|  |
| --- |
| Tekninen suunnitelma |
| Defence |
| Tornipuolustus |

|  |
| --- |
| Samuli Mononen  9.3.2017 |

Sisällys

[Yleiskuvaus 1](#_Toc476837208)

[Käyttöliittymän luonnos 1](#_Toc476837209)

[Tiedostot ja tiedostoformaatit 2](#_Toc476837210)

[Järjestelmätestaussuunnitelma 3](#_Toc476837211)

## 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.

## Käyttötapauskuvaus

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.

## Algoritmit

Sanallinen kuvaus käyttämistänne algoritmeista, eli siitä miten ohjelma suorittaa tarvittavat tehtävät. Esim. miten tarvittava matemaattinen laskenta tapahtuu (kaavat mukaan)? Miten algoritminne löytää lyhimmän tiereitin kahden kaupungin välille? Miten toteuttamanne pelin tekoäly toimii? Kaavioita tms. voi käyttää apuna tarpeen mukaan. Mitä muita ratkaisuvaihtoehtoja olisi ollut? Perustelkaa valintanne!

Tässä kohdassa on siis tarkoitus selostaa ne periaatteet, joilla ongelmat voidaan ratkaista, ei sitä, miten algoritmit koodataan. Siis ei luokkien tai metodien kuvauksia tai muitakaan Pythoniin tai ohjelmakoodiin liittyviä seikkoja tänne! (Vielä rautalangasta: älkää mainitko tässä mitään siitä, millaisilla metodeilla algoritmi saadaan toimimaan, millaisia silmukkarakenteita tarvitsee kirjoittaa, tms. Nämä ovat toteutusteknisiä yksityiskohtia, jotka eivät ole tämän osion, eivätkä oikeastaan koko projektisuunnitelman kannalta kiinnostavia.) Jos ohjelmassa on tarkoitus käyttää joitakin yleisesti tunnettuja algoritmeja kannattaa toki mainita algoritmit nimeltä.

## Tietorakenteet

Minkälaiset kokoelmatyypit/tietorakenteet soveltuvat parhaiten ohjelmassa tarvittavan tiedon varastoimiseen ja käsittelyyn? Miksi? Mitä muita valintamahdollisuuksia olisi ollut? Tarvitsetteko dynaamisia rakenteita (so. muuttuvan kokoisia, esim. listat) vai riittävätkö esim. taulukot? Kun käytätte Pythonin valmiita tietorakenteita, ei niiden tarkkaa määrittelyä tarvitse esittää, kunhan perustelee miksi minkäkin rakenteen on valinnut. Jos taas ohjelmoitte itse jonkin tietorakenteen, on myös sen toimintatapa selostettava.

## Aikataulu

Yrittäkää hahmotella itsellenne aikataulu ja karkea arvio eri vaiheissa kuluvista työtunneista. Se on vaikeaa, mutta opettavaista, kun aikataulua vertaa käytännössä eri vaiheisiin kuluvaan todelliseen aikaan. Älkää heittäkö suunnitelmaan mitä tahansa ensiksi mieleen tulevia lukuja vaan yrittäkää arvioida tilannetta realistisesti.

Kuvatkaa myös suunniteltu etenemisjärjestys, eli missä järjestyksessä ohjelmanne aiotte toteuttaa. Tässä kannattaa jo miettiä ohjelman testausta, ja sitä, mitkä osat muusta koodista käyttävät muita osia. Yleensä ohjelmassa on keskeisiä ominaisuuksia joista kannattaa aloittaa, sekä ominaisuuksia, joiden toteutus ei ole heti olennaista. Muista että kehitettäessä jotkin asiat voidaan korvata tynkäluokilla tai tynkämetodeilla, jotta ohjelma saadaan kääntymään, ja jotta toteutettua koodia voi testata. Kokonaisia luokkia ei myöskään tarvitse toteuttaa alusta loppuun vaan tarkentaa toteutusta aste asteelta, kun työ etenee.

## 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.

1. **Kirjallisuusviitteet ja linkit**

Suunnitteluvaiheessa on syytä selvittää, millaista aiheeseenne liittyvää materiaalia on tarjolla. Kirjastot ja nettihakukoneet ovat resursseja, joita kannattaa hyödyntää. Kertokaa tässä, mitä kirjoja, nettisivuja tai muuta materiaalia olette jo käyttäneet tai suunnitelleet myöhemmin käyttävänne. Kaikki lähteet tulee ilmoittaa, vaikka niihin kuuluisivat jokin käyttämäsi oppikirja ja Pythonin perusluokkakirjastojen API-kuvaus.

1. **Liitteet**

Lisäksi suunnitelmassa saa olla liitteitä, aiheesta riippuen.

[1] Isaiah658's Pixel Pack #1. <http://opengameart.org/content/isaiah658s-pixel-pack-1>. Verkkosivu. Viitattu 9.3.2017.  
[2] Dijkstran algoritmi. <https://fi.wikipedia.org/wiki/Dijkstran_algoritmi>. Verkkosivu. Viitattu 8.3.2017.

[3] Sithjester’s RMXP Resources. <http://untamed.wild-refuge.net/rmxpresources.php?characters>. Verkkosivu. Viitattu 9.3.2017.