Project Document for CS-A1121

Ilari Ojakorpi 609113, EST, 07.05.2020

Yleiskuvaus:

Vuoropohjainen strategiapeli jossa tekoälykäs tietokonevastustaja. Pelissä komennetaan erilaisia joukkoja graafisen käyttöliittymän avulla ja sen laajennettavuus jatkossa on suhteellisen helppoa.

Ohjeet käyttäjälle:

Ohjelma käytetään ajamlla Code kansiossa sijaitseva main.py. Tällöin avautuu Graafinen käyttöliittymä pelin main menuun. Tästä valikosta voidaan lähinnä ajaa peli painamalla new game ja valitsemalla valikosta etukäteen konfiguroitu mapfile jonka jälkeen painamalla play, itse peli-ikkuna avautuu.

Pelin ohjeet:

Right-click: valitse ruutu (jos ruudussa on yksikkö näkyvät sen tiedot alapalkissa)

Left-click: jos jokin ruutu jossa on yksikkö on valittuna voit left-clickillä liikuttaa sen toiseen ruutuun tai hyökätä viholliseen painamalla toista ruutua

Double-click: avaa kaupa näkymän painamalla linna tai kylä ruutua

W/A/S/D: Peli näkymän liikuttaminen ylös/vasen/alas/oikea

Scroll-wheel: Peli näkymän zoomaus

Next turn nappulasta peli antaa tietokoneen suorittaa oman vuoronsa ja Quit nappulasta pääsee takaisin valikkoon. Tavoitteena on liikkua vastustajan linna ruutuun.

Ohjelman rakenne:

Ohjelma koostuu erilaisista luokista jolla hallinoidaan pelin eri osia.

- **Game (game.py):** pelin hallinnoinnin pääluokka joka suorittaa suurimman osan pelissä tapahtuvista operaatioista ja toimii aktiivisen peli instanssin ylimpänä luokkana. Game sisältää pelin pelaajat (Player) ja pelilaudan (Level) ja sisältää lähes kaikki pelin hallinnolliset funktiot.

  Game on toteuttu Pyqt5 Qobject luokkana joak mahdollistaa pyqtSignalien käytön luokassa, nämä erilaiset signaalit ovat tapa jolla pelissä tapahtuvat muutokset välitetään Graafiselle käyttöliittymälle. Strategiapeli ei vaadi jatkuvaa informaatio virtaa kuten esimerkiksi reaaliajassa liikkuvat graafiset komponentit vaatisivat joten näin vältetään liika riippuvuus Game luokan ja graafisen käyttöliittymän välillä ja saadaan helppo toteus luokkien keskinäiselle kommunikaatiolle. Lisäksi esimerkiksi jos jatkossa haluttaisiin animoida pelin spritejä voitaisiin se toteuttaa helposti graafisen käyttöliittymän sisällä välittämättä siitä mitä Game luokassa tapahtuu.

- **Player (player.py):** Jokaista pelaajaa kohden on yksi player luokka joka sisältää pelaajan tiedot, pelaajan yksiköt ja pelaajan hallitsemat ruudut.

  Luokka toimii lähinnä informaation säilöntään jotta game luokan sisään ei tarvitsisi rakentaa monimutkaisia rakenteita saman informaation säilömiseen.

- **Unit (unit.py):** Jokaista pelin yksikköä mallinnetaan unit luokalla jossa on kyseisen yksikön informaatio.

- **Level (level.py):** Level luokka hallinoi pelilautaa ja sisältää n*m määrän Tile luokkia. Level sisältää myös reitinlöytö algoritmin get_movable jolla selvitetään mihin eri pelikentän ruutuihin yksikkö voi liikkua ja find_closest_movable jota tekoäly kätää etsiessään reittä vastustajan linnaan.

- **Tile (tile.py):** Jokaista pelikentän ruutua mallinetaan tile esineellä joka sisältää jokaisen ruudun tyypin, kenen hallinnassa ruutu on (state), ja sisältääkö ruutu Unit esineen.

- **GuiControl (gui_control.py):** Graafisen käyttöliittömän hallinnointi luokka jonka tehtävänä on vaihtaa näkymää MainWindow ja GameWindow esineiden välillä.

- **MainWindow (gui_menu.py):** PyQt5 toteutus graafiselle käyttöliittymälle jossa päävalikon eri sivuja hallinnoidaan QStackedLayoutin avulla, jos valikkoon haluttaisiin lisätä jokin uusi sivu voitaisiin se toteuttaa hyvin helposti luomalla uusi init funktio halutulle sivulle ja lisäämällä init_menu funktioon uusi Qpushbutton joka yhdistettäisiin funktioon self.make_display(n) jossa n on uuuden sivun numero -1. Make_display mahdollistaa sivujen dynaamisen lisäämisen ilman että jokaiselle sivuvaihdolle tarvistseee kirjoittaa omaa funktiota sillä PyQt5 .connect ei hyväksy funktioita joilla on syötteitä.

- **GameWindow (gui_game.py):** PyQt5 toteutus jolla halinnoidaan pelin graafista ilmettä, kuten edellä mainittu saa signaaleja Game esineeltä ja sisältää kyseisen game esineen. Sisältää Pyqt Graphics scene jossa hallinnoidaan GITile ja GIUnit esineitä.

- **GITile (gi_tile.py) ja GIUnit (gi_unit.py):** Luokat jolla piirretään pelin yksiköitä (Unit) ja tuutuja (Tile), piirtäminen toteutetaan pixmap funktiolla johon käytettävät png tiedostot lötyvät Code/resources kansiosta. Molemmat esineet sisältävät niitä vastaavan Unit tai tile esineen. Lisäksi GITilen kautta hallinnoidaan pelaajan hiiren käyttöä pelissä mousePressEvent ja mouseDoubleClickEvent funktioiden avulla. Näin vältetään hyvin matemaattinen ratkaisu joka voitaisiin toteuttaa GameWindowin sisään seuraamalla hiiren paikkaa ruudulla ja hakemalla siinä paikassa olevia esineitä, mutta nykyinen toteutus on huomattavasti helpompi ja nopeampi.

- **game_loader ja game_saver:** Sisältävät funktiot joilla luodaan teksti tiedostosta game esine ja tellennetaan game esineen tiedot tekstitiedostoon.

Algoritmit:

Get_movable on BFS algoritmi joka ottaa huomioon ruutuihin liikkumiseen menevän hinnan ja laskee kuinka monta ap:ta Unitin täytyy käyttää päästäkseen ruudusta x ruutuun y. Toteutus on tehty valmiin PriorityQue luokan pohjalle.

Find_closest_movable on algoritmi jota tekoäly käyttää etsiessään vastustajan linnaa ja hyödyntää siinä get_movablen tuottamaa matriisia josta se iteroi alkaen paikasta matrix[h][w] ja etsii pienimmän ”hintaisen” nappurin kunnes se löytää ruudun johon Unitilla riittää ap liikkua.

AI luokka on suhteellisen yksinkertainen ja tekee seuraavat toiminpiteet jokaisella vuorolla: ostaa uuden unitin castle ruutuun jos siihen on riittävästi kultaa, jokaiselle yksikölle hyökkää jos mahdollista liikkuu lähemmäksi vihollisen linnaa ja hyökkää jos ei jo hyökännyt aikaisemmin.

Näiden lisäksi game luokka sisältää liudan suhteellisen yksinkertaisia algoritmeja joilla hoidetaan pelin perus toiminnot.

Tietorakenteet:

Projekti sisältää pythonin perus tietorakenteita listoja, matriiseja, tupleja ja sanakirjoja. Näiden käyttö on intuitiivista esimerkiksi pelipöytää mallinnetaan matriisilla, koordinaatteja tupleilla ja esimerkiksi eri yksiköiden initialisointi informaatiota säilötään sanakirjoissa.

Tiedostot:

Peli käyttä tekstitiedostoa uuden pelin luomiseen jossa on seuraava rakenne:

```
1,0  / pelin vuoro, aktiivinen pelaaja
mie,0,0,100 / pelaaja1 nimi, pelaaja1 väri, ai (0 = False, 1 = True), kulta
#
p,p,p,p,p   / jokainen kirjan kuvastaa yhtä ruutua ja sen tyyppiä
p,c,p,c,p
p,p,f,c,p
#
0,0,1,1/ (pelissä olevat yksiköt, jokaiselle pelaajalle oma rivi) pelaaja, tyyppi, height, width
#
1,1,1    / (ruutujen statet) state,height,width
2,1,1

#
```

Testaus:

Testauksen toteutin lähinnä ajamalla mainia tai rakentamalla luokkia erillisinä muusta ohjelmasta.

Tunnetut puutteet ja viat ohjelmassa:

Game saver ei toimi tällä hetkellä vaan on rakennettu aikaisemman ohjelman version mukaan ja en ole tätä ehtinyt korjata. Perusperiaatteelta funktion toiminta on kuitenkin sama mutta muutamia informaation haku funktiooita pitäisi muuttaa.

Tietokoneen käyttämä funktio find_closest_movable ei ole täydellinen ja jossain tilanteissa se saattaa joutua tilanteeseen jossa se ei pysty löytämään ruutua johon Unit pystyisi liikkumaan. Tämä johtuu lähinnä siitä että toteutin get_movable funktion ennen tätä funktiota ja jouduin hieman soveltamaan get_movablen toteutusta, Jos teklisin algoritmin uudestaan kirjoittaisin luultavisti erillisen algoritmin jota ainoastaa tekoäly käyttäisi, jolloin voisin suoraan tehdä reitin etsintää ilman että täytyisi tutkia erikeen jokaista ruutua.

Moni ohjelman sanakirja voitaisiin siirtää yhteen tiedostoon ja kirjoittaa muutama funktio joka loisi esimerkiksi QPushButtoneja gui_gamen init_CastelUI:hin dynaamisesti sen mukaan monta erilaista yksikköä pelissä on. Tällä hetkellä esimerkiksi uusien yksikkö tyyppien lisääminen on hieman hankalaa sillä pitää muistaa muokata muuutamaa eri teidosta jotta saadaan haluttu toimivuus.

Tämän lisäksi peli ei ole tällä hetkellä kovin käyttäjä ystävällinen ja ei anna paljoa palautettta eri toiminnoista. Esimerkiksi valittaessa yksikköä ei nähdä ruutuja johon ei ole mahdollista liikkua tai joihin on mahdollista liikkua. Tämän kyseisen ongelman korjaisin joko uusilla QgraphicsItemeillä tai PyQtn piirtämis funktioilla ja lisää palautetta siitä mitä pöydällä tapahtuu saisi upottamalla gui_gamen GameUI layouttiin jonkinlaisen textboxin johon siirrettäisiin stringejä sen mukaan mitä Game luokassa tapahtuu.

3 parasta ja 3 huonointa aluetta:

Parhaat:

- Yleinen rakenne: Ohjelman yleinen rakenne on hyvin suunniteltu ja vaikka on muutama asia jotka toteuttaisin nyt toisin tällä hetkellä projektin perusrakenteen päälle voisi lähte rakentamaan monimutakisempiakin toimintoja ja ajattelinkin käytää projektin yleisideaa tulevissa projekteissa.
- Liikkumis algoritmi: vaikka tietokeneen käyttämässä funktiossa on pieniä ongelmia on get_movable kuitenkin aika hyvin rakennettu algoritmi joka pienellä hienosäädöllä voisi olla erinomainen ja onkin mielestäni projektin parhaiten onnistuneita puolia.
- Graafisen käyttöliittymän rakenne: tällä hetkellä gui_control, gui_menu ja gui_game ovat pohjia joihin olisi hyvin vaiavtonta lisätä lisää toiminnallisuutta ja olen itse tyytyväinen käyttöliittymän saavutettuun tasoon ajatellen kurssin vaatimuksia ja käytettyä aikaa. Varsinkin MainWindowin layouttien käsittely ja signaalien käyttö GameWindowin päivittämisessä olivat hyviä oivalluksia.

Huonot:

- Käytettävyys: kuten edellä mainittu peli ei ole tällä hetkellä kovin intuitiivinen pelata ja toimnnasta ei saa paljon palautetta näihin minulla toki oli selvät suunnitelma miten lähtisin paranatamaan käytettävyyttä
- Ominaisuudet: suurin osa ajasta meni perusrakentaiden hiomiseen jolloin en ehtinyt tehdä ohjelmaa paljoa syvyyttä, toisaalta tässä on hyvä pohja johon näitä ominaisuuksia voisi ruveta rakentamaan.
- Muokattavuus: suuren osan ohjelman konfiguraatiosta olisi voinut toteuttaa omaan tiedostoon ja tehdä muutaman dynaamisen funktion tämän tiedoston pohjalta jolloin ohjelman laajentaminen olisi helppoa ja intuitiivista.

Muutokset alkuperäiseen suunnitelmaan ja aikataulu:

Ohjelman toteutus sujui aikalailla alkuperäisen suunnitelman mukaan lukuunottamatta graafista käyttöliittymää jonka tein samalla kun opin käyttämään PyQta. Jälkikäteen ajatellen olisin toki tehnyt muuutamia asiota eritavalla.

Lopputulos:

Olen itse tyytyväine nlopputulokseen vaikka se onkin vielä jonkin matkaa siitä mitä ajattelin saavani aikaiseksi, vaaditun tehtävänannon pääpiirteet kuitenkin mielestäni toteutuvat ja jos lähtisin toteuttamaan projektia uudelleen olisi minulla huomattavasti selvemmät suunnitelmat siisä miten täläinen kokonaisuus kannattaisi suunnitella ja toteuttaa.

Ohjelman hyvistä ja huonoista puolista olen jo aikaisemmin dokumentaatiossa kertonutkin mutta voisin vielä lisätä että esimerkiksi luokkien välistä kommunikaatiota voisin selventää käyttämällä esimerkiksi luokka muuttujia luokkien välillä, tälllä hetkellä kukin luokka vaihtaa informaatiota periaatteella mitä olen tarvinnut kun olen ohjelmaa kirjoittanut mutta ohjlemasta voisi tehdä huomattavasti siistimmän ja selvemmän kokonaisuuden.

Lähteet:

https://doc.qt.io/qt-5/

https://www.redblobgames.com/pathfinding/grids/algorithms.html

https://www.redblobgames.com/pathfinding/grids/graphs.html

https://stackoverflow.com/questions/41290035/pyqt-change-gui-layout-after-button-is-clicked/41290921

https://gist.github.com/MalloyDelacroix/2c509d6bcad35c7e35b1851dfc32d161

https://www.learnpyqt.com/courses/start/creating-your-first-window/