Tower Defense

Péter Siket-Szász

589880

SCI, Tietotekniinkka

1. Vuosikurssi

26.4.2017

Yleiskuvaus

Projekti on klassinen tornipuolustuspeli (tower defence), jossa joukko erilaisia vihollisia pyrkii pääsemään maaliin niille määrättyä reittiä pitkin. Reitin varrelle rakennetaan torneja, jotka ampuvat vihollisia tai tuottavat pelaajalle rahaa ja näin pyrkivät estämään niiden pääsyn maaliin. Projektin grafiikat ovat palloja ja neliöitä (toteutettu scalan (ja javan) swing kirjastolla). Projekti toteuttaa kaikki vaikean tason perusvaatimukset ja on asetustiedostojen kautta konfiguroitavissa (lisävaatimus), joten omasta mielestäni projektin vaativuustaso on vaikea.

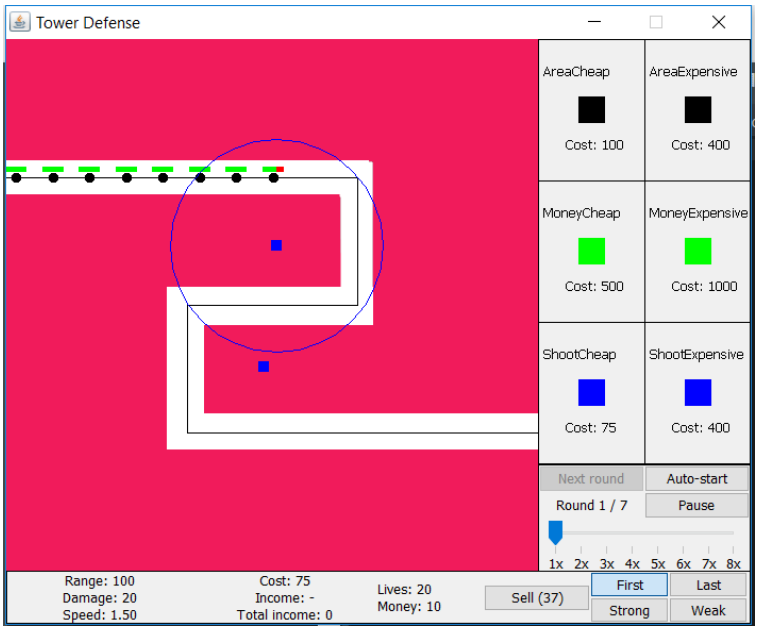
Käyttöhje

Ohjelma käynnistetään ajamalla tiedosto MainWindow.scala. Ohjelman käynnistyessä pelaajalle tulee näkyviin ruutu, jossa on Play-nappi. Play-nappia painettuaan pelaajan täytyy valita taso, jolla peliä halutaan pelata.

Tasot, tornit, kierrokset ja viholliset määritetään tiedostoihin niille varattuihin kansioihin projektin hakemistossa. Projektin mukana on muutamia esimerkkitiedostoja, joita voi hyvin käyttää pelaamiseen, mutta myös omia tiedostoja on helppoa ja suositeltavaa määritellä (ohjeet projektissa, tiedostossa 'Instructions.txt').

Valittuaan tason, pelaajalle ilmestyy pelinäkymä. Pelinäkymässä oikealla on erilaisia torneja, joita pelaaja voi ostaa klikkaamalla ja viemällä ne pelikentälle. Torneja ei voi asetella päällekäin, eikä niitä voi laittaa liian lähelle vihollisten rataa. Jos pelaaja on laittamassa tornia laittomalle alueelle, torni muuttuu punaiseksi, tarkoittaen, että sitä ei voi laittaa haluttuun kohtaan. Sama käy, jos pelaajan rahat eivät riitä torniin (Mustat tornit ampuvat tietylle alueelle ympärillään, siniset tornit ampuvat yhtä kohdetta ja vihreät tornit antavat rahaa kierrosten lopussa.) Pelinäkymässä on erilaisia nappeja, joilla voi aloittaa ja pysäyttää pelin, myydä tornin, aloittaa kierros tai asettaa kierrokset alkamaan itsestään sekä valita minkä vihollisen tietyt tornit ottavat kohteekseen. Pelin nopeutta voi säätää liukusäätimellä. Pelinäkymän alaosassa on pieni informaatiopaketti, joka kertoo pelaajan elämät ja rahat sekä erilaisia tietoja valitusta tornista.

Pelin alettua vihollisia ilmestyy radalle, jotka liikkuvat kohti maalia. Jos vihollinen pääsee maaliin pelaaja menettää elämiä riippuen siitä, kuinka paljon kyseisellä vihollisella oli elämää jäljellä. Peli loppuu voittoon, jos pelaaja läpäisee kaikki tietyn tason kierrokset, tai häviöön jos pelaaja menettää kaikki elämänsä. Pelin loputtua pelaaja voi aloittaa uuden pelin kuten aiemminkin.



Pelinäkymä

Ohjelman rakenne

Ohjelma koostuu kolmesta pakkauksesta:

(O) tarkoittaa, että kyseessä on objekti

(T) tarkoittaa, että kyseesstä on piirreluokka

1. gameLogic:

   Game(O), Course, Round, Vector, Coord

   Sisältää pelin logiikan toteuttamiseen tarvittavat luokat.

2. gameUnits:

   Tower(T), DamageTower, AreaTower, ShootingTower, MoneyTower, TowerCreator(O), TargetType(T + case classit: First, Last, Weak, Strong)

   Enemy(T), Pathfinder, EnemyCreator(O)

   Player, RoundCreator(O), CourseCreator(O)

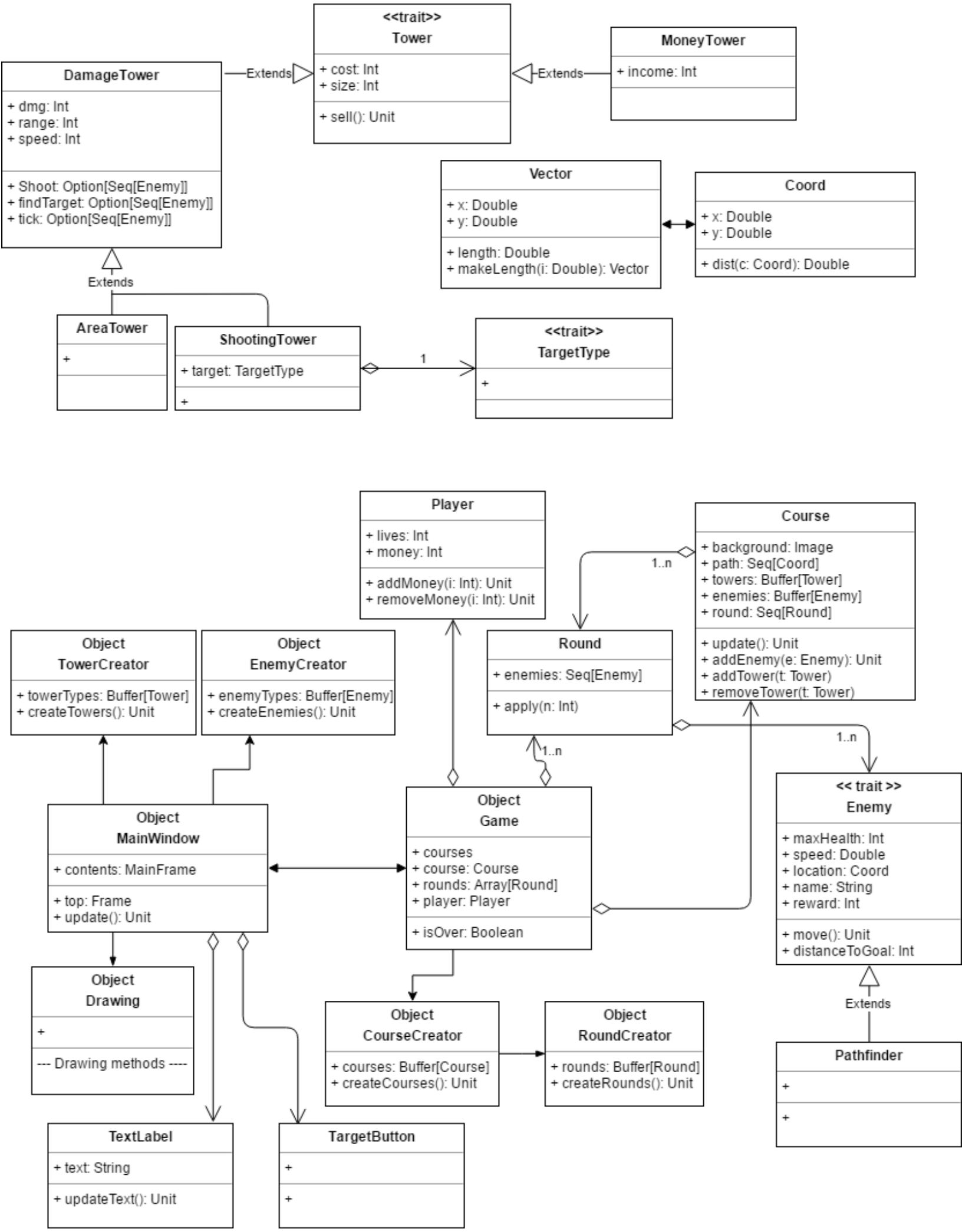
   Sisältää pelin yksiköt, joita esiintyy pelin aikana.

3. gui

   MainWindow(O), Drawing(O), TargetButton, TextLabel

   Sisältää kaiken, mikä liittyy käyttöliittymän piirtämiseen

UML – kaavio seuraavalla sivulla ja luokkien selitykset sen jälkeen


DamageTower
+ dmg: Int
+ range: Int
+ speed: Int
+ Shoot: Option[Seq[Enemy]]
+ findTarget: Option[Seq[Enemy]]
+ tick: Option[Seq[Enemy]]
Extends
<<trait>>
Tower
+ cost: Int
+ size: Int
+ sell(): Unit
Extends
MoneyTower
+ income: Int
Vector
+ x: Double
+ y: Double
+ length: Double
+ makeLength(i: Double): Vector
Coord
+ x: Double
+ y: Double
+ dist(c: Coord): Double
Extends
AreaTower
+
ShootingTower
+ target: TargetType
+
1
<<trait>>
TargetType
+
Player
+ lives: Int
+ money: Int
+ addMoney(i: Int): Unit
+ removeMoney(i: Int): Unit
Course
+ background: Image
+ path: Seq[Coord]
+ towers: Buffer[Tower]
+ enemies: Buffer[Enemy]
+ round: Seq[Round]
+ update(): Unit
+ addEnemy(e: Enemy): Unit
+ addTower(t: Tower)
+ removeTower(t: Tower)
1..n
Object
TowerCreator
+ towerTypes: Buffer[Tower]
+ createTowers(): Unit
Object
EnemyCreator
+ enemyTypes: Buffer[Enemy]
+ createEnemies(): Unit
Round
+ enemies: Seq[Enemy]
+ apply(n: Int)
1..n
1..n
Object
MainWindow
+ contents: MainFrame
+ top: Frame
+ update(): Unit
Object
Game
+ courses
+ course: Course
+ rounds: Array[Round]
+ player: Player
+ isOver: Boolean
<< trait >>
Enemy
+ maxHealth: Int
+ speed: Double
+ location: Coord
+ name: String
+ reward: Int
+ move(): Unit
+ distanceToGoal: Int
Extends
Pathfinder
+
+
Object
Drawing
+
--- Drawing methods ----
Object
CourseCreator
+ courses: Buffer[Course]
+ createCourses(): Unit
Object
RoundCreator
+ rounds: Buffer[Round]
+ createRounds(): Unit
TextLabel
+ text: String
+ updateText(): Unit
TargetButton
+
+

Ohjelman tärkeitä osakokonaisuuksia ovat: Tornit ja viholliset, käyttöliittymä, pelin alustaminen tiedostoista sekä itse logiikka

Torneja on pelissä kolmenlaisia, AreaTower vahingoittaa kaikkia vihollisia alueellaan, ShootingTower ampuu yhtä vihollista kerrallaan ja MoneyTower tuottaa pelaajalle rahaa kierrosten lopussa. AreaTower ja ShootingTower ovat DamageTowereita, niillä on metodi tick, jota kutsutaan jokaisella päivityksellä. Se kertoo ampuuko torni ja jos ampuu niin mitä. Torneilla on tiedossa, mitä vihollisia kentällä juuri tällä hetkellä on. Ne ampuvat vihollisia, jotka ovat tarpeeksi lähellä. ShootingTower lisäksi valitsee parhaimman vihollisen sen mukaan miten sitä on käsketty ampumaan.

Viholliset liikkuvat aina niille määritellyllä nopeudella kohti maalia tiettyä reittiä pitkin. Vihollisilla on metodi move(), joka siirtää niitä lähemmäs maalia käyttäen apuna Vector - luokkaa. Tätä metodia kutsutaan jokaisella päivityksellä.

Valmiiksi määriteltyä vihollisluokkaa Pathfinder käytetään radan pisteiden selvittämiseen. Pathfinder kulkee radan alusta loppuun ja palauttaa kaikki pisteet, joissa se kävi.

Vihollisilla ja torneilla on sijainti, jota kuvataan Coord – luokalla. Coord – luokka kuvaa koordinaatteja lukupareina. Yhdessä Vector – luokan kanssa niillä voi kuvata liikettä ja sijaintia hyvin. Vaikka kaaviossa ei olekaan viitenuolia näihin luokkiin muista luokista, niitä käytetään aina, kun halutaan viitata tai tallentaa jokin sijainti.

Coord- ja Vector-luokkien sijaan olisi voinut käyttää vain Vector-luokkaa mallintamaan sekä sijaintia, että vektoreita ajattelemalla, että koordinaatit ovat origosta lähteviä vektoreita. Toteutuksessa on kuitenkin käytetty myös Coord-luokkaa, sillä koordinaattien mallintaminen omana yksikkönään tuntui luonnollisemmalta.

Käyttöliittymä on toteutettu scalan swing kirjastolla, mutta joitakin asioita on lainattu myös javan vastaavasta kirjastosta. Kaikki mitä näytölle piirretään on MainWindow – olion sisällä. MainWindow – oliolla on update() – metodi, jonka avulla se piirtää näkymän, sitä kutsutaan jokaisella päivityksellä. MainWindow käyttää hyväkseen Drawing – oliossa määriteltyjä piirtometodeja esimerkiksi tornien ja vihollisten piirtämiseen. Myös omia luokkia TextLabel ja TargetButton käytetään apuna piirtämisessä. TextLabel – luokka vaatii toteutuksessaan updateText() – metodin, jolla se päivittää tekstin, jota se näyttää. TargetButtonia käytetään hyödyksi, kun halutaan muuttaa ShootingTowerin kohdetta. Käyttöliittymä kuuntelee pelaajan aiheuttamia tapahtumia ja reagoi niihin.

Pelikentälle piirretään ensin tason tausta, jonka jälkeen piirretään tornit ja viholliset. Sen jälkeen piirretään jokaiselle tornille sen mahdollisesti ampuma ammus, jotka Game-olio antaa. Myös pelialueen ulkopuolella olevaa tietoa päivitetään sen mukaan mitä pelissä tapahtuu.

Muita grafiikkakirjastoja, joita olisi voitu käyttää ovat esimerkiksi Scala Processing tai ScalaFX. Valinta oli lopulta kuitenkin swing, sillä se oli jo entuudestaan melko tuttu ja näin helpompi käyttää, eikä uuden kirjaston opetteluun kulunut valtava määrä aikaa. Swingillä sai hyvin toteutettua projektin, eikä mitään suuria ongelmia kirjaston valinnan takia ollut.

Pelin tornit, viholliset, kierrokset ja radat ladataan peliin tiedostoista, kun peli käynnistetään. Kaikilla tiedostotyypeillä on tietty formaatti, jota täytyy noudattaa (ohjeet projektissa tiedostossa ‘Instructions.txt’). Tiedostojen sisällön lukemiseen on käytetty javan File- ja scalan Source- luokkia.

Tiedostoista luetaan yksi rivi kerrallaan ja otetaan rivin arvo talteen muuttujaan, josta sitä myöhemmin hyödynnetään. Tällä tavalla luoduista luokista tallennetaan luomisolioon ilmentymät, josta niitä voidaan myöhemmin hakea. Jokaisella luokalla, joka on ladattu tiedostosta on metodi makeNew, joka palauttaa luokan uuden ilmentymän, ilman, että sille annetaan mitään parametreja. Tätä käytetään, jotta saadaan uusia ilmentymiä luokista ilman, että käytettäisiin niiden nimiä. Tiedostoja ladattaessa käytetään muihin tiedostoihin viittaamiseen niiden nimiä. Tiedoston nimi annetaan luokalle talteen.

Suurin osa pelin toiminnasta tapahtuu Course – luokassa. Course – luokka pitää tallessa pelissä olevat tornit, viholliset, niiden reitin sekä mitä kierroksia pelissä tulee. Vihollisten reitti on kokoelma pisteitä (Coord – luokan ilmentymiä), jotka kertovat missä vihollisen täytyy kääntyä. Eli aina, kun vihollinen törmää tällaiseen pisteeseen, sen kääntyy kohti seuraavaa pistettä, jossa sen täytyy kääntyä. Course – luokan update – metodi päivittää pelin tilannetta aina sitä kutsuttaessa. Sitä kutsutaan jokaisella päivityksellä.

Game – olio ohjaa peliä kokonaisuutena. Sillä on tallessa tämänhetkinen Course-luokan ilmentymä sekä Player- luokan ilmentymä. Game pitää myös tallessa kaikki eri kierrokset ja radat, jotka peliin on ladattu sen käynnistyessä. Game – oliossa kuunnellaan Timeria ja jokaisella Timerin tickillä – eli noin 60 kertaa sekunnissa – Game pyytää Coursea sekä käyttöliittymää päivittämään itsensä update – metodeillaan. Peliä voi nopeuttaa käyttöliittymässä olevalla liukusäätimellä, jolloin luodaan uusi Timer, joka lähettää tickejä eri nopeudella.

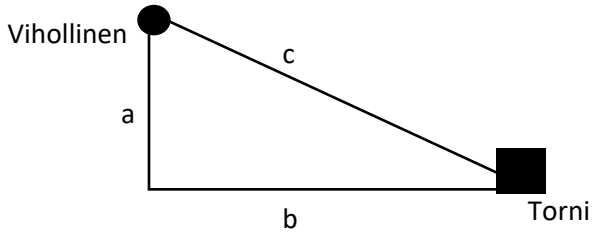
Lyhyesti sanottuna, pelin aikana Game – olio pyytää Coursea ja käyttöliittymää päivittämään itsensä. Käyttöliittymä piirtää Coursen uuden tilanteen näytölle ja reagoi kaikkeen, mitä käyttäjä on tehnyt.

Algoritmit

Ohjelmassa on muutamia algoritmeja, joista olennaisimmat ovat tornien kohteenvalinta, vihollisten reitinseuranta ja tarkistus, onko hiiri minkään tornin päällä.

Kohteenvalinta:

Torneilla on tiedossa kaikki kentällä olevat viholliset. Näistä tornit kuitenkin ampuvat vain niitä, jotka ovat tarpeeksi lähellä. Vihollisten etäisyys tarkistetaan käyttämällä hyödyksi Coord – luokan dist – metodia, joka kertoo kahden koordinaatin (pisteen) välisen etäisyyden käyttämällä hyödyksi Pythagoraan lausetta: $a^2 + b^2 = c^2$.



Jos tornin ampumissäde on pienempi kuin etäisyys c, torni ei tee viholliselle mitään. Jos vihollinen kuitenkin on ampumissäteen sisällä, torni tekee siihen vahinkoa. ShootingTowerit valitsevat kaikista tarpeeksi lähellä olevista vihollisista sen, joka parhaiten sopii niiden tähtäyslogiikkaan. Ne voivat ampua vihollisia, joilla on eniten elämää, vähiten elämää, jotka ovat lähimpänä maalia tai kauimpana maalista.

Vihollisen matka maaliin lasketaan laskemalla ensin sen matka nykyisestä sijainnista seuraavaan kääntöpisteeseen ja sen jälkeen jäljellä olevien kääntöpisteiden etäisyyksien summa. Etäisyyksiä lasketaan taas Coord – luokan avulla.

Paras kohde lopulta valitaan sen mukaan, millä vihollisista oli kriteeriin sopivin arvo.

Reitinseuranta:

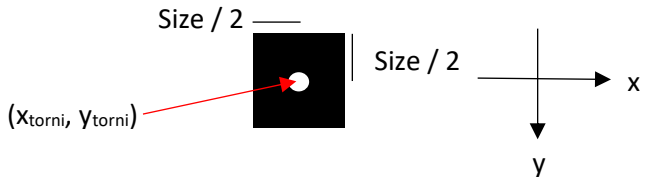
Kuten jo edellä mainittu, vihollisten reitti on kokoelma pisteitä, jotka kuvaavat reitin käännöspisteitä. Olkoot seuraava käännöspiste nimeltään kohde. Muodostetaan vektori nykyisestä sijainninsta kohteeseen. Jos vektorin pituus on pienempi kuin nopeus, jolla liikutaan, otetaan kohteeksi seuraava kohde ja muodostetaan uuteen kohteeseen vektori. Toistetaan tätä, kunnes saadaan kohde, joka on tarpeeksi kaukana tai ollaan maalissa. Muodostetaan nyt saadusta vektorista sen pituinen, millä nopeudella liikutaan. Vektorista saadaan tietyn pituinen muodostamalla siitä yksikkövektori, eli jakamalla jokainen komponentti vektorin pituudella. Yksikkövektorin voi sen jälkeen kertoa halutulla pituudella, jotta saadaan halutun pituinen vektori.

$$\hat{b} = \frac{\vec{b}}{|\vec{b}|} = \frac{(x,y,z)}{\sqrt{x^2+y^2+z^2}}$$

Kolmiulotteisen yksikkövektorin muodostaminen (ohjelmassa vain 2 ulottuvuutta)

Hiiren sijainnin tarkistus:

Aina hiirtä liikutettessa, tarkistetaan, onko se minkään olemassaolevan tornin päällä. Tämä tehdään vertaamalla hiiren sijaintia kentällä olevien tornien sijaintiin.



Hiiri on tornin päällä joss se täyttää seuraavat ehdot:

x_{hiiri} + size / 2 >= x_{torni}

x_{hiiri} - size / 2 <= x_{torni}

Vastaavat y-suunnassa

y_{hiiri} + size / 2 >= y_{torni}

y_{hiiri} - size / 2 <= y_{torni}

Tietorakenteet

Ohjelmassa käytetään useasti scalan valmiita kokoelmatyyppejä, enimmäkseen kuitenkin muuttuvatilaisia, sillä ohjelman aikana lähes kaikkiin kokoelmiin täytyy pystyä lisäämään ja poistamaan

tietoa. Käytetyin kokoelmatyyppi on scalan muuttuvatilainen Buffer. Ainoastaan joissain käyttöliittymän muuttumattomissa kokoelmissa sekä vihollisten reitin tallentamisessa on käytetty muita kokoelmatyyppejä.

Esimerkiksi Course - luokan tallessa olevat viholliset ja tornit on kätevä pitää Bufferissa, sillä niiden määrä muuttuu jatkuvasti. Muita muuttuvatilaisia kokoelmiakin scalassa toki on, mutta Buffer on tähän käyttötarkoitukseen riittävä. Muuttumattomien rakenteiden käyttö tähän tarkoitukseen olisi ollut erittäin epäkäytännöllistä ja vaivalloista.

Mitään omia tietorakenteita ohjelmaan ei ole luotu, sillä niille ei ollut tarvetta.

Tiedostot

Ohjelma käsittelee neljässä eri formaatissa olevia tekstitiedostoja; tornit, viholliset, kierrokset ja radat. Laaja kuvaus ohjelman käsittelemistä tiedostoista ja niiden formaateista on projektin mukana tiedostossa 'Instructions.txt'. Projektin mukana on muutamia esimerkkitiedostoja, joita voi käyttää pelin pelaamiseen. Torneja voi olla pelissä enintään 10 erilaista.

Testaus

Koodin testaus toteutettiin niin, että testausvaiheessa käskettiin ohjelmaa tulostamaan eri muuttujien arvoja konsoliin ja niitä verrattiin siihen, mikä oli odotettu arvo. Välillä tämä ei ollut yhtä helppoa kuin mitä aluksi oli ajatellut, mutta ongelma kyllä löytyi aina lopulta. Eclipsessä on toki debuggaustyökalu, mutta sen toimintaan en itse koskaan ole sen tarkemmin perehtynyt, enkä sitä tähänkään projektiin käyttänyt. Testauksen suunnittelu oli selkeää, sillä kaikki testauksen kannalta olennainen tieto saadaan tulostettua ja näin testaus on yksinkertaista. Ohjelma läpäisee suunnitelmassa esitetyt testit ja vaatimukset. Yksikkötestejä koodin toimivuudelle ei ole tehty. Luulen, että testien laatimiseen olisi mennyt enemmän aikaa, kuin koodin testaamiseen manuaalisesti meni.

Ohjelman tunnetut puutteet ja viat

Ohjelmassa tunnettu vika on, että torneja voi pelialueelle laittaa "kulmittain". Tavallisesti tornien täytyy olla vähintään yhden pikselin päässä toisistaan ja tämä tarkistetaan laskemalla tornien etäisyydet toisistaan. Tämä etäisyys on kuitenkin liian pieni kulmissa ja tästä johtuu, että tornit saa halutessaa hieman päällekäin.

Tämän voisi korjata lisäämällä tarkistukseen lisäehdon, joka tarkistaa myös tornien kulmittaisen etäisyyden siinä tapauksessa, että ne ovat päällekäin.

Toinen puutteellisuus on tasonvalintanäkymässä. Tasot tulevat näkyviin kolmen napin korkuisissa pylväissä. Jos nappien määrä ei ole kolmella jaollinen, pylvään alaosa jää "tyhjäksi". Eli siellä ei ole mitään, mutta se silti peittää taustaa.

Tämän voisi korjata muuttamalla nappien sijoittelulogiikkaa valintanäkymään.

Parhaat ja heikoimmat kohdat

Erityisen hyvänä toteutuksena pidän ShootingTowerien kohteenvalintaa. Case classien käyttö sopii erityisen hyvin tähän tarkoitukseen ja vastaava graafinen toteutus on onnistunut myös hyvin. Pelaaja voi valita millä logiikalla torni valitsee kohteen klikkaamalla nappia, jolloin valinta otetaan talteen ja nappi jää “alas”. Uuden kohteen valinta painaa sitä vastaavan napin alas, ja kaikki muut napit jäävät valitsemattomiksi.

Yksi ohjelman heikoista kohdista on tiedostojen käsittely. Ohjelma kyllä lukee ja tulkitsee tiedostot oikein, mutta niiden täytyy noudattaa annettua formaattia täsmällisesti. Jos jokin tiedosto on viallinen, ohjelma ei käynnisty, vaan antaa virheilmoituksen.

Tiedostojenkäsittelyä olisi voinut parantaa esimerkiksi antamalla virheellisille arvoille jotakin oletusarvoja, joita käytetään virheellisten arvojen sijaan.

Poikkeamat suunnitelmasta

Vaikka projektia koodatessa suunnitelma jäikin vähän sivummalle, näin jälkeenpäin suunnitelmaa lukiessa ei ole juurikaan poikkeamia. Ajattelutapa ei projektin aika muuttunut ja jonkinlainen käsitys tavoitteesta takaraivoissa koko ajan oli. Tietysti joitakin luokkia täytyi luoda, kun niitä tarvittiin, vaikkei niitä suunnitelmassa ollutkaan.

Ajankäyttö oli suunnitelmassa esitettyjen asioiden kannalta arvioitu hieman yläkanttiin, mutta suunnitelmassa mainitsemattomia asioita oli projektin aikana luematon määrä, mikä kasvatti projektin ajankäytön kokonaismäärää huomattavasti.

Toteutusjärjestystä ei suunnitelmassa ollut kovin yksityiskohtaisesti selitetty, mutta kuten oli suunniteltu, niin käyttöliittymän perustoiminnot luotiin ensiksi, jotta toimintojen testaus myöhemmin helpottuisi.

Toteutunut työjärjestys ja aikataulu

Todellinen toteutusjärjestys ja vaiheiden kesto on melko helppo nähdä git-committien ajankohdista. Yleisellä tasolla toteutusjärjestys näyttää suurin piirtein tältä:

- Projektin luonti ja perusluokkien lisäys (15.3)
- Projektin varsinainen aloitus ja käyttöliittymän alkeellinen toteutus (13.4)
- Tornien ja vihollisten demoluokkien kehitys lähes valmiiksi (19.4)
- Tornien luonti tiedostoista ja käyttöliittymän parantelu (21.4)
- Muiden luokkien luonti tiedostoista (23.4)
- Käyttöliittymän ja luokkien viimeistely (25.4)
- Pelin tasapainottaminen (26.4)

Kuten jo aiemmin sanottu, suunnitelmassa ei ollut selkeää toteutusjärjestystä, vaan järjestys muotoutui sen mukaan mitä milloinkin tarvittiin projektin etenemisen kannalta.

Arvio lopputuloksesta

Ohjelma on kokonaisuudessaan melko yksinkertainen, mutta hyvin toimiva toteutus klassisesta tornipuolustuspelistä. Ohjelman hyviä puolia ovat sen käyttöliittymän monipuolisuus ja helppokäyttöisyys sekä mahdollisuus luoda omia torneja ja vihollisia peliin helposti.

Mitään oleellisia puutteita ohjelmassa ei pitäisi olla, mutta esimerkiksi torneja voi laittaa kentälle niin, että niiden kulmat ovat päällekäin, mikä ei tietenkään ole tarkoitus. Tämä johtuu tarkistusalgoritmin pienestä puutteesta, se katsoo vain tornien välistä etäisyyttä, ja se on kulmittaisessa tapauksessa liian pieni, joten kulmat saa päällekäin.

Tulevaisuudessa ohjelmaan voi lisätä esimerkiksi uusia tornityyppejä ja torneille parannus- ja päivitysvaihtoehtoja pelin aikana. Myös alunperin laittomia rakennusalueita voisi lisätä ratoihin. Ehkä suurin parannus olisi pallo- ja neliögrafiikan päivitys parempaan.

Luulen, että nykyinen luokkajako toimii pelin kuvaamiseen hyvin. Joitakin menetelmiä – kuten aiemmin mainittu tornien päällekäisyys – voisi parannella ja hioa, ja vastaavasti myös käyttöliittymää, vaikka se toimiikin pelin pelattavuuden kannalta jo nyt hyvin.

Ohjelman joitakin osia voi laajentaa helposti, joitakin taas ei niin helposti. Esimerkiksi uusien torni- ja vihollistyyppien määrittely ei tuota ohjelman rakenteen kannalta hankaluuksia. Kaikki uudet luokat, jotka perivät Tower-piirreluokan ovat pelin kannalta samankaltaisia ja niitä käsitellään samalla tavalla, vastaavasti viholliset perivät Enemy -piirreluokan ja ovat myös keskenään samankaltaisia. Mutta toisaalta käyttöliittymän päivittäminen vaatii jo enemmän vaivaa, sillä swing-kirjastolla luotujen käyttöliittymien sisältö ja asettelu ei ole kovin dynaamista.

Viitteet

- Project: Pentaminoes https://github.com/Gugguru/Pentaminoes
- Project: ImageFilters
- https://plus.cs.hut.fi/studio_2/2017/
- http://stackoverflow.com/questions/13381797/filtering-a-scala-list-by-type
- http://stackoverflow.com/questions/7209728/how-to-pattern-match-multiple-values-in-scala
- http://stackoverflow.com/questions/13093108/store-a-sequence-of-specific-class-types-in-scala
- http://stackoverflow.com/questions/4652095/why-does-the-scala-compiler-disallow-overloaded-methods-with-default-arguments
- http://stackoverflow.com/questions/10375633/understanding-implicit-in-scala
- http://stackoverflow.com/questions/11344814/why-java-lang-object-can-not-be-cloned
- http://stackoverflow.com/questions/7425558/get-only-the-file-names-using-listfiles-in-scala
- http://stackoverflow.com/questions/8110975/how-to-make-a-rectangle-in-graphics-in-a-transparent-colour