**Y2 Strategiapeli – dokumentaatio 1.5.2020**

**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 (ja näin hyökkäyksen)
* 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

**Käyttöohje**

**Ohjelma käynnistetään ajamalla main.py-tiedosto komentoriviltä tai IDE:stä käsin. Ohjelma avautuu päävalikkoon. Päävalikon kautta pääsee jatkamaan tallennettua peliä (jos sellainen on), valitsemaan pelattavaa kenttää tai kenttäeditoriin. (selosta tämä readme-tiedostossa)**

**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 nimi 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 tulee 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 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:

**Kuva, joka sisältää kohteen teksti

Kuvaus luotu automaattisesti**

**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 käyttää 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 kohderuudunn 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 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 hallitsevien metodien toiminta riippuu valitusta yksiköstä.**

**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.**

**Yksiköt-luokka on 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. Yksikkögrafiikka 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 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 tekoälyn ohjaimen valitsemaa 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 toteutuksessa on, että koodiin kirjoitettavista lauseista tulee usein melko pitkiä.**

**Algoritmit**

-polunhaku (A\*)

-näkyvyys

-tekoäly (yleiset, mainitse lisäksi, että on yksiköille ominaisia)

-vahingon laskenta

-muuta?

**Algoritmit**

Sanallinen kuvaus käyttämistänne algoritmeista, eli siitä miten ohjelma suorittaa tarvittavat tehtävät. Esim. miten tarvittava matemaattinen laskenta tapahtuu? (kaavat mukaan) Miten algoritminne löytää lyhimmän tiereitin kahden kaupungin välille? Miten toteuttamanne pelin tekoäly toimii? Kaavioita tms. voi käyttää apuna tarpeen mukaan. Mitä muita ratkaisuvaihtoehtoja olisi ollut? Perustelkaa valintanne: Verratkaa toteutusta johonkin toiseen ratkaisuun, ja selittäkää miksi päädyitte juuri tähän.

Tässä kohdassa on siis tarkoitus selostaa ne periaatteet, joilla ongelmat on ratkaistu, ei sitä, miten algoritmit koodataan. Siis ei luokkien tai metodien kuvauksia tai muitakaan Pythoniin tai ohjelmakoodiin liittyviä seikkoja tänne. Pseudokoodiesitys keskeisimmistä ei-tunnetuista algoritmeista on kuitenkin hyvä olla sanallisen kuvauksen tukena. **HUOM!** Jokaisessa työssä on aina algoritmeja, toiset ehkä yksinkertaisempia kuin toiset, moni aivan itse alusta saakka keksittyjä. Kuvaa tässä niistä muutama kaikkein olennaisin.

**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 kuin pelin aikana 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 korvaavan tietorakenteen. Lisäksi tilavaikutukset ja yksikön ominaisuudet on sisällytetty kummatkin omaan tietorakenteeseensa. Tilavaikutukset-rakenne 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.**

**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 käsin testaamalla 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 jopa suorastaan typerää. Tietokoneen yksiköt saattavat esimerkiksi jäädä johonkin ruutuun, jota ne pitävät hyvänä puolustusasemana ja antaa pelaajan tykistön tuhota ne helposti. Tämän ongelman olisi varmaankin voinut ainakin osittain korjata muokkaamalla 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 muutaman sadan millisekunnin viiveisiin. Tilanteen korjaaminen todennäköisesti vaatisi koko algoritmin uudelleen kirjoittamisen, mutta yksinkertaistamalla algoritmia voisi jo päästä suhteellisen hyviin tuloksiin. Pelattavuuden kannalta suurin ongelma on se, että tekoälyn päätöksenteon aiheuttaman viiveen lisäksi on 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.**

**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. 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.

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 100-120 tuntia. 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 rakentaa 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**

**Arvio lopputuloksesta**

“Yhteenveto” ja itsearviointi joka voi toistaa yllämainittujakin asioita.

Arvioikaa ohjelman laatua, kertokaa sen hyvistä ja huonoista puolista. Onko työssä oleellisia puutteita ja mistä ne johtuvat (mahdollinen hyvä perustelu dokumentissa voi korvata pienet puutteet)? Miten ohjelmaa olisi voinut tai voisi tulevaisuudessa parantaa? Olisiko ratkaisumenetelmien, tietorakenteiden tai luokkajaon valinnan voinut tehdä paremmin? Soveltuuko ohjelman rakenne muutosten tai laajennusten tekemiseen? Miksi tai miksi ei?

**Liitteet**

**Liitteenä 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