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

Suunnitelman tärkein osuus. Ohjelman erottelu tärkeimpiin osakokonaisuuksiinsa, suunnitellun luokkajaon esittely. Minkälaisilla luokilla kuvaat ohjelman ongelma-aluetta? Mitä ongelman osaa kukin luokka mallintaa? Mitkä ovat luokkien väliset suhteet? Entä millaisia luokkia tarvitaan ohjelman käyttöliittymän kuvaamiseen? Miettikää mahdollisia muita ratkaisumalleja ja perustele valittu ratkaisu. Jos suinkin mahdollista, liittäkää mukaan jonkinlainen graafinen luokkakaavio (voitte käyttää esim. UML-luokkakaavionotaatiota, mutta se ei ole millään muotoa pakollista). Esittele luokkien keskeiset metodit. Huom. oleellista on vain se, mitä metodeilla tehdään, ei se, miten ne sisäisesti toimivat.

2. Käyttötapauskuvaus

Pelaajan käynnistäessä ohjelman hänelle avautuu pelin käyttöliittymä ja pelikenttä. Luokka GUI huolehtii näiden piirtämisestä käyttäen apunaan erilaisia GraphicsItemeja.

Esittäkää ainakin yksi realistinen ohjelman käyttötapaus, eli kuvaus tilanteesta, jossa käyttäjä käynnistää ohjelman ja tekee sillä joitakin ohjelmalle tyypillisiä asioita. Kertokaa sekä se, mitä toimenpiteitä käyttäjän on suoritettava päämääriensä saavuttamiseksi (eli minkä käyttöliittymän osien kanssa tämä on tekemisissä ja miten), että se, mitkä ohjelmanne osat aktivoituvat kussakin vaiheessa “kulissien takana” ja suorittavat tarvittavat tehtävät. Koodiyksityiskohdat eivät ole tässä kiinnostavia, vaan korkeamman tason työnjako.

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

Kuvatkaa tässä osiossa kuinka aiotte testata ohjelman keskeisimpiä osia toteutuksen edetessä. Koko ohjelman kaikkia ominaisuuksia ei ole tarkoitus käydä läpi, vaan keskittyä tässä ehkä ohjelman “ydinmetodeihin”, jotka tekevät sen keskeisimmän työn. Kuvatkaa

jälleen yleisluontoisesti, kuinka aiotte metodeja (valitkaa muutama) niiden valmistuttua kokeilla. Jälleen voi esittää keskeisiä syötteitä joilla ohjelman tulee toimia, mitä metodin tulee tällöin palauttaa ja mikä sen vaikutus ohjelman olioihin tulee olla. Vastaavasti voi pohtia sitä, kuinka metodeja voisi testata helposti ilman että tarvitsee toteuttaa valtavia apurakennelmia. (valitettavasti näiltä ei voi aina välttyä) Yksikkötestausta kannattaa sitten projektia toteuttaessa tehdä sopivassa määrin, jotta ei turhaan joudu etsimään bugia uusista koodiriveistä, kun virhe onkin jossain joka tehtiin aikaa sitten.

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

8. Liitteet

Lisäksi suunnitelmassa saa olla liitteitä, aiheesta riippuen.