1. Henkilötiedot

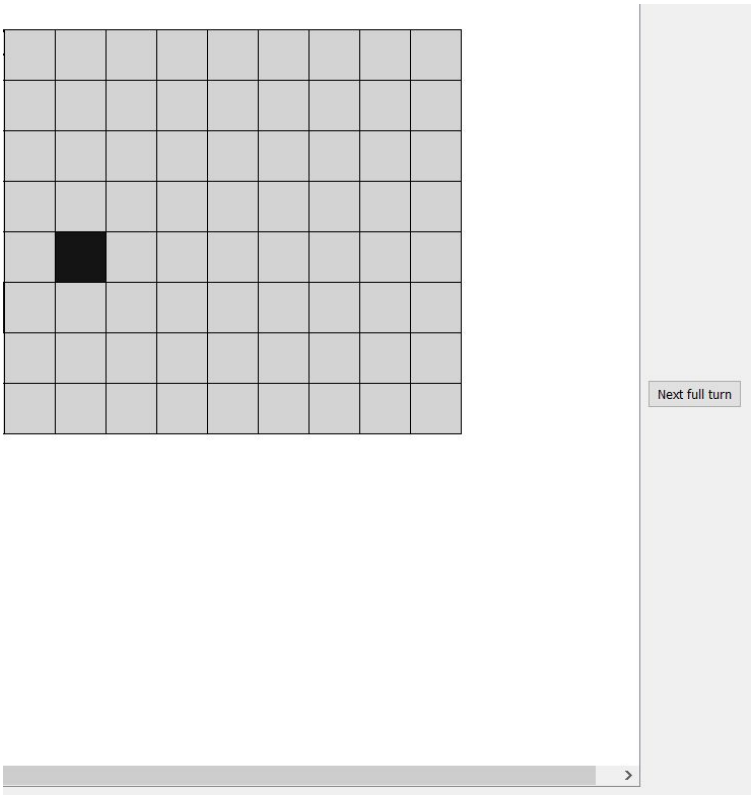
Kim Amir
713805
Bioinformaatioteknologia
2. vuosikurssin opiskelija
22.2.2021

2. Yleiskuvaus ja vaikeustaso

Toteutan vuoropohjaisen strategiapelin. Pelialueena toimii kaksiulotteinen taulukko, johon voi sijoittaa omia taistelijoita pelin alussa. Pelin alkaessa taistelijoita on käytettävä toisen osapuolen tuhoamiseen. Pelialueisiin satunnaisesti generoidaan esteitä, kuten kiviä tai seiniä, jotka on pakko kiertää tai tuhota halutessa. Toteutan lähtökohtaisesti projektin keskivaikealla vaikeustasolla ja siirryn vaativaan jos minulle jää aikaa. Tällä hetkellä ideana olisi luoda 4-5 eri hahmolajia. Pelin kulku on vielä auki, mutta lähtökohtaisesti kukin pelaaja voi vuoronsa aikana liikuttaa yhtä hahmoa ja hyökätä toisella (tai samalla) kerran.

3. Käyttötapauskuvaus ja käyttöliittymän luonnos

Lähtökohtaisesti ohjelma kommunikoi käyttäjän kanssa eri ilmoitusviestien kautta, jotka tulevat esiin riippuen käyttäjän hiiren klikkauksista. Eli aluksi ei ole tarvetta vastaanottaa minkäänlaista tekstiä käyttäjältä (ohjelma pyytää käyttäjää kirjoittamaan nimensä tallennettaessa peliä jos päätän toteuttaa tallennusominaisuuden). Käyttäjä klikkaa vuorollaan haluamaansa hahmoa, jolloin hahmon elämäpisteet ja hyökkäysvaihtoehdot tulevat esiin graafisen liittymän ikkunassa.



(Esimerkki graafisen käyttöliittymän ikkunasta. Musta neliö on este, kuten kivi/seinä, muut neliöt ovat vapaita paikkoja. Ikkunan oikeaan reunaan sijoitan klikattavia nappuloita kuten: end turn, move char, attack, end game, situation report (joka tulostaa pelin tilanteen). Tulostusviestit tulevat kartan alapuolella olevaan tyhjään alueeseen. Tulostusviesteihin kuuluu vihollisen toiminnan raportti (eli mitä vihollinen teki vuorollaan) ja valitseman sotilaan tilanneraportti sekä hyökkäysmahdollisuudet ja niiden ominaisuuksien kuvaus. Lisäksi myös omien hahmojen hyökkäysraportti kuuluu tulostusviesteihin. Pelin alustusikkinunaan kuuluu jokaista hahmoa kohti yksi nappula, jonka avulla kyseisen hahmotyypin saa lisättyä peliin)

Ohjelman käynnistäessä pelikenttä generoidaan ja avautuu ikkuna, jossa näkyy kenttä esteineen. Kumpikin pelaaja lisää saman verran hahmoja (noin 5), hahmojen tyyppi on vapaasti valittavissa. Tämän jälkeen pelin aloittamiseen olisi ‘start game’ näppäin, joka aloittaa pelin. Tässä vaiheessa kaikki hahmot on lisätty sceneen. Pelin kulku perustuu laajalti MousePressEventteihin (jokaisella klikkauksella jokin osa ohjelmaa aktivoituu, pääosin GUI, hahmojen olemassaolosta vastaava luokka ja maailman vapaita ja varattuja tiloja koskeva luokka). Tällä hetkellä on ideana, että peli päivittyy jokaisella toiminnolla, eikä tasaisin aikavälein (toisin kuin robotworld-tehtävä), tämä on toki muutettavissa vielä tässä vaiheessa.

4. Ohjelman rakennesuunnitelma

Alustava rakennesuunnitelma:

main.py:
Peli käynnistetään ajamalla tämä tiedosto. Alustaa pelikentän ja kutsuu muita tarvittavia luokkia (kuten Game).

game.py:
Sisältää luokan Game, joka:
Alustaa pelin luomalla luokan World olion, lisää pelaajat ja pyörittää peliä.
Game-luokka sisältää attribuutteinaan mm. pelimaailman (World-olio) ja pelaajat (Player-olioita). Game-luokasta pääsee käsiksi melkein mihin tahansa muuhun luokkaan.

world.py:
Sisältää luokan World, joka:
Luo kaksiulotteisen kartan ja täyttää sen Square-oliolla. Sisältää listan maailmassa olevista hahmoista sekä metodeja maailman mittasuhteiden saamiseen, esteiden tunnistamiseen ja hahmojen lisäämiseen.
World-luokan tehtävä on dokumentoida maailmassa olevien olioiden sijainteja.

player.py:
Sisältää luokan Player, joka:
Sisältää listan pelaajan hahmoista sekä metodeja vuoron pelaamista varten.
Vastaa pelaajan hahmojen seurannasta.

ai.py:

Sisältää luokan AI, joka:
On samanlainen kuin luokka Pelaaja, mutta sisältää ‘tekoälyn’ päätöksentekoon vaikuttavat metodit.
Vastaa vihollisen vuoron toteuttamisesta ja sen hahmojen seurannasta

gui.py:
Sisältää luokan GUI, joka:
Luo graafisen käyttöliittymäikkunan ja antaa käyttäjän vuorovaikuttaa sen kanssa.
Sisältää ikkunan luomista ja nappuloiden painamisesta vastaavia metodeja.
Vastaa graafisen käyttöliittymän toiminnasta.

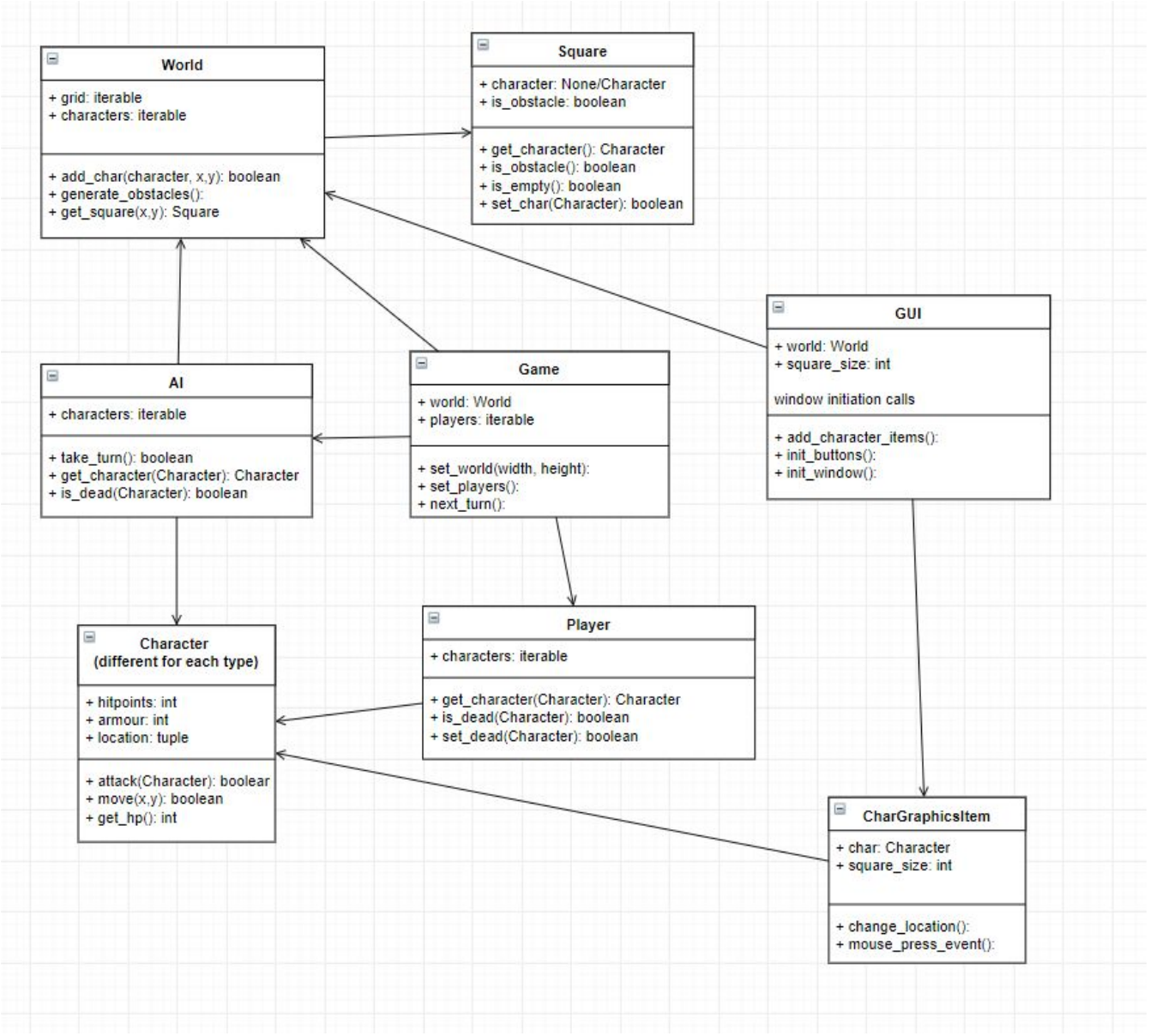
char_graphics_item.py:
Sisältää luokan CharGraphicsItem, joka:
Vastaa jokaisen hahmo-olion graafisesta representaationsa ja sen luonnista ja päivityksestä.
Sisältää myös metodin, jonka avulla käyttäjä voi vuorovaikuttaa kullakin hahmolla klikkaamalla sen graafista itemiä ruudulla. Jokaisella hahmotyypillä on oma muotonsa.
Käyttäjän hahmot ovat vihreitä ja vihollisen hahmot punaisia.
Vastaa hahmojen graafisesta toiminnasta

square.py:
Sisältää luokan Square, joka:
Voi olla seinä, vapaa ruutu, tai varattu ruutu (jolloin siinä on hahmo).
Square-luokalla on metodeja, joilla kyseisen neliön tilaa voi tarkastella tai muuttaa.
Pelimaailma täytetään Square-olioilla

char_type.py (jokaista hahmoa kohti oma tiedosto ja luokka):
(UML-kaaviossa kaikki hahmot yhdistetty yhdeksi Character-luokaksi)

Jokaisella hahmolla on eri ominaisuudet, joten jokaista hahmoa varten on oma luokka. Päätin toteuttaa luokat erillään (luomatta yhteistä yläluokkaa) juuri siksi, että niillä on täysin ainutkertaiset toimintamahdollisuudet. Jokaisella hahmolla on attribuutti self.owner, joka on Player-olio sekä elämäpisteet ja mahdolliset panssaripisteet (esim tankit).
Char-luokat vastaavat hahmojen ominaisuuksista ja hyökkäyksistä.

Muita luokkia saatan tietenkin lisätä projektin edetessä.



5. Tietorakenteet

Maailman luontiin kaksiulotteinen taulukko (helppo iteroida eikä sen koko muutu alustuksen jälkeen), hahmojen listaukseen lista (lyhyt ja helppo, noin 5 hahmoa per pelaaja).

6. Tiedostot ja tiedostoformaatit

Ohjelma käyttää ainoastaa py-tiedostoja, jotka sisältävät sen toiminnan mahdollistavat luokat ja metodit. Näihin kuuluvat mm.

- game.py
- world.py
- main.py

- player.py
- ai.py
- gui.py
- char_graphics_item.py
- square.py
- char_type.py
- test.py

Jos toteutan tallentamisominaisuuden, ohjelman tilanne voidaan tallentaa tekstitiedostoon. Tällöin tiedosto sisältäisi pelaajan nimen, maailman varattujen ruutujen tiedot, kullekin pelaajalle kuuluvien hahmojen sijainnit ja elämäpisteet. Tieto olisi esitetty samalla tavalla kuin ChunkIO-tehtävässä eli yhtenä merkkijonona, joka on helppo lukea tietokoneella.

7. Algoritmit

Tekoälyn hahmojen liikkumiseen käytän joko hyvin yksinkertaista algoritmia (‘liiku lähemmäs lähintä vastustajaa’) tai jotakin nopeimman reitin löytävää verkkoalgoritmia kuten DFS tai Djikstra. Jos tekoälyn hahmo on kuolemaisillaan, se perääntyy ja yrittää parantaa itsensä jos mahdollista.

Tekoälyn hahmojen hyökkäyskäyttäytymisen haluan noudattavan periaatetta ‘jos pystyn tuhoamaan vastustajan hahmon tällä vuorolla, teen sen’ eli tekoäly hyökkää heikoimman hahmon kimppuun. Jos vastustajan hahmo on panssaroitu, tekoäly pyrkii hyökkäämään sitä panssarintuhoamisaseilla.

Muita mahdollisia ratkaisuja (joita en todennäköisesti toteuta):
- Koneoppimismetodien käyttö tekoälyn päätöksenteon parantamiseen
- Eri persoonallisuuksien luonti (esim. aggressiivinen ‘all in’ tyyppinen AI vs. puolustautuva)
- AI, jonka liikkuminen perustuu osittain satunnaisuuteen

8. Testaussuunnitelma

Testejä kannattaa laatia alusta lähtien, jotta voi varmistua siitä, että jokainen vaihe on toteutettu oikein alusta asti. Tällaisiin testeihin kuuluvat esimerkiksi:
- Onko maailma luotu oikein? (testataan tietyillä siemenluvuilla)
- Onko pelaajat lisätty peliin?
- Löytyykö tietty hahmo tietyltä ruudulta?
- Onnistuuko hahmon hyökkäys tai kuolema?
- Kenen vuoro on?
- Löytyykö hahmo oikealta ruudulta liikkumisen jälkeen?
- Virhetilanteet (ohjelma ei salli hahmojen laittomia liikkeitä)

Koska käyttöliittymän pitää aina päivittää taustalla tapahtuva tilanne graafiseksi esitykseksi, molempia voidaan testata erikseen.

Kursoria edeltäviin tapauksiin voi kuulua esimerkiksi tietyn hahmon asettaminen pelilaudalle kursoria käyttäen ja sitten kyseisen ruudun tilan testaaminen.

9. Kirjastot ja muut työkalut

Tällä hetkellä kaksi ohjelman toiminnan kannalta ehdotonta kirjastoa ovat PyQt5 sekä sys. Sys-kirjaston metodilla exit() ohjelma lakkaa toimimasta virheen sattuessa. PyQt5 sisältää kaikki vaadittavat kirjastot ja metodit graafisen käyttöliittymän luontiin.

10. Aikataulu

Aloitan kirjoittamalla main.py, game.py, world.py ja square.py koodit. Arvioin tähän kuluvan noin 5-10h. Tässä vaiheessa voin myös kirjoittaa tests.py-tiedoston, jolla voin testata maailman generoinnin onnistumista. Tässä vaiheessa en oleta tiedostojen olevan täydellisiä. Ne toteuttavat sen mitä tähän asti pitää pystyä toteuttamaan.

Seuraavaksi haluan saada pelikentän näkyviin, joten kirjoitan gui.py-tiedoston **(ilman mitään nappuloita, joita voi kursorilla klikata)** , johon menee ehkä 5h+. Tässä vaiheessa ajamalla main.py-tiedoston, käyttäjän pitäisi nähdä graafinen näkymä pelikentästä esteineen.

Seuraavaksi kirjoitan player.py ja char_type.py-tiedostot. Tähän kuluu noin 5-10h.
Nyt alkaa projektin vaikein osa: saada pelaaja asettamaan haluamansa hahmot järkevästi pelikentälle. Tämä vaatii char_graphics_item.py:n kirjoittamista ja gui.py:n modifiointia eli metodien lisäämistä. Arvioin tähän kuluvan 20-30h. Tässä vaiheessa pelaajan on kyettävä asettamaan haluamansa hahmot pelikentälle. Testitiedostolla voidaan myös tässä vaiheessa testata hahmojen oikea asetus ja sijainti pelikentällä.

Kirjoitan tiedoston ai.py, joka sisältää metodin vastustajan hahmojen asettamiseen. Hahmot tulevat näkyviin vasta kun käyttäjä painaa ‘start game’, jotta käyttäjä ei voi taktikoida tietämällä vastustajan hahmojen sijainnit. Tämä metodi perustuu laajalti satunnaisuuteen, kuitenkin siten, että vastustajan kokoonpano on jokseenkin järkevä. Tähän kuluu noin 3h+.

Lisää koodia gui.py-tiedostoon, jotta painamalla ‘start game’ peli alkaa (5h?).
Tekoälyn käyttäytymisen ohjelmointi (10-20h).
Pelin loppumisen käsittely 2h+

Jokaisessa vaiheessa kehitän samalla test.py-tiedostoa ja testaan sen toimintaa sisäisesti.

11. Kirjallisuusviitteen ja linkit

CS-A1121 Ohjelmoinnin peruskurssi Y2
CS-A1141 Tietorakenteet ja algoritmit Y ja kurssikirja:
https://tarjotin.cs.aalto.fi/cs-a1141/OpenDSA/Books/CS-A1141/html/index.html
Python-dokumentointi: https://docs.python.org/3/
PyQt5 dokumentaatiot:
https://www.riverbankcomputing.com/static/Docs/PyQt5/
https://doc.qt.io/qt-5/

12. Liitteet

Ei liitteitä