Toteutusdokumentti

Ohjelman yleisrakenne

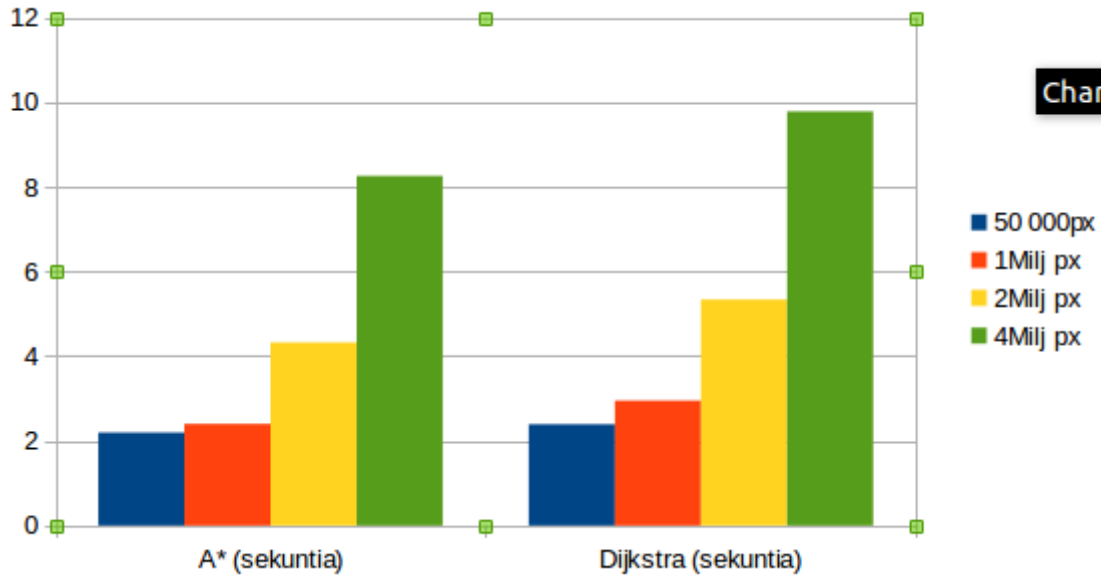
Ohjelman koodi on jaettu neljään pakkaukseen. Algorithms, datastructures, domain ja UI. Algorithms pakkauksesta löytyy luokat AstarSolver, DijkstraSolver sekä Solver interface jonka molemmat luokista toteuttavat. Ideana on, että erilaisia labyrintin ratkaisualgoritmejä on helppo lisätä ohjelmaan, kunhan ne toteuttavat Solver interfacen, eli palauttavat reitin NodeStackina.

Algoritmit on toteuttettu käyttäen itsekirjoittamiani tietorakenteita, heappia, nodelistiä ja stackia. Kyseiset tietorakenteet löytyvät pakkauksesta datastructures. Domain pakkaukseen olen eritellyt oliot joita käytän ongelmia ratkoessa, eli Maze ja Node. Labyrinttikuvaa luettaessa labyrintin pikselit muunnetaan Node olioiksi, ja Nodeolioiden kaksiulotteinen array talletetaan Maze olioon johon talletetaan myös muita olennaisia tietoja labyrintista.

Logic pakkauksesta löytyy luokat jotka "ohjailevat" tietoa ohjelman sisällä, tai eivät suoranaisesti liity itse algoritmeihin. MazeHandler nimensä mukaisesti käsittelee annettua labyrintin kuvaa, sekä osaa piirtää kuvaan annetun reitin. MazeSolver luokka välittää käyttäjän antamat tiedot MazeHandlerille ja MazeHandlerin antamat tiedot algoritmeille. Ohjelma toimisi myös ilman MazeSolver luokkaa, mutta se selkeyttää ohjelman rakennetta ja helpottaa ohjelman laajennettavuutta.

Suorituskyky

Kuvat ovat esteettömiä kuvia(=eivät sisällä seiniä) joissa aloitetaan vasemmasta yläkulmasta ja lopetetaan oikeaan alakulmaan. Suorituskyvyn mittaus ohjelmaa ajamalla on kuitenkin hieman ongelmallista, koska suuri osa ajasta kuluu kuvan muuntamisessa kaksiulotteiseksi taulukoksi. Testausdokumentaatiosta löytyy muutamia suorituskykytestejä lisää.



Tuloksista voidaan päätellä että aikavaativuus nousee verrannollisesti syötteen kokoon nähden. Kun pikseleiden määrä kaksinkertaistuu, kaksinkertaistuu myös aikavaativuus. Ainoastaan 50 000-> miljoonaan ei tapahdu näin, koska kuvan lukeminen arrayksi vie suurimman osan ajasta. Eli aikavaavituus on noin O(|V|+|E|), mutta koodia analysoimalla saamme Dijkstran aikavaativuudeksi O(|V|+|E|*log |V|) ja A* aikavaativuudeksi O(|V|+|E|). Aikavaativuudet ovat siis samat joihin pyrittiin.

Puutteet ja parannusehdotukset

Ohjelmassa voisi olla sisäänrakennettu aloitus ja lopetuspisteen lisäystoiminto. Lisäksi vedetty viiva voisi olla paksumpaa. Myöskin jonkinlainen visualisointi siitä, missä mikäkin algoritmi on käynyt voisi olla hyvä. Itse algoritmeihin en enää keksi suurempia parannuksia.