CS-A1121 Ohjelmoinnin peruskurssi Y2

Loppuraportti

Strategiapeli

Arno Alho

424628

ELEC / Space science and technology

27.02.2018

Sisällys

1. Yleiskuvaus

Tämä raportti kuvaa vuoropohjaisen strategiapelin toteutuksen tekoälykkäällä tietokonevastustajalla. Peli sijoittuu neliöpohjaiseen ruudukkoon, jossa pelaaja komentaa joukkoa erilaisia hahmoja, joilla on erilaisia vuoron aikana suoritettavia toimintoja. Peliin on mahdollista lisätä kenttiä, pelaajia sekä pelaajien toimintoja ulkoisten tiedostojen avulla. Tämä ohjelma on suunniteltu toteutettvaksi vaikealla vaikeusasteella.

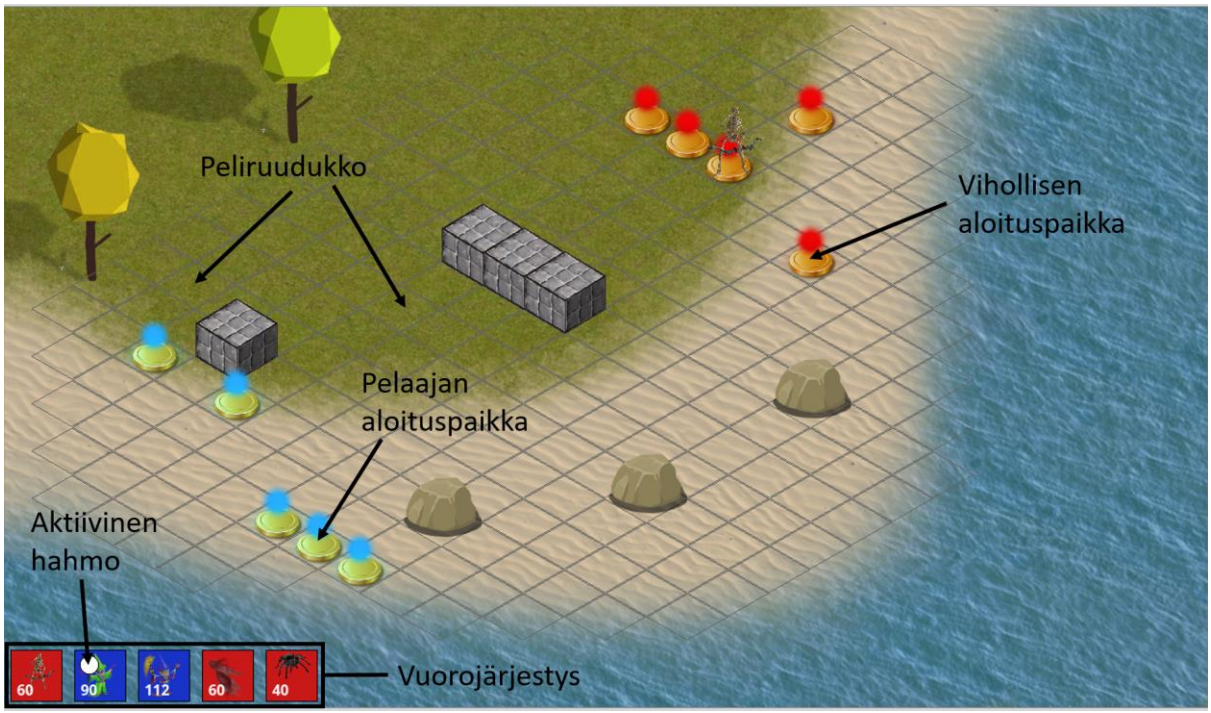
2. Käyttöohje

Pelin tarkat käyttöohjeet on selostettu README.md-tiedostossa, mutta tämä kappale esittelee lyhyesti, millaisia toimintoja pelaajalla on valittavanaan. Lisäksi tämä kappale esittelee monia pelin ominaisuuksia kuvina, joista saattaa olla lisäapua selventämään README:n epäselvyyksiä.

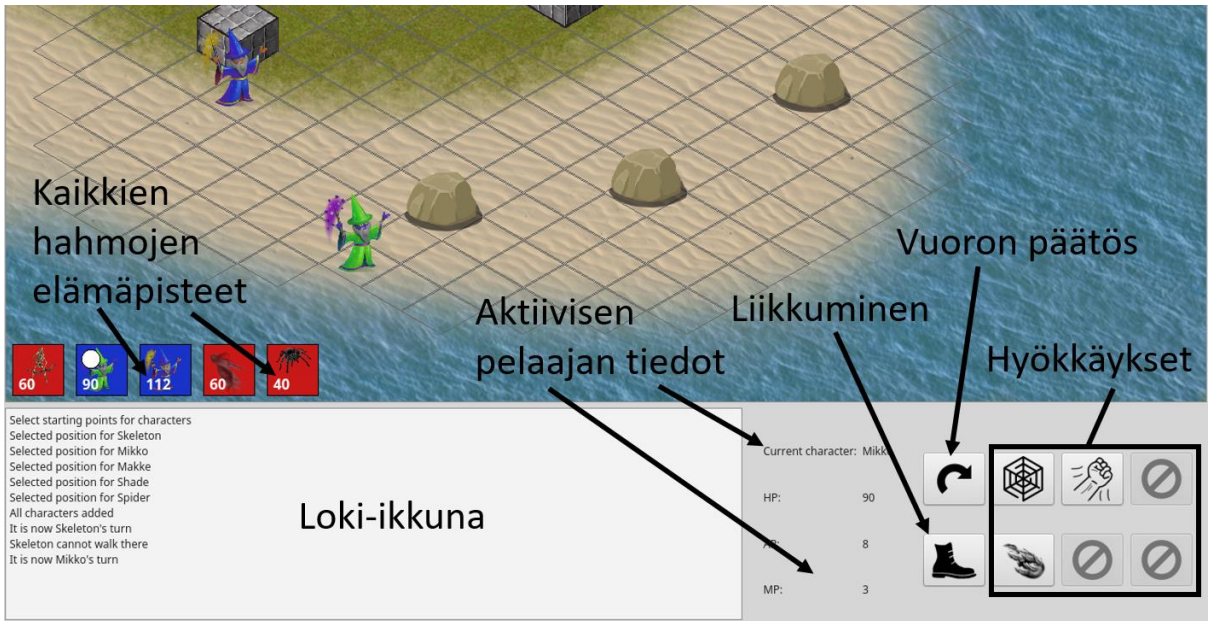
Peli käynnistetään ajamalla komentoriviltä main.py-ohjelma, minkä jälkeen kaikki pelin toiminnot tapahtuvat graafisen käyttöliittymän kautta. Heti komennon ajamisen jälkeen käyttäjälle avautuu dialogi-ikkuna, josta hän voi valita haluamansa pelin. Kuva 1 näyttää dialogi-ikkunan käynnistämisen jälkeen.

Kuva 1. Dialogi-ikkuna pelin valitsemiseksi

Pelin valittuaan käyttäjän eteen avautuu graafinen ikkuna, jossa hän voi valita pelaajiensa aloituspaikat. Kuva 2 on ruudunkaappaus aloituspaikkojen valinnasta. Kaikkien hahmojen valittua omat aloituspaikkansa, peli alkaa. Pelissä jokainen hahmo pelaa vuoron kerrallaan ja tavoitteena on tappaa kaikki vastapuolen hahmot. Peliruudun alareunassa on pelin hallinnan painikkeet sekä informaatiota pelin tilanteesta. Kuva 3 näyttää peliruudun keskeisimmät elementit. Hahmon hyökkäyksistä voi saada lisäinformaatiota osoittamalla painiketta hiirellä, jolloin kursorin kohdalle ilmestyy infoikkuna, joka sisältää hyökkäyksen tiedot.



Kuva 2. Aloituspaikan valinta



Kuva 3. Pelin hallintapainikkeet

3. Ulkoiset kirjastot

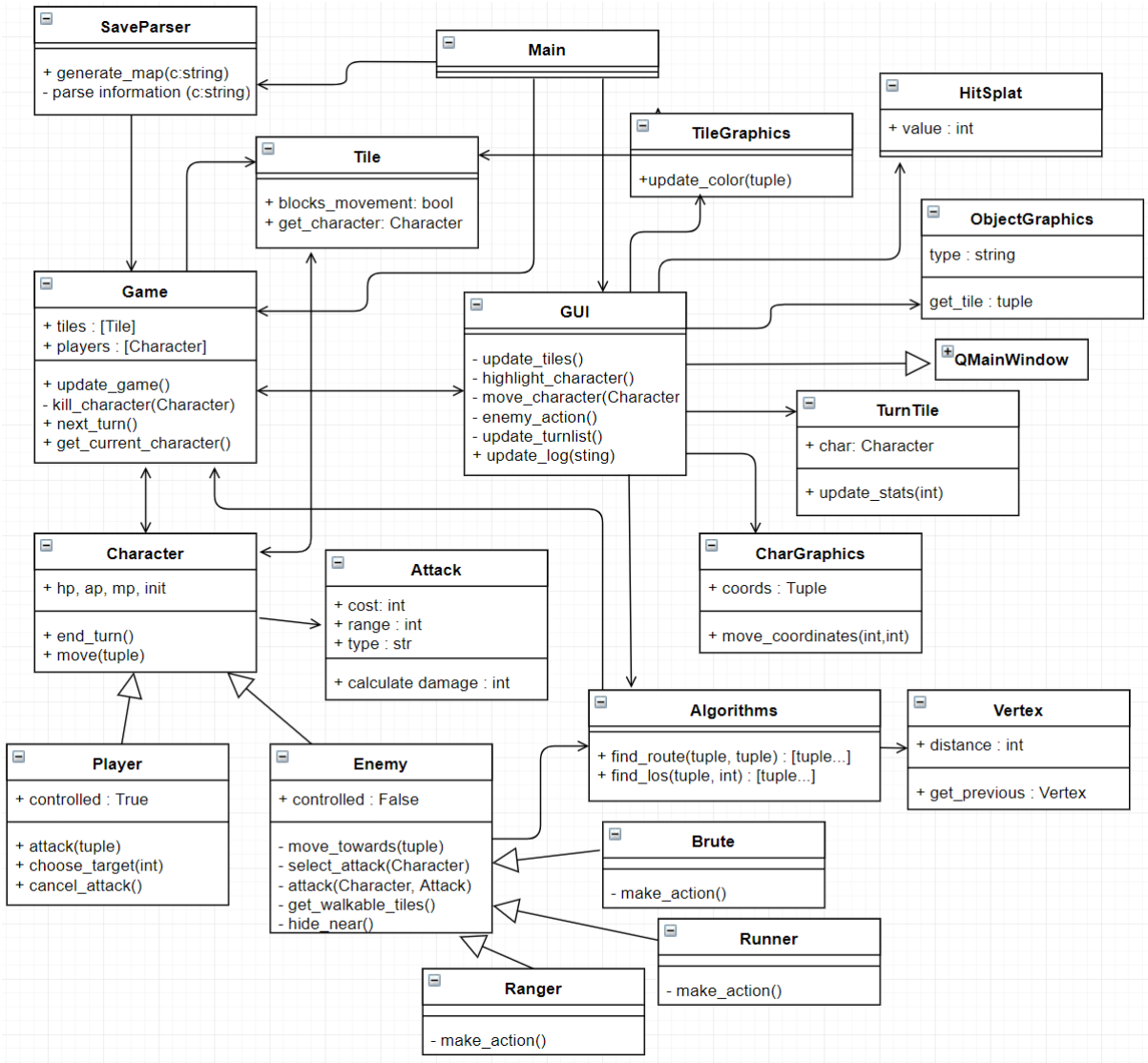
Ohjelmassa ei ole käytetty lainkaan pythonin ulkopuolisia kirjastoja. Alkuperäisessä suunnitelmassa pelin karttojen lukemiseen käytettiin PIL-kirjastoa, mutta ohjelman valmiissa versiossa on itse kirjoitettu bitmap-kuvien tulkki. Ohjelma käyttää lisäksi joissakin osin pythonin random-kirjastoa, jonka tarkoituksena on tuoda peliin hieman satunnaisuutta. Kirjaston funktioita ei kuitenkaan ole käytössä vihollisten tekoälyn osissa, sillä tämä oli erillinen tehtävänannon vaatimus. Lopuksi, ohjelman kaikki graafiset osat perustuvat PyQt-käyttöliittymäkirjastoon, jota on käytetty useammassa ohjelman luokassa.

4. Ohjelman rakenne

4.1 Ohjelman yleisrakenne

Kuvassa 4 on esitetty karkea UML kaavio ohjelman rakenteesta.

Ohjelma jakautuu selkeästi graafisiin luokkiin kuten GUI, CharGraphics ja ObjectGraphics sekä graafisten olioiden taustalla peliä pyörittäviin luokkiin kuten Game, Character ja Tile. Näiden luokkien on kyettävä keskustelemaan tehokkaasti keskenään, sillä pelaajan komennot annetaan kaikki graafisille olioille, mutta komennot on käsiteltävä varsinaisissa peliluokissa. Tämän jälkeen peliluokkien tulokset on uudelleen näytettävä käyttäjälle graafisissa luokissa. Kuten kuvastakin saattaa huomata, tämän kommunikaation ytimessä on luokkien Game ha GUI välinen linkki, joka muodostaa kommunikaation selkärangan.



Kuva 4. Ohjelman rakennekaavio

4.2 Keskeisten luokkien ja metodien kuvaukset

Game-luokka vastaa itse pelin pyörittämisestä kuten vuorojen vaihtumisesta, pelilaudan tilanteen seuraamisesta sekä pelin päättymisen tarkastamisesta. Game-luokka tarkistaa pelin päättymisestä update_game -metodilla, joka samalla myös poistaa pelilaudalta kuolleet pelaajat. Peli vaihtaa vuoroa next_turn -metodilla, jota kutsutaan GUI-luokasta vuoron päättyessä.

GUI-luokka vastaa graafisen käyttöliittymän piirtämisestä sekä pelaajan komentojen vastaanottamisesta. Luokka käyttää algoritmiluokkaa pelaajan kävelyreittien sekä hyökkäysten mahdollisten kohderuutujen korostamiseen. Lisäksi luokka käyttää useita muita graafisia objekteja pelaajien, pelikentän ruutujen sekä muiden esineiden piirtämiseen. GUI -luokka päivittää peliruutua 60 kertaa sekunnissa, jolloin se päivittää ruutujen korostamisen sekä graafisten objektien piirtojärjestyksen.

GUI-luokka on myös valitettavan sitoutunut vihollisten tekoälyn ohjaamiseen. Tämä johtuu siitä, että vihollisten toimintaa on haluttu hidastaa, jotta se olisi pelaajalle helpommin seurattavaa. Tämän toteuttaminen vaatii ajastimien käyttöä, jotka on toteutettu GUI-luokassa.

Character-luokka sisältää hahmon perustiedot ja sillä on kaksi eri aliluokkaa: ihmisen hallitsema Player sekä tietokoneen hallitsema Enemy. Ainoa näille aliluokille yhteinen metodi on hahmon liikuttaminen. Muut vuoron kulkuun ja hyökkäämiseen tarkoitetut metodit on toteutettu aliluokissa erikseen.

Enemy-luokka käyttää algoritmitiedoston funktioita ja luo niiden perusteella korkeamman tason metodeja, joita sen aliluokat käyttävät vuoron aikana päätösten tekemiseen. Kaikki viholliset ovat joitakin Enemy-luokan aliluokkia, jotka käyttävät näitä algoritmeja eri tavoin tuottaakseen eri lailla käyttäytyviä vihollisia. Tässä luokassa on myös toteutettu vihollisten hyökkäysmetodit, jotka ovat samat kaikille aliluokille.

SaveParser-luokka lukee pelaajan antaman tallennustiedoston ja luo sen perusteella game-olion. Tämä luokka käyttää kuvassa 4 merkitsemätöntä image-luokkaa, joka tulkitsee bitmap-kuvia pelikentiksi.

5. Algoritmit

5.1 Dijkstan algoritmi

Pelissä käytetään Dijkstran algoritmia hahmojen kulkemien reittien valitsemiseksi. Algoritmi etsii lyhyimmän reitin kahden pisteen välille, kun kentällä on mahdollisia reitillä olevia esteitä. Dijkstran algoritmia käytetään pelissä sekä helpottamaan pelaajien käyttökokemusta, että vihollisten tekoälyn apuna. Sen avulla käyttäjä voi suoraan valita ruudun, johon pelaaja liikkuu, jolloin ohjelma näyttää käyttäjälle lyhyimmän reitin kyseiseen pisteeseen. Vihollisen tekoäly käyttää samaa algoritmia löytääkseen reitin joko suoraan pelaajan luo tai paikkaan, josta se kykenee hyökkäämään pelaajan kimppuun. Kuva 5 näyttää algoritmin ehdottaman reitin pelaajalle.

Algoritmi toimii jakamalla kentän useisiin soluihin, joista jokainen tietää oman etäisyytensä aloituspisteeseen. Algoritmi on toteutettu Wikipedian pseudokoodin perusteella [1]:

1. Merkitse aluksi kaikki kentän ruudut vierailemattomaksi sekä etäisyydet aloitusruutuun äärettömäksi.
2. Tarkastele ruutuja yksi kerrallaan lähtien valitusta aloitusruudusta. Jokaiselle tarkasteltavan ruudun vierailemattomalle naapurille lasketaan etäisyys aloituspisteeseen lisäämällä

alkuperäisen ruudun etäisyys tarkasteltavan ruudun etäisyyteen. Jos etäisyys on pienempi kuin ruudun sen hetkinen etäisyys, korvataan solun etäisyys lyhyemmällä arvolla.

2.1 Tämän ohjelman ratkaisussa kaikkien ruutujen keskeneräiset etäisyydet ovat samat ja siten jokaisen ruudun etäisyys naapuriinsa on 1.

3. Kun kaikki tarkasteltavan ruudun naapurit on käyty läpi, merkitse tarkasteltava ruutu vierailluksi.
4. Siirry ruutuun, jolla on sillä hetkellä lyhyin etäisyys aloituspisteeseen.
5. Jatka prosessia, kunnes lopetuspiste on merkitty vierailluksi.

Kuva 5. Dijkstran algoritmin ehdottama reitti vihreälle pelaajalle

5.2 Bresenhamin algoritmi

Toinen tärkeä algoritmi, jota käytetään näkyvyyden testaamiseen, on Bresenhamin viiva-algoritmi. Algoritmi approksimoi joukon ruutuja, jotka parhaiten asettuvat kahden päätepisteen määräämälle suoralle. Bresenhamin viiva-algoritmia käytetään näkyvyyden testaamiseen piirtämällä viiva jokaiseen ruutuun, johon pelaajan hyökkäys yltää. Jos viivan määrittämässä ruutulistassa on yksikin ruutu, joka on määritelty esteeksi, ei alun perin tarkasteltava ruutu ole näkyvissä. Näkyvyysalgoritmin toiminta on nähtävissä kuvassa 6.

Algoritmin toiminta toteutetaan seuraamalla suoran kulmakerrointa. Algoritmin kaavat ovat hieman erilaiset riippuen x ja y muutosten etumerkeistä sekä kokoerosta, minkä takia itse ohjelma joutuu tarkastelemaan näistä muuttujista syntyvät kahdeksan eri tapausta erikseen. Tarkastellaan esimerkiksi tilannetta, jossa:

$$\Delta x = x_1 - x_0 > 0, \qquad \Delta y = y_1 - y_0 > 0, \qquad \Delta x > \Delta y$$

Koska x:n muutos on suurempi, kuin y:n muutos, täytyy suoran x-komponentin kasvaa yhdellä joka askeleella. Täten algoritmi laskeekin, kuinka paljon y:n arvo muuttuu joka askeleella ja pyöristää sen lähimpään kokonaislukuun. Käytännössä algoritmi toimii pseudokoodina seuraavasti [2]:

Virhetekijä $e = 0$

kulmakerroin $\frac{\Delta x}{\Delta y}$

Jokaiselle x arvolle välillä $x_0 \longrightarrow x_1$:

$e += \frac{\Delta x}{\Delta y}$

Jos $e > 0.5$:

$y += 1$

$e -= 1$

Lisää listaan ruutu (x, y)

Kuva 6. Vihreän pelaajan näkemät ruudut korostettu sinisellä

5.3 Vihollisten tekoäly

Näiden kahden perusalgoritmin lisäksi tietokonevastustajilla on oltava korkeamman tason tekoäly, joka käyttää edellä mainittuja algoritmeja toimintojen valintaan. Nämä korkeamman tason algoritmit on määritelty Enemy-luokassa. Ohjelmassa on määritelty Enemy-luokalle alaluokkia, jotka käyttävät näitä algoritmeja eri tavalla. Tällä hetkellä pelissä olevia luokkia ovat:

- Brute: Pyrkii aina lyömäetäisyydelle ja lyö niin monta kertaa kuin pystyy.

- Runner: Pyrkii lyömäetäisyydelle, jonka jälkeen juoksee mahdollisimman kauas karkuun. Osaa mahdollisesti käyttää myös kaukoiskuja, jos hahmolla on niitä käytössä.

- Ranger: Pyrkii osumaan pelaajiin mahdollisimman kaukaa, jonka jälkeen hakautuu lähimpään piiloon.

Kaikki nämä koneälyt käyttävät samoja Enemy-luokan metodeja, mutta Ranger on aliluokista monimutkaisin, joten esittelemällä tarkemmin sen toimintaa, saa myös kuvan muiden tekoälyjen toiminnasta. Jokaisella vuorolla Ranger toimii seuraavasti:

1. Valitse lähin pelaaja kohteeksi
    a. Käy kaikki pelaajat läpi ja mittaa käveltävä etäisyys jokaiseen.
    b. Valitse pelaaja, johon on lyhyin etäisyys.
2. Etsi lähin piste, josta voi hyökätä valittuun pelaajaan
    a. Selvitä suurin hyökkäyksen kantama kaikista hyökkäyksistä
    b. Selvitä kaikki pisteet, joihin hahmo yltää kävelemään
    c. Etsi pisteistä lähin, josta on näköyhteys pelaajaan
    d. Kävele valittuun pisteeseen
    e. Jos pistettä ei ole, kävele pelaajaa kohti kaikilla liikkumispisteillä
3. Hyökkää pelaajaan
    a. Valitse hyökkäyksen kohde
        i. Valitse näkyvillä olevista pelaajista se, jolla on vähiten elämäpisteitä (tämä kohde voi olla eri kuin kävelyn kohde, mutta tämän askeleen lisääminen tekoälyyn teki vihollisesta huomattavasti kiinnostavamman)
    b. Valitse hyökkäys siten, että pelaaja menettää mahdollisimman paljon elämäpisteitä
    c. Hyökkää pelaajaan niin monta kertaa, kun hahmolla riittää toimintapisteitä
        i. Hahmo saattaa vaihtaa hyökkäystä, jos toinen hyökkäys vaatii vähemmän pisteitä
4. Pakene lähimpään paikkaan, johon mahdollisimman harva pelaaja pääsee hyökkäämään
    a. Selvitä kaikki pisteet, joihin hahmo yltää kävelemään
    b. Laske, kuinka moni pelaaja kykenee hyökkäämään kuhunkin pisteeseen
    c. Valitse pisteet, joihin harvin määrä pelaajista näkee
    d. Valitse näistä pisteistä se, johon on lyhyin matka
    e. Kävele valittuun pisteeseen

6. Tietorakenteet

Pelin sisällä suurin osa tiedosta on tallennettu käyttämällä pythonin tarjoamia valmiita tietorakenteita. Suurin osa laajemmista tiedoista kuten esimerkiksi tieto kaikista ruuduista tai pelaajista on tallennettu listoihin. Esimerkiksi pelaajien määrät voivat muuttua pelin varrella, ja listat tarjoavat tarvittavat työkalut pelaajien käsittelyyn.

Ohjelmassa käytetään lisäksi runsaasti monikoita merkitsemään sekä RGB-väriarvoja että koordinaatteja. Päätin olla luomatta koordinaateille omaa luokkaa ohjelmakoodin yksinkertaistamiseksi, sillä kaikki koordinaattiluokan tärkeimmät metodit on helppo toteuttaa Tile-luokassa.

Algoritmien yhteydessä on käytetty linkitettyjä listoja, sillä esimerkiksi Dijkstran algoritmi ei käsittele pelikentän ruutuja selvässä järjestyksessä. Algoritmin toimintatavasta johtuen on merkittävästi helpompaa linkittää lyhyimmän reitin omaava ruutu aina seuraavan ruutuun. Tätä varten ohjelmassa on luotu oma luokka Vertex, jota käytetään ainoastaan algoritmien laskennassa.

Ohjelmaan on myös luotu useita eri luokkia tiedon tallentamiseksi. Suurin osa näistä on erilaisia graafisia luokkia, joiden täytyy sisältää tieto omasta kuvakkeestaan sekä informaatiota esimerkiksi pelaajasta, joita ne esittävät. Monet näistä luokista on luotu joko QGraphicsPixmapItemin tai QGraphicsItemGroupin aliluokkina. Lähes kaikki tieto näissä luokissa on muuttumatonta, sillä graafisen objektin lähes kaikki parametrit pysyvät vakiona yhden pelin aikana. Jos peliin olisi luotu kattavampia animaatioita, tämä lähestymistapa ei tulisi toimimaan.

7. Tiedostot

Pelin eri skenaariot sekä pelaajien toimintojen tiedot on tallennettu helposti luettaviin tekstitiedostoihin. Tämä tiedostomalli on valittu, jotta tiedostoista on helppo lukea kunkin pelin tilanne sekä pelaajien toimintojen vaikutukset ymmärtämättä itse koodin toimintaa. Tekstitiedostoja on lisäksi nopea kirjoittaa, jolloin peliä on helppo laajentaa jälkikäteen.

Skenaariotiedostossa oleva informaatio on jaettu alla esitettäviin lohkoihin. Jokaisen hahmolohkon on sisällettävä tietty määrä informaatiota tai muuten peli antaa virheilmoituksen. Pelin saves-kansiossa on useita esimerkkejä skenaariotiedostoista.

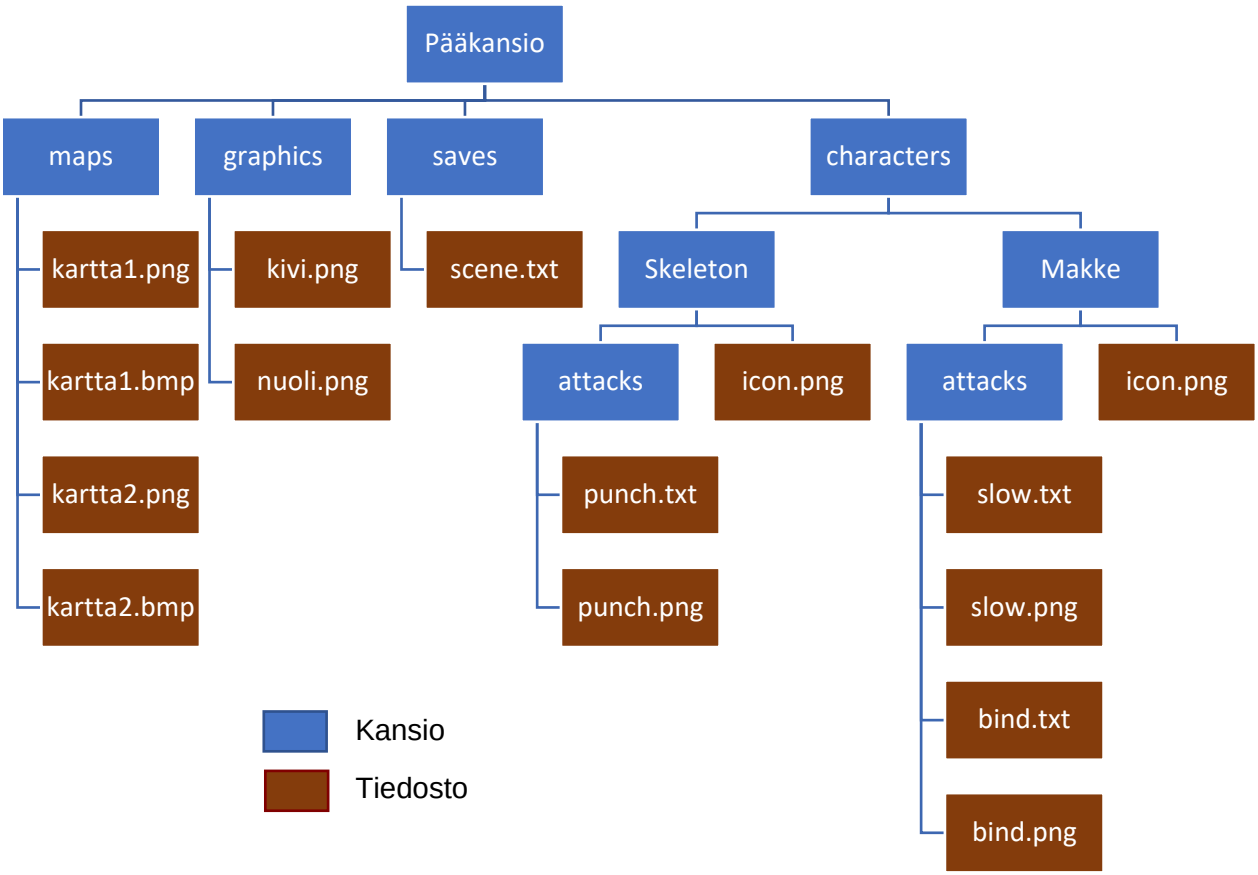
[HEADER]
#
[BLOCK NAME] (: [AI if block is enemy])
[PARAM 1] : [VALUE 1]
[PARAM 2] : [VALUE 1]: [VALUE 2]: [VALUE 3]
/#

Pelaajien hyökkäykset on kukin tallennettu erillisiin tiedostoihinsa, jotka noudattavat samankaltaista rakennetta kuin skenaariotiedostot. Tämän lisäksi hyökkäystiedoston kanssa samaan kansioon voi tallentaa samannimisen .png kuvan, joka toimii kuvakkeena hyökkäykselle. Tämä rakenne yksinkertaistaa huomattavasti eri skenaarioiden luomista ja mahdollistaa saman vihollisen uudelleenkäyttämisen hieman eri asetuksilla. Tämä voi olla hyödyllistä, jos pelissä tarvitaan samasta vihollisesta useampi hieman eritasoinen versio, joilla on kuitenkin samat hyökkäykset käytössään. Lisäksi vihollisten käyttäytyminen valitaan skenaariotiedostossa, mikä antaa vielä enemmän mahdollisuuksia saman vihollisen monipuolistamiseksi.

Pelikentät on tallennettu kuvaformaattiin, jossa jokainen pikseli vastaa yhtä pelikentän ruutua. Kentät tallennetaan kuviin, koska kenttien luominen kuvankäsittelyohjelmilla on huomattavasti tekstitiedostoja helpompaa. Kuvista on helppo nähdä pelin sisältö, mikä helpottaa peliskenaarioiden suunnittelua, sillä peliä ei tarvitse käynnistää, jotta näkisi suunnitelmansa tuloksen. Pelikenttä on tallennettava bitmap-muotoon, jotta ohjelma osaa tulkita sen sisällön. Tämän lisäksi karttakansioon voi lisäksi tallentaa saman nimisen tiedoston .png muotoon, jolloin peli käyttää kyseistä tiedostoa taustakuvana kartalle.

Lopuksi, kuvaformaatit myös tarjoavat lähes rajattomasti laajentumisvaraa pelille, sillä jokaiselle pikselin RGB-arvolle voidaan määrittää oma merkityksensä pelissä. Haasteena tässä lähestymistavassa on muistaa pikselivärien merkitykset, minkä tähden ne on hyvä dokumentoida esimerkiksi README.md tiedostoon.

Pelin kansiorakenne on esitetty kuvassa 7. Kansiorakenteen tavoitteena on, että muokkaamalla ainoistaan tiedostoja saves-kansiossa, on mahdollista luoda mahdollisimman paljon erilaisia pelitilanteita.



Kuva 7. Ohjelman kansiorakenne

8. Testaus

Ohjelman testaus tapahtui hyvin pitkälti järjestelmätasolla, mutta muutamia projektin osia yksikkötestattiin myös. Ohjelman testaus tapahtui ajamalla ohjelmaa usein ja tarkastamalla koodiin tehtyjen muutosten vaikutus. Jatkuvalla ja tiheään tapahtuvalla ohjelman suorittamisella koodissa syntyneet virheet oli mahdollista havaita nopeasti ja niiden korjaaminen oli tehokasta.

Ohjelman rakentamisen aikataulu oli myös suunniteltu siten, että ohjelman testaus olisi mahdollisimman tehokasta. Toteutin ohjelman UI:n lähes ensimmäisenä, jolloin ohjelmaa oli helppo ajaa ja eri UI elementteihin pystyi lisäämään erilaisia debug-toimintoja, joilla pystyi tarkastelemaan muuten hankalasti saavutettavia reunatapauksia. Näillä komennoilla pystyin esimerkiksi varmistamaan, että ohjelma ei kaadu tilanteessa, jossa viimeinen vihollinen tappaa itsensä. Vaikka kyseinen tilanne ei olekaan mahdollinen pelin tämänhetkisessä tilassa, saattaisi se olla mahdollista, jos peliin lisäisi tulevaisuudessa lisäominaisuuksia.

Yksi hankalimpia testattavia asioita oli vihollisten tekoälyn toiminta. Tekoälyjen toimintaa oli suhteellisen helppo seurata tilanteissa, jossa on yksi pelaaja ja yksi vihollinen, sillä silloin on mahdollista päätellä itse, mitä vihollisen tulisi oman suunnitelman mukaan tehdä. Kuitenkin tilanteissa, joissa sekä pelaajia että vihollisia on useampia, vastaavan analyyttisen ratkaisun selvittäminen on huomattavasti haastavampaa ja siten näissä tilanteissa keskityin enemmän subjektiiviseen analyysiin arvioimalla, kuinka mielekästä kyseistä vihollista vastaan on pelata.

Alkuperäisen suunnitelman yksikkötestausosio oli hyvin kunnianhimoinen, enkä kyennyt yksikkötestaamaan tekoälyjen toimintaa lainkaan. Tästä huolimatta suoritin kuitenkin joitakin yksikkötestejä Bresenhamin viiva-algoritmille sekä SaveParser-luokalle.

Yksikkötestit olivat erittäin tehokas tapa varmistaa viiva-algoritmin toiminta, sillä algoritmilla on kyettävä käsittelemään kahdeksan erilaista tapausta ja niiden kokeileminen käsin veisi merkittävästi aikaa. Lisäksi kahdeksan eri tapauksen erisuuruuksissa ja etumerkeissä on helppo tehdä huolimattomuusvirheitä, jotka eivät välttämättä näy suoraan peliä pelatessa. Yksikkötestit olivat siten erittäin tehokas tapa valmistaa algoritmin toiminta.

SaveParser luokan yksikkötestit varmistavat, että luokka antaa oikeanlaisen varoituksen, jos käyttäjä yrittää avata korruptoitunutta skenaariotiedostoa. Yksikkötestien avulla oli helppo kokeilla kaikki eri mahdolliset tavat, jolla tiedosto saattaisi olla viallinen.

9. Tunnetut puutteet ja viat

Tällä hetkellä pelaajien hyökkäykset ovat suhteellisen yksinkertaisia. Ohjelmassa on joitakin rakenteita jo valmiina, joilla esimerkiksi hyökkäysten mahdollisia kohderuutuja voisi muuttaa. Tällä hetkellä kaikkien hyökkäysten kantama on pelaajan ympärille piirretty ympyrä, mutta lineaariset tai esteitä ohittavat hyökkäykset tekisivät pelistä huomattavasti kiinnostavamman.

Toinen hyökkäyksiin liittyvä puute on niiden osuma-alueen yksinkertaisuus. Pelissä olisi hyvä olla hyökkäyksiä, jotka kykenisivät osumaan useampaan ruutuun kerralla. Tämä vaatisi kuitenkin melko merkittävää hyökkäysten tallennustiedoston muutosta.

Projektin edetessä, pelin kansiorakenne on monimutkaistunut merkittävästi, mikä vaikeuttaa uuden sisällön luomista peliä varten. Vaikka pelin siirtyminen isometriseen kuvakulmaan on tehnyt siitä miellyttävämmän näköisen, se on myös hankaloittanut uusien skenaarioiden luomista. Toinen ominaisuus, joka poistui pelistä isometrisen kuvan myötä, oli pelin graafisen ikkunan helppo skaalaaminen. Ikkunan koon dynaaminen muuttaminen on yhä mahdollista, mutta sen toteutus on hieman hankalampaa enkä nähnyt sen tuomaa hyötyä siihen käytetyn ajan arvoiseksi.

Pelin suurin puute kuitenkin on pelien aloittamisessa ja lopettamisessa. Peli täytyy käynnistää joka kerta, että pelaaja voi pelata uuden pelin, ja yhden pelin päätyttyä ikkuna on suljettava, jotta voi aloittaa toisen pelin. Pelikokemusta voisi parantaa huomattavasti, jos ohjelma muistaisi käytyjä taisteluja ja pelissä voisi siirtyä suoraan taistelusta toiseen. Yksi syy, miksi tämä käyttökokemus on hieman heikko, onkin pelin mahdollinen laajentaminen. Minulla on tavoitteena mahdollisesti jatkaa tämän projektin parissa vielä eteenpäin luomalla taisteluiden ympärille eräänlaista roolipelityyppistä metapeliä. Tämän takia en ole halunnut luoda liian hienoja valikoita pelien välille siirtymiseen. Tämä sama projektin mahdollinen jatkaminen on myös osasyynä kansiorakenteen monimutkaistumiseen.

10. Ohjelman vahvuudet ja heikkoudet

Ensimmäinen ohjelman vahvuuksista on mielestäni sen ulkonäkö ja käytettävyys. Vaikka isometriseen näkökulmaan siirtyminen pakotti kirjoittamaan suuren osan koodista uusiksi sekä hieman vaikeutti uusien karttojen luomista, on sen tuoma graafinen muutos ongelmien arvoinen. Lisäksi mielestäni pelin käyttöliittymä on melko intuitiivinen ja nappien tarjoamat vihjeet auttavat pelaajia, jotka eivät ole varmoja niiden toiminnasta.

Toinen ohjelman vahvuus on ollut bitmap kuvan käyttö kartan tallennusmuotona. Kuvan käyttäminen on tehnyt karttojen luomisesta erittäin helppoa ja se mahdollista karttojen luomisen, vaikka niiden tekijä ei edes pääsisi käsiksi ohjelman lähdekoodiin. Lisäksi kuvien avulla karttaan voi lisätä lukemattomasti erilaisia esineitä. Esimerkiksi ohjelman mukana tulevissa esimerkeissä eriväriset pikselit vastaavat erilaisia kiviä tai seinän osia, jotka ovat kuitenkin peliteknisesti identtisiä.

Kolmantena ohjelmakoodin vahvuutena haluan mainita tekoälyä ohjaavien metodien toteutukset. Vaikka tekoälyt itsessään eivät ole kovin monimutkaisia, on niiden toteutus Enemy-luokan metodeina mielestäni hyvin jäsenneltyä ja selkeää. Tämän seurauksena itse tekoälyjen toteutus vie vain muutaman rivin Enemy-luokan aliluokissa. Toinen jäsentelyn puolesta puhuva havainto on, että pystyin palaamaan kyseisen ohjelman osan pariin pienen tauon jälkeen ja ymmärsin hyvin nopeasti, mitä olin aikaisemmin ajatellut.

Mahdollisesti ohjelman heikoin osuus on vuorojärjestystä ohjaavan koodin monimutkaisuus sekä GUI-luokan sekaantuminen lähes jokaiseen muuhun luokkaan. Vuorojärjestyksen monimutkaisuus on aiheuttanut itselleni paljon päänvaivaa ohjelman toteutuksen aikana, mutta silti en kyennyt vähentämään GUI-luokan osuutta siinä. Tämä johtuu osittain siitä, kuinka GUI luokka ohjaa vihollisia ohjaavia ajastimia, minkä takia luokassa on toteutettu enemmän pelin kulkuun liittyviä asioita kuin olisin alun perin halunnut. Toinen syy GUI-luokan paisumiselle on se, että pelaaja ei voi tehdä toimintoja silloin kun hahmon animaatio on käynnissä. Täten luokalla on myös vaikutusta pelaajan vuorolla tapahtuviin asioihin.

Toinen pelin heikkous on sen tekoälyn toiminnan yksinkertaisuus. Vaikka tekoälyt vastaavatkin alkuperäistä suunnitelmaa, tuntuu niiden toiminta kuitenkin melko yksinkertaiselta. Jos jatkan projektin parissa, saatan harkita tekoälyn toiminnan muuttamista täysin toisenlaiseksi käyttämällä esimerkiksi ulkoisia kirjastoja.

Kolmas pelin heikkous on sen pelitekninen osuus. Suurin osa tästä projektista on keskittynyt ohjelman teknisten osien ratkaisuun, minkä takia en ole käyttänyt juurikaan aikaa pelin pelattavuuden miettimiselle. Suurin osa karttojen rakenteista sekä hahmojen asetuksista on suurpiirteisiä arvauksia, minkä takia pelit ovat usein joko liian helppoja tai liian vaikeita.

11. Poikkeamat suunnitelmasta

Mahdollisesti suurin ero alkuperäiseen suunnitelmaan on pelien tallennusmahdollisuuden poistaminen. Kehittämisen aikana huomasin, että pelien kesto on liian lyhyt, jotta niiden tallentaminen kesken taistelun olisi tarpeellista tai mielekästä. Tämän sijaan päätin keskittyä enemmän siihen, kuinka taisteluiden jälkeiset vaikutukset voisivat vaikuttaa tuleviin taisteluihin. Vaikka tätä taistelusta toiseen siirtymistä ei voi tehdä projektin tässä vaiheessa, on ohjelman kansiorakenne ja hahmojen tallennetut tiedostomuodot sellaiset, että tämän toiminnon lisääminen on mahdollista.

Toinen poikkeama alkuperäisestä suunnitelmasta oli isometrisen näkökulman lisääminen. Tämä muutti monia graafisia osia ohjelmasta sekä poisti joitakin yksinkertaisen ruudukon tarjoamia ominaisuuksia, mutta samalla se teki pelistä paljon uskottavamman näköisen.

Lisäksi joitakin osia alkuperäisestä suunnitelmasta jäi puuttumaan kuten kyky iskeä useampaan ruutuun yhdellä kerralla sekä puutteet yksikkötestaussuunnitelman toteuttamisesta. Näitä molempia puutteita on kuvailtu tarkemmin kappaleissa 8 ja 9.

12. Työjärjestys ja aikataulu

Aloitin ohjelman kirjoittamisen jo hyvissä ajoin ja olin toteuttanut useita osia siitä jo ennen ensimmäisen suunnitelman kirjoittamista. Tämä helpotti lopun ohjelman kirjoittamista eikä minulle tullut missään vaiheessa projektia merkittävää kiirettä. Tästä huolimatta jouduin kuitenkin hieman kiristämään tahtia juuri ennen välitarkastusta sekä ennen loppupalautusta, sillä viimeisten ongelmien poistaminen ohjelmasta vie usein suurimman osan ajasta. Alkuperäisessä suunnitelmassani olin olettanut olevani valmis neljännen periodin arviointiviikolla, joten tästä

aikataulusta olin jäljessä. Lisäksi toteutin kaiken yksikkötestauksen vasta viimeisinä päivinä, sillä koin ohjelman toimivan kohtuullisen hyvin ilman sitä. Kokonaisuudessaan projektin aikataulu näytti GitLabin committien seurannan mukaan seuraavalta:

Viikko 6:

9.2 Projektiaiheen valinta ja rakenteen pohdinta

Viikko 7:

Projektin työstämistä lokaalisti koneella
Luodaan yksinkertaiset versiot luokista: Game, SaveParser, Character, Gui, …
Dijkstran ja Bresenhamin algoritmit

Viikko 8:

20.2 GitLab projekti luotu
23.2 Algoritmien näyttäminen ruudukolla
25.2 Projektisuunnitelman palautus

Viikko 9:

26.2 Hahmojen kuoleminen
27.2 Vuorojärjestelmä
28.2 Hahmoille annettu hyökkäykset ja lisätty pelin päättyminen
1.3 Projektisuunnitelman demosessio
2.3 Kävelyanimaatiot
3.3 Vihollisten kävelyanimaatiot, hahmoikonit, bmp parser, hyökkäysinteraktio
4.3 Tooltipit, kansiorakenteen muutos, hyökkäysikonit

Viikko 10:

5.3 Ensimmäinen grafiikkapäivitys, ikonit kaikkialla
7.3 Toinen grafiikkapäivitys, siirtyminen isometriseen kuvakulmaan, taustakuva kartalla

Viikko 11:

Viikko 12:

Viikko 13:

26.3 Ensimmäisen tekoälyn lisääminen

28.3 Projektin checkpoint

Viikko 14:

Viikko 15:

Viikko 16:

22.3 Hitsplatit, 2 uutta tekoälyä

Viikko 17:

23.3 Lisää grafiikkaa, esimerkkikarttoja ja skenaarioita
24.4 Korjattu yllättävä itsemurhaonglema, lisää yksikkötestausta
27.4 Loppuraportin palautus

13. Arvio lopputuloksesta

Kokonaisuudessaan ohjelma on mielestäni toimiva ja helppokäyttöinen kokonaisuus, joka tarjoaa suhteellisen sujuvan pelikokemuksen. Ohjelma toteuttaa kaikki tehtävänannossa vaaditut asiat kuten useamman eri tavalla käyttäytyvän tietokonevastustajan, erilaisia ja monipuolisia hyökkäyksiä, vaihtelevat kartat sekä hyökkäysten mahdolliset väliaikaiset vaikutukset hahmoihin.

Ohjelma on lisäksi erittäin monipuolisesti muokattavissa ulkoisilla tiedostoilla, jotka ovat helposti luettavissa ja muutettavissa. Pelin isometrinen näkymä, hahmojen ikonit sekä kartan taustakuva antavat pelistä uskottavan vaikutelman, joka ensisilmäyksellä saattaisi vaikuttaa ehkä jopa mobiilipelitasoiselta toteutukselta.

Suurimpana puutteena peli vaatii helpon tavan siirtyä ottelusta toiseen sekä mahdollisesti tavan muistaa hahmojen asetuksia yksittäisten taistelujen välillä. Vaikka tämä on keskeinen puute ohjelman tämänhetkisessä toteutuksessa, on tämä laajentamismahdollisuus pidetty mielessä ohjelman suunnittelussa, tiedon tallennusmuodoissa sekä kansiorakenteessa.

Vaikka pelin tekoäly on hyvin toteutettu, sen toiminta on melko yksinkertaista ja sen parantamista voisi harkita ulkoisia kirjastoja käyttämällä. Lisäksi ohjelman luokkajakoa voi harkita uusiksi, jos keksin paremman tavan toteuttaa vuorojen etenemisen.

14. Viitteet

Ohjelman rakenteen suunnittelun aloituspisteenä on käytetty kurssin harjoituskierros 5:n tehtävää RobotWorld. Erityisesti graafisen käyttöliittymän perusteita on lainattu tästä koodista. Lisäksi saman harjoituskierroksen tehtävää 5.4 Reading a human-writeable file on käytetty apuna tiedostojen lukemisessa.

Apuna pythoniin liittyvissä ongelmissa on lähtökohtaisesti käytetty kurssin omaa materiaalia, joka on saatavilla osoitteessa: https://plus.cs.hut.fi/y2/2018/

Lisäksi projektin toteutuksessa on luettu erittäin runsaasti Qt5:n dokumentaatiota: http://doc.qt.io/qt-5/

Algoritmit on toteutettu seuraavien wikipedia-arttikeleiden yleisten kuvausten perusteella:

1. Wikipedia - Dijkstra's algorithm, Saatavilla: https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm , Luettu: 24.04.2018

2. Wikipedia - Bresenham's line drawing algorithm, Saatavilla: https://en.wikipedia.org/wiki/Bresenham%27s_line_algorithm , Luettu: 24.04.2018

15. Liitteet

Projektin lähdekoodi on saatavilla osoitteessa:

https://version.aalto.fi/gitlab/alhoa1/strategy-game

Ohjelman voi ajaa käyttämällä komentoa **./main.py** kansiossa **/src**.