Y2 projektityön dokumentti

Akseli Konttas 587031, AIT, 8.5.2017

Yleiskuvaus

Projektityöni aiheena oli vuoropohjainen strategiapeli. Projektikuvauksessa kehotettiin miettimään toteutusta siten, että hahmoilla on erilaisia hyökkäyksiä, käytössä on erilaisia tavaroita, ohjelma on konfiguroitavissa ulkoisten tiedostojen avulla, ja niin edelleen. Tämä oli pyrkimykseni peliä rakentaessa.

Alkuperäiseen suunnitelmaan nähden lopullisessa työssäni on hieman vähemmän toimintoja, esimerkiksi tavaroita en loppujen lopuksi toteuttanut ollenkaan, vaikka se olisikin ollut suhteellisen helppoa rakentaa olemassaolevan koodin päälle – ne eivät esimerkiksi enää tuntuneet sopivan peliin.

Käyttöohje

Peli käynnistetään ajamalla main-tiedosto. Tämä luo uuden graafisen käyttöliittymän ikkunan. Peli on palautushetkellä alustettu siten, että se lataa heti käyttöliittymän alustamisen jälkeen pelin mukana olevasta tiedostosta save.txt (tämä ei kuitenkaan ole välttämätöntä, vaan ohjelman voisi laittaa avaamaan oletuksena jonkun toisen tiedoston tai olla avaamatta mitään tiedostoa).

Kun peli on ladattu, näytöllä näkyy pelikartta, jossa eriväriset laatat kuvaavat eri maastotyyppejä. Lisäksi kartalla näkyvät pelihahmot: sinisävyiset ovat pelaajan ohjattavissa, punasävyiset taas tietokoneen. Lisäksi kartan viereen aukeaa toinen ikkuna (otsikolla Info), joka näyttää kaikkien pelihahmojen perustiedot: luokan, ruudun, jossa hahmo sijaitsee sekä senhetkiset elämäpisteet. Kilkkaamalla tässä ikkunassa olevia hahmojen kuvia aukeaa uusi ikkuna, joka kertoo tarkemmin hahmon tiedoista.

Pelaajan tarkoituksena on tappaa tietokoneen ohjaamat hahmot. Tämä tapahtuu liikuttamalla omia hahmoja ja hyökkäämällä niillä vastustajiin. Kun pelaaja klikkaa omaa hahmoaan, sen lähellä olevat ruudut värjäytyvät violetin sävyisiksi. Nämä ovat ne ruudut, joihin hahmo voi sillä vuorolla liikkua. Klikkaamalla jotain näistä ruuduista hahmo siirtyy kyseiseen ruutuun. Mikäli pelaaja klikkaa jotain muuta ruutua, hahmo ei liiku ja kaikki ruudut palautuvat normaalin värisiksi, jolloin pelaaja voi valita jonkun toisen hahmon.

Mikäli pelaaja päättää siirtää hahmoaan, aukeaa uusi ikkuna. Ikkunassa on painikkeet joka asialle, jota hahmo voi tehdä: jokaiselle hyökkäykselle omansa, sekä jokaiselle aktivoitavalle kyvylle oma. Lisäksi siellä on painikkeet pass, jonka valitsemalla hahmo päättää vuoronsa tekemällä muuta, sekä cancel, jolla pelaaja voi peruuttaa valintansa. Jos pelaaja valitsee cancel tai painaa ikkunan kiinni, hahmo jää siihen ruutuun mihin se siirrettiin, mutta se ei ole vielä lopettanut vuoroaan vaan se voi vielä liikkua (tosin vain samoihin ruutuihin kuin aiemminkin, ei edemmäs) sekä hyökätä.

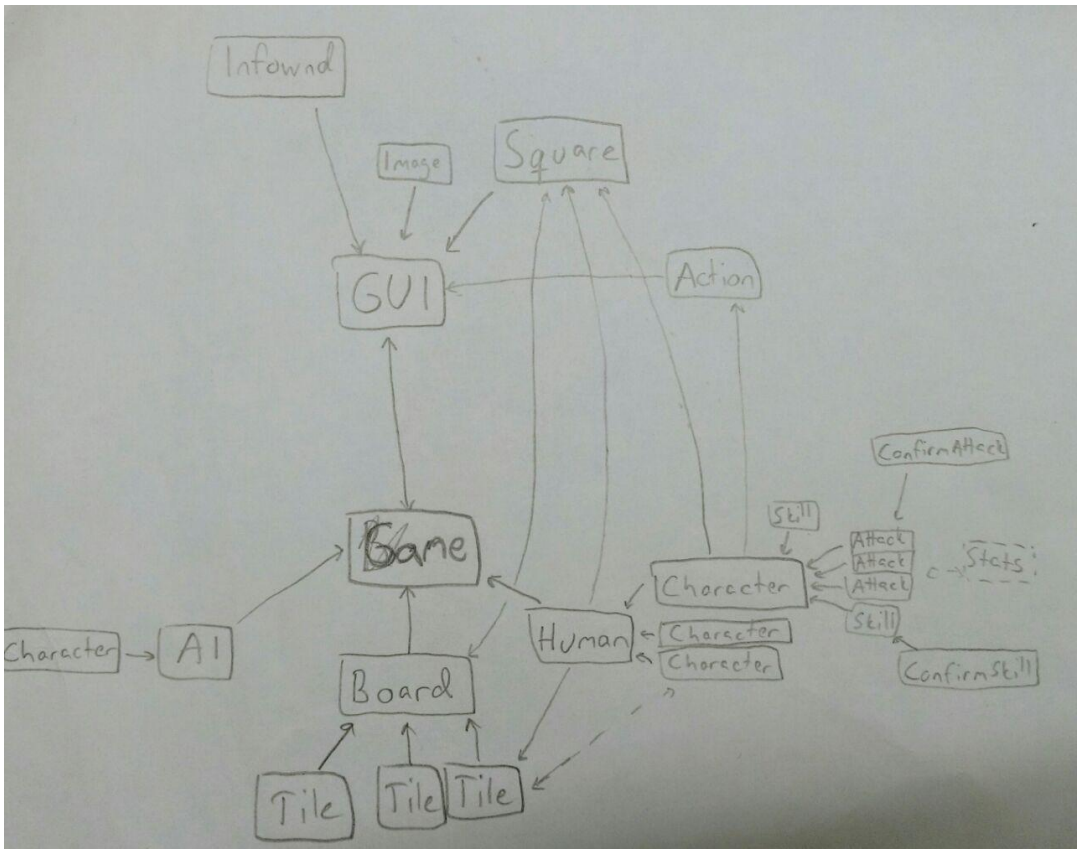
Jos pelaaja valitsee hyökkäyksen, peli laskee, onko vihollisia lähellä, jotta hyökkäyksen voisi toteuttaa. Eri hyökkäyksillä on eri etäisyyksiä, kuinka kauas niillä voi hyökätä: esimerkiksi miekalla voi iskeä vain viereiseen ruutuun, mutta jousella voi ampua kahden ruudun päähän. Mikäli lähellä ei ole vihollisia, alanurkan ohjeteksti kertoo tämän. Hahmolle käy samoin kuin jos sen liikkuminen olisi peruutettu, eli sitä voi vielä liikuttaa. Jos taas lähellä on vihollisia, niiden ruudut värjäytyvät punaiseksi merkkinä siitä, että niihin voi hyökätä. Napsauttamalla jotain punaisista ruuduista aukeaa vielä varmistusdialogi, joka kysyy käyttäjältä varmistuksen hyökkäyksestä ja kertoo samalla perustiedot: osumatarkkuuden prosentteina sekä vahingon määrän, jos hyökkäys osuu. Jos pelaaja hyväksyy hyökkäysen, hahmo hyökkää vastustajaan, vastustaja ottaa vahinkoa (jos ottaa) ja hahmo on käyttänyt vuoronsa ja muuttuu sen merkiksi harmaasävyiseksi. Sitä ei voi enää liikuttaa ennen pelaajan seuraavaa vuoroa. Hyökkäyksen lisäksi pelaaja voi käyttää jotain kykyä; niiden käyttö tapahtuu samoin kuin hyökkäyksenkin, mutta kyvystä riippuen kohteena voi olla myös pelaajan oma hahmo.

Liikutettuaan kaikkia hahmojaan tulee tietokoneen vuoro, joka liikuttaa hahmojaan ja hyökkää niillä samoin säännöin kuin pelaajakin. Tämän jälkeen tulee taas pelaajan vuoro.

Pelin häviää se, kumman kaikki hahmot ovat ensin kuolleita.

Ohjelman rakenne

Alla on hieman nopeasti piirretty kuva ohjelman periaatteellisesta luokkajaosta, jonka avulla pyrin esittelemään pelin luokkajaon.



Pelin perusluokka (ennen graafisen käyttöliittymän toteuttamista) on Game. Sen vastuulla ei varsinaisesti ole mitään muuta kuin pitää kirjaa kaikista muista pelin luokista ja olioista. Game sisältää yhden Board-olion sekä yhden Human- ja yhden AI –olion. Näistä Human ja AI ovat Player-luokan alaluokkia, joille on määritelty hieman omia toiminnallisuuksia.

Board-luokan peruselementti on kaksiulotteinen lista, jonka avulla se pitää kirjaa pelikartan tilanteesta. Luokan vastuulla on kaikki, mikä jollain tapaa liittyy kartan muuttamiseen tai analysointiin. Se hoitaa esimerkiksi pelihahmojen siirtämisen sekä sen määrittämisen, minne nämä hahmot edes pystyvät siirtymään kartalla esteiden ja muiden hahmojen vuoksi. Olio osaa myös kertoa esimerkiksi jonkun hahmon sijainnin kartalla.

Alunperin ajatuksena oli, että pelihahmoja liikutettaisiin suoraan board-olion listassa, mutta tämä osoittautui nopeasti vaikeaksi, sillä ajatuksenani oli toteuttaa myös erilaisia maastoja peliin. Tällöin pitäisi olla jokin toinen tietorakenne tämän määrittämiseen. Päädyin uuteen luokkaan Tile, jonka ilmiintymät asetetaan Board-olion kartan kenttiin. Näillä Tile-olioilla on eri ominaisuuksia, kuten se, ketkä pystyvät liikkumaan hahmon läpi tai kuinka paljon askeleita sen läpi kulkeminen vaatii. Lisäksi jokaisella Tilella voi olla yksi Object, joka siis kuvaa laatalla seisovaa hahmoa. Kun peli siirtää hahmoa toiseen ruutuun, Board-olio hakee tämän hahmon Tile-oliolta, asettaa Tilen objectiksi None sekä siirtää hahmon toiseen Tileen. Tärkein Tilen oma toiminnallisuus on sen määrittäminen, kuinka monta askelta sen läpi kulkeminen vaatii. Tästä lisää myöhemmin.

Yhdellä pelaajalla, siis Human- tai AI-oliolla, on nollasta ylöspäin pelihahmoja, Character-luokan ilmiintymiä (tosin jos hahmoja on nolla, on pelaaja hävinnyt). Character-luokka sisältää tiedon pelihahmosta, kuten sen

hyökkäyksistä ja kyvyistä sekä statseista. Erityyppiset pelissä olevat hahmot on toteutettu perinnällä; esimerkiksi Knight ja Archer kuvaavat erityyppisiä hahmoja, mutta ovat molemmat Character-luokan aliluokkia.

Jokaisella hahmolla on yhdestä ylöspäin hyökkäyksiä sekä nollasta ylöspäin kykyjä. (Näiden pelitekninen ero on se, että hyökkäyksen tarkoitus on aina aiheuttaa vahinkoa viholliseen, kun taas kyvyillä voi olla eri tarkoituksia.) Hyökkäyksiä kuvataan Attack-luokalla. Alunperin tätä luokkaa ei ollut tarkoitus tehdä, vaan sen toiminnallisuudet oli tarkoitus toteuttaa muissa luokissa. Kävi kuitenkin ilmi, että koska jokaisella hyökkäyksellä on niin monta eri ominaisuutta (nimi, kuvaus, osumatarkkuus, teho, kantama, jne), oli vaikeaa pitää kirjaa niistä listana tai monikkona; lisäksi myöhemmin, kun otin mukaan erityyppiset hyökkäykset, listoilla leikkiminen kävi hyvin hankalaksi. Tätä varten tein Attack-luokan. Sen vastuulla on paitsi pitää kirjaa kunkin hyökkäyksen ominaisuuksista, myös tarvittaessa laskea tehty vahinko, kun hyökkäystä käytetään johonkin hahmoon. Attack-luokalla on kaksi alaluokkaa, Melee ja Magic, jotka eroavat toisistaan sillä, että Melee-tyyppisellä hyökkäyksellä hyökättäessä ratkaisevaa vahingon määrittämisessä on hyökkääjän Attack ja puolustajan Defense –kentät, kun Magic-tyyppisellä hyökkäyksellä vastaavat kentät ovat Magic ja Resistance. Vastaavalla perinnällä olisi erittäin helppo luoda uudenlaisia hyökkäyksiä, kutenhyökkääjän Magicilla puolustajan Defenseen tai vaikka hyökkääjän Defensellä puolustajan Resistanceen, mutta palautushetkellä pelissä on vain kaksi edellämainittua.

Uusien hyökkäysten luominen Attack-luokan avulla on hyvin helppoa: pitää vain luoda uusi olio, jolle syötetään parametrina vaaditut tiedot, ja lisätään hyökkäys hahmon listaan, joka pitää niistä kirjaa. Tässä onnistuin mielestäni hyvin: kun luo uuden hahmon, sille luo uuden hyökkäyksen minuutissa.

Skill-luokan oli taas alunperin tarkoitus olla pelkästään enumeraatiota varten, koska niiden avulla muidenkin luokkien olisi ollut helppo pitää kirjaa hahmon kyvyistä ja miten niiden pitää suhtautua, mutta jossain vaiheessa tuli tarve toteuttaa omaa toiminnallisuutta luokan sisällä. Tästä on esimerkkinä kyky Heal (joka onkin ainoa tällainen kyky kirjoitushetkellä), joka parantaa muita hahmoja (tai itseään); tämäkin oli itseasiassa alunperin tarkoitus toteuttaa Attack-luokan avulla, mistä saattaa vielä näkyä viitteitä koodissa ja kommenteissa, mutta päädyin lopulta Skill-luokan käyttöön. Haasteena sen toteuttamisessa oli se, että toisin kuin hyökkäysten kanssa, joka kyvylle tarvitsi omanlaisensa toiminnallisuudet. Siksi niitä ei pelistä juuri löydykään, vaikka nyt koodi onkin sen verran hyvin jo rakennettu, ettei uusien lisääminen olisi tajuton vaiva.

Seuraava luokka on Stats. Se on puhtaasti enmurointia varten, ja sen avulla esitetään hahmojen statit, kuten hyökkäys. Luokka ei olisi missään nimessä pakollinen, mutta sen käyttö vähensi virheitä, kun esimerkiksi aina hahmon Attack-kenttään viitattaessa voitiin ilmaista Stats.ATTACK muun vastaavan sijasta.

Viimeinen ei-graafisen käyttöliittymän luokka on IO. Sen tehtävänä oli luoda uusi peli tekstitiedoston pohjalta ja tallentaa peli vastaavaan tiedostoon. Alunperin Game-olio loi uuden IO-olion, joka lisäsi Gamelle Board-olion sekä siihen hahmot. Lopullisessa toteutuksessa kuitenkin ensin luodaan IO, joka luo pelin kokonaisuudessaan – tämä helpootti graafisessa käyttöliittymässä pelin lataamisen kanssa.

Tähän asti ohjelman hierarkia oli mielestäni aika hyvin pysynyt kasassa, vaikka vähän olisi toki voinut olla vielä selkeämmin rajatut vastuut. Silti olen aika tyytyväinen jakoon, mutta graafisen käyttöliittymän lisääminen sekoitti aika pitkälti koko rakenteen.

Graafisen käyttöliittymän pääluokka on GUI. Alunperin hierarkiassa Game oli GUIn yläpuolella, mutta lopullisessa toteutuksessa tämä on toisin päin: ensin luodaan GUI, minkä jälkeen sille vasta luodaan peli IO:n avulla. Luokan vastuulla on päivittää karttaa: tätä varten se luo jokaista kartassa olevaa Tile-oliota kohden yhden Square-olion, joka on itseluotu QgraphicsRectItem-luokan alaluokka. Tämän jälkeen GUI värittää laatat oikean värisiksi – väritiedot on tallennettu Tile-olioihin, jolloin niiden muokkaaminen on helppoa. GUIn vastuulla on myös päivittää karttaa poistamalla vanhat kuvakkeet kartasta ja laittamalla uudet tilalle, kun hahmoa siirretään. Luokka myös värjää laatat, kun esimerkiksi hahmoa siirretään. Lisäksi GUIn vastuulla on yhteistyössä IO:n kanssa luoda uusi peli. Uuden pelin voi myös ladata, vaikka vanha olisi vielä käynnissä, ylävalikon Load game-toiminnolla.

Kartalla näkyvät kuvakkeet ovat Image-luokan ilmeentymiä. Image taas on Qpixmapin alaluokka. Sillä ei ole mitään omaa toiminnallisuutta, mutta harjoitellessani PyQtn käyttöä helpotti omat luokat. Image hakee kuvakkeensa lähdekoodin mukana tulevasta images-hakemistosta; tiedostopolun se saa hahmolta, jota se esittää.

Square on luokka, jonka kohdalla tuli ongelmia hierarkian kanssa. Periaatteessa luokan oli tarkoitus ainoastaan esittää yhtä neliötä kartassa, mutta nopeasti tuli ongelmaksi käyttäjän klikkauksiin reagoiminen. Mietin myös muita vaihtoehtoja kuin klikkausten kautta tapahtuvaa kommunikointia, mutta loppujen lopuksi päädyin lopulliseen, pelissä olevaan ratkaisuun. Käytännössä luokan sisällä tapahtuu kaikki hahmojen liikuttamiseen liittyvä toiminta (itse liikuttamista lukuunottamatta, mikä tapahtuu Board-luokassa), eli liikkuvan hahmon määrittäminen, hyökkäyksen määrittäminen, ja niin edelleen. Tästä johtuen luokan koodi on melko vaikealukuista, ja uusien toiminnallisuuksien lisääminen on vaikeaa. Lisäksi, kuten myöhemmin totesin, klikkaus ei edes toimi virheettömästi dialogien ilmestymisen jälkeen – niille en keksinyt kuitenkaan parempaakaan korvaajaa, joten nyt täytyy tyytyä siihen, että kartta tuntuu usein reagoivan klikkaukseen eri ruudussa kuin mitä oli tarkoittanut.

Action on luokka, jonka ilmeentymänä toimiva popupikkuna kysyy käyttäjältä liikuttamisen jälkeen, mitä hahmo aikoo tehdä. Se saa parametriksi liikutettavan hahmon, ja luo sen perusteella tarvittavat painonapit. Ongelmaksi tuli, etten keksinyt, miten voisin palauttaa pääohjelmalle valinnan, mutta keksin kiertotavan, jossa square-oliolle välitetäänkin tieto napissa olleesta tekstistä, jonka avulla Square päättelee, mikä oli haluttu toiminto.

Viimeiset toiminnalliset luokat ovat ConfirmAttack ja ConfirmSkill. Ne kysyvät käyttäjältä varmistuksen toiminnalle, kun hän aikoo hyökätä tai käyttää kykyä. Käytännössä ne toimivat samoin, ero on se, että ensimmäinen saa parametrikseen hyökkäyksen, jonka perusteella määrittää ikkunassa näkyvän tekstin, jälkimmäinen taas saa parametrina tekstin, joka näkyy suoraan ikkunassa. Nämä olisi toki voinut yhdistää yhdeksi luokaksi ja käyttää käytännössä vain ConfirmSkill-luokkaa, mutta koska loin ConfirmAttackin ensin, tuntui liian vaivalloiselta enää poistaa sitä ja muuttaa kaikki jo olemassa oleva koodi sitä varten.

Näiden lisäksi pelissä on kolme error-luokkaa: CorruptedMapDataException, CorruptedSaveFileException sekä IllegalMoveException. Jälkikäteen ajateltuna näitä kaikkia ei olis edes tarvittu: alkuperäisenä ajatuksena oli, että kartta ja hahmojen tiedot tulevat erillisissä tiedostoissa, mutta nyt koska ne ovat samassa tekstitiedostossa, toinen ensimmäisestä kahdesta on vääjäämättä turha. Molempia kuitenkin käytetään IO-luokassa, kun havaitaan virheitä. IllegalMoveException taas heitetään kun pelaaja yrittää siirtää hahmoa ”laittomaan” ruutuun. GUI reagoi siihen muuttamalla vihjetekstin asiaan sopivaksi.

Alunperin ajattelin käyttäväni poikkeuksia enemmän ja monipuolisemmin, mistä seurauksena monet erityyppiset tyypit ovatkin. Lopullisessa toteutuksessa ne jäivät vähälle käytölle.

Algoritmit

Pelissä muutamia tärkeimpiä algoritmejä ovat se, miten peli määrittelee lailliset ruudut, joihin hahmo saa siirtyä, sekä tekoälyn toiminta.

Board-luokan legal_squares määrittelee ruudut, joihin sille parametrina annettu hahmo saa sillä vuorolla siirtyä. Tehokkaampiakin tapoja olisi voinut olla, mutta tämä tapa oli yksinkertainen kirjoittaa (ja pienillä askelmäärillä tietokoneen vaatima laskentateho ei ole kovin suuri).

Periaate on määrittää hahmon käytössä olevat askeleet ja lähteä hahmon senhetkisestä ruudusta yhden askeleen johonkin suuntaan, vaikka vasemmalle. Tämä ruutu lisätään listaan. Tästä ruudusta siirrytään vasemmalle niin kauan, kuin askeleet riittävät tai kunnes törmätään seinään. Tällöin vaihdetaan suuntaa tai palataan askel taaksepäin ja kokeillaan toista suuntaa. Kun kaikki ruudut on näin käyty läpi, palataan alkuperäiseen ruutuun ja lähdetään ylöspäin, ja niin edelleen.

Vaikeuksia tuli silti sen kanssa, että for-looppeja tarvitaan suuntia varten niin monta, kuin on askeleita. Tämän takia käytin rekursiota: legal_squares kutsuu ensin funktiota pass_by_square, joka kutsuu itseään uudestaan niin kauan kunnes askeleet loppuvat kesken. Näin tulee kaikki mahdolliset ruudut käytyä läpi.

Omassa pelissäni tosin haastaita aiheutti sekin, että eri maastot vievät eri määrän askeleita – ja vielä mahdollisesti eri määrän eri hahmoille. Tähän otin mukaan Tile-luokan, jota kutsutaan joka ruudussa uudestaan. Tile määrittelee pass_by-funktionsa avulla, kuinka monta askelta hahmolla menee ylittää laatta, ja palauttaa jäljellä olevan askelmäärän sekä sen, voiko hahmo pysähtyä kyseiseen ruutuun. Esimerkiksi jos ruudussa on oma pelaaja, ei siihen voi pysähtyä, mutta sen läpi voi silti mennä; jos ruudussa on vihollispelaaja, ei ruudun läpikään voi mennä, jolloin pass_by ilmoittaa jäljellä olevan 0 askelta.

Alla pseudokoodiesitys:

```
def legal_squares(char):
    range = hahmon askelten lkm
    squares = lista ruuduista, joihin voidaan siirtyä (aluksi tyhjä)
    for suunta in suunnat:
        new_coordinates = yksi ruutu hahmon ruudusta suuntaan suunta
        pass_by_square(char,range,new_coordinates,squares)

def pass_by_square(char,steps,coordinates,squares):
    add, steps = pass_by(char,steps)
   if add:
```

```
        lisätään senhetkinen suutu listaan squares
    if steps > 0:
        for suunta in suunnat:
            new_coordinates = yksi ruutu hahmon ruudusta suuntaan suunta
            pass_by_square(char,steps,new_coordinates,squares)
```

```
def pass_by(char,steps):
    jos hahmo ei voi edes olla kyseisessä ruudussa (esim. seinä):
            return False, 0
    jos ruutu on tyhjä ja hahmo voi päätyä kyseiseen ruutuun:
            return True, define_steps_left(char,steps)
    jos ruudussa on ystävä:
           return False, define_steps_left(char,steps)
    jos ruudussa on vihollinen ja hahmo pystyy kulkemaan sen läpi:
           return False, define_steps_left(char,steps)
    return(False,0)

def define_steps_left(char,steps):
     steps_left = määrittelee kuinka monta askelta hahmolta menee läpi menemiseen
     return steps_left
```

Toinen tehtävässä käytetty algoritmi oli tekoälyn toiminta. Alkuperäinen ajatukseni oli, että tietokone kävisi kaikki kombinaatiot hahmojensa liikkeistä läpi va valitsisi niistä parhaan, mutta nopeasti ymmärsin, ettei se olisi kovin mielekästä – tietokoneelta kuluisi tuhottomasti tehoja, jos hahmoja olisi yhtään enempää, ja

mitä todennäköisimmin olisin epäonnistunut ohjelmoinnissa. Seuraava ajatus oli, että tekoäly ensin katsoisi esimerkiksi jos lähistöllä on pelaajan hahmoja, ja sen perusteella käskisi ensin kaikki hahmot yhden hahmon kimppuun ja sitten antaisi loppujen toimia yksin, mutta tämäkin meni vaikeaksi. Lopullisessa toteutuksessa jokainen tekoälyn ohjaama hahmo päättää siis itsenäisesti, mitä tekee.

Hahmo määrittää ensin kaikki ruudut, joihin se ylipäänsä voi mennä. Jokaista näitä ruutua kohden hahmo päättelee, mitä se voi ruudussa tehdä (hyökätä, jne). Jos hahmo voi ruudussa hyökätä, se laskee, kuinka paljon vahinkoa se todennäköisesti tekisi viholliseen. Lisäksi hahmo katsoo, mihin vastustajan hahmot voivat hyökätä ensi vuorolla, ja pyrkii välttämään näitä ruutuja.

Näistä kaikista tekijöistä hahmo määrittelee jokaiselle toiminnolle, siis johonkin ruutuun liikkumiselle ja siinä jotain tekemiselle, luvun, joka kuvaa, kuinka kannattava tämä siirto olisi. Lukua kasvattaa se, että se pystyy vahingoittamaan vihollista (ja kasvattaa hyvin paljon se, jos hahmo pystyy iskullaan tappamaan vihollisen), tai se, että hahmo esimerkiksi parantaa ystävän elämät täyteen, ja laskee se, jos hahmo arvelee vihollisen pystyvän tappamaan hänet seuraavalla vuorolla. Nämä kaikki kombinaatiot hahmo kerää listaan, joista se lopuksi arpoo yhden siten, että jokaista vaihtoehtoa painotetaan sen kannattavuusluvulla. Tämän toiminnon hahmo sitten tekee.

Tämä toteutus luonnollisesti tarkoittaa sitä, ettei hahmo aina tee ”fiksuinta” siirtoa. Se johtuu paitsi siitä, että algoritmi, jolla kannattavuus lasketaan, on hyvin vaillinainen, mutta myös siitä, että halusin poistaa hieman tekoälyn ennalta-arvattavuutta. Ajatuksena oli se, että jos hahmo huomaa vihollisen hahmon vähillä elämillä lähistöllä se käydään tappamassa, mutta jos itsellä on elämät vähissä, hahmo lähtee karkuun ja toivoo, että joku tulee parantamaan sitä. Kaikkein paras hahmon mielestä on se, jos se on paikassa josta se voi hyökätä viholliseen mutta vihollinen ei siihen (esim. jousiampuja) – tällaisia tilanteita sattuikin muutaman kerran kun testasin peliä.

Toisaalta tekoäly on hyvin vaillinainen: minulla ei millään riittänyt aika tehdä tekoälystä niin hyvää kuin halusin, vaikka koko ajan sitä kehittelinkin kun tuli uusia ideoita. Lisäksi tietokoneellani meni hahmosta riippuen jopa useampi sekunti päätöksen tekemiseen – ennätys lähenteli kymmentä sekuntia, kun hahmolla oli vähän enemmän askeleita. Algoritmit eivät siis olleet kovin tehokkaita, mutta ne täyttivät tehtävänsä.

Loppujen lopuksi on melko selvää, ettei tekoälyni ihmiselle pärjää, mutta on ilahduttava huomata, miten usein se tekee jopa varsin fiksuja päätöksiä, ja voi olla oikeastikin haaste, jos vihollisia on enemmän – tai sitten ei, riippuu tuurista.

Tietorakenteet

Tärkein tietorakenne pelissä on pelikenttää kuvaava kaksiulotteinen lista. Muutenkin käytin listoja paljon toteutuksessa: pelaajien hahmot ovat listassa, hahmon hyökkäykset ovat listassa, ja niin edelleen. Näissä lista oli tosin melko tarkoituksenmukainen.

Monikkoja käytin monessa paikassa funktioiden paluuarvoina ja parametreina, mutta näissä olisi toki yhtä hyvin voinut käyttää listoja.

Muuten pelissä on melko vähän tietorakenteita, yhtään en tainnut itse ohjelmoida.

Tiedostot

Peli käyttää tekstitiedostoja pelin tallentamiseen ja lataamiseen. Pyrin tekemään formaatista sellaisen, että karttoja ja tallennuksia olisi helppo tehdä käsin, ja onnistuin mielestäni suhteellisen hyvin. Alla on esimerkki tallennusformaatista.

#Info

p plain

w water

x wall

f forest

m mountain

#Map

pwpppp

pwwppp

ppwmwp

pxffpp

pxfffp

ppffff

#Char

O Player

C Mage

S (0,3)

#Char

O Computer

C Assassin

S (3,3)

Tiedosto alkaa #Info –segmentillä. Tässä listataan merkkien selitykset karttaan, joka on tiedostossa alempana. Merkki ja selite erotetaan toisistaan välilyönnillä. Merkit voivat olla mitä vain ascii- merkkejä tai kirjaimia (ei kuitenkaan välilyöntejä tai rivinvaihtoja).

Seuraava segmentti, #Map, kuvaa pelikartan. Jokainen merkki neliössä kuvaa yhtä laattaa kartassa. Kartan täytyy olla suorakulmion muotoinen, eli joka rivillä tulee olla yhtä monta merkkiä. Jos jotain merkkiä ei ole listattu aiemmin info-kohdassa, ei karttaa voida ladata; jos jotain nimeä ei tunnisteta (esim. jos laatan nimi on kirjoitettu väärin), laitetaan sen sijaan Plain, joka on pelin peruslaatta.

Loput segmentit ovat #Char-segmenttejä. Jokainen niistä kuvaa yhtä hahmoa, ja sisältää kolme tietoa: Ohjaaja eli pelaaja tai tietokone (merkitään O Player tai O Computer), luokka eli hahmon tyyppi (C luokka) sekä hahmon aloituskoordinaatit (S (x,y), missä x ja y ovat koordinaatit). Jos luokkaa ei tunnisteta, luodaan sen sijaan TestChar-tyyppinen hahmo. (Testchar oli testauksessa käyttämäni hahmo, jota ei ehkä kannata oikeasti käyttää peleissä :P) Jos taas koordinaatit ovat kartan ulkopuolella tai kaksi hahmoa ovat samassa ruudussa, ei karttaa ladata.

Formaatti on suhteellisen tarkka semantiikasta; isoilla ja pienillä kirjaimilla ei ole merkitystä, ja tyhjillä riveillä ei ole merkitystä, mutta muuten tiedoston tulee olla tarkalleTen edellä ilmaistun näköinen. Esimerkiksi järjestyksen tulee olla info – map – char, eikä ylimääräisiä välilyöntejä vaikkapa koordinaattien välissä hyväksytä. Näistä puutteista huolimatta tallennusten kirjoittaminen ja lataaminen tiedostosta on suhteellisen helppoa – ja varmasti nopeampaa kuin kartan kirjoittaminen kiinteästi koodiin!

Tallennus noudattaa samaa formaattia, jotta ne voitaisiin ladata takaisin peliin. Merkkien selitteiksi peli valitsee yksinkertaisesti aakkosten alkupäästä niin monta kirjainta kuin on tarpeen.

Lähdekoodin mukana on tallennukset save.txt, save2.txt, demo.txt sekä testsave.txt. Näistä testsaven loin ainoastaan testausta varten, ja save2:lla testasin kartan tekemistä. Save.txt sen sijaan on ”oikea”, pelattavissa oleva kartta, jonka peli oletuksena lataakin käynnistyksen yhteydessä. Samoin demo.txt on pelattava; sitä käytettiin mukana olevien peliohjeiden laatimisessa. Valitsemansa kartan voi ladata valitsemalla yläpalkin game-valikosta Load game. Jos lataaminen ei jostain syystä onnistu, se kerrotaan alanurkan statustekstissä.

Testaus

Yksikkötestejä en juuri pelille tehnyt. Mukana olevassa test.py-tiedostossa on lähinnä muodon vuoksi muutama, jolla testasin hahmon liikuttamista kartassa ja sitä, tunnistaako peli lailliset ja laittomat siirrot toisistaan. Niistä ei sen enmpää kerrottavaa ole; käytännössä testit liittyvät siihen, tunnistaako peli jos aion esimerkiksi liikkua seinän tai vihollispelaajan läpi tai hahmon kantaman ulkopuolelle.

Pääasiallinen testaus tapahtuikin peliä itse pelaamalla. Useita virheitä tekoälyssä huomasin, mutta juurikaan mitään ohjelmointivirheisiin liittyvää ei löytynyt – muutamat harvat selostettuna myöhemmin.

Testasin ohjelmaa sen rakennusvaiheessa ehkä vähän turhan harvoin, jolloin vikojen metsästäminen oli suhteellisen työlästä, mutta silti tarpeeksi usein, ettei se täysin mahdotonta ollut! Testauksesta ovat mukana esimerkiksi jo edellämainittu TestChar-hahmotyyppi, jonka ei ole tarkoituskaan olla tasapainoinen pelihahmo vaan jolla lähinnä testattiin eri hyökkäyksiö ja kykyjä, sekä Character- ja Board- luokkien __str__ - metodit, jotka tulostavat hahmon tiedot ja kartan konsoliin. Kun sain graafisen käyttöliittymän jokseenkin toimimaan, testaus siirtyi suurelta osin siihen – paitsi nimenomaan käyttöliittymään liittyvissä ongelmissa edellä mainitut metodit olivat erittäin hyödyllisiä!

Ohjelman tunnetut puutteet ja viat

Yksi olennainen puute on käyttöliittymän reagoiminen: kartan laattojen klikkaaminen ei toiminut, ja muutin sen niin, että kaikkien toimintojen tekeminen vaatii käytännössä kaksoisnapsautusta.

Toinen ongelma liittyy pelin stabiiliuteen: pitkistä debuggaussessioista huolimatta peliin on ilmeisesti jäänyt muutama bugi, jotka voivat kataa ohjelman, ja joiden syitä en silti ole löytänyt. Tätä tapahtui itselläni vain harvoin, mutta varmuuden vuoksi laitoin pelin tallentamaan itsensä backup.txt-tiedostoon joka vuoro, jotta mahdollisen kaatumisen sattuessa koko peliä ei menetetä.

Kolmas ongelma on jo aiemmin mainittu koodin epäselkeys ja epähierarkkisuus. Esimerkiksi käytännössä koko peli pyörii gui-luokassa (ja jossain määrin myös Square-luokassa), vaikka tarkoitus oli saada se game luokkaan – tämä ei kuitenkaan ollut kovin mielekästä, sillä se olisi vaatinut enemmän säätämistä kuin mitä se olisi tosiasiassa ratkaissut ongelmia, ainakin siinä määrin kuin itse yritin.

Parhaat ja huonoimmat kohdat

Ohjelman parhaat toteutukset liittyvät mielestäni laajennettavuuteen. Ensimmäisenä parhaana kohtana pitäisinkin sitä, kuinka helppoa peliin on lisätä uusia hahmoja, hyökkäykiä ja laattatyyppejä. Uuden hahmon lisääminen vaatii sen kuvien lisäämisen images-kansioon (yksi pelaajan sinisävyinen, yksi vastustajan punasävyinen ja yksi harmaasävyinen, jo liikkuneen hahmon kuva), uuden vakion lisäämistä Character-luokkaan, uuden luokan lisäämisä CharacterData-tiedostoon (minkä voi käytännössä kopioda suoraan vanhasta) ja sen perustietojen, kuten stattien ja rangen muokkaaminen. Lisäksi IO-luokkaan pitää lisätä pari riviä koodia. Tähän kaikkeen menee aikaa muutama minuutti, ja haastetta ei ole nimeksikään kun vain

muistaa kaiken. Uusien hyökkäysten luominen on vielä helpompaa; helpoin tapa on kopioda valmis hyökkäys jostain muusta luokasta ja muokata tiedot sopiviksi. Tähän menee aikaa yhdestä kahteen minuuttia. Toisaalta myös hahmojen kykyjen muuttaminen onnistuu hyvin helposti.

Toinen, mielestäni onnistunut asia ohjelmassa on ulkonäkö; peli näyttää ruudulla hyvältä, vaikka esimerkiksi ikkunoiden asetteluun olisi voinut nähdä enemmän vaivaa.

Huonoimpiin kohtiin sisältyy jo edellä mainittu koodin sekaisuus; kaikki asiat hoituvat, mutta voivat hyppiä tarpeettoman monen luokan kautta, ja luokkien vastuut ovat häilyviä. Tämä teki debuggaamisesta tarpeettoman vaikeaa, ja kuten edellä totesin, peli sisältää ilmeisesti vieläkin bugeja.

Toinen huono kohta on käyttöliittymän kanssa interaktio. Nyt se toimii jo paremmin kuin testausvaiheessa, mutta vaatii hieman totuttelua, eikä ehkä ole kovin intuitiivinen.

Poikkeamat suunnitelmasta

Muuten projekti noudatti aika hyvin suunnitelmaa, mutta aikataulun kanssa tuli ongelmia (myöhemmin lisää). Luokkia tosin tuli reilusti lisää kuin mitä suunnitelmassa oli, mutta se oli odotettavissakin. Toteutusjärjestys oli jokseenkin sama kuin suunnittelin: ensin kaikki muu ja sitte graafinen käyttöliittymä, joskin lisäsin peliin paljon sisältöä myös guin tekemisen jälkeen.

Toteutunut työjärjestys ja aikataulu

Teoriassa projekti toteutettiin juuri kuten suunnittelin: lähdin liikkeelle ylemmistä luokista kohti alempia, ja lopuksi toteutin guin. Ajankäyttö ei vain mennyt ihan kuten suunnittelin: käytännössä tein pelin alkupuolen yhdessä viikossa, jonka jälkeen muut kiireet veivät kaiken ajan; Graafinen käyttöliittymä sekä loppuosa pelistä valmistui noin kahdessa viikossa. Aikaa meni siis reilusti vähemmän kuin suunnitelmissa, vaikkakin varsinkin loppupuolella ohjelmointi olikin lähes kokopäiväistä. Tätä kirjoitan muutama päivä ensimmäisen deadlinen jälkeen, sillä palautuspäivänä peli ei kerta kaikkiaan ollut valmis palautettavaksi.

Arvio lopputuloksesta

Kaiken kaikkiaan olen melko tyytyväinen lopputulokseen. Lisää aikaa olisi voinut käyttää, ja olen melko varma että jos nyt olisi viikko tai pari lisää aikaa ohjelmasta olisi tullut paljon parempi; silti se on palautushetkellä mielestäni melko hyvä.

Oleellisia puutteita ei jo mainittujen lisäksi juuri ole, pelillisesti projekti vain on melko suppea. Jos nyt jatkaisin projektin kanssa (ja eiköhän sen kanssa tule vielä jotain tehtyä!), ensimmäisenä varmaan lisäisin peliin uusia maastotyyppejä, hahmoja ja kykyjä sekä mahdollisesti jopa tavarat, jotka kertaalleen jätin jo pois. Luokkajako oli melko onnistunut ja tärkeimmät algoritmit toimivat. Teköäly tosin kaipaisi vielä reipasta jatkokehitystä.

Pelin toteutus mahdollistaa sen jatkokehityksen ainakin tietyissä määrin.

Viitteet

Python-dokumentointi : https://docs.python.org/3/

Zetcoden pyqt-tutoriaalit: http://zetcode.com/gui/pyqt5/

PyQt-referenssi: http://pyqt.sourceforge.net/Docs/PyQt4/classes.html

A+:sta löytyvä kurssin oppimateriaali

Lisäksi pelimekaniikoista suuri(n) osa on lainattu Fire Emblem –peleistä.

Liitteet

Pelin lähdekoodi löytyy gitin projektin src-kansiosta, pelihahmojen kuvat src-kansiosta löytyvästä images-kansiosta.

Karkeat peliohjeet löytyy Suunnitelmat & dokumentointi –kansiosta.