Tiralabra / Hannu Honkanen, periodi 6/2014

Testausdokumentti

Kehityksen aikaiset tulostukset

Kehityksen aikana tehtiin paljon tulostuksia eri muuttujien ym. arvoista. Näin varmistuttiin että joka paikassa on se tieto mikä pitääkin olla. Nämä tulosteet olivat usein hyödyllisiä bugien korjaamisessa. Junit-testeihin näitä tulostuksia on myös jätetty.

Junit-testaus

Pyrittiin tekemään mahdollisimman kattavat Junit-testit erityisesti toteutettaville tietorakenteille. Luokilla PriorityQueueKekoAlkionaPaikka ja OmaKekoAlkionaPaikka on samanlaiset testit. Kun testit tehtiin ensin minimikeon Java-toteutukselle, varmistettiin että testit on tehty oikein. Sitten samat testit olikin helppo kopioida minimikeon omalle toteutukselle.

Myös pinon omalle toteutukselle (luokka OmaPinoAlkionaPaikka) tehtiin paljon testejä.

Junit-testejä voi toistaa lähes rajatta ja ne ovatkin oivallisia huolehtimaan ettei mitään toteutettua jo rikota.

Algoritmien testauksessa keskityttiin tulosten (suoritusaikojen ja reittikuvien tarkasteluun).

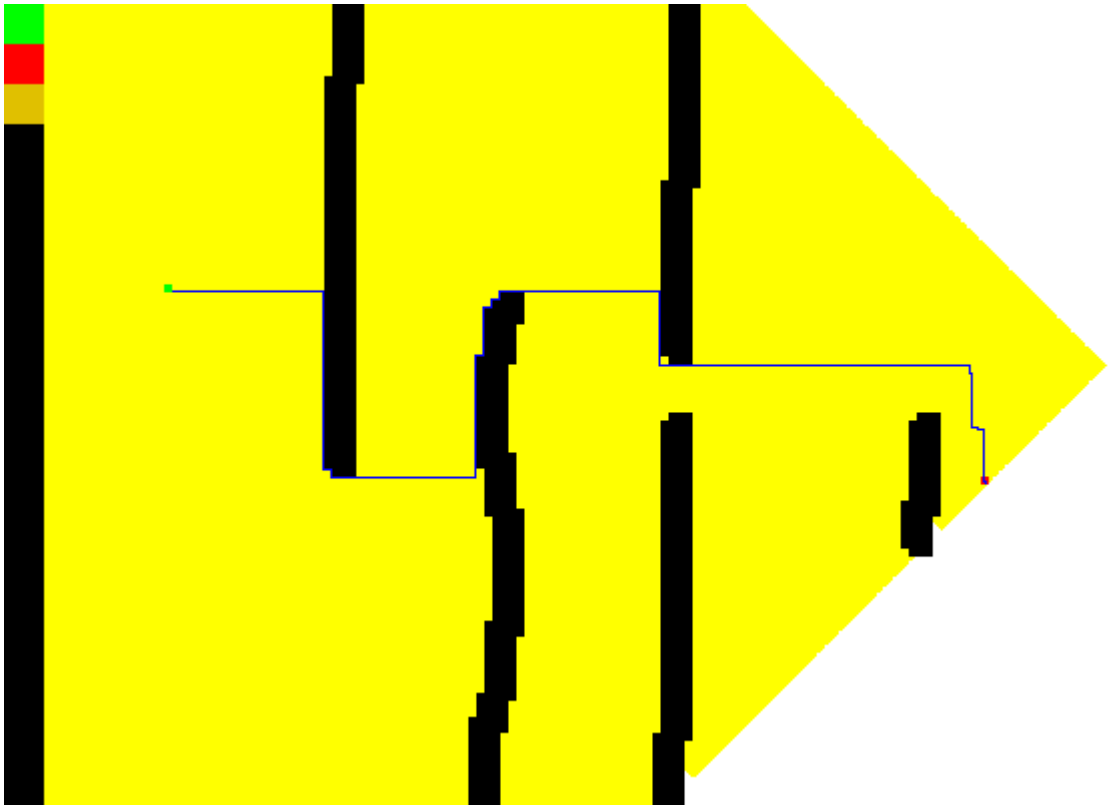
Oman tietorakenteen suorituskykytestaus verrattuna Javan tietorakenteeseen

Oma minimikeon toteutukseni toimi nopeammin kuin Javan Priorityqueue:n avulla tehty toteutus. Varsin tehokasta. Syynä voi olla esim. se että oma toteutukseni on varsin karsittu (ei esim. virhetarkistuksia).

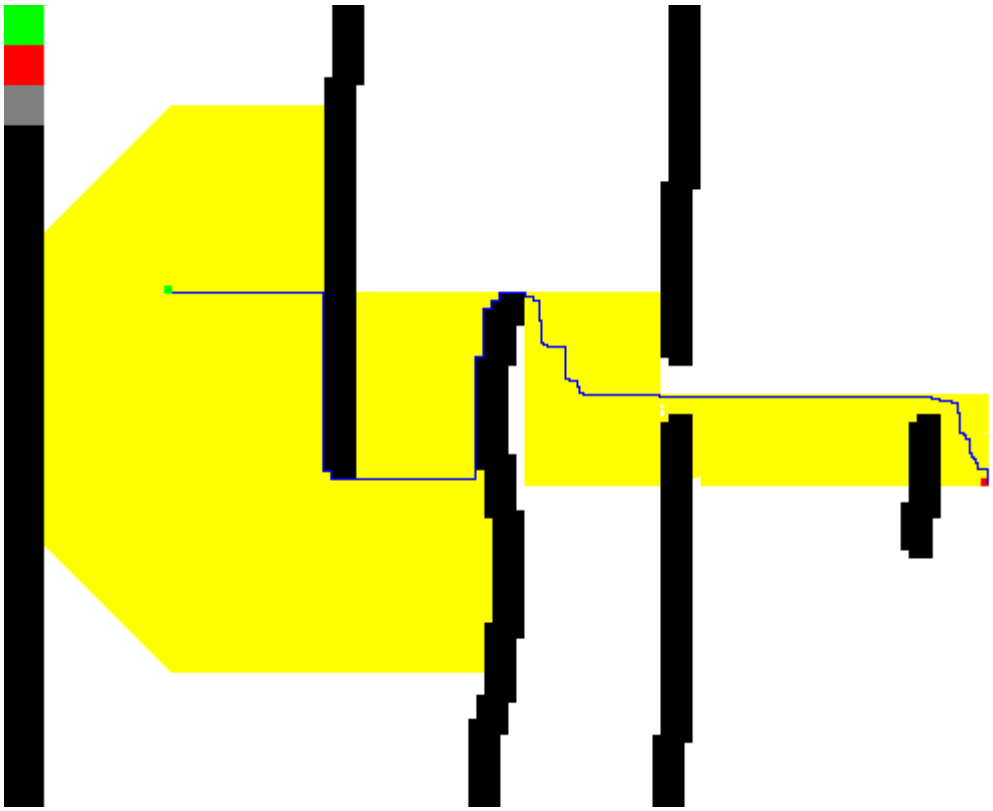
Visuaalinen testaus

Koska tulokset (sekä nopein reitti että kaikki käydyt solmut) saadaan kuvina, on niitä helppo tarkastella visuaalisesti. On varsin helppo päätellä onko ratkaisu (varsinkin Astar) oikeanlainen. Myös Dijkstran ja Astarin ero käy tuloskuvista hyvin ilmi: missä on käyty ja missä ei kullakin algoritmilla.

Esimerkki Dijkstralla ratkaistuna:



Ja sama Astarilla ratkaistuna:

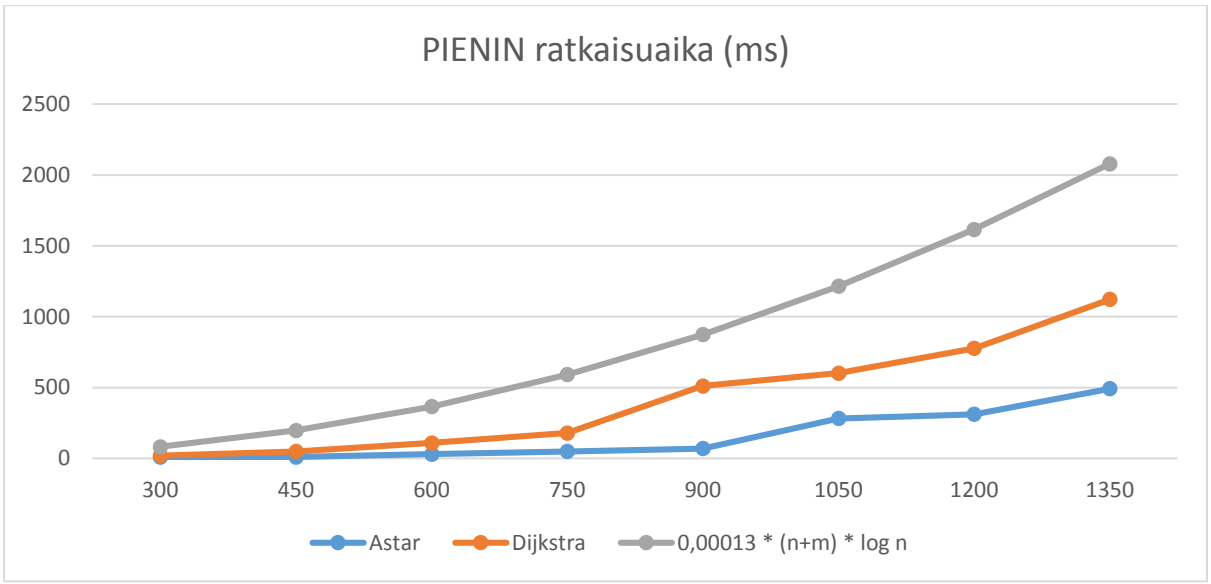


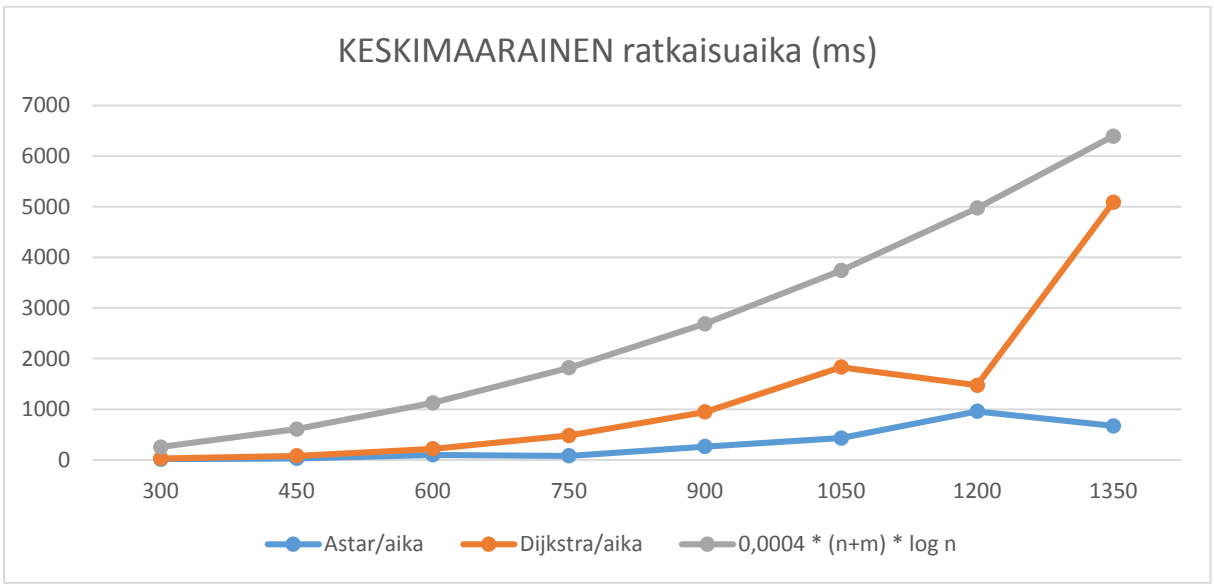
Suorituskykytestaus

Itse suoritettavalla ohjelmalla voi ajaa testisarjoja, joten testien toistaminen on helppoa.

Ajettiin 4 testisarjaa. Laskettiin 20 iteraatiota, joista jälkimmäiset 15 otettiin mukaan algoritmien ratkaisuajan keskiarvon laskentaan. Lisäksi otettiin talteen ratkaisuaikojen minimit. Nämä minimit käyttäytyivätkin johdonmukaisemmin kuin keskiarvot. Tulkitsen tätä niin että 20:n sarjoilla löytyi aina jokin rako jossa saatiin varsin häiriötön suoritus. Keskiarvojen laskemiseen kului aina 20 iteraation aika, jolloin Windows läppärini muut prosessit pääsivät häiritsemään tuloksia, vaikka koitin tätäkin minimoida käynnistämällä koneen uudestaan, sulkemalla kaikki muut ohjelmat sekä antamalla testiprosessille korkeimman mahdollisimman prioriteetin.

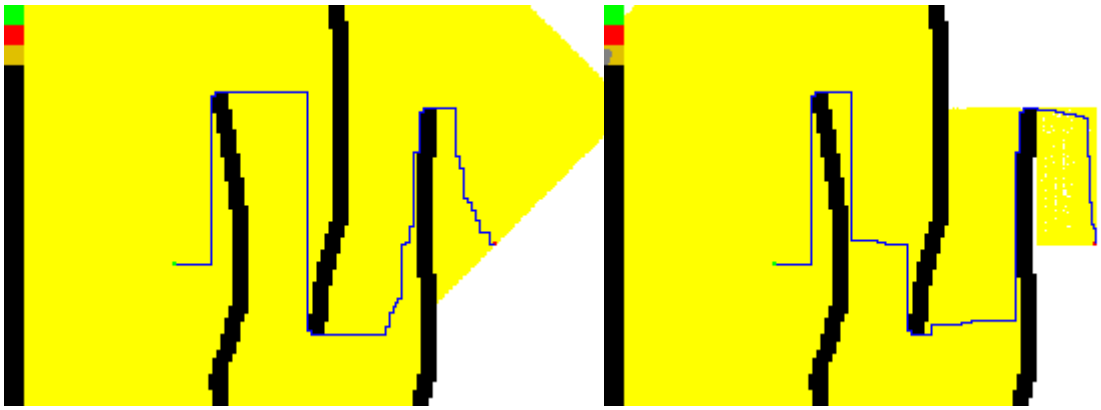
Testisarja eiEsteita: kartalla ei ole ollenkaan esteitä. Varioitiin lähtötiedoston kokoa. Vaaka-akselillla kartan koko pikselimäärä vaakasuunnassa (pikselin määrä pystysuunnassa on vaakapikselit*2/3).

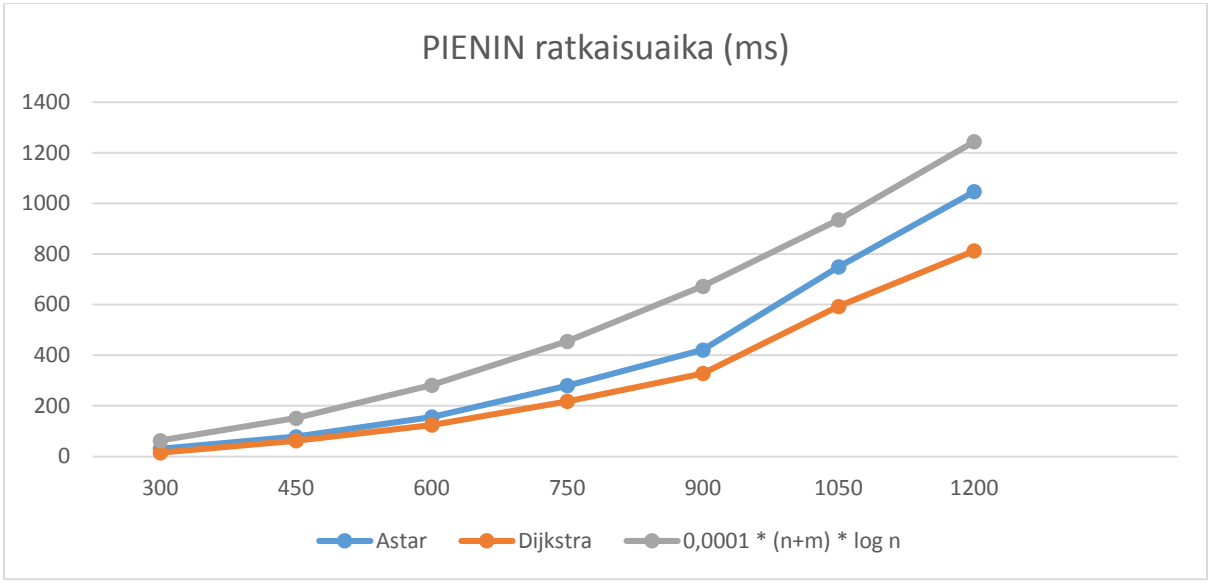


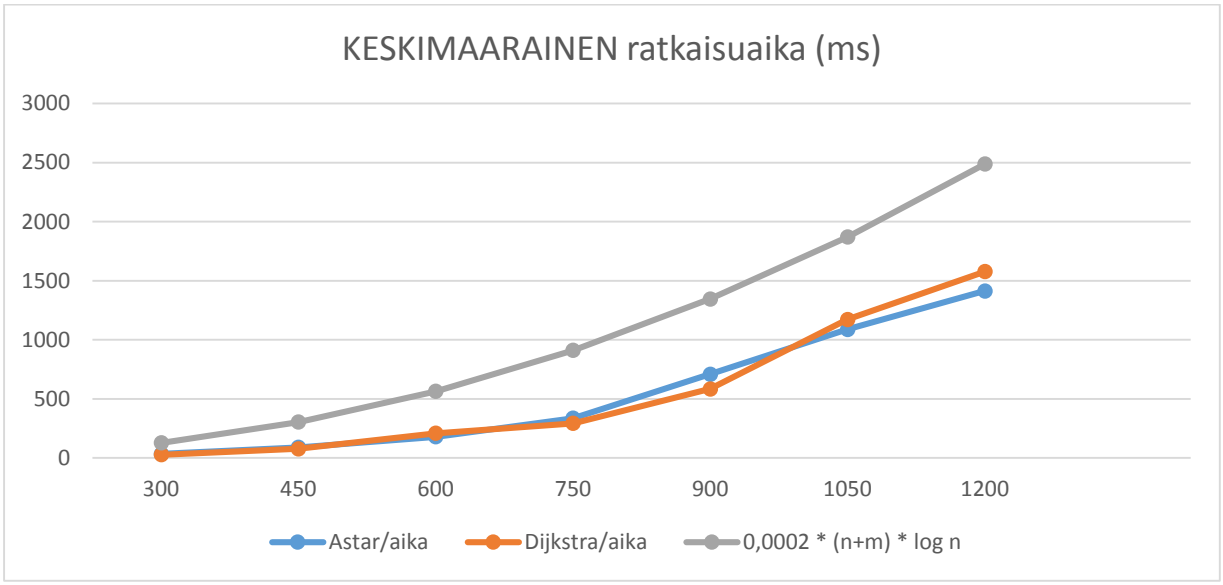


Testisarja esteA: Varioitiin lähtötiedoston kokoa. Vaaka-akselillla kartan koko pikselimäärä vaakasuunnassa (pikselin määrä pystysuunnassa on vaakapikselit*2/3). Tässä Astarille (Manhattan-etäisyys heurestiikkafunktiona) epäedullisessa kartassa Dijkstra oli nopeampi. Muutoin Astar oli aina nopeampi.

Tuloskuvat. Vasemmalla Dijkstra ja oikealla Astar.

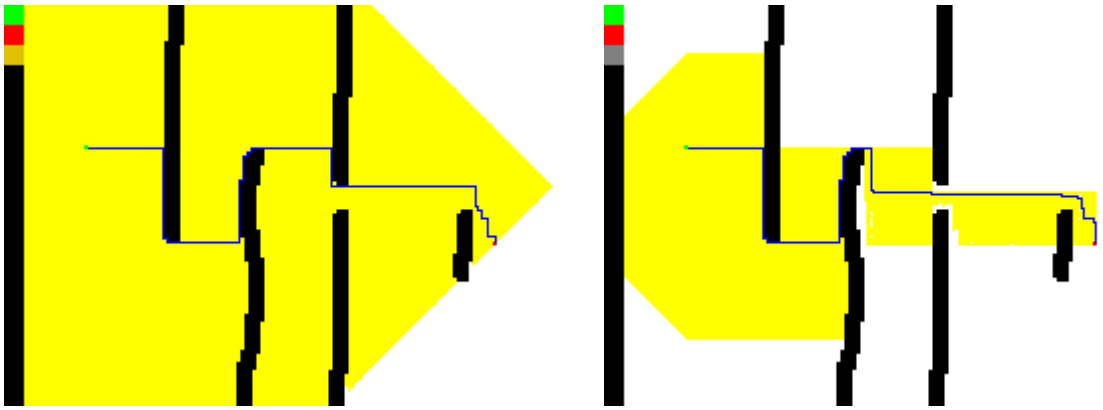


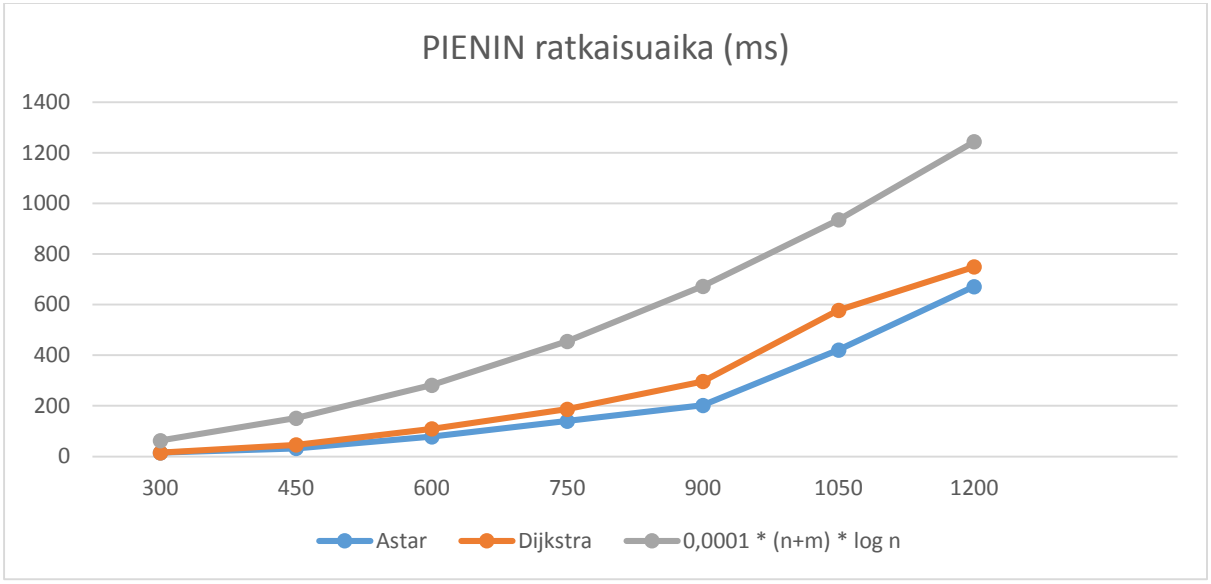


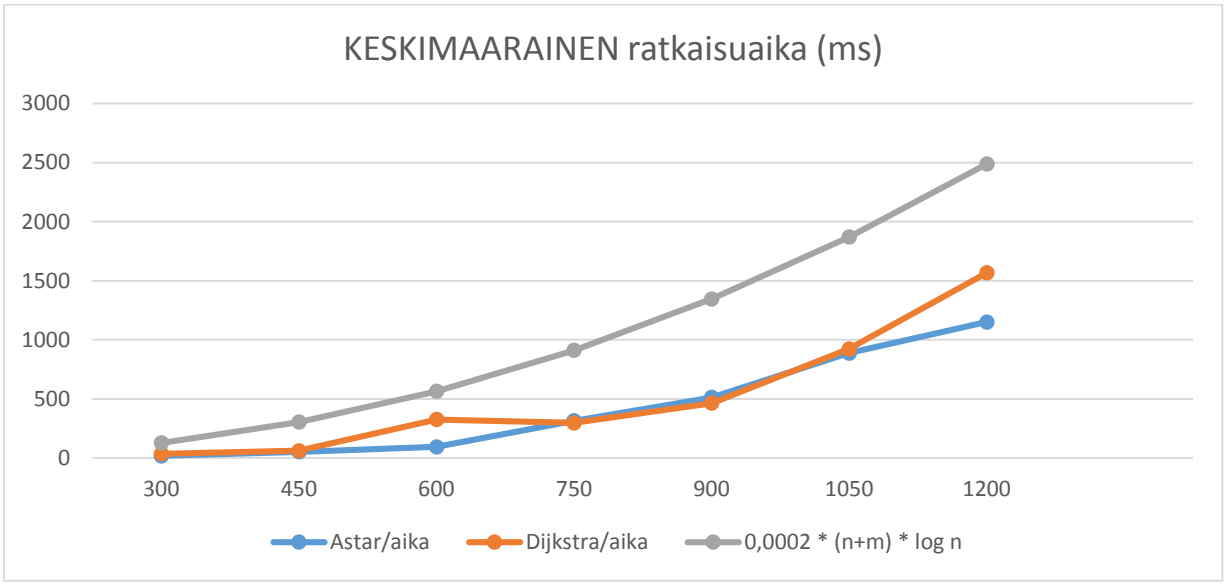


Testisarja esteB: Varioitiin lähtötiedoston kokoa. Vaaka-akselillla kartan koko pikselimäärä vaakasuunnassa (pikselin määrä pystysuunnassa on vaakapikselit*2/3). Tässä Astarille (Manhattan-etäisyys heurestiikkafunktiona) edullisessa kartassa Astar oli nopeampi

Tuloskuvat. Vasemmalla Dijkstra ja oikealla Astar.

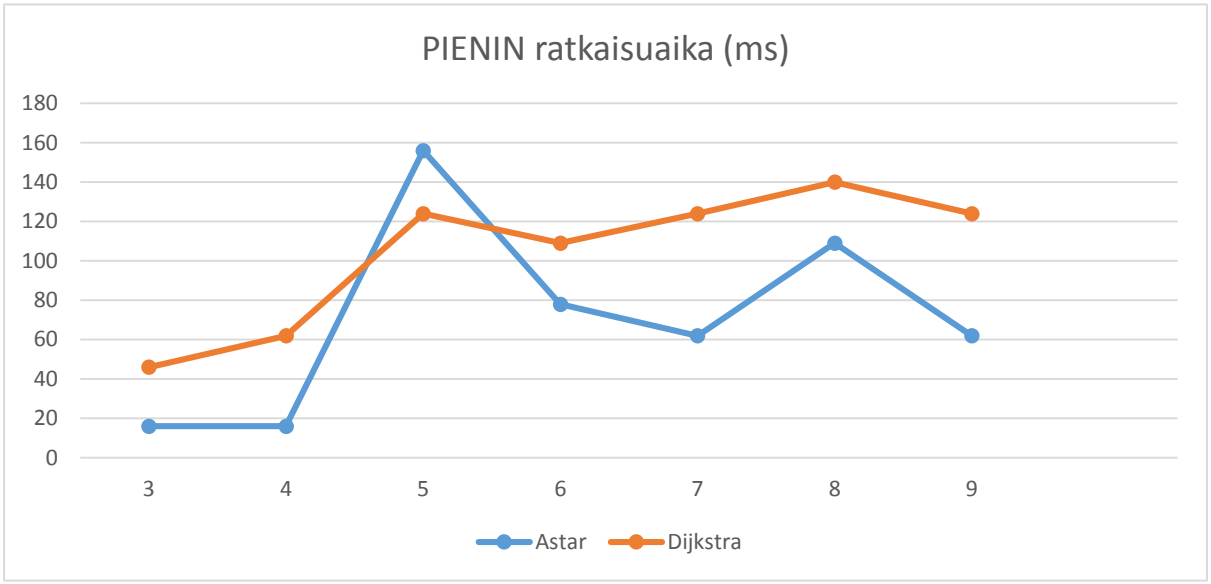


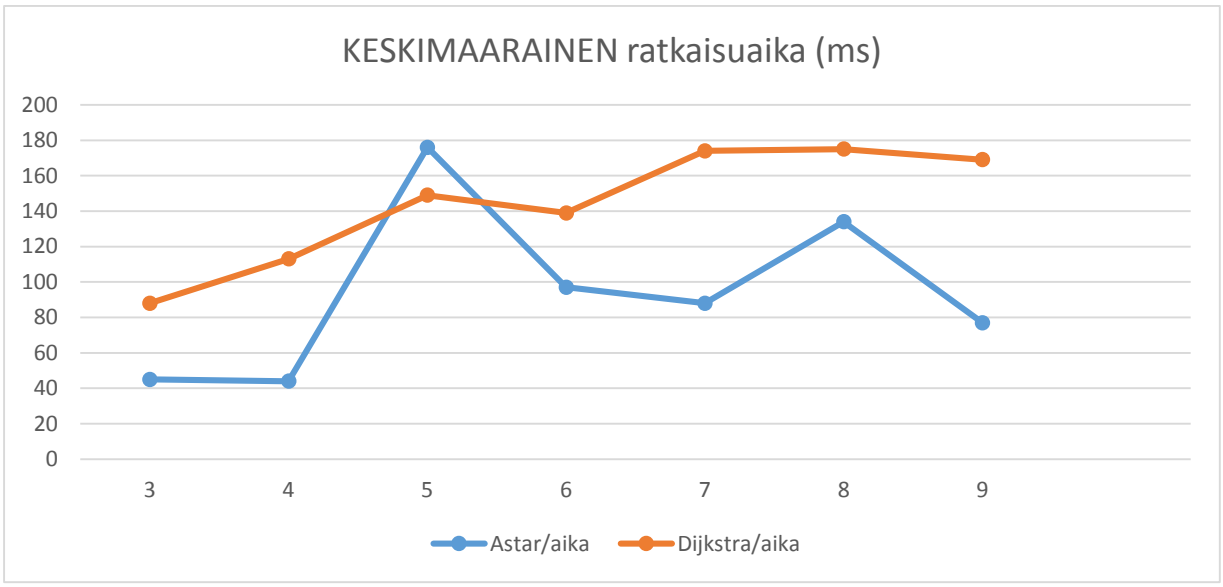




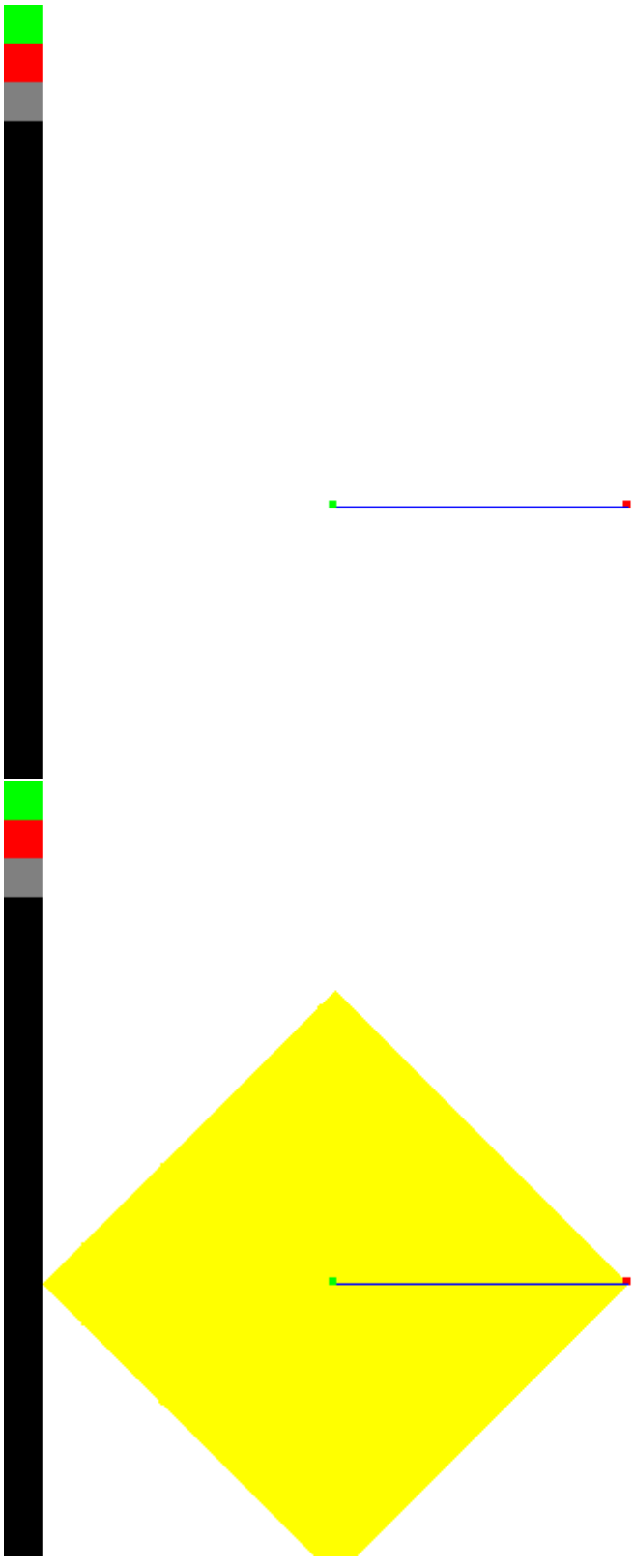
Testisarja 600x400kartta: Varioitiin lähtötietokartan esteiden ja vaikeakulkuisten alueiden muotoa. Vaaka-akselillla kartan numero. Vain testisarja esteA:n mukaisessa tapauksessa (nro 5) Dijkstra oli nopeampi, muutoin Astar.

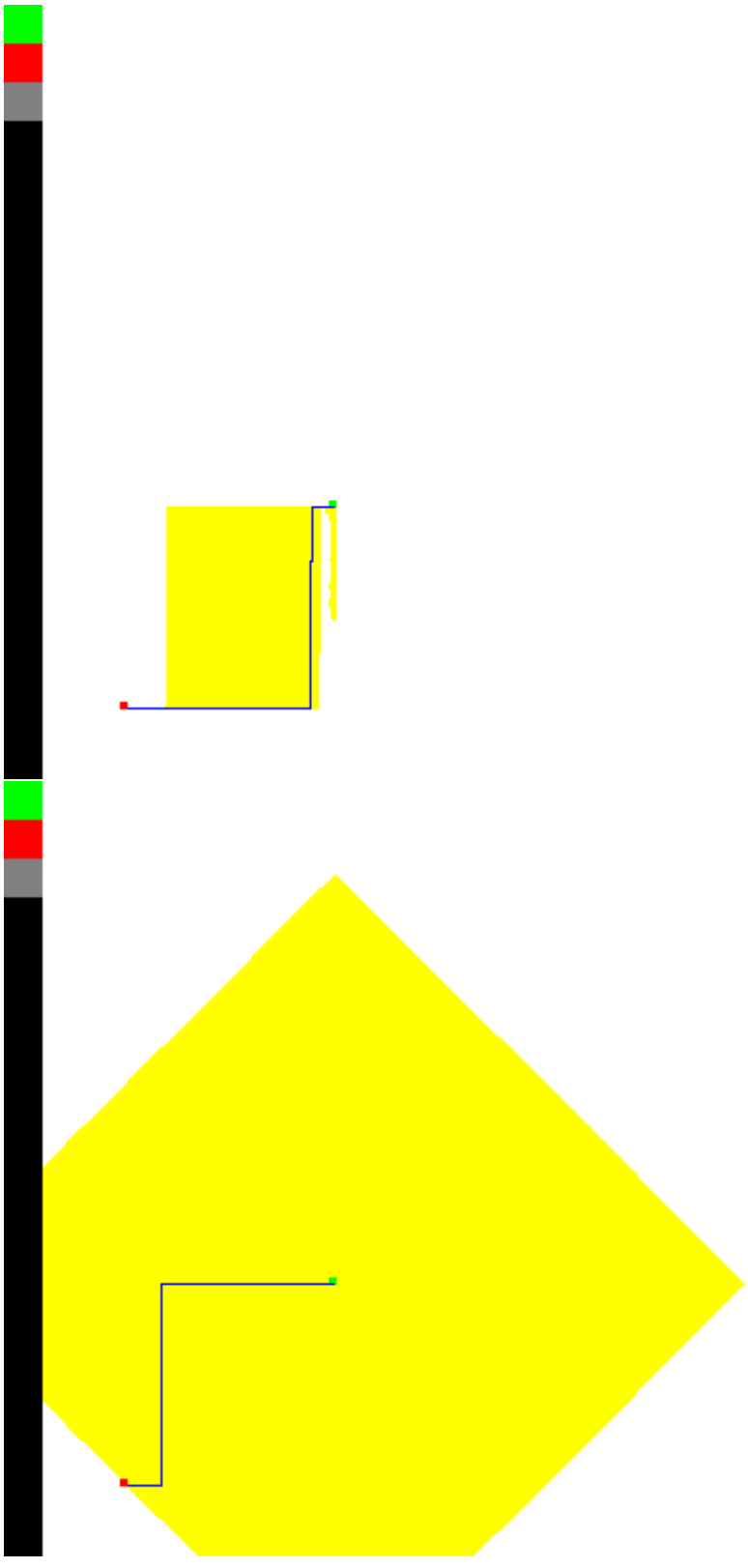
Tulokset



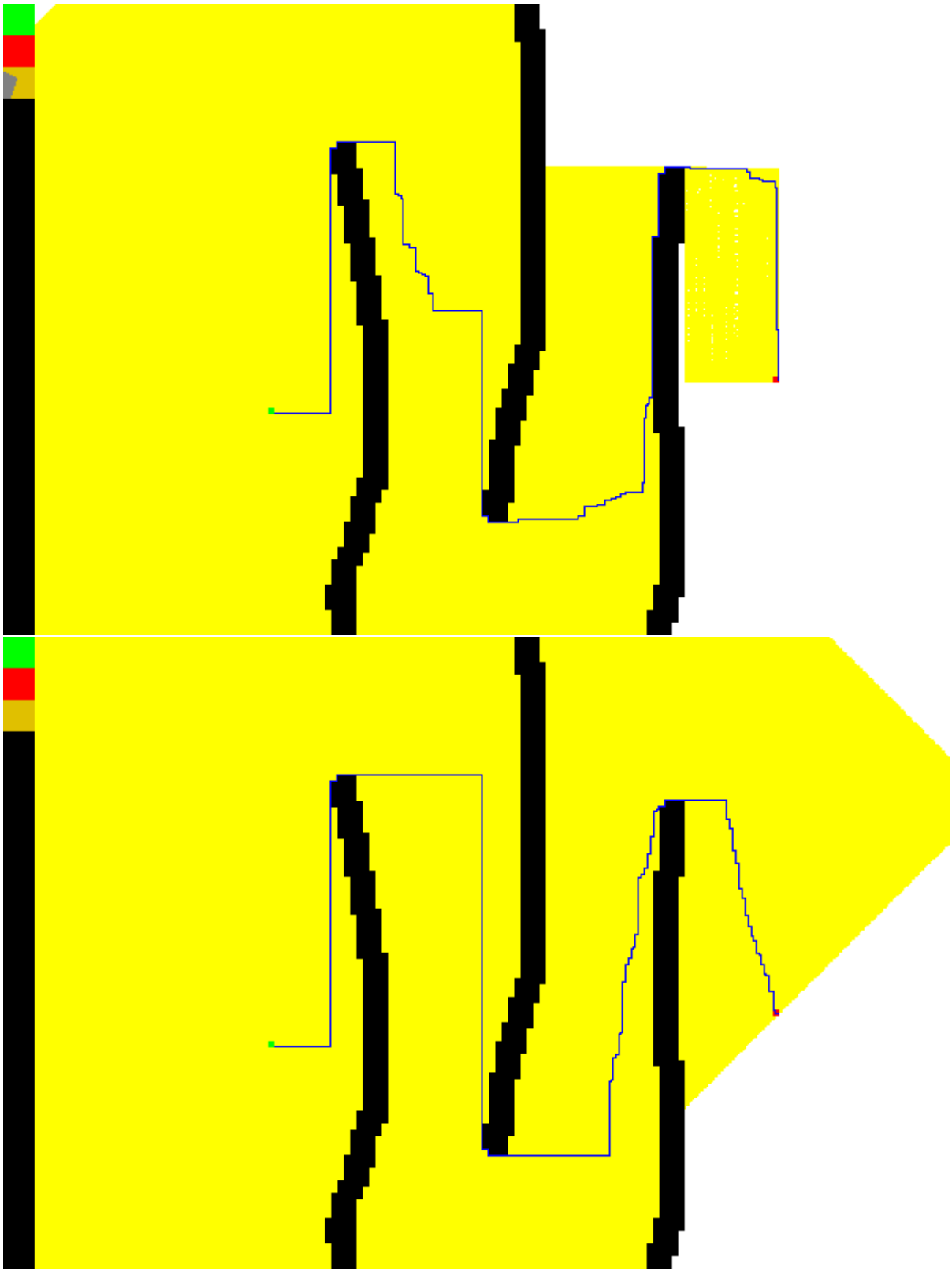


Tuloskuvat esitetty seuraavilla sivuilla numerojärjestyksessä. Kullakin sivulla ylempänä Astar ja alempana Dijkstra.

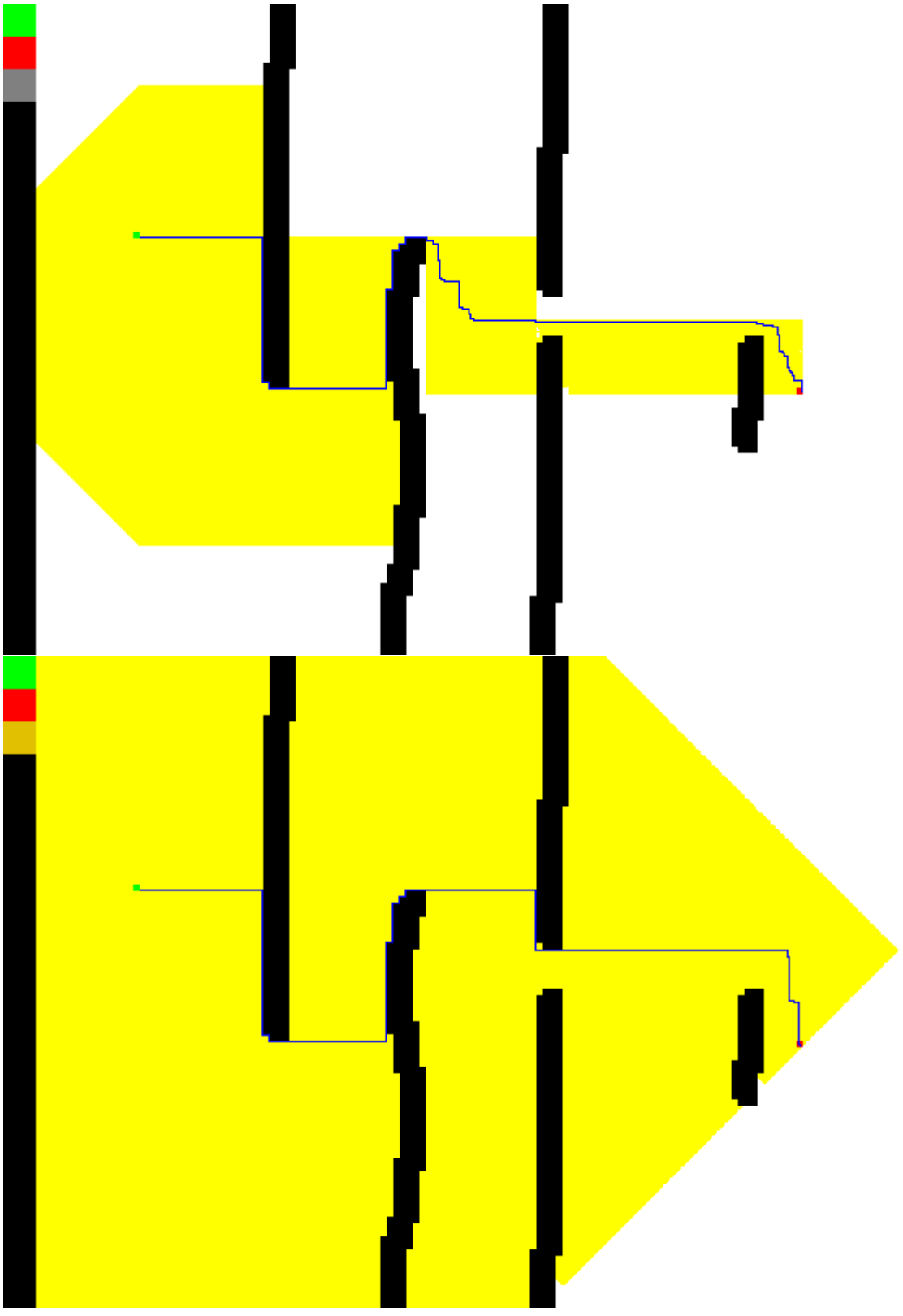




5



6



7