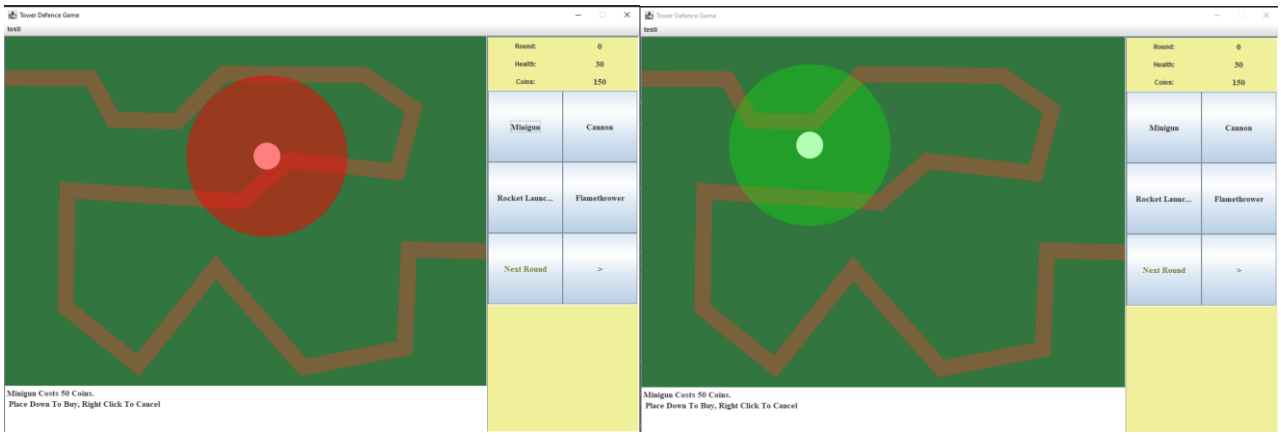
2. Yleiskuvaus

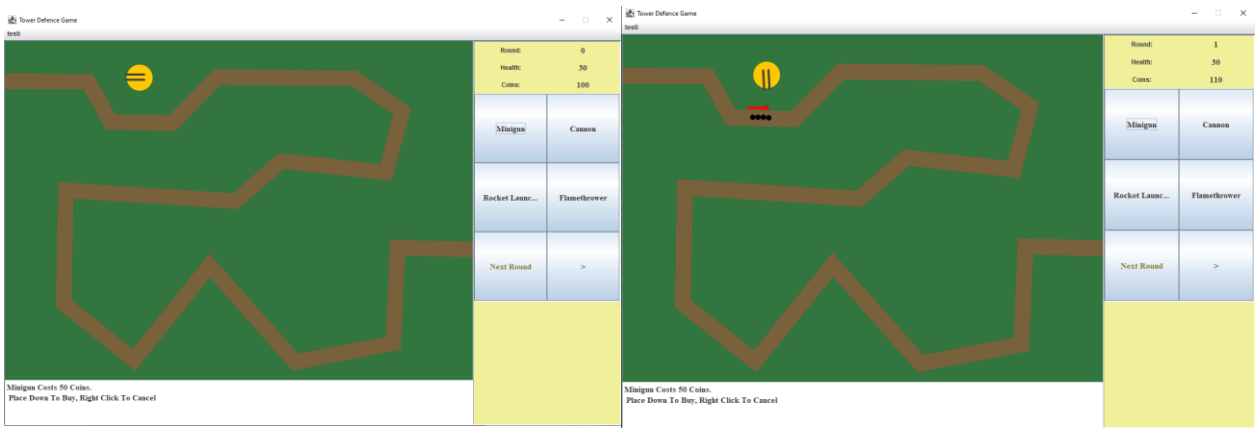
Projektityössä tein yksinkertaisen torninpuolustuspelin, josta löytyy kaikki genrelle tyypilliset peruspiirteet, kuten tasoittain, tiettyä reittiä pitkin kulkevat viholliset, sekä pelaajan asettamat, vihollisia ampuvat tornit. Peliä pelataan graafisesta käyttöliittymästä käsin ja erilaiset tornit ja viholliset toisistaan erottaa pääasiassa väri. Erikoisempia visualisointeja en peliin lisännyt. Suunnitelmasta lopputulos ei juurikaan eroa.

3. Käyttöohje

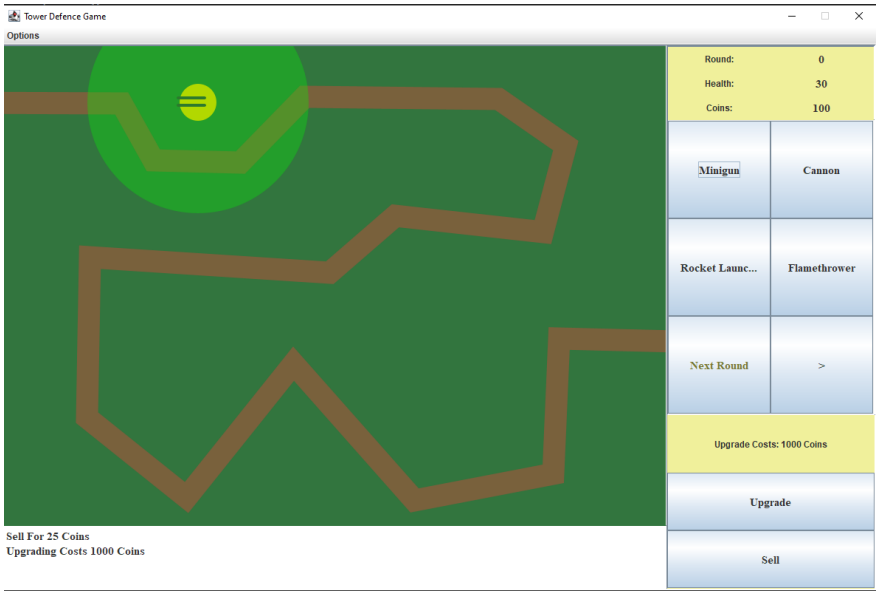
Ohjelma käynnistetään käynnistysoliosta ’GameLauncher’. Tämä luo uuden TowerDefenceGameOlion, joka käynnistää graafisen käyttöliittymän ja pelin. Pelaamiseen tarvitaan vain hiirtä. Pelin alussa pelaaja voi ostaa torneja, joista rahaa riittää kuitenkin vain kahteen eri vaihtoehtoon. Kun torni valitaan kaupasta, se leijuu pelaajan hiiren kohdalla. Tässä vaiheessa pelaaja näkee tornin ampuma-alueen, koon, sekä sen, pystyykö tornia asettamaan hiiren osoittamaan paikkaan. Tässä vaiheessa pelaaja voi vielä peruuttaa oston painamalla hiiren oikeaa näppäintä. Osto tapahtuu vasemmalla hiiren klikkauksella, jolloin peli lisää torniolion peliin ja torni ilmestyy kartalle ja tornin hinta vähentyy pelaajaan rahamäärästä.



Tornia ei voi asettaa vihollisen kulkemalle reitille, eikä toisten tornien päälle.



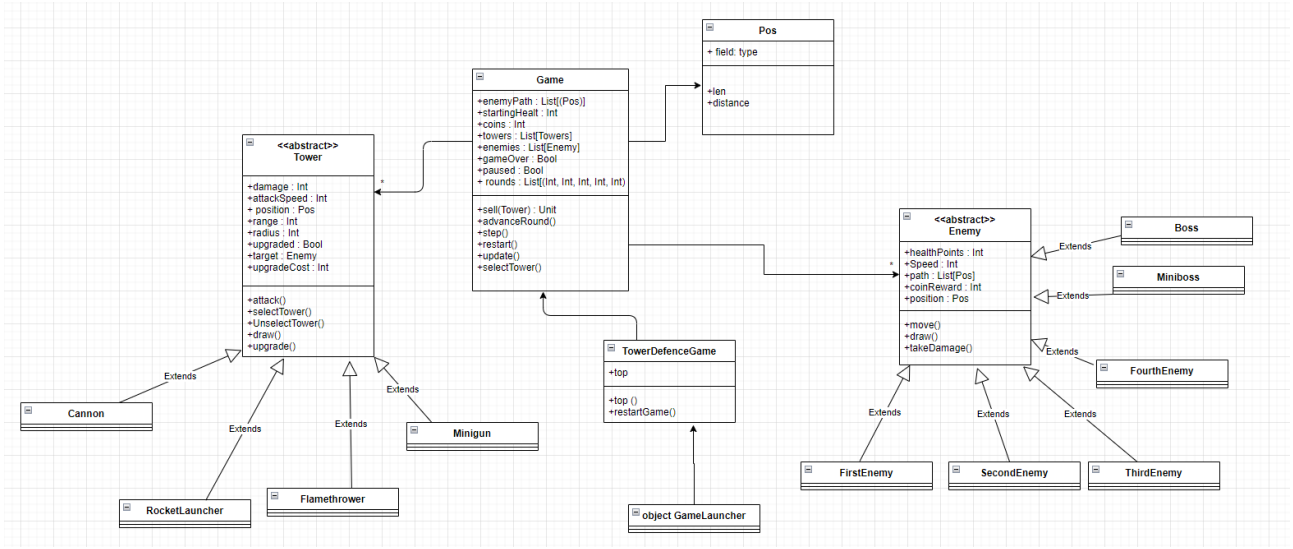
Kun pelaaja on valmis aloittamaan ensimmäisen vihollisaallon hän voi painaa ”Next Round” painiketta. Kuudes nappula antaa pelaajan vaihdella pelin nopeusta. Tämänlainen ominaisuus löytyy monesta torninpuolustuspelistä. Torneja pystyy myös ostamaan ja asettamaan kesken vihollisaallon,



Kun pelaaja on valinnut tornin, peli antaa vaihtoehdon myydä, tai päivittää valitun tornin. Tornin päivittäminen maksaa tornityypistä riippuen eri määrän rahaa. Myymisestä saa aina puolet torniin käytetystä rahamäärästä.

Pelissä on myös ”Options” valikko, josta voi aloittaa uuden pelin.

4. Ohjelman rakenne



Ohjelman keskeisin luokka on Game-luokka, joka pitää kirjaa peli-instanssin tapahtumista ja muuttujista, kuten rahamäärästä, meneillään olevasta kierroksesta, sekä pelaajan elämäpisteistä. Se

pitää myös kirjaa torneista ja vihollisista ja sisältää metodit tornien ja vihollisten lisäämiseen ja poistamiseen. AdvanceRound on yksi luokan keskeisistä metodeista. Se luo vihollliset rounds-listan mukaan seuraavalle kierrokselle ja aloittaa seuraavan vihollisaallon. Toinen luokan keskeinen metodi on step()- metodi, joka liikuttaa vihollisia ja torneja, ja tarkistaa vihollisten elämäpisteiden määrän, sekä tarvittaessa poistaa vihollisen pelistä. Se myös tarkistaa onko kierroksen kaikki viholliset tuhottu, eli kierros läpäisty.

TowerDefenceGame-luokassa luodaan pelin graafinen käyttöliittymä, joka kuuntelee hiiren painalluksia pelin aikana, ja ohjaa Game-luokalle käskyjä näiden painalluksien mukaan. Tässä luokassa on myös tickeri, joka päivittää käyttöliittymän näkymän, ja kutsuu Game-luokan step() ja update() metodeita 6ms välien. TowerDefenceGame-luokka piirtää myös sopivannäköisen kartan Game-luokan path muuttujan mukaan. Pelattavan ”kartan” muokkaaminen tai vaihtaminen on siis melko helppoa. Luokalla on käynnisobjekti GameLauncher, joka käynnistää pelin.

Towers luokka on abstrakti luokka. Towers-luokan attack()-metodi on toteutukseton ja eri alaluokilla on kokonaan erilaiset hyökkäystavat. Towers-luokalla on toinen abstrakti metodi upgrade, joka päivittää tornin funktiossa määritellyin arvoin.

Torneja on neljä erilaista, ja niillä kullakin on omat vahvuutensa. Minigun- torni ampuu keskipitkälle etäisyydelle jatkuvaa, pientä vahinkoa, joka osuu vain yhteen viholliseen. Cannon-torni ampuu harvakseltaan, paljon vahinkoja tekeviä laukauksia. Sillä on pieni ampumaetäisyys. Rocket Launcher-torni ampuu jopa kolmea vihollista samaan aikaan. Sillä on hidas hyökkäysnopeus, mutta se tekee suhteellisen paljon vahinkoa. Sillä on myös omalaatuinen hyökkäysalue. Flamethrower-torni ampuu kaikki sen ampuma-alueella olevia vihollisia tehden pientä vahinkoa pienellä alueella. Alaluokilla voi olla myös omia metodeja piirtämään yksinkertaisia ampumisanimaatioita.

Enemies-luokka muistuttaa hyvin paljon towers-luokkaa siten että sekin on abstrakti luokka, jolla on monta vain hieman toisistaan eroavaa aliluokkaa. Vihollistyypeistä en aluksi meinannut tehdä erillisiä alaluokkia ja vihollisen erilaiset tilastot olisi vain asetettu Game-luokan createEnemies-metodissa, mutta olisi estänyt vihollistyyppien piirtämisen erivärisiksi.

Pos-apuluokkaa käytetään monissa muissa luokassa positioiden käsittelemisen apuna.

5. Algoritmit

Vihollisten liikkumiseen käytetään algoritmia. Tämä algoritmi katsoo, mikä path listan pos-piste on viimeiseksi ohitettu, ja laskee suunnan sen, ja seuraavan pos-pisteen välillä. Sitten se vaihtaa enemyn positiota speed-muuttujan mukaan tähän suuntaan. Tämän jälkeen se tarkistaa, onko liikuttu seuraavan pos-pisteen ”ohi”, ja jos ollaan, niin se vaihtaa viimeiseksi ohitettua pos-pistettä.

Tornin ampumiskohde määräytyy myös algoritmisesti. Torniolio iteroi enemies-listan ja lisää kaikki rangen-etäisyydellä olevat vihollisoliot closestEnemy-listaan. Sen jälkeen se alkaa aiheuttamaan vahinkoa closestEnemy-listan ensimmäiseen vihollisolioon. Se siis ampuu aina ensimmäisenä ampumisalueellensa tullutta vihollista, kunnes tämä vihollinen on tuhottu tai poistuu ampumisalueelta. Se ei siis ammu lähintä vihollista.

Game-luokassa on metori createEnemies, joka luo viholliset rounds-listan mukaisilla määrillä. Rounds-lista kertoo kuitenkin vain 19-ensimmäistä kierrosta, jonka jälkeen vihollistyyppien lukumäärä määräytyy createRounds-metodin mukaan. CreateRounds-metodin olisi tarkoitus löytää pelaamisen kannalta haastava, mutta hauska määrä vihollisia kierroksen 20 jälkeisille vihollisaalloille. Tällä hetkellä funktio näyttää tältä:

```
def createRounds(currentRound : Int, towers : Buffer[Tower], healthPoints : Int, coins : Int) : (Int, Int, Int, Int, Int, Int) = {
//random "recipe", not optimized one bit
((currentRound * 50 + towers.length*2) / coins, towers.length*5,4*healthPoints * currentRound,currentRound*3,currentRound *2, (currentRound -20)*(currentRound -20))
}
```

Kukin outputin Int-arvo vastaa tiettyjen vihollistyyppien määrää.

6. Tietorakenteet

Vihollisia ja torneja pidetään scala.collections.mutable.Buffer()-tietorakenteessa. Niistä niitä on helppo lisätä ja poistaa.

Enemypath ja rounds ovat List-tyyppisiä, ja koska niiden kohdalla käytetään oikeastaan vain apply-metodia, voisi käydä järkeen käyttää jotain nopeammin indeksihakuja suorittavaa tietorakennetta, kuten arraySeq-tietorakennetta. Nämä listat eivät kuitenkaan ole kauhean pitkiä, joten suurta merkitystä ei ole.

Tornien attack()-metodissa olisi voinut käyttää jonkinlaista pinoa lähimpien vihollisten listana, jossa vaikka olisi halunnut tornin hyökkäävän lähimpään viholliseen, tai jonkun toisenlaisen hyökkäyskäyttäytymisen.

7.Tiedostot ja verkossa oleva tieto

Pelissä ei käsitellä tiedostoja tai lueta tietoa verkosta.

8. Testaus

Ohjelman testaus onnistui helposti graafisen käyttöliittymän avulla, eikä erillistä testiohjelmaa ole tarvinnut tehdä.

9. Ohjelman tunnetut puutteet ja viat

Vihollisten move()-metodi on oikkuileva. Välillä se kääntää viholliset liian aikaisin seuraavaa pistettä kohti, välillä liian myöhään. Tämä virhe voimistuu etenkin vihollisen nopeuden ollessa suuri, minkä voi huomata esimerkiksi nopeiden punaisten vihollispallojen kulkevan hiukan radan ohi. Enemypath-listan pos-pisteiden muuttaminen ei useimmissa tilanteissa riko vihollisten move-metodia, mutta jos pisteitä laittaa liian kauas toisistaan ja aiheuttamaan liian loivan käännöksen, voi vihollisten positio karata kauaksi niille tarkoitetusta reitistä. Oletan tämän johtuvan joistain move-metodissa tapahtuvissa pyöristysvirheistä, tai muista epätarkkuustekijöistä seuraavan path_posin määrittelyssä.

Toinen selvä vika on pelin huono optimointi pelattavuuden suhteen. En ole ehtinyt löytämään sopivia arvoja vihollisten määrälle, elämäpisteille, nopeudelle tai tornien vahingolle ja hinnoille saadakseni pelistä haastavan ja hauskan. Ainakin pelin nykykonfiguraatiossa loppupelissä päädytään tilanteeseen, jossa rahaa tulee valtava määrä, eikä viholliset anna tarpeeksi haastetta, tai niitä tulee liikaa.

Suurin ongelma, mihin olisin toivonut keksiväni ratkaisun, on vihollisten luonti. Viholliset luodaan nyt samanaikaisesti, ja ovat liian lähellä toisiaan. Olisin halunnut, että vihollisia tulee jatkuvana jonona, eikä niin ”rypälöityneinä”, mutta nykyistä createEnemies-metodia en onnistunut muokkaamaan toimimaan näin. Nykyinen tilanne johtaa siihen, että flamethrower-torni on liian ylivoimainen ja tekee heikoimmista vihollisista täysin turhia loppupelissä. Se aiheuttaa myös pientä lagia isojen vihollisryppäiden tuhoutuessa samanaikaisesti.

10. 3 parasta ja 3 heikointa kohtaa

Minulla oli scala swing kirjaston kanssa paljon pähkäiltävää, mutta uskon käyttöliittymän näyttävän siistiltä ja helppolukuiselta. Tykkään siitä, että ”karttaa” voi muuttaa helposti ja peli toimii silti normaalisti, eli viholliset kulkevat tätä uutta reittiä, kartta piirtyy oikein ja torneja ei pysty rakentamaan vihollisten reitin päälle.

```
if (new BasicStroke(3).createStrokedShape(drawMap(peli.enemyPath)).intersects(point.x-30, point.y-30, 60, 60)) {
  isPlaceable = false
}
```

Tätä pohtiessa jouduin selailemaan pitkään swing-kirjaston dokumentaatiota.

Tykkään siitä, että tornien rangen näkee jo ennen asettamista, ja rangealueen väri vaihtuu sen mukaan, onko torni asetettavissa vai ei. Pidän myös siitä, että torni kääntyy ampumaansa vihollista kohti, ja tykkään minigunin yksinkertaisesta, mutta tyydyttävästä ampumisanimaatiosta. Tykkään erityyppisistä torneista, ja etenkin RocketLauncher-tornin erityyppisestä ampuma-alueesta, joka rajoittaa sen sijoitusvaihtoehtoja.

Toteutuksessa on paikoittain heikkoja käytäntöjä, esimerkiksi GUI-luokan monet sisäkkäiset if-lauseet ja game luokan turhat apumuuttujat.

Torneja valitessa käytän apuna dictionarya joka mappaa eri tornityyppien vakioita arvoihin. Tämän avulla valitun tornityypin ”rangealue” pystytään piirtämään jo ennen kuin torniolio on luotu, mutta toteutus tuntuu silti kömpelöltä ja vaikeuttaa muokattavuutta. Ylimääräinen tietorakenne hankaloittaa uusien tornityyppien lisäämistä.

```
//info about towers in a map
val minigun = Map("Damage" -> 10, "range" -> 300, "cost" -> 50)
val cannon = Map("Damage" -> 50, "range" -> 120, "cost" -> 80)
val rocketLauncher = Map("Damage" -> 50, "range" -> 500, "cost" -> 250)
val flamethrower = Map("Damage" -> 2, "range" -> 150, "cost" -> 1500)
```

Tornien päivittämisen suhteen vastaan tuli muutamia komplikaatioita. Aluksi meinasin antaa jokaiselle tornityypille vain yhden päivityksen, ja loin totuusarvoa ilmaiseva muuttuja upgraded, joka muutetaan trueksi, kun torniolio päivitetään. Tämä osoittautui kuitenkin ongelmalliseksi, kun halusin lisätä kolmannen päivityksen minigun-luokkaan. Parempi keino olisi voinut olla antaa

jokaiselle tornille uusi attribuutti level, joka seuraa tornin tasoa. Upgraded-totuusarvo onkin muokattavuuden kannalta huono.

11. Poikkeamat suunnitelmasta, toteutunut työjärjestys ja aikataulu

Itse ohjelmoinnissa kesti noin 50-60 tuntia. Suurin osa työstä tapahtui viiden päivän aikana 6.4–8.4. Aloitin tekemällä käyttöliittymän perusosat. Tein sen jälkeen mahdollisuuden asettaa torneja kartta-alueelle, ja kartan piirtävän metodin. Seuraavat lisäykset olivat vihollisten move-metodi ja tornien attack-metodi. Sitten lisäsin round-mekaniikan. Tämän jälkeen lisäsin visuaalisia parannuksia ja muita torni- ja vihollistyyppejä, sekä pieniä parannuksia. Ajankäyttöarvio vastasi melko hyvin tapahtunutta.

12. Kokonaisarvio lopputuloksesta

Peli ottaa genren peruspiirteet hyvin haltuun. Se toimii, mutta ei ole hauska tai haastava. Vihollisten luomismetodin korjaus, ja peliarvojen tasapainottaminen tekisi pelistä hauskemman.

Ohjelman tämänhetkinen rakenne vaikeuttaa esimerkiksi päivitysten lisäämistä torneihin, ja jopa erilaisten tornien lisäämistä peliin. Pelikokemusta voi muuttaa helposti muokkaamalla karttaa tai vihollisaaltoja. Pelin peruspiirteet toimivat halutusti.

13. Viitteet

Scala swing dokumentaatio: https://www.scala-lang.org/api/2.12.0/scala-swing/scala/swing/