CS-A1121 Ohjelmoinnin peruskurssi Y2

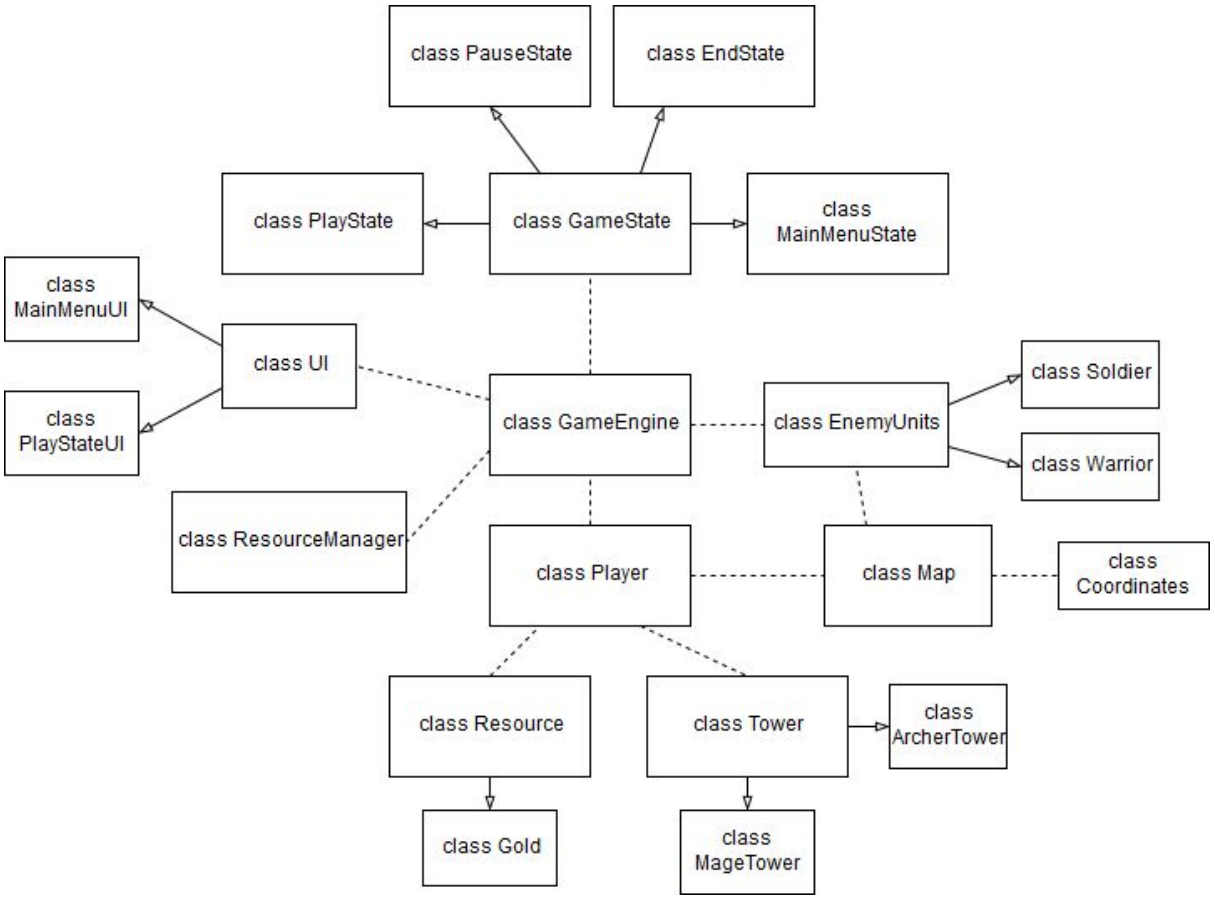
Tekninen suunnitelma, Tornipuolustus

Juuso Pulkkinen, 431284

ENG, KJR, -14

26.2.2019

1. Ohjelman rakennesuunnitelma



Kuva 1: Ohjelman luokkajako

Ohjelman alustava luokkajako on esitetty yläpuolella. Pelin arkkitehtuuri keskittyy GameEngine luokkaan joka alustetaan main funktiossa. GameEngine pyörii PyQT -ikkunassa, ja looppaa (näin päivittäen peliä) niin kauan kunnes ikkuna sammutetaan. GameEnginen tehtävänä on mm. vaihdella ohjelman tilaa.

Peli on suunniteltu toimivan eri tiloissa jotka periävät luokasta GameState. Kun käyttäjä pelaa varsinaisesti peliä peli on PlayState tilassa, kun käyttäjä on päävalikossa peli on

MainMenuState tilassa, kun peli pausetaan peli on PauseState tilassa ja kun pelaaja saavuttaa ehdot voittamiselle tai häviämiselle peli menee EndState tilaan. Jokainen GameState hoitaa itse resurssiensa päivittämisen sekä piirtämisen, joten GameEnginen ei täydy huolehtia niistä.

UI luokalla voidaan määrittää jokaiselle tilalle oma tietynlainen käyttöliittymä. Päävalikossa käytetään MainMenuUI:ta ja pelin aikana PlayStateUI:ta.

ResourceManager luokka huolehtii tekstuurien ja spritejen lataamisesta, jotta muistia pystytään käyttämään tehokkaammin. (Huom. luokka saattaa olla tarpeeton, selviää projektin edetessä)

Player luokka huolehtii pelaajan resurssien sekä rakennusten päivittämisestä. Lisäksi pelaajan syöte todennäköisesti tullaan käsittelemään tässä luokassa (tai vaihtoehtoisesti GameStatessa). Luokka sisältää mm. listan pelaajan omistamista rakennuksista josta on helppo tarkistaa jokaisen rakennuksen HP tilanne, ja poistaa tuhoutuneet rakennukset jolloin niitä ei enää piirretä. Luokista Resource ja Tower periytyvät mahdolliset resurssit ja tornit mitä pelaaja voi rakentaa. Tower luokka sisältää myös metodeja tornien tilanne kyselyille (esim. tyyliä getDamage()) joita Player luokka tarvitsee.

Luokasta EnemyUnits periytyvät kaikki mahdolliset joukot joita vastaan pelaaja taistelee. Joukoilta pitää tehdä erilaisia tilanne kyselyitä jotta esimerkiksi kuolleet joukot voidaan poistaa ikkunasta.

Map luokassa alustetaan pelin kentät. Kentät on suunniteltu luettavan .txt tiedostosta joka sisältää tarvittavat koordinaatit vihollisten joukkojen kulkureitille, tornien paikoille, maalin paikalle jne. Näitä koordinaatteja muokkaamalla on helppo tehdä uusia kenttiä. Luokka Coordinates helpottaa koordinaattien käsittelyssä.

2. Käyttötapauskuvas

Pelaaja avaa pelin. Ohjelman main funktiossa alustetaan PyQt - ikkuna ja instanssi GameEnginestä. Jos ResourceManager luokkaa tullaan käyttämään, sen avulla ladataan tekstuurit ja luodaan spritet tässä vaiheessa. GameEngine asettaa pelin tilaksi MainMenuState. MainMenuState piirtää valikon jossa on nappulat pelin aloittamiselle ja lopettamiselle. Kun pelaaja painaa pelin aloitus nappulaa, GameEngine vaihtaa tilaksi PlayState. Valittu kenttä alustetaan. PlayState luokka piirtää ja päivittää pelin aikana tapahtuvia asioita, samalla kun pelaaja pelaa peliä. Player luokka tarkistaa ja päivittää pelin aikana pelaajan resursseja. Kun pelaaja saavuttaa määrätyn voitto/häviö ehdon peli siirtyy EndStateen, josta pelaaja voi valita joko uuden pelin tai palaamisen takaisin päävalikoon.

3. Algoritmit

Ohjelmassa tarvitaan algoritmeja lähinnä joukkojen liikkumiseen, elämäpisteiden laskemiseen, sekä tornien ampumismekaniikoihin.

Joukkoihin vahingon tekeminen on yksinkertaista: kohteen nykyisistä elämäpisteistä vähennetään tornin hyökkäyspisteet, ja jos erotus on pienempi kuin nolla kohde on tuhottu ja poistetaan pelistä.

Joukot liikkuvat pitkin ennalta määrättyä reittiä ns. Waypointtien välillä. Uniteilla on todennäköisesti eri liikkumisnopeus joten tarvitaan myös algoritmi joka pitää huolen että jokainen unit liikkuu oikean matkan yhden framen aikana. Tämä on helppo laskea trigonometrian avulla kun tiedetään lähtö- ja päätepisteen koordinaatit sekä unitin liikkumisnopeus.

Torneille tarvitaan myös algoritmi hoitamaan kohteen valintaa. Todennäköisesti tornit olisi hyvä ohjelmoida niin, että ne aloittavat aina ampumaan lähintä kohdetta jos niillä ei ole aikaisempaa kohdetta tai niiden aikaisempi kohde kuolee. Tähän tarvitaan taas algoritmia joka laskee etäisyyden kahden pisteen välillä, eli tiedoiksi riittää taas tornin sekä mahdollisen kohteen koordinaatit jolloin trigonometrialla etäisyys on helppo laskea.

4. Tietorakenteet

Ohjelmassa tarvitaan dynaamisia rakenteita. Pelaajan omistamat rakennukset ja vihollisen joukot on aina hyvä tallentaa esimerkiksi listoihin, ja poistella sitä mukaan kun ne tuhoutuvat. Lisäksi GameEngine pystyy hoitamaan GameStatejen vaihtoa jos käytetään listaa jonka viimeisenä objektina on aina haluttu State.

Listoja on helppo manipuloida sekä ne riittävät ominaisuuksiltaan edellä mainittuihin tilanteisiin mainiosti jos jokaiselle rakennukselle ja unitille määritellään erikseen oma id arvo jotta oikea kohde löytyy listasta varmasti. Yksi vaihtoehto olisi myös käyttää esim. Sanakirjoja.

5. Aikataulu

Viikko 10
Pelimoottorin ohjelmointi, PyQt testailut - 10h

Viikko 11
Pelimoottori valmis, GameStatejen ohjelmointi alkaen päävalikosta, jotain piirtoa jo ikkunaan testailuksi - 10h

Viikko 12
Käyttöliittymän ohjemointi, statejen viimeistely, karttojen rakennus - 10-15h

Viikko 13
Tässä vaiheessa pelin olisi engine tasolla hyvä toimia jo täydellisesti, jolloin itse gameplayn lisäämisen tulisi olla melko vaivatonta. Resurssien ja tornien implementointi, grafiikat torneille ja vihollisen joukoille (todennäköisesti neliö/pallo grafiikat). - 15h

Viikko 14
Jatkoa viime viikosta, pelimekaniikkojen lisäilyä / testailua - 15h

Viikko 15
Pelimekaniikkojen lisäilyä / testailua, bugien korjausta, mahdollisten lisäominaisuuksien implementointi - 15-20h

Viikko 16
Pelin tulisi olla valmis. Testailua, bugien korjausta, mahdollisten lisäominaisuuksien implementointi ajan salliessa - 15-20h

6. Yksikkötestaussuunnitelma

Joitakin kohteita missä yksikkötestauksesta on paljon hyötyä ovat esimerkiksi joukkojen “spawnaus” metodit joita voidaan testata eri arvoilla (myös virheellisillä) ja tarkistaa että joukkoja ilmestyy kartalle oikea määrä. Myös hiiren syötteen lukemiselle tulee tehdä testifunktiot (emuloidaan hiiren painallusta eri kohteisiin ja tarkistetaan että ohjelma käyttäytyy halutulla tavalla).

Karttojen lukemiselle tarvitaan myös paljon testejä jotta tarvittavat koordinaatit saadaan luettua oikein, ja että virheet eri paikoissa eivät johda tilanteisiin joihin ohjelman ei pitäisi päätyä.

7. Kirjallisuusviitteet ja linkit

Joitakin resursseja joita varmasti hyödynnetään projektin aikana:

[1] PyQt5:n dokumentaatio: http://pyqt.sourceforge.net/Docs/PyQt5/
[2] Qt5:n dokumentaatio: https://doc.qt.io/qt-5/
[3] Python Standard Library: https://docs.python.org/3/library/
[4] Stackoverflow: https://stackoverflow.com/