Aleksi Hirvensalo
728719
Tietotekniikka
Vuosikurssi: 2019
27.4.2020

Projekti dokumentti

Yleiskuvaus

Olen tehnyt tornipuolustus pelin. Ideana on että käyttäjä voi luoda karttoja tai pelata ennalta luoduista kartoista, joissa on tietä ja nurmikkoa. Vihollisia tulee kartan vasemmasta reunasta ja pyrkivät tietä pitkin oikeaan reunaan. Reunan saavuttaessa pelaaja menettää yhden elämäpisteen.

Pelaajan tehtävä on estää vihollisen pääsy maaliin asettamalla nurmikolle torneja. Torneja on kahdenlaisia, hyökkäys- ja tukitorneja. Tornit maksavat, jotta niitä voi asettaa, joten pelaajan pitää kerätä rahaa tuhoamalla vihollisia. Hyökkäystornit soveltuvat vihollisten tuhoamiseen. Tukitornit taas tukevat hyökkäystorneja antamalla niille lisää kantamaa, tulivoimaa ja/tai nopeutta. Pelaaja voittaa pelin kun on saavuttanut tietyn kierrosmäärän.

Pelille on luotu käyttöliittymä ja asetustiedostoja, joissa voi muokata ja luoda torneja, vihollisia, karttoja ja asetuksia. Pelissä on 3 vaikeustasoa, easy, medium ja hard. Pelaaja pystyy tallentamaan meneillään olevan pelin, tallennuksia voi kuitenkin vain olla koko ajan maksimissaan yksi. Mielestäni työ on toteutettu tasolla vaativa.

Suunnitelmaa on noudatettu pääpiirteittäin. Pelaajalle on annettu enemmän valtaa muokata peliä kuin alussa oli tarkoitus. Pelaaja pystyy esim. muokkaamaan rahan ja elämien määrä sekä luoda karttoja.

Käyttöohje

Ennen ohjelman käynnistystä pelaaja voi luoda uusia torneja, vihollisia tai karttoja projektin tekstitiedostoissa. Nämä tiedostot ovat nimetty asianmukaisesti ja niissä on tarkemmat ohjeet ja esimerkit formaatista. Myös asetuksia voi muokata settings.txt tiedostosta. Nämä eivät kuitenkaan ole pakollisia, sillä niissä on jo alustavasti luotu pelille tarvittavat asiat. Ohjelman voi siis käynnistää ilmankin muokkausta.

Ohjelma käynnistetään Interface paketin TowerDefenseApp-oliosta. Käyttäjälle avautuu valikko, josta voi valita uuden pelin tai ladata pelin. Pelin lataus on mahdollista, jos peliä on pelattu aikaisemmin ja siellä on vanha tallennus. Jos pelaaja valitsee uuden pelin niin hän pääsee valitsemaan kartan. Valittuaan kartan pelaajalle aukeaa itse pelin käyttöliittymä.

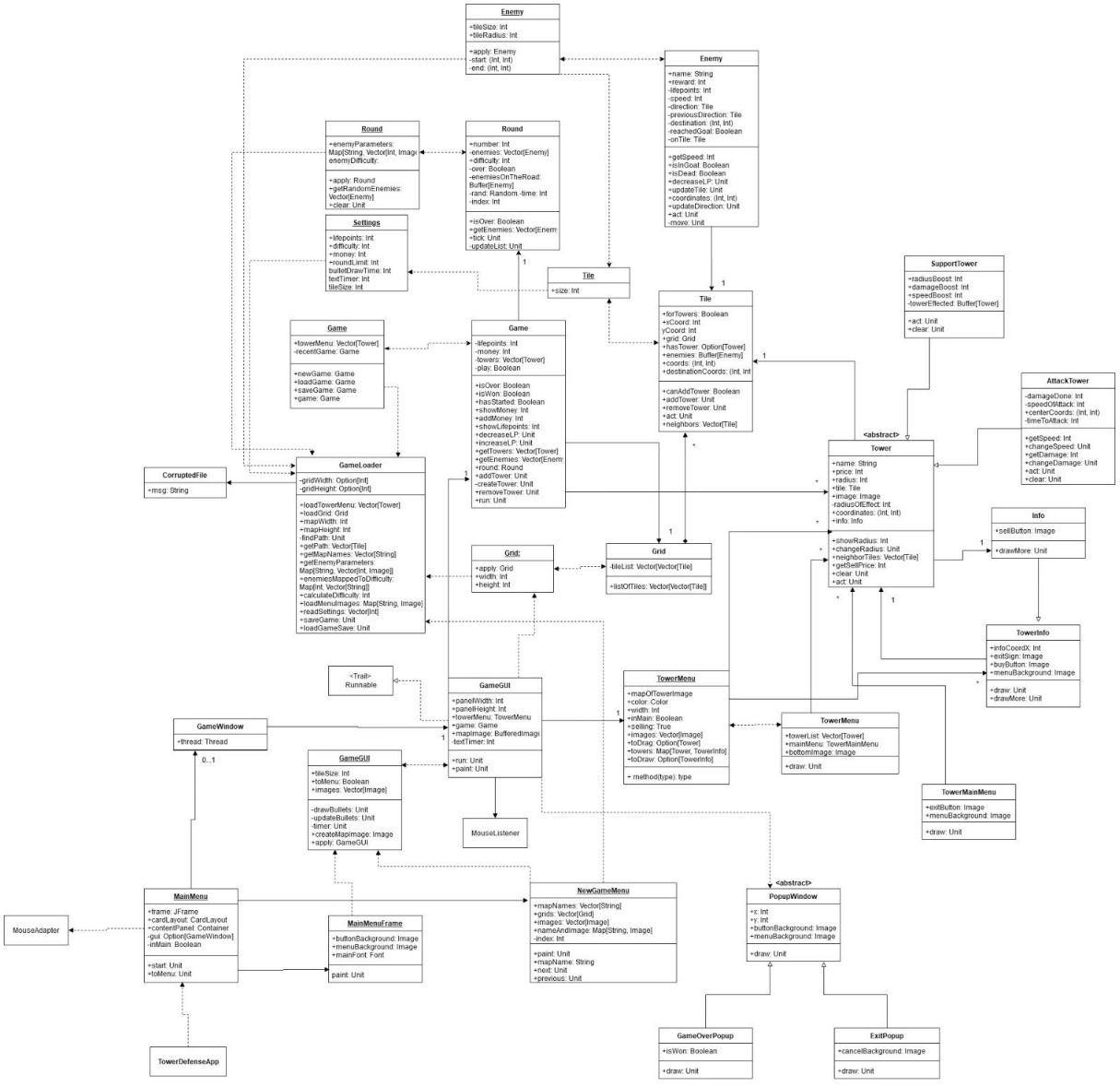
Pelissä oikeassa reunassa näkyy jokaisen tornin kuva. Tornin kuvaa painaessa aukeaa näyttö, jossa on informaatiota tornista ja nappi, josta hiirellä painaessa ja vetäessä saa

ostettua tornin. Tornin voi vetää vapaalle nurmikkolaatalle päästämällä hiirestä sen kohdalla. Tornia ostettaessa kannattaa katsoa sen kantama ja myös huomioida se että rahat riittävät ostoon.

Kun on valmis ensimmäiseen kierrokseen voi painaa Play-nappia alareunasta ja kierros lähtee käyntiin. Kierroksen aikana voit lisätä uusia torneja tarvittaessa. Kun viholliset loppuvat kierros on ohi ja uuden kierroksen voi taas aloittaa play-näppäimellä.

Pelin voi lopettaa painamalla oikeasta yläreunasta X-näppäintä. Lopettaessa voi valita haluaako tallentaa pelin (tallennus korvaa aiemman tallennuksen) tai haluaako poistua valikkoon ilman tallennusta.

Ohjelman rakenne



Huom! UML-kaavio ei ole täysin yksityiskohtainen, jotkin riippuvuudet esim. Swing-komponentteihin on jätetty merkitsemättä ja pitkät listat epäolennaisia muuttujia jätetty pois. Myöskään funktioille ei ole merkitty parametreja, selkeämmän lopputuloksen vuoksi. Kaaviolla pyrin kuvaamaan pääpiirteittäin olemassa olevia luokkia ja olioita sekä niiden suhdetta.

Torneilla on abstrakti yläluokka Tower, jossa on muuttujia, jotka ovat yhteisiä kaikille tornityypeille, kuten nimi ja hinta. Tornilla on myös yksi abstrakti metodi act(), jota kutsutaan aina tornin vuorolla. Torni on abstrakti siksi, koska kaikki pelissä luodut tornit tulee joko olla hyökkäys- tai tukitorneja. Yksi alaluokista on AttackTower, joka kuvastaa hyökkäystorneja. Niille on määritelty sille ominaisia, muuttujia kuten attackSpeed ja damage. Toinen alaluokista on SupportTower eli torni, joka tukee hyökkäystorneja, sillä on itselleen ominaisia muuttujia, kuten attackBoost ja speedBoost. Olisin myös voinut luoda alaluokkia vielä näillekin, mutta se ei vaikuttanut kovin järkevältä, sillä luokkia olisi tullut sekavan paljon ja myös tekstitiedostoissa tornien määrittely olisi ollut paljon monimutkaisempaa.

Enemy-luokka kuvaa vastustajaa ja sen keskeisin metodi on move, jonka avulla se pystyy liikkumaan eteenpäin, kun act metodi sitä kutsuu. Luokalla on muuttujia, kuten onTile pitämään sijaintia tallessa ja direction pitämään suunnan tiedossa. Enemy voisi myös olla abstrakti yläluokka, mutta mielestäni jaottelu on tässä turhaa, kun pelissä olevat viholliset voi hyvin erotella muuttujien avulla ja yksi vihollisluokka on myöskin selkeä. Siinä tapauksessa, jos olisin ehtinyt luoda esim. lentäviä vihollisia niin silloin olisi ollut aiheellista miettiä kannataisiko tehdä alaluokat maa- ja ilmavihollisille.

Enemyllä on myös kumppaniolio, jolla on tiedossa reitti, jota pitkin vihollisen pitää kulkea. Viholliset voivat kysyä oliolta aina reittiä, kun liikkuvat. Kumppanioliolla on myös apply metodi, joka luo vihollisen annetuilla parametreilla.

Round-luokka kuvastaa yksittäistä kierrosta, sillä on mm. kierroksen numero ja vaikeusaste. Round-luokalla on tick-metodi, jota kutsuttaessa kierros etenee yhden askeleen. Jokaisella askelella liikutetaan vihollisia ja mahdollisesti luodaan niitä lisää kentälle. Kierroksella olevat viholliset arvotaan luokan kumppaniolion apply-metodissa, jossa luodaan uusi kierros ja kasvatetaan sen vaikeustasoa hieman. Metodissa kutsutaan olion getRandomEnemies metodia, joka tuottaa satunnaisesti joukon vihollisia, jotka vastaavat vaikeustasoa. Mahdolliset viholliset ja niiden vaikeustason kumppaniolio saa GameLoader-oliolta. Round-luokan toteutus on tehty, sillä selkeyttää kokonaisuutta ja jakaa vastuuta. Nyt myöskään kierrosominaisuutta ei tarvitse sisällyttää esim. Game-luokan sisälle.

Game-luokka kuvastaa itseä peliä. Game-luokan tehtävä on hallita torneja eli luoda ja poistaa niitä. Game-luokalla on myös Grid, joka kuvastaa karttaa. Luokka osaa siis tähän lisätä torneja. Game pitää kirjaa elämäpisteistä ja rahatilanteesta. Gamella on run-metodi, jota kutsuessa peli etenee yhden askeleen. Gamessa on määritelty voitto- ja häviämisehdot, joiden jälkeen peli on ohi. Game-luokka luo aina tarvittaessa uuden kierroksen. Game-luokka tuo selkeyttä projektiin ja se on keskeinen osa toteutusta. Periaatteessa tornien luominen ja poistaminen olisi voinut yhtä hyvin antaa Grid-luokalle, mutta päädyin

tähän toteutukseen, sillä mielestäni on selkeämpää, että tornit luodaan ja poistetaan peli-luokan toimesta ja peli pystyy olemaan suoraan yhteydessä torneihin.

Game-luokalla on kumppaniolio, jolla on metodi newGame, joka luo uuden pelin valitulle kartalle. saveGame taas tallentaa pelin ja loadGame lataa pelin tallennuksesta. Olio kommunikoi GameLoader-olion kanssa, joka hoitaa pelin latausta ja tallennusta tekstitiedostojen kanssa.

Grid-luokka kuvastaa siis karttaa. Se on jaoteltu Tile-olioihin. Jaottelu on tehty siksi että tornien luominen omille Tile-oliolle on luontevaa ja kartan piirtäminen helpompaa. Tällöin karttaa voi myös helposti kuvata myös kaksiulotteisella taulukolla. Gridin kumppanioliolla on kartan leveys ja korkeus pikseleinä. Tämä on siksi että muut luokat voivat helposti saada tietoonsa kartan koon ilman että sitä syötettäisiin parametrina. Gridillä on myös apply-metodi.

Tile-luokka kuvastaa koordinaatiston yhtä laattaa. Sillä voi olla torni tai vihollisia. Tiilellä on act-metodi, jota tornit osaavat kutsua. Tämä tarkastaa onko tornin lähellä vihollisia ja jos on niin kertoo käyttöliittymälle, että piirtää tornin ja vihollisen välille viivan eli luodin. Tile voisi myös olla yläluokka ja sillä voisi olla alaluokat GrassTile ja RoadTile, mutta tätä en kumminkaan ehtinyt toteuttaa, sillä tulin ajatelleeksi asiaa liian myöhään. Tämä olisi selkeyttänyt toteutusta ja eliminoisi riskin, että viholliset voisivat jotenkin päästä nurmikolle (tätä ei siis ole tapahtunut, mutta voisi olla teoriassa mahdollista.). Tilellä on myös kumppaniolio, jolla on tieto laatan koosta pikseleinä. Tämä on sen takia, että muut osat (lähinnä käyttöliittymä) voi tiedustella laatan kokoa ilman että sitä annetaan parametriksi.

GameLoader-olio hoitaa tiedostojen lukemisen. Sillä on metodit tornien ja vihollisten lukemiselle. Se osaa lukea kartat ja tuottaa niistä taulukon. Se hoitaa myös pelin tallentamisen tekstitiedostosta ja osaa palauttaa tallennetun pelin. Nämä metodit on tarkoitettu kutsuttavaksi useista eri kumppaniolioista, kuten Round, joka haluaa tietää kaikki pelissä olevat viholliset tai Game, joka haluaa tietää kaikki mahdolliset tornit. GameLoader on erittäin tärkeä välikäsi, sillä moni asia on määritelty tekstitiedostoissa. GameLoaderin voisi myös jättää pois ja laittaa esim. jokainen kumppaniolio lataamaan itse tarvittavat tiedot tekstitiedostoista, mutta se tekisi, joistain olioista tarpeettoman isoja eikä niiden muut vastuut korostuisi niin paljon esim. koodia lukiessa. Tekstitiedostojen käsittelyä varten on mielestäni selkeämpi, että on siihen erikoistunut olio luotuna. GameLoader-olion pilkkominen jotenkin pienempiin osiin voisi olla järkevää, sillä tällä hetkellä.se on aika iso ja sen lukeminen voi olla työlästä, kun metodit ovat niin isoja. Myöskin metodien pilkkominen saattaisi auttaa.

Settings-oliolla on tallennettuna tietoa keskeisistä asetuksista, joita se kysyy ohjelman alussa GameLoaderilta. Olio on luotu selkeyttämään toteutusta. Nyt voidaan vaan kysyä Settings-oliolta pelin asetukset.

TowerDefenseApp-olio käynnistää ohjelman kutsumalla MainMenu-olion start metodia. Tämä asettaa MainMenun näkyväksi. MainMenu on ikkuna, jossa on kaksi näkymää NewGameMenu ja MainMenuFrame. MainMenussa on MouseAdapter, joka kuuntelee käyttäjän klikkauksia.

MainMenuFrame kuvastaa käyttöliittymän aloitusnäkymää. Se on JPanelin alaluokka, joten sillä on paint-metodi, jossa piirretään itse kuva.

NewGameMenu kuvastaa uuden pelin luomisvalikkoa, jossa voi valita eri kartoista. Tämä on myös JPanelin alaluokka , joten sillä on paint-metodi, jossa piirretään itse kuva.

MainMenu, NewGameMenu, ja MainMenuFrame voisivat olla luokkia objektien sijaan, mutta en nähnyt sille tarkoitusta, sillä aina ohjelman alussa kumminkin luodaan nämä oliot.

Kun käyttäjä on valinnut haluamansa pelin, MainMenu luo uuden GameWindow-olion ja MainMenu poistuu näkyvistä. GameWindow-olio luo uuden GameGUI-olion, joka kuvastaa itse pelin käyttöliittymää. GameWindow itsessään on ikkuna, jonka päällä GameGUI on. GameWindow luo uuden säikeen, jossa GameGUI pyörii. Säikeen luominen ei ole pakollista, mutta sen avulla voi pelin nopeutta säädellä kätevästi ja tarvittaessa pysäyttää.

GameGUIlla on runnable piirre, joten siitä voi helposti luoda uuden säikeen. GameGUI hoitaa kartan piirtämisen. GameGUIlla on tallessa nykyinen peli, jota se osaa lukea ja piirtää sen perusteella. GameGUI kuuntelee myös käyttäjän hiirtä, jolloin se osaa keskustella käyttäjän kanssa. Käyttäjä siis pystyy hallitsemaan peliä GameGUIn kautta. GameGUIlla on myös TowerMenu-muuttuja, jota kautta se kutsuu sen draw-metodia, joka vastaa tornivalikon piirtämisestä kartan viereen ja myös alareunan piirtämisestä. GameGUI voisi myös hoitaa tornivalikon piirtämisen, mutta silloin toteutuksesta tulisi liian iso.

GameGUIn kumppaniolio osaa luoda uuden GameGUIn apply-metodin avulla. Olio lataa jonkin verran piirtämiseen vaadittavia kuvia heti kun ohjelma käynnistyy, sillä olen huomannut sen tekevän ohjelmasta sulavamman tuntuisen. Oliolla on myös Boolean-muuttujia, joita pystyy muokkaamaan ulkopuolelta, tämä on huonoa koodia sinänsä, että sitä voidaan muokata mistä vain, mutta se oli paras toteutus mitä keksin sillä tämä mahdollistaa esim. että voidaan kommunikoida GameGUIn ulkopuolelta, että halutaan kirjoittaa jotain tekstiä ruudulle.

TowerMenu kuvastaa siis tornivalikkoa kartalla, sillä on draw-metodi, joka kysyy kumppanioliolta, että mitä halutaan piirtää eli jonkun kyseisen tornin osto- tai myyntivalikko vai päävalikko torneille. GameGUI taas kommunikoi TowerMenun kumppanioliolle, että mitä piirtää. Päävalikkoa kuvastaa taas TowerMainMenu-luokka, jolla on draw-metodi, joka osaa piirtää päävalikon näkymän.

TowerInfo kuvastaa tietyn tornin ostovalikkoa, sillä on viittaus Toweriin ja sillä on draw-metodi, joka piirtää valikon tornin tiedoilla. Info on taas TowerInfon alaluokka ja se kuvastaa jokaisen tornin myyntivalikkoa. Jokaisella pelissä luodulla tornilla on siis oma Info-olio, joka pystyy piirtämään myyntivalikon kyseiselle tornille.

Abstrakti luokka PopupWindow kuvastaa ponnahdusikkunoita, jotka on esiintyvät kun pelaaja haluaa poistua pelistä tai peli on päättynyt. Luokalla on abstrakti metodi draw. Jonka tarkoitus on pystyä piirtämään ponnahdusikkuna. Luokan alaluokat ovat GameOverPopup ja

ExitPopup. Nämä voisivat myös olla suoraan PopupWindow-olioita eikä se olisi abstrakti luokka, mutta ponnahdusikkunat voivat erota niin paljon toisistaan, että on parempi luoda niille omat luokkansa, jotta niiden piirtäminen olisi helpompaa, mutta koodirivejä on enemmän.

GameOverPopup kuvastaa ponnahdusikkunaa, kun peli on päättynyt ja sen draw-metodi osaa piirtää voitto- ja häviö-näkymän. ExitPopup osaa piirtää poistumisvalikon.

PopupWindown alaluokat, TowerMenu, TowerInfo ja Info on suunniteltu piirtämään GameGUIn ikkunaan. Niiden kutsuminen siis lähtee GameGUIsta.

Yleisesti olisin voinut vähentää kumppaniolioiden käyttöä, mutta se selkeyttää toteutusta ja niiden kautta on helppo hakea tarvittavaa tietoa ilman, että joutuu syöttämään jotain olioita parametreiksi joka puolelle.

Algoritmit

Monimutkaisia algoritmeja pelissä ei ole, sillä lähdin alussa sillä ajatuksella, että en halua paljoa laskentaa pelin aikana. Myöhemmin kumminkin tajusin, että sillä ei ole merkitystä tässä pelin sujuvuuden kannalta. Viholliset eivät siis itse laske polkuaan kokoajan vaan polku on ennalta laskettu. Polku lasketaan siten, että lähdetään alkupisteestä, joka on tiedossa koska se määritellään jokaiselle kartalle. Lähdetään siis käymään läpi koordinaatistoa, jostain tiilestä. Katsotaan mihin suuntaan voidaan kulkea. Kulkea voidaan, jos kyseisessä suunnassa on tietä ja siitä ei ole aikaisemmin kuljettu. Tätä lasketaan kunnes saavutaan maaliin, joka on tiedossa.

Tornit katsovat koordinaatistonsa läheisyydestä jokaisen tiilen, jotka on järjestetty oikealta alhaalta ylös vasemmalle, jotta katsotaan aina ensin maalia lähinnä oleva tiili.
Jokaiselle viholliselle lasketaan vaikeustaso, siten että vaikeustasoja on 1-3. Selvitetään ensiksi jokaisen ominaisuuden keskiarvo kaikista olemassa olevista vihollisista, jonka jälkeen tarkastellaan kyseisen vihollisen ominaisuuksia ja katsotaan poikkeavatko ne yli 33% jompaan kumpaan suuntaan keskiarvosta ja sen perusteella päätetään vihollisen keskiarvo.

Kierrokselle arvotaan viholliset siten että täytetään kokoelmaa satunnaisesti arpomalla olemassa olevia vihollisia ja jokaisen arvonnan jälkeen tarkastetaan onko saavutettu jokin yhteenlaskettu tavoite vaikeustaso. Arvonta lopetetaan kun tämä ollaan saavutettu.

Uusi vaikeustaso otetaan aina viimeisimmän kierroksen vaikeustasosta ja lisätään siihen alkuperäinen vaikeustaso (1, 2 tai 3) ja kerrotaan se kierroksen numerolla.

Tietorakenteet

Pääasiassa käytän Vectoria, Bufferia, Mappia ja Optionia.

Vectoria käytän tilanteissa missä tarkastellaan joukkoa asioita, jotta ei tule vahingoksi muokanneeksi mitään. Tässä voisi toki käyttää muitakin kokoelmia, mutta ne eivät tarjoa tätä suojaa.

Bufferia käytän tilanteissa, joissa haluan päivittää jotain joukkoa. Esim tornit pelissä, voin vain helposti lisätä ja poistaa niitä Bufferista. Tässä voisi yhtä hyvin myös käyttää Arrayta, sillä tiedetään maksimi tornien määrä. Mutta esim. vihollisten luomisessa maksimimäärä ei ole tiedossa, joten Bufferin on siihen mielestäni paras ratkaisu.

Mappia käytän kun halutaan tallentaa, jotain tietoa, jota voi myöhemmin hakea jollain tunnisteella. Esim. tornit ja niiden nimet. Tämän voisi tehdä myös vaikka Vector[(Int, Int)], mutta Map on luotu juuri tähän käyttötarkoitukseen. Mapista käytän immutable ja mutable versioita hieman riippuen tarvitaanko muokkausta vai ei.

Optionia käytän taas tilanteissa, joissa alkuarvoa ei välttämättä ole mutta myöhemmin sellainen voi olla olemassa. Esim. vaikka viimeisin kierros, sillä sitä ei ole olemassa vielä ensimmäisellä kierroksella. Tähän en oikeen keksi hyvää vaihtoehtoa muuta kuin alustaminen nullilla, mutta sen takia juuri taitaakin olla, että Optionit on luotu Scalaan.

Tiedostot ja verkossa oleva tieto

Ohjelma käsittelee tekstitiedostoja ja kuvatiedostoja. Kuvatiedostoista käyttäjän ei tarvitse huolehtia, muuta kuin jos haluaa luoda tornin tai vihollisen niin sille on annettava kuva ja se on lisättävä Image-kansioon.

Tornien, vihollisten, karttojen ja asetusten määrittelyyn on luotu tekstitiedostot. Tekstitiedostoissa on yksityiskohtaisemmat ohjeet. Nämä tiedostot ovat towers.txt, enemies.txt, maps.txt ja settings.txt. Tiedostoihin on myös valmiiksi luotu kaikki tarvittava, joten niiden muokkausta ei vaadita, mutta niitä voi muokata halutessaan. Näistä voi myös katsoa esimerkkiä formaatista, kun luo uusia olioita.

Testaus

Ohjelmaa on testattu paljon käyttöliittymän kautta, sillä se on ollut helpointa ja siitä näkee nopeasti useitakin asioita mitkä ovat menneet pieleen. Olen kumminkin luonut muutamia yksikkötestejä asioille, joita ei välttämättä suoraan näe käyttöliittymän kautta. Käyttöliittymän kautta testausta tuli tehtyä jatkuvasti, kun testasin aina uutta koodia.

Testaan muun muassa sitä että kartat, viholliset ja tornit osataan lukea oikein tekstitiedostoista. Testit kattavat vain olennaisimmat kohdat. Jokaista mahdollista variaatioita annettavasta syötteestä on mahdotonta keksiä ja testata. Toisin sanoen, jos testi menee läpi se kertoo vain, että peruskäytössä toiminto toimii odotetusti. Tiedostojen lukemiselle olen luonut omat tekstitiedostot, sillä metodini on suunniteltu lukemaan tiedostoja suoraan eikä pelkkää tekstiä.

Testasin, että kartoista osataan ladata oikea karttaa ja testasin myös, että olemattoman kartan lataaminen ja virheellisen kartan lataaminen tuottaa virheilmoituksen.

Torneilla ja vihollisilla testasin, että ne osataan lukea oikein ja ne saavat oikeat parametrit. Testejä en ole luonut virheellisen syötteen varalta, mutta sitä olen testannut käsin yksinkertaisesti kirjoittamalla virheellisiä määrittelyjä. Tämä sen takia, että tiedoston lukemismetodi lukee koko tiedoston kerrallaan, joten minun pitäisi luoda jokaiselle virhetapaukselle oma tekstitiedostonsa.

Olen myös testannut, että kierrosten edetessä vaikeustaso kasvaa kokoajan.

Olin ajatellut suunnitelmaa tehdessä, että testaisin paljon enemmän toiminnallisuutta luomalla testejä niille, mutta oli paljon kätevämpää ja nopeampaa luoda ne käyttöliittymän kautta. Tosin siinä tulee ongelmaksi se että jos muokkaa ohjelmaa hieman laajemmin niin periaatteessa pitäisi käsin testata taas kaikki ominaisuudet, että voidaan varmistua toimivuudesta.

Ohjelman tunnetut viat ja puutteet

Koodin turvallisuudessa on pieniä puutteita esim. kumppaniolioiden osalta, sillä joidenkin kumppaniolioiden muuttujia voi muokata ulkopuolelta. Tämän toteutuksen muokkaminen vaatii itseltäni paljon miettimistä ja yksi vaihtoehto olisi muokata ohjelmarakennetta uusiksi, tähän minulla ei valitettavasti ole aikaa enkä näe sitä niin kriittisenä virheenä tämän projektin näkökulmasta. Yksi vaihtoehto tälle olisi, että syötetään olioille parametreina toisia olioita, joita halutaan muokata, mutta silloin taas joutuisimme antamaan koko olion vain yhden asian takia.

Jos luo kartan, joka on yli 9x9 niin jostain syystä viholliset alkavat tökkiä. En ole tätä saanut korjattua. Selkeästi vaikuttaisi, että juurikin kyseessä on kartan koko eikä vihollisten tai tornien määrällä ole merkittävää vaikutusta asiaan. Olen tätä koittanut muuttaa hidastamalla säikeitä ja luomalla uusia, mutta en ole onnistunut löytämään ratkaisua. Olen myös koittanut muokata piirtämisen järjestystä ja tehnyt pientä hienosäätöä , mutta en ole huomannut vaikutusta.

Periaatteessa pelattavuuden voisi laskea puutteeksi. Tällä hetkellä pelikokemus on ihan ok, mutta pelattavuutta voisi parannella vielä enemmän. Korkeilla kierroksilla pelin pelaaminen on liian helppoa. Tätä voisi tasapainottaa vaikka kiihdyttämällä kierrosten vaikeustason kasvua.
Toisaalta taas ohjelmaan pelaaja pystyy luomaan mitkä viholliset, tornit ja kartat tahansa sekä asettaa itse sääntönsä, joten pelattavuus on yksi ominaisuus mitä pelaajan on mahdollista itse säädellä.

Testauksen osalta puutteita on siinä, että pelin muokattavuuden takia on mahdotonta varmistua kaikista mahdollisista tilanteista, mitä peli voi saada syötteenä tekstitiedostoista. Joten bugeja voi periaatteessa olla erittäin paljon, mutta niiden löytäminen vaatisi paljon

testaamista. Tätä pystyisi vaan paikkaamalla sen mukaan kun ohjelman ajossa löytyy virheitä.

Jos kartta tiedosto on määritelty virheellisesti niin ohjelma heittää heti asianmukaisen virheen eikä avaudu ja jos tornit tai viholliset on määritelty virheellisesti niin peli ei anna pelata karttaa ja heittää asianmukaisen virheilmoituksen. En ole varma onko tämä puute/vika, ja en keksi parempaa tapa reagoida asiaan, sillä jos määrittelyt tiedostoissa ovat pielessä niin peliä ei voi pelata.

Löysin viimeisellä pelikerralla bugin, joka aiheutti sen että kun painoin yhden tornin kuvaa valikossa niin se aukaisi väärän tornin kuvan. Poistun tornin näkymästä ja koitin muutaman kerran uudestaan niin bugi katosi ja en ole sen jälkeen huomannut sitä vaikka olen sitä paljon testannut käyttöliittymän kautta. En ymmärrä miten tämä virhe on mahdollinen.

3 parasta ja 3 heikointa kohtaa

Hyvät kohdat:

Mieleeni ei tule mitään tiettyä kohtaa, mutta olen tyytyväinen siihen että ohjelma toimii sujuvasti ja olen onnistunut luomaan asetustiedostot hyvin. Yleisesti luokkajako on mielestäni tehty hyvin ja muuttujia käytetty oikein.

Huonot kohdat:

Koodin sekavuus on yksi kohta. Koodini on mielestäni ihan hyvää, mutta paikoin se on hieman sekavaa. Esim. Interface paketissa kaikkiin luokkiin tutustuminen on hieman hankalaa. GameLoader-olio on aika suuri ja sen metodit myös. Tätä olisi voinut hieman pilkkoa osiin, mutta en ole ihan varma mikä olisi järkevin tapa. Suurilta osin olen kumminkin mielestäni käyttänyt julkisia ja yksityisiä muuttujia hyvin.

Käyttöliittymän grafiikkoihin en ole tyytyväinen. Tähän tarvitsisin parempaa tietotaitoa käyttöliittymien luomisesta Scala/Javalla ja myöskin taiteellista silmää.

Poikkeamat suunnitelmasta, toteutunut työjärjestys ja aikataulu

0 - 2vk:

Suunnitelman mukaisesti aloitin, tekemään torneja ja vihollisia. En tosin saanut niitä valmiiksi ja en aloittanut myöskään toteuttamaan kierrosluokkaa. Aikaa käytin noin 5h ja olin suunnitellut 16h.

Aika: 5h

2 - 4vk:

Sain luotua suunnitelman mukaisesti peli-luokan. Tornit, viholliset ja koordinaatiston sain valmiiksi myös. Olin suunnitellut aloittavani käyttöliittymää, mutta en ehtinyt, myöskin karttaa olin suunnitellut aloittavani, mutta teinkin sen tässä välissä. Aikaa olin varannut 20h ja käytin noin 10h

Aika: 10h

4 - 6vk:

Suunnitelmani oli jatkaa käyttöliittymän ja kartan toteutusta. Todellisuudessa aloitin vasta käyttöliittymää ja kartta oli jo valmiina. Aikaa olin varannut 20h ja käytin 15h.
Aika: 15h

6 - 8vk:

Tein ylivoimaisesti eniten töitä kahdella vikalla viikolla. Tein käyttöliittymän loppuun ja lisäsin muutamia ominaisuuksia. Suunnitelman mukaisesti hion peliä loppuun ja testasin peliä.
Aikaa olin varannut 30h ja käytin aikaa lähemmäs 50h.

Kokonaisaika: 80h

Kokonaisaika on aika lähellä suunnitelman kokonaisaikaa, mutta toteutusjärjestyksessä on poikkeavaisuuksia ja myös työn tahdituksessa.

Projektia tehdessä opin vielä enemmän suunnitelmien tarpeellisuudesta. Olisin voinut vielä enemmän miettiä suunnitelmaani.

Kokonaisarvio lopputuloksesta:

Toteutukseni on mielestäni hyvä, mutta se ei ole mitenkään erinomainen. Se toimii hyvin ja tekee tehtävänsä, mutta olisin halunnut siihen lisätä vielä hieman enemmän toimintoja, kuten musiikkia ja tornien päivitystä. Oleellisia puutteita ei siis ole.

Ohjelmassa olen pyrkinyt miettimään laajennettavuutta esim. uusien tornien, vihollisten ja karttojen luominen on helppoa. Uusien sivujen luominen käyttöliittymään on myöskin helppoa.

Ohjelmaa olisi voinut vielä suunnitella paremmin ja ohjelman toteuksen aikana pysähtyä pidemmäksi aikaa miettimään toteutusta, jotta luokkajako olisi vielä parempi eikä tulisi yhtään huonon näköistä koodia, joka johtuu siitä että joutuu kiertämään jonkun pienen asian minkä on toteuttanut jo aikoja sitten ja sen muokkaaminen olisi liian työlästä.

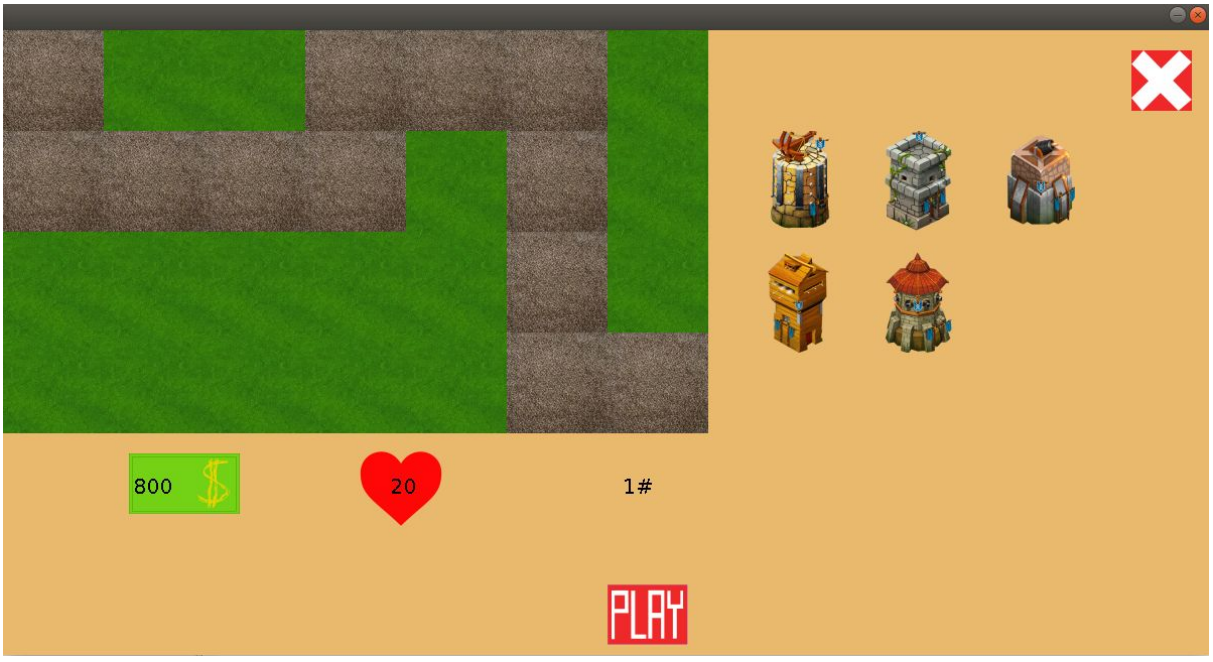
Jos aloittaisin uudestaan projektin tekemisen niin aloittaisin ensiksi käyttöliittymästä enkä torneista ja peli logiikasta. Käyttöliittymä oli vaikein osa ja toiminnalisuuden testaaminen on vaikeampaa ilman sitä. Pelin logiikka on myös helppo toteuttaa.

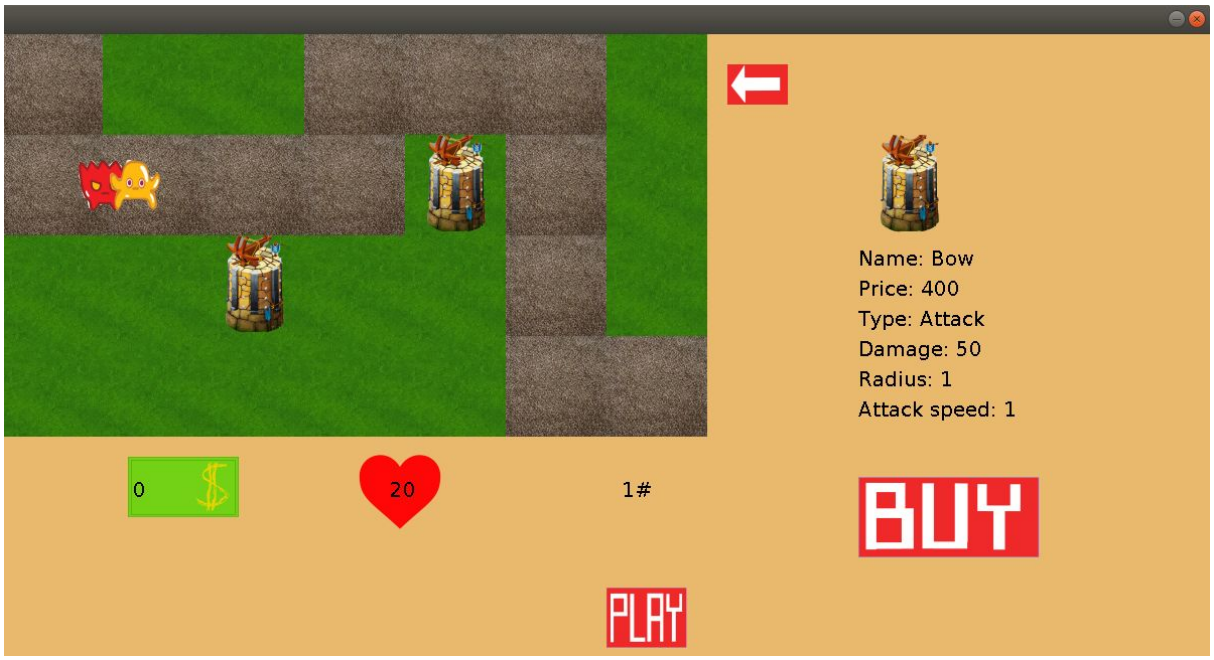
Viitteet:

StackOverflow, https://stackoverflow.com/
Scala Documentation, https://www.scala-lang.org/
Wikipedia
Ohjelmointistudio 2 materiaali
Java Documentation, https://docs.oracle.com/javase/7/docs/api/

Liitteet:


Tower Defense
Grassfield
Choose map


800
20
1#
PLAY


Name: Bow
Price: 400
Type: Attack
Damage: 50
Radius: 1
Attack speed: 1
0
20
1#
BUY
PLAY