaivannut havainnollisempia ohjeita.

Kaikki paitsi yksi isompien testausten testeistä tapahtui eri puolilla maailmaa. Tämä yksi testi tapahtui seurantani alaisena, ja se paljasti vakavia ongelmia kenttien sisällön suhteen. Useat kentistä pystyi ratkaisemaan turhan monellakin eri tapaa. Osa näistä eri tavoista teki kenttien ratkaisemisesta aivan liian helppoa.

Testauksessa ilmi tulleet asiat kävivät hyvänä muistutuksena siitä, miten eri ihmiset ajattelevat asioita eri tavoin. Se tapa, jonka oli suunnitellut olevan jonkin kentän ratkaisu, saattoikin olla vain yksi monista eri tavoista. Näin pienen pelin testauttaminen muilla ei vielä vaatinut suuria järjestelyitä. Jos kyseessä olisi isompi peli, voisi testaajille jakaa esimerkiksi listaa, jossa olisi lueteltu pelin eri osa-alueita ja niitä vielä jaettu pienempiin komponentteihin, joita testaajat kävisivät järjestelmällisesti läpi.

4 Pelin tulevaisuus ja markkinointi

Koska peli ei ole vielä lähellekään tarpeeksi valmis ja viimeistelty ollakseen myyntikelpoinen, esitän seuraavaksi pohdintoja siitä, mitä pelistä puuttuu ja mitä siinä pitäisi parantaa jotta se tulisi myyntikelpoiseksi. Myös valmiin pelin mahdollista markkinointia käsitellään pikaisesti.

4.1 Pelin vaatima lisätyö

Alkuperäisessä suunnitelmassa peli olisi sisältänyt kenttäeditorin, jonka avulla olisi ollut helppo tehdä uusia kenttiä. Editori olisi näkymältään samantyyppinen kuin itse peli, mutta sisältänyt muutamia painikkeita, joilla voisi tallentaa ja testata tekemäänsä kenttää. Kenttiä voisi myös tallentaa yhteen tiedostoon tehden siitä kenttäpaketin. Tämä vaatisi tiedostoihin lisätietoa sen sisältämistä kentistä tai erillisen tiedoston, josta nämä luetaan. Kun pelaajilla olisi mahdollisuus luoda omia kenttäpaketteja kuuluisi valmiiseen peliin myös mahdollisuus jakaa niitä muille verkon välityksellä. Tällöin pelissä pitäisi olla jokin sisäinen tarkastusvaihe, jossa tekoäly testaa, voiko kentän läpäistä, ennen kuin sen voi julkaista muille.

Pelimekaniikallisesti pitäisi myös testata vaihtoehtoista toimintaa, joka mahdollisesti korjaisi nykyisessä olevan vian, joka mahdollistaa kenttien liian monella tapaa ratkaisun. Vaihtoehtoisessa mallissa karsittaisiin kaikkia eri toimintoja, joita nyt on pelaajan liikkumiseen laitettu ja myös vaihtaa välilyönnistä tapahtuvan yksittäisen laatikon työnnön kiipeämiseksi. Tarkoituksena olisi päästä eroon nykyisestä tarpeesta sijoittaa jokin este laatikkopinon taakse voidakseen niiden päälle kiivetä. Tämä nykyinen toimintatapa rajoittaa kenttäsuunnittelua, koska laatikoita pitää asetella seinien viereen. Lisäksi pino, jonka päällä uloskäynti sijaitsee, tulisi tehdä liikkumattomaksi.

Uusia lattiaelementtejä voisi myös lisätä, esimerkiksi jäätä, liukuhihnaa ja teleportaatio-alustaa. Nämä mahdollistaisivat monipuolisempaa kenttäsuunnittelua. Pelissä voisi myös mahdollisesti olla päivittäinen haaste, jossa kenttä on satunnanvaraisesti muodostettu siten, että sen ratkaisemiseen tarvitaan vähintään vaikkapa 20 siirtoa. Pelaajan tekemistä siirroista voisi pitää kirjaa laskurilla ja parhaat ratkaisijat saisivat nimimerkkinsä näkyviin verkkosivulle.

Näillä lisäyksillä saisi enemmän vaihtelevuutta ja syvyyttä pelin kenttäsuunnitteluun. Haastekentillä ja ladattavilla karttapaketeilla saisi pelaajia palaamaan pelin pariin uudelleen senkin jälkeen, kun valmiskenttäpaketit olisivat ratkaistu.

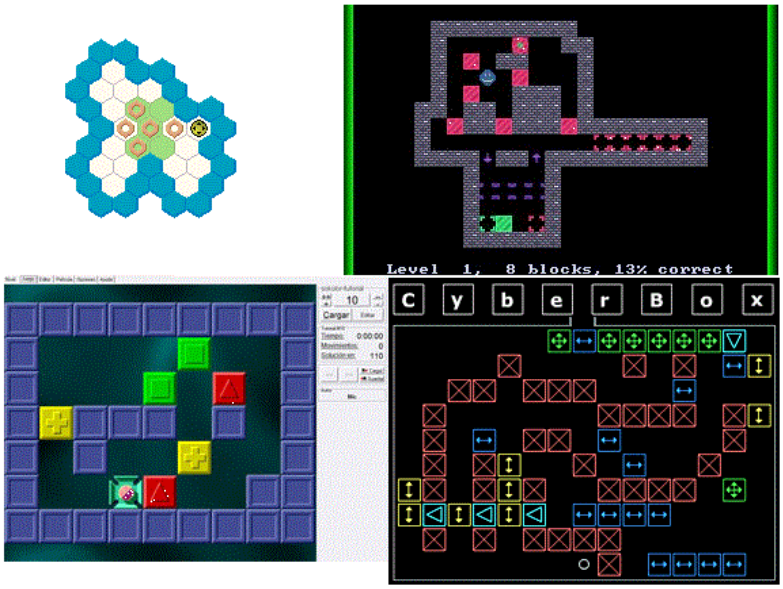
Pelin nykyistä grafiikkaa voisi muokata ammattimaisemman näköiseksi tai sisällyttää useita eri tyylejä, joista pelaaja voisi valikosta valita haluamansa. Nämä eri tyyliset grafiikat voisi myös olla pelin alussa lukittuina, mutta ratkaisemalla kenttiä saisi niitä avattua käyttöönsä. Peliin tulisi myös lisätä musiikkia ja äänitehosteita.

4.2 Olemassa olevat samankaltaiset pelit

Kuten aikaisemmin todettiin, niin laatikoiden työntely eri tavoilla ei ole mikään uusi idea ja vuosien varrella on ilmestynyt lukemattomia erilaisia versioita, joissa on alkuperäisiin Sokobaniin nähden muuteltu peruskenttäasettelua käyttämällä kolmioita (Trioban) tai kuusikulmia (Hexoban) neliömäisten ruutujen sijaan. Joissakin variaatiossa pelaaja voi ohjata useaa eri hahmoa (Multiban, Interlock).

Myös monia erilaisia kentän voittokriteereitä on ilmestynyt vuosien varrella. Joissakin on värikoodattuja laatikoita (Block-o-Mania), jotka tulee saada samanvärisille alustoille, joissakin eriväriset laatikot pitää saada samanvärisen viereen (Sokolor), joissakin on tietty järjestys, jossa laatikot tulee saada tavoiteruutuun.

Päämääräisesti peli muistuttaa eniten CyberBox-nimistä peliä, jossa myös tavoitteena on siirtyä uloskäynnille. Mutta pelinä nämä kaksi ovat varsin erilaisia, sillä Cyberboxissa uloskäynti on reikä kentän laidan seinässä ja liikkuminen tapahtuu vain kahdessa ulottuvuudessa kun taas Boxlessa uloskäynti on jokin kentällä oleva laatikko ja pelissä pystyy myös kiipeämään laatikoiden päälle (Kuva 8). [5.]



Kuva 8. Kuvat peleistä Hexoban, Block-o-Mania, Sokolor ja Cyberbox.

4.3 Muut markkinoilla olevat kilpailijat

Sen sijaan, että peli kilpailisi kaikkien mahdollisten pelien kanssa, keskittyy Boxle vetoamaan puzzle-pelien ystäviin. Jotkin tämän genren alaisuudessa olevista peleistä ei ole millään tasolla vaikeita tai haastavia. Tämä tuli todettua itse suorittamalla katsauksella Appstoresta löytyvien ilmaispelien osalta. Tähän vaikuttaa se, että jos pelissä on vähänkään minkäänlaista ongelmanratkaisua, voidaan se luokitella myös puzzle genreen kuuluvaksi pelin muiden genrejen lisäksi.

4.4 Pelin markkinointi

Nykypäivänä pelejä on markkinoilla valtavasti ja oman pelin erottuminen pelkästään myyntiin pistämällä on onnenkauppaa, ellei peli ole jollain tapaa todella mullistava ja itsenäisesti saa aikaan valtavaa huomiota, kuten Minecraft aikoinaan teki. [6.]

Todennäköisesti valmis peli tulisi jakoon joko pienellä maksulla ostettavaksi tai kokonaan ilmaiseksi. Ilmaisjakelulla tuloja voi koittaa hankkia peliin sisälletyillä mainoksilla, mutta ollakseen oikeasti tuottavaa, tulisi pelin kohderyhmän oltava valtava. [7.]

Toinen nykyään muotiin tullut tapa rahoittaa pelejä on joukkorahoitus, jossa ihmiset vapaaehtoisesti antavat rahaa johonkin projektiin, ja jos projektille määritelty rahasumma tulee kasaan, siirtyyvät rahat projektin tekijöille. Tämäntapaisessa rahankeruussa lahjoittajille yleensä luvataan valmiin tuotteen lisäksi muuta oheismateriaalia riippuen lahjoitussumman suuruudesta. [8.]

Sosiaalisen median käyttö mainontakanavana voisi myös olla yksi vaihtoehto tai lisä. Jos saa kaverin kaverin levittämään tietoa pelistä, niin pikkuhiljaa tieto pelin olemassaolosta saattaa levitä pitkällekin. Etuna tässä tavassa on se, että se on ilmaista.

5 Yhteenveto

Projektin alussa asetetut vaatimukset pelin valmiudesta tätä työtä silmälläpitäen täyttyivät kenttäeditorin, pelin äänien ja musiikin puuttumista lukuun ottamatta. Projektin päätarkoitus oli toimia lisäopetuksena ohjelmoinnissa ja projektin kuluessa on tullut paljon uutta opittua. Varsinkin projektin loppua kohden ajan puutteen vuoksi tehdyt oikotiet saatoivat olla hyvä esimerkki siitä, miten mahdollisesti toimitaan erilaisissa ohjelmointiprojekteissa työelämässä, kun määräaika lähenee.

Jos jatkossa tulee tehtyä uusia pelejä, niin tämän kokemuksen opettamana suunnitteluun tulee panostaa vieläkin enemmän. Tosin seuraavalla kerralla suunnittelu tulee olemaan myös jonkin verran helpompaa, koska nyt tietää enemmän asioita joita voi tulla vastaan.

Kokemus on myös antanut uusia menetelmiä ja ajattelumalleja oman ohjelmoinnin tueksi jatkoa varten.

Lähteet

1 Sokoban History 2014. Verkkodokumentti. Games4Brains. <http://www.games4brains.de/sokoban_history.htm> Luettu: 9.4.2014

2 Code::Blocks 2014. Verkkodokumentti. Code::Blocks. <http://www.codeblocks.org/> Luettu 10.4.2014

3 SFML 2.1 tutorials 2014. Verkkodokumentti. Laurent Gomila. <http://www.sfml-dev.org/tutorials/2.1/> Luettu 10.4.2014

4 Texture atlas 2013. Verkkodokumentti. Wikipedia. <http://en.wikipedia.org/wiki/Texture_atlas> Luettu 10.4.2014

5 Sokoban 2014. Verkkodokumentti. Wikipedia. <http://en.wikipedia.org/wiki/Sokoban> Luettu 11.4.2014

6 Why Minecraft Is So Damn Popular 2011. Verkkodokumentti. Kotaku. <http://kotaku.com/5724989/why-minecraft-is-so-damn-popular> Luettu 11.4.2014

7 Tech Crunch How Free Apps Can Make More Money Than Paid Apps 2012. Verkkodokumenti. Tech Crunch. <http://techcrunch.com/2012/08/26/how-free-apps-can-make-more-money-than-paid-apps/> Luettu 11.4.2014

8 What is Kickstarter 2014. Verkkodokumentti. Kickstarter. <https://www.kickstarter.com/hello?ref=footer> Luettu 11.4.2014