



# Protecting MOUNT EVEREST

## Projektityön dokumentti

**Aihe:** Tower Defense

**Nimi:** Petra Pyy

**Opiskelijanumero:** 593902

**Koulutusohjelma:** Teknistieteellinen kandidaattiohjelma

**Vuosikurssi:** 2016

24.4.2019

## **KUVAUS**

Protecting Mount Everest on Scalalla koodattu tornipuolustuspeli, jossa pelaaja pyrkii tuhoamaan viholliset ennen kuin ne saavuttavat puolustettavan alueen. Tässä turistit ovat vihollisia, jotka uhkaavat Mount Everestin huippua.

Pelissä turistit etenevät vuorella eri pelitasoissa telttailualueelta toiselle ja viimeisessä tasossa päämääräänä on vuoren huippu. Pelaajan tehtäväänä on estää eri tasolla turistien eteneminen, mikä onnistuu asettamalla turistien kulkureitin varrelle telttoja ja lumipallokoneita. Teltoista hyppii sherpojen mukiloimaan turisteja ja lumipallokoneet ampuvat hengenvaarallisia lumipalloja turistien niskaan. Pelaaja saa hoidettua tehtävänsä, jos hän pystyy eliminoimaan kaikki turistit ennen kuin yksikään niistä saavuttaa vuoristotien pään eli pelaajan itse päättämään tason lopun. Jos viholliset kuitenkin pääsevät tien pähän asti, pelaaja herättää vuoren raivon ja häviää tason.

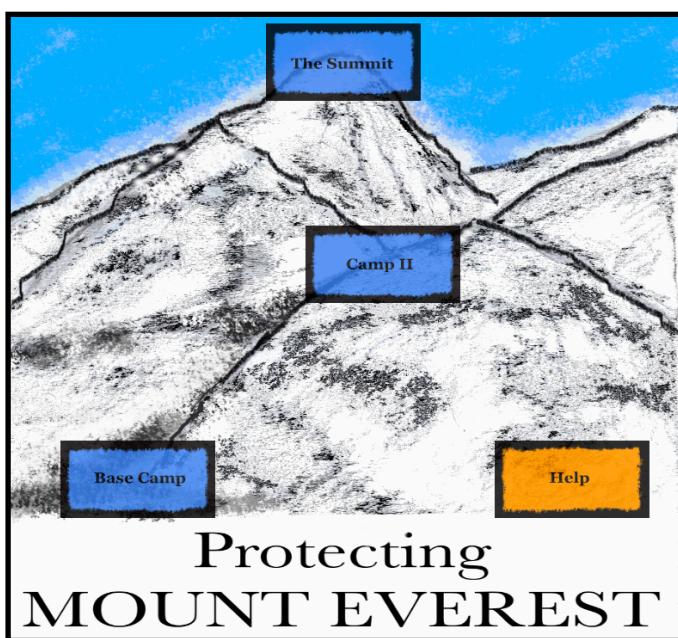
Pelin toteutus on vaikeustasoltaan vaikea, sillä pelin graafinen käyttöliittymä on tarkoin suunniteltu ja kuvat täysin omaa tuotantoa. Kaikki kuvat on piirretty käsin ja hiottu kuvankäsittelyohjelman avulla. Pelin laatua nostaa myös selkeät asetustiedostot, joita pelaaja voi muokata ja luoda sopivan pelikokemuksen. Pelaaja pääsee näiden tiedostojen kautta muokkaamaan tasojen vaativuutta, kuten esimerkiksi lisäämällä tai vähentämällä hyökkäysalojen määrää.

Pelin toteutus ei ole muuttunut huomattavasti suunnitelmaan nähdien. Tiedostojen sisältö eli ominaisuudet, joita tiedoston muokkaamisen kautta pystyy säätelemään, on kokenut kuitenkin pienä muutosta. Esimerkiksi hyökkäysalojen frekvenssiä ei pysty säätelemään, mutta aaltojen määrää ja vihollisten lukumäärää jokaisessa aallossa voi muokata. Pelaaja voi myös halutessaan muokata sherpojen ja vihollisten kävelynopeutta.

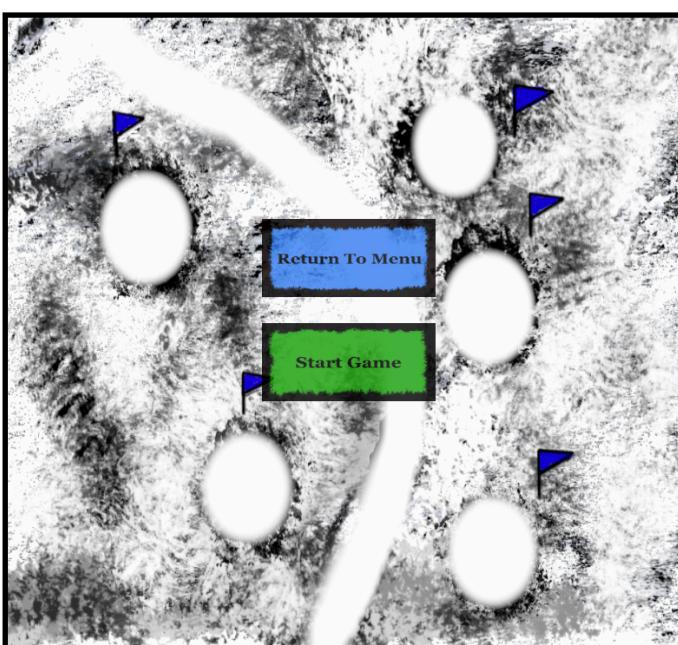
## KÄYTTÖLIITTYMÄ

Pelin käyttöliittymä on graafinen, ja se kommunikoi käyttäjän kanssa kuuntelemalla käyttäjän hiiritoimintaa. Kaikki vihollisten ja puolustusjoukkojen teot eivät ole reaalialjassa säädetäväissä, mutta asetustiedostoihin laitetaan kaikki tarvittavat tiedot hahmojen toimintoja varten ja ne luetaan aina joka tason alussa. Peli saa tietoja hiiren toiminnasta Processing 2 -kiraston tarjoamilla mouseEvent -toimintoilla.

Peli ajetaan graafisen käyttöliittymän kautta eli projektin src -kansiosta etsitään gui - paketti ja ajetaan sen sisältämä View -luokka Scala -applikaationa. Eclipsen kautta tämän voi suorittaa painamalla vasenta hiiren painiketta View -luokan kohdalla ja valitsemalla valikosta Run As -> Scala Application. Tarkista vielä, että kaikki projektin kirjastot (mainittu dokumentin lopussa) ovat käytössä ennen kuin yrität ajavaa ohjelmaa. Projektin juuressa olevat kirjastojen jar -tiedostot ja Scala Standard Library (2.12.3) täytyy olla yhdistetty projektiin, jotta ajo onnistuu.

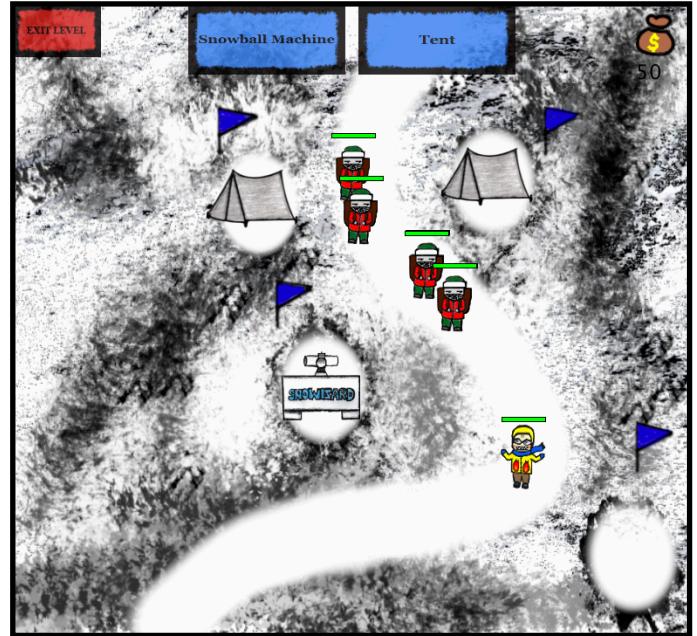


Pelissä on päävalikko (kuva vasemmassa), joka nähdään heti ensimmäisenä kun pelin graafinen käyttöliittymä avataan. Tästä valikosta pelaaja voi valita tason ja aloittaa pelaamisen. Tasuja on tarjolla päävalikossa kolme ja niiden vaikeustaso kasvaa alhaalta ylöspäin alkuperäisten asetustiedostojen arvoilla. Tasot ovat nimeltään Base Camp, Camp II ja The Summit. Päävalikosta löytyy myös Help -nappi, jonka takaa pelaaja voi löytää yksinkertaiset peliohjeet. Help -näkymästä pääsee takaisin etusivulle klikkaamalla näkymää.



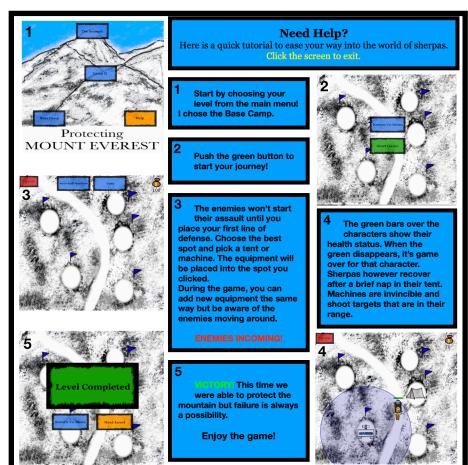
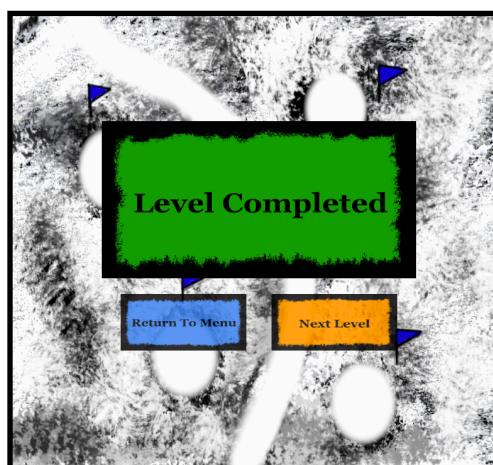
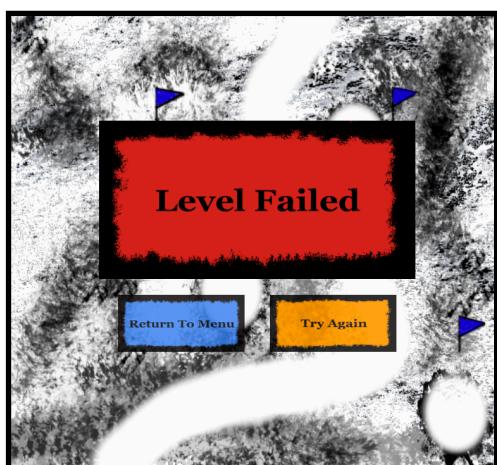
Peli ei kuitenkaan heti lähde käyntiin pelaajan klikatessa tasonappulaa. Ensiksi ruudulle tulee näkyviin valikko (kuva vasemmassa), josta pelaaja voi vielä palata päävalikkoon tai aloittaa pelaamisen. Valikko mahdollistaa tason esikatselun, koska tason karttakuva ilmestyy taustalle, ja antaa pelaajalle vielä mahdollisuuden tarkastaa valinneensa oikean tason.

Pelitasot ovat vuoristoreiteistä koostuvia taistelukenttiä (kuva oikealla tasosta Camp II). Jokaisen reitin varrella on sinisillä lipuilla merkittyjä paikkoja, joihin pelaaja voi asettaa telttoja tai lumipallokoneita. Tämä tapahtuu klikkaamalla hiirellä tyhää paikkaa ja valitsemalla pelin yläreunaan ilmestyvistä vaihtoehdista toisen. Käyttöliittymä prosessoi napinpainalluksen ja kyseinen puolustusväline asetetaan valitulle tyhälle rakennuspaikalle. Klikkaamalla jo asetettua lumipallokonetta voi saada näkyviin ampumasäteen, jonka sisällä sen laukomat lumipallot osuvat vihollisiin. Udestaan klikkaamalla konetta voi piilottaa säteen.



Pelin aikana pelaaja näkee jatkuvasti rahatilanteensa ruudun oikeassa yläkulmassa. Hän pystyy myös tarkkailemaan tiellä kävelevien turistien ja sherpojen terveydentilaan hahmojen päiden päällä leijuvista terveyspalkeista. Ruudun keskikohdille ilmestyy napit tarjolla olevista puolustusvälineistä, ja ruudun vasemmassa yläkulmassa näkyy poistumisnappi.

Pelin päätyttyä avautuu loppuvalikko, josta pelaaja voi valita palaavansa päävalikkoon tai jatkavansa pelaamista. Tason loppuessa häviöön pelaaja voi yrittää tasoa uudestaan ja voittaessaan tason pelaaja pystyy siirtymään seuraavalle tasolle. Viimeisellä tasolla loppuvalikosta pystyy ainoastaan siirtymään takaisin päävalikkoon. Kuvat alla ovat loppuvalikoista (pelin loppuessa häviöön ja voittoon) ja Help -näkymästä.



## **RAKENNE**

Ohjelma on jaoteltu kolmeen pakkaukseen, jotka sisältävät graafisen käyttöliittymän luokat, pelin ydinluokat eli päätoiminnot ja testiluokan. Kokonaisuudessaan ohjelman muodostaa yhdeksäntoista luokkaa (kuva päivitetystä luokkadiagrammista rakennekuvausen jälkeen). Nämä luokat ovat View, Level, DataProcessor, Coordinate, Character, Sherpa, Enemy, Wave, Tent, SnowballMachine ja LevelButton. Testiluokka ei kuulu näihin peliluokkiin, joten sitä käsitellään tässä dokumentissa vasta testauksen yhteydessä.

View -luokka toimii graafisena käyttöliittymänä ja suorittaa pelaajan hiirellä antamien komentojen toteuttamisen sekä tunnistamisen. Processing 2 -kirjaston MouseEvent -luokka tarjoaa mouseClicked -metodin, jonka avulla käyttöliittymä saa tiedon pelaajan hiiritoiminnasta ja antaa mahdollisuuden reagoida siihen. View-luokka myös sisältää asetukset peli-ikkunan piirtämiselle draw -metodilla ja skaalaikselle setup -metodin avulla. Tämän lisäksi pelitasojen ominaisuuksien piirtäminen tapahtuu View -luokan sisällä drawLevel -, drawMachineRange - ja drawHealthBar -metodien avulla. Metodit piirtävät näytölle näkymät muuttujien gameOn, mainMenuOn ja helpTextOn avulla. Luokan sisään on upotettu myös toinen luokka, LevelButton, joka tekee kuvista sävyä muuttavia nappeja kun hiiri on niiden yllä. LevelButton -luokan koko on pieni ja sen metodit tarvitsevat toimiakseen Processing 2 -ominaisuksia, joten se sijoitettiin lopulta View -luokan sisälle jossa kyseinen kirjasto on paljon käytössä.

Käyttöliittymän toiminta riippuu tason eli Level –luokan tiedoista, koska kaikki pelitasojen ominaisuudet määritetään siellä ja View –luokka toteuttaa niitä vastaavan kokonaisuuden pelinäkymään. Level –luokka sisältää paljon pelinkululle tärkeitä muuttujia. Luokasta löytyy tasokartan vapaat rakennusalueet freeSpots –muuttujasta, kaikki pelitason käytössä olevat lumipallokoneet machines –muuttujasta ja teltat tents – muuttujasta. Tasokartan vihollisille suunniteltu kulkureitti löytyy myös luokan path –muuttujasta, pelitilanne eli onko taso suoritettu nähdään isFinished –muuttujasta ja muuttuja money sisältää pelaajan rahat.

Level –luokan toimintaa varten ovat elintärkeitä Wave –luokka, Coordinate – luokka, Character –luokka, Tent –luokka ja vielä SnowballMachine –luokka. Hyvän tason muodostumista varten tarvitaan jokaisen luokan ominaisuuksia, koska ilman loogisesti jäsenneltyjä hyökkäysalustoja, hahmoja, puolustusvälineitä ja pelikoordinaatteja ei saataisi aikaiseksi selkeää tornipuolustuspeliä. Täten Level –luokalla on paljon metodeja näiden luokkien käsitteily varten. Tason syntyessä sinne asetetaan hyökkäysalot asetustiedoston ohjeiden mukaan, jolloin Wave –luokan ominaisuudet otetaan käyttöön Level -luokan addWave –metodin kautta. Wave -luokkien sisältämien hahmojen piirtäminen ajoitetaan

latestWaveStarted -muuttujan avulla. SnowballMachine – ja Tent –luokat välitetään Level –luokalle myös, koska tasoille tuodaan näiden luokkien esiintymää addMachine – ja addTent – funktioiden avulla. Character –luokan yhteys Level –luokkaan on myös määritetty kaikkien hahmojen terveydentilan tarkailua varten. Eri hahmojen ja laitteiden sijainnista tasokartalla vastaa Coordinate –luokka ja se välittää Level –luokalle tiedot peliikkunan x- ja y-koordinaateista sekä sisältää laskufunktioita muun muassa koordinaattien välisien etäisyysien määrittämistä varten.

Coordinate –luokka on tekemisissä melkein kaikkien luokkien kanssa, koska hahmon ilmestyessä peli-ikkunaan sille pitää antaa muun muassa aloituskoordinaatit ja puolustusvälineet on asetettava niitä varten piirrettyihin pisteisiin tasokartalla. Coordinate -luokan ensisijainen tehtävä koko pelikokonaisuudessa onkin mahdolistaa looginen liikerata ja sijoitus karttakuvan sisällä. Kuten aikaisemmin on mainittu, Coordinate – luokan sisältä löytyy funktoita erilaisia laskutoimituksia varten, jotka auttavat hahmoja löytämään hyökkäyksen kohteeksi ja liikkumaan oikeaan suuntaan pelitasoissa. Erityisesti length –metodia voidaan käyttää välimatkan määrittämistä varten kun sherpa löytää vihollisen vuoristotiellä ja normalize -metodi toimii hyvin suuntavektorin pituuden säätelyssä.

Wave –luokka sirottelee Enemy –luokan esiintymää pelikentälle hyökkäysaltoina. Sillä on tärkeä metodi addEnemy, joka nimensä mukaisesti lisää aaltoon vihollisia eli enemies – muuttujan puskuriin Enemy –luokan esiintymää. Wave –luokan sisällä on myös isOver – metodi, joka mahdolistaa hyökkäysaallon tilan tarkistamisen ja uuden aloittamisen. Wave –luokka on yhteydessä Level –luokkaan, koska pelitaso tarvitsee toimiakseen tiedon hyökkäysaalloista, vihollisten ominaisuuksista asetustiedostojen ohjeiden täytäntöönpanoa varten ja aaltojen vihollisista tuodakseen ne View –luokalle piirrettäväksi.

Sherpa- ja Enemy- luokat laajentavat Character –luokkaa ja sisältävät hahmoille ominaisia hyökkäys-, liike- ja terveydentilan metodeja. Sherpoilla ja vihollisilla on paljon samanlaisia metodeja. Molemmilta löytyy direction – metodi, joka kertoo mihin suuntaan hahmon tulee liikkua saavuttaakseen päämääränsä. Vihollista voi tähystää Sherpa –luokan findTarget –metodin avulla ja Enemy –luokan vastaava metodi pyrkii viemään vihollista määritettyä vuoristotietä pitkin. Muuttuja target sisältää aina lähimän vihollisen, jos sellainen lähipiiristä löytyy. Molempien luokkien käytössä on myös startHealth –muuttuja, joka sisältää hahmon terveyspisteet tämän ilmestyessä peliin mukaan. Hahmoilla on metodi action ja sen toimintaa määrittää muuttuja actionPerformed, jotka prosessoivat jatkuvasti hahmon seuraavaa liikettä pelissä. Kaikki hahmojen ja jopa lumipallokoneiden

toiminta pelikentällä määritetään kutsumalla drawLevel -metodin sisällä näiden luokkien action -metodeja, mikä tekee siitä erittäin tärkeän metodin pelinkulun kannalta.

Muutama eriävä metodi kuitenkin luokilta löytyy. Enemy –luokalla on dropMoney –metodi, joka lisää aina vihollisen kuollessa sopivan omaisuuden pelaajan käyttöön. Sherpa –luokalla on yhteys Tent –luokkaan, koska pelikentälle tuodaan aina sherpät teltan sisällä. Sherpa –luokassa esiintyy täten hahmon kotiteltan tiedot tent –muuttujassa. Sherpät tarkailevat horisonttia vihollisten varalta metodilla targetIsClose, joka kertoo onko vihollinen tarpeeksi lähellä sherpaa hyökkäystä varten. Sherpa -luokan aivan omat metodit patrol ja guardTent ovat puolustustoimintaan optimoituja metodeja, joiden tehtävä on auttaa hahmoa tarkailemaan ympäristöä, lepäämään ja pysymään telttojen läheisyydessä.

Character –luokan muuttujat damage, health ja speed periytyvät Sherpa – ja Enemy – luokille. Näillä määritetään hahmojen voimakkuus eli kyky tuottaa vahinkoa, terveystilastot ja nopeus tasokartalla. Hahmoille myös määritetään metodit move ja attack, jotka siirtävät hahmoja eteenpäin ja suorittavat hyökkäyksen. Pelissä move –metodin tarkoitus on liikuttaa hahmoa jatkuvasti kohti päämäärää, minkä jälkeen liikkuminen pysäytetään ja suoritetaan joku toinen kutsu (esim. attack). Lisäksi sherpalla ja vihollisella periytyy Character -luokan kautta metodi alive, joka kertoo onko hahmo vielä käytettävissä pelissä ja onko hahmon health –muuttujan arvo enemmän kuin 0.

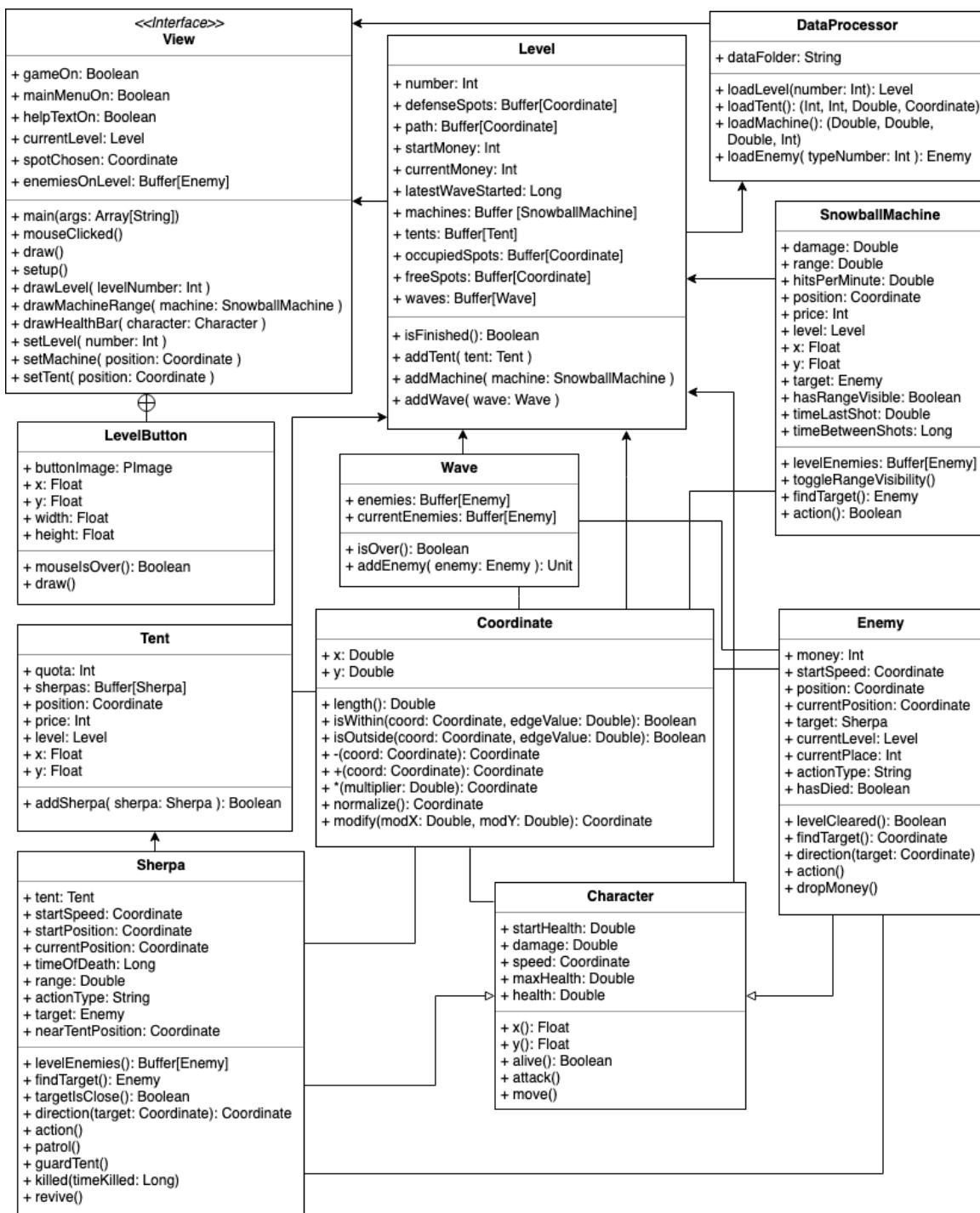
Tent –luokka on tärkeä Sherpa – ja Level –luokalle, koska se välittää tasolle teltan sisältämien sherpojen määrään ja ominaisuuksiin. Teltalla on muuttujat price, joka pelaajan pitää maksaa sijoittaakseen teltan sherpoineen pelikentälle, ja sherpas eli teltan sisältämät Sherpa –luokat puskurissa. Tärkeä Tent –luokan metodi on addSherpa, koska se lisää teltan sisään sherpaa puolustustöihin. Tent –luokan muuttuja on myös level ja position, jotka esiintyvät SnowballMachine –luokassa. Näitä tarvitaan sijoittamaan teltat ja koneet oikeisiin pisteisiin oikeissa pelitasoissa.

SnowballMachine –luokalla on näiden muuttujien lisäksi vihollisien löytämiseen metodi ja tallentamiseen muuttuja, findTarget ja target. Se osaa toimia pelikentällä action -metodin avulla ja pyrkii ampumaan hitsPerMinute -muuttujan ohjeistamaan tahtiin. Se ajoittaa ampumisensa timeBetweenShots -muuttujan avulla. Koneen ampumasäädettä voi tarkastella pelikentällä hasRangeVisible -muuttujan avulla, mikä auttaa pelaajaa arvioimaan sopivan säteen asetustiedostoja muokatessaan.

Viimeisenä luokkalistassa on DataProcessor eli asetustiedostojen luentaan ja käännyttämiseen erikoistunut yksikkö. Se käsittelee tekstitiedostoja, jotka löytyvät dataFolder -muuttujan ehdottamasta kansiossa projektin juuressa. Metodit loadLevel, loadTent, loadMachine ja loadEnemy lukevat kansion asetustiedostoja, jotka liittyvät tasojen,

telttojen, lumipallokoneiden ja vihollisten ominaisuuksiin. DataProcessor on yhteydessä View -luokkaan, sillä se pyrkii lukemaan sopivat tiedostot aina pelaajan aloittaessa käyttöliittymän kautta pelaamaan jotaan tiettyä tasoa. DataProcessor tallentaa tiedostojen kautta luetut ohjeet tason instanssiin ja tarjoaa sen käyttöliittymän käsiteltäväksi.

Lopullinen luokkarakenne on erittäin suoraviivainen ja looginen, koska jokaista selkeää kokonaisuutta varten on oma luokka ja tärkeä Level -luokka toimii kaikkien pelin keskeisten luokkien kokoajana. LevelButton -luokan olisi ehkä voinut sijoittaa selkeyden vuoksi käyttöliittymän eli View -luokan ulkopuolelle ja yhdistää saman paketin alla, mutta nappiluokka oli niin pieni ja yhtä riippuvainen Processing -kirjastosta kuin koko muu käyttöliittymä ettei siirrolla olisi ollut suurta arvoa. Koska päämäääränä oli pitää yhdessä selkeitä kokonaisuuksia, vaikuttaa lopullinen rakenne hyvin toteutetulta ja toimivalta.



## ALGORITMIT

Pelin tärkeimmät laskut tapahtuvat Coordinate –luokassa, jossa määritetään koordinaatit lähes kaikelle pelissä näkyvälle. Koordinaattien väliset laskut vaativat erittäin suoraviivaisia algoritmeja. Muun muassa kohinaa luodaan hyökkäysaallon tai sherpojen liikkeen eläväöittämiseksi lisäämällä x- ja y-koordinaatteihin satunnaisarvoja modify -metodin sisällä. Etäisyydet koordinaattien välillä voidaan suorittaa scala.math –kirjaston tarjoamalla hypot –metodilla, joka laskee etäisyyden kohdekoordinaattiin ja samalla vektorin pituuden. Normalize-metodilla taas suoritetaan yksinkertaista suuntavektorin suunnan vahvistamista eli vektori jaetaan sen pituudella, mikä helpottaa liikkumista pelin koordinaatistossa. Valitut koordinaattien algoritmit ovat suoraviivaisimmat ratkaisut, koska ne eivät vaadi erityisen hankalien metodien kokoamista.

Vihollisten liikkuminen ei vaadi vaikeita laskuja, koska asetustiedoston yhteydessä annetaan valmiit polkukoordinaatit hahmoille. Vihollinen liikkuu yksinkertaisesti pisteestä toiseen kunnes saavuttaa viimeisen koordinaatin. Sherpat taas tähystää lähietäisyydellä olevia polkukoordinaatteja ja liikkuvat niihin havaitessaan siellä sijaitsevan vihollisen. Coordinate –luokan algoritmit hoitavat hienosäädon ja muun muassa hahmojen kulkunopeuden lisäämällä hahmojen paikkakoordinaatteihin aina sopivan nopeuden muutoksen. Hahmon suunta määritetään vähentämällä kohdekoordinaateista hahmon omat koordinaatit ja luodaan normalize -metodin avustamana vektori, joka kertoo suunnan. Tämän vektori kerrotaan nopeusvektorin pituuden arvoilla. Seuraava tehtävä on lisätä vektorin ehdottama seuraava koordinaatti hahmon sijainnin koordinaattiin ja voidaan luoda liike haluttuun suuntaan sopivalla nopeudella. Nopeus päivitetään siten ettei se ylitä maksimaalista nopeusarvoa, jotka on annettu molemmille hahmoille.

Koska tiedostojen luennan ja graafisen käyttöliittymän metodit tekevät raskaimmat työt ohjelman pyörittämisessä, pelin monimutkaisemmat algoritmit voi löytää näissä tehtävissä käytettyjen Processing 2 –kirjaston ja Java –kirjaston metodien takaa. mouseClicked –metodin, image –metodin ja FileReader –luokan toimintaa voi tarkemmin silmällä kirjastojen omasta dokumentaatiosta.

## **TIETORAKENTEET**

Hyvin yleisenä tietorakenteena pelissä nähdään Buffer eli puskuri, joka on Scalan kirjaston mutable -tietorakenne. Tähän tallennetaan pääosin erilaisten luokkien esiintymiä, tekstiä String –muodossa ja mahdollisesti Int - tai Double -arvoja.

Puskuria on mahdollista päivittää ja laajentaa, mikä tekee siitä joustavan tietorakenteen ja erittäin helpon valinnan peliin. Puskurin sisältöä muutetaan pelin aikana usein tasojen vaihtuessa ja erityisesti erilaisten add –metodien yhteydessä on tärkeää, että rakenteeseen voi lisätä jatkuvasti uutta informaatiota. Se on todettu aikaisempien projektien myötä erittäin joustavaksi tavaksi tallentaa tietoja ja ohjelointikurssien ohella niiden kanssa on päässyt työskentelemään paljon, joten tietorakenteen valitseminen tuntui luonnolliselta.

Tason muodostuksessa tai koordinaattien tallentamisessa voisi myös käyttää muita tietorakenteita, kuten esimerkiksi Vector –rakennetta tai Array -rakenteita. Vector -rakenteen immutable -ominaisuus ei olisi ollut eduksi jatkuvan päivittämisen yhteydessä, mutta se olisi sopinut hyvin reitin koordinaattien kertaluonteiseen tallentamiseen. Array vastaavasti olisi sopinut kertaluonteiseen tallentamiseen, mutta siihen lisääminen tarkoittaisi aina uuden Arrayn luomista. Kuitenkaan yhtä pientä osiota varten en kokenut tarpeelliseksi käyttää erilaista tietorakennetta ja Buffer -tietorakenteen ominaisuudet olivat parhaat metodien toimintaan nähdyn.

## **TIEDOSTOT & TIEDOSTOFORMAATIT**

Pelin tiedostot ovat kuva- ja tekstitiedostoja, koska pelin käyttöliittymä on täysin graafinen ja tasojen ominaisuuksia muokataan asetustiedostojen kautta. Pelissä näkyvät kuvat eli kaikki napit, taustat, hahmot ja puolustusvälineet ovat käsin suunniteltuja. Piirrokset on hiottu kuvankäsittelyohjelmalla mahdollisimman tarkoiksi png. -kuvatiedostoiksi ja sijoitettu pictures -kansioon projektin juureen.

Pelin tekstitiedostot sijaitsevat myös projektin juuressa omassa data -kansiossa. Ne on tarkoitettu tasojen ja puolustustarvikkeiden asetustiedostoiksi, joita pelaaja voi muokata ja luoda omannäköisen pelikokemuksen. Tekstitiedostot ovat erittäin yksinkertaisessa plain text –muodossa (ASCII text). Asetustiedostoihin kirjoitetaan jokaiselle tasolle, puolustustarvikkeelle ja vihollistyyppille omat ohjeet. Tiedosto luetaan riveittäin, joten jokaisen rivin alkuun on laitettava muokattavan ominaisuuden nimi, lisätä sen jälkeen kaksoispiste ja kirjoittaa muokkausarvo. Ominaisuuden puuttuessa tulevat voimaan vakioarvot, mutta tason reitille ja puolustuspisteille täytyy aina antaa selkeät arvot. “-“ -merkki rivin alussa viestii kokonaisuudesta tietoja, jotka seuraavat tätä ja määrittävät tiettyjä arvoja esimerkiksi tason ominaisuuksille.

Tämän kappaleen lopusta löytyy kuvat asetustiedostoista, jotka pelaajalla on käytössä pelin aikana. Ensimmäisenä on kuva tason asetustiedostosta ja sitä seuraa vihollisten, telttojen ja lumipallokoneiden asetustiedostojen kuvat. Tiedostoista löytyy aina Setup -kokonaisuus, jonka jälkeen eri riveillä listataan muokattavat ominaisuudet jotka pätevät käsittelävissä olevan luokan instansseille. Esimerkiksi tason asetustiedostossa määritetään pelaajan käytössä oleva rahamääärä (“Money”) tason alussa, koordinaatit vihollisten kulkureitille (“Path”) ja puolustustarvikkeiden (“DefenseSpots”) asettamista varten. Koska koordinaattien keksiminen reittiä tai puolustuspisteitä varten on pelaajan näkökulmasta hieman haastavaa, pelissä on tulostuskutsu joka sylkee pelaajan klikkaamien pisteiden koordinaatteja Console -ikkunaan. Sieltä voi poimia koordinaatteja ja luoda niistä uuden reitin valitsemansa tason asetustiedostoon. Myös jokaisen pelitason hyökkäysaallon vihollismäärä ja vihollistyyppit on määritetty tason asetustiedostoissa Wave Settings -rivin jälkeen. Jokainen aalto (“Wave”) ilmoitetaan ja kaksoispisteen jälkeen kirjoitetaan vihollisen tyyppi eli numero joka löytyy Enemy -tiedoston nimistä (esimerkiksi Enemy 1) ja pilkun jälkeen laitetaan vihollisten lukumäärä. Kuvan asetustiedosto täten kertoo, että tasolle luodaan kolmen vihollisen aalto käyttämällä Enemy 1 -tiedoston asetuksia. Palautetun projektin data -kansiosta löytyy myös Enemy 2 -tiedosto, jonka avulla pelaaja voi kokeilla luoda eri ominaisuuksilla varustettuja vihollisaaltoja.

Lisää tiedostoja ja niiden ominaisuuksia näkyy seuraavan sivun kuvissa. Selvennyksenä vielä, että Tent -tiedoston ominaisuus Quota kuvastaa teltan sisältämää sherpojen määrää.

*Kuva tiedostosta Level 1:*

Base Camp Level Information

- Setup

Money: 100

DefenseSpots: 164, 247#281, 550#520, 152#559, 341#563, 621

Path: 374, 731#376, 720#381, 703#391, 679#403, 654#414, 627#425, 595#438, 569#448, 538#460, 487#460, 441#460, 417#457, 386#449, 361#442, 326#424, 283#402, 257#378, 227#346, 186#318, 163#276, 131#250, 109#214, 89#132, 25#113, 3

- Wave Settings

Wave: 1. 3

*Kuva tiedostosta Tent:*

Tent Information

- Setup

Quota: 4

Price: 20

- Sherpa Settings

Health: 100

Damage: 2.0

Speed: 1.3

*Kuva tiedostosta Enemy 1:*

Enemy Information

- Setup

Health: 100.0

Damage: 2.0

Speed: 1.1

Money: 15

*Kuva tiedostosta Machine:*

Machine Information

- Setup

Price: 40

Range: 250.0

Damage: 20.0

HitsPerMinute: 65

## **TESTAUS**

Pelin testaus aloitettiin hieman aikaisemmin kuin suunnitelmassa oli ajateltu, mutta suuri osa suunnitelmassa mainituista testattavista osista jäi ilman virallisia yksikkötestejä. Koska projektin kannalta tärkein ominaisuus oli asetustiedostojen luenta, luotiin hyvin alkuvaiheessa sille kaikki suunnitellut testit ja keskityttiin sen hienosäättämiseen. Olennaisinta pelissä on kuitenkin load -metodit ja asetustiedostojen käsittely, koska tasojen ominaisuudet ovat pelaajan muokattavissa ja riippuvat siksi lähes täysin asetustiedostojen sisällöstä.

Luokka EverestUnitTests sisältää tärkeimmät testitapaukset jokaiselle load -metodille ja jopa addMachine -metodille. Luokan jokainen testi on läpäisty toistaiseksi. Luokka sisältää yksikkötestejä, jotka on luotu JUnit -kirjaston avulla. Testit pääasiassa kutsuvat load -metodeita, jotka lukevat asetustiedostoja testData -kansion sisältä. Lopuksi luennan tuloksia verrataan odotettuihin arvoihin. Luokka sisältää myös addMachine -metodin testejä kaksi, koska teltan vastaava metodi toimi hyvin samalla tavalla ja sen metodia ei nähty tarpeelliseksi testata. Suurin syy myös testien pysähtymiseen tämän metodin kohdalla on ajanpuute, mutta projektin jatkokehityksessä pyrittäisiin parantamaan testikattavuutta ja vähintään lisättäisiin yksikkötestejä lopuille add -metodeille. Myös testaussuunnitelmassa mainitut metodit drawLevel, mouseClicked, findTarget ja attack jäivät ilman omia yksikkötestejä, mutta niiden testausta harjoitettiin tulosteiden avulla pitkin kehitysvaihetta.

Kun virheellistä tai hieman epäilyttää käytöstä ilmaantui pelissä, käytin myös usein Eclipse:n debug -työkalua. Epäilyttävät koodirivit oli helppo merkata ja seurata kutsuja kunnes ongelmarivit löytyivät. Tulostetestaus oli ehkä kuitenkin yleisin tapa, jolla koodin toimintaa testattiin. Testit toteutettiin tuotantovaiheessa asettamalla tulostuskäskyjä koodiin ja tarkastelemalla Eclipse:n Console –ikkunan tulostuvaa tekstivirtaa.

Tulostuneesta tekstillä pääteltiin koodin toimintaa ja pystyi korjaamaan nopeasti väärin toimivia kokonaisuuksia. Tulosteita sijoitettiin erityisesti molempien hahmojen move -metodin yhteyteen ja actionPerformed -muuttujan arvoja tulostettiin pelinkulun aikana.

Direction -metodin testaus suoritettiin pääosin tulostamalla arvot, jotka algoritmi antaa uutta suuntaa laskiessaan. Debuggaus onnistui helposti tulosteiden viestejä seuraamalla ja paikantamalla kohdat, joiden jälkeen tulosteiden arvot eivät vastanneet enää odotuksia.

Piirtämiseen erikoistuneet metodit olivat riippuvaisia erilaisten muuttujien arvoista, joita myös tulostin näkymiin hahmojen ja puolustusvälineiden asettamisen yhteydessä. Eri draw -metodit kuitenkin vaativat jatkuvasti silmämääräistä testausta graafisen käyttöliittymän kautta, koska kyseisiä metoduja on todella vaikea testata ilman visuaalista vahvistusta. Kokonaisuuden kannalta oli tärkeää nähdä ettei kuvat sijoitu epäloogisiin

paikkoihin ruudulla. Erityisesti tarkkailin käyttöliittymässä hiiren toimintaa, koska peli ottaa siltä vastaan paljon käskyjä pelin aikana. Oli tärkeää vahvistaa ettei reagointia klikkauksiin tapahtuisi turhaan, mikä johti visuaalisiin klikkaustesteihin eli erilaisten yllättävien ruudun osien klikkailuun pelissä. Hiiren kanssa tehtiin paljon tällaista manuaalista testausta pelin rakennuksen aikana ja tarkkailtiin kaikkien nappien toimivan oikein klikkauksen yhteydessä.

### **HAVAITUT BUGIT JA PUUTTUUVAT OMNAISUUDET**

Pelistä ei puutu mitään suunnitelmaan kirjattuja avainomaisuuksia ja ohjelmaan on jopa pystytty lisäämään joitain ominaisuuksia kuten koneen ampumasäteen piirtäminen. Myös pelaajan muokattavissa olevia ominaisuuksia on enemmän kuin suunnitelmassa on mainittu (muun muassa hahmojen nopeus ja lumipallokoneen ampumafrekvenssi).

Havaittuja bugeja on ikävä kyllä muutama, vaikka haluaisin nähdä koodini toimivan aivan täydellisesti. Yksi ajoittain ilmenevä bugi on hahmojen hetkellinen pysähtyminen ja taas suhteellisen nopeasti matkan jatkaminen. Liikkeen algoritmi välillä aiheuttaa sen, että hahmo haluaa ohjautua samaa koordinaattia kohti missä itse jo seisoo. Tämän huomatessani asetin pienen if -rakenteen suuntametodin yhteyteen, joka korjasi ongelman. Korjauksen yhteydessä tapahtuva koordinaattien muokkaus aiheuttaa kuitenkin nyt hahmon pieni mietintähetken kartalla. Tämän ilmiön voisi estää kitkemällä kokonaan algoritmista saman koordinaatin ilmestymisen. Vaihtoehtoisia korjauskeinoja olisi kohinan lisääminen algoritmiin kertomalla toista sijaintia aina jollakin vakioarvolla, jolloin saman koordinaatin ilmestyminen muuttuu melkein mahdottomaksi.

Usein kun pelaaja avaa pelin ja aloittaa pelaamisen jollakin tasolla voi pelaaja huomata että siirtyminen Start Game -napin painamisen jälkeen itse peliin kestää hetken. Tämän hetkellisen jäätymisen taustalla peli rakentaa tasoa ja tallentaa sen tietoja, jolloin luodaan pohja myös pelin muille tasolle saman istunnon aikana. Kyseinen pitkä lataushetki ei kuitenkaan toistu enää ensimmäisen pelin aloituksen jälkeen. Jos oikein aktiivisesti painaa hiirellä ruutua tämän latautumisen aikana, voi onnistua painamaan latautuvan tason (tällöin näkymättömiä) nappeja ja jopa asettamaan jo ennen ruudun päivitymistä puolustusvälineitä vapaisiin kohtiin. Kyseinen bugi esiintyi todella vaihtelevasti, koska lataushetken pituus saattoi riippua itse tietokoneen suorituskyvystä tai jostain muusta tuntemattomasta tekijästä ja aika tehdä näitä näkymättömiä nappivalintoja oli kuitenkin aina hyvin lyhyt. Kaikkea tietoa ei vältämättä tarvitse heti pelin alussa ladata muistiin ja tämän kautta voisi pyrkiä löytämään ratkaisun, joka nopeuttaisi tasojen latautumista.

Bugin syytä pitäisi myös tutkia lisää, koska se ei välillä ilmestynyt ollenkaan testauksen aikana.

Pelitilanteen aikana on nähtävissä myös yksi bugi, joka on ilmennyt erityisesti hahmojen nopeuksien ollessa eri. Vihollinen voi päästää sherpojen ohi pelissä ilman, että kukaan puolustusjoukoista ehtii aloittamaan hyökkäyksen, jos vihollinen liikkuu nopeammin kuin sherpät. Korjaaminen vaatii sherpän tähystyssäteen ja hyökkäyssäteen laajentamista reaktioajan parantamiseksi. Vihollisen voisi koodata myös väistämään sherpaa, jolloin se antaisi hieman lisääikaa hyökkäyksen valmisteluun.

### **3 PARASTA OMINAISUUTTA JA 3 HEIKKOUTTA**

Parhaisiin ominaisuksiin ohjelmassa kuuluu ehdottomasti graafinen käyttöliittymä eli kaikki kuvat (erityisesti upeat hahmot) ja pelaajan kyky navigoida helposti napeilla pelin sisällä. Kuvien tekemiseen on käytetty paljon aikaa, mikä nostaa niiden arvoa huomattavasti. Graafinen käyttöliittymä on pelin jalokivi, jonka draw - ja drawLevel - metodien toiminta syventää pelikokemusta. Erityisen ylpeä olen lumipallokoneesta, joka piirtometodien ansiosta selkeästi ampuu ruudulla jotain valkoista kohti vihollisia.

Toinen erinomainen ominaisuus pelissä on action -metodi, joka erityisesti Sherpa - luokassa paloittelee sherpän monimutkaisen puolustusjärjestelmän helposti ymmärrettäviin osiin. Se kokoaa siististi kaikki hahmojen toimintaan liittyvät metodit ja kutsuu niitä actionType -muuttujan arvon perusteella.

Heikoimpien ominaisuuksien kirjoon kuuluu tiedostojen luentaan erikoistuneet load - metodit. Erityisesti loadLevel on erittäin raskasta luettavaa, koska kaikki sen sisällä tapahtuva on yhdessä pitkässä pötkössä. Metodissa on myös turhan paljon try - ja catch - alueita. Toisaalta niiden tarkoitus on helpottaa asetustiedostojen lukua ilman pienien virheiden aiheuttamaa pelin jähmettymistä, mutta näitä osia koodissa olisi voinut jakaa pienempiin metodeihin ja keventää loadLevelin sisältöä.

Toinen heikkous pelissä on mouseClicked -metodi ja sen sisältämä epästabili if -viidakko. Metodi on myös turhan pitkä ja monimutkainen, jolloin välillä koodaaja itsekin hämmentyy. Tähänkin apuna olisi metodin jakaminen erillisiin metodeihin jokaista näkymää kohden, jolloin metodin sisällä voisi käyttää lyhyempiä if - tai match - rakenteita.

Viimeinen ohjelman heikkous liittyy pelin jalokiveen eli käyttöliittymään ja erityisesti hahmojen piirtymiseen eri tasoiissa. Hahmot joskus asettuvat päällekkäin yhteen pinoon ja pelaajan on vaikea erottaa, kuinka monta hahmoa kartalla liikkuu. Tähän auttaisi

tarkistusmetodi, joka tutkii missä sijaitsee jo hahmoja ja asettaa muut hahmot tietylle etäisyydelle jo varatuista koordinaateista. Tietyn säteen ylläpitäminen hahmon kuvan ympärillä on paras tapa välttää päälekkäinen piirtäminen, mutta toisaalta hyökkäystilanteessa hahmon pitäisi päästää lähellä toista. Täten metodin on myös priorisoitava piirtäminen eli sen pitäisi tarkastaa kuka hahmoista tulee piirtää kuvan alle eli on y -koordinaatistossa ylempänä.

### **TOTEUTETTU AIKATAULU JA EROAVAISSUDET SUUNNITELMASTA**

Projekti aloitus oli suunniteltu viikko ennen kuin todellinen työnteko lähti käyntiin. Tämä on vaikuttanut projektin myöhästymiseen suunnitellusta aikataulusta ja aiheuttanut tiiviimmän työskentelytahdin lähellä suunniteltua projektin palautuspäivää.

Projekti tekeminen alkoi graafisesta käyttöliittymästä suunnitelman mukaisesti vasta 28.2, jolloin pelin kuvat luotiin ja keskeiset kirjastot ladattiin. Taustalla tehtiin myös tutkimusta mahdollisista tornipuolustuspelin ominaisuuksista ja Processing 2 -kirjaston sisällöstä. Projektisuunnitelmassa arvioitu työskentelyaika tälle osiolle oli viikko, mikä osui aivan oikeaan.

Maaliskuun ensimmäisinä päivänä tehtiin ensimmäiset kunnolliset lisäykset projektiin ja luotiin piirrustusmetodit, jolla tuotiin ruudulle näkyviin alustava päävalikko ja ensimmäinen taso. Samalla hiiritoimintaa tarkkaileva mouseClicked -metodi luotiin reagoimaan napinpainalluksiin. Maaliskuun 7. päivä aloitettiin Sherpa -luokan hidan rakennus lisäämällä sille muuttuja, jotka liittyivät sen ominaisuuksiin pelissä ja löytyivät jo projektisuunnitelman luokkakaaviosta. Maaliskuun 10. päivä alkoi tiedostojen luennan työstäminen, ensimmäinen tason asetustiedosto luotiin ja DataProcessor -luokka lisättiin luokkarakenteeseen, mikä oli suunnitelmassa jo hieman ennakoitu päätös. Aikataulu aikaisemmassa suunnitelmassa sisälsi asetustiedostojen lukemiseen liittyviä tehtäviä maaliskuun 11. päivä lähtien, mutta ennakoiti myös projektin sisältävän jo melkein kaikki muut luokat. Tähän vaikutti varmasti työtahdin hitaus ja aloituspäivän siirtyminen, mutta suurin tekijä oli tarve tehdä tarkkaa työtä ja alusta asti pyrkiä dokumentoimaan projektia. Toiminnan testausta tehtiin tässä vaiheessa jatkuvasti joko silmämäärisesti navigoimalla pelin käyttöliittymässä tai luomalla JUnit -testejä haastavia metodeja varten. Tämä kuului myös alkuperäisen suunnitelman aikatauluun, mutta hidasti muiden osioiden valmistumista.

Maaliskuun 12. päivä toi uusia tuulia mukanaan ja aikataulussa siirryttiin yhdellä viikolla taas taaksepäin eli viikolle 1.3-10.3. Suunnitelman aikataulussa kyseiselle viikolle oli laitettu paljon eri luokkien rakentamista, mikä näin jälkeenpäin ajateltuna oli operaatio joka kesti kuukauden. Testaukselle oltiin aikataulutettu kaksi kokonaista viikkoa aikaa, jotka menivätkin täysin loppujen luokkien rakentamiseen. Tästä eteenpäin suunnitelman aikataulu ei mitenkään ollut toteutettavissa ja syntyi uusi aikataulu, joka keskittyi pelin toiminnan hiomiseen.

Lumipallokoneiden lisääminen oli seuraava looginen askel, koska asetustiedostojen luennan toimintaa haluttiin testata ja nähdä lumipallokoneen piirtyvän oikeaan pisteesseen. Alkuperäisessä suunnitelmassa olisi siirrytty jo vihollisten ja hyökkäysaaltojen ominaisuuksien rakentamiseen, mutta aikataulusta oltiin toisiaan viikko myöhässä. Maaliskuun 16. päivä oli tehty Coordinate -luokkaan tärkeitä puolustusvälineiden asettamiseen tarvittavia koordinaattimuuttuja ja vektorien etäisyyskiä laskevia metodeja. Maaliskuun 18. päivä välineiden asettaminen oli valmis ja asetustiedostot oli luotu sekä tasolle etä koneille. Tätä seurasi testien kirjoittaminen lumipallokoneiden asetustiedostoja lukevalle load -metodille, ja tasolle koneiden lisääminen testattiin myös tässä vaiheessa JUnit -testeillä. Tämän jälkeen luotiin Sherpa - ja Tent -luokan muuttuja, joiden ohella teltan ensimmäinen asetustiedosto luotiin ja sen luenta lisättiin mukaan load -metodien valikoimaan. Alkuperäisessä suunnitelmassa oli ajateltu projektin olevan pelikunnossa 19.3, mutta iso osa projektista oli vielä luomatta toisin kuin alkuperäinen suunnitelma oli ajatellut ja projektin aloituksen myöhästyminen hidastivat todella työskentelyä. Käsitys myös omasta ajankäytöstä ja muiden kurssien tehtävämääristä oli liian optimistinen, jolloin koodaustahti ei ollut niin tehokasta kuin olisi toivonut.

Level -luokan laajentamista jatkettiin maaliskuun loppua kohden onnistuneiden testien ja draw -metodin lupaavan toiminnan innostamina päivittämällä vihollisten ja hyökkäysaaltojen muuttuja. Enemy - ja Wave -luokat oli alkuperäisessä suunnitelmassa jätetty viimeiseksi lisättäviksi osiksi projektissa ja niin ne myös todellisuudessa otettiin kehitykseen mukaan aivan viimeisenä. Täten 28.3 niiden alustavat ominaisuudet lisättiin projektiin ja tasoihin. Aaltojen piirtyminen tasolle työstettiin mukaan vähitellen ja maaliskuun lopulla vihollisten ylle saatiin lisättyä myös terveyspalkit.

Huhtikuun alussa (7.4) päästiin lisäämään pelille tärkeimmät ominaisuudet eli hahmojen liikkumisen ja hyökkäämisen mahdollistavat metodit niiden luokkiin. Tätä seurasi oleellisen action -metodin luominen hahmoille ja lumipallokoneille, joka aina piti huolta oikean toiminnan tapahtumisesta oikeaan aikaan. Huhtikuun 13. päivä ja 16. päivä

sisälsivät paljon näiden metodien hienosäätiöä ja oikeiden kohteiden löytymistä tasolla hyökkäystä tai liikkumista varten. Tällöin lisättiin myös lopulliset napit navigaatiota varten, alettiin tarkastelemaan pelin testejä, metsästettiin bugeja ja viimeisteltiin pelin dokumentaatiota. Projektin aikataulu oli vihdoin takaisin raiteilla ja dokumentaatiota työstettiin viimeiset päivät.

Huhtikuun 18. päivä tehtiin viimeiset koodin ja dokumentaation siistimiset. Seuraavana päivän palautettiin projekti ja aikataulussa oltiin neljä päivää jäljessä suunnitellusta palautuspäivästä. Vaikka projektin aikataulu oli ollut aivan liian optimistinen ja sisälsi oletuksen taivaallisista superkoodausvoimista, ei lopulta palautuspäivä toteutunut niin lähellä deadlinea kuin pelkäsin.

## **LOPPUARVIO**

Projektin koodin jälki on tasaisen hyvää, koska kokonaisuudet on erotettavissa koodista järkevien muuttujien ja metodien nimien ansiosta. Aikataulun kanssa tappeleminen kuitenkin tuottaa aina joitain rumia ratkaisuja ja osa isommista metodeista jäi siksi paloittelematta pienempiin osiin. Etenkin tiedostojen luennan keskuudessa esiintyy todella pitkiä ja raskaita metodeja, joiden rakennetta muuttaisin ehdottomasti jatkossa. Näiden jaottelu pienempiin osiin tekisi koodista paljon miellyttävämmän lukea ja myös estäisi mahdolliset virheet luennassa.

Huomasin, että pieni tauko Scalan käyttämisessä aiheutti koodaustahdin hidastumisen paikoittain. Tietorakenteita piti uudelleen tutkia ja notaatiota tarkastaa usein, mitä en ollut ottanut aikataulussani huomioon. Aikatauluun oli kuitenkin varattuu aikaa hieman muulle opiskelulle, mutta juuri koodiin liittyvä tutkimustyötä ei helposti tajunnut priorisoida muiden kurssien yli.

Ehdottomasti kauneinta nähtävää koodissa on action -metodi ja sen siistit match-case -kerrokset. Samanlaista rakennetta olisin halunnut päästää käyttämään koodissa enemmän ja leikkaamaan sen avulla isoja metodeja paloiksi. Käyttöliittymän hahmot ovat varmaan action -metodin jälkeen suurin ylpeyden aihe, koska kuvat on tehty käsin ja vaivalla hiottu kuvankäsittelyohjelmalla.

Nykyisessä versiossa eniten häiritsee ettei aina pysty erottamaan hahmojen määrää kentällä ja välillä toinen sherpa kävelee toisen pään päällä. Jos projektia jatkokehittää, korjaisin heti ensimmäiseksi kaikki havaitut bugit ja varmistaisin hahmojen kuvien

piirtyvän erilleen. Muuttaisin piirtojärjestystä siten, että kauempana olevat hahmot piirretään ensiksi eli kuvat päätyisivät tällöin lähempänä olevien taakse. Lisäksi jatkossa piirtäisin peliin ehdottomasti lisää erilaisia hahmoja, tasokarttoja ja puolustuskoneistoja. Hahmoille voisi luoda myös omia luokkia ja muuttaa niiden käytöstä. Erityisesti olisi hauskaa kehittää sellainen hahmo, joka jahtaisi väsymättä vihollisia ympäri tasokarttaa. Samalla kun piirtometodeja parantelee pilkkoisin isoja metodeja myös pienempiin osiin. Tiedostojen luennan metodit erityisesti tarvitsevat tästä käsittelyä.

Tietorakenteet ovat nyt todella yksipuolisia (melkein vain käytetty Buffer -rakennetta) ja niiden valinta on pääosin liittynyt siihen, että ne ovat minulle ennaltaan tuttuja ja sopivat hyvin nykyiseen tarkoitukseensa. Kuitenkin tietorakenteiden metodit ovat jotkut nopeampia kuin toiset, mikä voisi olla hyvä ottaa huomioon enemmän valinnassa. Toisenlaisen tietorakenteen valinnalla voisi parantaa ohelman nopeutta ja vähentää eri näkymien latausaikaa.

Ohelman rakenne on suhteellisen selkeä ja sen laajentaminen tai muokkaaminen pitäisi onnistua helposti ymmärrettävän kokonaisuuden ansiosta. Ohelman laajentamista voi kuitenkin kuka vain pelaaja tehdä asetustiedostojen kautta eli hän voi muokata olemassaolevia tiedostoja tai lisätä aivan omia plain text -tiedostoja. Muun muassa erilaisille vihollistyyppille voi tehdä eri typpinumeroilla varustettuja tiedostoja, mutta tasojen asetustiedostossa täytyy muistaa käyttää tästä uutta vihollistyyppin numeroa jos haluaa lisätä sen hyökkäysaaltoon.

Jos aloittaisin projektin uudestaan, sopisin paremmat git -käytännöt. Commit -viestien sisältöä olisi voinut vielä parannella ja sopia käytäntö viestien muodosta, koska nykyiset viestit olivat paikoittain epäselviä ja sisälssivät todella paljon asiaa. Branchien tekeminen olisi auttanut isoja kokonaisuksien kanssa ja ohjannut keskittymään yhteen asiaan kerrallaan. Huomasin myös ettei tarpeeksi usein tajunnut tallentaa etärepolle uutta koodia. Nykyisen projektin kanssa helposti teki monta päivää töitä ennen kuin tajusi pitää breikin ja puskea koodia talteen. Tämä aiheutti usein sen ettei muistanut, mihin oli jäänyt aikaisempaan päivänä ja puski kerralla todella paljon muutoksia. Suunnitelman aikataulu onneksi toimi hyvinä viitekehysenä, jos tuli sellaisia hetkiä ettei heti osannut ottaa seuraavaa askelta. Suunnittelin myös omat viikon kestävät Sprintit varmuuden vuoksi ettei vain unohtuisi, mitä asiaa on työstämässä. Pieni Agilen -soveltaminen on varmasti suurin syy, miksi onnistuin tekemään projektin loppuun ja luomaan toimivan kokonaisuuden.

Tekisin myös aikataulun eri tavalla, jos aloittaisin projektin alusta. Pyrkisin jakamaan useammalle viikolle eri luokat ja pysyä todella laatimassani aikataulussa. Aikataulussa on vaikea pysyä, jos kerran on lähtenyt kulkemaan omia teitä. Heti alkuun on osattava ennakoida omaa työtahtia ja luoda omien kykyjen mukainen suunnitelma, mutta se on tietenkint helpommin sanottu kuin tehty.

## **LÄHTEET JA LINKIT**

Projektin lähdekoodi on nähtävissä ProtectingMountEverest -projektiin juureessa sijaitsevan src -kansion sisällä. Se on jaoteltuna eri pakettien sisään ja pakettien nimet antavat selkeästi ymmärtää, mihin alueeseen sisältyvä koodi vaikuttaa (“core” sisältää pelin ydintoiminnot, “gui” graafisen käyttöliittymän ja “tests” testit). Pelin kaikki kuvat ovat omaa tuotantoa ja ne sijaitsevat pictures -kansion sisällä.

Tornipuolustuspelin tehtävänannon yhteydessä A+ -sivuilla olevat ohjeet on otettu huomioon projektia tehdessä.

Projektissa on käytetty erilaisia ohjelmia ja kirjastoja, joiden dokumentaatiota käytettiin projektin rakentamisen aikana. Kirjastojen nimet ja linkit niiden dokumentaatioon:

[1] Processing 2.2.1 -ohjelman dokumentaatio ja sen core.jar -tiedoston ydintoimintoja,

<https://processing.github.io/processing-javadocs/core/>

<https://processing.org/reference/>

[2] Java SE 8 [1.8] -kirjaston dokumentaatio,

<https://docs.oracle.com/javase/8/docs/api/>

[3] Scala Standard Library 2.12.3 dokumentaatio ja math –paketin metodit,

<https://www.scala-lang.org/api/current/scala/math/index.html>

[4] JUnit 4.12 dokumentaatio testejä varten,

<https://junit.org/junit4/javadoc/4.12/overview-summary.html>