# Jarno Kilander, 305433, TIK, -13, 6.2.2016

## Projektityön dokumentti

## Muumilaakson puolustus -peli

## Yleiskuvaus

Kyseessä on tällä hetkellä suhteellisen suosittu pikkupeliformaatti tower defence, jossa joukko vihollisia pyrkii (yleensä jotakin ennalta määrättyä reittiä pitkin) pääsemään maaliin. Pelaajan pitää estää tämä asettamalla torneja reitin varrelle, jotka ampuvat tai jollain muulla tavalla haittaavat hyökkääjiä pääsemästä maaliin.

Peli sisältää graafisen käyttöliittymän ja itse Photoshopia hyödyntämällä suunnitelin vähän näyttävämpää graafista sisältöä kuin vain neliöitä ja palloja. Tämä vaihe toteutin jo hyvin puolivälissä kehitysprojektiani vaikka sen olin suunnitellut tekeväni vasta loppupuolella. Vaikeusaste jolla olen suorittanut mielestäni projektin, on keskivaikea/vaikea. Sain pelistä suhteellisen näyttävän graafisesti ja moniulotteisen vaikka tosin joitakin graafisia lisäyksiä esim. Ammuksien grafiikka ja animaatiot en saanut mahdutettua projektin teko aikatauluuni. Sain muumeista hauskan idean taustalle kuvittamaan graafisesti peliä.

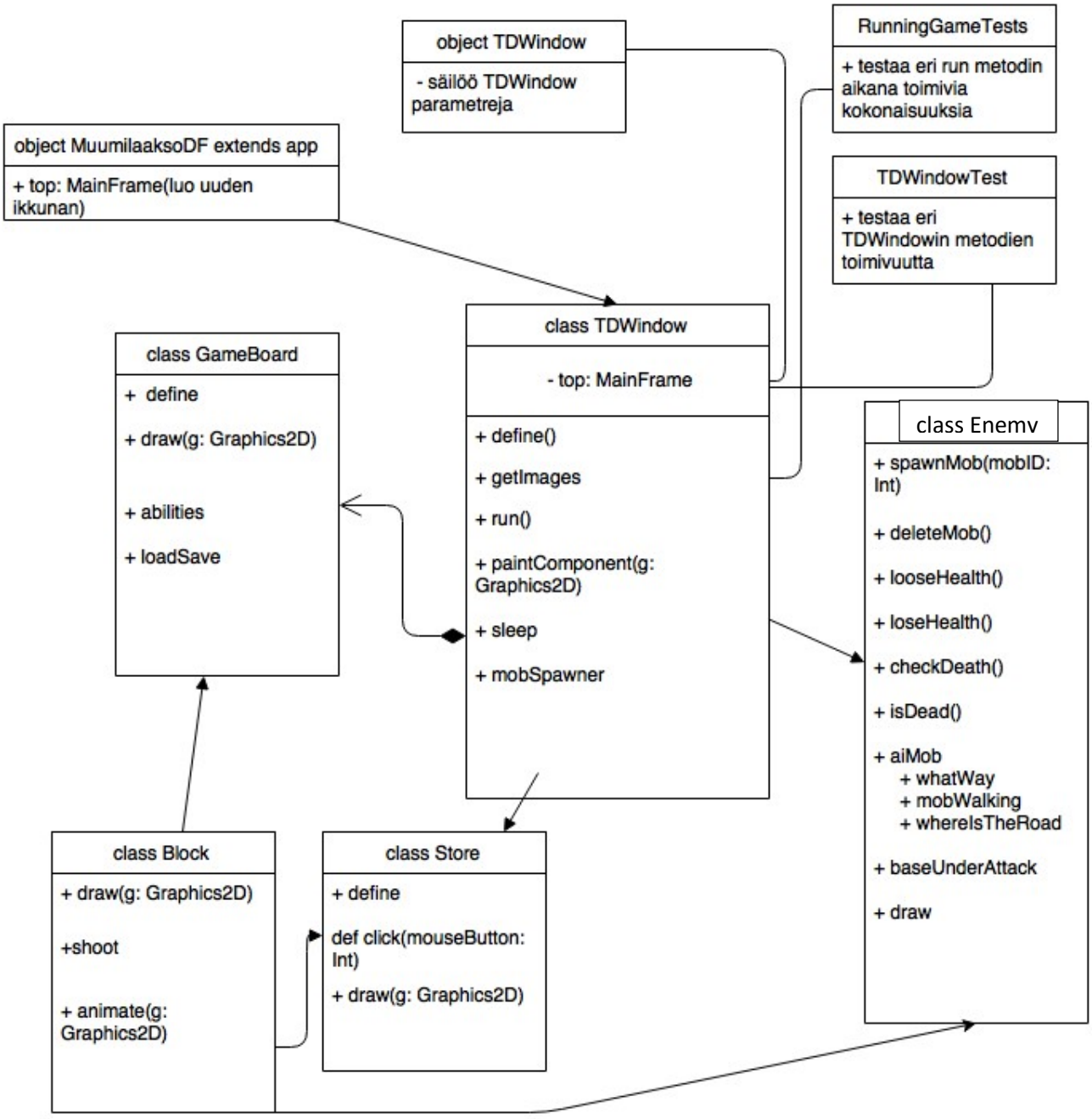
## Käyttöohje

Ohjelma käynnistetään Eclipse Scala ympäristössä ajamalla ohjelma src/towerdefencegame.gui/DF.scala Scala Class: ista komennolla run as -> scala application. Näin ohjelma käynnistyy ja käyttäjän näytölle ilmestyy aloitusikkuna josta käyttäjä voi käynnistää pelin Start – nappulaa hiirellä klikkaamalla. Pelin ohjaus ja komennot tapahtuvat hiirellä siirtelemällä ja klikkailemalla (kumpikin hiiren näppäimistä edustaa tässä pelissä samaa toimintoa tekevää tarkoitusta). Hiirellä voi valita minkä tornin haluaa ostaa ja siirtää reitin varrelle haluamalleen paikalle (kuitenkin niin ettei torni voi päätyä vihollisten kulkuväylälle, tai toisen tornin päälle tai kartan ulkopuolelle). Ostos tapahtuma tapahtuu vasta kun torni on siirretty ja jätetty klikkaamalla haluamalleen paikalle. Tornia voidaan vaihtaa kesken lennon toiseen torniin. Pelaajan tarkoitus on tuhota viholliset torneja hyödyntämällä ennen kuin ne pääsevät muumitalolle ilman että sydämet jotka edustavat pelaajan elämäpisteitä loppuvat. Torneilla on erilaiset hinnat ja hinnat kertovat myös niiden tekemän vahingon suhteellisen suuruuden. Pelaaja saa aina tuhottuaan vihollisen yhden raha pisteen ja merkinnän vihollisen tuhoamisesta. Vihollisia on tarkoitus tuhota oikeassa ala-laidassa näkyvän vihollisten määrän verran. Määrä kasvaa sitä korkeampi taso (level) on kyseessä. Aina tämän vihollismäärän täytteessä ilman että

elämänpisteet pääsevät tippumaan nollaan siirtyy pelaaja seuraavalla tasolle, lyhyt aikaa kestävä väli-ikkuna kertoo tästä. Jos käyttäjä tosin häviää eli elämän pisteet pääsevät tippumaan nollaan tulee ikkuna ”Game Over”, josta BACK – nappia painamalla käyttäjä pääsee takaisin aloitusikkunalle. Kun pelin vika taso on suoritettu, tulee ”Game Won” ikkuna. Ikkuna sulkeutuu tämän jälkeen aika pian. Pelin voi käynnistää uudelleen toistamalla kaikki vaiheet taas alusta alkaen.

# Ohjelman rakennesuunnitelma

Graafinen luokkakaavio, jossa on kaikki metodit eri luokista ja vähän sovellettu ei siis täysin mene uml-kaavioiden ohjeiden mukaan.



Pääluokka on ehdottomasti TDWindow – luokka, joka hoitaa muiden luokkien kutsunnan, mutta sitä edeltää kuitenkin object MuumilaaksoDF -objektin ajaminen. MuumilaaksoDF – objekti sisältää alku canvasin/ikkunan luonnin ja siihen tarvittavat tiedot sekä huolehtii hiiren paikan, liikkeiden ja painalluksien kuuntelemisesta. TDWindow – luokka pitää sisällään käyttöliittymän päivityksen(repaint) ja alku (startScreen), ”next level”, ”game over” ja ”game won” – näkymät. Sen apu objektina toimii object TDWindow, joka säilöö tarvittavat parametrit.

GameBoard – luokka huolehtii pelilaudan palikoiden luonnista, pelilauta siis koostuu pituus suunnassa 12 ja leveys kertaa 8 palikkataulukosta. Tähän taulukkoon GameBoard – metodi loadSave lukee tiedostosta kartan.  GameBoardin aliluokka on Block luokka joka huolehtii jokaista block oliota koskevan datan tallennuksesta ja niiden luonteenpiirteistä.

Luokka Store määrittelee kauppapaikan mistä voi ostaa torneja ja näyttää myös muuta dataa käyttäjälle (elämäpisteet, rahat, levelin ja tapettavien vihollisten määrän).

Enemy – luokka määrittelee viholliset minionit olioina, jokainen huolehtii jokaisen yksittäiset vihollisen liikkumisesta ja ominaisuuksista esim. haisuli liikkuu kaksi kertaa nopeammin kuin hattivatit ja möröllä on kaksi kertaa enemmän elämänpisteitä. Luokka huolehtii niiden piirtämisestä ja niille tapahtuvista asioista ja sen aloituspaikan määrittelystä.

TDWindowTest ja RunningGameTest – testiluokat testaavat perustestejä monimutkaisimmissa ja ohjelman kannalta keskeisimmistä metodeista.

# Algoritmit

Tärkeimmät algoritmit, jotka on syytä toimia ohjelmassa tarvittavan hyvin, ovat: tornien vihollisiin tähtäys/taistelu (shoot) algoritmi, vihollisen etenemis (aiMob) ja synnytys (spawnMob)algoritmi. Lisäksi kartan luonti algoritmi tekstitiedostosta (loadSave) on syytä toimia moitteetta ja virheen sieppaaminen on tärkeää kaikissa näissä vaiheissa.

Vihollisen etenemisestä vastaava aiMob toimii hyvin yksinkertaisella tavalla kolmen apumetodin avulla. Vihollisen suunta koostuu neljästä akselista (ylös, alas, oikea ja vasen). Ne ovat merkitty kahdella eri tavalla, on nykyinen suunta ja mennyt suunta. Sen avulla vihollisen tekoäly tietää minne se on viimeksi kävellyt. Eikä valitse samaa suuntaa uudestaan takaisin. Lisäksi vihollisminioni pitää huolen siitä että suunnan tarkistus tapahtuu aina yhden blockin pituuden kävelyn jälkeen. Lisäksi algoritmi

tarkistaa aina onko se jo saavuttanut tukikohdan ja muumitalon.  Tähän kohtaan olisi voinut kehittää myös paljon hienomman algoritmin ja esim. randomisti toimivan suunnan valinnan ja lyhimmän reitin laskennan, mutta oletan että kuljettava tie on aina yksisuuntainen eikä koostu missään kohtaa ympyröistä, joihin tällä algoritmilla viholliset jäisivät pyörimään ympyrää.

Tornien taistelu/tähtäys algoritmi on toteutettu Block – luokassa. Se tarkistaa hyvin usein ampumisetäisyyden perustella olevat lähimmät viholliset ja ottaa ensimmäisen kohteekseen vihollisen joka ylittää tämän (myös piirretyn) tornin ampumisetäisyyden ja pitää vihollisen tähdättynä ja tulitettuna niin kauan kunnes tämä poistuu tornin ampumisetäisyydeltä tai vihollinen tuhoutuu. Torni ottaa tämän jälkeen heti seuraavan vihollisen kohteekseen joka on alueen sisällä tai joka tulee ampumisalueelle.

Vihollisten synnytys algoritmi on toteutettu osiksi TDWindowissa ja Enemy luokassa. TDWindowissa mobSpawner on pyörivä metodi joka huolehtii ajallisesti Enemy – luokassa olevan spawnMob – metodin kutsumisesta sekä antaa vihollisille yksilölliset ominaisuudet ja kutsuu niitä jako-osamäärään perustuen eri välein.

Kartan luontiin käytän Javan omaa Scanneriä, jolla helposti voi lukea tekstitiedostoa ja spacebarilla eroteltuja numeroita kutsumalla vain nextInt – metodia. Algoritmi huolehtii siitä, että kartan määrittely tapahtuu oikeanlaisen tekstitiedoston mukaan, jossa on oikea määrä kartan luontiin vaadittavia rivejä ja alku ja loppu pystysuorallarivillä ovat olemassa ainoastaan yksi lähtö ja lopetus kohta. Jos virhe sattuu jossain näistä ilmoittaa ohjelma graafisesti käyttäjälle että tiedosto on korruptoitunut (fileCorrupt). Kartan luonnissa käytetään kahta eri pintaa maapinta (maaID) ja ilmapinta (ilmaID). MaaID merkkaa tietä tai ruohoa ja ilmaID merkkaa mihin kohtaan ja millainen torni on sijoitettu sekä missä on maalipiste.

# Tietorakenteet

Tietorakenteina olen käyttänyt Scalan omaa kaksi ulotteista taulua (var block: Array[Array[Block]]) Block – palikoiden tallentamiseen. Lisäksi käytetään listoja/tauluja erilaisten kuvien tallentamiseen.

## Tiedostot

Tiedostona, jota luetaan, olen käyttänyt perinteistä tekstitiedostoa (.txt). Tätä tiedostoa vain luetaan eikä siihen koiteta kirjoittaa mitään. Esimerkkinä annan

ensimmäisen tason tekstitiedoston. Teksti tiedoston tiedostosijainti annetaan TDWindowissa suhteellisen alkuvaiheessa.

```
10

0  0  0  0  0  0  0  0  0  0  0  0
1  1  1  0  1  1  1  1  1  1  1  0
0  0  1  0  1  0  0  0  0  0  1  0
0  0  1  0  1  1  1  1  0  0  1  0
0  0  1  0  0  0  0  1  0  0  1  0
0  0  1  0  0  0  0  1  0  0  1  0
0  0  1  1  1  1  1  1  0  0  1  0
0  0  0  0  0  0  0  0  0  0  1  1

-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1  8
```

Ensimmäinen numero kertoo tasolla tapettavien vihollisten määrän. Seuraavaksi tuleva taulu kertoo pelilaudan muodostukseen tarvittavan tie pinnan kulkuväylän (1 numerolla) ja muiden pintojen ruohopinnat (0). Tätä taulua käytetään myös vihollisten paikan määrittelyyn. Tämän jälkeen tuleva taulu kertoo ensimmäisen taulun päällä olevan ilmataulun, johon sijoitetaan ohjelman aikana tornit sekä se kertoo missä maali sijaitsee numerolla(8). Maalipisteen ja viimeisen ensimmäisessä taulussa olevan tiepisteen on oltava samassa kohtaa kummassakin taulussa.

Karttaan ei myöskään kuulu tehdä silmukoita, koska vihollisten tekoäly ei ole luotu niitä varten. Lisäksi tien pitää olla yhtenäinen polku ja aloitus pystyrivissä ja lopetuspystyrivissä saa olla vain yksi lähtö ja lopetus tienpalanen.

## Testaus

Ydinmetodien testaus on tärkeää sillä ne on itse suunniteltuja ja rakennettuja ja niissä voi sattua helposti pieniä ja päällepäin huomaamattomia virheitä. Ydinmetodeita ovat: tekstitiedoston oikea luku että saadaan aikaiseksi oikeanlainen ja toimiva pelikartta, vihollisen liikkumisalgoritmi on syytä testata ja lisäksi tornien ampumisalgoritmi. Nämä kaikki tuli testattua mutta rakenteeltaan ne eivät kata koko algoritmin käyvää testausta. Tämä olisi vaatinut liikaa aikaa ja työtä. Siksi näillä yksinkertaisilla testeillä testataan vain muutamia metodien kulkemia polkuja ja

rakenteita. Ohjelman testaus suoritettiin oikeastaan vasta kunnollisena toteutuksena loppuvaiheessa projektia mutta pientä yksikkö kokeilua console ikkunassa toteutettiin koko projektin ajan.

## Ohjelman tunnetut puutteet ja viat

Ohjelmalle ei ole vielä kirjoitettu täysimääräisesti omia algoritmejani testaavia testejä. Vihollisten tekoäly on hyvin yksinkertainen eivätkä ne osaa laskea lyhintä mahdollista reittiä maaliin. Tämä olisi ollut parempi toteutus. Ne eivät myöskään pysty tunnistamaan silmukoissa oikeaa reittiä maaliin. Lisäksi tornien ampumis animaatio on vain viiva, joka olisi voitu toteuttaa paremmin jos olisi ollut enemmän aikaa. Lisäksi heikon laskennan koneissa voi tulla hidastuksia koska koko ohjelma pyörii yhden threadin varassa.

## 3 parasta ja 3 heikointa kohtaa

3 parasta kohtaa ovat graafinen toteutus pelille, aiMob – metodi ja koko projektin toteutus, koska kyseessä on suuri ja paljon koodirivejä vaatinut kokonaisuus.

1. Olen enemmän front-end puolelle erikoistunut devaaja. Sen takia grafiikka on itselle tärkeää. Tosin tätäkin voisi vielä parantaa vaikka kuinka paljon.
2. loadSave – metodi eli tekstitiedoston lukumetodi on toteutettu yksinkertaisesti ja nopeasti Javan scanner – metodilla.
3. Kyseessä oli suuri peli projektin toteutus jossa on hyvin paljon koodirivejä ja paljon koodattavaa kuitenkin yksinkertaisesti toteutettua ilman että olisin käyttänyt liiaksi Scalan erikoisia metodeita.

3 heikointa kohtaa ovat toteutus yhden ytimen varaan, vihollisten tekoäly ja testien riittämätön kattavuus.

1. Kaikki toteutus on luotu yhden ytimen varaan, joka voi haitata huonommilla koneilla pelin ajoa.
2. Vihollisten tekoäly on hyvin yksinkertainen eivätkä ne osaa laskea lyhintä mahdollista reittiä maaliin. Tämä olisi ollut parempi toteutus. Ne eivät myöskään pysty tunnistamaan silmukoissa oikeaa reittiä maaliin.
3. Ohjelmalle ei ole vielä kirjoitettu täysimääräisesti omia algoritmejani testaavia testejä, koska ne olisivat vaatineet enemmän aikaa kuin oli käytettävissä.

## Poikkeamat suunnitelmasta

Suoritin testien kirjoittamisen vasta loppuvaiheessa vaikka olin suunnitellut tekeväni sen jo jokaisessa välissä hiljalleen. Aikataulun noudattaminen ei aivan osunut

nappiin, muut kurssit ja kesätyönalkaminen söi aikaa projektin hiljattaisesta viimeistelystä. Erilaisten vihollisten luominen käyttäjälle on vaikeaa, vaikka se annetaan alustavassa suunnitelmassa mahdollisuutena. Projektin suunniteltu toteutusjärjestys täyttyi osittain, muttei aivan.

## Toteutunut työjärjestys ja aikataulu

Lähdin toteuttamaan projektia heti nollapäivästä alkaen, koska tiesin että keväällä suhteellisen aikasin alkavat kesätyöt syövät projektiin käytettävää aikaa silloin. Repositoryn luonti tapahtui 27.2.2016 ja ensimmäinen peli projektin aloittava commit samana päivänä.

27.2.2016

- MainFrame – ikkunan luonti

29.2.2016

- TDWindow – luokka ja GameBoard – luokkien luonti alustavasti tyngillä metodeilla.

2.3.2016

- Tynkien metodien täydennys niin kuin loadSave – metodi

8.3.2016

- Enemy – luokan luonti ja vihollisten tekoälymetodi

11.3.2016

- Store – luokan luonti ja metodeita sinne

15.3.2016

- Testikirjaston lisääminen ja luokan luonti

21.3.2016

- Piirtäminen on valmis kokonaisuudessaan ja pelikartta ja peli toimii jo alustavasti. Nyt enää ominaisuuksien lisäystä ja bugien korjausta ja virheisiin varautumista.

20.4.2016

- Pelin hiontaa ja testien kirjoittamista ihan viimeiseen päivään 5.5.2016 asti hieman kiireessä

Suunnitelmassa poikettiin siinä että testit jätettiin aivan viimeiseksi toteutukseksi.

## Arvio lopputuloksesta

Arvioni lopputuloksesta on todella hyvä. Sain toteutettua mielestäni omaan tasooni nähden hyvin kaikki haluamani ominaisuudet. Kuitenkin oletin itseltäni vielä hieman lisää mutta en aivan saavuttanut sitä lopputulosta. Missä vihollisilla olisi ollut älykkäämpi teko-äly, vihollisten jälkikäteen luonti olisi ollut helpompaa, ohjelman laskennan rakenne olisi jaettu useammalle ytimelle, testit olisivat olleet kattavampia ja esim. torneilla olisi ollut omat ampumis animaationsa, grafiikka olisi vielä hiottu hienommaksi ja pelilautaa olisi voinut skaalata erikoikoiseksi. Nämä kaikki jäi toteuttamatta, koska projekti oli jo ilman näitä todella työläs ja aikaa vievä kokonaisuus.

Jos jatkaisin vielä projektia, toteuttaisin ensimmäiseksi viholliselle paremman teko-älyn.

## Viitteet


Introduction to the Art of Programming Using Scala – Mark C. Lewis

https://greengoblin.cs.hut.fi/o1_s2015/course/index.php

http://www.scala-lang.org/api/current/#package

https://docs.oracle.com/javase/7/docs/api/

https://www.youtube.com/watch?v=_DqMjz4GIEc

https://www.youtube.com/watch?v=_7dU6-L0Dps