Toteutusdokumentti, Hike-reitinetsintä

Ohjelman yleisrakenne

Ohjelma sisältää 6 pakkausta:

- Algorithms. Reitinetsintään käytettävän Astar algoritmin luokassa ”Pathfinder”.
- Graph. Verkon kuvaamisen tarkoitetut luokat, ”Node” esittää solmua ja ”Edge” kaarta.
- ImageTable. Luokka ”ImageTable” muuttaa saamansa kuvatiedoston kaksiulotteisiksi solmutaulukoksi.
- Structures. Reitinetsintään tarvittavat tietorakenteet. ”LinkyList” on linkitetty lista jota käytetään solmusta lähtevien kaarien tallentamiseen käyttäen apuna ”LinkElement” luokkaa. ”MinHeap” on minimikeko jota käytetään reitinhaussa pienimmän etäisyyden solmun valintaan. ”PathStack” on pino jota tarvitaan valmiin reitin tulostukseen.
- Controls. ”ClickListener” on ActionListener, jolla tarkkaillaan käyttäjän klikkauksia käyttöliittymässä.
- GameWindow. Käyttöliittymän näyttämiseen liittyviä luokkia. ”MenuDraw” ja ”MenuWindow” hoitavat alkuvalikon, ”MapDraw” ja ”MapWindow” selvitetyn reitin näyttämisen. ”ShowPicture” lukee kuvia.

Saavutetut aika- ja tilavaatimukset

Ohjelman käyttämien tietorakenteiden ja algoritmien merkittävät metodit.

LinkyList:

Yhteen suuntaan linkitetty lista.

- Add(): lisää uuden kaaren listan alkuun ja sitä seuraavaksi vanhan alun. O(1).
- Muut metodit ovat vain talletetun arvon suoraan palauttavia ja vakioaikaisia. O(1).
- Javan for-looppeihin tarvittava Iterator-luokka sisältää myös vain vakioaikaisia metodeita.
- Luokan aikavaatimus on siis O(1). Lista sisältää kolme muuttujaa eikä listoja, joten myös tilavaatimus on O(1), kuitenkin jos otetaan huomioon että next muuttujan takana voi olla n-määrä olioita, on tilavaatimus O(n).

MinHeap:

Minikeko, binäärikeko.

- Insert(): lisää olion listan viimeiseksi ja tekee tarvittavat nostot kunnes keko-ehto pätee. Toteutettu kirjallisuuden mukaisella tavalla, joten aikavaatimus on O(log n). Kekoon lisäämäni solmun indeksin seuranta ja päivitys vievät vakioajan aikaa.
- Heapify(): Siirtää solmua alaspäin kunnes lapset ovat suurempia. Osa löytämästäni tiedosta väittää operaatiota O(n) operaatioksi, itse väitän tämän olevan O(log n).
- Swap(): Vaihtaa kaksi solmua joiden indeksit tunnetaan. O(1).

- PrintHeap(): Tämä on ainoastaan virheiden etsintään käytettävä luokka, jota ei varsinaisesti tarvita.
- RemoveMin(): Poppaa pienimmän arvon keosta ja suorittaa heapify()-metodin. O(log n).
- DecHeap(): vähentää solmun arvoa annetussa indeksissä, jonka jälkeen tekee swap()-operaatioita tarvittavan määrän. O(log n).
- Luokan aikavaatimus on O(log n). Tilavaatimus on kaikki nodet sisältävän taulukon vuoksi O(n).

PathStack:

Yksinkertainen pino.

- Push(): Lisää noden pinon loppuun. O(1).
- Pop(): Palauttaa pinon viimeisen arvon ja vähentää pituutta yhdellä. O(1).
- Pinon aikavaatimus on O(1). Tilavaatimus on taulukon takia O(n).

PathFinder:

Dijkstran algoritmi ja A* algoritmi joko Diagonal Distancella tai Manhattan Distancella.

- FindRoute(): Luokka poimii minimikeosta pienimpiä arvoja ja lähettää ne CheckNeighbours()-metodiin kunnes löytää etsityn. O(n).
- CheckNeighbours(): Lähettää solmusta lähteviä kaaria Relax()-metodiin. O(n).
- Initialize(): Alustaa taulukon alkupisteen ja lisää kaikki nodet kekoon. O(n log n).
- Relax(): Tarkistaa onko uusi reitti nopeampi. O(1).
- Heuristic(): Laskee heuristiikan tiedetyistä arvoista. O(1).
- Aikavaatimus on O(n log n). Alussa keko on aivan täynnä, ja tilavaatimus on O(n). Heuristiikat eivät välttämättä nopeuta suoritusta kovin paljon kun kartta on epäsuotuisa, siis myös A* on samaa vaatimusluokkaa.

Suorituskyky- ja O-analyysivertailu

Testien tulokset löytyvät testausdokumentista.

Puutteet ja parannusehdotukset

Tiedostonvalitsin, nyt tiedoston on oltavat oikeassa paikassa ja oikealla nimellä. Diagonaalisen haun parantaminen, nyt tulokset eivät ole aina oikeita käyttäessä Diagonal Searchia. Ilman viistoon liikkumista tulokset ovat oikeat ja nopeat.

Lähteet:

Tietorakenteet ja algoritmit-kurssimateriaali.

Amit’s A* Pages:
Varsinkin osio heuristiikasta.
http://theory.stanford.edu/~amitp/GameProgramming/

http://en.wikipedia.org/wiki/Dijkstra's_algorithm