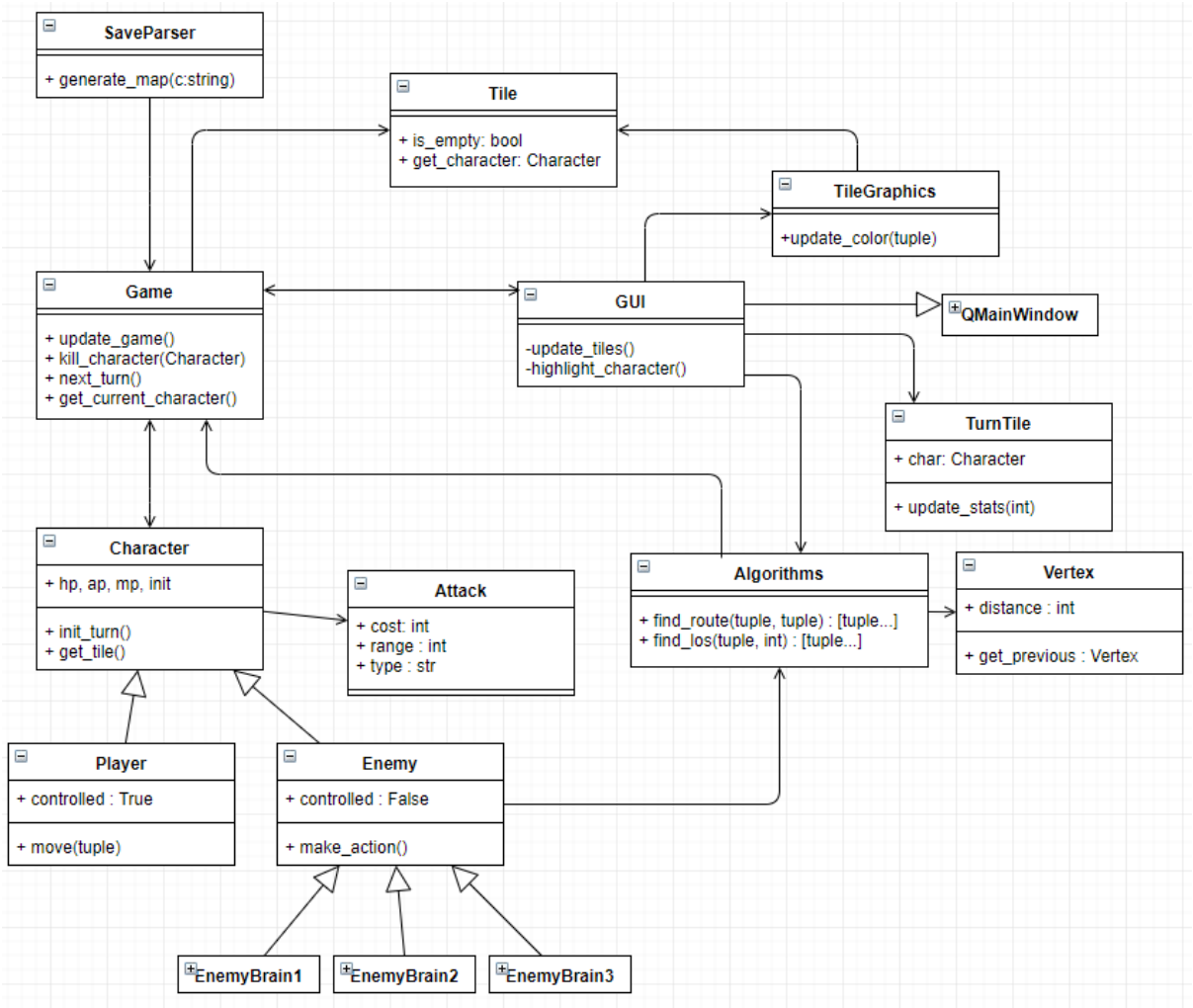
CS-A1121 Ohjelmoinnin peruskurssi Y2 Arno Alho 424628

Tekninen suunnitelma 25.02.2018

1. Ohjelman rakennesuunnitelma

Kuvassa 1 on esitetty karkea UML kaavio ohjelman rakenteesta.

Ohjelma jakautuu selkeästi graafisiin luokkiin kuten GUI ja TileGraphics 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



Kuva 1. Ohjelman rakennekaavio

Ohjelman keskeisten luokkien perustoiminnot on kuvattu alla:

Game-luokka vastaa itse pelin pyörittämisestä kuten vuorojen vaihtumisesta, pelilaudan tilanteen seuraamisesta sekä pelin päättymisen tarkastamisesta.

Character-luokka sisältää hahmon perustiedot ja sillä on kaksi eri aliluokkaa: ihmisen hallitsema Player sekä tietokoneen hallitsema Enemy. Näillä luokilla on määritelty erikseen vuoron kulkua kuvaavat metodit.

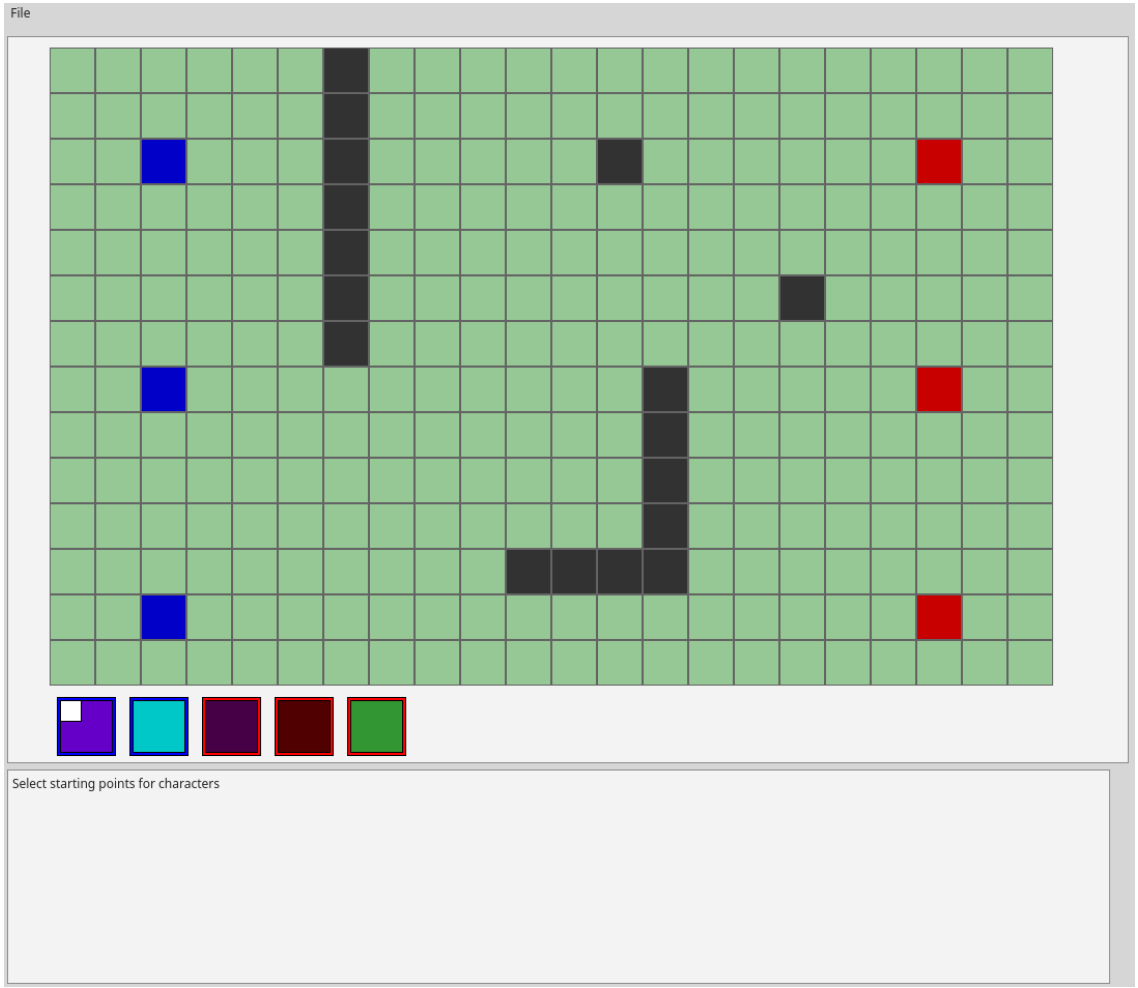
Enemy-luokka käyttää algoritmiluokan metodeja ja luo niiden perusteella korkeamman tason metodeja, joita se käyttää vuoron aikana päätösten tekemiseen. Tällä luokalla on useita erilaisia aliluokkia, joiden avulla voidaan luoda eri tavalla käyttäytyviä vihollisia.

GUI-luokka vastaa graafisen käyttöliittymän piirtämisestä sekä pelaajan komentojen vastaanottamisesta. Luokka käyttää algoritmiluokkaa pelaajan kävelyreittien sekä hyökkäysten mahdollisten kohderuutujen korostamiseen.

2. Käyttötapauskuvaus

Ohjelman alkaessa käyttäjälle avautuu dialogi-ikkuna, josta hän voi valita haluamansa skenaarion tai aikaisemmin tallennetun pelin. SaveParser luokka käyttää dialogi-ikkunasta valittua tiedostoa ja luo Game-olion sen perusteella. Tämä olio sisältää kaiken pelin kannalta keskeisen informaation kuten listan pelaajista sekä tiedon kartan tilanteesta.

Luotu Game-olio esitetään käyttäjälle GUI-luokan avulla. Aluksi käyttäjälle näytetään ainoastaan pelikenttä, jossa hän valitsee omien pelaajiensa aloituspisteet. Tietokone valitsee vastaajalle omat aloituspisteensä vuorojärjestyksessä. Kuva 2 kuvaa tilannetta pelaajien aloituspisteen valinnan aikana.

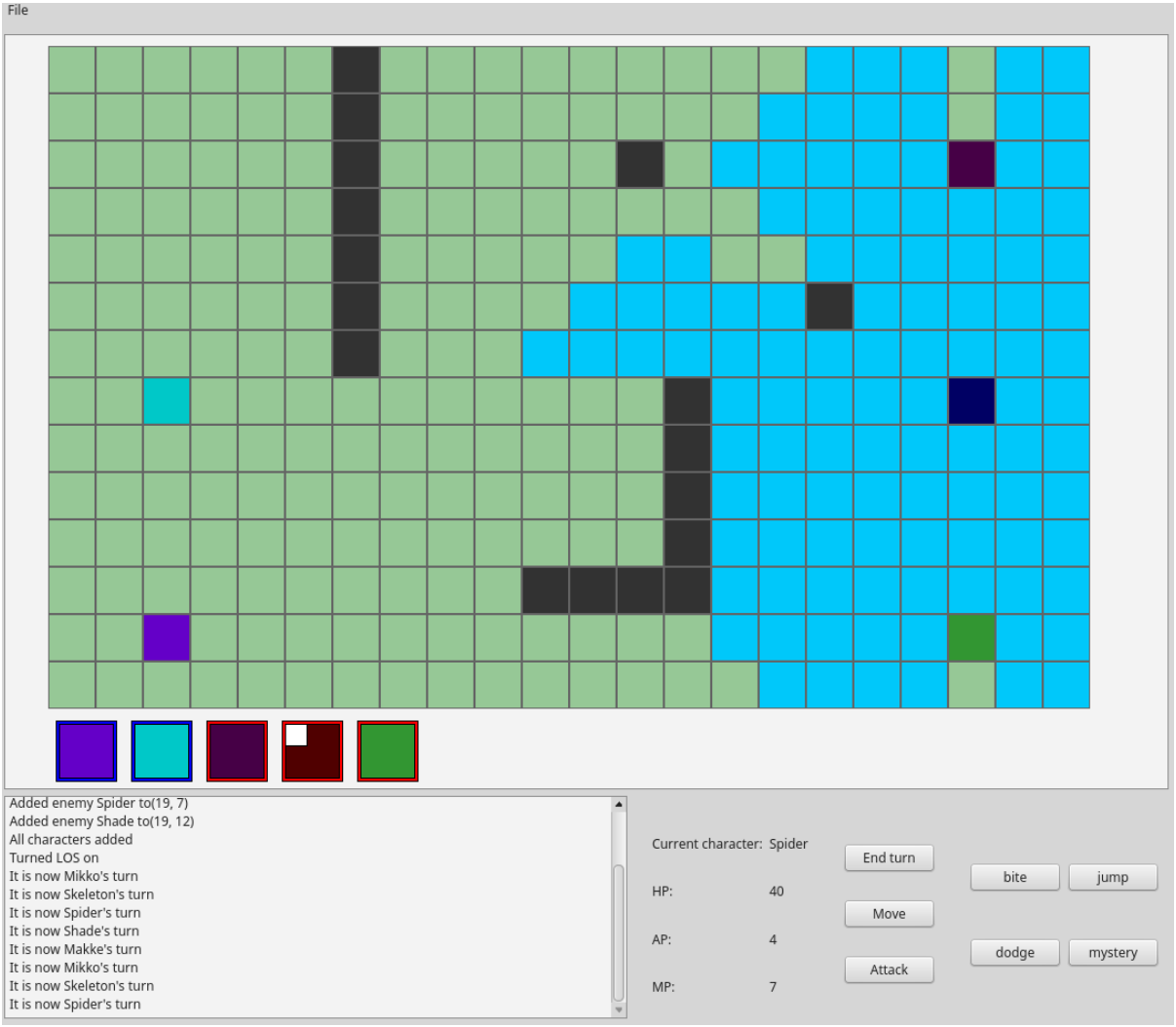


Kuva 2. Pelaajien aloituspaikkojen valinta

Kun kaikkien pelaajien aloituspisteet on valittu, alustaa GUI-luokka lopun peli-ikkunasta, jossa näkyy nyt kunkin pelaajan toimintavaihtoehdot sekä tilastot. Peli etenee pelaajien graafiseen ikkunaan merkityn vuorojärjestyksen mukaan, jossa kullakin hahmolla on oma

vuoronsa. Game-olio käsittelee vuorojen etenemistä sekä päättää pelin päättymisestä, kun kaikki toisen puolen hahmoista ovat kuolleet.

Pelaajahahmon vuorolla käyttäjä voi liikuttaa sitä hahmon liikkumispisteiden verran (MP) ja tehdä toimintoja hahmon toimintapisteillä (AP). Toimintojen suorittaminen tapahtuu valitsemalla toiminta grafiikkaikkunan alapuoleisesta toimintapaneelista, jonka jälkeen pelikentässä korostetaan ruutuja, joihin pelaajan toiminto voi kohdistua. Kuvassa 3 on esitetty pelaaja suorittamassa hyökkäystä, joka vaatii suoran näkölinjan maksimissaan 10 ruudun päähän.



Kuva 3. Tummansininen pelaaja suorittamassa hyökkäystä

Ruutu, johon toiminto kohdistuu, valitaan graafisesta ikkunasta klikkaamalla kohderuutua. Pelaaja voi lisäksi pyytää ohjelmalta muiden hahmojen pistetilanteita klikkaamalla hahmoja joko suoraan pelikentästä tai vuorolistasta. Pelaajien graafiset ruudut eroavat tavallisista QGraphicsRectItemeistä siten, että ruutu sisältää tiedon siinä olevasta hahmoluokasta. Tällöin hahmoihin on helppo kohdistaa toimintoja graafisen käyttöliittymän kautta.

Pelaajan tekemät hyökkäykset voivat vaikuttaa pelin muihin hahmoihin, jolloin Game-olio muuttaa sen sisältämien Character-luokkien asetuksia. Esimerkiksi pelaajan tekemät hyökkäykset vähentävät toisen pelaajan elämäpisteitä (HP). Vuoro päättyy pelaajan painaessa vuoron lopetusnappia eikä peli anna pelaajalle aikarajaa vuoron mitalle. Kun

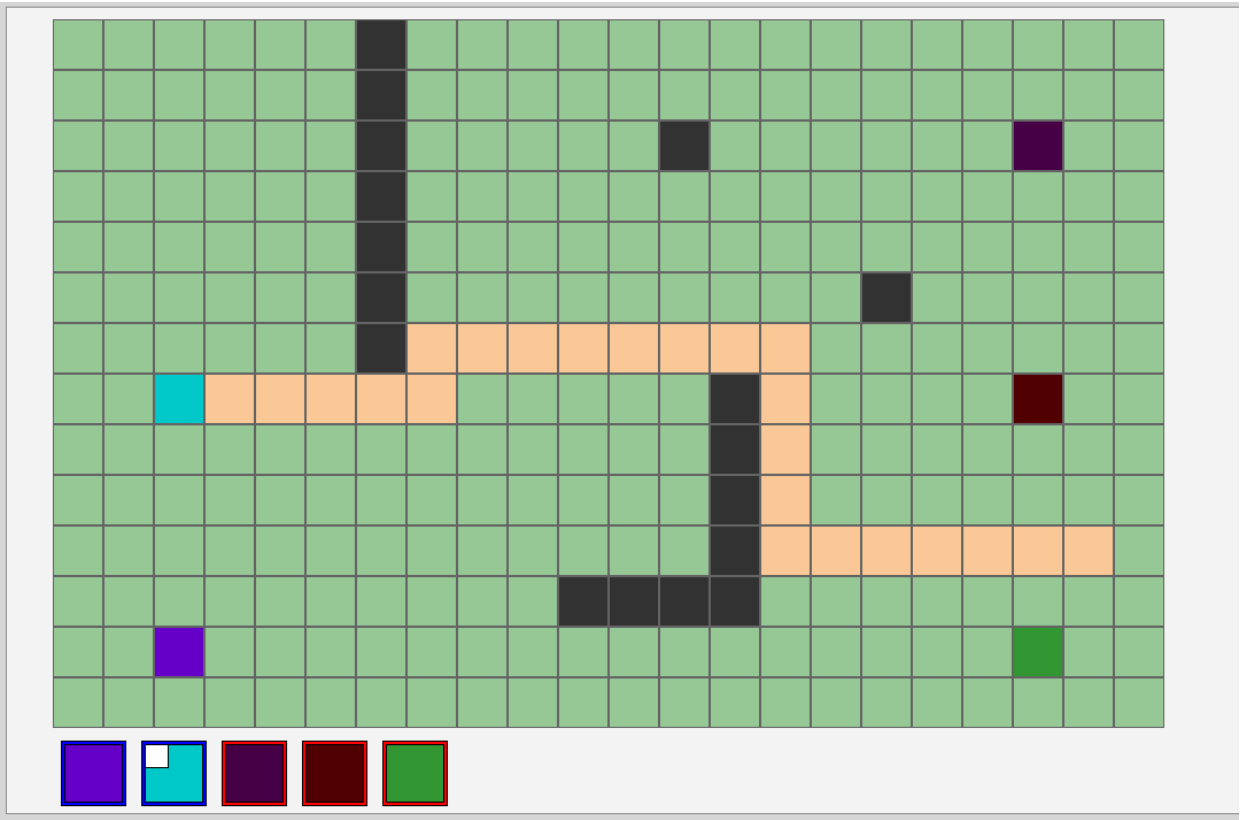
hahmon elämäpisteet (HP) putoavat nollan alapuolelle, hahmo kuolee ja se poistetaan pelialueelta.

Vihollisen vuorolla kutsutaan Enemy-olio suorittaa toimintojaan yksi kerrallaan kutsumalla metodia make_action. Jokaisen toiminnon välissä on lyhyt viive, joka helpottaa pelin seuraamista. Character-olioille syötetään Game-olion tiedot, minkä perusteella Enemy-olio luo vihollisille toimintoja. Muuten vihollishahmot ovat hyvin samanlaisia pelaajahahmojen kanssa, ja molemmat perivät ominaisuutensa luokasta Character. Peli päättyy, kun kaikki toisen puolen hahmot ovat kuolleet.

3. Algoritmit

Peli hyödyntää reitinvalinnassa Dijkstran algoritmia. Algoritmi merkitsee aluksi kaikki kentän ruudut vierailemattomaksi, jonka jälkeen se tarkastelee ruutuja yksi kerrallaan lähtien valitusta aloitusruudusta. Jokaiselle tarkasteltavan ruudun vierailemattomalle naapurille lasketaan etäisyys aloituspisteeseen lisäämällä alkuperäisen ruudun etäisyys tarkasteltavan ruudun etäisyyteen. Tämän ohjelman ratkaisussa kaikkien ruutujen keskeneräiset etäisyydet ovat samat ja siten jokaisen ruudun etäisyys naapuriinsa on 1. Kun kaikki tarkasteltavan ruudun naapurit on käyty läpi, merkitään ruutu vierailluksi ja siirrytään ruutuun, jolla on sillä hetkellä lyhyin etäisyys aloituspisteeseen. Tätä prosessia jatketaan, kunnes lopetuspiste on merkitty vierailluksi. [1]

Dijkstran algoritmia käytetään pelaajien liikkumiseen, jolloin käyttäjä voi suoraan valita ruudun, johon pelaaja liikkuu, ja ohjelma antaa käyttäjälle lyhyimmän reitin kyseiseen pisteeseen. Lisäksi vihollisen tekoäly käyttää samaa algoritmia löytääkseen reitin joko suoraan pelaajan luo tai paikkaan, josta se kykenee hyökkäämään pelaajan kimppuun. Algoritmin toiminta on näkyvissä kuvassa 4.



Kuva 4. Reitinetsintäalgoritmi vaaleansiniselle pelaajalle

Toinen tärkeä algoritmi, jota käytetään näkyvyyden testaamiseen, on Bresenhamin viiva-algoritmi. Algoritmi approksimoi joukon ruutuja, jotka parhaiten asettuvat määrätylle suoralle. Tämä toteutetaan seuraamalla suoran kulmakerrointa esimerkiksi x-suuntaan tietyin välimatkoin ja lisäämällä kulmakertoimen vaikutus virhetekijään. Kun virhetekijä on kasvanut riittävän suureksi, on siirryttävä ruudukon seuraavalle riville y-suunnassa. Algoritmi toimii ainoastaan tietylle kulmakertoimen etumerkille ja suuruudelle, minkä takia tämän ohjelman koodissa käsitellään kahdeksan eri tapausta, joissa algoritmin virhetekijöiden sekä kulmakertoimien merkit vaihtuvat.

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 3.

Näiden kahden perusalgoritmin lisäksi tietokonevastustajilla on oltava korkeamman tason koneäly, joka käyttää edellä mainittuja algoritmeja toimintojen valintaan. Pelissä on määritelty muutama yleinen Enemy-luokan alaluokka, joista on valittava yksi hahmon luonnin yhteydessä. Yksinkertaisin esimerkki on hahmo, joka pystyy hyökkäämään vain 1 ruudun päähän. Tämä hahmo pyrkii aina kävelemään lähimmän pelaajahahmon viereen ja käyttämään kaikki toimintapisteensä hyökkäämään pelaajaa päin. Algoritmeista on mahdollista kehittää myös erittäin paljon monimutkaisempia, jolloin ne ottavat huomioon esimerkiksi kaikkien pelaajien asetukset kuten liikkumis-, toiminta-, ja elämäpisteet. Tavoitteena on luoda vähintään kolme erilaista vihollistyyppiä, joiden yksinkertaistetut toimintamallit ovat:

- Pyrkii aina lyömäetäisyydelle

- Pyrkii kurkkaamaan esteen takaa ja menemään piiloon hyökkäyksen jälkeen

- Pyrkii osumaan useampaan pelaajaan kerralla erikoishyökkäyksillään

Tämä on osa ohjelmaa, jossa kehittämismahdollisuudet ovat rajattomat, minkä takia olen pyrkinyt olemaan hieman konservatiivinen omien suunnitelmieni osalta. Jos projekti etenee muuten hyvää vauhtia, tämä on alue, johon voi käyttää kaiken ylimääräisen ajan.

4. Tietorakenteet

Pelin tallennetiedot sekä pelaajien toimintojen tiedot on tallennettu helposti luettavaan tekstitiedostoon. 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.

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 myös suoraan nähdä pelin sisältö, mikä helpottaa peliskenaarioiden suunnittelua, sillä peliä ei tarvitse käynnistää, jotta näkee kartan sisällön. 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 sisällä suurin osa tiedosta on tallennettu käyttämällä pythonin tarjoamia valmiita tietorakenteita. Suurin osa laajemmista tiedoista kuten esimerkiksi tieto kaikista pelaajista on tallennettu listoihin. 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 algoritmiluokassa.

5. Ohjelma on tämän suunnitelman kirjoitushetkellä jo melko hyvässä vaiheessa ja se kykenee jo luomaan graafisen pelin annettujen tiedostojen perusteella. Itse pelin toiminnot kuitenkin vielä puuttuvat eikä ohjelma siedä virheitä tallennustiedostoissa. Graafinen käyttöliittymä helpottaa merkittävästi itse ohjelman toiminnan arviointia, minkä takia olen pyrkinyt tekemään sen ensimmäisenä.

Projekti tulee tavoitteen mukaisesti etenemään alla olevan listan määräämässä järjestyksessä. Toimintojen välissä lisätty karkeita välietappeja, mutta jokaiselle osalle ei ole määrätty omaa deadlinea.

- Hyökkäysten lukeminen tiedostoista

- Tiedostolukemisen yksikkötestit

- Pelaajien liikuttaminen

Viikko 10 (5.3-11.3)

- Algoritmien yksikkötestit

- Lyönitetäisyysteköälyn luominen, jotta pelin muita ominaisuuksia voidaan testata järjestelmätasolla

Checkpoint 19.3-29.3

- Enemy:n yksikkötestit

- Pelaajien välinen interaktio toimintojen kautta

- Pelikokonaisuuden testaaminen järjestelmätasolla

- Monimutkaisempien tekoälyjen lisääminen

- Monimutkaisempien tekoälyjen yksikkötestit

IV periodin arviointiviikko (3.4-6.4)

6. Yksikkötestaussuunnitelma

Tärkeimpiä yksikkötestauksen aiheita tulevat olemaan tiedostojen lukeminen sekä tekoälyn ratkaisujen analysointi. Tiedostojen lukemisen testaaminen tulee olemaan näistä vaihtoehdoista huomattavasti helpompaa, sillä testien tarvitsee vain tarkastaa, huomaako SaveParser luokka tiedostoissa olevia virheitä.

Keskeisin SaveParser-luokan metodi on generate_map, joka luo Game-olion metodille annetun tiedoston perusteella. Metodin on tärkeä tunnistaa tiedostossa olevat perustiedot,

tulkita kartan kuva oikein sekä tunnistaa kaikki tiedostossa mainitut pelaajat. Metodin toiminta voidaan varmistaa kutsumalla Game-luokan metodeita get_characters ja get_tiles, jotka palauttavat pelissä olevat hahmot sekä pelin kartan. SaveParser-luokan on siedettävä tiedostoja, joissa on ylimääräisiä kommentteja sekä pyrittävä luomaan pelitilanne, vaikka hahmoille olisi määritelty parametrejä, joita peli ei tunne.

Tietokonetekoälyjen on toimittava kohdassa 5 kuvatulla tavalla. Tätä toimintaa tullaan testaamaan asettamalla tekoälyjä erilaisiin karttoihin ja vertaamalla niiden käyttäytymistä oletettuun malliin. Jälleen kerran Gamen metodit ovat kriittisessä asemassa, mutta näissä testeissä on saatettava luoda ylimääräisiä rakenteita, joilla simuloidaan pelaajan läsnäoloa.

Testien tulosten oikeuden arviointi voi olla hankalaa, sillä tekoälyt saattavat käyttäytyä hyvin, muttei täysin optimaalisesti. Näissä tilanteissa on tehtävä yleisemmällä tasolla olevia järjestelmätestejä ja pyrittävä arvioimaan hahmojen toiminnan järkevyyttä.

Toinen tapa lähestyä tätä ongelmaa on tehdä testeistä riittävän yksinkertaisia, jotta ”oikea” ratkaisu olisi selvä. Esimerkiksi yksikertaisimman pelaajan luo pyrkivän tekoälyn on aina käveltävä mahdollisimman tehokasta reittiä, ja pelaajan saavuttaessaan sen on hyökättävä pelaajan kimppuun kaikilla toimintapisteillään. Näiden kaikkien metodien suoritusta voidaan analysoimalla kutsumalla Character-luokan metodeita, jotka kertovat hahmojen eri pistetilanteet sekä sijainnin.

Ennen kuin on mahdollista testata tekoälyjen toimintaa, on kuitenkin varmistettava, että tekoälyn taustalla toimivat algoritmit toimivat oikein. Algorithms-luokan metodeja onkin hyvin helppo yksikkötestata, sillä metodit ottavat sisäänsä vain parin koordinaatteja ja palauttavat listan koordinaatteja. Algorithms-luokalla on kuitenkin oltava tieto pelikentästä, joten esimerkiksi optimaalisen reitin löytämistä varten on jälleen käytettävä Game-luokan metodeja.

Yksikkötesteissä on hyvä testata algoritmiluokan toimintaa tavallisissa tilanteissa ja varmistettava, että palautetut koordinaattilistat vastaavat niiden taustalla olevia matemaattisia malleja. Tulosten tarkistamista varten olevat listat on siis laskettava käsin. Algoritmien on myös hyvä sietää virheellisiä syötteitä, jolloin niiden on palautettava tyhjä lista.

7. Linkit

Ohjelman rakenteen suunnittelun aloituspisteenä on käytetty kurssin harjoituskierros 5:n tehtävää RobotWorld. Erityisesti graafisen käyttöliittymän perusteita on lainattu voimakkaasti 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 tähänastisessa 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: 25.02.2018

2. Wikipedia - Bresenham's line drawing algorithm, Saatavilla: https://en.wikipedia.org/wiki/Bresenham%27s_line_algorithm , Luettu: 25.02.2018