Määrittelydokumentti
A*- ja Dijkstra-reitinhakualgoritmien vertailulle

Tietorakenteiden harjoitustyö
Santeri Martikainen
014152902
Helsingin yliopisto
Kesä 2013

Ratkaistava ongelma ja harjoitustyön tausta

Kevään 2013 ensimmäisessä periodissa tein ohjelmoinnin harjoitustyönä pienen pelin, jonka teemana oli pubi ja siellä s(e)ik(k)ailu. Se on toteutettu merkkigrafiikalla ja asettuu ”roguelike”-genreen, eli kuvakulma on suoraan ylhäältä ja jokainen pelikentän elementti kuvataan yhdellä ASCII-merkillä.

Tarkoitukseni on elävöittää tätä peliä soveltamalla siihen reitinetsintäalgoritmeja, jotta pelin ei-pelaajahahmot saadaan liikkumaan reaalimaailmaa muistuttavalla tavalla. Eli esim. pubin asiakas hakeutuu tuolilta baaritiskille ostamaan juotavaa, palaa sen kanssa takaisin pöytään, käy myöhemmin vessassa jne. Tällaista toiminnallisuutta ei ole oikein mieltä ns. 'kovakoodata', vaan ennemmin toteuttaa hahmojen siirtymiset kohteisiin soveltamalla reitinhakualgoritmeja.

Luon tässä harjoitustyössä siis algoritmien *A** ja *Dijkstra* toteutuksen, sekä näiden apuna minimikekoperiaatteella toimivan prioriteettijonon. Valitsin nämä algoritmit, koska katson niiden olevan perustyökaluja, joiden onnistunut soveltaminen luo pohjaa kenen tahansa aloittelevan tietojenkäsittelijän ammattitaidolle.

Harjoitustyössä luotavat algoritmit ja tietorakenteet

Dijkstra

”Dijkstran algoritmi etsii graafille lyhyimmän polun yhdestä pisteestä kaikkiin muihin pisteisiin. Algoritmi toimii suunnatuilla graafeilla, joiden viivojen painot ovat ei-negatiivisia. Algoritmia käytetään muun muassa tietoliikenneverkkojen reitityksessä.” [1]

Näin siis Wikipedia kertoo meille. Tämän projektin kohdalla luonnehtisin ko. algoritmia ennemminkin näin:

Dijkstra pyrkii löytämään maalin käymällä läpi jo tutkittujen solmujen naapureita, sekä näiden naapureita jne. aina siihen asti, kunnes haluttu solmu on löytynyt.

A*

”A*-algoritmi (lausutaan A tähti) on tekoälyssä käytetty algoritmi ratkaisun etsimiseen hakupuusta. Sen tarkoituksena on evaluoida lehtisolmuja funktion $f(n)=g(n)+h(n)$ avulla, missä $g(n)$ kuvaa kustannusta saavuttaa tietty solmu ja $h(n)$ on kustannusarvio solmusta maalitilaan. Tällöin f approksimoi kustannusta lähtösolmusta maalisolmuun. A*-algoritmi on optimaalinen jos h on luvallinen. Tämä tarkoittaa sitä, että h ei koskaan yliarvioi kustannusta saavuttaa maalisolmu.” [2]

A* siis pyrkii tutkimaan ensisijaisesti sellaisia solmuja, joiden laskennallinen arvo (etäisyys lähtöpisteeseen + arvioitu etäisyys maaliin) antaa ymmärtää kyseessä olevan potentiaalinen reitti kohteeseen. A*:n ero Dijkstran algoritmiin on siinä, että A* siis nojaa myös arvioon etäisyydestä kohteeseen.

Priority queue

Jotta sekä A*, että Dijkstra voivat toimia oikein, on niiden käsiteltävä näkökulmastaan aina otollisin alkio. Tämä tarkoittaa käytännössä käsiteltävien alkioiden pitämistä sellaisessa järjestyksessä, että niistä voidaan helposti poimia aina se, mikä pitäisi käsitellä seuraavaksi. Tämä toteutetaan tässä harjoitustyössä itse tehdyllä prioriteettijonolla.

Ohjelman syötteet

Pelikenttä luodaan lukemalla tekstitiedosto ja muodostamalla siinä olevien merkkien perusteella pelimaailman muodostavat objektit.

Pelissä esiintyvät ihmishahmot luodaan koodissa annettujen lukumäärien perusteella eri puolille pelikenttää, pitäen kuitenkin huolen loogisuudesta. Eli esimerkiksi asiakkaita ei luoda baaritiskin taakse, tai miespuolisia asiakkaita naistenvessaan jne.

Reitinhakualgoritmeja käyttävät ei-pelaajahahmot simuloivat pubiasiakkaiden muuttuvia tarpeita hakeutumalla tarpeen mukaan haluttujen kenttäobjektien luokse. Tarkoitus on tehdä tämä osin satunnaisesti (ei väliä millä wc-pytyllä käy) ja osin ei-satunnaisesti (juoman hakenut asiakas hakeutuu istumaan samalle tuolille missä oli aiemmin, ellei sitä ole viety jne).

Aika- ja tilavaativuustavoitteet

Dijkstran tilavaativuus on O|V| ja aikavaativuus O((|E| + |V|) log |V|).

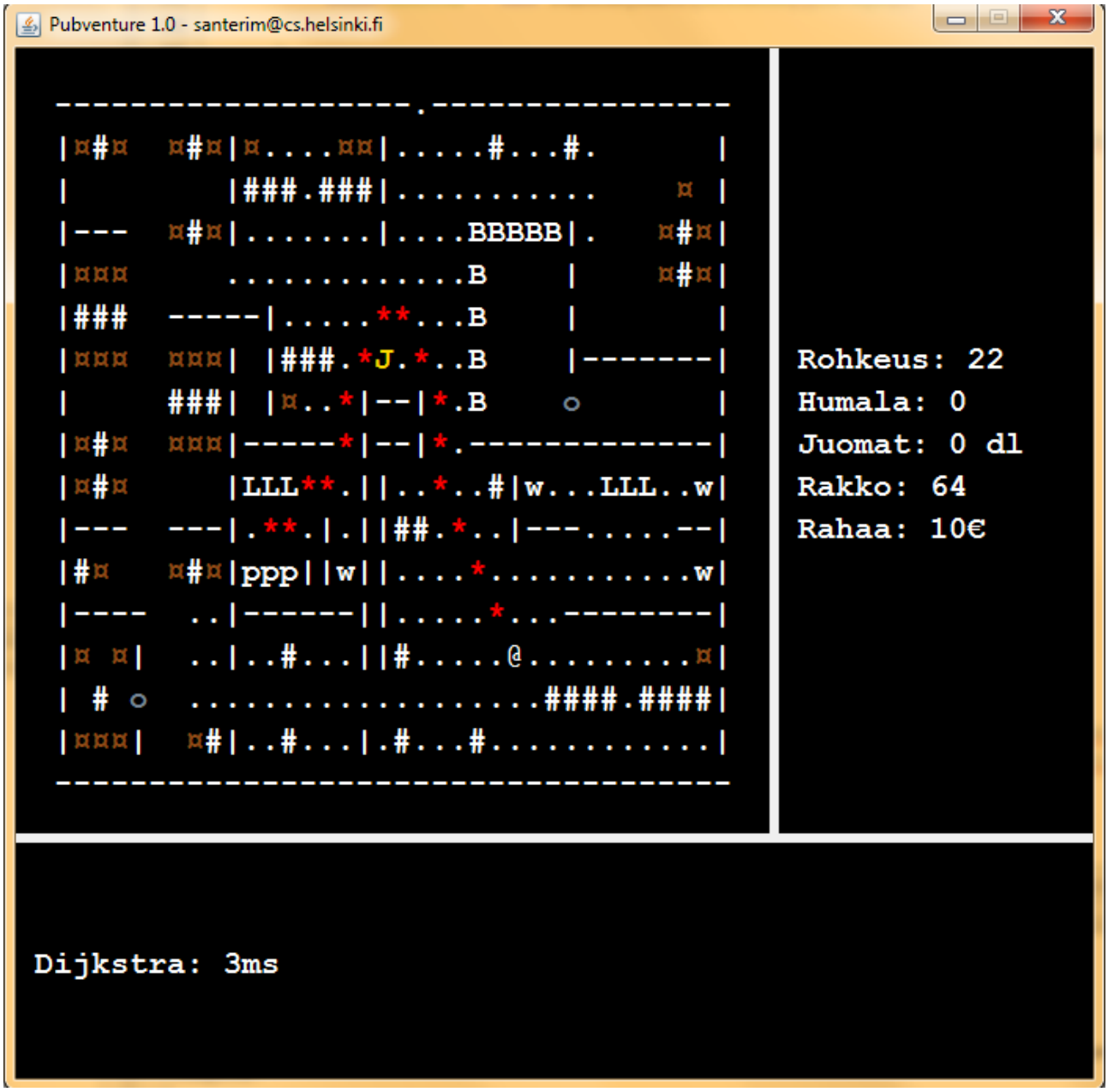
A*-algoritmin tilavaativuus on sama, mutta pahimman tapauksen aikavaativuus sama kuin Dijkstralla. Yleensä A* löytää siis reitin Dijkstraa nopeammin.

Prioriteettijonon aikavaativuuden tavoite on O(log n) ja tilavaativuuden O(n).

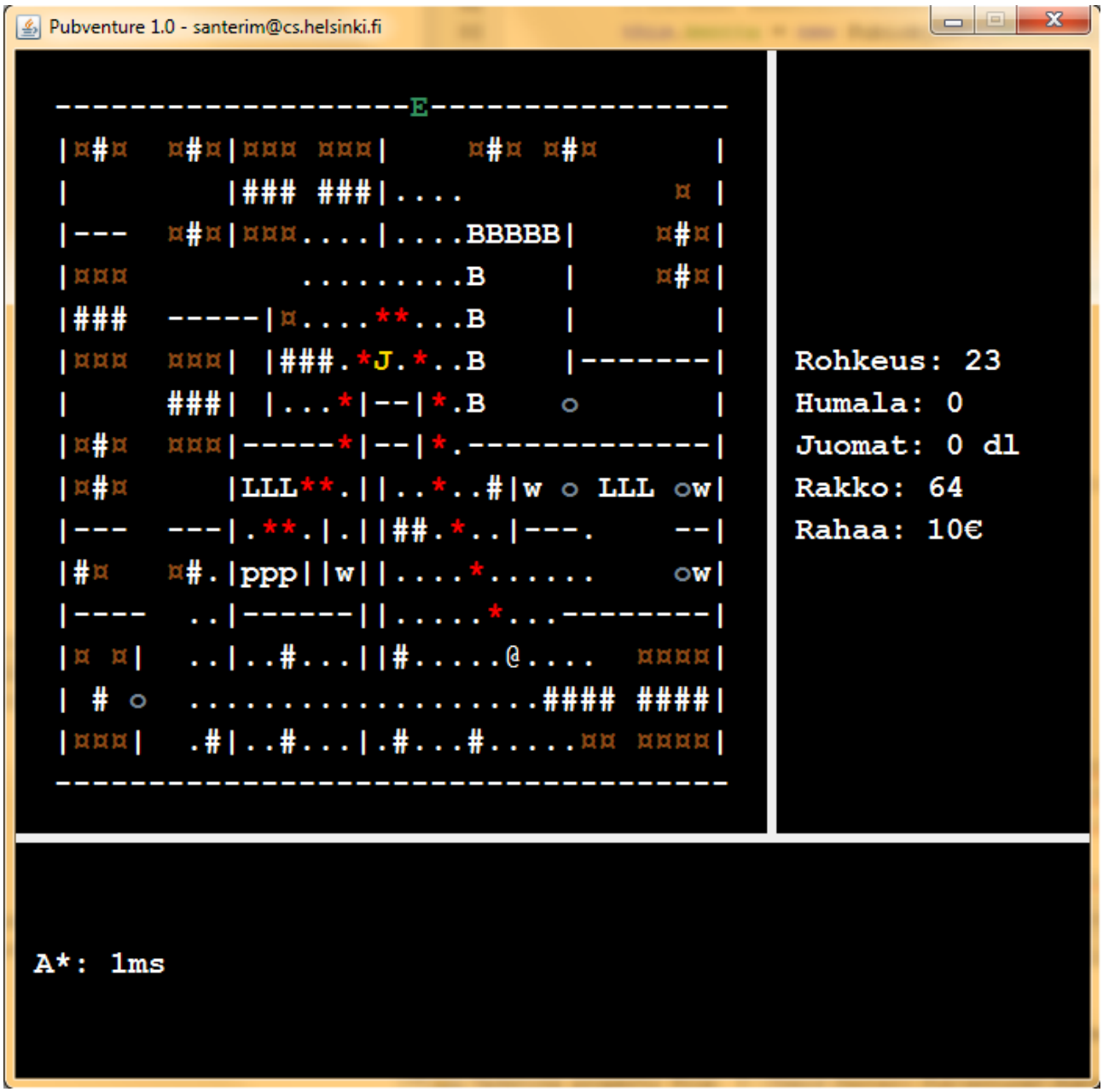
Jos ja kun prioriteettijono on toteutettu minimikeko-periaatteella, niin jonon ensimmäisen (tässä tapauksessa pienimmän halutun arvon omaavan) poiminta onnistuu logaritmisessa ajassa johtuen keon rakenteesta.

Tilavaativuus puolestaan riippuu jonoon tulevien alkoiden määrästä. Mitä suurempi kenttä, sitä enemmän voimme olettaa jonoon tulevan alkioita, ja sitä suuremmaksi jonon tilavaativuus luonnollisesti muodostuu.

Kuvitusta



Kuva 1: Kuva 1: Dijkstran löytämä reitti päähenkilöltä (@) pisuaarille (p). Valkoiset pisteet osoittavattutkitut sijainnit, punaiset tähdet löydetyn reitin.



Kuva 2: Kuva 2: A-algoritmin löytämä reitti samoilla lähtö- ja maalipisteillä. Symbolit samat kuinKuvassa 1.*

Lähteet

[1] Wikipedia: Dijkstran algoritmi (10.5.2013)
[2] Wikipedia: A*-algoritmi (10.5.2013)