Toteutusdokumentti

Yleisrakenne:

Ohjelmaa suorittaessa luodaan ensin menu-ikkuna, johon käyttäjä syöttää haluamansa labyrintin koon. Ohjelma toimii suurillakin syötteillä, mutta ulkoasun ja käytännöllisyyden vuoksi syötteelle on annettu rajaksi [2-40]. Kun annettu koko täyttää ehdon, luodaan solmut (myöh. "maapalat"). Koska *maapalat* luodaan jo suorituksen alkuvaiheessa (ennen haun käynnistymistä), itse reitin haku toteutuu nopeammin.

Graafiseen käyttöliittymään luodaan "nappulat" edustamaan maapaloja, joita hiirellä painamalla asetetaan tiettyihin toiminnallisuuksiin. Nämä "nappulat" on talletettu 2D- taulukkoon, joten niiden haku koordinaattien perusteella toteutuu vakioajassa. Hiiren vasemmalla painikkeella painaessa *maapala* muuttuu joko seinäksi tai läpäistäväksi riippuen siitä, missä tilassa solmu on ennen painallusta. Hiiren oikealla painikkeella painaessa *maapala* muuttuu alku- tai loppupisteeksi riippuen sen tilasta ennen painallusta.

Kun yksi alku- ja loppupiste on alustettu, voidaan lyhimmän reitin etsintä käynnistää. Etsintä on toteutettu A*- algoritmilla käyttäen kahta listaa "avoinLista" ja "suljettuLista." Avoin lista on toteutettu käyttäen minimikeko tietorakennetta ja suljettu lista on puolestaan toteutettu käyttäen linkitettyä listaa.

Algoritmi käy avoimen listan läpi ja etsii sieltä *maapalan*, jolla on pienin kokonaisarvo *(kokonaisarvo = heuristinen arvo + (liikkumiskustannus(10) * tarvittujen liikkeiden määrä*). Kun haettu *maapala* on löydetty, asetetaan sen naapureille oikeat kokonaisarvot, tieto vanhemmasta (parent-solmu) ja siirretään ne avoimelle listalle. Kun kaikki naapurit (ei diagonaaliset) on asetettu avoimelle listalle, siirretään jo käsitelty yksilö "suljetulle listalle."

Edellä mainittua tapahtumasarjaa suoritetaan niin kauan, kunnes avoin lista on tyhjä eli uusia solmuja ei ole käsiteltävänä tai kun loppupiste on saavutettu. Jos loppupiste on löytynyt, saadaan lyhin reitti esiin kutsumalla while- loopissa saavutetun pisteen vanhempaa, kunnes vanhemmuus on arvossa null eli alkupiste on saavutettu.

Ohjelman logiikan aika- ja tilavaatimukset:

Seuraavaksi avaan tuottamani algoritmin tehokkuus- ja tilavaatimuksien toteutumista ohjelman logiikan kannalta. Tämä kattaa labyrintin alustuksen sekä itse reitin etsimisen.

Maapalojen luonti:

Ohjelma luo maapalat ajassa O(n), sillä jokainen *maapala* täytyy käydä lävitse luontiprosessissa.

Näin ollen vaadittu aika on riippuvainen syötteen eli labyrintin koosta: O(n)
Tilavaativuus on myöskin riippuvainen syötteen koosta, joten sekin on: O(n)
Koodissa oleva muuttuja *koko* on labyrintin sivun pituus, joten syöte on todellisuudessa *(koko x koko)*, mistä johtuu vaativuus O(n) eikä O(n^2). Jokainen koordinaatti käydään läpi kerran eikä kahta kertaa.

Maapalojen alustus:

Kun maapalat on luotu, rakennetaan seinät ja asetetaan lähtö- ja loppupisteet.
Nämä operaatiot tapahtuvat vakioajassa, sillä maapaloilla on omat muuttujat sille, ovatko ne seiniä, vai ei. Tämän lisäksi Maapalarekisterille syötetään tiedot lähdön ja lopun koordinaateista, jotka myös ovat vakioaikaisia operaatioita.

Sijoitus operaatiot ovat vakioaikaisia, joten operaatiot ovat kokonaisuudessaan vakioaikaisia.

Tämän jälkeen *maapaloille* asetetaan heuristiset arvot sen mukaan, miten kaukana ne ovat loppupisteestä x-koordinaatin ja y-koordinaatin suhteen. Tämä lasketaan kaavalla*(h(maapala) = |loppuX - maapalaX| + |loppuY - maapalaY|)*. Jokainen alkio käydään läpi kerran, joten aikavaatimuksena on O(n), kuten ensimmäisessä kohdassa on selitetty. Heuristisen arvon laskeminen sekä sijoitus kyseiselle maapalalle ovat vakioaikaisia operaatioita, joten metodin vaativuudeksi tulee O(n).

Maapalojen alustus on vaativuudeltaan siis O(n)- luokkaa.

Avoimen listan läpikäynti:

Avoin lista on toteutettu käyttämällä minimikekoa, joten kaikki sen toiminnallisuudet tapahtuvat ajassa O(*log n*). Koska keon toiminnallisuudet ovat riippuvaisia ainoastaan puun korkeudesta, kutsuja tapahtuu pahimmassa tapauksessa puun korkeuden verran. Keko on melkein täydellinen binääripuu, joten korkeus n- alkioisessa puussa on *log n*.

Osa toteuttamistani toiminnallisuuksista keossa, kuten korkeuden haku sekä alkion etsintä sijainnin perusteella, ovat triviaaleja toiminnallisuuksia, joten ne toimivat vakio ajassa O(1).

Lyhimmän reitin etsintä:

Jokainen alkio (maapala) asetetaan enintään kerran avoimelle listalle. Tämän lisäksi jokaista alkiota avoimella listalla tarkastellaan enintään kerran ennen kuin suoritus lopetetaan. Tästä johtuen pahin

tapaus, mikä voisi tapahtua vaatisi, että kaikki alkiot käytäisiin läpi ennen lopun löytymistä eli jokaiselle alkiolle toteutettaisiin kekoon lisäys, jolloin aikavaativuudeksi tulisi O(*n log n*), missä *n = labyrintin maapalojen määrä*.