**Mr. Glass 2**

Olio-ohjelmointi 2

Olli Nissinen H8593

Miika Avela H4211

Dokumentointi

12/2015

Ohjelmistotekniikan koulutusohjelma

Tekniikan ja liikenteen ala

Sisällysluettelo

[1 Yleiskuvaus 2](#_Toc437433151)

[1.1 Projektin kuvaus ja toimeksianto 2](#_Toc437433152)

[1.2 Pelin kuvaus 2](#_Toc437433153)

[1.3 Käytetyt teknologiat 2](#_Toc437433154)

[1.4 Tekijät 3](#_Toc437433155)

[2 Työnjako ja vastuualueet 3](#_Toc437433156)

[3 Aikataulu 4](#_Toc437433157)

[3.1 Suunniteltu ajankäyttö 4](#_Toc437433158)

[3.2 Toteutunut ajankäyttö 4](#_Toc437433159)

[4 Ohjelman rakenne ja toiminta 7](#_Toc437433160)

[4.1 UML-kaavio 7](#_Toc437433161)

[4.2 Kentät 8](#_Toc437433162)

[4.3 Pelissä käytetyt tekniikat 11](#_Toc437433163)

[4.4 Pelattavuus ja tapahtumat 12](#_Toc437433164)

[5 Testaus, ongelmakohdat ja niiden ratkaisut 12](#_Toc437433165)

[6 Itsearviointi projektista 12](#_Toc437433166)

# Yleiskuvaus

## Projektin kuvaus ja toimeksianto

Toimeksiannon saimme koulun olio-ohjelmointi opettajalta, Ari Rantalalta. Työn tavoitteena oli perehtyä C++ ohjelmointiin ja oppia mahdollisimman paljon. Aiheenamme on perinteinen 2D tasohyppelypeli, joka on tehty SFML kirjaston avulla. Pelissä hahmo yrittää päästä paikasta A paikkaan B osumatta esteisiin. Peli on suunnattu kohderyhmältään tasohyppelyn ystäville.

## Pelin kuvaus

Mr. Glass2 on C++:lla SFML kirjastoa hyödyntäen tehty perinteinen tasohyppelypeli, missä pelin ideana on pelastaa ihmiset talven viimalta tuomalla heille ikkuna. Reissu ei kuitenkaan suju aivan ongelmitta, sillä matkan varrella sankarimme kohtaa erilaisia vaaroja joista hänen pitää selvitä yhtenä kappaleena. Vaikutteita pelin luonne on saanut vanhoista tasohyppelypeleistä kuten super mario world ja ’n’ The Way of the Ninja.

## Käytetyt teknologiat

Vielä mietintävaiheessa pelin ulkoasua pohdittiin piirtelemällä laatikoita NinjaMock:lla, mutta tämä todettiin nopeasti hankalakäyttöiseksi ja melko rajoittuneeksi ohjelmaksi, joten suunnittelu tapahtui pääosin Paintin ja lehtiön avustuksella. Varsinainen graafinen toteutus suorittui pieniltä osin koulussa Adobe Fireworks CS6:lla, sekä pääosin kotona Adobe Photoshop CS6:lla. Pixel-art –tekniikoiden opiskelussa tietoa haettiin Googlen välityksellä. Luokkakaavion suunnittelussa käytettiin heti Violet UML editoria. Kaikki ohjelmointi ja testaus tapahtui VisualStudio 2015:sta. Projektinhallinta ja -seuranta toteutettiin Githubin projektikansiolla, mitä päivitettiin jokaisten muutosten jälkeen, sekä yhteydenpito samanaikaisesti hoidettiin pääosin Mumble VoiP-ohjelman välityksellä. Myös projektipalavereja pidettiin kasvokkain muutaman kerran viikossa.

## Tekijät

Projektin tekijöinä toimivat kaksi Jyväskylän ammattikorkeakoulun, ohjelmistotekniikan linjan, toisen vuoden opiskelijaa. Miika Avela ja Olli Nissinen. Olio-ohjelmointi 2 kurssille tultaessa kummankin C++:n lähtötaso perustui ohjelmoinnin perusteet kurssiin.

# Työnjako ja vastuualueet

Vastuualueet jaoimme alustavasti suunnitelman jälkeen siten, että molemmille tulisi suunnilleen yhtä paljon tehtävää. Ongelman ilmestyessä miettisimme ratkaisua kuitenkin yhdessä. Alueet jakautuivat seuraavasti:

**Olli:**

Collision, tilemap, objektit, vektorit ja niiden käsittely, dynaaminen muistin varaus.

**Miika:**

Valikko, liikkuminen, hyppy, gravitaatio, poikkeusten hallinta, tiedostoonkirjoitus.

**Yhteistä:**

Pixel Art, dokumentointi, UML-kaavio, peli-idean kehittäminen, kenttäsuunnittelu, piirtopinta, tiedostonluku.

# Aikataulu

## Suunniteltu ajankäyttö

Lähtökohtaisena ajankäytön suunnitelmana oli käyttää tunteja seuraavasti:

Ideointi ja Suunnitelman teko 5h

Graaffinen suunnittelu 10h

Pixel art 15h

Ohjelmointi 75h

Testaus 10h

Dokumentaatio 10h

**yhteensä 125h**

## Toteutunut ajankäyttö

Todellinen ajankäyttö ja projektin edistyminen.

**Viikko 48**

Aiheeksi valittiin 2D tasohyppelypeli. Pelille suunniteltiin tarina ja hyvin raaka luokkakaavio.

Olli: Pohjustus 2h, Ideankehittäminen 2h, Collision 4h, Tilemap 3h, Pelaaja luokka 1h, Dokumentaatio 1h, pixel art 3h, testaus 1h, SFML-kirjastoon tutustumista 2h.

Miika: Projekti ympäristön pystytys 1h, Pohjustus 2h, Ideankehittäminen 2h, Pelaaja luokan liikkuminen kellon mukaan 1h, pixel art 3, SFML-kirjastoon tutustumista 2h.

**Viikko 49**

Olli: Dokumentaatio 1h, Ongelman ratkaisua 3h, Objektit 2h, Objektien perintä samaan vektori taulukkoon, piirto ja collision vektori taulukosta 5h, pixel art 3h, testaus 2h.

Miika: Painovoima 1h, Hyppy 2h, ongelman ratkaisua 4h, Pelaajan kääntyvyys, hajoaminen 3h, Exception Handler 2h, pixel art 3h, testaus 1h.

**Viikko 50**

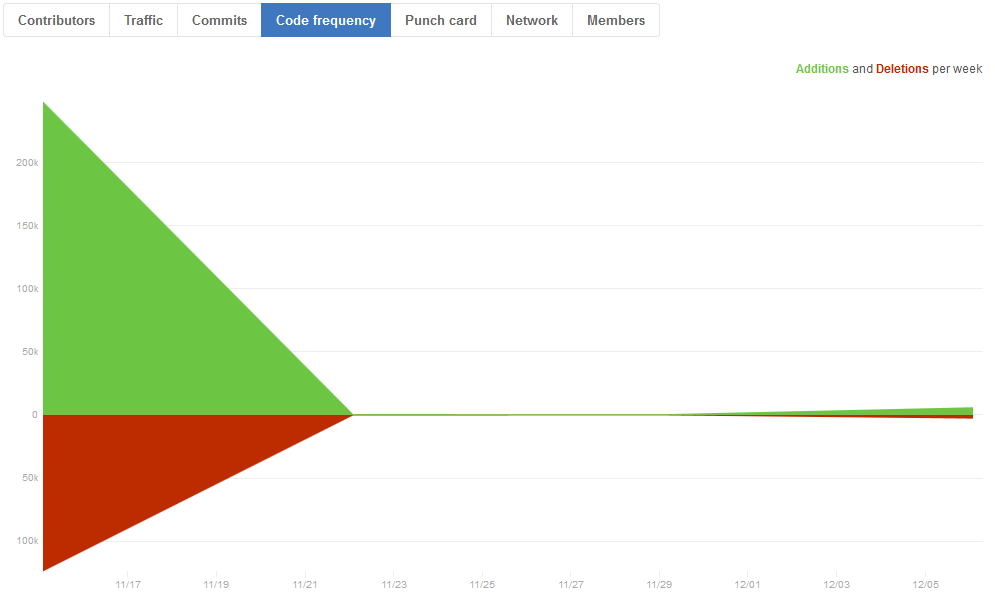
Olli: Ongelman ratkaisua 6h, dokumentaatio 5h, pixel art 1h, testaus 1h, viimeistelyä ja koodin siistimistä 3h.

Miika: Testaus 3h, dokumentaatio 2h, pixel art 2h, Level2, levelinvaihto, läpihyppy 4h, pistelaskuri 1h, tiedostoon tallennus ja luku 2h viimeistelyä 3h.

**Yhteensä: 95h**

Olli: 51h

Miika: 44h

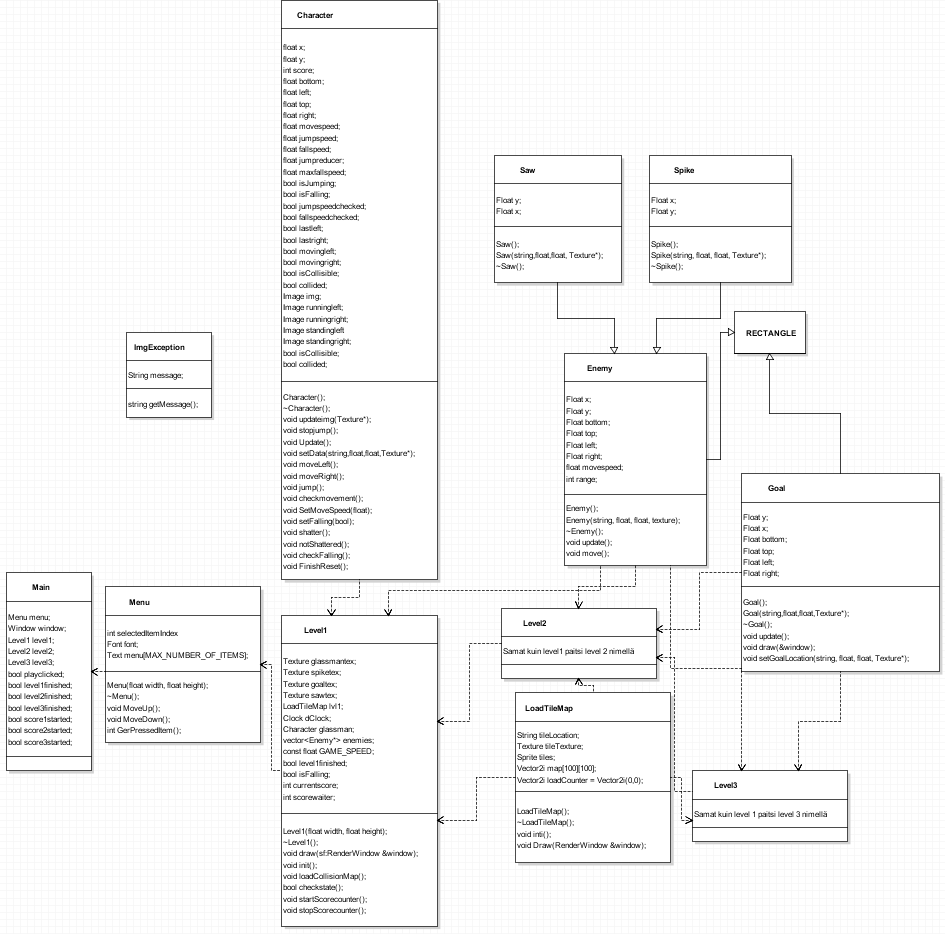
  


Taulukko 1: Github projektin edistyminen

Kuten kaaviosta voidaan todeta, projektia on päivitetty tasaisesti noin kolmen viikon ajan. Projektin alkuvaiheilla koodin lisäämiset ja poistot olivat huomattavasti suurempia projektin alustustöiden, kuten SFML-kirjaston lisäämisen, sekä pelin perusrakenteita pystyttäessä. Projektin edetessä keskityttiin pienempiin kokonaisuuksiin ja siksi koodimäärätkin kutistuivat jatkuvasti.

# Ohjelman rakenne ja toiminta

## UML-kaavio

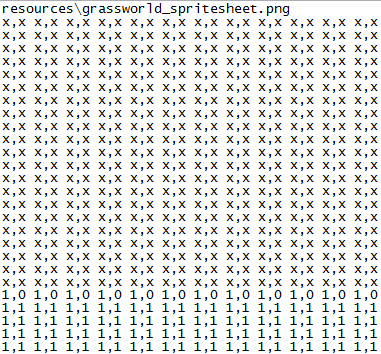


Kuva 1: UML-kaavio

Kuvan 1 UML-kaaviosta saa hyvän käsityksen, kuinka ohjelma kokonaisuudessaan rakentuu.

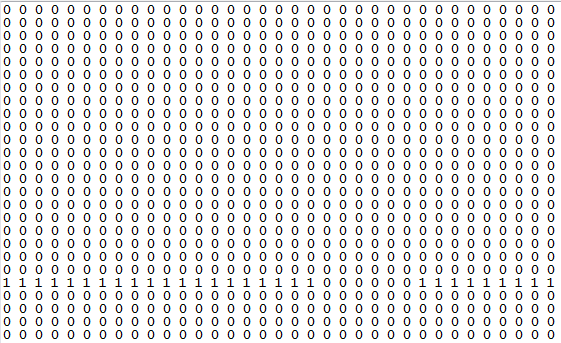
## Kentät

Alkuperäinen idea oli tehdä useampi kenttä, mutta teimme ajan puitteissa vain kolme. Kenttiä tosin olisi hyvin nopea tuottaa sillä kenttien ja collisionin tuotto tapahtuu tekstitiedostoista lukuina.



Kuva 2: TileMap

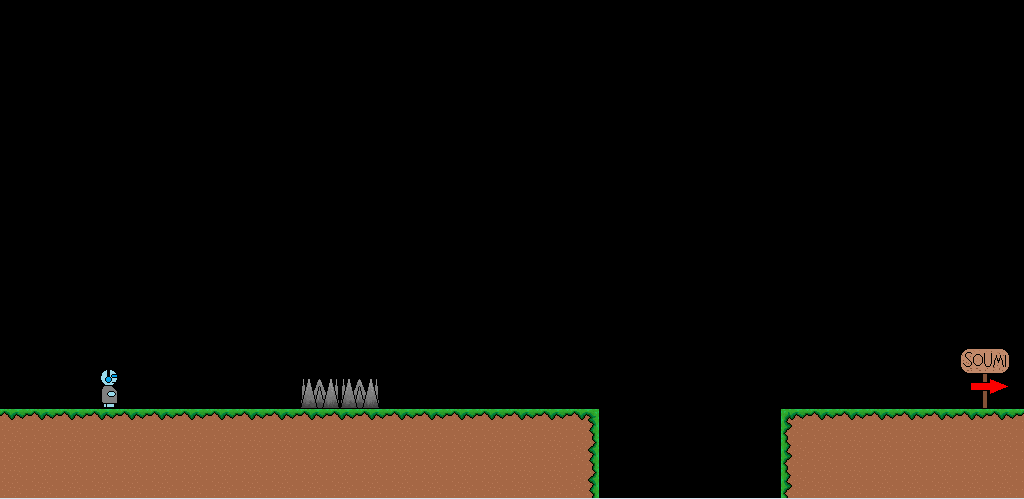
Tilemapissa luetaan ensiksi spritesheet josta tilet luetaan. x,x vastaa tyhjää ja numeeriset arvot kertovat sijainnin spritesheetissä.



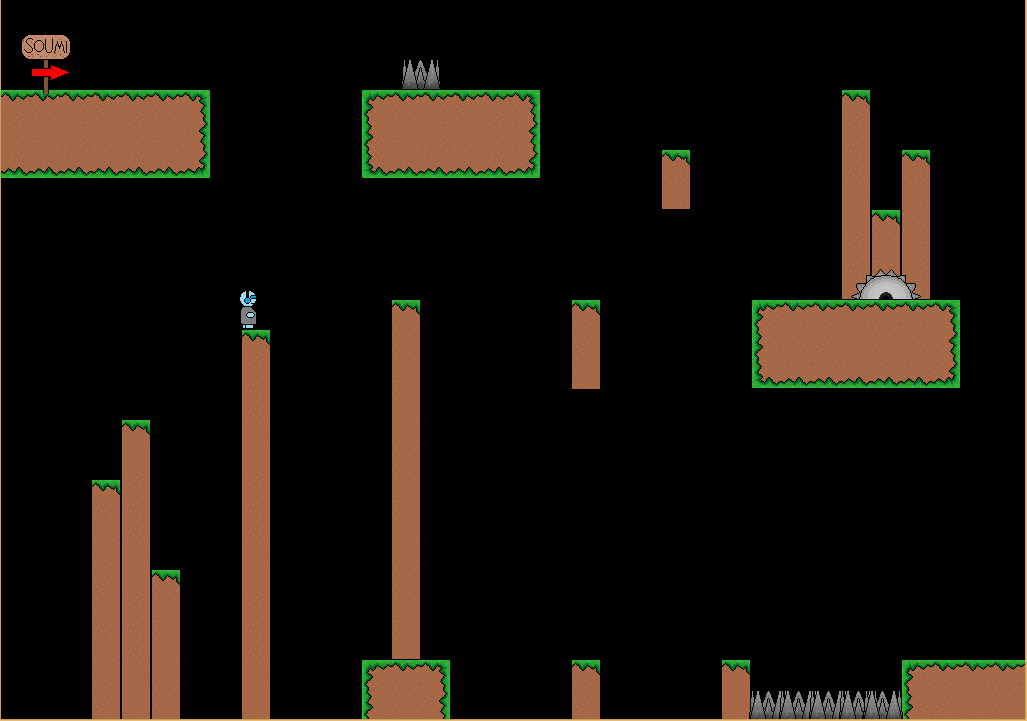
Kuva 3:CollisionMap

CollisionMapin idea on hyvin yksinkertainen. Nolla kuvastaa aluetta jossa voidaan liikkua ja ykkönen on objekti jonka läpi ei voi mennä. Tässä tapauksessa kyseessä on maapalikka.

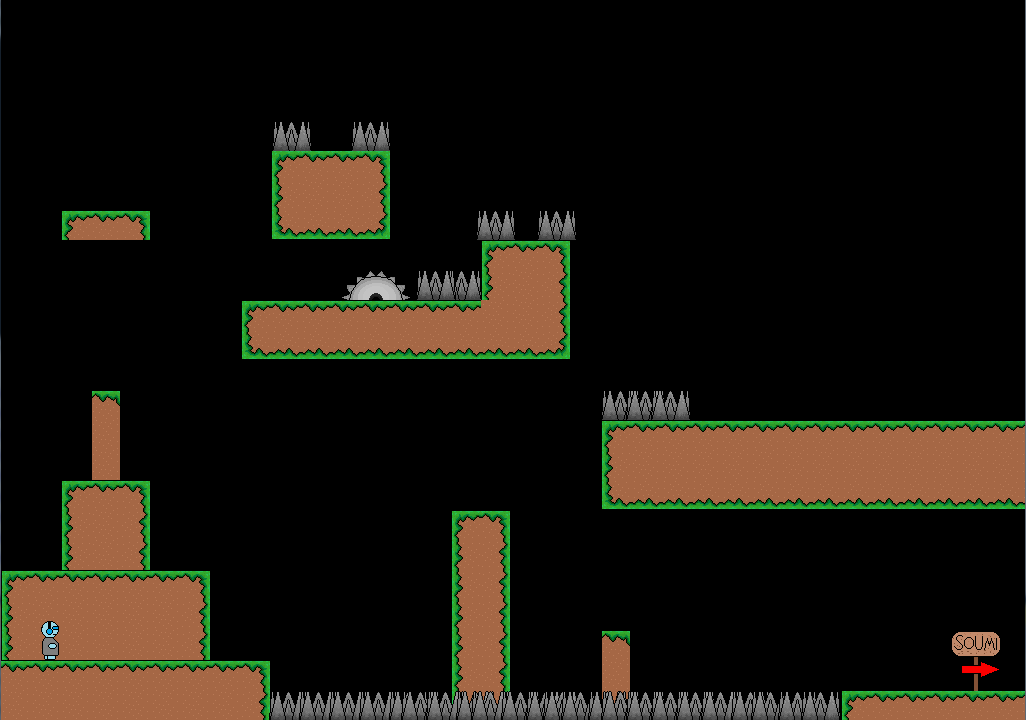
Ohessa muutama kuva pelimaailmasta.



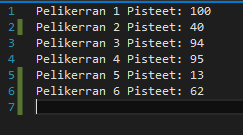
Kuva 4: Level 1



Kuva 5: Level 2



Kuva 6: Level 3



Kuva 7: Pelikertojen pisteet

Pelin pisteet muodostuvat resetointimäärien perusteella, ja laskevat aina kun pelaaja joutuu aloittamaan kentän alusta. Pelaajan läpäistessä pelin, gamescores.dat –tiedosto avataan ja sieltä tarkistetaan kuinka monta kertaa peli on pelattu läpi, ja kirjataan uusi tulos listan loppuun.

## Pelissä käytetyt tekniikat

Pelissä toteutuu seuraavat kurssilla vaadittavat asiat:

* Dynaaminen muistinvaraus
* Pointterit, vektorit, iteraattori ja pointterivektorit
* Tiedoston luku ja kirjoitus
* Poikkeustenkäsittely
* Oliot ja niiden rakenne
* Ulkoinen kirjasto (SFML)
* Perintä
* Taulukointi

Muuta:

* CollisionMap, TileMap, SpriteSheet
* Hyppy ja gravitaatio
* PixelArt

## Pelattavuus ja tapahtumat

Päätimme panostaa enemmän koodin monimuotoisuuteen, kuin itse pelattavuuteen. Pelattavuus jäi hieman helpoksi, mutta olisi korjattavissa haastavammilla kentillä ja hyökkäyksellä, joka olikin alun perin tarkoitus tehdä. Itse pelaajan kääntyminen tapahtuu tarkistamalla liikkeen muuttujat ja päivittää tekstuurit sen mukaisesti. Jos pelaajan latauksessa tapahtuu virheitä, niin poikkeusten käsittelijä ilmoittaa tapahtumasta. Hypättäessä painovoima alkaa vaikuttamaan hyppynopeuteen pelaajan irrotessa maasta. Hypystä ja painovoimasta yritettiin tehdä mahdollisimman oikeankaltaisia, tässä hyvin onnistuen. Collisionissa haimme vanhan kunnon Super Marion tuntua, jossa vain palikan yläosa vaikuttaa pelaajan liikkeisiin.

# Testaus, ongelmakohdat ja niiden ratkaisut

Vaikka testasimme peliä aina kun lisäsimme siihen jotain uutta, ei ongelmakohdilta vältytty. Suurimmaksi ongelmaksi muodostui tilemap collision, joka otti collisionin myös palikan sisältä. Ongelma korjaantui liikuttamalla collisionin ”rajapintoja” muutamalla pixelillä. Pienempänä ongelmana oli liikkumisen aiheuttama collisionin tarkistus, mikä tiputti pelinopeutta huomattavasti. Asia korjaantui ehtolauseita muokkaamalla ja siistimällä. Toinen isompi ongelma muodostui, kun pointteri oliota pistettiin dynaamisesti push\_back komennolla vektoriin, probleemaa pohdittiin pitkään myös opettajan kanssa. Itse olioiden sijoittaminen onnistui hyvin, mutta niiden piirto näytölle tuotti ongelmia. Ratkaisu oli inhimillisessä unohduksessa sijoittaa parametri paikoilleen ja korjaantui helposti sen löydyttyä. Painovoiman ja hypyn kanssa jouduimme vaihtamaan kellon staattiseksi arvoksi, koska sen arvot vaihtelivat 0,3 ja 0,6 välillä. Varsinkin projektin loppuvaiheessa testasimme peliä jatkuvasti, jotta löytäisimme mahdolliset virheet.

# Itsearviointi projektista

Mielestämme projekti on ollut hyvin opettavainen ja mielenkiintoinen kokonaisuus. Tunnilla tehdyt asiat ovat olleet oikein hyviä ja hyödyllisiä esimerkkejä. Silti suuremman kokonaisuuden hallitseminen on haastavaa ilman aukotonta suunnitelmaa. Projektin edetessä olemme huomanneet UML-kaavion tärkeyden ja kuinka se helpottaa koodin kirjoittamista, sekä projektin toiminnallisuuden ymmärtämistä.

Lopputulokseen olemme tyytyväisiä, sillä aikaa projektin tekemiseen oli hyvin rajallisesti rankan syksyn takia, silti onnistuen käyttämään monipuolisesti kurssilla käytyjä asioita. Tulevaisuudessa suunnittelimme tekevämme useamman kentän ja hiovamme pelillisiä ominaisuuksia, jotta tästä paketista tulisi laajempi kokonaisuus. Mainittakoon lopuksi, että kaikki pelissä käytetyt grafiikat ovat itse tehtyjä ja suunniteltuja.