Y2 Strategiapeli – dokumentaatio 6.5.2020

Henkilötiedot

Nimi	Ville Synkkänen
Opiskelijanumero	710688
Koulutusohjelma	Automaatio- ja informaatioteknologia
Vuosikurssi	2019

Yleiskuvaus

Projekti on vuoropohjainen strategiapeli, jossa pelataan tietokonevastustajaa vastaan. Pelikenttä on ruudukkopohjainen. Pelissä komennetaan joukkoa erilaisia yksiköitä, joilla on erilaisia vuoron aikana suoritettavia toimintoja, kuten erilaisia hyökkäyksiä ja muita kykyjä, jotka voivat aiheuttaa omiin ja vastustajan yksiköihin erilaisia vaikutuksia. Pelissä on erilaisia kenttiä, joissa on eri tavoin yksiköihin vaikuttavia maastoja. Pelitilanne on mahdollista tallentaa tiedostoon, mikä mahdollistaa kesken jääneen pelin jatkamisen myöhemmin. Lisäksi pelissä on kenttäeditori, jonka avulla pystyy helposti luomaan uusia kenttiä tai muokkaamaan olemassa olevia kenttiä.

Ominaisuuksia

- Graafinen käyttöliittymä
- Yksinkertaiset grafiikat
- Ruudukkopohjainen
- Omaan algoritmiin perustuva tekoäly
- A*-polunhakualgoritmi
- Erilaisia yksikkötyyppejä, joilla useita kykyjä
    - energiasysteemi: kykyjen käyttäminen vaatii energiaa, jota yksikkö saa pelin edetessä
    - passiivisia kykyjä, jotka vaikuttavat yksiköiden ominaisuuksiin
- Erilaisia maastoja
- Näkyvyysjärjestelmä: jotkut maastot voivat estää näkyvyyden
- Eri kokoisia ja maastoltaan vaihtelevia pelikenttiä
- Konfiguroitavuus ulkoisten asetustiedostojen kautta
    - yksiköiden ominaisuudet
    - eri maastojen ominaisuudet
    - käyttöliittymän asetukset
- Peliloki, johon tulostetaan pelin tapahtumat
- Mahdollisuus tallentaa pelitilanne tiedostoon
- Kenttäeditori
    - uusien kenttien luonti
    - olemassa olevien kenttien muokkaus

Mielestäni projektin ominaisuuslista on sen verran kattava, että projektin vaikeustaso on vaativa.

Käyttöohje

Ohjelma avautuu päävalikkoon. Päävalikon kautta pääsee jatkamaan tallennettua peliä (jos sellainen on), valitsemaan pelattavaa kenttää tai kenttäeditoriin luomaan uutta kenttää tai muokkaamaan olemassa olevaa kenttää.

Kenttäeditorissa on aluksi valittavana kaksi vaihtoehtoa: uuden kentän luominen ja olemassa olevan kentän muokkaaminen. Uusi kenttä luodaan syöttämällä kentän mitat ensimmäiseen tekstilaatikkoon välilyönnillä erotettuna ja painamalla UUSI KENTTÄ-nappia. Olemassa olevaa kenttää pääsee muokkaamaan syöttämällä kentän nimen ilman tiedostopäätettä toiseen tekstilaatikkoon ja painamalla MUOKKAA KENTTÄÄ-nappia. Tämän jälkeen ruudulle piirtyy kenttä. Kentälle voi lisätä elementtejä valitsemalla nappia painamalla haluamansa elementti ja klikkaamalla kentällä olevaa ruutua. Erityyppisten elementtien välillä voi vaihtaa painamalla VAIHDA YKSIKÖT/MAASTO-nappia. Kentän voi tyhjentää painamalla TYHJENNÄ KENTTÄ-nappia. Jos kentän haluaa tallentaa, se tapahtuu syöttämällä kentän nimi ilman tiedostopäätettä ensimmäiseen tekstilaatikkoon ja painamalla TALLENNA KENTTÄ-nappia. Olemassa olevan kentän tapauksessa voi joko tallentaa kentästä uuden version eri nimellä tai tallentaa vanhan kentän päälle syöttämällä vanhan kentän nimen tekstilaatikkoon. Editorista voi poistua tallentamatta kenttää painamalla POISTU EDITORISTA-nappia

Painamalla PELAA-nappia päävalikossa pääsee valitsemaan haluamansa kentän. Kenttiä voi selata käyttöliittymän napeilla. Kentän voi valita painamalla VALITSE-nappia, jolloin peli lataa valitun kentän. Valikosta pääsee takaisin päävalikkoon painamalla POISTU VALIKOSTA-nappia. JATKA-napin avulla pääsee suoraan tallennettuun peliin. Tallennettuja pelejä voi olla vain yksi kerrallaan.

Pelinäkymässä pelaaja voi valita yksikön klikkaamalla sitä, jolloin yksikkö muuttuu väriltään hieman kirkkaammaksi. Yksikön liikuttaminen tapahtuu klikkaamalla ruutua, joka on yksikön kantaman sisällä. Ruudut, joihin on mahdollista liikkua, on värjätty tummemmalla värillä kuin muut ruudut. Jos yksikkö pystyy hyökkäämään jonkin vihollisen kimppuun, pelaaja voi painaa HYÖKKÄÄ-nappia, jolloin mahdolliset hyökkäyksen kohteet muuttuvat väriltään kirkkaammiksi ja kantamalla olevat ruudut muuttuvat hieman tummemmiksi. Pitämällä hiirtä kohteen yläpuolella näkyviin tulevat yksikön tiedot ja hyökkäyksen odotettu tulos. Klikkaamalla vihollista yksikkö hyökkää vihollisen kimppuun. Hyökkäyksen tai kyvyn käyttämisen voi perua painamalla PERU KOHTEEN VALINTA-nappia. Kun yksikkö ei pysty enää tekemään mitään tällä vuorolla, se muuttuu väriltään tummemmaksi (kun yksikön valinta poistetaan).

Yksikön kyvyistä saa lisätietoa valitsemalla yksikön ja painamalla YKSIKÖN TIEDOT-nappia, jolloin pelilokin paikalle ilmestyy kuvaus yksikön kyvyistä. Samat tiedot saa näkyviin pitämällä hiirtä kykynapin päällä. Pelilokin saa takaisin näkyviin painamalla PELILOKI-nappia. Lisäksi maastoista ja yksiköistä saa lisätietoa pitämällä hiirtä niiden yläpuolella. Pelaaja voi selata toimivia yksiköitään käyttöliittymän napeilla. Yksikön valinnan voi poistaa painamalla POISTA YKSIKÖN VALINTA-nappia. Pelaaja voi päättää vuoronsa painamalla PÄÄTÄ VUORO-nappia. Pelistä voi poistua joko painamalla TALLENNA JA POISTU-nappia tai tallentamatta peliä painamalla POISTU PELISTÄ-nappia

Pelin edetessä yksikkö saa energiaa, jota se voi käyttää kykyihin. Jos yksikkö pystyy käyttämään kyvyn, kykynappi on aktiivinen. Nappia painamalla pääsee joko valitsemaan kohdetta kyvylle tai

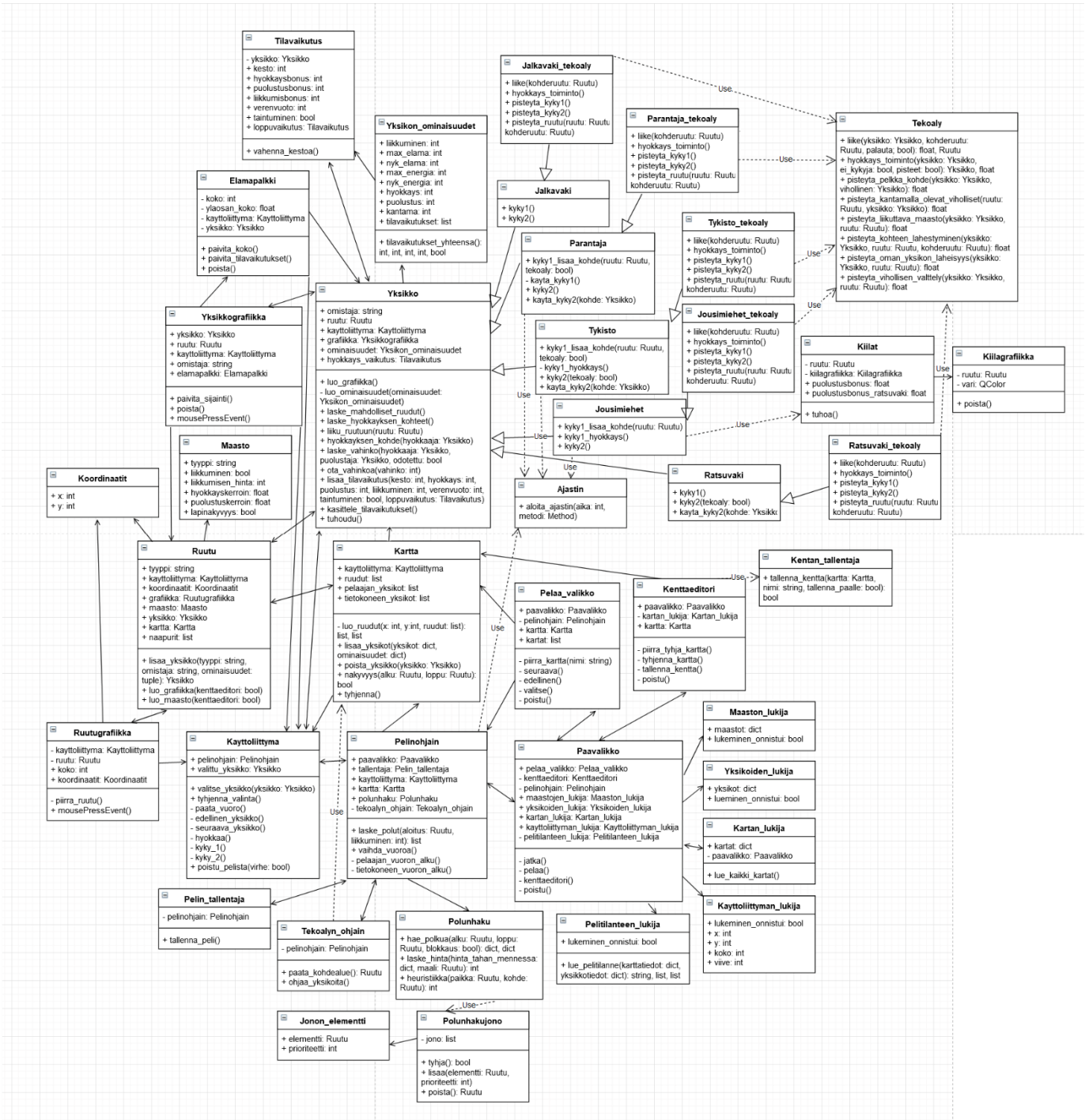
painallus aktivoi kyvyn suoraan, riippuen kyvystä. Jos yksikkö saa tilavaikutuksen, sen elämäpalkki muuttaa väriään. Pitämällä hiirtä palkin yläpuolella tilavaikutuksesta saa lisätietoa.

Ulkoiset kirjastot

Käyttöliittymän toteuttamiseen on käytetty PyQt5:ä.

Ohjelman rakenne

Luokkakaavio:



Ohjelma voidaan jakaa kolmeen keskeiseen osakokonaisuuteen: käyttöliittymä, pelin ohjaus ja yksiköt. Koko ohjelman perustana on päävalikko, joka hallinnoi tiedostojen lukemista ja ohjelman toimintaa korkeimmalla tasolla. Päävalikko on riippuvainen tiedostojen lukijoista, joihin kuuluvat Maaston_lukija, Yksiköiden_lukija, Kartan_lukija, Käyttöliittymän_lukija ja Pelitilanteen_lukija. Tiedostojen lukijat lukevat kaikki tiedostot ohjelman avautuessa ja ilmoittavat päävalikolle, jos lukemisessa tapahtuu virheitä. Päävalikon jatka-metodi jatkaa tallennettua peliä luomalla Pelinohjain-instanssin ja lisäämällä pelikentälle tallennetun tilanteen mukaiset yksiköt. Pelaa-metodi luo instanssin Pelaa_valikko-luokasta, jonka avulla pääsee valitsemaan haluamansa kentän. Kenttaeditori-metodi avaa kenttäeditorin uuteen ikkunaan ja poistu-metodi sulkee ohjelman.

Päävalikon jälkeen seuraavana hierarkiassa ovat Kenttäeditori, Pelaa_valikko ja Pelinohjain. Kenttäeditori hallinnoi nimensä mukaisesti kenttien luontia ja muokkausta. Lisäksi luokka on vastuussa editorin käyttöliittymän esittämisestä. Kenttäeditori käyttää Kartta-luokkaa kentän esittämiseen ja tietojen hallintaan. Kentän tallentamiseen käytetään Kentän_tallentaja-luokkaa. Kenttaeditorin piirra_tyhja_kartta-metodi luo tyhjän kartan muokkausta varten. Tyhjenna_kartta-metodi palauttaa kenttäeditorin alkutilaansa. Tallenna_kentta kutsuu Kentan_tallentaja-luokassa olevaa samannimistä metodia, joka tallentaa sillä hetkellä kenttäeditorissa auki olevan kentän.

Pelaa_valikko-luokka hallinnoi pelattavan kentän valitsemista ja siihen liittyvää käyttöliittymää. Luokan piirra_kartta-metodi piirtää sillä hetkellä valittuna olevan kartan. Seuraava- ja edellinen-metodeita käytetään karttojen selaamiseen. Valitse-metodi valitsee sillä hetkellä valittuna olevan kartan. Kun kartta on valittu, se luo Pelinohjain-luokan instanssin, joka ohjaa pelin kulkua.

Pelinohjain toimii pelitilanteen kuvaamisen perustana. Kun pelinohjain luodaan, se luo instanssin Kayttoliittyma-luokasta, joka hallinnoi pelin käyttöliittymää pelitilanteessa. Lisäksi pelinohjain käyttää Kartta-luokkaa pelikentän kuvaamiseen, Tekoälyn_ohjain-luokkaa tietokoneen yksiköiden hallintaan ja Polunhaku-luokkaa polkujen etsimiseen sekä Pelin_tallentaja-luokkaa pelitilanteen tallentamiseen. Tekoalyn_ohjain-luokan paata_kohdealue-metodi määrittelee kohderuudun tietokoneen yksiköille pelaajan yksiköiden sijaintien perusteella. Ohjaa_yksiköitä-metodi käy tietokoneen yksiköt yksitellen läpi ja suorittaa niiden toiminnot. Pelinohjaimen laske_polut-metodi laskee jokaiseen yksikön kantamalla olevaan ruutuun liikkumisen hinnan Polunhaku-luokan metodien avulla. Vaihda_vuoroa-metodi vaihtaa vuoron pelaajan vuorosta tietokoneen vuoroon tai toisin päin kutsumalla pelaajan_vuoron_alku-metodia tai tietokoneen_vuoron_alku-metodia. Nämä metodit käyvät läpi kaikki vuoron lopussa ja alussa suoritettavat asiat (kuten tilavaikutusten läpikäynti) ja vaihtavat sen jälkeen vuoroa.

Polunhaku-luokan hae_polku-metodi etsii algoritmin avulla lyhyimmän polun ruudusta toiseen. Metodi palauttaa kaksi sanakirjaa, joka sisältää läpi käydyt ruudut ja niihin liikkumisen hinnat. Laske_hinta-metodin avulla saadaan laskettua tiettyyn ruutuun liikkumisen hinta, kun sille annetaan parametrinä edellä mainitun metodin palauttama sanakirja. Heuristiikka-metodi laskee kahden ruudun välisen lyhyimmän etäisyyden. Polunhaku-luokka käyttää Polunhakujono-luokkaa apuna polkujen etsimisessä. Polunhakujonon alkioina käytetään Jonon_elementti-luokan instansseja. Polunhakujonon lisaa-metodi luo uuden alkion annettujen parametrien perusteella ja lisää sen jonoon. Poista-metodi poistaa ja palauttaa jonosta elementin, jonka prioriteetti on suurin.

Käyttöliittymä-luokkaa käytetään pelin käyttöliittymän kuvaamiseen pelitilanteessa. Käyttöliittymän valitse_yksikko-metodi valitsee parametrinä annetun pelaajan yksikön ja muuttaa

käyttöliittymää sen mukaisesti. Tyhjennä_valinta taas poistaa yksikön valinnan ja muuttaa käyttöliittymän alkuperäiseen muotoonsa. Paata_vuoro-metodi kutsuu pelinohjaimen vaihda_vuoroa-metodia. Edellinen_yksikko ja seuraava_yksikko-metodien avulla voidaan selata pelaajan yksiköitä. Hyokkaa-, kyky_1- ja kyky_2-metodeita käytetään valitun yksikön hyökkäyksen ja kykyjen käytön hallintaan. Hyokkaa-metodi värjää mahdolliset kohteet ja yksikön kantamalla olevat ruudut. Kykyjä kuvaavien metodien toiminta riippuu valitun yksikön tyypistä.

Kartta-luokka on keskiössä pelikentän ja -tilanteen kuvaamisessa. Kartta käyttää Ruutu-luokkaa pelikentän ruutujen kuvaamiseen ja Yksikko-luokkaa kentällä olevien yksiköiden kuvaamiseen. Kartta-luokan luo_ruudut-metodi luo kentän ruudut parametrinä annetun sanakirjan, leveyden ja korkeuden perusteella ja palauttaa ruuduista kaksi listaa, jotka tallennetaan Kartta-instanssiin. Lisaa_yksikot-metodi lisää kentälle yksiköt annettujen parametrien perusteella. Poista_yksikko-metodi poistaa parametrinä annetusta ruudusta siinä olevan yksikön. Nakyvyys-metodi palauttaa, näkeekö parametreina annetuista ruuduista toisiinsa.

Ruutu-luokka kuvaa pelikentän yksittäistä sijaintia. Luokka käyttää sijainnin kuvaamiseen Koordinaatit-luokkaa ja Maasto-luokkaa ruudun ominaisuuksien kuvaamiseen. Ruudun esittämiseen pelinäkymässä käytetään Ruutugrafiikka-luokkaa, jonka piirra_ruutu-metodi luo graafisen esityksen ruudusta ja siirtää sen oikeaan paikkaan pelikentällä. Ruutu-luokan lisaa_yksikko-metodi luo annettujen parametrejen mukaisen yksikön ja lisää sen ruutuun. Vastaavasti luo_grafiikka- ja luo_maasto-metodit luovat instanssin Ruutugrafiikka- ja Maasto-luokista.

Yksikot-luokka toimii pelissä yksikköjen kuvaamisen perustana. Luokka toimii eri yksikkötyyppien yläluokkana, jolta eri yksikkötyyppejä kuvaavat luokat perivät. Yksikön ominaisuuksia kuvaavien muuttujien säilömiseen käytetään Yksikon_ominaisuudet-luokkaa. Yksikköön vaikuttavia tilavaikutuksia kuvataan Tilavaikutus_luokalla, jonka vahenna_kestoa-metodi vähentää tilavaikutuksen kestoa yhdellä. Yksikköön vaikuttavat tilavaikutukset ovat säilöttynä Yksikön_ominaisuudet-luokassa sijaitsevassa listassa. Yksikön esittämiseen pelikentällä käytetään Yksikkografiikka-luokkaa. Yksikon luo_grafiikka-metodi luo Yksikkografiikka-instanssin ja luo_ominaisuudet-metodi luo Yksikon_ominaisuudet-instanssin annettujen parametrien perusteella. Laske_mahdolliset_ruudut-metodi laskee ruudut, joihin yksikkö pystyy liikkumaan ja laske_hyokkayksen_kohteet-metodi laskee kohteet, joita vastaan yksikkö voi hyökätä. Liiku_ruutuun-metodi liikuttaa yksikön parametrinä annettuun ruutuun. Hyokkayksen_kohde-metodia kutsutaan, kun yksikkö joutuu hyökkäyksen kohteeksi. Metodi laskee hyökkäyksen molemmille osapuolille aiheutuvan vahingon laske_vahinko-metodin avulla ja aiheuttaa molempiin vahinkoa kutsumalla ota_vahinkoa-metodia. Lisaa_tilavaikutus-metodi luo annettujen parametrien perusteella uuden Tilavaikutus-instanssin ja lisää sen Yksikon_ominaisuudet-luokassa sijaitsevaan listaan. Kasittele_tilavaikutukset-metodi käy läpi kaikki yksikön tilavaikutukset ja poistaa niitä, jos niiden kesto menee nollaan. Yksikön tuhoudu-metodi tuhoaa yksikön.

Yksikkografiikka-luokkaa käytetään yksikön esittämiseen pelikentällä. Kun Yksikkografiikka-instanssi luodaan, se piirtää pelikentälle yksikön luodun yksikkötyypin perusteella. Luokan paivita_sijainti-metodi siirtää grafiikan yksikön koordinaattien mukaiseen paikkaan pelikentällä. Poista-metodi poistaa grafiikan kentältä. Yksikkografiikka-luokka käyttää tukenaan yksikön esittämisessä Elamapalkki-luokkaa, joka esittää, kuinka paljon elämää yksiköllä on jäljellä. Lisäksi elämäpalkki muuttaa väriään, jos yksikkö saa tilavaikutuksen. Elamapalkki-luokan paivita_koko-

metodi muuttaa palkin kokoa sen perusteella, kuinka paljon elämää yksiköllä on. Vastaavasti paivita_tilavaikutukset-metodi muuttaa palkin väriä ja tooltippiä sen mukaan, onko yksiköllä tilavaikutuksia vai ei. Poista-metodi poistaa elämäpalkin.

Jalkavaki-, Jousimiehet-, Parantaja-, Tykisto- ja Ratsuvaki-luokat kuvaavat pelin eri yksikkötyyppejä. Ne perivät Yksikko-yläluokalta. Jokaisessa yksikkötyyppiä kuvaavassa luokassa on muuttujia ja metodeja, joilla kuvataan yksikköjen kykyjen toimintaa. Näiltä luokilta vuorostaan perivät Jalkavaki_tekoaly-, Jousimiehet_tekoaly -, Parantaja_tekoaly -, Tykisto_tekoaly - ja Ratsuvaki_tekoaly-luokat, jotka ovat tietokonevastustajan versioita edellä mainituista yksikkötyypeistä. Nämä luokat sisältävät metodeita, jotka ohjaavat tekoälyn toimintaa. Liike-metodi arvioi parhaan ruudun liikkumista varten pisteyta_ruutu-metodin avulla. Hyokkays_toiminto-metodi arvioi parhaan hyökkäyksen kohteen. Pisteyta_kyky1- ja pisteyta_kyky2-metodit pisteyttävät kykyjen käytön senhetkisen tilanteen perusteella. Kykyjen pisteytykseen käytetyt metodit ovat jokaiselle yksikkötyypille yksilöllisiä.

Kaikki edellä mainitut tietokoneen yksiköt käyttävät toiminnan pisteytyksessä apuna Tekoaly-luokan metodeja. Tekoaly-luokan liike-metodi pisteyttää kaikki ruudut, joihin yksikön on mahdollista liikkua ja liikuttaa yksikön sinne. Hyokkays_toiminto-metodi pisteyttää kaikki mahdolliset hyökkäyksen kohteet ja kyvyt sekä valitsee niistä parhaan vaihtoehdon. Pisteyta_kantamalla_olevat_viholliset-metodi pisteyttää ruudun sen perusteella, mitä yksiköitä vastaan yksikkö pystyisi hyökkäämään, jos se liikkuisi kyseiseen ruutuun. Pisteyta_liikuttava_maasto-metodi pisteyttää ruudun sen maaston perusteella. Pisteyta_kohteen_lahestyminen-metodi pisteyttää ruudun sen perusteella, kuinka lähellä se on parametrinä annettua kohderuutua. Pisteyta_oman_yksikon_laheisyys-metodi pisteyttää ruudun omien yksiköiden läheisyyden perusteella ja pisteyta_vihollisen_valttely-metodi pisteyttää ruudun vihollisen yksiköiden läheisyyden perusteella.

Jousimiehet-luokalla on kyky, joka pystyttää kiilat ruutuun, jossa yksikkö on. Kiilojen kuvaamiseen käytetään Kiilat-luokkaa ja graafiseen esitykseen Kiilagrafiikka-luokkaa. Lisäksi pelinohjain ja osa yksikkötyypeistä käyttää Ajastin-luokkaa, jonka aloita_ajastin-metodilla voidaan asettaa viive jonkin metodin suorittamiseen.

Monissa luokissa on säilöttynä viittaus käyttöliittymään, jonka kautta voidaan viitata lähes mihin tahansa objektiin pelissä. Tällä tavalla voidaan välttää suurien parametrimäärien pyörittely koodissa ja koodia on helpompaa laajentaa, kun kaikki tarvittavat viittaukset ovat helposti saatavilla. Huono puoli tässä toteutustavassa on se, että koodiin kirjoitettavista lauseista tulee helposti melko pitkiä.

Algoritmit

Ohjelman olennaisimmat algoritmit ovat polunhakualgoritmi, näkyvyysalgoritmi, tekoälyalgoritmi ja vahingon laskenta-algoritmi.

Ohjelman polunhaussa on käytetty A*-algoritmia. Algoritmi yrittää löytää polun, jota pitkin pääsee kohteeseen mahdollisimman halvalla. Se toimii muodostamalla polkupuun alkupisteestä loppupisteeseen. Aluksi algoritmi luo listan, jossa on vain aloitusruutu. Algoritmi laajentaa listaa lisäämällä sinne ruutuja, joiden arvioitu etäisyys kohteesta on mahdollisimman pieni. Listaa

laajennetaan, kunnes päädytään haluttuun kohteeseen. Tämän jälkeen kuljettava polku voidaan rakentaa käymällä listan alkiot läpi. Vaihtoehtoisesti voidaan vain laskea, kuinka paljon kohteeseen liikkuminen maksaa. Pelissä käytetään vain jälkimmäistä vaihtoehtoa, koska itse kuljetuilla poluilla ei ole merkitystä.

Algoritmin hyvä puoli on se, että se on hyvin tarkka ja suhteellisen tehokas. Testatessani algoritmia se päätyi aina valitsemaan lyhyimmän mahdollisen polun. Lisäksi algoritmi ei aiheuta pelin aikana juurikaan viivettä, vaikka jokaista yksikön kantamalla olevaa ruutua varten joudutaan laskemaan polku erikseen. Pelissä olisi voinut käyttää A*-algoritmin sijaan Dijkstran algoritmia, mutta se ei olisi ollut ihan yhtä tehokas kuin A*.

Näkyvyysalgoritmia käytetään, kun halutaan tietää, näkeekö jostain ruudusta toiseen ruutuun vai onko niiden välissä maastoa, joka estää näkyvyyden. Algoritmi toimii siten, että se ikään kuin piirtää janan ruutujen keskipisteiden välille. Sitten se lähtee kulkemaan janaa pitkin pienin vakiopituisin askelein. Jokaisen askeleen jälkeen tarkistetaan, onko edetty uuteen ruutuun. Jos ruutu vaihtuu, tarkistetaan, näkeekö ruudun läpi. Näin jatketaan, kunnes päästään loppuun, tai vastaan tulee ruutu, jonka läpi ei näe. Näkyvyysalgoritmin toteutus on mielestäni hyvä, koska se toimii suhteellisen tehokkaasti ja tarpeeksi tarkasti. Tarkkuutta voisi lisätä lyhentämällä askelten pituutta, mutta käytännössä sille tuskin on tarvetta, koska algoritmi on ainakin testauksessa toiminut täydellisesti.

Pseudokoodiesitys:

```
def näkyvyys(alku, loppu):
    laske alku- ja loppupisteiden välinen etäisyys
    if etäisyys > 0:
        laske askel niin, että askeleen pituus on vakio riippumatta etäisyydestä
    aseta nykyisen ruudun koordinaateiksi alkuruudun koordinaatit
    toistot = 0
    while toistojen määrä ei ole tullut täyteen:
        toistot += 1
        etene askel eteenpäin muuttamalla x- ja y-koordinaatteja
        if x-koordinaattien erotus > 0.5:
            nykyinen x-koordinaatti += 1
        elif x-koordinaattien erotus < -0.5:
            nykyinen x-koordinaatti -= 1
        if y-koordinaattien erotus > 0.5:
            nykyinen y-koordinaatti += 1
        elif y-koordinaattien erotus < -0.5:
            nykyinen y-koordinaatti -= 1
        if nykyinen ruutu ei ole läpinäkyvä ja nykyinen ruutu ei ole alku tai loppu:
            return False
    return True
```

Vahingon laskentaan käytettävät vakiot:

```
perusvahinko = 10
minimivahinko = 2
maksimivahinko = 40
satunnaisuuskerroin = 0.15
tukikerroin = 1.15
```

Vahingon laskentaan käytettävä algoritmi laskee hyökkääjän ja puolustajien voimien suhteen, joiden avulla se määrittelee molempien ottaman vahingon. Ensin algoritmi laskee molempien yksiköiden perusvoimat:

```
hyökkääjän voima = hyökkäys * maaston hyökkäyskerroin * (0.5 * (hyökkääjän elämä /
hyökkääjän maksimielämä) + 0.5)
puolustajan voima = puolustus * maaston puolustuskerroin * (0.5 * (puolustajan elämä /
puolustajan maksimielämä) + 0.5)

if hyökkääjän voima < 0:
    hyökkääjän voima = 0.00001
if puolustajan voima < 0:
    puolustajan voima = 0.00001
```

Sitten määritellään, onko hyökkääjällä muita yksiköitä tukemassa hyökkäystä ja jos on, hyökkääjän voima kerrotaan tukikertoimella. Tukibonus on mahdollista saada vain, jos hyökkääjä ja puolustaja ovat vierekkäisissä ruuduissa:

```
tukibonus = False
if etäisyys == 1:
    tarkista tukibonus
if tukibonus:
    hyökkääjän voima *= tukikerroin
```

Seuraavaksi tarkistetaan, onko hyökkääjänä jousimiehet, jotka saavat bonuksen hyökkäyksessä jalkaväkeä ja ratsuväkeä vastaan:

```
if hyökkääjä on jousimiehet ja puolustaja on jalkaväki tai ratsuväki:
        hyökkääjän voima *= jousimiesten hyökkäysbonus
```

Tämän jälkeen tarkistetaan, saavatko yksiköt inspiraatiobonusta parantajalta:

```
hyökkääjän voima *= hyökkääjän inspiraatiobonus
puolustajan voima *= puolustajan inspiraatiobonus
```

Sitten tarkistetaan, onko puolustaja kiiloissa ja annetaan puolustusbonus, jos on. Bonus on huomattavasti suurempi ratsuväkeä vastaan:

```
if puolustaja kiiloissa:
    if hyökkääjä on ratsuväki:
        puolustajan voima *= kiilojen ratsuväkibonus
    else:
        puolustajan voima *= kiilojen bonus
```

Jos tykistö hyökkää tykistöä vastaan, aiheutettu vahinko on paljon pienempi. Tämä toteutetaan jakamalla perusvahinko kolmella:

```
if puolustaja on tykistö ja hyökkääjä on tykistö:
    perusvahinko /= 3
```

Joissain tapauksissa on mahdollista hyökätä oman yksikön kimppuun (tiettyjen kykyjen avulla). Tällöin hyökkäävä yksikkö ei ota vahinkoa. Hyökkääjä ei ota myöskään vahinkoa, jos se ei ole puolustajan vieressä. Osapuolien ottama vahinko on niiden voimien suhde kertaa perusvahinko:

```
if etäisyys == 1 and puolustajan omistaja ei ole hyökkääjän omistaja
    hyökkääjän vahinko = (puolustajan voima / hyökkääjän voima) * perusvahinko
    if hyökkääjän vahinko < minimivahinko:
        hyökkääjän vahinko = minimivahinko
    elif hyökkääjän vahinko > maksimivahinko:
        hyökkääjän vahinko = maksimivahinko
else:
    hyökkääjän vahinko = 0

puolustajan vahinko = (hyökkääjän voima / puolustajan voima) * perusvahinko
if puolustajan vahinko < minimivahinko:
    puolustajan vahinko = minimivahinko
elif puolustajan vahinko > maksimivahinko:
    puolustajan vahinko = maksimivahinko
```

Kun molempien osapuolien vahinko on määritelty, arvotaan todellinen vahinko. Vahinko voi vaihdella (satunnaiskertoimen mukaisesti) +-15 %:

```
hyökkääjän minimivahinko = hyökkääjän vahinko * (1 - satunnaisuuskerroin))
hyökkääjän maksimivahinko = hyökkääjän vahinko * (1 + satunnaisuuskerroin)

puolustajan minimivahinko = puolustajan vahinko * (1 - satunnaisuuskerroin)
puolustajan maksimivahinko = puolustajan vahinko * (1 + satunnaisuuskerroin)

arvo hyökkääjän vahinko minimivahingon ja maksimivahingon väliltä
arvo puolustajan vahinko minimivahingon ja maksimivahingon väliltä

return hyökkääjän vahinko, puolustajan vahinko
```

Vahingon laskennan olisi voinut toteuttaa hyvin monella eri tavalla, mutta käyttämäni tapa tuntui luonnolliselta ja vaikuttaa testauksen perusteella sopivan peliin hyvin.

Pelin selvästi suurin algoritmi on tekoälyalgoritmi. Algoritmin perusidea on se, että jokainen yksikkö käy läpi kaikki mahdolliset toimintavaihtoehdot, pisteyttää ne ja valitsee niistä parhaan. Tekoäly on hyvin aggressiivinen: yksiköt pyrkivät lähestymään pelaajan yksiköitä.

Tietokoneen vuoron alussa tekoälyn ohjain määrittelee ensin kohderuudun, jota kohti yksiköt pyrkivät liikkumaan. Kohderuutu määritellään laskemalla pelaajan yksiköiden koordinaattien painotettu keskiarvo. Erityyppisillä yksiköillä on eri painokertoimet. Jos kohderuuduksi tulee tällä tavalla ruutu, jossa ei ole pelaajan yksiköitä, etsitään ruutua lähimpänä oleva ruutu, jossa on pelaajan yksikkö ja asetetaan se kohderuuduksi:

```
x = 0
y = 0
määrä = 0
kerroin = 1
for yksikkö in pelaajan yksiköt:
    kerroin *= yksikön painokerroin
    x += kerroin * yksikön x-koordinaatti
    y += kerroin * yksikön y-koordinaatti
    määrä += kerroin
if määrä > 0:
    laske koordinaattien keskiarvot
```

```
    pyöristä keskiarvot kokonaisluvuiksi
    if kohderuudussa on pelaajan yksikkö
        return kohderuutu
    else:
        etsi ruutua lähin vihollinen
        kohderuutu = lähimmän vihollisen ruutu
        return kohderuutu

return ruutu, jossa on pelaajan ensimmäinen yksikkö
```

Tämän jälkeen tekoälyn ohjain ohjaa tietokoneen yksiköitä yksi kerrallaan: ensin suoritetaan kaikkien yksiköiden liikkuminen, jonka jälkeen suoritetaan kaikkien yksiköiden hyökkäykset. Sekä liikkumisessa ja hyökkäyksessä yksiköillä on tietty järjestys, jotta päästäisiin mahdollisimman optimaaliseen tulokseen. Järjestyksessä tapahtuva liikkuminen ja hyökkäys toteutetaan lajittelemalla yksiköt sisältävä lista ennen toimintojen suorittamista. Lajittelua varten on määritelty etukäteen liikkumis- ja hyökkäysjärjestykset:

```
def lajittele listat(toiminto):
    luodaan uusi tyhjä lista
    if toiminto on liikkuminen
        i = 0
        while i < yksikkötyyppien määrä
            for yksikkö in tietokoneen yksiköt:
                if yksikön nimi == liikkumisjärjestys[i]:
                    lisää yksikkö listaan
            i += 1
    elif toiminto on hyökkäys
        i = 0
        while i < yksikkötyyppien määrä
            for yksikkö in tietokoneen yksiköt:
                if yksikön nimi == hyökkäysjärjestys[i]:
                    lisää yksikkö listaan
            i += 1
    tietokoneen yksiköt = uusi lista
```

Liikkumisen pisteytys tapahtuu käymällä kaikki yksikön kantamalla olevat ruudut läpi ja arvioimalla niitä erilaisten parametrien kannalta. Pisteytykseen vaikuttavat mahdolliset hyökkäyksen kohteet, maaston puolustus- ja hyökkäyskertoimet, kohderuudun lähestyminen, omien yksiköiden läheisyys ja joillain yksikkötyypeillä etäisyys vihollisista. Seuraavaksi esitellään metodit, joiden avulla pisteytys tapahtuu. Koodissa esiintyvät erilaiset kertoimet ovat yksikölle ominaisia vakioita, jotka ohjaavat yksikön toimintaa. Kertoimet on määritelty yksikkötyyppien tekoälyä kuvaavissa luokissa.

Kantamalla olevat viholliset pisteytetään laskemalla mahdolliset hyökkäyksen kohteet, jos yksikkö olisi kyseisessä ruudussa. Kaikki mahdolliset kohteet käydään läpi ja jokaista varten lasketaan kerroin. Kertoimen laskukaavat on esitetty alla olevassa pseudokoodissa.

```
def pisteytä kantamalla olevat viholliset(ruutu, yksikkö):
    pisteet = 10
    lasketaan kantamalla olevat viholliset
    for vihollinen in viholliset:
        kerroin = yksikön priorisaatiokerroin  # (riippuu vihollisen tyypistä)

        elämäkerroin = 1 / sqrt(vihollisen elämän osuus maksimielämästä)
        if elämäkerroin > yksikön maksimielämäkerroin
```

```
            elämäkerroin = yksikön maksimielämäkerroin
        kerroin *= elämäkerroin

        puolustuskerroin = yksikön hyökkäys / vihollisen puolustus
        if puolustuskerroin > yksikön maksimipuolustuskerroin:
            puolustuskerroin = yksikön maksimipuolustuskerroin
        elif puolustuskerroin < yksikön minimipuolustuskerroin:
            puolustuskerroin = yksikön minimipuolustuskerroin
        kerroin *= puolustuskerroin

        maastokerroin = 1
        maastokerroin *= 1 / sqrt(vihollisen maaston hyökkäyskerroin)
        maastokerroin *= 1 / sqrt(vihollisen maaston puolustuskerroin)
        if maastokerroin > yksikön maksimimaastokerroin:
            maastokerroin = yksikön maksimimaastokerroin
        elif maastokerroin < yksikön maksimimaastokerroin:
            maastokerroin = yksikön maksimimaastokerroin
        kerroin *= maastokerroin


        if vihollisen vieressä oma yksikkö ja ruutu vihollisen vieressä:
            kerroin *= yksikön tukikerroin

        if vihollisen ruudussa kiilat:
            if yksikkö on ratsuväki
                kerroin *= 1 / kiilojen puolustusbonus ratsuväkeä vastaan
            else:
                kerroin *= 1 / kiilojen puolustusbonus

    määrittele suurin kerroin
    pisteet *= suurin kerroin
    return pisteet
```

Maaston pisteytys tapahtuu lisäämällä kertoimeen maaston hyökkäyskertoimen ja puolustuskertoimen vaikutus alla olevan pseudokoodin mukaisesti. Vaikutus on positiivinen, jos maaston kerroin on > 1 ja negatiivinen, jos se on < 1. Jos ruudussa on kiilat, kerroin kerrotaan lopuksi kiilojen puolustusbonuksella.

```
def pisteytä maasto(yksikkö, ruutu):
    maastokerroin = 1
    maastokerroin += yksikön hyökkäyskerroin * (maaston hyökkäyskerroin- 1)
    maastokerroin += yksikön puolustuskerroin * (maaston puolustuskerroin - 1)
    if ruudussa on kiilat:
        maastokerroin *= kiilojen puolustusbonus
    return maastokerroin
```

Kohteen lähestyminen pisteytetään laskemalla lyhin etäisyys ruudusta kohderuutuun. Jos etäisyys on yksikön maksimietäisyyttä pienempi (tai yhtä suuri), kerroin lasketaan suoran yhtälöstä alla esitetyllä tavalla. Muussa tapauksessa kerroin on yksi.

```
def pisteytä kohteen lähestyminen(yksikkö, ruutu, kohderuutu):
    kerroin = 1
    lasketaan lyhin etäisyys ruudusta kohderuutuun
    if etäisyys <= yksikön maksimietäisyys kohteesta:
        kerroin = suoran kulmakerroin * etäisyys + maksimilähestymisbonus
    return kerroin
```

Oman yksikön läheisyys pisteytetään laskemalla kaikki yksiköt, jotka ovat läheisyysbonuksen vaatimalla etäisyydellä ruudusta. Jos yksiköitä on tarpeeksi, etäisyyskerroin on yksikölle tyypillinen lähestymisbonus. Muussa tapauksessa etäisyyskerroin on yksi.

```
def pisteytä oman yksikön läheisyys(yksikkö, ruutu):
    etäisyyskerroin = 1
    yksiköt = 0
    for oma yksikkö in tietokoneen yksiköt :
        laske oman yksikön etäisyys ruudusta
        if etäisyys < yksikön läheisyysbonuksen kantama ja oma yksikkö ei ole yksikkö:
            yksiköt += 1
    if yksiköt >= bonukseen tarvittavien yksiköiden määrä:
        etäisyyskerroin = yksikön lähestymisbonus
    return etäisyyskerroin
```

Vihollisten välttelyn pisteytys tapahtuu laskemalla kaikkien vihollisten liikkumisen hinta pisteytettävään ruutuun. Jos liikkumisen hinta on yksikön kantamaa pienempi, kerrointa pienennetään alla esitetyllä tavalla. Jos ruudun lähellä on useita vihollisia, lopulliseksi kertoimeksi tulee lasketuista kertoimista pienin.

```
def pisteytä vihollisten välttely(yksikkö, ruutu):
    kerroin = 1
    for vihollinen in pelaajan yksiköt:
        laske vihollisen liikkumisen hinta ruutuun (etäisyys)
        if etäisyys < yksikön kantama:
            uusi kerroin = (etäisyys / yksikön kantama)**yksikön etäisyyseksponentti
            if uusi kerroin < kerroin:
               kerroin = uusi kerroin
    return kerroin
```

Edellä esitellyistä metodeista saadut kertoimet kerrotaan keskenään, jolloin saadaan yhden ruudun pistemäärä. Tämän jälkeen yksikkö valitsee ruuduista parhaan ja liikkuu siihen.

Hyökkäyksen pisteytys toimii pitkälti samalla periaatteella kuin liikkumisen pisteytys. Ensin lasketaan jokaiselle hyökkäykselle odotettujen vahinkojen suhde, joka kerrotaan hyökkäyksen kohteesta riippuvalla painokertoimella ja vielä lopuksi kohteen elämästä riippuvalla kertoimella. Tarkemmat kaavat on kuvattu pseudokoodissa. Tämä pistemäärä tallennetaan sanakirjaan myöhempää käyttöä varten.

```
def hyökkäyksen pisteytys(yksikkö):
    luo tyhjä vaihtoehdot-sanakirja
    laske mahdolliset hyökkäyksen kohteet
    for vihollinen in hyökkäyksen kohteet
        laske hyökkääjän ja puolustajan odotettu vahinko
        suhde = (puolustajan vahinko + 0.001) / (hyökkääjän vahinko + 0.001)

        suhde *= yksikön priorisaatiokerroin   # (riippuu vihollisen tyypistä)
        elämäkerroin = 1 / sqrt(vihollisen maksimielämä / vihollisen nykyinen elämä)
        if elämäkerroin > yksikön maksimielämäkerroin:
            elämäkerroin = yksikön maksimielämäkerroin
        suhde *= elämäkerroin
        tallenna pisteet vaihtoehdot-sanakirjaan
```

Tämän jälkeen pisteytetään kyvyt (jos yksiköllä on tarpeeksi energiaa niiden käyttämiseen). Pisteytykseen käytetään jokaiselle yksikkötyypille yksilöllisiä pisteytysalgoritmeja, joita en aio käydä tässä tarkemmin läpi, koska se veisi liikaa aikaa. Pisteet tallennetaan samaan sanakirjaan kuin hyökkäyksen kohteet. Pisteytyksen jälkeen sanakirja käydään läpi ja sieltä valitaan parhaat pisteet omaava vaihtoehto, joka palautetaan.

```
    korkeimmat pisteet = 0
    paras kohde = None
    for vaihtoehto in vaihtoehdot:
        if vaihtoehto ei ole None
            if vaihtoehdon pisteet > korkeimmat pisteet ja vaihtoehdon pisteet >= 1:
                korkeimmat pisteet = vaihtoehdon pisteet
                paras kohde = vaihtoehto
    return paras kohde
```

Tämän jälkeen yksikkö suorittaa parasta kohdetta vastaavan toiminnon. Toiminto voi olla hyökkäys, kyvyn käyttäminen tai jossain tapauksessa ei mitään.

Tämä kuvaus ei kata tekoälyn toimintaa täysin, mutta siitä pitäisi saada suhteellisen hyvä kuvan sen toiminnasta. Tekoälyn toteutuksesta tuli monimutkaisempi kuin olin alun perin suunnitellut. Tässä toteutustavassa hyvänä puolena on se, että tekoälyn toimintaa pystyy hienosäätämään helposti pelkkien muuttujien arvojen avulla. Lisäksi tekoäly kykenee monissa tilanteissa yllättävänkin älykkääseen päätöksentekoon. Tekoäly osaa myös hyödyntää kaikkia yksiköiden kykyjä. Huonona puolena siinä on se, että algoritmi on suhteellisen monimutkainen eikä ole kovinkaan tehokas. Lisäksi tekoäly tekee usein liikkeitä, jotka ovat selvästi tyhmiä tai vähintäänkin epäoptimaalisia. Algoritmin puutteista huolimatta se on mielestäni paljon parempi kuin tilakoneen avulla toteutettu tekoäly olisi ollut. Tilakoneen avulla olisi ollut hyvin hankala toteuttaa näin monimutkaista tekoälyä ja siitä olisi todennäköisesti tullut vain iso sotku. Algoritmia jatkokehittämällä siitä voisi tulla oikeasti hyvä, mutta se vaatisi hyvin suuria muutoksia.

Tietorakenteet

Ohjelmassa on käytetty monipuolisesti erilaisia tietorakenteita. Tapauksissa, joissa on tarvittu muuttuvia rakenteita, on käytetty listoja tai joissain tapauksissa sanakirjoja. Listoja on käytetty muun muassa pelikentän ruutujen, yksiköiden, tilavaikutusten ja hyökkäyksen mahdollisten kohteiden käsittelyyn, koska näitä tulee pystyä muuttamaan pelin aikana. Ruutujen säilömisessä olisi voinut periaatteessa käyttää tupleja, mutta kenttäeditoria varten olisi joka tapauksessa tarvinnut muuttuvan rakenteen. Käyttämällä listoja pystyttiin käsittelemään ruutuja samalla tavalla kaikissa käyttötapauksissa ja säästyttiin ylimääräisen koodin kirjoittamiselta. Sanakirjoja on käytetty muun muassa karttojen tietojen tallentamisessa ja asetustiedostojen tietojen käsittelyssä, jotta tiedostoista luettuja muuttujia pystytään käyttämään mahdollisimman helposti. Lisäksi polunhakualgoritmi käyttää sanakirjoja ruutuun liikkumiseen vaadittavan pistemäärän tallentamiseksi. Tuplejen käyttö on jäänyt ohjelmassa vähemmälle. Niitä on käytetty erilaisissa tilanteissa, joissa on varmaa, että tietoja ei tarvitse muuttaa myöhemmin. Hyvä esimerkki tästä on tiedostosta luettujen karttojen tiedot, jotka tallennetaan kukin omaan tupleensa, johon sisältyy kartan mitat, yksiköt ja ruudut.

Ohjelmassa on lisäksi käytetty useita omatekoisia tietorakenteita. Koordinaatit-luokka esittää nimensä mukaisesti koordinaatteja x, y-koordinaatistossa. Prioriteettijonon elementti sisältää

varsinaisen elementin ja sen prioriteettia kuvaavan luvun. Polunhakujono on jono, joka sisältää edellä mainittuja jonon elementtejä. Jonoa käytetään polunhaussa lyhyimmän polun rakentamiseen. Jonoon on mahdollista lisätä elementti tai poistaa ja palauttaa elementti, jolla on suurin prioriteetti (pienin prioriteettiluku). Vaihtoehtoisesti polunhaussa olisi voinut käyttää Pythonin omaa heapq-moduulia, mutta se ei sopinut yhteen polunhaun toteutustapani kanssa, joten jouduin kehittämään oman korvaavan tietorakenteen. Lisäksi tilavaikutukset ja yksikön ominaisuudet on sisällytetty kummatkin omaan tietorakenteeseensa. Tilavaikutukset-luokka sisältää tiedon tilavaikutuksen omaavasta yksiköstä, vaikutuksen kestosta ja muista vaikutukseen liittyvistä muuttujista. Vastaavasti yksikön_ominaisuudet-rakenne sisältää tiedot yksikön tyypistä ja yksikön ominaisuudet määrittelevistä muuttujista. Omatekoisten rakenteiden sijaan olisi ollut mahdollista käyttää tupleja tai listoja, mutta omien rakenteiden käyttö on varsinkin koodin luettavuuden kannalta paljon selkeämpää.

Tiedostot

Ohjelman käsittelemät tiedostot voidaan jakaa kolmeen päätyyppiin: tallennustiedostoihin, asetustiedostoihin ja karttatiedostoihin. Kaikki tiedostot ovat tekstitiedostoja.

Ohjelma tallentaa pelitilanteen pelitilanne-kansiossa sijaitsevaan pelitilanne.txt-tiedostoon. Tiedoston alussa on kerrottu kentän nimi tiedostopäätteen kanssa. Tämän jälkeen tiedostossa on listattu yksiköt jokainen omalla rivillään. Yksikön tiedot ovat pilkuilla erotettuna järjestyksessä: x-koordinaatti, y-koordinaatti, omistaja (PLR tai COM), tyyppi, elämä, energia, liikkuminen käytetty, hyökkäys käytetty, loppuvaikutus ja tilavaikutukset. Tilavaikutuksessa on listattu sanan tilavaikutus ja kaksoispisteen jälkeen vaikutukseen liittyvät parametrit järjestyksessä: kesto, hyökkäysbonus, puolustusbonus, liikkumisbonus, verenvuoto, taintuminen ja loppuvaikutus, jonka jälkeen tulee puolipiste. Yksiköiden jälkeen omalla rivillään on listattu kiilojen paikat kentällä. X- ja y-koordinaatit on erotettu pilkuilla ja kiilat on erotettu kaksoispisteillä. Tiedoston viimeisellä rivillä on sana LOPPU. Käyttäjän on mahdollista halutessaan muokata tiedostoa. Dokumentaatio-kansiossa on esimerkkitiedosto, joka havainnollistaa, miltä tiedostoformaatti näyttää.

Ohjelman asetustiedostot on tallennettu maastot-, yksikot- ja muut-kansioihin. Maastot-kansio sisältää maastoihin liittyvät asetustiedostot. Jokaisessa tiedostossa on listattu maaston tyyppi, liikkuminen, liikkumisen hinta, puolustus- ja hyökkäyskertoimet, läpinäkyvyys ja ruudun väri RGB-muodossa kukin omalla rivillään. Tiedoston viimeisellä rivillä on sana LOPPU. Yksiköt-kansio sisältää vastaavasti yksiköihin liittyvät asetustiedostot. Tiedostojen formaatti on hyvin samankaltainen kuin maastoilla. Jokaisesta yksiköstä on listattu tyyppi, liikkumispisteet, maksimielämä ja -energia, hyökkäys, puolustus, kantama ja pistehinta (ei tee mitään pelin nykyisessä versiossa) sekä yksikön kykyihin liittyvät parametrit, jotka ovat jokaiselle yksikölle yksilöllisiä. Muut-kansiossa on käyttöliittymän asetustiedosto, jonka formaatti on myös muiden asetustiedostojen kaltainen. Tiedostossa on listattuna näytön resoluutio (erottimena *-merkki), joka määrittää, mihin kohtaan näyttöä ohjelman ikkuna siirretään alussa, ikkunan koko ja tekoälyn toimintojen välinen viive.

Testaus

Ohjelmaa on testattu sekä kokeilemalla sen ominaisuuksia käytännössä että yksikkötestauksen avulla. Ohjelman ominaisuuksien toimintaa on testattu käytännössä kehityksen aikana sitä mukaa,

kun ne ovat tulleet valmiiksi. Lisäksi ohjelman valmistumisen jälkeen käytiin vielä läpi kaikki ohjelman keskeiset osa-alueet ja varmistettiin niiden oikea toiminta. Käyttöliittymää testattiin kokeilemalla erilaisia syöteyhdistelmiä nappeihin ja pelikenttään eri tilanteissa.

Yksikkötestauksen avulla on testattu ohjelmassa tapahtuvaa laskentaa, tekoälyn yleisiä metodeja, näkyvyysjärjestelmää ja polunhakua. Laskentaa testattiin kutsumalla laskentaan liittyviä metodeja ja tarkastamalla, että ne palauttavat oikeanlaisia tuloksia. Tekoälyn metodeja testattiin siten, että tekoälylle annettiin erilaisia toimintavaihtoehtoja ja katsottiin, valitseeko se vaihtoehdoista parhaan. Jokaiselle yksikkötyypille yksilölliset tekoälyn osat jäivät ajan puutteen vuoksi testaamatta. Näkyvyyttä testattiin yksinkertaisesti laskemalla näkyvyys monessa eri tilanteessa ja tarkistamalla, että palautetut arvot olivat oikeita. Polunhakua testattiin laskemalla, kuinka monta pistettä johonkin ruutuun liikkumisessa kuluu ja tarkistamalla, ovatko palautetut arvot oikeita. Lisäksi testattiin tapauksia, joissa kohderuutu oli blokattuna.

Testaus vastasi muuten suunnitelmassa esitettyä, mutta tiedostojen lukemisen testaus tapahtui yksikkötestien sijaan testaamalla, miten ohjelma reagoi tiedostojen muokkaamiseen. Tein näin siksi, että sain tällä testaustavalla hyviä tuloksia eikä yksikkötestaus olisi tuonut testaukseen merkittävää lisäarvoa.

Puutteet ja viat

Ohjelman merkittävimmät puutteet ovat tekoälyssä ja käyttöliittymässä. Tekoäly toimii joissain tilanteissa yllättävänkin hyvin, mutta joskus se saattaa tehdä jotain, joka on selvästi epäoptimaalista tai suorastaan typerää. Tietokoneen yksiköt saattavat esimerkiksi jäädä johonkin ruutuun, jota ne pitävät hyvänä puolustusasemana ja antaa pelaajan tykistön tuhota ne helposti. Tämän ongelman olisi voinut ainakin osittain korjata hienosäätämällä tekoälyn parametrejä, mutta siihen ei ollut valitettavasti aikaa. Toinen ongelma liittyy liikkumiseen: koska tietokoneen yksiköillä on tietty liikkumisjärjestys, ne saattavat blokata toistensa liikkeen joskus. Tämän olisi voinut välttää antamalla yksiköiden liikkua toistensa läpi, mutta se olisi vaatinut muutoksia pelin muihin osiin. Ongelman olisi voinut myös ratkaista jonkinlaisella dynaamisella liikkumisjärjestyksellä.

Lisäksi tekoälyalgoritmin tehokkuus voisi olla parempi. Varsinkin yksiköillä, joilla on pitkä kantama, on paljon erilaisia vaihtoehtoja, jotka kaikki täytyy pisteyttää, mikä voi johtaa joissain tilanteissa muutaman sadan millisekunnin viiveisiin. Tilanteen korjaaminen todennäköisesti vaatisi koko algoritmin uudelleen kirjoittamista, mutta yksinkertaistamalla algoritmia voisi jo päästä suhteellisen hyviin tuloksiin. Pelattavuuden kannalta suurin ongelma on se, että tekoälyn päätöksenteon aiheuttaman viiveen lisäksi on olemassa toiminnan välinen viive. Tietokoneen vuorojen kestoa voi vähentää vähentämällä toiminnan välistä viivettä asetustiedostoista, mutta tällöin ongelmana on, että eri yksiköiden toimintojen välinen viive vaihtelee suuresti ja pelistä tulee vaikeampi seurata. Jos viiveessä voisi jotenkin huomioida päätöksenteon keston, viive voisi olla aina vakio ja ongelma ei olisi enää läheskään yhtä merkittävä, mutta en tiedä miten sen voisi toteuttaa.

Pelin käyttöliittymässä on muutama selkeä puute. Tekstien fontteja ei pysty muuttamaan, joten tekstiä voi olla hankalaa lukea pieneltä näytöltä ja pienemmällä ikkunan koolla tekstit saattavat mennä joissain tilanteissa päällekkäin. Tekstien fontit voisi lisätä tiedostosta luettaviin asioihin. Lisäksi pelilokin seuraaminen on hankalaa, koska siellä jätetään monista asioista yksityiskohdat

kertomatta. Lisäksi pelilokiin mahtuu vai pieni määrä tekstiä. Tämän ongelman voisi ratkaista tekemällä lokista elementin, jota pystyy selaamaan ylös- ja alaspäin. Lisäksi peli voisi antaa käyttäjälle enemmän ohjeita ja käyttöliittymän ulkoasu voisi olla hieman hiotumpi.

Ainoa tuntemani bugi ohjelmassa on se, että joskus yksiköiden valitseminen klikkaamalla ei toimi. Tämä korjautuu, kun yksiköitä selaa nappien avulla. En tiedä mistä tämä bugi johtuu.

Parhaat ja heikoimmat kohdat

Ohjelman parhaiden kohtien valitseminen on hyvin hankalaa, koska mielestäni ohjelma on melko tasalaatuinen, eikä siinä ole mielestäni mitään tiettyä ominaisuutta, joka on muuta ohjelmaa selvästi parempi. Parasta ohjelmassa on mielestäni sen monipuolisuus ja toimivuus. Pelissä on monia erityyppisiä maastoja ja yksiköitä, joilla on monipuolisia kykyjä ja pelin tekoäly osaa hyödyntää näitä kaikkia. Kenttäeditorin avulla on helppo luoda peliin lisää sisältöä ja peli on suureksi osaksi konfiguroitavissa asetustiedostojen kautta. Pelin ominaisuudet toimivat hyvin enkä ole havainnut ohjelman viimeisimmässä versiossa testauksen aikana yhtään kaatumisia. Yhtenä hienona ominaisuutena mainitsen vielä sen, että pelaajan yksiköt, jotka eivät pysty tekemään mitään, muuttuvat väriltään tummemmaksi ja niiden yli hypätään, kun yksiköitä selataan käyttöliittymän napeilla.

Ohjelman selvästi heikoimmat kohdat ovat tekoälyn toiminta, tekoälyalgoritmin tehokkuus ja käyttöliittymä Puutteet ja viat -kohdassa esitettyjen syiden vuoksi.

Poikkeamat suunnitelmasta

Projekti sujui pääosin suunnitelman mukaisesti. Yksiköiden asettelu pelikentälle jäi toteuttamatta ajan puutteen vuoksi, mutta muuten sain kaikki suunnittelemani ominaisuudet toteutettua. Kaiken kaikkiaan projektiin kului selvästi enemmän aikaa kuin olin suunnitellut. Suunnitelman mukainen ajankäyttöarvio oli noin 80-90 tuntia, mutta todellisuudessa projektiin kului noin 110-125 tuntia (pois lukien dokumentaation kirjoittamiseen kulunut aika). Tämä johtui lähinnä siitä, että monia pelin ominaisuuksia varten piti kirjoittaa huomattavasti enemmän koodia kuin olin alun perin arvioinut. Projekti sujui melko tasaista vauhtia enkä jäänyt jumittamaan mihinkään tiettyyn kohtaan kauaksi aikaa. Toteutusjärjestykseen jouduin myös tekemään joitain muutoksia, koska alkuperäinen järjestys ei jokaisessa vaiheessa tuntunut kovinkaan luontevalta. Järjestys oli kuitenkin pääosin sama kuin suunnitelmassa.

Työjärjestys ja aikataulu

Toteutusjärjestys

- Alustava käyttöliittymä ja kartan piirtäminen (29.2-1.3)
- Kartan ja maastojen lukeminen tiedostosta (1.3-3.3)
- Polunhaku (5.3-6.3)
- Yksiköiden toteutus (6.3-26.3)
- Käyttöliittymän parantelu (24.3-29.3)
- Yksikkötestausta (29.3-30.3)
- Tekoäly (4.4-13.3)

- Kenttäeditori (14.4-17.4)
- Tiedostojen lukemisen parantelua (17.4-20.4)
- Yksikkötestausta (21.4-22.4)
- Pelitilanteen tallennus (23.4-26.4)
- Viimeistelyä (27.4-30.4)

Projektin työjärjestys poikkesi muutamalla olennaisella tavalla suunnitelmasta. Päätin toteuttaa kartan, maastojen ja yksiköiden lukemisen projektin aikaisessa vaiheessa, jotta minun ei tarvitsisi koodata väliaikaista rakennelmaa näiden tietojen säilömiseen. Lisäksi käyttöliittymän toteutus eteni sitä mukaa, kun projektin muut ominaisuudet sitä vaativat. Tämä oli hyvin luonteva tapa toteuttaa käyttöliittymä, koska kussakin vaiheessa oli selkeä tieto siitä, mitä käyttöliittymän tulisi tehdä. Lisäksi toteutin kenttäeditorin ennen tallennusta, jotta saisin tehtyä peliin helposti lisää kenttiä. Muilta osin toteutusjärjestys noudatti pääpiirteittäin suunnitelmaa.

Arvio lopputuloksesta

Loppujen lopuksi projekti onnistui hyvin. Ohjelman hyviä puolia ovat sen monipuolisuus, toiminta ja ominaisuuksien määrä. Kaikki ohjelman osat toimivat hyvin ja siinä on monipuolisesti ominaisuuksia ja pelattavuuskin on suhteellisen hyvä. Ohjelman keskeisimmät puutteet ovat käyttöliittymän pienet puutteet, kuten pelilokin ulkoasu sekä tekoälyalgoritmin tehokkuus ja toiminta joissain tilanteissa. Käyttöliittymän puutteet johtuvat lähinnä siitä, että minulla ei ollut yksinkertaisesti tarpeeksi aikaa panostaa ominaisuuksien hiomiseen. Tekoälyn puutteet johtuvat osittain ajan puutteesta ja osittain siitä, että tämä oli ensimmäinen kerta, kun koodasin mitään tämän kaltaista. Tämän huomioon ottaen tekoälystä tuli yllättävän hyvä. Projektiin tuli käytettyä ehkä liikaakin aikaa ja joitain ominaisuuksia olisi voinut ihan hyvin jättää pois tai ainakin yksinkertaistaa.

Ohjelmaa voisi tulevaisuudessa parantaa keskittymällä edellä mainittujen puutteiden korjaamiseen. Lisäksi ohjelmaan voisi vielä lisätä suunnitelmassa mainitun yksiköiden valitsemisen ja asettelun ja asetustiedostojen määrää olisi myös mahdollista lisätä. Käyttämäni ratkaisumenetelmät ja tietorakenteet toimivat hyvin enkä todennäköisesti muuttaisi niitä merkittävästi. Luokkajakokin onnistui hyvin, mutta joitain luokkia olisi voinut ehkä jakaa pienemmiksi luokiksi. Esimerkiksi kenttäeditorin käyttöliittymän ja muun toiminnallisuuden olisi voinut jakaa kahteen eri luokkaan. Muuten en näe luokkajaossa merkittäviä ongelmia. Ohjelman koodin laatu on mielestäni hyvä ja esimerkiksi luokkien rajapinnat ovat selkeästi määriteltyjä, mutta joihinkin pikkuasioihin olisi voinut ehkä kiinnittää enemmän huomiota. Esimerkiksi tekoälyn toteutuksessa on jonkin verran toisteisuutta, jonka pystyisi poistamaan. Ohjelman laajentamisen ja muokkaamisen pitäisi olla melko suoraviivaista, koska ohjelman olemassa olevat ominaisuudet mahdollistavat monien asioiden helpon lisäyksen tai muokkauksen. Esimerkiksi uudet yksikkötyypit voisivat pitkälti käyttää samoja metodeja kuin vanhat ja tekoälyäkään tuskin joutuisi merkittävästi muokkaamaan tai laajentamaan.

Liitteet

Liitteenä on kuvia ohjelman käytöstä kuvat-kansiossa ja pelimekaniikat.txt-tiedosto, jossa on lyhyt kuvaus keskeisistä pelimekaniikoista.

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/

http://zetcode.com/gui/pyqt5/

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

https://realpython.com/working-with-files-in-python/#pythons-with-open-as-pattern

https://stackoverflow.com/questions/2397141/how-to-initialize-a-two-dimensional-array-in-python

https://www.geeksforgeeks.org/priority-queue-set-1-introduction/

https://stackoverflow.com/questions/510972/getting-the-class-name-of-an-instance

https://www.journaldev.com/15797/python-time-sleep

https://www.programiz.com/python-programming/property

https://stackoverflow.com/questions/1641219/does-python-have-private-variables-in-classes

https://www.geeksforgeeks.org/private-variables-python/

https://www.geeksforgeeks.org/private-methods-in-python/?ref=rp

https://stackoverflow.com/questions/46656634/pyqt5-qtimer-count-until-specific-seconds

https://stackoverflow.com/questions/6784084/how-to-pass-arguments-to-functions-by-the-click-of-button-in-pyqt

https://www.geeksforgeeks.org/default-arguments-in-python/

https://stackoverflow.com/questions/41545300/equivalent-to-time-sleep

https://stackoverflow.com/questions/613183/how-do-i-sort-a-dictionary-by-value

https://stackoverflow.com/questions/53225320/open-a-new-window-when-the-button-is-clicked-pyqt5

https://pythonspot.com/pyqt5-textbox-example/

https://stackoverflow.com/questions/27188538/how-to-delete-qgraphicsitem-properly

https://www.guru99.com/reading-and-writing-files-in-python.html

https://stackoverflow.com/questions/30362391/how-do-you-find-the-first-key-in-a-dictionary

https://stackoverflow.com/questions/6825994/check-if-a-file-is-open-in-python

https://stackoverflow.com/questions/2632205/how-to-count-the-number-of-files-in-a-directory-using-python