Tekninen suunnitelma CS-A1121

Akseli Konttas 587031

1. Ohjelman rakennesuunnitelma

Tärkeimmät pelissä olevat luokat:

Game: Luokka, joka kasaa yhteen muut luokat.
Kenttiä:
- Board: pelikenttä

Metodeja:
- Init: Luo uuden pelin: luo uuden pelikentän sekä yhden pelaajan ja yhden tietokonevastustajan.
- Get_board: palauttaa pelikentän.

Board: Sisältää tiedon pelikentästä pelin aloitustilanteessa sekä kulloisenkin vuoron jälkeen.
Kenttiä:
- Height: kentän korkeus ruutuina
- Width: kentän leveys ruutuina
- Board: Kentän tila eli sen ruudut ja niiden sisältämät objektit. Tieto tallennetaan kaksiulotteisena listana.

Metodeja:
- Get_height: palauttaa kentän korkeuden.
- Get_width: palauttaa kentän leveyden
- Get_piece: palauttaa parametreina annetussa ruudussa olevan objektin.
- Get_square: Käänteinen edelliselle; palauttaa parametrina annetun objektin sijainnin pelilaudalla (tai None, jos ei löydy).
- Move_piece: Saa kaksi parametria: ruutu, josta halutaan siirtää hahmo ja ruutu, johon se halutaan siirtää. Metodi tarkistaa, onko siirto laillinen ja siirtää hahmon jos näin voi tehdä.
- Load_map: Lataa tiedostosta kartan ja asettaa pelikentän sen mukaiseksi. (Käytännössä init-metodin apumetodi.)

Player: Kuvaa pelaajaa. Kokoaa kasaan muun muassa pelaajan ohjaamat hahmot sekä käytössä olevat tavarat.
Kenttiä:
- Characters: Lista, joka sisältää pelaajan ohjaamat pelihahmot.
- Backpack: Lista, joka sisältää pelaajan käytössä olevat tavarat. (Voisi olla myös vaikka sanakirja muodossa tavara-määrä?)
- Alive: Oletuksena True. Pelaaja on hävinnyt, kun kaikki ihänen ohjaamansa hahmot ovat kuolleet. Tällöin muuttuu Falseksi.

Metodeja:
- Is_alive: Palauttaa kentän alive (siis kertoo onko pelaaja vielä mukana pelissä).
- Add_character: Lisää pelaajan characters-listaan parametrina annetun hahmon.
- Remove_character: Poistaa pelaajan characters-listasta parametrina annetun hahmon. Jos lista on tämän jälkeen tyhjä, muuttaa is_alive-kentän Falseksi.

Human: Luokan Player alaluokka, joka kuvaa ihmispelaajaa. (Voisi toki nimetä jotenkin paremmin :P)

Metodeja:

- Do_turn: Metodi, jonka avulla pelaaja voi tehdä vuoronsa. Käytännössä pitää kirjaa siitä, mitä hahmoja on jo liikutettu vuoron aikana ja jatkaa eteenpäin, kun kaikki ovat liikkuneet.

AI: Luokan Player alaluokka, joka kuvaa tietokonepelaajaa.
Metodeja:

- Do_turn: Sama kuin luokalla Human, paitsi ei ota käskyjä pelaajalta vaan päättää itse mitä tekee.

Lisäksi apumetodeja tekoälyn toteuttamiseen. Niiden toiminnasta lisää myöhemmin tässä suunnitelmassa.

Character: Kuvaa yhtä pelilaudan hahmoista ja säilyttää sen informaation, kuten elämäpisteet sekä tila. Sisältö saadaan erillisestä tiedostosta.
Kenttiä:

- HP: Hahmon elämäpisteet. Hahmo kuolee, kun sen elämä menee nollaan (tai sen alapuolelle).
- MaxHP: Hahmon maksimielämäpisteet. Toimii paitsi aloitusmääränä, myös varmistaa sen ettei hp mene korkeammalle kuin pitäisi.
- Class: Kuvaa hahmon luokkaa.
- Alive: Kertoo, onko hahmo hengissä.
- Range: Kuvaa hahmon ”nopeutta”, eli sitä, kuinka monta ruutua se voi siirtyä vuorossa.
- Player: Hahmoa ohjaava pelaaja (eli joko AI tai Human-olio).
- Square: Hahmon senhetkinen ruutu.
- Status: Kuvaa hahmon senhetkistä tilaa.

Lisäksi muita kuvaamaan hahmon voimakkuutta yms.
Metodeja:

- Add_hp: Lisää hahmolle parametrina annetun määrän elämäpisteitä. Mikäli uusi määrä olisi yli MaxHP:n, tulee määräksi MaxHP.
- Remove_hp: Poistaa hahmolta parametrina annetun määrän elämäpisteitä. Mikäli uusi määrä on 0 tai alle, hahmo kuolee ja sen alive-kentäksi tulee False.
- Get_hp: Palauttaa hahmon elämäpisteet.
- Get_range: Palauttaa hahmon rangen.
- Get_player: Palauttaa hahmoa ohjaavan pelaajan.
- Set_square: Kutsuu Board-olion get_square-metodia, ja muuttaa square-kentän sen mukaisesti. Tämä tapahtuu aina liikkumisen jälkeen, mutta itse hahmon liikuttaminen tapahtuu board-olion metodien avulla.
- Attack: Hahmo suorittaa hyökkäyksen. Alustavan suunnitelman jokaisella hahmolla on kaksi hyökkäystä, joista pelaaja voi valita. Hyökkäyksestä ja hahmosta riippuen pitää mahdollisesti vielä valita kohde jne.
- Deal_damage: Saa parametrikseen hyökkäyksen kohteessa olleessa ruudussa olevan hahmon. Metodi laskee, kuinka paljon vahinkoa tehdään, ja kutsuu sen perusteella toisen hahmon remove_hp-metodia.
- Is_enemy: Määrittää, onko jokin tietty hahmo vihollinen sen perusteella, onko niillä sama player-kenttä. Palauttaa True tai False.
- Set_status: Asettaa tilan hahmolle.

Item:
Kaikkien tavaroiden yliluokka. (Tämä on vielä vähän suunnittelun alla.)
Kenttiä:

- Owner: Tavaran omistavan pelaajan nimi
- Type: Kuvaa tavaran tyyppiä (nimeä tai muuta vastaavaa)

Metodeita:

- Use: Käyttää tavaran. Määritellään tarkemmin aliluokassa.

IO: Tallentaa/lataa pelitilanteen.

Edellä olevien luokkien metodeita voisi periaatteessa selkeyttää ja muutenkin joissain paikoissa ehkä yksknkertaistaa luokkien välistä työnjakoa, mutta periaatteessa koin, että helpointa on tehdä omat olionsa melkein kaikelle, mitä pelistä löytyy. Tavaroiden kanssa jäin kyllä vielä pohtimaan, onko siinä järkeä.

Lisäksi sisältää luokkia graafista käyttöliittymää varten. En ole valitettavasti kovin paljoa ehtinyt tutustua PyQT:hen, mutta alla on alustava luokkajako (voi muuttua paljonkin):

BoardGUI: Piirtää pelikentän tilanteen ruudulle.
CharacterGUI: Piirtää yksittäisen hahmon, BoardGUI laittaa sen oikealle paikalleen.
Log: Pitää pelilokia ja tulostaa sitä aina, kun jotain tapahtuu.
HUD: Piirtää erikseen kaikkien hahmojen elämäpisteet ja tilan (myöhemmin määritettävällä tavalla).
GUI: Kaikki graafisen käyttöliittymän luokat kokoava luokka, joka muodostaa tarvittavat ikkunat ja päivittää niitä.

2. Käyttötapauskuvaus

Yksinkertainen esimerkki: pelaaja pelaa tietokonetta vastaan pelin, jossa molemmilla on vain yksi hahmo, ja kenttä on 5x5 ruudukko.

Ensin main-funktio luo uuden Game-olion. Ensin Game luo kaksi pelaajaa; toinen on Human- ja toinen AI-olio. Molemmat luokat luovat init-metodeissaan pelihahmot (tässä tapauksessa vain yhdet) ja lisäävät ne characters-listaansa. Sitten Game luo pelikenttää kuvaavan Board-objektin, joka saa kentän ulkoasun erillisestä tiedostosta. Board lisää annetut pelihahmot ennalta määrättyihin paikkoihin kentällä (selviää tiedostosta; jos paikkoja on liian vähän tms, on vielä vähän auki mitä pitää tehdä :P todennäköisesti hahmot laitetaan sitten satunnaisiin paikkoihin). Sitten graafinen käyttöliittymä päivitetään näyttämään tätä tilannetta.

Ensin on pelaajan vuoro, joka päättää siirtää hahmoaan valitsemaansa ruutuun. Hän klikkaa pelihahmoaan sekä sen jälkeen ruutua, johon sen haluaa siirtää. Board-olio määrittää, onko siirto laillinen. Jos ei ole, mitään ei tapahdu. Jos on, siirretään hahmo valittuun ruutuun. Sitten tietokone siirtää hahmoaan (myöhemmin lisää algoritmista) ja hyökkää pelaajan hahmoon. Käytännössä tietokone selvittää, mikä hahmo ruudussa on, ja sen jälkeen kutsuu hyökkäävän hahmon deal_damage-metodia, joka laskee molempien hahmojen attribuuttien perusteella tehtävän vahingon ja välittää tiedon edelleen hyökkäyksen vastaanottavalle hahmolle. Gui päivittyy, samoin kuin loki.

Periaatteessa näin jatketaan, kunnes toisen hahmon (nyt vaikka tietokoneen) HP putoaa nollaan. Hahmo kuolee, gui poistaa sen graafisesta käyttöliittymästä ja se poistuu myös itse pelissä. Nyt AI-olion alive-kenttä muuttuu Falseksi. Main funktio huomaa tämän ja julistaa pelin loppuneeksi ja tässä tapauksessa pelaajan voittaneeksi.

3. Algoritmit

Muutamia määriteltäviä algoritmeja pelissä:

- Miten peli tarkistaa siirtojen laillisuuden
- AI!!
- Tehtävän vahingon määrä

Viimeinen on suht yksinkertainen, mutta pelin pelattavuuden kannalta tärkeä. Käytännössä pitää löytää hyvät, tasapainossa olevat kaavat.

Siirtojen laillisuuden kanssa voi tulla ongelmia. Ajatuksena on, että hahmot voivat liikkua vaihtelevan määrän ruutuja vuorossa riippuen hahmon kyvyistä. Yksi tapa olisi laskea lähtöruudusta kaikki ruudut, jotka sijaitsevat hahmon maksimiliikkumismäärän (”rangen”) sisäpuolella, mutta se on aika raskas tapa, jos range on iso. Kuitenkin se on paras tapa, mikä tuli itselle mieleen.

Ja sitten se tekoäly. Sen kanssa on toki vielä paljon pohdittavaa, mutta alla joitain suunnitelmia.

Ensinnäkin jo projektikuvauksessa mainittu, tekoäly ei saa perustua vain satunnaisuuteen. Ajattelin kuitenkin käyttää jonkin verran satunnaisuutta, jotta kaikki pelit olisivat vähän erilaisia. Ajatuksena siis, että tietokone määrittää vuoronsa alussa kaikki ruudut, joissa on vihollisia. Sitten jokaiselle hahmolleen erikseen se ensin määrittää, mihin ruutuihin hahmo voi liikkua, ja seln jälkeen mihin vihollisen hahmoihin se voi hyökätä. Mikäli näitä hahmoja on monta, valitsee tietokone niistä sen, jolla on korkein ”prioriteetti” (parantajat, jo valmiiksi vähillä HP:illa olevat) ja hyökkää siihen (todennäköisesti).

4. Tietorakenteet

Pelissä käytettäviä teitorakenteita ovat ainakin pelilauta sekä hahmojen tallentamiseen käytetty lista.

Valitsin kaksiulotteisen listan kuvaamaan pelilautaa, sillä se on yksinkertainen ja toimivaksi koettu tapa. Sen indeksointi on helppoa ja on helppo hahmottaa, missä on mitäkin.
Lisäksi kunkin pelaajan ohjaamat hahmot ajattelin tallentaa listaan. Tämä siksi, että siihen voi helposti lisätä ja siitä voi poistaa hahmoja, vaikkei ennestään tietäisikään tulevien hahmojen määrää. Lisäksi listan kaikki hahmot on helppo käydä läpi for-silmukalla.

5. Aikataulu

Tarkoituksena on saada peli rakennettua niin, että ensin saan valmiiksi pelin muut luokat ja niiden toimiessa siirryn graafisen käyttöliittymän kimppuun. Suhteellisen realistinen aika-arvio ensimmäiseen osaan olisi noin kuukauden kieppeillä. Alustavalla suunnitelmalla luokkia olisi 8 (joskin lisää todennäköisesti tulee), joista monet ovat aika yksinkertaisia toteuttaa – itseasiassa ainoat, joiden uskon tuottavan suurempaa haastetta, on Board sekä Character (sekä AI). Kuitenkin minulla on seuraavan kuukauden aikana paljon muutakin menoa, joten keskittyminen jotuu jakaantumaan muaallekkin.
Tämän jälkeen aikaa jää vielä lähes kuukausi graafisen käyttäliittymän koodaamiseen. Tämä tulee tarpeeseen, sillä PyQT on itselleni melko vieras enkä usko, että kovin yksinkertaisetkaan elementit sujuvat minulta kovin nopeasti. Graafisen käyttöliittymän aloitan tietenkin ensin tärkeimmistä, joilla pelin saa ylipäänsä toimimaan, siitä siirtyen spesifimpiin ja hienompiin, jos jää aikaa. Esimerkiksi lokin voi jättää kokonaan pois, jos aika jää lopussa tiukille.

Aloitan käytännössä listan yläpäästä tekemään luokkia: ensin yleisemmät luokat, mistä mennään eteenpäin spesifimpiin. Aikaisemmin mainitut vaikeammat luokat tosin jäävät varmaan alussa tynkäluokiksi.

6. Yksikkötestaussuunnitelma

Alla on kuvattu joitain metodeita, jotka ovat hyvin tärkeitä ohjelman toiminnan kannalta ja siksi niitä tuleekin testata heti ja kattavasti.

Pelilaudan move_piece –metodi on ehkä olennaisin metodi koko pelissä. Jos se ei toimi oikein, ei peli toimi käytännössä ollenkaan.
Metodin valmistuttua sitä voi kokeilla, kunhan Character-luokka on aloitettu (sen metodeita ei juuri tarvita). Tarkoituksena on ajaa metodia eri hahmoilla, erilaisilla kentillä ja eri syötteillä tarkoituksena selvittää, onko funktion paluuarvo oikein (eli onko siirto laillinen) ja siirtyykö hahmo oikeasti haluttuun paikkaan (tämän todentamiseen käy hyvin laudan get_square – metodi).

Toinen hyvin olennainen metodi on Character- luokan attack-metodi. Metodille ei anneta parametrina ruutua, johon aiotaan hyökätä, sillä jotkut hyökkäykset saattavat osua useampaan ruuttuun samanaikaisesti. Ajatuksena on, että metodi selvittää ensin, voiko hahmo ylipäätään osua kehenkään siitä ruudusta, missä hän on, niillä hyökkäyksillä, joita hän voi käyttää. Jos voi ja kohteena voi olla vain yksi pelaaja, täytyy edelleen selvittää, mihin ruutuun halutaan hyökätä. Sitten tapahtuu vielä laskeminen ja vahingon tekeminen.
Tätä metodia on vaikea testata, ellei Board-luokka ole jo melko valmis. Sen jälkeen on mahdollista luoda useita hahmoja, asettaa ne sopiviin paikkoihin kentällä ja kokeilla, mitä hyökkäykset tekevät. Pitää siis kokeilla, määrittääkö peli oikein hyökkäyksen laillisuuden (esim. omia hahmoja ei saa lyödä) ja tapahtuuko oikea vahingon tekeminen, vaikka metodi muuten toimisikin.

Kolmas ja ehkä vaikein on pelitilanteen tallennus ja lataus. Tämä ei kuitenkaan ole kovin oleellinen osa peliä, joten se jää aika viimeiseksi. Sen testaamiseen tarvitaan jo melko valmis peli, jolloin voidaan testata tallentaa peli, ladata sama peli ja katsoa mitä tapahtuu.

7. Kirjallisuusviitteet ja linkit

Itse peliin toteutukseen en ole vielä etsinyt vinkkejä, (suunniteltuja) pelimekaniikkoja olen kyllä lainannut useistakin peleistä.
Lähteitä, joita tulen varmasti käyttämään:

- PyQT –tutoriaalit, kuten tämä: http://zetcode.com/gui/pyqt5/
- Pythonin documentaatio: https://docs.python.org/3/