(A*) lyhimmän polun etsijä / visualisoija

Määrittelydokumentti

Saku Säisä / 013106344
saku.saisa@helsinki.fi

Helsingin yliopisto
Aineopintojen harjoitustyö: tietorakenteet ja algoritmit
Vuodenvaihde 2013-2014

Työn aihe

Valitsin harjoitustyöni aiheeksi lyhimmän polun etsimisen pääsääntöisesti A* -algoritmia käyttäen, sillä olen kiinnostunut peliohjelmoinnista jossa tämä on yksi hyvinkin keskeinen osa-alue. Halusin siis valita sellaisen aiheen jonka oppimisesta olisi itselleni mahdollisimman paljon konkreettista hyötyä. Minun pitää muutenkin hiljalleen alkaa opettelemaan enemmän polunetsinnän, tekoälyn jne ohjelmointia, joten tämä työ toimii siihen hyvänä ponnahduslautana.

Toteutan työn lyhimmän polun visualisoijana. Ohjelma tulee näyttämään graafisen ruudukon ja siihen piirrettynä paitsi lyhimmän polun aloituspisteestä lopetuspisteeseen, myös alueet jotka algoritmi tutki ennen kuin löysi lyhimmän polun jos polkua on olemassa lainkaan. Käyttäjä voi siirtää aloitus- ja lopetuspistettä, piirtää ruudukolle läpäisemättömiä esteitä, sekä asettaa tiettyjä asetuksia jotka säätelevät algoritmin toimintaa. Näin ohjelmalla pystyy havainnollistamaan miten algoritmi suoriutuu erilaisissa tilanteissa ja erilaisilla asetuksilla.

Käytetyt tietorakenteet ja algoritmit

Pohjana polunetsinnässäni tulen käyttämään *Dijkstran* algoritmia. Lisäämällä siihen heuristiikkafunktion joka arvioi etäisyyden maaliin ja valitsee seuraavan askeleen hyödyntäen tätä tietoa, muuntuu algoritmi *A** -algoritmiksi. Ohjelmassa voi valita, käyttääkö se A* -algoritmia vai ei. Lisäksi haaveena on toteuttaa vielä kolmantena vaihtoehtona A* joka käyttää hyödykseen *JPS*:ää (Jump Point Search). Tämä viimeinen ei ole kuitenkaan vielä tässä vaiheessa täysin varma vaan riippuu lähinnä siitä, kuinka hyvin saan kaiken muun ensin toimimaan ja jääkö sen jälkeen vielä aikaa saada myös JPS toimintakuntoon.

Alkuun toteutan ohjelman käyttäen Javan valmista PriorityQueue -rakennetta jonka korvaan myöhemmin omalla binäärikekototeutuksella. Algoritmin käyttämä ”closed set” toteutuu alkuun Javan ArrayListina ja myöhemmin joko itse toteutettuna linkitettynä listana tai ehkä paljon yksinkertaisemmin puhtaasti taulukkona boolean-arvoja joista näkee suoraan tietyn ruudun kuulumisen siihen. Tämä ainakin nopeuttaisi toimintaa huomattavasti, sillä ruudun kuulumisen joukkoon saa selvitettyä vakioajassa sen sijaan, että sitä pitäisi etsiä minkäänlaisesta listarakenteesta tai vastaavasta.

Ohjelman saamat syötteet

Tulen toteuttamaan ohjelman niin, että sille voi antaa parametreina kaksi kokonaislukua jotka määrittävät käytetyn ruudukon koon x- ja y-akselilla. Jos parametreja ei anna, käytetään jotain vakiota jossa koko ohjelmaikkuna varmasti mahtuu kerralla näytölle vähän huonommallakin resoluutiolla.

Vaativuuksista

Pyrin toteuttamaan työn niin, että binäärikekoa käyttävän A* -algoritmin yleiset aika- ja tilavaativuudet (silloin kun heuristiikkafunktio toimiivakioajassa) toteutuvat.

Lähteitä

- http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html
- http://www.cs.helsinki.fi/u/floreen/tira2013/tira.pdf
- http://www.policyalmanac.org/games/aStarTutorial.htm
- http://en.wikipedia.org/wiki/A*_search_algorithm