Aalto-yliopisto
Sähkötekniikan korkeakoulu
Elektroniikka ja Sähkötekniikka

Projektisuunnitelma

Tornipuolustuspeli

Joel Lavikainen
479848
3. vuosikurssi
12.3.2017

Sisältö

1 Ohjelman rakennesuunnitelma

Peliin on tarkoitus toteuttaa graafinen käyttöliittymä käyttämällä PyQT5-kirjastoa. Tulevaisuuden varalta haluan kuitenkin pitää pelilogiikan ja graafisen käyttöliittymän erillään toisistaan, jolloin peli voidaan jakaa karkeasti kahteen osa-alueeseen: pelilogiikkaan ja graafiseen käyttöliittymään.

1.1 Pelilogiikka

Pelilogiikan kannalta tärkeitä luokkia ovat:

- Torni (Tower)
- Vihollinen (Enemy)
- Pelimaailma (GameWorld)
- Ruutu (Square)

1.1.1 Pelimaailma

Pelimaailma toimii yleiskuvauksena pelikentästä, ja se hallitsee pelin kulkua, vihollisia ja torneja. Pelimaailma pitää kirjaa kentän ruuduista, elossa olevista vihollisista ja torneista. Kentän tiedot luetaan pelin alussa kenttätiedostosta ja pelimaailma olioon alustetaan listaan vastaavat ruutu oliot, kuin kenttätiedostossa määritetään. Kenttätiedostosta luetaan myös vihollisaaltojen tiedot, jotka alustetaan pelimaailma olioon listaan.

Keskeisiä metodeja pelimaailmalle ovat:

- add_tower(), käytetään uusien tornien kenttään lisäämiseksi
- add_enemy(), käytetään vihollisten lisäämiseen aloitusruutuun
- remove_dead_enemies(), metodi tarkistaa mitkä viholliset ovat kuolleita ja poistaa nämä
- next_wave(), käynnistää seuraavan kierroksen kentässä

1.1.2 Torni

Tornipuolustuspelin tärkein olio eli torni sisältää tietoa tornin ominaisuuksista ja sijainnista. Torni on myös tietoinen pelimaailmasta, johon se kuuluu. Myös tornin mahdolliset päivitykset ja nykyinen päivitystaso säilytetään torni oliossa. Eri tornit ja niitä vastaavat tiedot luetaan erillisestä konfiguraatiotiedostosta, joka on xml muodossa. Tämä mahdollistaa uusien tornien luonnin ja tasapainottamisen lennosta ilman, että arvoja oltaisiin kovakoodattu. Tornin tyypin kuvausta varten on olemassa lueteltu tyyppi TowerType. Myös eri päivitysten kuvaamista varten käytetään lueteltua tyyppiä UpgradeType.

Keskeisiä metodeja tornille ovat:

- attack(), tornin hyökkäysmetodi. Tästä voi olla eri variaatioita riippuen tornin tyypistä
- upgrade(), päivittää halutun päivityksen torniin

1.1.3 Vihollinen

Pelissä kentällä olevaa reittiä pitkin kulkevat viholliset toteutetaan omana olionaan. Kyseinen olio sisältää tietoa vihollisen elämistä, nopeudesta, paikasta ja rahapalkkiosta. Vastaavasti kuin torneilla viholliset luetaan omasta konfiguraatiotiedostostaan, joka on myös xml formaatissa. Vihollisen eri tyyppien kuvaamiseen on käytössä lueteltu tyyppi EnemyType.

Keskeisiä metodeja viholliselle ovat:

- move(), vihollinen liikkuu seuraavaan ruutuun ennaltamäärätyllä reitillä
- damage(), kutsutaan, kun torni tekee vauriota viholliseen. Muuttaa myös vihollisen kuolleeksi, jos elämät menevät nollaan
- is_at_goal(), tarkastaa onko vihollinen päässyt maaliruutuun

1.1.4 Ruutu

Ruutu on kuvaus pelikentän ruudukon yhdestä ruudusta. Ruudun eri tyyppejä kuvataan luetellulla tyypillä SquareType. Ruudun tyyppejä ovat reitti-, alku- ja maaliruudut sekä muut ruudut, joihin voidaan sijoittaa torneja. Ruutuihin sisällytetään ruudun päällä oleva olio eli joko torni tai vihollinen. Viholliset syntyvät alkuruutuun ja tuhoutuvat maaliruudussa vähentäen pelaajan pisteitä.

Keskeisiä metodeja ruudulle ovat:

- set_tower(), käytetään tornin asettamiseen ruutuun
- set_enemy(), vastaava kuin set_tower(), mutta viholliselle

1.2 Graafinen käyttöliittymä

Graafisen käyttöliittymän kannalta tärkeitä luokkia ovat:

- GUI
- Ammus (Projectile)
- TowerGraphicsItem
- EnemyGraphicsItem

1.2.1 GUI

Erillään olevaa graafista käyttöliittymää kuvataan hallinnoivalla oliolla GUI, joka pitää sisällään tiedon kentän, vihollisten ja tornien piirrosolioista. GUI oliossa on myös logiikan ja graafiikan päivittämistä hallinoivat ajastimet. Pelilogiikka liitetään graafiseen käyttöliittymään antamalla GUI oliolle oma pelimaailma olio.

Keskeisiä metodeja GUI:lle ovat:

- add_enemy_graphics_items(), lisää graafisen esityksen vihollisille, joilta se puuttuu
- add_tower_graphics_items(), vastaava kuin add_enemy_graphics_items()
- add_map_squares(), piirtää ruudukkoon oikeat grafiikat
- init_window(), alustaa graafisen käyttöliittymän ikkunan
- init_buttons(), alustaa käyttöliittymään painikkeet ja liittää niihin oikeat toiminnot
- update_enemies(), päivittää vihollisten uuden sijainnin grafiikkaan

1.2.2 Ammus

Ammus on pelkästään graafinen efekti esittämään tornien hyökkäämistä vihollisiin. Eri ammukset luetaan tornien xml-tiedostosta. Ammusten tyyppien esittämistä varten on lueteltu tyyppi ProjectileType. Perii Qt:n QGraphicsItem luokan kuvien esittämistä varten.

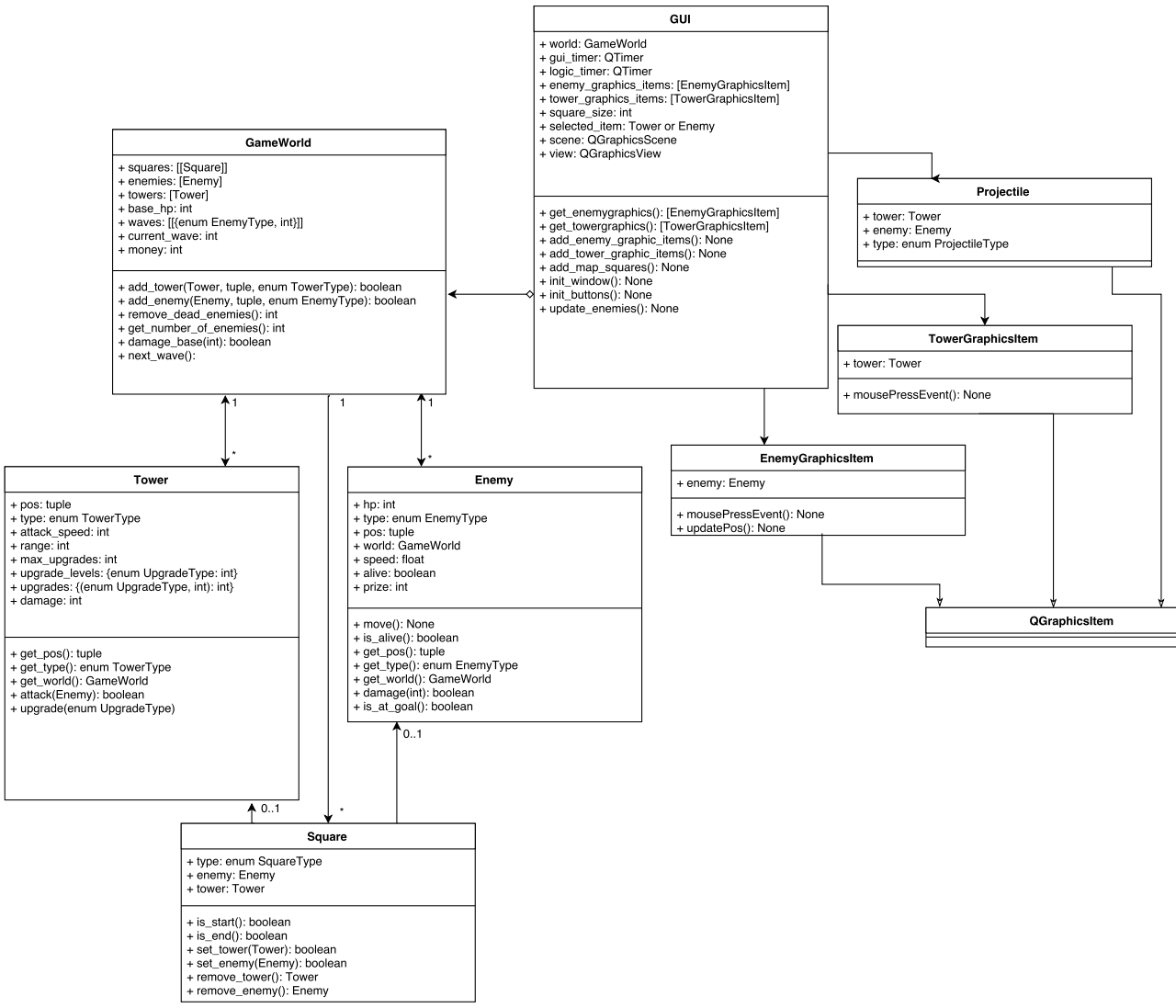
1.2.3 TowerGraphicsItem

Tornien grafiikkaa varten on TowerGraphicsItem olio, johon liitetään aina sitä vastaava torni. TowerGraphicsItem perii Qt:n QGraphicsItem luokan, jolloin sille on määritelty valmiiksi paljon perusominaisuuksia, kuten kuvien esittäminen. TowerGraphicsItem toteuttaa uudestaan hiirelläklikkausmetodin mousePressEvent(), jotta torni voidaan aktivoida päivitettäväksi ja tietojen näyttämistä varten.

1.2.4 EnemyGraphicsItem

Vastaavasti vihollisten grafiikka esitetään EnemyGraphicsItem oliolla, johon liitetään vastaava vihollinen. Perii myöskin QGraphicsItem luokan. Hiirellä klikkaamalla saadaan vastaavasti vihollisen tiedot esille ruudulle. Tärkeänä metodina on update_pos(), jolla päivitetään grafiikan sijainti vastaamaan liitetyn vihollisen koordinaatteja.

1.3 UML-kaavio



Kuva 1: UML-kaavio luokkien välisistä suhteista

2 Käyttötapauskuvaus

Pelaaja aloittaa pelin valitsemalla graafisesta käyttöliittymästä uuden pelin. Tämä käynnistää maailman luonnin, jolloin luodaan GameWorld ja GUI oliot. Näihin luetaan konfiguraatio- ja kenttätiedostoista oikeat tiedot vihollisille, torneille ja kentälle. Tällöin käyttöliittymään piirtyy tyhjä kenttä. Seuraavaksi pelaajalle on jonkin kiinteän ajan verran aikaa sijoittaa aloitustorninsa ennen pelin alkua. Pelaaja sijoittaa aloitustorninsa ensiksi valitsemalla tornivalikosta haluamansa tornin aktiiviseksi ja seuraavaksi klikkaamalla sopivaa ruutua pelikentällä. Kun varsinainen peli alkaa, synnytetään aloitusruutuun hiljakseen kierrokseen kuuluvia vihollisia, jotka lähtevät liikkumaan kohti maalia. Peli etenee tällä tavalla itsekseen, kunnes kaikki kierroksen viholliset on tuhottu tai ne ovat päässeet maaliruutuun ja pelaajan elämät ovat loppuneet. Pelaaja voi kierroksen ollessa käynnissä lisätä torneja tai päivittää niitä.

3 Algoritmit

Vihollisten liikkuminen alkuruudusta maaliruutuun on toteutettu hyvin yksinkertaisesta johtuen ennaltamääritetystä reitistä. Viholliset tietävät entuudestaan koko reitin eli mihin ruutuun seuraavaksi liikutaan. Vihollinen pitää kirjaa siitä missä ruudussa on tällä hetkellä ja move() metodia kutsuttaessa vihollinen siirtyy järjestyksessä seuraavaan ruutuun.

Tornit tietävät tarkalleen kaikkien vihollisten sijainnin, minkä takia ne voivat tarkastella toistuvasti ovatko ne ampumaetäisyydellä vihollisesta. Tornien ampumaetäisyys on ruudukossa määritetty etäisyys jokaiseen väli- ja pääilmansuuntaan. Viholliset voivat kuitenkin vain sijaita yhden ruudun levyisellä ennaltamäärätyllä reitillä, joten algoritmin tarvitsee tarkastella vaan etäisyydellä olevia reitin ruutuja.

4 Tietorakenteet

Pelimaailmassa vihollinen ja torni oliot säilytetään listassa, koska niiden määrä muuttuu dynaamisesti pelin edetessä. Eri tyyppien ilmaisemiseen käytetään lueteltua tyyppiä (enum), koska se on kuvaavampi tapa ilmaista kuin pelkät numerot. Lueteltu tyyppi toimii myös mukavasti sanakirjoissa avaimena.

Tornien päivitykset säilytetään sanakirjassa, jonka avaimena on lueteltun tyypin UpgradeType ja kokonaisluvun muodostama tuple. Sanakirjan arvona on päivitystä vastaava ominaisuuden nouseminen kokonaislukuna. Tornien päivitystilanteesta pidetään kirjaa upgrade_levels sanakirjalla, jonka avaimena on päivityksen tyyppi UpgradeType. Kun halutaan päivittää tornia, voidaan ensin kysyä päivitystä vastaava kokonaisluku tältä sanakirjalta, jonka jälkeen oikeaa tasoa vastaava päivitys saadaan toisesta sanakirjasta. Sanakirja on hyvä valinta, koska päivitykset eivät muutu kesken pelin vaan ne alustetaan pelin alussa kerran.

Pelin kierrokset säilytetään listassa, joka sisältää sanakirjoja, joiden avaimina on vihollistyyppi EnemyType ja arvoina vihollisten määrä kokonaislukuna. Esimerkiksi [{enemy_a:3, enemy_b:5}, {enemy_a:10, enemy_b:8}], jolloin ensimmäisellä kierroksella enemy_a olisi 3 kpl ja enemy_b 5 kpl. Vastaavat määrät seuraavalla kierroksella olisivat 10 ja 8 kpl.

5 Aikataulu

Pelin toteuttaminen lähtee liikkeelle pelilogiikalle olennaisten perusolioiden, kuten vihollinen, ruutu ja torni, toteuttamisesta. Peruskäyttöön soveltuvan version toteuttamiseen menee aikaa arviolta yksi tunti.

Seuraavaksi toteutetaan pelimaailma olio ja kenttätiedostojen lukeminen tiedostosta. Tähän kuluu arviolta 3-8h, riippuen kuinka vaikeaksi tiedostoformaatin lukeminen osoittautuu.

Jotta peli saadaan pyörimään on tärkeää toteuttaa graafista esitystä varten GUI, TowerGraphicsItem ja EnemyGraphicsItem oliot. Tähän kuluu arviolta kaksi tuntia, että peruspohja saadaan luotua. Seuraavaksi on hyvä toteuttaa perus logiikan kutsuminen käyttöliittymään, jotta alkeellinen peli saadaan pyörimään ruudulla. Tähän arvion mukaan voisi kulua 5-8h.

Seuraavaksi toteutetaan konfiguraatiotiedostojen luku. Arvioisin tähän kuluvan vain 2-3h, koska käytetty xml-formaatti on yksinkertainen ja siihen löytyy valmis lukukirjasto Pythonista.

Kun konfiguraatiotiedostoista saadaan halutut tiedot, voidaan toteuttaa torneille päivityssysteemi. Tähän arvioin aikaa kuluvaksi 2-3h.

Lisäksi aikaa kuluu graafisen järjestelmän hiomiseen, kuten ammusten lisäämiseen ja vihollisten liikkeen interpolointiin. Arviolta aikaa kuluu 5-8h.

6 Yksikkötestaussuunnitelma

Kenttien luomista voidaan testata yksikkötesteillä esimerkiksi kokeilemalla, että onko haluttu ruutu oikean tyyppinen. Vihollisten ja tornien alustamista voidaan myös tutkia samanlailla yksikkötesteillä tarkastellen niiden ominaisuuksia. Myös vihollisten liikkumista voidaan testata ennalta määrätyllä reitillä tarkistamalla, että onko vihollisen sijainti muuttunut oikeaan ruutuun.

7 Kirjallisuusviitteet ja linkit

XML-tiedostojen lukuun:
https://docs.python.org/3.6/library/xml.etree.elementtree.html
PyQT5 dokumentaatio:
http://pyqt.sourceforge.net/Docs/PyQt5/
Esimerkki kierrosten toteuttamisesta pelissä:
https://www.raywenderlich.com/37701/how-to-make-a-tower-defense-game-tutorial