**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](#_Toc437276992)

[1.1 Projektin kuvaus ja toimeksianto 2](#_Toc437276993)

[1.2 Pelin kuvaus 2](#_Toc437276994)

[1.3 Käytetyt teknologiat 2](#_Toc437276995)

[1.4 Tekijät 3](#_Toc437276996)

[2 Työnjako ja vastuualueet 3](#_Toc437276997)

[3 Aikataulu 4](#_Toc437276998)

[3.1 Suunniteltu ajankäyttö 4](#_Toc437276999)

[3.2 Toteutunut ajankäyttö 4](#_Toc437277000)

[4 Ohjelman rakenne ja toiminta 6](#_Toc437277001)

[4.1 UML-kaavio 6](#_Toc437277002)

[4.2 Kentät 6](#_Toc437277003)

[4.3 Pelattavuus ja tapahtumat 8](#_Toc437277004)

[5 Testaus, ongelmakohdat ja niiden ratkaisut 8](#_Toc437277005)

[6 Itsearviointi projektista 8](#_Toc437277006)

# Yleiskuvaus

## Projektin kuvaus ja toimeksianto

Toimeksiannon saimme koulun olio-ohjelmointi opettajalta, Ari Rantalalta. Työn tavoitteena oli tutustua C++ ohjelmointiin ja oppia mahdollisimman paljon 2D pelin tekemisestä SFML kirjastolla. Aiheenamme on perinteinen 2D tasohyppelypeli, missä hahmo yrittää päästä paikasta A paikkaan B osumatta esteisiin. Peli on suunnattu kohderyhmältään tasohyppelyn ystäville.

## Pelin kuvaus

Mr. Glass2 on C++ tehty perinteinen tasohyppelypeli, missä pelin ideana on pelastaa ihmiset talven viimalta tuomalla heille ikkunan. 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.

**Miika:**

Valikot, liikkuminen, hyppy, gravitaatio, poikkeusten hallinta.

**Yhteistä:**

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

# Aikataulu

## Suunniteltu ajankäyttö

Lähtökohtaisena ajankäytön suunnitelmana oli käyttää tunteja seuraavasti:

Ideointi ja Suunnitelman teko 5h

Graaffinen suunnittelu 15h

Pixel art 15h

Ohjelmointi 75h

Testaus 10h

Dokumentaatio 5h

**yht 125h**

## Toteutunut ajankäyttö

Todellinen ajankäyttö ja työn edistyminen. Pelin testaus on sisällytetty ohjelmointiin:

**Viikko 48**

Aiheeksi valittiin 2D tasohyppelypeli. Suunniteltiin pelille tarina ja alustava luokkakaavio.

Olli: Pohjustus 2h, Ideankehittäminen 2h, Collision 4h, Tilemap 3h, Pelaaja luokka 1h, Dokumentaatio 1h, pixel art 3h.

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

**Viikko 49**

Olli: Dokumentaatio 1h, Bugien korjausta 2h, 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 1h, pixel art 3h, testaus 1h.

**Viikko 50**

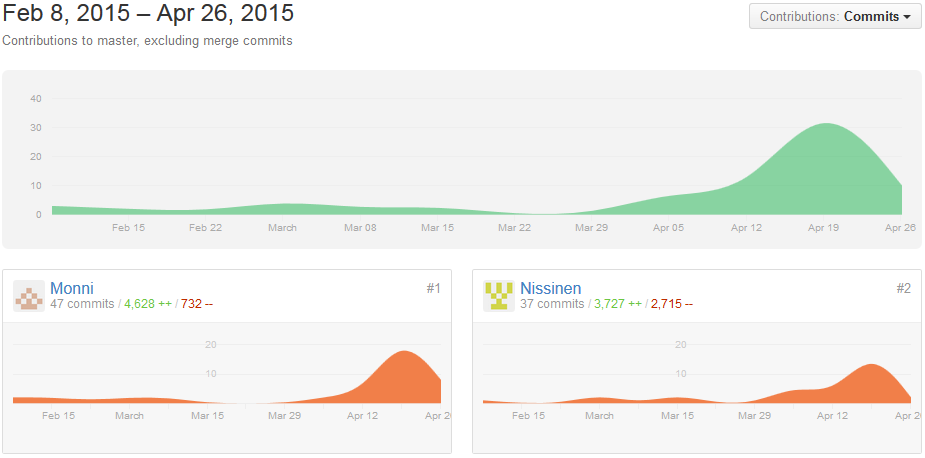
Olli: Ongelman ratkaisua 3h, Dokumentaatio 2h, pixel art 1h.

Miika: Testaus 1h, Dokumentaatio 2h, pixel art 2h.

**Yhteensä:**

Olli: 36h

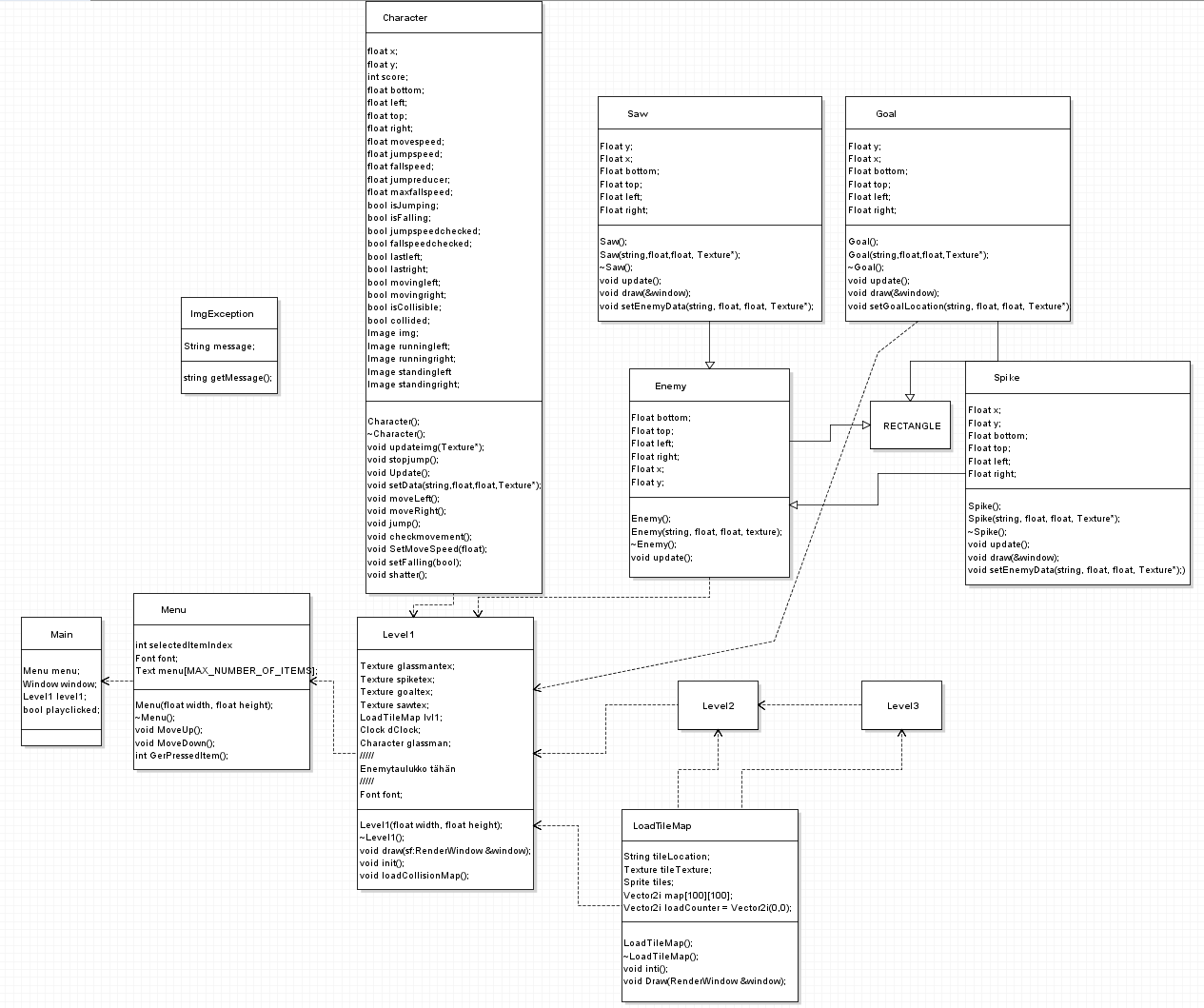
Miika: 26h



Taulukko 1: Github projektin edistyminen

# Ohjelman rakenne ja toiminta

## UML-kaavio

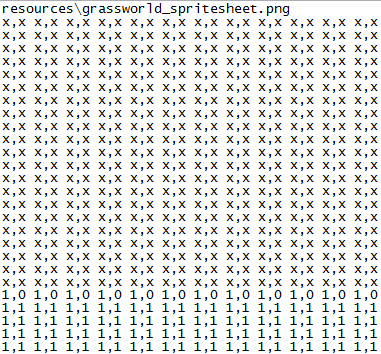


Kuva 1: UML-kaavio

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

## Kentät

Alkuperäinen idea oli tehdä useampi kenttä, mutta teimme ajan puitteissa vain yhden. Kenttiä tosin olisi hyvin nopea tuottaa sillä kenttien ja collisionin tuotto tapahtuu tilemapin avulla 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 objekti jonka läpi ei voi mennä. Tässä tapauksessa kyseessä on maapalikka.

**/#¤/%(#¤”/%#(¤/%”#¤()/%¤”#)(/%()**

**KUVA KENTÄSTÄ TÄHÄN!!!!**

**/(&/)P#¤/%”)#%/”#)&/”#=)¤&/(”#=¤)&/**

## Pelattavuus ja tapahtumat

Päätimme panostaa enemmän koodin monimuotoisuuteen, kuin itse pelattavuuteen. Pelattavuus jäi hieman helpoksi, mutta olisi korjattavissa liikkuvilla objekteilla ja uusilla kentillä. 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.

# Testaus, ongelmakohdat ja niiden ratkaisut

Vaikka testasimme peliä aina kun lisäsimme siihen jotain uutta, ei ongelmakohdilta vältytty. Suurimmaksi ongelmaksi muodostui tilemap collision, joka otti useamman collisionin samaan aikaan. 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. Itse olioiden sijoittaminen onnistui hyvin, mutta niiden piirto näytölle tuotti ongelmia, sillä piirto tapahtui täysin väärässä paikassa. 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. 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.