Olio-ohjelmointi 2 harjoitustyö loppuraportti

Ryhmä

Tuomas Kyttä H3408

Arttu Heinonen H3425

Joonas Katainen H3248

Joulukuu 2014

Ohjelmistotekniikan koulutusohjelma

Tekniikan ja liikenteen ala

Sisältö

[1 Johdanto 2](#_Toc405912721)

[2 Suunnitelma 2](#_Toc405912722)

[2.1 Lähtö suunnitelma 2](#_Toc405912723)

[2.2 Työnjako suunnitelma ja tavoitteet 3](#_Toc405912724)

[2.3 Toteutuksen suunnitelma 3](#_Toc405912725)

[3 Toteutus 4](#_Toc405912726)

[3.1 Toteutus tilakoneesta ja pelimoottorista 4](#_Toc405912727)

[3.2 Toteutus kartasta 8](#_Toc405912728)

[3.3 Toteutus hahmoista ja vihollisista 11](#_Toc405912729)

[4 Aikataulutus ja työn jako 11](#_Toc405912730)

[4.1 Tuomas Kyttä 12](#_Toc405912731)

[4.2 Joonas Katainen 13](#_Toc405912732)

[4.3 Arttu Heinonen 13](#_Toc405912733)

[5 Yhteenveto ja itsearviointi 13](#_Toc405912734)

[6 Viitteet 13](#_Toc405912735)

# Johdanto

Tämän kurssin harjoitustyönä toteutimme ja suunnittelimme 2 ulotteisen tasohyppelyvideopelin. Ryhmässä toimi kolme jäsentä, joille kullekin oli asetettu tehtäväksi ohjelmoida oma osuus peliä. Nämä pelin osuudet sitten koottiin yhdeksi kokonaisuudeksi, jonka kehitystä jatkettiin ryhmässä. Toteutuksessa käytimme SFML multimedia kirjastoa ja Githubin projektihallinta työkalua.

# Suunnitelma

## Lähtö suunnitelma

Aloitimme suunnittelun jo varhain syksyllä ja aloimme keräämään ideoita projektin toteuttamiseen. Melkein välittömästi kaikki ryhmän jäsenet tulivat tulokseen, että tekisimme harjoitustyö projektiksi pelin. Jatkoimme pohdintaa millaista peliä alkaisimme kehittämään ja kokosimme ajatuksia videopelistä, mikä olisi suhteellisen mahdollista toteuttaa annetuilla resursseilla ja mikä meitä ryhmänä kiinnostaisi tehdä. Päädyimme lopulta ideaan 2D tasoloikka pelistä, jossa maasto ja viholliset luotaisiin satunnaisesti. Pelaajalla olisi myös mahdollisuus ampua vihollisia ja kerätä satunnaisesti putoavia voimavaroja, joilla pelaajasta tulisi tehokkaampi. Kokonaisen valmiin pelin saaminen valmiiksi kuulosti toki isolta ja työläältä urakalta, mutta halusimme kokeilla rajojamme ohjelmisto kehittäjinä ja testata kuinka pitkälle pystyisimme kehittämään tätä projektia. Pitkin syksyä heittelimme toisillemme ideoita pelimekaanikoista ja teknisen puolen toteutuksista, joista pikku hiljaa alkoi hahmottua selkeämpi ja selkeämpi kuva siitä mitä meidän oli aikomus tehdä.

## Työnjako suunnitelma ja tavoitteet

Kaikki olivat yhtä mieltä siitä, että olisi kaikille mukavampaa, jos jokainen saisi oman itsenäisen osan ohjelmasta, jota voisi kehittää. Että jokaisella olisi oma pieni ”nurkka” koodista jonka parissa kukin pystyisi työskentelemään, kunnes lopulta osat kasattaisiin yhteen ja niistä luotaisiin peli kokonaisuus. Työnjaossa sovittiin, että Tuomas Kyttä tekisi karttojen luomisen, tallentamisen ja satunnaisen generoinnin pelimaailmaan pelaajan ja vihollisten ympärille. Arttu Heinonen tekisi pelin rungon, päivityksen, piirron ja pelimoottoria. Joonas Kataisen tehtävänä oli luoda peliin sisältöä, kuten vihollisia ja tavaroita joita poimia. Tarvittaessa muut auttaisivat myös muita. Sovittiin, että kukin sai aloittaa milloin itselle parhaaksi sopi, kunhan vain työt eivät estäisi muiden töitä.

## Toteutuksen suunnitelma

Koska peliin oli suunniteltu monta erilaista tilaa, niin pelin runkoon kuului tehdä toimiva tilakone, joka pitää kaikki pelitilat saman katon alla. Kun ohjelman aluksi käynnistäisi, menisi peli päävalikkotilaan, josta pystyisi vaihtamaan tilaa kartanluontiin, pelitilanteeseen ja takaisin päävalikkoon. Päävalikon alle oli myös suunniteltu asetusvalikko, josta pystyisi vaihtamaan erinäisiä peliin liittyviä asetuksia. Asetukset tallennettaisiin tiedostoon ja luettaisiin aina pelin avautuessa uudelleen.

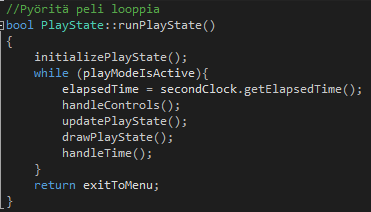
Kartanluontiin suunniteltiin käytettäväksi satunnaista kartanluontia ennalta määrätyistä karttapaloista. Nämä karttapalat piti erikseen rakentaa ja siksi sen helpottamiseksi tehtiinkin karttapalaeditori. Karttapalaeditorissa pystyisi lisäämään karttapaloihin ennalta määrättyjä karttaobjekteja.

Vihollisiin ja hahmoihin suunniteltiin, että luodaan pelaajalle ja vihollisille yhteinen kantaluokka, jonka molemmat perisivät. Hahmot päivitettäisiin ja piirrettäisiin kaikki erillisesti ja ne tarkastettaisiin törmäiltävän maaston kanssa. Pelaajalle uniikkeja ominaisuuksia olisi mm. pelaajan ohjaaminen näppäimistöllä. Vihollislitania tallennettaisiin linkitettyyn listaan ja käsiteltäisiin sieltä.

# Toteutus

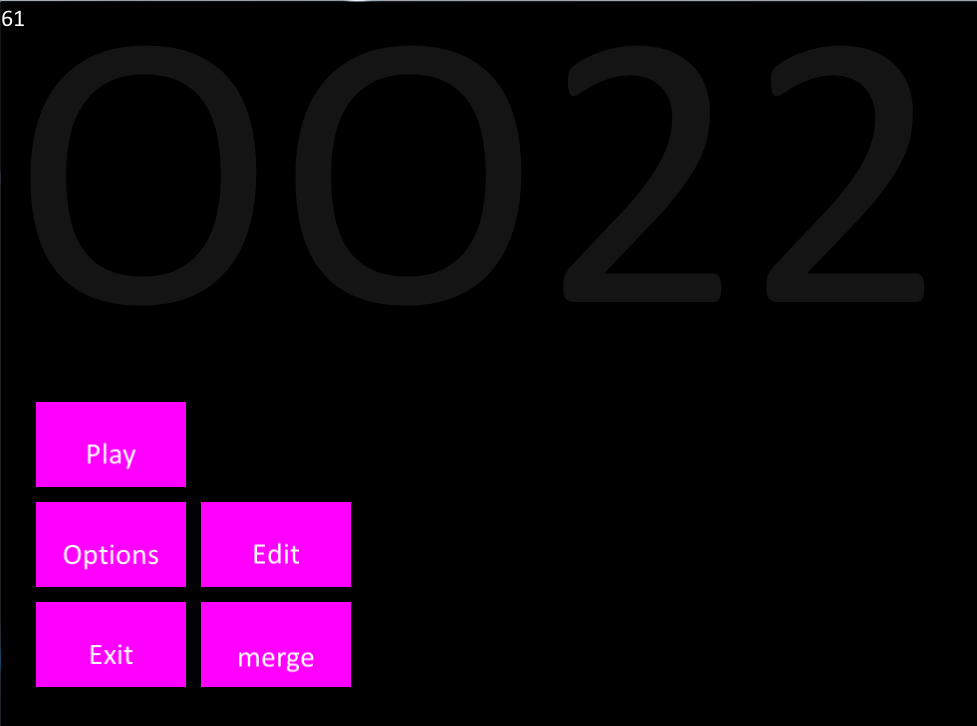
## Toteutus tilakoneesta ja pelimoottorista

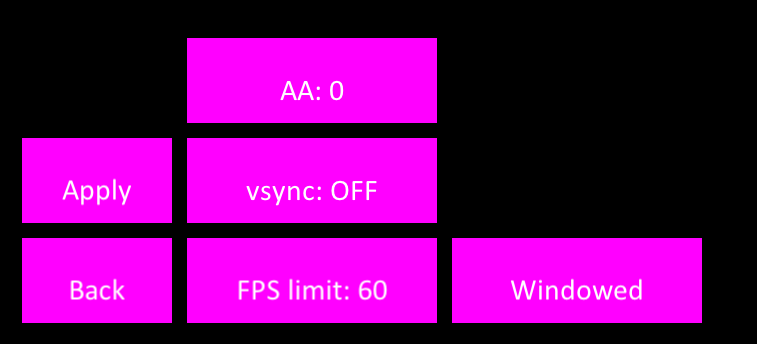
Pelimoottorin ydin on tilakone. Tilakone hallitsee mitä tilaa pelissä pyöritetään ja vaihtaa sitä switch case:lla, kunnes tulee tila käsky lopettaa ohjelma. Tiloista itsessään olisi voinut tehdä virtuaalisen luokan, josta olisi voitu luoda muut tilat, mutta ajan puutteen ja käytännöllisen hyödyn olemattomuus jätti sen tekemisen pois. Ohjelman tilojen pääidea on kiteytettynä seuraavanlainen:



Eli, tila alustetaan, jonka jälkeen siirrytään tilan pyöritys looppiin. Loopissa katsotaan kauan viime looppiin meni aikaa, käsitellään näppäimet, päivitetään tila, piirretään tila ja käsitellään aika.

Päävalikkotila on ensimmäinen tilakoneen tuottama tila, joka aukeaa käynnistäessä. Päävalikon alussa alustetaan valikkojen napit ja ladataan käyttäjän asetukset asetustietueeseen. Jos asetuksia ei ole, ne luodaan. Asetusten määritykset vaikuttavat pelin suoritukseen. Esimerkiksi anti aliasing vaikuttaa kuinka pehmeältä kuvat näyttävät. Päävalikon päivityksessä kuunnellaan lähinnä hiirtä, ja sitä painetaanko ruudulla näkyviä nappeja. Jos Esc painiketta tai ruudun raksia painaa, niin tilakone siirtyy ”exit” tilaan, jolla ohjelma sulkeutuu. Päivityksessä säädetään myös otsikon näkyvyyttä hieman näkyvämmäksi, kunnes se on täysin näkyvä.

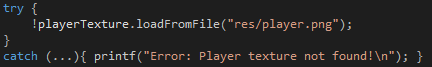




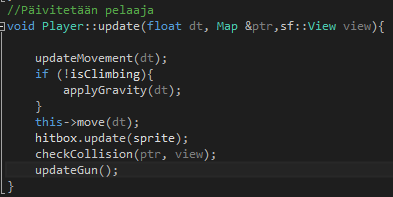
Oikeassa yläkulmassa näkyy fps lukema, jonka saa näkyviin/piiloon F1 näppäimellä. Siirtyessä Play, Edit tai Merge tilaan ladataan kartta tiedostosta, jossa kestää tovi. Tätä varten ruudulle piirretään latausruutu.

Pelitilassa ladataan kartta, pelaaja ja generoidaan viholliset. Vihollisten generointi toimii satunnaisesti kartan koon mukaan. Viholliset generoidaan linkitettyyn listaan, josta ne myös päivitetään ja piirretään.

Pelaajan luonnin yhteydessä ladataan sprite muuttujaan Content luokasta tekstuuri. Jos pelaajan tekstuuria ei syystä tai toisesta löydy, niin heitetään konsoliin virheilmoitus. Pelaajan tekstuurin puuttuminen itsessään ei ole kriittistä ohjelman suorittamiselle, mutta pelaajan tilalle tulee vain valkoinen suorakulmio.



Pelaajalle myös luodaan gun olio, joka hoitaa pelaajan ampumiskäden. Nykyisessä versiossa sen käyttö on tosin jouduttu aika syistä karsimaan. Pelaaja myös periytyy Character luokasta, jossa hoidetaan liikkuminen, painovoima ja piirtäminen.

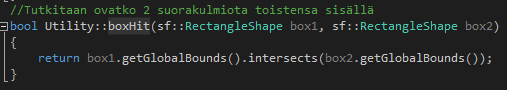


Pelaajan päivityksessä hoidetaan sen liikkuminen, painovoima, Hitboxien seuraaminen pelaajaa, törmäystarkastelu ja aseen erillinen päivitys. Liukuluku arvo dt edustaa delta-aikaa. Aikaa, kuinka suuri eri ruudunpäivitystahdilla on. Ilman tätä arvoa ruudunpiirtonopeus vaikuttaisi liikkumisnopeuteen ja sitä emme halua. Metodeissa kummittelee myös Map tyyppinen pointteri kartan paloihin. Tätä tietoa tarvitaan, kun tutkitaan osuuko pelaaja kartanpaloissa oleviin objekteihin.

Pelaajan törmäystarkastelu onkin hieman hankalampi ja olisi varmasti pystytty toteuttamaan viisaammin. Periaate on seuraavanlainen:

* Asetetaan törmäysarvot epätosiksi
* Tutkitaan pelaajan nykyinen karttasektori
* Loopataan pelaajan ympärillä olevat karttasektorit, jos sektori on sallittujen rajojen sisällä
* Käydään läpi sektorin jokainen pala
* Jos kartan objektiin voi törmätä, niin tarkistetaan törmättävät objektit pelaajan kaikkien Hitboxien kanssa
* Jos kyseessä ei ole törmättävä objekti, vaan esm. tikkaat ja pelaaja koskee siihen, annetaan mahdollisuus kiipeämiseen
* Kun kaikki objektit on käyty, niin korjataan arvoja riippuen mihin törmättiin

Törmäystä tarkastellaan Utility luokassa. Metodiin annetaan kaksi suorakulmaista laatikkoa ja metodissa katsotaan koskevatko laatikot toisiinsa.

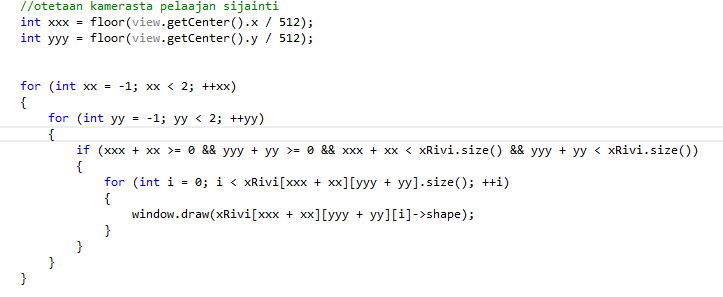


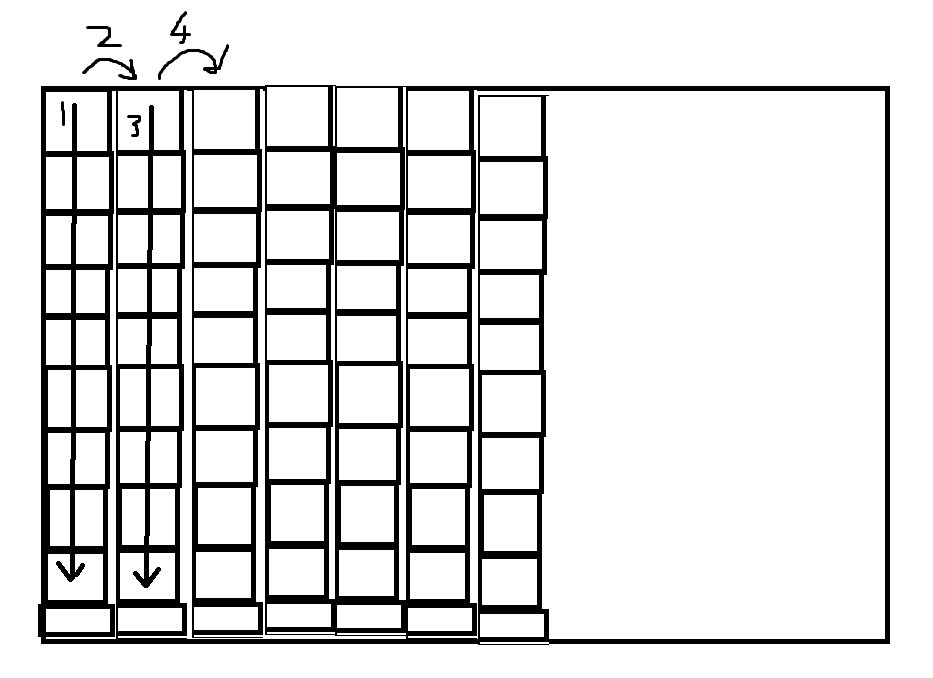
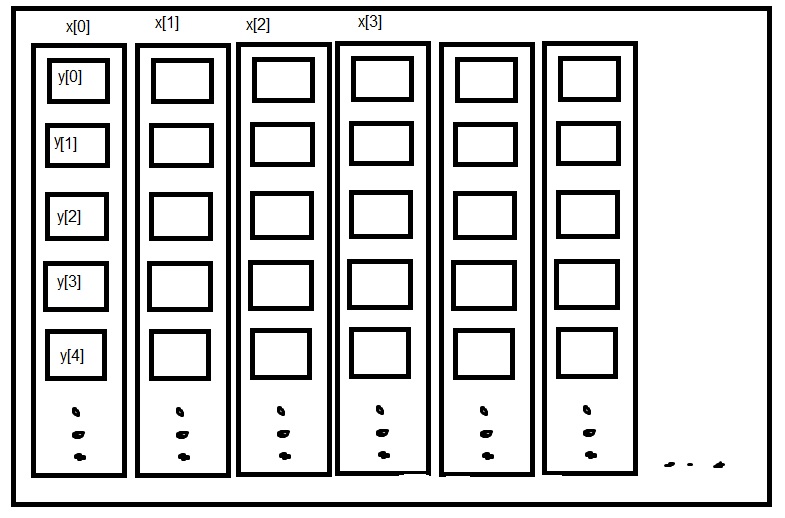
Viholliset käyttävät suurin piirtein samaa logiikkaa.

## Toteutus kartasta

Suunnittelu vaiheessa kartalle oli asetettu seuraavat tavoitteet: kyky muokata kartan paloja ja koota kartta luoduista palasista. Lähdin luomaan editoria. Editoriin loin aluksi ruudukon johon asetettaisiin tulevia objekteja. Tämän toteutin vertex taulukolla joka käytännössä iski pisteitä haluttuihin kohtiin ja pisteiden väliin piirrettiin myöhemmin viivat. Loin nappulat käyttäen luomaani ja jakamaani Button luokkaa. Tallennuksen tein siten että jokainen objekti kirjoitetaan tekstitiedostoon omalle rivilleen nimi, x ja y sijainnit muodossa kansioon Files\mapPartX.txt. Tieto montako palaa on luotu kirjoitetaan Files\fact.txt kansioon ensimmäiselle riville.

Seuraavaksi oli osien niputtaminen yhdeksi kartaksi. Nimesin luokan missä tämä tehtäisiin Mergeksi ja se aloittaa toimintansa lukemalla facts.txt;stä kartan palojen määrän. Tämän jälkeen se lukee kaikki mapPartit rivi kerrallaan, parsii rivin kolmeen osaan: nimi, x ja y. Vertailee nimiä mahdollisiin objekteihin, luo halutun objektin halutulle paikalle ja sijoittaa ne vektoriin. Tämän jälkeen paloja asetellaan satunnaisesti ensin 0,0 -> 0,512 -> 0,512\*2 jne kunnes vaihdetaan x tasoa. Kartan voi sekoittaa uudelleen X:llä ja tallentaa Z:lla.Wasd:lla voi liikuttaa kameraa ja tutkailla onko kartta mieleinen. Olin ensimmäinen projektissa joka tarvitsi kameraa ja opittuani kameran käytön lisäsin sen kylmästi playState luokaan seuraamaan pelaajaa.

Alun perin Mergessä tallentamisen olisi pitänyt olla vain yksi iso lista objekteja ja sijainteja jolloin se oltaisiin voitu lukea suoraan Map luokassa. Tämä on suurin syy miksi mergen tallentaminen ja rakentaminen voivat näyttää kummalliselta mutta vaatimusten muuttuessa oli pakko pistää mutkia suoriksi jossain. Olimme päättäneet jo etukäteen ettei vihollisia tai törmäyksiä laskettaisi tai piirrettäisi mikäli pelaaja olisi liian kaukana. Sain palautetta Kataiselta, että oli liian raskasta isoilla kentillä laskea jokaisen vihollisen etäisyys pelaajaan. Lähdin pohtimaan ratkaisuja ongelmaan ja serkkuni kanssa keksimme jakaa kartan osiin x ja y akseleilla ja laskea indeksit missä akseleilla oltaisiin kamerasta, koska pelaaja on aina kameran keskellä.Tämä oli loistava ratkaisu mutta aiheutti päänvaivaa Mergessä. Mergen tallentamista piti muuttaa siten että Map luokka tietäisi asettaa kartan palaset oikeisiin vektoreihin. Tästä syntyi hirviö missä asetimme objekteja vektorista niiden y ja x akselin suhteen tekstitiedostoon ja aina kun y:n arvo kasvoi yli 512 kirjoitimme "--" merkiksi ruudun vaihtumiselle. Tätä jatkettiin kunnes saavuttiin kentän alalaitaan ja vaihdoimme x akselia ja teimme vastaavat muutokset taas. Tallennetut mapit löytyvät Files\Maps\mapX.txt. Tällä hetkellä ohjelmassa on tietoisesti ominaisuus että ensimmäisellä tallennus kerralla (Z) se ylikirjoittaa vanhan mapin, mutta kun painaa uudelleen niin ohjelma luo uuden mapin, mihin ei tosin pääse mitenkään voittomahdollisuuksien puuttumisen takia. Saatuani tämän toimimaan selitin algoritmin toimintaperiaatteen Heinoselle ja Kataiselle, jotta he voisivat käyttää sitä törmäystarkastelussa.

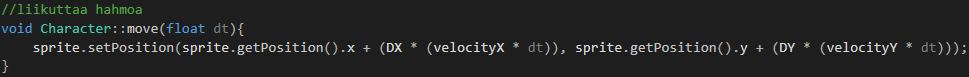
Map luokan oli tarkoitus lukea haluttu map.txt tiedosto ja luoda kenttä sen mukaan. Alussa kerrotaan monesko tiedosto luetaan, kerrotaan sen koko (kartat ovat neliöitä, mahdollista muokata) luetaan tiedostoa rivi kerrallaan. Pätkitään rivi nimeksi, x ja y arvoksi. Jos rivi on "--" tiedetään että yksi kartan pala on luettu. Tämän jälkeen täytetään karttaa seuraavan kuvan osoittamalla tavalla: jokainen pieni ruutu on editorissa tehty kartan pala. Samalla kun kenttää luodaan näin järjestellään luetut palat 3 ulotteiseen vektoriin seuraavasti:

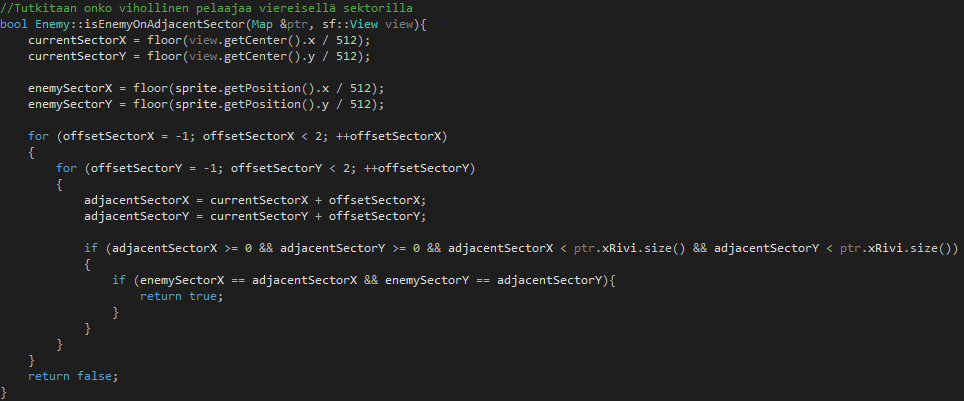
Käytännössä jokainen kuvan laatikko on vektori ja niiden sijainti isommassa laatikossa on kerrottu kuvassa. Tämä kuva todennäköisesti selventää algoritmia jonka kuvan esittelin Mergestä kirjoittaessani. Tämä kuitenkin kevensi pelin resurssi tarvetta ajon aikana. Kartan koko vaikuttaisi ainoastaan luonnin aikana. Kartta vaikutti karulta joten lisäsin siihen taustan ja pilviä joiden sijaintia päivitettiin pelaajan liikkeeseen verrattuna ja ruudun ylittäessään niiden sijainti vaihdetaan toiselle puolelle ruutua lähemmäs pelaajaa.

MapObject luokka on perittävä luokka jonka jokainen objekti perii ja muokkaa perimiään attribuutteja konstruktorissaan. Boolean passable on vain sitä varten, että kollegoitteni työ törmäystarkastelussa olisi hitusen helpompaa.

Tallennetuissa tiedostoissa on seuraavanlaisia ongelmia vielä: facts.txt on pakko mennä muuttamaan jos käyttäjä tuhoaa mapPart.txt tai map.txt tiedostoja ja asetettava luvut siten, että ylin on mapPart tiedostojen määrä ja alempi kenttien määrä. Editorilla on myös mahdollista tehdä mapPart1, mapPart2 ja mapPart5, mikä ei aiheuta ohjelman kaatumista vaan rikkoo kartan piirto logiikan. Tyhjät kentänpalat eivät kerro olemassa olostaan tällä hetkellä kirjoittamalla map.txt:n "-- rivin vaihto --". Tämä tulee vastaan jos käyttäjä haluaa käyttää harvempaa kartan palaa kartan luomisessa kuin on jo luotu.

## Toteutus hahmoista ja vihollisista

Character on luokka, jonka vihollinen ja pelaaja molemmat perivät. Character periytyy SFML-kirjaston RectangleShape luokasta. Characterissa hoidetaan luotujen hahmojen initialisointi, piirto, hitboxien piirto, spriten suunta, painovoima ja liike. Liikkuminen tapahtuu seuraavasti: missä DX ja DY on liikkeen suunta, velocityt nopeus ja dt kulunut aika viime ruudunpäivityksestä.

Vihollisen luonnin yhteydessä sille annetaan randomilla generoitu aloitussijainti ja tyyppi, jonka perusteella viholliselle määritellään tekstuuri ja käyttäytyminen. Vihollisen käyttäytyminen tapahtuu vertailemalla pelaajan sijaintia vihollisen sijaintiin ja reagoiden vihollisen tyypille ominaisella tavalla. Vihollisen päivittyessä, se tarkistaa onko se tarpeeksi lähellä pelaajaa, jos vihollinen ei ole, se ei päivity ollenkaan, säästäen resursseja. 

Vihollinen käyttää Characterista perittyjä metodeja samoin kuin pelaaja. Kuten pelaajan päivityksessä, vihollisen päivityksessä hoidetaan sen käyttäytyminen, liikkuminen, painovoima, hitboxien seuraaminen ja törmäystarkastelu, joka noudattaa pelaajan vastaavaa tarkastelua, lisättynä pelaajan ja vihollisen törmäys. Vihollisen törmäystarkastelusta puuttuu myös viereisten sektorien tutkiminen, koska vihollisen läheisyys todetaan jo päivityksen alussa.

# Aikataulutus ja työn jako

<https://github.com/JamkOlioOhjelmointi2HarkkaTiimiHKK/oo22/commits/master?page=1>

Oheisessa linkissä näkyy kunkin työnjako ja päivityshistoria projektille.

Projekti tehtiin GitHubia käyttäen, jonka ansiosta kaikki projektiin liittyvät lisäykset ja päivitykset näkyvät helposti koottuna commit-historiana. Listasta näkee milloin, mitä ja kuka teki mitäkin. Projektin virallinen työstäminen aloitettiin 23.10.2014. ja lopetettiin 9.12.2014.

## Tuomas Kyttä

|  |  |
| --- | --- |
| vk 43 | Annoin Heinoselle vanhoista projekteistani luokkia mm. Utility, Content, Controls, Button. |
| vk 44 | Loin kilpailevan statehandlerin ja mainmenun. |
| vk 45 | Editorin alku. Ruudukko ja nappulat. |
| vk 46 | Editori ei piirrä valittua objektia ennekuin se on ruudukossa ja tallennus systeemi. |
| vk 47 | Merge luotu ja tallennus ratkaistu. |
| vk 48 | Committasin editorin ja mergen gittiin. Muokkasin muiden koodia jotta omani menisi jatkoksi ja mm. lisäsin extern ikkunan ettei sitä tarvitse heittää kaikkialle parametrina.  Map luokka. Tikkaat, spawni pointit. Mergen viilailuaja kameran lisääminen. |
| vk 49 | Kartan optimointia ja lopullisen piirto algoritmin keksiminen -> paljon koodia uusiksi |
| vk 50 | Tekstuurit kartan objekteille, tausta ja sen pilvet. |

Suunnittelu ja opetteleminen: 14h  
Koodaus(yritykset ja epäonnistumiset): ~40h   
Grafiikka: 0,25h

## Joonas Katainen

|  |  |
| --- | --- |
| vk 46 | Ensimmäisen vihollisen sprite ja pelaajan fysiikan aloitus |
| vk 47 | ------- |
| vk 48 | Vihollisluokan luonti, ensimmäinen vihollistyyppi, hyppyfysiikan viimeistely ja X-akselin liikkeen kiihtyvyys. |
| vk 49 | 3 Uutta vihollistyyppiä, niiden käyttäytymiset, enemy update optimointia, pelaajan fysiikan viimeistely ja enemy törmäystarkastelun aloitus. |
| vk 50 | Enemy törmäystarkastelun viimeistely ja pelaajan törmäys viholliseen tarkastelu. |

Suunnittelu ja pohdinta ~15h  
Opetteleminen ~5h  
Bugfixing ja testaus ~5h  
Koodaus ja epäonnistumisien kiroilu ~30h  
Grafiikka ~1h

## Arttu Heinonen

# Yhteenveto ja itsearviointi

Loppujen lopuksi pelinkehitys jäi erittäin kesken, mutta saimme kuitenkin aikaan paljon ja pelinkehitys onnistui yllättävän pitkälle. Toiveet pelinlopputilasta oli kaikilla hieman saavutettua korkeammat, mutta voimme kaikki olla ylpeitä saavutuksestamme ja opimme projektin aikana paljon asioita. GitHubin käyttäminen auttoi projektin etenemistä merkittävästi ja ilman sitä emme varmaan olisikaan päässeet näin pitkälle. Siitä huolimatta koodin yhdistämisessä tuli moneen otteeseen hankaluuksia ja aiheutti ongelmia usealla taholla.

Projektia olisi auttanut selkeämpi suunnitelma ja paremmat rajapinnat koodeille, joita muut olisivat voineet käyttää. Projekti oli sinänsä opettavainen ja vastaisuudessa käytämme varmasti suunnitteluun enemmän aikaa.

# Viitteet

<https://github.com/JamkOlioOhjelmointi2HarkkaTiimiHKK/oo22>

<http://sfml-dev.org/tutorials/2.1/>