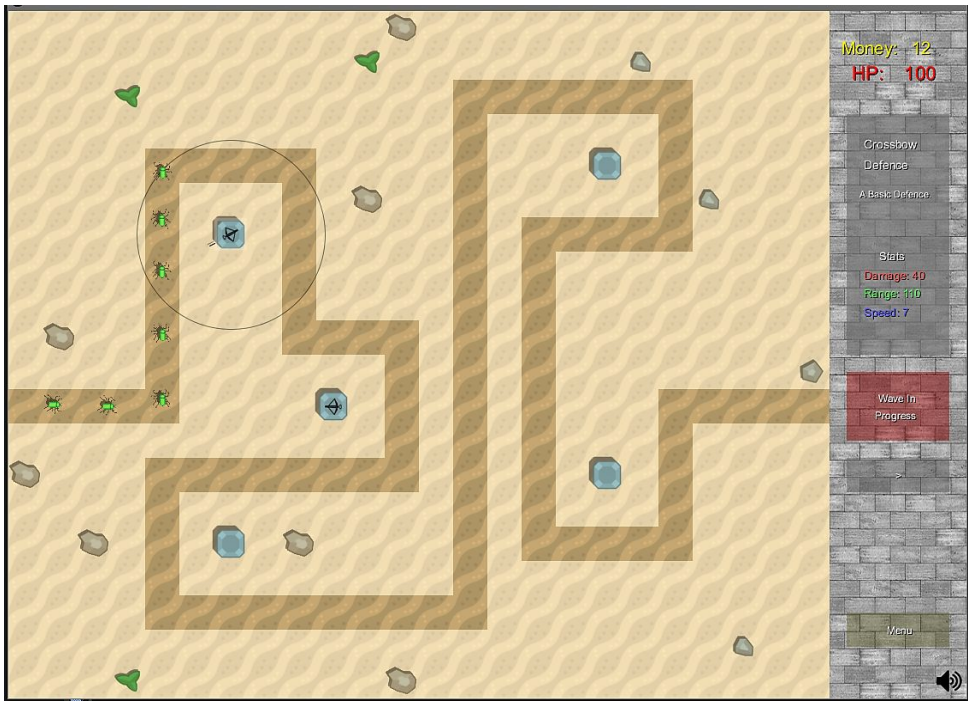
Tornipuolustus Projekti

Valtteri Kortteisto, 612698

Tietotekniikka, Vuosikurssi 2017

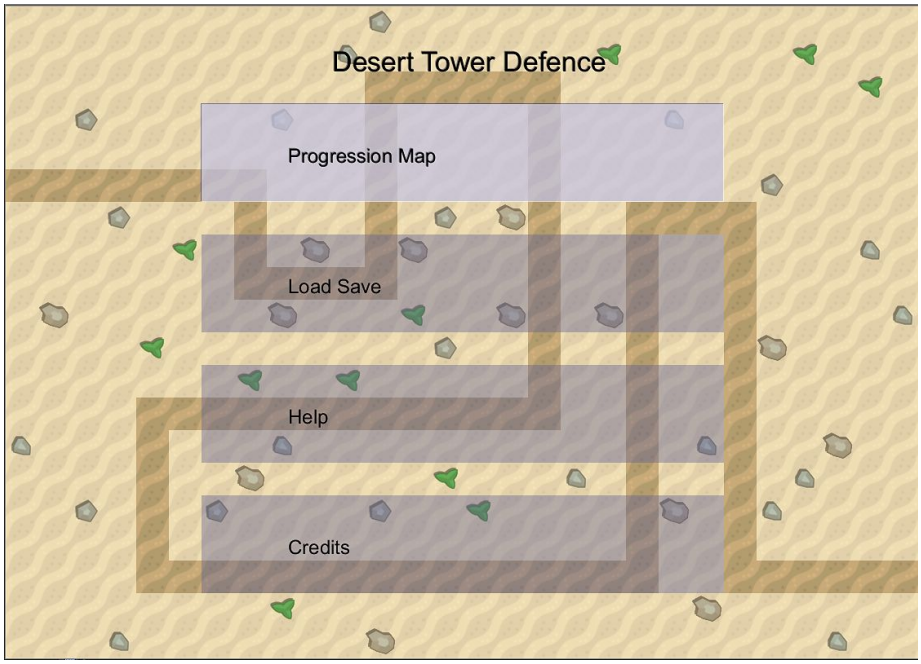
27.4.2018

Yleiskuvaus



Tornipuolustus peli, jossa on kaikki sellaiseen kuuluvat perusominaisuudet:

- Erilaiset viholliset tulevat aalloissa, ja seuraavat niille tehtyä reittiä
- Jos viholliset pääsevät reitin loppuun, pelaaja menettää hp:ta
- Jos pelaajan hp loppuu, peli päättyy
- Torneja pystyy rakentamaan minne tahansa missä on tyhjää
- Torneihin on valittavissa kolme erilaista puolustusta
- Kun torneja tai puolustuksia ostaa, niistä pitää maksaa
- Puolustukset ampuvat vihollisia kunhan ne ovat sen etäisyydellä
- Pelaaja saa rahaa vihollisten tappamisesta
- Kun kaikki vihollis aallot on hoideltu, pelaaja voittaa pelin



Lisäksi on myös:

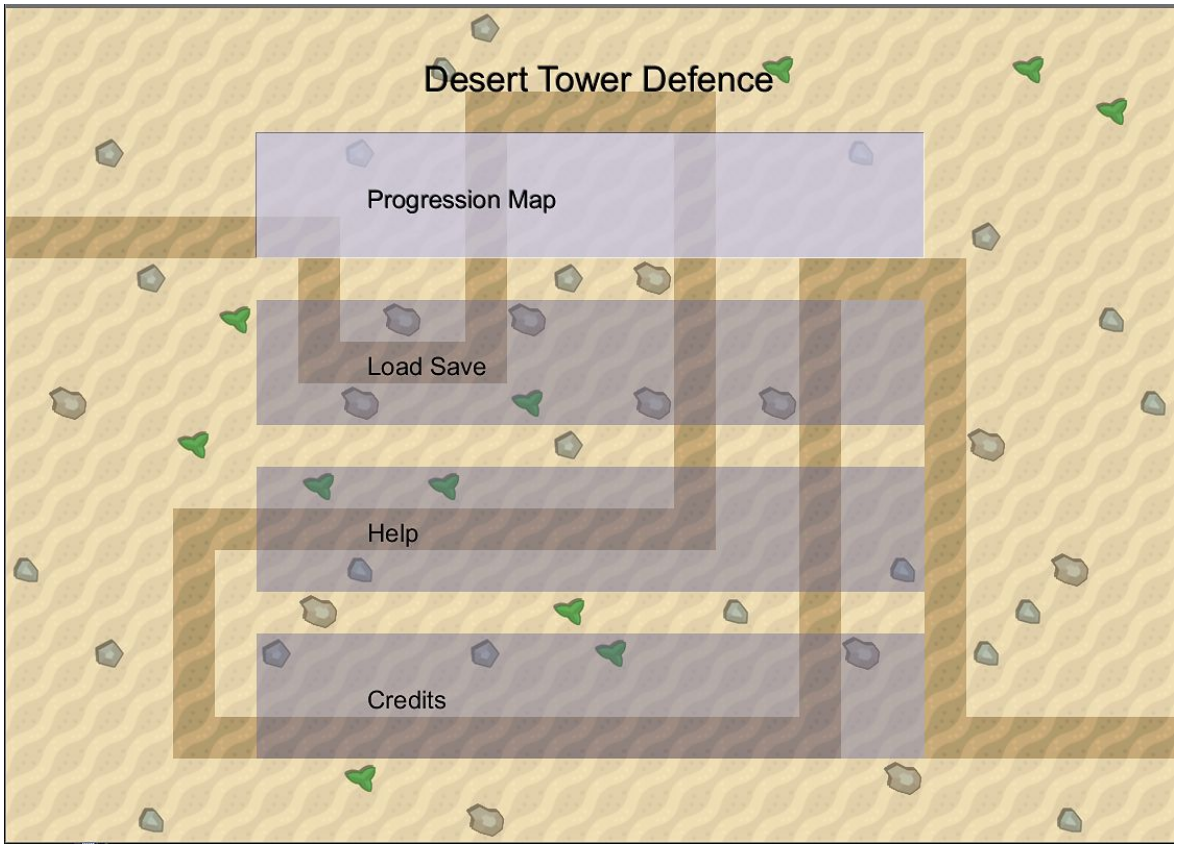
- Aloitus menu, jossa voi lukea apuja tai laukea pelin tekemiseen vaadituista asioista, sekä valita tallennus slotin josta lataa oman pelin progression
- Pelin pisteytys, joista parhaat pisteet tallennus pitää myös muistissa
- ‘progressio kartta’, jossa leveli valitaan kartasta
- Äänet, ja mahdollisuus asettaa äänet pois
- Mahdollisuus parannella ostettuja puolustuksia
- Tietoa defensseistä ja mistä tahansa muusta saa viemällä hiiren sen päälle
- Mahdollisuus nopeuttaa pelin kulkua 4x nopeammaksi
- Mahdollisuus pausettaa peli ja palata pää menuun, tai aloittaa leveli alusta

Menin suunnilleen suunnitelman mukaan muilta osin paitsi siinä, miten asioita ostetaan kentälle. Sivu menusta ei valita ostettavia asioita, vaan suoraan kentällä tapahtuu kaikki ostot, mikä tekee asioiden ostamisesta huomattavasti kätevämpää. Suunnitelmassa en myöskään ollut ajatellut monia asioita, jotka päätin kuitenkin toteuttaa, esim. alku menu, pelin tallennus, pausetus ja nopeutus.

Mielestäni työ on toteutettu haastavana, syystä että toiminnallisuutta on hyvin paljon, kokonaisuus on selkeä, ja peli on helposti ymmärrettävä ja toimii kuin pitää.

Käyttöohje

Ohjelma käynnistyy ajamalla Desert Tower Defence/src/user_interface/Game.scala Scalaohjelmana. Siitä eteenpäin ohjelman käyttäminen on aikalailla itsestään selvää, mutta kerron kuitenkin.



Ohjelma avautuu pää menuun, jossa käyttäjällä on neljä vaihtoehtoa:

Progression Map

-> Progression mapin kautta valitaan leveli jota halutaan pelata

Load Save

-> Voit ladata yhden neljästä tallennuksesta, joissa on pelin edistyminen ja pisteet Tallenettuna, nämä näkyvät taas Progression Mapissa

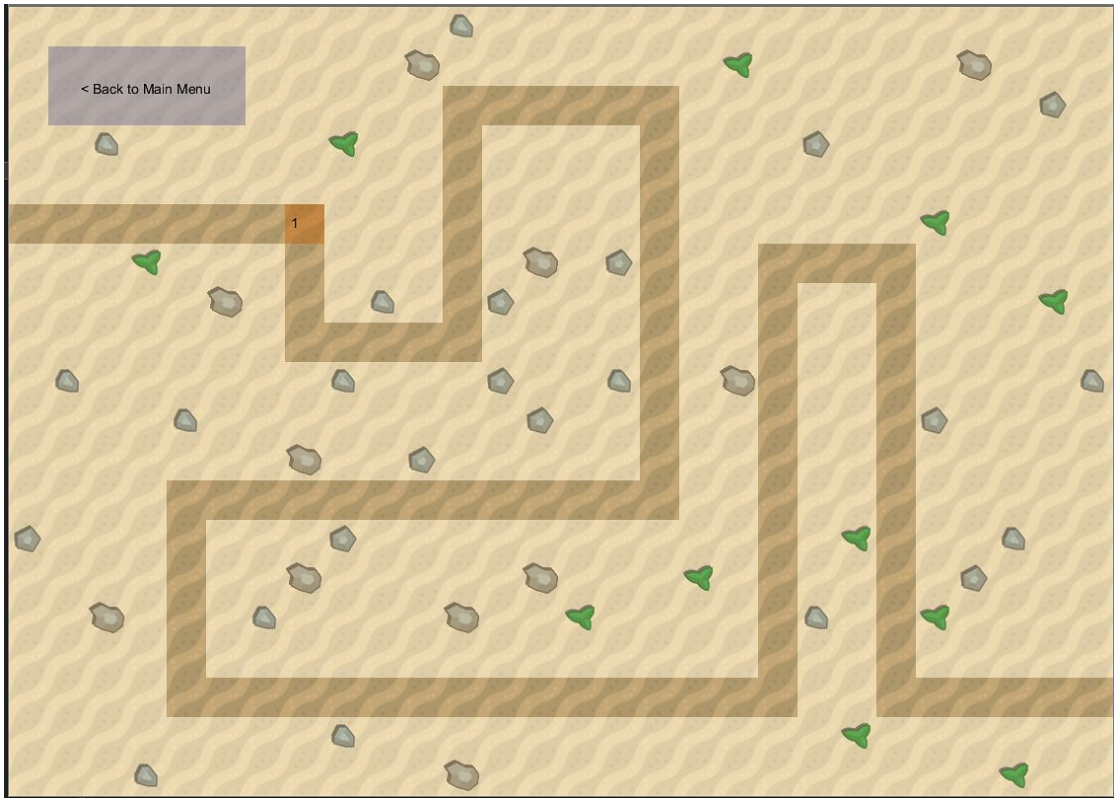
Help

-> Kertoo miten aloitat pelin, sekä muuta

Credits

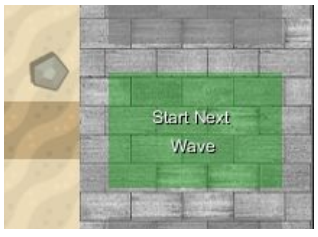
-> Kertoo tekemiseen vaadituista asioista, käytetyistä kirjastoista jne.

Jos peli halutaan aloittaa, mennään Progression Mappiin.



Tässä tallennuksessa ensimmäistäkään leveliä ei ole päästy läpi, koska vain Level 1 on lukitsematta. Takaisin aiempaan näkymään voisi palata painamalla ylhäältä vasemmasta kulmasta Back to Main Menu, mutta jos pelin haluaa aloittaa, pitää painaa tuosta numerosta. Tämä aloittaa pelin heti. Mitään hätää ei kuitenkaan ole, koska viholliset aloittavat vasta kun painat 'Start Next Wave'-nappulaa.

Ennen tätä pelaaja voi asettaa puolustuksensa painamalla tyhjästä tornista, ja valitsemalla jotain mihin on varaa.
Tämän näkee ostettavan asian taustan väristä, sekä oikealla sivussa sijaitsevasta tieto ruudusta, jossa kerrotaan asioista kun hiiri on niiden päällä. Omat statsit: rahat ja elämät ovat myös oikeassa laidassa, ylhäällä.



Torneja voi myös rakentaa painamalla tyhjiin paikkoihin, mutta tornit eivät itsessään tee mitään hyödyllistä pelin kannalta, siihen tarvitaan puolustuksia. Kun ‘Start Next Wave’ on painettu, voi pelaaja jatkaa asioiden ostelua normaalisti, mutta viholliset alkavat tulemaan kentän vasemmasta laidasta. Jos ei jaksa odottaa että viholliset laahustavat puolustuksesi luo, voit nopeuttaa pelin nelinkertaiseksi painamalla ‘>’-nappia, tai vaihtoehtoisesti painamalla Enteriä. Saman uudelleen painaminen myös palauttaa normaalin nopeuden. Voit myös keskeyttää pelin painamalla ‘Menu’-nappia, tai vaihtoehtoisesti Tabulaattoria. Tällöin avaan myös menun ja voit esim. Aloittaa pelin alusta tai palata aiempaan menuun.

Kun olet voittanut levelin, palaat takaisin Progressio Mappiin, ja pisteesi nyt näkyvät kun laitat hiiren ‘1’-kohdan päälle. Myös uusi levelin on nyt aukaistu, ja voit pelata sitä. Jos nyt suljet ohjelman ja käynnistät sen uudelleen, huomaat, että pelituloksesi on yhä tallessa! Voit myös aloittaa uuden pelin menemällä alku menuusta ‘Load Save’-osioon, ja valitsemalla ‘Save 2’-kohdan. Nyt Progression Menu on taas kuten aluksi. Tallennus jonka peli ottaa aina automaattisesti on siis Save 1.

Ohjelman Rakenne

Ohjelman lähdekoodi koostuu viidestä packagesta:
files, general, map, objects, user_interface

Files packagessa on tiedostot Level.scala ja Progress.scala.
Nämä luokat ovat tiedostojen lukua varten, ja kuvaavat tiedostoihin liittyvää tietoa ohjelman ymmärrettävässä muodossa. Level-oliolla on tieto tietyn levelin sisällöstä, eli sillä on tietyt Areena, Player ja Vihollis-aallot (Waves).. Level olio jäsentää parametriksi saaneensa tiedoston heti kun se luodaan, luoden Arenan, Playerin, sekä vihollis-Wavet.
Progress-oliolla on tieto tallennuksesta, eli siitä montako leveliä on päästy läpi, ja mitkä läpäistyjen levelien parhaat pisteet ovat. Progress-olio pystyy lataamaan tallenteen tiedostosta, sekä tallentamaan tilansa tiedostoon, mitkä ovat sen tärkeimmät tehtävät, sekä lisäksi kertoa nykyisen progression tilanteesta.

General packagessa on tiedostot Helper.scala ja Sounds.scala.
Nämä luokat ovat, kuten paketin nimi kertoo, yleisesti hyödyllisiä, eli niitä käytetään hajanaisesti eri osissa ohjelmaa. Helper luokan tarkoitus on antaa muille luokille

yleishyödyllisiä vakioita, jotta nämä vakiot voivat olla yhdessä paikassa. Muut luokat siis laajentavat Helperiä, saaden vakiot suoraan käyttöön. Tärkeimpiä vakioita ovat esimerkiksi sqSize (yhden Squaren koko) ja hiiren koordinaatit kertovat metodi. Sounds luokka laajentaa Äänikirjasto Minim luokkaa, ja sen kautta kaikki äänen toisto tapahtuu ohjelmassa. Sellaisen luotaessa kaikki äänitiedostot ladataan muistiin, ja niitä voidaan käyttää sen play-metodin kautta, joka kutsuu Minimin play sekä rewind metodeita. Sounds-luokka on erityisen kätevä, koska riittää että yksi sellainen luodaan Game:n setup-metodissa, että kaikki äänet saadaan ladatuksi.

Map packagen tiedostot ovat Arena.scala, Square.scala ja Tower.scala.
Nämä muodostavat kokonaisuutena Pelin pohjan, tai areenan ruudukon, joka koostuu Squareista, jotka voivat olla joko Tyhjä (Empty), Tie (Path), Obstacle (Este), tai Torni (Tower). Arena-olio on osa jotakin Leveliä, ja se käyttää Helperin antamia vakioita. Arena-oliolla on 2-ulotteinen Array sen sisältämistä Squareista, jotka se osaa piirtää peliin drawArena-metodilla. Tässä luokassa on myös muita tärkeitä metodeita, kuten:
Path-metodi, joka luo reitin vihollisille, setRow-metodi, jonka avulla level-tiedoston merkeistä saadaan makeSquare-metodin kanssa luotua Areenan sisältö, resetSquares-metodi, joka resetTowers-metodin kanssa palauttaa Areenan levelin alkutilaan jos leveli halutaan pelata useaan kertaan.
Square on Trait josta periytyy neljänlaisia Squareita, kuten edellä on mainittu.
Niillä on kaikilla indeksi, joka kertoo mikä kuvan niille piirretään, sekä basic-boolean arvo, joka kertoo onko squaren tyyppi empty tai path. Nämä luokat ovat paririvisiä, paitsi Tower. Tower-oliolla voi olla Defence, jolla voidaan tehdä asioita doStuff-metodin avulla, joka kutsuu sisältämänsä Defencen doStuff-metodia, jos Defence on olemassa.

Objects packagen tiedostot ovat Defence.scala, Mob.scala, Player.scala, Projectile.scala ja Wave.scala. Nämä luokat muodostavat pelin muun sisällön, jos Arena ajatellaan pohjaksi. Defence-luokka on abstrakti, jotta erilaisilla defenceillä voi olla erillaisia spesialiteetteja. Defence kuuluu tiettyyn torniin, kuten mainittu edellä, ja sillä on tietyt statsit etäisyydelle, damagelle ja nopeudelle. Myös se hyödyntää helperin vakioita. Defencellä on aina tietty kohde Mob, sekä jokin sprite ja ääni. Defence osaa piirtää itsensä, sekä ampua kohde Mobia. Aliluokkia Defencellä on BasicDefence, GunDefence, IceDefence ja FireDefence (joka ei ole käytössä). BasicDefencellä ei ole speciality-metodia (tai se on tyhjä), mutta sen

doStuff-metodi overridataan yliluokasta, koska defenssi halutaan piirtää eri tavalla. GunDefence on BasicDefencen aliluokka, ja muuten sama paitsi sillä on eri kuva ja ääni. IceDefencellä on spesialiteetti, jonka avulla se hidastaa sen etäisyydellä olevia vihollisia. Mob-oliolla on myös tietyt statsit (health, nopeus, koko, raha-arvo), sijainti, suunta, sekä oma HealthBar-olio. HealthBar- luokalla ei ole muita tietoja kuin mihin Mobiin se kuuluu, ja kuinka suuri määrä healthia siinä on. Se osaa piirtää itsensä ja vähentää health-arvoaan, sekä tietää onko sen mob kuollut. Mobin tärkeimmät metodit ovat: doStuff, jolla se piirtää itsensä, act-metodi, jonka avulla Mob liikkuu oikeaan suuntaan, move-metodi jonka avulla Mob liikkuu.
Player-oliolla on tiedot omasta hp:staan ja rahoistaan. Siltä voidaan kysyä onko varaa tiettyyn maksuun, ja voidaan kuluttaa rahaa. Luokka on varsin lyhyt ja yksinkertainen. Jokaisella Levelillä on oma Player, koska tietojen on kuitenkin resetoiduttava jokaisessa levelissä. Projectile-olioita luodaan aina kun Defence ampuu uuden shoot-metodillaan. Projetilella on 'ikä', jonka avulla määritellään milloin uusi projectile ammutaan Defencestä. Uusien Projectiilien luominen joka kerta kun ammutaan toimii parhaiten, koska tällöin Projektiilillä voi olla vain yksi selkäy kohde, jolloin ei käy tilannetta jossa sama luoti yrittäisi ensin osua yhteen viholliseen, mutta sitten muuttaisi suuntaa toiseen kun kohde vaihtuu. Projektiilin kiintoisin metodi on sen direction-metodi, joka laskee yhdenpituisen 2-ulotteisen vektorin kohdetta päin, jonka avulla projektiili osaa liikkua oikeeseen suuntaan. Wave-olio kuuluu tiettyyn Leveliin, ja se käyttää helperin vakioita (kuten myös aiemmat paitsi projectile). Wave-olio tärkein sisältö on sen Array Mobeja, joka luodaan Waven parametrien mukaan reset-metodissa, jota kutsutaan konstruktorissa. Wave osaa piirtää Mobit, ja saa myös kävelyanimaation toimimaan niiden kulkeman matkan avulla.

User_Interface packagen tiedostot ovat Game.scala, GameMenu.scala, InfoScreen.scala, IntroMenu.scala, Store.scala ja StoreMenu.scala. Nämä luokat muodostava pelin käyttöliittymän. Game-luokka on pelin pääluokka, jonka kautta Game-olion avulla peli ajetaan. Game laajentaa PApplettia, joka on Processingin applikaatioluokka. Tämän luokan pakolliset päämetodit ovat setup(), settings() ja draw(). Setup-metodissa peliin ladataan kaikki tarvittavat kuvat ja äänitiedostot levyltä. Settings-metodissa määritellään ikkunan koko, ja draw-metodi piirtää ikkunaan asioita halutuin väliajoin. Game-oliolla on tieto kaikista peliin tarvittavista osista: Leveleistä, Progressiosta, Menusta, Soundsisita.

GameMenu-luokka kuvaa pelitilanteen menua ja käyttöliittymää. Se koostuu muutamasta nappulasta, InfoScreenistä ja storeMenusta. InfoScreenin metodeja kutsutaan vain GameMenu:n kautta. Pääasiassa kaikki mitä GameMenu tekee materialisoituu sen doStuff- ja clickingStuff-metodeissa. InfoScreen on GameMenun osa, johon piirtyy tekstiä asioista joita pelissä katsotaan. InfoScreenillä on vain työkalut joiden avulla asioita voidaan kirjoittaa kätevästi siihen, mutta sisältö tulee Store-, StoreMenu- ja Defence-luokista.
IntroMenu-luokka kuvaa pelin aloitusmenuta. Sen apuna on Button-luokka, jonka avulla on helppo luoda samannäköisiä nappuloita. IntroMenulla on Progress, jonka kautta se tietää mitä piirtää progression mappiinsa. IntroMenulla on 5 mahdollista eri 'statea', kun se on aktivoitu: Start, Progress Map, Saves, Help ja Credits. Riippuen tästä statesta se piirtää eri asiat draw-metodillaan. Store-luokassa tapahtuvat kaikki pelin ostokset, ja uusien asioiden asettamiset. Siellä on myös määritelty defenssien alkustatsit, upgradet sekä asioiden hinnat.
StoreMenu luokka Kuvaa menua mikä ilmestyy kun pelaaja painaa tornin kohdalta. StoreMenu osaa piirtää itsensä ja tietää mitä valintoja siinä milläkin hetkellä on.

Algoritmit

Ohjelman tärkeimmät/mielenkiintoisimmat algoritmit lienevät:

- Projektiilin suunnan määritys
- Mobin reitin luominen suuntajonona
- Scoren laskeminen

Projektiilin suunnan määritys

Projectile-luokan metodin dir tulisi palautta kaksiulotteinen vektori suunnasta, johon projektiilin on tarkoitus mennä, jonka pituus on 1. Pituuden konsistenssi on tärkeä, jotta vauhti pysyy samana. Suunta on Defenssin ja kohde-Mobin välinen, Defenssistä Mobiin.
Yksi mahdollinen tapa olisi laskea ensin arctanilla suunnan kulma x-akselista, josta kosini ja sini arvot x ja y komponenteiksi, koska ne muodostaisivat automaattisesti yksikköympyrän kautta yhdenpituisen vektorin. Tyyliin: (cos(arctan(y/x)) , sin(arctan(y/x))).
Tämä on kuitenkin varsin paljon laskentatehoa vaativa operaatio, koska lasketaan arctangenttia ja siniä ja kosinia. Laskutavan tulisi olla kevyt, koska tätä saatetaan joutua laskemaan useita kertoja ruudunpäivityksessä, jopa satoja tai tuhansia kertoja sekunnissa jos defenssejä on enemmän.

Toinen, tehokkaampi tapa on käyttää hyväksi pythagoraan lausetta ja saman muodon skaalautuvuutta. Jos x-etäisyys jaetaan pythagoraan lauseen avulla lasketulla hypotenuusalla, saadaan vektorin x-komponentti, ja kun y-etäisyys jaetaan samalla, saadaan vektorin y-komponentti. Tämä tapa ratkaista sama ongelma vaatii vain yhden hypotenuusan laskemisen, ja kaksi jakolaskua.

Mobin reitin luominen suuntajonona

Koska mobin reitti tarvitsee luoda vain kerran (ts. Se ei muutu), olisi kätevintä jos Mobille luotaisi valmiiksi levelin luonnin yhteydessä 'reitti' jota pitkin se pysytyy kulkemaan. Ensimmäinen tapa, jolla tein Mobien reitin etsinnän oli jokaisella mobilla erikseen, missä se määrittää oman suuntansa edessä olevan Squaren perusteella, ja tarkistaa sitä tietyin väliajoin. Kuitenkin ongelmaksi kehkeytyi, että jos Mobin nopeus oli liian suuri, se ei onnistunut tarkistamaan edessä olevaa Squarea oikeaan aikaan, ja eksyi reitiltä. Lisäksi tuntui tyhmältä joutua tutkimaan reittiä 'livenä', vaikka reitti ei muutu. Vaihtoehto oli luoda Arena-luokkaan metodi path, joka palauttaisi kokoelman vektorisuuntia, jotka toimisivat ns. 'Käskyinä' Mobeille. Tämä kokoelma pitäisi luoda vain kerran leveliä kohden, ja Mobit voivat ottaa kokoelman indeksin niiden kulkeman matkan perusteella. Algoritmi käy reitin läpi while-loopissa, kunnes sijainti on kentän oikea laita, johon reitti aina päättyy. Jokaisen Squaren kohdalle tutkitaan edessä olevaa kohtaa, ja jos se on Path:ia, niin siihen jatketaan, muulloin oikealle, ja jos sinne ei pääse, sitten vasemmalle. Tätä jatketaan ja tallennetaan jokainen askel bufferiin. Askeleiden määrästä myös pidetään kirjaa, koska on mahdollista luoda kenttä, jossa jäädään tutkimaan luuppia, jolloin metodin toiminta keskeytyy 2000 askeleen jälkeen.

Scoren Laskeminen

Tämä ei sinäänsä ole pelin kannalta kovin keskeinen, mutta sen voisi tehdä hyvin monella tavalla. Pelaajan läpäistessä levelin, score-metodia kutsutaan, ja se laskee maksimimatkan erotuksen jokaisen levelissä olleen Waven jokaisen Mobin kulkemasta matkasta yhteen, joka jaetaan "parhaasta mahdollisesta matkasta, jonka viholliset olivat menemättä", ja tämä myös suhteutetaan pelaajan lopputilanteen health pointsiin, ja lopulta kerrotaan tuhannella, ja ollaan tilanteessa, jossa luku 0 ja 1000 välillä kertoo, kuinka tehokkaasti leveli päästiin läpi.

Tietenkin, on täysin mahdotonta saada täysiä pisteitä tällä tavoin, mutta luku todella kertoo kuinka hyvin peli meni.

Tietorakenteet

Koska kyseessä on peli, on huomattavasti kätevämpää ja selkeämpää (ja varmaan tehokkaampaa) käyttää paljon muuttuvaisia tietorakkenteita. Esimerkiksi vihollisten tapauksessa, jos koitettaisiin olla käyttämättä muuttujia, ja mennä täysin funktionaalisesti, joutuisi jokaisen Mob-olion jatkuvasti luomaan uudelleen, mikä johtaa vaikeuksiin.
Pyrin kuitenkin käyttämään mahdollisimman paljon muuttumattomia rakenteita, silloin kun muuttuminen ei ollut välttämättömyys.

Var-muuttujien tilalla oli monessa tapauksessa kätevää käyttää lazy val-tyyppisiä arvoja, koska monessa osaa kun pelissä ladataan asioita, on tärkeää, että asioita luodaan vasta kun niiden tarvitsemat edellytykset on luotu. Lazy valit ovat juuri tähän käyttötarkoitukseen käteviä, ja esierkiksi yhden pitkän debuggauksen päätteksi riitti, että yksi arenan konstruktorissa ollut val muutettiin lazy valiksi, jotta se ei evaluiodu heti Arenaa luodessa. Myös tilanteissa, missä ei ole varmaa tarvitaanko tiettyä tietoa, tehostavat lazy valit ohjelmaa, koska ne evaluoidaan vaan jos niitä tarvitaan. En tiedä onko lazy valien käyttö ns. Hyvää ohjelmointityyliä, mutta ainakin tuntuvat kovin käteviltä.
Var-muuttujiakin tulin käyttäneeksi kun sitä vaadittiin, varsinkin liikkuvien asioiden paikan tiedon tallentamisessa. Jouduin myös käyttämään vareja tallentamaan joitain pelin kannalta keskeisiä suuria kokonaisuuksia siksi, koska Processing-kirjasto vaatii sitä. Oletan että tämä johtuu siitä, että java-ohjelmointityyliin tämä on tavallisempaa, jonka kanssa Processingin on tarkoitettu toimivan. Esimerkiksi pelin ääniä hoitelevaan Sounds-olioon viittaava muuttuja on pystyttävä lataamaan vasta Processingin vaatimassa setup()-metodissa, jotta äänitiedostot tulevat ladattua. Metodin sisälle ei kuitenkaan ole mahdollista luoda arvoja, jotka olisivat käytössä sen ulkopuolella, joten ainut ratkaisu oli luoda var-muuttuja joka on aluksi null, ja setupissa saa oikean arvonsa kun oliot luodaan. Myös kuvatiedostot oli tästä syystä kätevintä ladata Arrayhin jotka ovat aluksi tyhjiä. Olen myös pyrkinyt pitämään mahdollisimmat monet var-muuttujat yksityisinä kun mahdollista, ja joissain luokissa niihin pääsee käsiksi vain tiettyjen metodien kautta.

Arenan Squaret ovat tallessa Array[Array[Square]]-tyyppisesti, koska areenan koko ei muutu, mutta sen yksittäiset ruudut voivat muuttua, koska sinne voidaan ostaa torneja. Mobit ovat samaan tapaan Wavessa Arrayssa, jotta ne voidaan resetoida kun leveliä halutaan pelata uudestaan. Levelissä taas Wavet luodaan Bufferiin, koska ei ole alkutietoa siitä, montako Wavea luodaan.
IntroMenu-luokassa keksin kätevän tavan kuvata menun tiloja yksinkertaisesti numeroilla. Menun pää drawStuff-metodissa matchataan sen currentState näihin numeroihin, ja piirretään näin oikea osa.

Tiedostot

Ohjelma käsittelee kahdenlaisia tiedostoja: Leveli-tiedostoja ja Progressio-tiedostoja. Leveli-tiedosto kuvaa yhtä pelin leveliä, eli alkurahatilannetta, sen areenan sisältöä ja Mob aaltojen sisältöä. Progressio-tiedosto kuvaa pelin edistys-tallennusta tietona siitä, kuinka monta leveliä on päästy läpi, sekä mitkä näiden levelien highscoret ovat. Levelitiedostoja ohjelma osaa ainoastaan lukea, mutta Progressiotiedostoja ohjelma myös kirjoittaa, koska sen on tarkoitus tallentaa tietoa. Ohjelma osaa käsitellä myös virhe syötteitä heittämällä itse määriteltyjä poikkeuksia, jotka kertovat virheestä. Molemmat tiedostotyypit luetaan tekstitiedostoina, ja tietoa tarkemmasta formaatista löytyy:
Levelitiedostot:
Desert Tower Defence/lvls/readme.txt
Progress-tiedostot:
Desert Tower Defence/saves/readme.txt

Testaus

Suurin osa ohjelman testauksesta tapahtui ohjelman toimintaa katsomalla, tai asioiden kokeilusta pelissä. Katsoen testaussuunnitelmaa, nykyinen ohjelma läpäisee kaikki nuo kriteerit. Tiettyjen asioiden testauksessa ei riittänyt katsoa mitä ohjelma piirsi käyttöliittymään, vaan piti tutkia ohjelman metodien palauttamia arvoja printtaamalla tuloksia konsoliin. Yksi hankalimmista ongelmista oli paikantaa ongelma Arenan piirtämisen kanssa, jossa x- ja y-akselit olivat jostain syystä väärin päin. Tätä ongelmaa ratkaistessa tein Arenalla toString-toteutuksen, joka palauttaa squaret string-muodossa, samaan tapaan kuin ne on esitetty Level-tiedostoissa. Myös joitain ongelmia Levelin luonnin järjestykseen

liittyen oli haastava paikantaa, jossa perus debug-työkalujen käyttö auttoi. Unit-testien luontia en nähnyt tämä kaltaiselle ohjelmalle sopivaksi, koska suurin osa asioista on niin helposti nähtävissä ohjelman pyöriessä.

Ohjelman tunnetut puutteet ja viat

Ohjelmassa on neljä tunnettua bugia:

1. Jos Basic- tai GunDefenssin Rangea suurentaa liikaa (about 320:n), projektiilit vain katoavat, eivätkä osu mihinkään alueen reunoilla
2. Pelin nopeutus joskus toimii pelaajaa vastaan, koska projektiilit eivät tahdo osua yhtä hyvin kuin normaalilla nopeudella, varsinkin ongelma kun defenssin nopeutta on kasvatettu enemmän
3. Mobien kävelyanimaatio jäätyy niiden viimeisellä squarella pathissa
4. Joskus pelin nopeutus saa Mobit siirtymään tieltä sivuun hiukan, varsinkin nopeilla mobeilla

Miten ehkä korjaisin:

1. Ongelma johtuu projektiilin ‘age’-arvosta, joka saa projektiilin ‘kuolemaan’ tietyn ajan jälkeen. Tämän arvo tulisi jotenkin siis suhteuttaa rangeen, jolloin nopeus kylläkin saattaisi hidastua samalla, mutta olisi muuten hyvä ratkaisu.
2. Tämän korjaaminen saattaisi vaatia projektiilin viholliseen osumisen uudelleen ajattelemista. Jos projektiilin ei tarvitsisi edes oikeasti osua viholliseen, vaan riittää että se on liikkunut oikean matkan, että tekee damagea, ne osuisivat aina eikä tämä ole ongelma enää.
3. Johtuu siitä, että animaation freimi otetaan mobin kulkemasta matkasta, jota ei enää päivitetä viimeisessä squaressa. Ratkaisu voisi olla jatkaa matka-muuttujan kasvattamista, mutta jotenkin olla huomioimatta sitä path-suuntakomentojen seurauksessa.
4. Tähän en keksi heti ainakaan mitään ratkaisua. Ongelma on, että mobit tietävät kääntyä vasta kun ovat liikkuneet squaren keskikohdan ohi. Mahdollisuus olisi esim. Että mobin hidastavat ennen kääntymistään, mutta jo tämä vaatisi aika paljon muutoksia.
5.

Muita puutteita:
Ei sinänsä mitään kovin vakavaa, peli on aika hyvä kokonaisuus. Olisi tietenkin kiva jos olisi vähän enemmän erilaisia defenssejä, mutta niitä olisi helppo luoda lisää. Myös levelit eivät tietty viellä ole valmiita kaikki, mutta koitan saada demoa varten.

3 Parasta ja 3 Heikointa kohtaa

3 Parasta:

- IntroMenu-luokka ja Button-luokka, ne muodostavat hyvin selkeän ja kätevän kokonaisuuden ja lisäykset ovat helppoja
- Projectile-luokka on lyhyesti toteutettu, ja toimii tavallaan funkionaalisesti, koska niitä luodaan uusia aina kun ammutaan
- Level- ja Progress-luokat osaavat kertoa, jos tiedostoissa on tiettyjä virheitä

3 Heikointa:

- GameMenu-luokassa on paljon toistuvuutta, ja clickingStuff-metodin olisi voinut toteuttaa selkeämmin -> Olisi voinut korjata tekemällä tälle oman Button luokan tai käyttämällä IntroMenun Button-luokkaa, ja siirtämällä clickingStuffin asiat erillisiin osakokonaisuuksiinsa, joita kutsutaan tuosta metodista
- StoreMenu-luokka ei tue sen laajentamista kovin helposti, ja eri tilanteen on korjattu aika purkkamaisesti -> StoreMenu tulisi uudelleenmiettiä, ja sen koon tulisi olla määretitetty suoraan sen sisällöstä mielummin kuin erikseen
- Arenan squaresTransposed-hässäkkää ei pitäisi olla olemassa -> Arenaa luodessa se pitäisi luoda kerralla oikein, eli ottaa huomioon kaikkialla, että arenan sisällön ensimmäinen parametri on y-koordinaatti
- Myös muita: esim. Defence-luokan shoot-metodi on aikalailla purkka jota en edes ymmärrä, mutta se toimii, pitäsi selkeyttää

Poikkeamat Suunitelmasta

Kuten jo aiemmin mainittu, StoreMenun idea poikkeaa suunnitelmasta, mutta muuten lopputulos seuraa suunnitelma suurin piirtein. Ajankäytön olin arvoinut paljon alakanttiin, kuten vähän oletinkin, veikkaan että aikaa meni vähintään 3-kertaa niin paljon kuin olin arvioinut. Toteutusjärjestys oli itseasiassa suurin piirtein sama kuin karkeassa arviossa, mutta lisäsin tekstuureja ja ääniä myös projektin loppupuolella, ja varsinkin vihollisten ja puolustusten mallintaminen ja oikean toimivuuden löytäminen kesti huomattavasti kauemmin kuin odotettu arvio. Myös valikkoihin ja Storen toimintaan kului paljon enemmän aikaa.

Toteutunut Työjärjesty ja Aikataulu

No tuossa ylempänä jo mainitsin. Tarkat tiedot löytyvät gitlabista. Arenan ja Levelitiedoston mallinnus olivat ensimmäisenä, jonka jälkeen tornien asetus. (around 4.3) Tuon jälkeen aloin mallintaa vihollisia ja niiden liikettä, jonka jälkeen defenssien perustoimintoja. (around 9.3) Tuon jälkeen projektiilit, mobit aaltoina, äänet. (around 10.4) Tämä jälkeen Store Asiat ja StoreMenu, pelin nopeutus ja InfoScreen. (around 18.4) Lopuksi Progressio filet, IntroMenu ja tallennuksen tekeminen. (around 24.4) Suunnitelmasta poikkesin vasta 14.4, kun lisäsin StoreMenun.

Arvio Lopputuotoksesta

Olen tyytyväinen ohjelman kanssa, se on mielestäni selkeä kokonaisuus, ja ihan uskottavan näköinen esim. Nettipeliksi. Tietenkin vaihtoehtoja pelissä voisi olla enemmän, mutta oleellisia puutteita en näe siitä mitä tällaiselta peliltä odottaisi. Ohjelmaa voisi parannella sen laajetevuudessa, erityisesti StoreMenu-luokka on kriittinen jos peliin haluaa lisää toiminnalisuutta. Jotkut osat ohjelmasta taas soveltuvat erittäin joustavasti laajennuksiin: Defencejä voidaan määrittää uusia helposti, ja Mobeja myös. Levelien teko on lastenleikkiä. Myös uuden tyyppisiä asioita, kuten tielle tulevia Trappeja ei olisi vaikea mallintaa nykyisen ohjelman pohjalta. Myös IntroMenu on tehty hyvin joustavaksi lisäyksiin, jos se niitä tarvitsee.

Viitteet

Scala API: **https://www.scala-lang.org/api/current/index.html**

Processing reference: **https://processing.org/reference/**

Processing tutorials: **https://www.youtube.com/thecodingtrain**

Tower Defence Assets: **http://kenney.nl/assets/tower-defense-top-down**

Minim reference: **http://code.compartmental.net/tools/minim/**

Lisää löytää myös pelin Credits-sivulta.