**Y2 Strategiapeli –**  **tekninen suunnitelma 21.2.2020**

**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.

-luokkajako, perintä jne.

-alustava uml kaavio (ks. kuva puhelimesta)

**Käyttötapauskuvaus**

Kun käyttäjä käynnistää ohjelman, ohjelma etsii kaikki Pelikentät-kansiossa olevat tekstitiedostot ja lukee ne läpi varmistaakseen, että niissä ei ole virheitä. Virheettömät kentät esitetään valikossa, josta käyttäjä pääsee valitsemaan haluamansa kentän. Kentät esitetään kenttiä vastaavina kuvina, joiden alla on kentän nimi. Nappien avulla käyttäjä voi selata eri kenttiä. Painamalla Valitse-nappia ohjelma lataa valitun kentän. Ohjelma luo peliä varten tarvittavat objektit (käyttöliittymä, pelinohjain, kartta jne.) ja asettelee yksiköt ja maastot kartalle tiedoston mukaisesti (tai antaa pelaajan valita ja asetella yksiköt itse).

Tämän jälkeen on pelaajan vuoro. Pelaaja voi valita haluamansa yksikön klikkaamalla sitä. Kun pelaaja klikkaa yksikköä, valittu yksikkö tallentuu pelinohjaimeen. Samalla polunhaku laskee, mihin kaikkiin ruutuihin yksikkö kykenee liikkumaan ja antaa tästä tiedon käyttöliittymälle, joka värjää ruudut, joihin liikkuminen on mahdollista. Pelaaja valitsee jonkin ruudun klikkaamalla sitä, jolloin yksikkö liikkuu kyseiseen ruutuun. Yksikön grafiikka saa tiedon muutoksesta ja päivittää yksikön sijainnin ruudukossa. Tämän jälkeen pelaaja voi vielä käyttää jotain yksikön kyvyistä, jos lähellä on ainakin yksi mahdollinen kohde. Kun pelaaja painaa käyttöliittymän oikeassa ylälaidassa sijaitsevaa Hyökkää-nappia, pelinohjain rekisteröi, että pelaaja on valitsemassa hyökkäyksen kohdetta. Lisäksi käyttöliittymä värjää mahdolliset kohteet punaisiksi. Kun pelaaja klikkaa yhtä punaisista kohteista, kohde vastaanottaa tiedon hyökkäyksestä ja ottaa vahinkoa. Yksikön elämäpalkki päivittyy vastaamaan uutta tilannetta. Hyökännyt yksikkö on nyt käyttänyt vuoronsa loppuun ja yksikön grafiikka muuttaa väriään kertoakseen tästä muutoksesta

Tämän jälkeen pelaaja voi valita muita yksiköitä ja suorittaa niillä samankaltaisia toimintoja. Pelaaja voi myös päättää vuoronsa heti painamalla käyttöliittymän Päätä vuoro -nappia. Peliohjain rekisteröi, että on tietokoneen vuoro, ja käskee käyttöliittymää ottamaan pelaajalta ohjauksen pois. Tämän jälkeen pelinohjain kutsuu jokaisen vihollisyksikön tekoälykomponenttia kerrallaan. Tekoäly kertoo jokaiselle yksikölle, mitä niiden tulee tehdä. Muuten niiden toiminta menee samalla tavalla kuin pelaajan yksiköillä. Kun kaikki yksiköt on käyty läpi, pelinohjain siirtyy takaisin pelaajan vuoroon ja käyttöliittymä palauttaa ohjauksen takaisin pelaajalle.

Tämän jälkeen pelaaja päättää lopettaa ja tallentaa pelin painamalla käyttöliittymän Tallenna ja poistu -nappia. Pelinohjain kutsuu tallennusmetodia, joka käy läpi pelilaudan, siellä sijaitsevat yksiköt ja niiden tilan sekä vuorotilanteen ja kirjoittaa tiedot tiedostoon. Tämän jälkeen ohjelma ilmoittaa, että tallennus onnistui ja ohjelma sulkeutuu.

**Algoritmit**

Pelissä tarvitaan algoritmeja yksiköiden polunhakuun ja tekoälyä varten. Polunhakua varten on tarkoitus käyttää A\*-algoritmia, koska se on hyvin tehokas ja sopii polunhakuun, jossa liikkumisen hinta vaihtelee. Toinen vaihtoehto olisi käyttää Dijkstran algoritmia, mutta se ei olisi yhtä tehokas kuin A\*. Käytännössä tehokkuudella ei kuitenkaan todennäköisesti olisi suurta merkitystä.

Pelin tekoälyä varten aion kirjoittaa oman algoritmin, joka perustuu pistesysteemiin. Algoritmi tutkii, mitä kaikkia toimintoja yksikön on mahdollista suorittaa. Jokaista toimintoa varten on määritelty pistemäärä, joka kuvaa, kuinka paljon tekoäly painottaa kyseisen toiminnon suorittamista. Algoritmi valitsee toiminnon, jolla on eniten pisteitä. Vaihtoehtoinen toteutus voisi olla tilakone, jonka perusteella yksikkö valitsee suoritettavan toiminnon. Pisteisiin perustuva algoritmi tuntuu paremmalta vaihtoehdolta, koska yksiköillä on monia kykyjä ja paljon mahdollisia toimintoja, minkä vuoksi tilakoneella voisi olla hyvin hankala toteuttaa hyvää tekoälyä. Lisäksi pisteisiin perustuvaa tekoälyä on helppo hienosäätää ja laajentaa tarpeen tullen.

**Tietorakenteet**

Ohjelmassa tarvitaan sekä dynaamisia että muuttumattomia tietorakenteita. Pelilaudalla olevien yksiköiden käsittelyyn aion käyttää listoja, koska yksiköt voivat tuhoutua pelin aikana. Listoja aion käyttää myös yksiköihin kohdistuvien vaikutusten käsittelyssä, koska vaikutuksia voi olla samaan aikaan monta tai joskus ei välttämättä yhtään. Vaikutukset aion tallentaa omaan tietorakenteeseensa, joka sisältää vaikutuksen keston, hyökkäys- ja puolustusbonukset, yksikön taintumisen ja verenvuodon. Tällä tavalla vaikutusten käsittely on selkeämpää, kuin esimerkiksi moniulotteisia listoja käytettäessä. Myös pelilaudan koordinaatteja varten aion tehdä oman tietorakenteen, joka sisältää x- ja y-koordinaatin (samalla tavalla kuin RobotWorld-tehtävässä).

Pelikentän ruudut ja ruutujen sisällä oleva tieto naapureista eivät tule muuttumaan pelin aikana, joten niiden käsittelyyn voi käyttää tupleja.

**Kirjastot**

Käytän PyQt5:ä graafisen käyttöliittymän toteuttamiseen.

**Aikataulu**

1. käyttöliittymä (noin 7 tuntia)
2. polunhaku (noin 7 tuntia)
3. polunhaun testaus (noin 2 tuntia)
4. maastot (noin 4 tuntia)
5. yksiköt (noin 10 tuntia)
6. pelin ohjaus (noin 8 tuntia)
7. laskennan testaus (noin 3 tuntia)
8. tekoäly (noin 15 tuntia)
9. tekoälyn testaus (noin 3 tuntia)
10. tapahtumaloki (noin 3 tuntia)
11. tiedostojen lukeminen (noin 3 tuntia)
12. tiedostojen lukemisen testaus (noin 2 tuntia)
13. lisää pelikenttiä (noin 3 tuntia)
14. pelitilanteen tallennus (noin 2 tuntia)
15. kenttäeditori (noin 8 tuntia)
16. yksiköiden valitseminen, sijoittelu (noin 6 tuntia)
17. muuta testausta, viimeistelyä (noin 4 tuntia)

**Yksikkötestaussuunnitelma**

Ohjelmasta kannattaa ainakin testata polunhakua, tekoälyn toimintaa ja pelissä tapahtuvan laskennan oikeaa toimintaa. Polunhaun testauksen voisi toteuttaa siten, että valitsee pelikentältä ruudun lähtöpaikaksi ja jokaisen ruudun kohdalle tulostetaan numero, joka kertoo, kuinka paljon kyseiseen ruutuun liikkumiseen kuluu liikkumispisteitä. Tällä tavalla voidaan varmistaa, että polunhaku toimii oikein ja optimaalisesti. Tekoälyä voisi testata… . Laskennan testaamista voisi koodata luokan, joka kutsuu laskentaan käytettäviä metodeja ja testaa, palauttavatko ne oikeanlaisia arvoja. Muitakin keskeisiä ominaisuuksia olisi ehkä hyvä testata.

**Kirjallisuusviitteet ja linkit**

<https://plus.cs.aalto.fi/y2/2020/toc/>

<https://docs.python.org/3.8/>

<https://www.riverbankcomputing.com/static/Docs/PyQt5/>

<https://doc.qt.io/qt-5/>

<https://www.redblobgames.com/pathfinding/>

<https://www.geeksforgeeks.org/a-search-algorithm/>

<https://www.gamasutra.com/view/feature/129959/designing_ai_algorithms_for_.php>

<https://www.gamasutra.com/blogs/JakobRasmussen/20160427/271188/Are_Behavior_Trees_a_Thing_of_the_Past.php>

<https://stackoverflow.com/questions/3133273/ai-for-a-final-fantasy-tactics-like-game>