Tekninen suunnitelma

Defence

Tornipuolustus

Samuli Mononen
12.3.2017

Ohjelmoinnin peruskurssi Y2 Samuli Mononen
Tornipuolustus 426477
Tekninen suunnitelma EST
12.3.2017 3. vuosikurssi

Sisällys

1. Ohjelman rakennesuunnitelma

Ohjelman rakenne on esitetty suunnitelman liitteessä 1, UML-kaaviona.

Ohjelman tärkeimmät luokat ovat GUI ja Game. GUI huolehtii grafiikan piirtämisestä ja Game logiikan tarkastelusta. Gamen metodit add_enemy ja add_missile lisäävät pelikenttään vihollisia ja ammuksia. Gamen check_missile_hits tarkastelee, ovatko ammukset saavuttaneet määränpäänsä ja check_enemy_escapes tutkii, onko vihollisia päässyt karkuun.

GUI:n metodeista useat liittyvät käyttöliittymän alustamiseen. Init_window ja init_buttons luovat pelaajalle ikkunan, joka sisältää tarvittavat informaatiot pelin tiedoista ja mahdollisuuden keskeyttää peli ja rakentaa torneja. GUI hyödyntää Tower-, Enemy-, Missile-, Text- ja LifeGraphicsItem luokkia, jotka jokainen huolehtivat nimensä mukaisesti kyseisen luokan graafisesta objektista. Ne myös tarkastelevat, mikäli kyseistä objektia klikataan.

Game hyödyntää Board luokkaa, johon tallennetaan reittipisteet ja tornien sijainnit. Board käyttää Square luokkaa, joka kuvaa yhtä Boardin neliötä. Boardin metodi add_tower huolehtii tornien lisäämisestä. Square luokan is_empty tutkii onko ruutu vapaa uudelle tornille, set_route asettaa kyseiseen ruutuun reittipisteen ja set_tower asettaa kyseiseen ruutuun tornin.

Tower luokka mallintaa tornia. Tornit voivat olla erilaisia ja siksi sen tyyppi määritetään setType metodilla. Tornin sijainti asetetaan set_game metodilla ja vihollisten liikettä tornin läheisyydessä tutkitaan enemy_in_range metodilla. Tornit luovat Missile objekteja, jotka kuvaavat ammusten logiikkaa.

Game sisältää myös vihollisia, joita mallinnetaan Enemy luokalla. Enemy luokan set_world asettaa vihollisen peliin. Direction tarkastelee vihollisen suuntaa, eli suuntaa, johon se on liikkumassa. Move metodi liikuttaa vihollista sen nopeus-ominaispiirteen verran oikeaan suuntaan ja lower_hitpoints vähentää sen elämäpisteitä ammuksen osuessa siihen. Play_death_sound hoitaa kuolemisefektiäänen toistamisen vihollisen elämäpisteiden loppuessa ja samalla increase_money lisää peliin rahaa ja increase_points lisää peliin pisteitä.

2. Käyttötapauskuvaus

Pelaajan käynnistäessä ohjelman hänelle avautuu pelin käyttöliittymä ja pelikenttä. Luokka GUI huolehtii näiden piirtämisestä käyttäen apunaan erilaisia GraphicsItem luokkia. Käyttöliittymään piirretään esimerkiksi pelaajan elämät, pisteet ja rahat luokan Gamen mukaan. Game huolehtii myös pelikentästä eli siihen GUI:n piirtämien ammusten, tornien ja vihollisten logiikasta, kuten esimerkiksi paikkatiedoista.

Pelin tarkoituksen mukaisesti pelaajan on tarkoitus torjua vihollisten hyökkäys tornien avulla. Pelaaja asettaakin torneja pelikentälle klikkaamalla kyseisen tornin kuvaa. Tällöin TowerGraphicsItem luokka havaitsee klikkauksen ja siirtää käskyn Gamelle. Game tarkistaa, että tornin rakentamiselle on tarpeeksi rahaa. Mikäli rahaa on tarpeeksi, pelaaja voi klikata haluttua kohtaa pelikentältä, jolloin Board luokka saa käskyn tarkistaa, että valittu ruutu on vapaa. Jälleen positiivisessa tapauksessa käsky etenee siten, että Gamen taulukkoon sijoitetaan haluttu torni.

Pelaajan rakennettua haluamansa määrän torneja hän jää odottamaan vihollisten hyökkäystä. Enemy luokaksi nimetyt viholliset liikkuvat aina Boardin route_points listan seuraavaa alkiota kohti, kunnes ne saavuttavat viimeisen alkion, eli pääsevät maaliin. Samalla Tower luokaksi nimetyt tornit tarkastelevat, ovatko viholliset heidän ampumaetäisyydellä oman metodinsa avulla ja tämän jälkeen joko ampuvat oman erityispiirteensä mukaan tai eivät.

Vihollista ammuttaessa sen elämäpisteitä vähennetään ja mikäli ne loppuvat kokonaan, toistetaan viholliselle ominainen kuolemisääni ja lisätään Gameen pisteitä ja rahaa. Mikäli vihollinen pääsee karkuun, Gamesta vähennetään elämä. Kun Gamen elämät kuluvat loppuun, peli päättyy ja ruudulle tulostetaan saavutettu pistemäärä.

3. Algoritmit

Ohjelmassa on paljon triviaaleja yhteen- ja vähennyslaskuja vaativia algoritmeja, kuten kokonaispisteiden, vihollisten elämäpisteiden ja rahamäärän laskenta.

Ohjelman vaativammat algoritmit puolestaan liittyvät ammuksien ja vihollisten liikuttamiseen. Laskenta tapahtuu siten, että ensin lasketaan määränpään ja nykyisen sijainnin etäisyys Pythagoraan lauseen avulla. Tämän jälkeen muodostetaan vektori nykyisestä sijainnista määränpäähän ja normalisoidaan se. Tämän jälkeen riittää, että normalisoitua vektoria kerrotaan vihollisen tai ammuksen nopeudella.

Sekä ammusten että vihollisten liikkeestä tulee myös tarkastella, milloin ne saavuttavat määränpään. Tämän selvittämiseksi ohjelma vertaa jokaisella laskentakierroksella nykyisen sijainnin ja määränpään välistä etäisyyttä. Mikäli etäisyys on pienempi, kuin mitä ammus tai vihollinen liikkuisi kyseisellä vuorolla, ammuksen tapauksessa sen lasketaan osuneen maaliin ja vihollisen tapauksessa vihollisen tulee vaihtaa määränpäätään seuraavaan reittipisteeseen ja liikkua sitä kohti ylijäänyt matka. Vihollisen osalta tulee myös tarkastella, että reitin loppuun päästessään se tulkitaan karanneeksi.

Liikkeen laskentaan on valittu koordinaattien käyttäminen peliruudukon ruutujen sijasta siksi, että ammukset eivät liiku ruutujen, vaan koordinaattien mukaan. Tällöin vältytään kahden eri laskennan ohjelmoinnilta. Vihollisien ei myöskään haluta liikkuvan hyppien ruudusta toiseen, vaan niiden on tarkoitus liikkua sulavasti reittiä pitkin. Myös tästä syystä liikettä ei voida tehdä vaan ruudusta toiseen.

Suunnan selvittämiseen olisi voinut käyttää myös kulmaa, jolloin liike x- ja y-suunnassa olisi voitu laskea trigonometrian avulla. Suurimmaksi haasteeksi olisi kuitenkin muodostunut suunnan suhteellisuuden määrittäminen. Kahden pisteen muodostaman janan kulma koordinaattiakseliin riippuu koordinaattiakselin valinnasta. Negatiivisiltakaan kulmilta ei olisi vältytty, sillä määränpää voisi missä suunnassa tahansa.

Mikäli projekti ei muutu ylivoimaisen haastavaksi, toteutetaan ohjelmaan myös pelikentälle, jossa viholliset eivät liikkuisi määrättyä, vaan vapaata kenttää pitkin. Tämän laskentaan käytettäisiin Dijkstran algoritmia.

4. Tietorakenteet

Ohjelmassa tietoja tallennetaan listoihin, taulukoihin ja tupleihin. Taulukkoon tallennetaan esimerkiksi pelikenttä, johon pelaaja voi asettaa torneja. Listoihin puolestaan tallennetaan ammukset ja viholliset. Tupleihin tallennetaan ammusten ja vihollisten sijainnit liukulukuina. Lisäksi tuplea käytetään tornien sijainnin tallentamiseen kokonaislukuna, sillä niiden osalta riittää tietää missä kohtaa taulukkoa ne sijaitsevat.

Listat on valittu niiden muuttuvuuden ja helpon käsittelyn takia. Pythonilla listan läpikäyminen on helppoa ja esimerkiksi jokaisen ammuksen läpikäyminen esimerkiksi sijainnin muuttamista ja osumisen tarkistusta onnistuu vaivatta. On myös tärkeää, että ammusten ja vihollisten luominen ja poistaminen onnistuvat vaivatta, sillä molempia tuhoutuu ja syntyy uudelleen jatkuvasti. Näin ollen muuttuvana tietorakenteena lista täyttää vaaditut kriteerit. Lista pitää myös huolen järjestyksestä, mikä on hyödyllistä vihollisten järjestyksen selvittämiseen. Tornit eivät nimittäin välttämättä ammu kaikkiin vihollisiin yhtä aikaa, vaan niiden täytyy osata valita jokin vihollinen. Pisimmälle edennyt vihollinen on tähän toimiva ratkaisu ja se saadaan selville yksinkertaisesti listan järjestyksestä.

Tuple on toimiva tiedostomuoto sijainnin tallentamiseen, sillä siihen saadaan mahtumaan juuri sopivasti x- ja y-koordinaatti. Myös lista tai jopa merkkijono toimisivat, mutta niiden käyttäminen ei ole yhtä yksiselitteistä ja niitä käyttäessä tulisi huomioida merkittävästi enemmän virhetilanteita.

Käytettävien tiedostomuotojen osalta tietoa tallennetaan suoraan koodiin, tekstitiedostoihin, kuva (JPG, PNG) ja äänitiedostoihin (WMA, MP3). Kuva- ja äänitiedostot ovat osittain eri formaateissa, koska käytän ilmaisia vapaaseen käyttöön julkaistuja ”spritejä” ja ääniefektejä, joten en voi olettaa saavani kaikkia tiedostoja samassa tiedostomuodossa.

Osa tiedoista, kuten mahdollisesti esimerkiksi ammuksien tiedot (esimerkiksi voimakkuus ja nopeus), kirjoitetaan suoraan koodiin, sillä tällöin ammusten luontia varten ei tarvitse kirjoittaa metodia, joka lukisi tiedot erillisestä tiedostosta. En myöskään voi luoda vain yhtä metodia tiedoston lukemista varten, sillä esimerkiksi ammukset, viholliset ja tornit tarvitsevat jokainen eri tietoja, joten tiedostoon tallentaminen täytyy tehdä jokaiselle luokalla hieman eri tavalla.

Tekstitiedostoja kuitenkin käytetään, että peli saadaan helpommin tasapainotettua. Ohjelmoinnin viimeisessä vaiheessa on tarkoitus muuttaa tornien ja vihollisten parametreja niin, että pelistä saadaan mielekkään haastava ja tasapainottaa tornit siten, ettei niissä ole yhtä selvästi parasta vaihtoehtoa. Tätä varten on selkeämpää, että voidaan muuttaa vain tekstitiedostossa olevia parametreja, kuin muokata itse koodia.

5. Aikataulu

Ohjelmointi alkaa viikon 11 lopussa ja sitä jatketaan viikoilla 12, 13, 15, 17 ja 18. Työtunnit ja ohjelmoitava osa-alue on esitetty alla olevassa taulukossa. Myös ohjelmoinnin etenemisjärjestys selviää samasta taulukosta.

Viikko	**Työmäärä**	**Ohjelmoinnin tavoite**
11	6 h	Game ja Board siten, että pelikenttään voidaan asettaa torneja
12	20 h	Tower, ainakin yksi TowerType, Square, GUI, TowerGraphicsItem
13	20 h	Coordinates, Enemy, loput TowerTypet ja TextGraphicsItem
15	30 h	EnemyGraphicsItem, MissileGraphicsItem ja Game viimeistään tässä vaiheessa täysin valmiiksi
17	25 h	Grafiikkojen syöttäminen GraphicsItemeihin, musiikin ja ääniefektien syöttäminen, vihollis-, torni- ja ammustiedostojen kirjoittaminen
18	6 h	Varattu grafiikkojen viimeistelylle ja tornien tehokkuuden tasapainottamiselle

6. Yksikkötestaussuunnitelma

Testaamista varten luodaan oma test.py tiedosto, johon kirjoitetaan testit pelin poikkeustapauksia varten. Näitä ovat esimerkiksi tilanteet, joissa pelaaja yrittää luoda tornia, vaikka hänellä ei olisi rahaa tai että hän yrittää luoda tornia ruutuun, joka on jo varattu ja se on varattu vihollisten reitille.

Testausta toteutetaan myös graafisesti. Aina kun ammuksen, tornin, vihollisen tai tekstikentän grafiikkaelementti on implementoitu, sen toiminta testataan ajamalla ohjelmaa ja tarkastelemalla, että grafiikka piirtyy oikein.

Pelin loppuvaiheessa tarkistetaan peliä pelaamalla, että ammukset piirtyvät tornien ja vihollisten päälle ja että nuolien osumis- ja vihollisten kuolemisäänet toistetaan oikein.

Tarkempi testilistaus on esitetty jo yleisessä suunnitelmassa ja sama listaus on esitetty tämän suunnitelman liitteessä 2.

Samuli Mononen
426477
EST
3. vuosikurssi

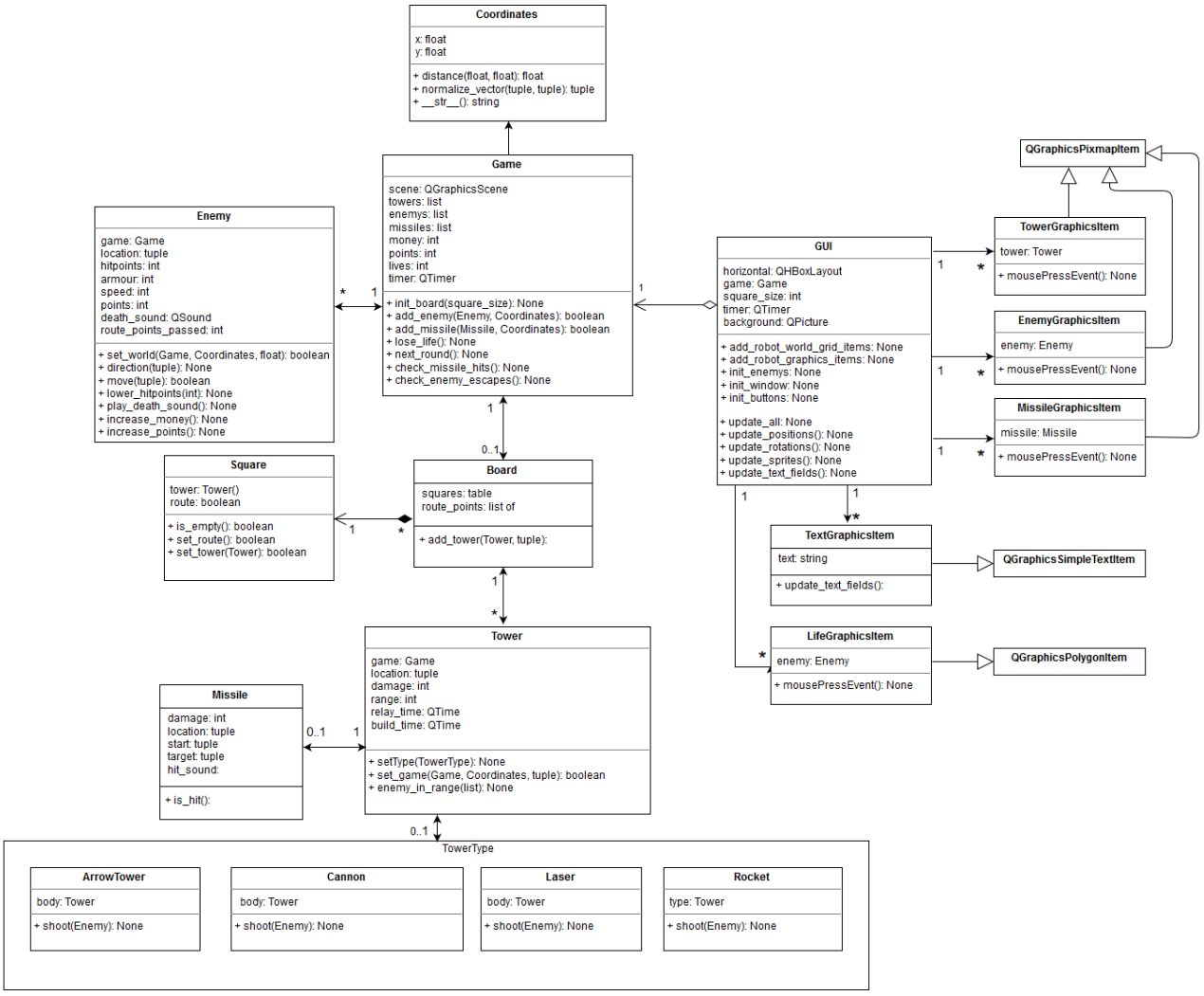
7. Kirjallisuusviitteet ja linkit

Qt 5.8 Documentation	http://doc.qt.io/qt-5/index.html
QGraphicsSimpleTextItem	https://doc.qt.io/qt-5/qgraphicssimpletextitem.html
QGraphicsPixmapItem	http://doc.qt.io/qt-5/qgraphicspixmapitem.html
QGraphicsPolygonItem	http://doc.qt.io/qt-5/qgraphicspolygonitem-members.html
QSound	http://doc.qt.io/qt-5/qsound.html
The Python Standard Library	https://docs.python.org/3.3/library/index.html
Isaiah658's Pixel Pack #1	http://opengameart.org/content/isaiah658s-pixel-pack-1
Sithjester's RMXP Recources	http://untamed.wild-refuge.net/rmxpresources.php?characters
Dijkstran algoritmi	https://fi.wikipedia.org/wiki/Dijkstran_algoritmi
Moving from A(x,y) to B(x1,y1)	http://gamedev.stackexchange.com/questions/23447/moving-from-ax-y-to-bx1-y1-with-constant-speed

Samuli Mononen
426477
EST
3. vuosikurssi

8. Liitteet

Liite 1: UML-kaavio



Liite 2: Esimerkki testilistauksesta

- Kaikkia torneja tulee voida luoda
- Torneja pitää pystyä asettamaan kentän jokaiseen ruutuun
- Torneja ei tule voida asettaa reitille
- Torneja ei tule voida asettaa toistensa päälle
- Ammuksen osuessa viholliseen
    - Ammuksen tulee kadota
    - Osumisefektin ääni tulee toistaa
    - Mahdollinen efektikuva tulee piirtää
    - Vihollisen elämäpisteiden tulee vähentyä
        - Mikäli vihollisen elämäpisteet loppuvat
            - Vihollinen tulee poistaa
            - Pisteiden tulee lisääntyä
            - Rahan tulee lisääntyä
- Pelaajan elämien loppuessa
    - Pelin tulee päättyä
    - Ennätystulos tulee tallentaa ”Highscores.txt” tiedostoon
- Vihollisen karatessa
    - Pelaajan elämien tulee vähentyä
    - Vihollisen karkaamisefektin ääni tulee toistaa