Tampereen yliopisto

Mikko Metsola

HEURISTIIKKA REITINHAUSSA JA SEN VAIKUTUS A*-ALGORITMIN SUORITUSKYKYYN

Informaatioteknologia ja viestintä
Kandidaatintyö
Toukokuu 2019

TIIVISTELMÄ

Mikko Metsola: Heuristiikka reitinhaussa ja sen vaikutus A*-algoritmin suorituskykyyn
Kandidaatintyö
Tampereen yliopisto
Tietotekniikka
Toukokuu 2019

A*-algoritmi on yleinen reitinhakualgoritmi, joka hyödyntää heuristiikkaa. Heuristiikka on arvio kahden pisteen välisestä etäisyydestä ja sen tavoitteena on nopeuttaa reitinhakua. Heuristiikan liiallinen painottaminen voi kuitenkin johtaa huonoihin tuloksiin. Tässä työssä tutkitaan A*-algoritmin käyttäytymistä neljällä eri heuristiikalla ja erilaisilla painotuksilla. Heuristiikan painottamisen on tarkoitus korostaa heuristiikan vaikutusta reitinhaussa ja nopeuttaa hakua entisestään.

A*-algoritmin suorituskykyä tarkkailtiin kaksiulotteisessa ruudukossa JavaScript-ohjelmalla reitinhakuun soveltuvaa kirjastoa käyttäen. Tutkielman tuloksissa tulee esiin heuristiikkojen väliset ainutlaatuiset piirteet sekä niiden muutos heuristiikan painottamisen myötä. Vaikka Manhattan-etäisyydellä laskettu heuristiikka on testikohteelle tarkoitettu tapa mitata heuristiikkaa, ovat sen erot muihin heuristiikkoihin nähden vähäisiä.

Avainsanat: heuristiikka, A*-algoritmi, reitinhaku

SISÄLLYSLUETTELO

LYHENTEET JA MERKINNÄT

h(n)	heuristiikkafunktio
GBFS-algoritmi	engl. Greedy best-first search, heuristiikkaa käyttävä reitinhakualgoritmi
NPC	engl. Non-playable character, pelihahmo, jota pelaaja ei ohjaa

1. JOHDANTO

Yksinkertaisimmillaan reitinhaku on kahden pisteen välisen tehokkaimman reitin laskemista. Tänä päivänä reitinhakualgoritmeja käytetään laajasti esimerkiksi GPS-laitteiden navigoinnissa ja videopelien tietokoneohjattujen hahmojen ohjaamisessa. Reitinhakualgoritmit voivat kuitenkin ajautua ongelmiin hakuavaruuksien laajetessa. Hakuun käytetty aika saattaa kasvaa hyvin nopeasti, mikäli ratkaisua ei löydetä välittömästi suuresta hakualueesta. Perinteisesti reitinhakualgoritmit etsivät polkua nopein operaatioin noudattaen yksinkertaisia sääntöjä, mutta suurilla avaruuksilla joudutaan usein kuluttaa paljon aikaa turhien vaihtoehtojen käsittelyyn, joka heikentää algoritmin tehokkuutta. Algoritmeissa tehokkuudella voidaan viitata ajankäytön lisäksi muun muassa algoritmin saaman polun pituuteen, käytettyjen operaatioiden määrään tai jopa käytetyn muistin määrään.

A*-algoritmi on yksi merkittävimmistä hakualgoritmeista, ja sitä käytetään hyvin laajasti reitinhaussa [1]. Se hyödyntää hauissaan heuristiikkaa, joka käytännössä tarkoittaa sitä, että se on tietoinen halutun päämäärän sijainnista. Sen tavoitteena on vähentää haettavia solmuja suosimalla vaihtoehtoja, jotka vaikuttavat päämäärän saavuttamisen kannalta lupaavimmilta. Tämän ansiosta A*-algoritmi voi nopeuttaa hakua hyvin merkittävästi, sillä sen ei tarvitse kuluttaa niin paljon aikaa epäolennaisten vaihtoehtojen käsittelemiseen. Heuristiikasta riippuen haun tulos ei toisaalta ole aina optimaalinen, eli algoritmin saama tulos ei ole välttämättä paras mahdollinen vaihtoehto.

Tämän kandidaatintyön tavoitteena on tutkia reitinhakualgoritmeissa hyödynnettyä heuristiikkaa ja tarkkailla A*-algoritmin käytöstä erilaisilla heuristiikoilla. Työssä esitellään erilaisia keinoja laskea heuristiikka kahden pisteen välille. Näitä heuristiikkoja testataan kaksiulotteisessa ruudukossa tarkkaillen eri menetelmien suorituskykyä. Tässä työssä suorituskykyä mitataan algoritmin saaman ratkaisun optimaalisuuden lisäksi myös algoritmin suorittamien operaatioiden perusteella.

Työn luvussa 2 esitetään mitä heuristiikka on, kuinka sitä voidaan hyödyntää reitinhaussa sekä tietoisten ja perinteisten hakualgoritmien välisiä eroja. Luku 3 syventyy erilaisiin tapoihin arvioida kahden pisteen etäisyyttä. Luvussa 4 käsitellään A*-algoritmia ja käydään tarkemmin läpi, miten algoritmi käyttäytyy ja miten heuristiikka vaikuttaa A*-algoritmin toimintaan. Luku 5 syventyy työssä tarkasteltavaan testiongelmaan ja sen mielenkiintoisiin ominaisuuksiin. Luku 6 kokoaa työssä havaitut tulokset yhteenvedolla.

2. HEURISTIIKKA REITINHAUSSA

Heuristiikka juontaa juurensa filosofiaan, jossa se määritellään ongelman ratkaisun löytämiseksi [2, 3]. Tietotekniikassa heuristiikkaa käytetään hakualgoritmeissa nopeuttamaan ongelmanratkaisua, kun tavallisin menetelmin saadut tulokset eivät ole tyydyttäviä [1]. Heuristisilla funktioilla pyritään karsimaan reitinhaun operaatioiden määrää priorisoimalla vaihtoehtoja, jotka vaikuttavat oleellisilta maalin tavoittelun kannalta. Mikäli maalia ei ensisijaisilla hauilla olla saavutettu, laajennetaan hakua vähitellen kattamaan vaihtoehtoja, jotka eivät olleet ensisijaisen tärkeiltä, kunnes jokin polku löydetään maaliin edellyttäen, että maaliin on edes yksi polku olemassa.

2.1 Reitinhaku algoritmisena ongelmana

Reitinhaussa käytetään yleisesti käsitettä verkko (*network*) kuvaamaan solmuista (*node*) ja kaarista (*edge*) muodostuvaa joukkoa. Solmu on verkossa oleva itsenäinen olio. Kaari yhdistää kaksi solmua toisiinsa siten, että niiden välillä voidaan kulkea. Kaarella on myös paino (*weight*), joka määrittää kaaren läpi kulkemisen kustannuksen. Reitinhaun tavoitteita voivat olla vaikkapa lyhimmän tai nopeimman polun (*path*) löytäminen. [4] Tästä yksinkertaisena esimerkkinä voidaan ajatella valtiota, jonka kaupungit ja tiet muodostavat verkon. Jokainen kaupunki on itsenäinen solmu tässä verkossa, ja jokainen kahden kaupungin välillä kulkeva tie on kaari, jonka pituus on kaaren paino.

Monilla ohjelmilla on ongelmana tehokkaan polun löytäminen verkosta kahden annetun solmun välille. Tämän polun löytämiseksi käytetään reitinhakualgoritmeja käsittelemään verkossa olevia solmuja yksitellen. Reitinhakualgoritmit etenevät laajentaen jotain avointa solmua. Avoimia solmuja ovat siis kaikki ne solmut, joihin tiedetään olevan jokin polku alkusolmusta, mutta joita ei olla vielä käsitelty. Esimerkiksi monien reitinhakualgoritmien suorituksen alussa laajennetaan alkusolmu, sillä alkupisteen tiedetään aina olevan reitti. Kun alkusolmu on laajennettu, avoimia solmuja ovat nyt kaikki ne solmut, jotka ovat kaarella yhdistettyjä alkusolmuun. Näitä avoimia solmuja laajennetaan vähitellen algoritmin määrittämässä järjestyksessä, kunnes maalisolmu tavoitetaan. Kun maali löydetään, palataan takaisin alkuun samaa reittiä pitkin muistaen mitä polkua pitkin maaliin kuljettiin.

2.2 Reitinhakualgoritmi polun etsintään

Perinteiset reitinhakualgoritmit, jotka eivät hyödynnä heuristiikkaa, toimivat tietämättä maalisolmun sijaintia. Ne siis käyvät systemaattisesti läpi solmuja alkusolmusta lähtien noudattaen tiettyjä sääntöjä, kunnes löytävät maalisolmun. Tämä johtaa usein siihen, että nämä perinteiset algoritmit joutuvat tekemään paljon turhaa työtä löytääkseen parhaan ratkaisun. Perinteisistä reitinhakualgoritmeista tunnetuin on Dijkstran algoritmi [1, 5]. Se toimii käsittelemällä aina alkusolmusta lähintä solmua, mitä ei olla vielä käsitelty. Tämän ansiosta Dijkstran algoritmi löytää verkon mille tahansa kahdelle pisteelle optimaalisen reitin virheettä. Tästä huolimatta sen suorituskyky heikkenee erittäin nopeasti hakuavaruuden laajetessa. [1, 6]

Ahnas paras ensin –hakualgoritmi (*Greedy best-first search, GBFS*) on nimensä mukaisesti käyttäytymiseltään röyhkeä hakualgoritmi [7]. Sen sijaan, että laajennettaisiin solmuja, jotka ovat lähellä alkusolmua, kuten Dijkstran algoritmissa, GBFS-algoritmi laajentaa hakuaan suoraan maalisolmua kohti valitsemalla solmun, joka on lähimpänä maalia. Yksinkertaisissa verkoissa, kuten esteettömissä verkoissa, tämä algoritmi pystyy löytämään reitin kahden pisteen välille minimaalisin kustannuksin. Tämän ansiosta ahnas paras ensin –algoritmi voi suoriutua reitinhakuongelmista vaivatta tilanteissa, missä Dijkstran algoritmi tai vaikkapa leveyshakualgoritmi joutuvat tekemään huomattavan paljon enemmän operaatioita polun etsimistä varten. GBFS-algoritmista saatetaan käyttää myös nimitystä paras ensin –algoritmi (*best-first search*), mutta tämä usein sekoitetaan toisen muun samannimisen käsitteen kanssa. [1, 6, 8]

Vuonna 1968 Hart, Nilsson ja Raphael esittivät A*-algoritmin [9], joka Dijkstran algoritmista sekä leveys- ja syvyyshakualgoritmeista poiketen hyödynsi hauissaan heuristiikkaa. A*-algoritmia sekä muita algoritmeja, jotka hyödyntävät heuristiikkaa, kutsutaan usein tietoisiksi hakualgoritmeiksi (*informed search algorithm*) [1]. Vaikka nämä algoritmit ovat ajankäytön kannalta usein huomattavasti tehokkaampia löytämään ratkaisuja suurista hakuavaruuksista kuin perinteiset hakualgoritmit, nämä tietoiset reitinhakualgoritmit saattavat joutua kuitenkin uhraamaan osan täsmällisyydestään tai tarkkuudestaan nopeampia tuloksia saavuttaakseen. Toteutuksesta riippuen heuristiikkaa hyödyntävillä ratkaisuilla ei aina välttämättä siis saada parasta ratkaisua ongelmaan, toisin kuin esimerkiksi Dijkstran algoritmilla, joka käsittelee kaikki verkon solmut.

2.3 Heuristiikka reitinhaussa

Monissa heuristiikkaa hyödyntävissä algoritmeissa, kuten A*-algoritmissa, käytetään solmujen arvon mittaamiseen funktiota $f(n)$, missä n on jokin käsiteltävän verkon solmu [1]. Funktio muodostuu kahdesta arvosta: $g(n)$ ja $h(n)$. Arvolla $g(n)$ kuvataan kuljettua matkaa alkusolmusta solmuun n, kun arvo $h(n)$ on puolestaan itse heuristiikka, eli arvio minimikustannuksesta maalisolmuun solmusta n.

$$f(n) = g(n) + h(n) \tag{1}$$

Mitä pienempi funktion $f(n)$ arvo on, sitä edullisempi solmuun n kulkeminen on. Koska tavoitteena on päästä maalisolmuun mahdollisimman pienin kustannuksin, reitinhaussa suositaan solmuja, jotka omaavat pienen $f(n)$:n arvon. [1]

Heuristiikkaa hyödyntävien hakualgoritmien käytöstä voidaan ohjata heuristiikan avulla hyvin merkittävästi [8]. Koska algoritmien käyttäytymiselle on ominaista laajentaa hakua siihen suuntaan, missä kaavan (1) funktion $f(n)$:n arvo on mahdollisimman pieni, voidaan heuristiikan avulla vaikuttaa siihen, kuinka paljon suositaan solmuja, jotka ovat suoraan lähempänä maalia. Heuristiikan tulee kuitenkin olla hyväksyttävä (*admissible heuristic*), mikäli halutaan saada optimaalisia tuloksia. Tällöin heuristiikan arvon ei tulisi ylittää todellista kustannusta maaliin [1]. Mikäli heuristiikka ylittää todellisen kustannuksen maalisolmuun, se ei ole enää hyväksyttävä ja voidaan olettaa, ettei algoritmin saama tulos ole välttämättä optimaalinen.

Heuristisen reitinhakualgoritmien toimintaa voidaan edelleen nopeuttaa painottamalla heuristiikkaa asettamalla arvolle kerroin [10]. Tällöin kaava (1) voidaan johtaa nyt muotoon:

$$f(n) = g(n) + w * h(n), \tag{2}$$

missä $w \geq 1$ on nyt heuristiikkaa painottava muuttuja. Heuristiikkaa painottamalla voidaan helposti ylittää hyväksyttävän heuristiikan arvo. Tämä johtaa siihen, että reitinhakualgoritmin ratkaisujen laskemiseen saatetaan käyttää paljon vähemmän aikaa, mutta samalla ratkaisut ovat hyvin todennäköisesti epäoptimaalisia. [10]

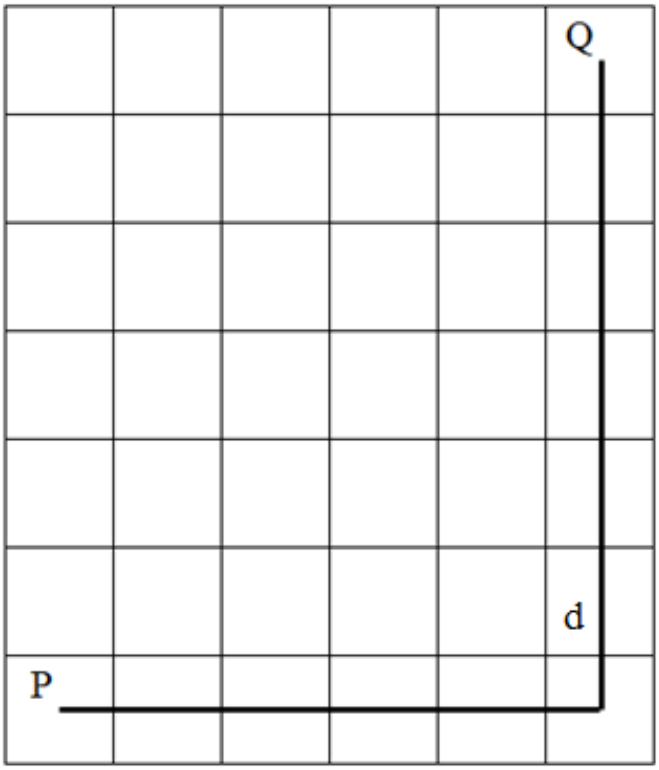
Ääritapauksissa A*-algoritmille voitaisiin antaa heuristiikan painoksi 0. Tämä johtaisi siihen, että vain $g(n)$-arvolla on vaikutusta kaavan 1 funktioon $f(n)$. Tällöin A*-algoritmi käyttäytyy lähes samoin kuin Dijkstran algoritmi laajentaen hakua alkusolmua lähimmistä solmuista vähitellen kaukaisempiin solmuihin. Vastaavasti heuristiikan arvon ollessa suuri arvolla $g(n)$ ei ole merkittävää vaikutusta funktion $f(n)$ arvoon, jolloin käytännössä heuristiikka ja sille mahdollisesti annettu painotus määräävät funktion arvon. Näissä

tapauksissa algoritmin käytös vastaa kutakuinkin ahnasta paras ensin –hakualgoritmia, suosien laajentamista suoraan kohti maalia. [1, 8]

3. HEURISTIIKAN MITTAAMINEN

3.1 Manhattan-etäisyys

Manhattan-etäisyydellä [8] (*Manhattan distance*) mitataan kahden pisteen välistä etäisyyttä tilanteessa, missä voidaan liikkua kuvan 1 kaltaisesti neljässä suunnassa eli vain pysty- tai vaakasuunnassa. Tilanteissa, joissa voidaan liikkua vain neljässä suunnassa, Manhattan-heuristiikka ei koskaan yliarvioi todellista etäisyyttä maalisolmuun edellyttäen, että heuristiikkaa ei painoteta.



Kuva 1. *Manhattan-etäisyys kahden pisteen välillä.*

Manhattan-etäisyys voidaan yleisesti laskea kaavalla

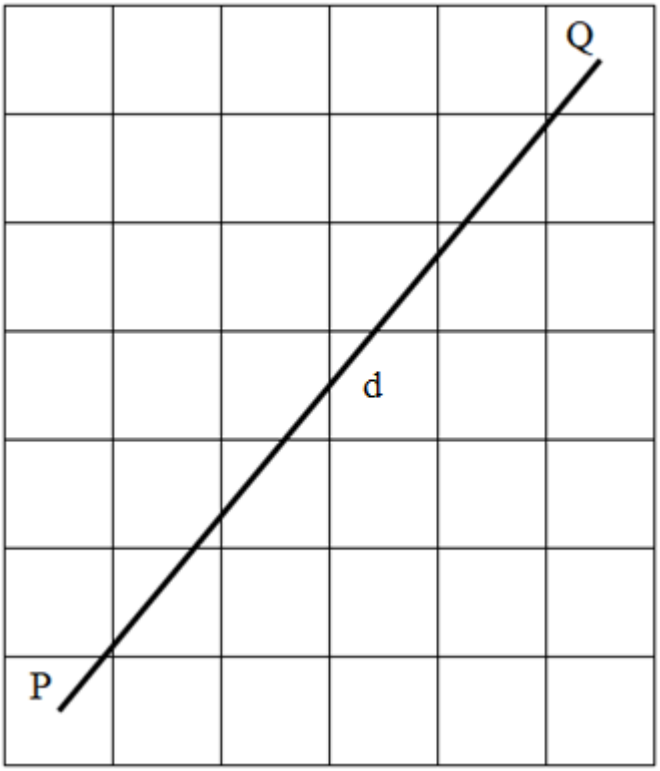
$$d = |x_Q - x_P| + |y_Q - y_P|, \tag{3}$$

missä d on laskettu etäisyys ja Q sekä P ovat laskettavat pisteet.

Manhattan-etäisyydestä käytetään myös nimitystä korttelimitta (*city-block metric*) [11], sillä etäisyyden laskeminen muistuttaa paljon korttelimaisessa kaupungissa liikkumista. Manhattan-etäisyys soveltuu heuristiikkana hyvin ruudukkokartoihin, joissa voidaan kulkea vain kardinaalisissa suunnissa.

3.2 Euklidinen etäisyys

Euklidinen etäisyys (*Euclidean distance*), arkikielessä kutsutaan myös nimellä linnuntie, on kahden pisteen välinen suora etäisyys [8, 11]. Tällä menetelmällä laskettu heuristiikka soveltuu parhaiten tilanteisiin, missä voidaan liikkua täysin vapaasti mihin suuntaan tahansa. Tästä huolimatta euklidista etäisyyttä voidaan käyttää myös muissakin tilanteissa, kuten neli- ja kahdeksansuuntaisissa ruudukoissa.



Kuva 2. *Euklidinen etäisyys pisteiden P ja Q välillä.*

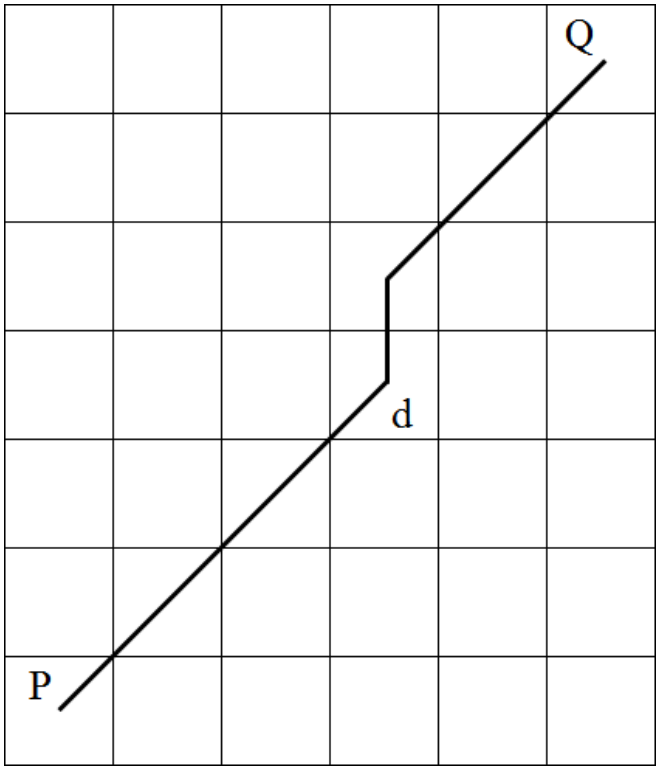
Euklidinen etäisyys lasketaan kaavalla

$$d = \sqrt{(x_Q - x_P)^2 + (y_Q - y_P)^2}. \tag{4}$$

Heuristiikkana euklidinen etäisyys on hyvin turvallinen tilanteesta riippumatta, sillä kuten kuvasta 2 voidaan nähdä, tämän pituus on aina lyhin mahdollinen polku kahden pisteen välillä. Täten heuristiikan arvo on lähes aina pienempi kuin matkan todellinen arvo ja heuristiikan voidaan olettaa olevan hyväksyttävä ainakin reaaliverkoissa. Tällä heuristiikalla voidaan siis yliarvioida todellinen matka maaliin mistä tahansa pisteestä vain tilanteissa, missä joko ruudukko on painotettu tai heuristiikalle on annettu painokerroin.

3.3 Oktaali etäisyys

Oktaali etäisyys (*Octile distance*, myös *diagonal distance*) on suosittu tapa laskea kahden pisteen välistä heuristiikkaa [8]. Aiemmista heuristiikan mittaamisen menetelmistä poiketen oktaalia etäisyyttä käytetään harvemmin tapauksissa, joissa voidaan liikkua vain neljässä suunnassa. Sen sijaan sille on usein käyttöä, kun kardinaalisten suuntien lisäksi voidaan liikkua myös diagonaalisissa suunnissa kuvan 3 kaltaisesti.



Kuva 3. *Esitys heuristiikasta, joka on laskettu oktaalilla etäisyydellä.*

Oktaalia etäisyyttä laskettaessa vinottain liikkumiselle annetaan usein arvoksi neliön lävistäjän arvo √2 neliön sivun pituuden ollessa yksi. Tämä arvo saatetaan kuitenkin välillä korvata arvolla 1,5 [12]. Tätä voidaan perustella sillä, että se on ohjelmakoodin luettavuuden kannalta selkeälukuisempi ja samalla tarpeeksi tarkka arvio √2:sta. Lisäksi desimaaliluvun käsittely vaatii tietokoneelta vähemmän resursseja kuin neliöjuuriarvon laskeminen, mikä etenkin vanhoilla koneilla saattoi vaikuttaa jonkin verran ohjelman suoritukseen. Tämä diagonaalisessa suunnassa liikkuminen on havainnollistettu kuvassa 4. Tässä työssä muuttujan c arvona käytetään √2:ta.

$2\sqrt{2}$	$1+\sqrt{2}$	2	$1+\sqrt{2}$	$2\sqrt{2}$
$1+\sqrt{2}$	$\sqrt{2}$	1	$\sqrt{2}$	$1+\sqrt{2}$
2	1		1	2
$1+\sqrt{2}$	$\sqrt{2}$	1	$\sqrt{2}$	$1+\sqrt{2}$
$2\sqrt{2}$	$1+\sqrt{2}$	2	$1+\sqrt{2}$	$2\sqrt{2}$

3	2.5	2	2.5	3
2.5	1.5	1	1.5	2.5
2	1		1	2
2.5	1.5	1	1.5	2.5
3	2.5	2	2.5	3

Kuva 4. *Oktaalin etäisyyden kaarien painon arviointi. Vasemmalla vinottain liikkuminen on painoltaan $\sqrt{2}$ ja oikealla puolella paino on 1,5.*

Oktaalille etäisyydelle on ominaista, että se kulkee niin pitkään diagonaalisessa suunnassa kuin mahdollista ja loput matkasta kardinaalisessa suunnassa. Lisäksi se kulkee yhtä monen ruudun kautta, kuin pisteiden välisen kahden akselin suuremman välin arvo on. Tämä voidaan nähdä kuvasta 3, missä pisteiden P ja Q välinen etäisyys on x-akselin suunnassa viisi ruutua sekä y-akselin suuntaan kuusi ruutua. Lisäksi heuristiikka kulkee viisi ruutua vinosuunnassa sekä yhden ruudun suoraan. Heuristiikka siis liikkuu saman verran diagonaalisessa suunnassa kuin kahden akselin pienemmän välin arvo on suuri. Yksinkertaistaakseen tätä, oktaalia etäisyyttä laskettaessa heuristiikka voidaan jakaa kahteen erillään laskettavaan osaan: diagonaaliseen ja kardinaaliseen etäisyyteen.

Tämän avulla oktaali etäisyys kahden pisteen välillä voidaan ratkaista seuraavasti:

$$d = \min(x, y) * c + \max(x, y) - \min(x, y), \tag{5}$$

missä c on vinottain liikkumiselle annetun painon arvo sekä x ja y ovat vastaavasti kahden pisteen väliset x-akselin ja y-akselin suuntaiset arvot. Yhtälössä $\min(x, y) * c$ vastaa heuristiikan diagonaalista osaetäisyyttä ja $\max(x, y) - \min(x, y)$ puolestaan sen kardinaalista osaetäisyyttä. Tätä yhtälöä voidaan kuitenkin vielä sieventää:

$$d = \max(x, y) + (c - 1) * \min(x, y). \tag{6}$$

Kaavoissa (5) ja (6) esiintyvät alifunktiot min() sekä max() antavat vastaavasti parametreista pienimmän ja suurimman arvon.

3.4 Chebyshev-etäisyys

Chebyshev-etäisyys [11] on oktaalin etäisyyden laskemisen lisäksi toinen tunnettu heuristiikan laskentatapa ruudukoille, missä voidaan liikkua kahdeksassa suunnassa. Vaikka etäisyyden laskeminen Chebyshev-etäisyydellä muistuttaa paljon oktaalia etäisyyttä, missä voidaan myös liikkua kahdeksassa eri suunnassa, Chebyshev-etäisyyttä laskettaessa kardinaalisissa ja diagonaalisissa suunnissa liikkuminen lasketaan samanarvoisina. Tämä tulee ilmi kuvasta 5.

2	2	2	2	2
2	1	1	1	2
2	1		1	2
2	1	1	1	2
2	2	2	2	2

Kuva 5. *Chebyshev-etäisyyden painon kehittyminen ruudukossa.*

Chebyshev-etäisyydelle heuristiikan laskeminen on huomattavasti yksinkertaisempi kuin oktaalille etäisyydelle:

$$d = \max(x, y). \tag{7}$$

Kuten oktaalia etäisyyttä laskettaessa huomattiin, Chebyshev-etäisyydelle on ominaista, että heuristiikka kulkee yhtä monen ruudun kautta kuin pisteiden välisen kahden akselin suuremman välin arvo on.

4. A*-ALGORITMI

Ytimeltään A*-algoritmi noudattaa Dijkstran algoritmia, mutta tämän lisäksi se nopeuttaa hakujaan käyttämällä ahnas paras ensin –algoritmin kaltaisesti heuristiikkaa. Sen voidaan ajatella siis yhdistävän sekä Dijkstran algoritmin että ahnas paras ensin –algoritmin hyviä ominaisuuksia [6]. Yksinkertaisuudessaan A*-algoritmi käyttää solmujen arvon mittaamiseen luvussa 2 kaavassa 1 esitettyä funktiota $f(n)$. Funktion arvon $g(n)$ voidaan ajatella vastaavan Dijkstran algoritmia ja arvo $h(n)$ puolestaan ahnas paras ensin –algoritmia. Tämän funktion on tarkoitus tarjota näiden kahden ääripään hakualgoritmeille kompromissin, jota voitaisiin soveltaa sellaisenaan hyvin monella eri osa-alueella. [6]

Kun A*-algoritmi alkaa käsitellä solmua, lisätään solmu suljettu-listaan. Tämä lista sisältää kaikki käsitellyt solmut koko verkossa, jottei solmuja tarvitsisi käsitellä useammin kuin kerran. Solmua laajentaessa algoritmi tarkastaa, onko käsiteltävä solmu haluttu maalisolmu. Mikäli solmu ei ole maalisolmu, käsitellään tarkasteltavan solmun naapureita eli solmuja, jotka ovat kaarella yhdistettyjä tarkasteltavaan solmuun. Kullekin naapurille lasketaan etäisyys alkusolmusta itseensä sekä arvioitu etäisyys naapurista maalisolmuun heuristiikan avulla.

Avoimia solmuja on usein monta, joten algoritmin on osattava valita, mitä solmua laajentaa missäkin järjestyksessä. A*-algoritmi käyttää avoimien solmujen järjestämiseen Dijkstran algoritmin kaltaisesti prioriteettijonoa; se järjestää avoimet solmut alkusolmun etäisyyden ja heuristiikan summan perusteella pienuusjärjestykseen. Tällöin käsitellään aina ensisijaisesti niitä solmuja, jotka näyttävät olevan lähimpänä maalin tavoittelua.

```
1  function A*(piste P, piste Q):
2        /* syöte: alkupiste P, loppupiste Q
3        ulostulo: polku alkupisteestä loppupisteeseen ja [] jos polkua
4        ei löydetä */
5        suljettu = []
6        auki = [P]
7        /* suoritetaan silmukkaa, kunnes jokin polku löydetään tai
8        kunnes auki-lista on tyhjä */
9        while auki != []:
10             /* valitaan avoimesta listasta se solmu, jonka f:n arvo on
11             pienin, ja poistetaan se auki-listasta */
12             m = auki[0]
13             auki.remove(m)
14             suljettu.push(m)
15             if m == Q:
16                   return jaljita(m)
17             /* tarkastellaan kaikkia solmun naapureita. naapureita
18             ovat kaikki ne solmut, jotka ovat kaarella yhteydessä
19             solmuun m */
20             for each n in naapurit:
21                   if n in suljettu:
22                         continue
23                   gn = g(m) + d(m, n)
24                   /* tarkistetaan, mikäli solmua ei olla vielä aiemmin
25                   käsitelty tai jos solmuun päässään vähemmin
26                   kustannuksin */
27                   if n not in auki or gn < g(n):
28                         /* lasketaan solmulle arvot g, h ja f sekä
29                         määritellään sen edeltäjä polun määrittämistä
30                         varten */
31                         g(n) = gn
32                         h(n) = Heuristic(n, Q)
33                         f(n) = g(n) + h(n)
34                         n.edeltaja = m
35                         /* jos solmua ei olla käsitelty, lisätään se
36                         nyt auki-listaan ja asetetaan solmu tilaan
37                         auki */
38                         if n not in auki:
39                               auki.insert(n)
40                         /* mikäli jo käsitellylle solmulle ollaan
41                         löydetty parempi polku, päivitetään sen arvo
42                         auki-listassa */
43                         else:
44                               auki.update(n)
45       return []
```

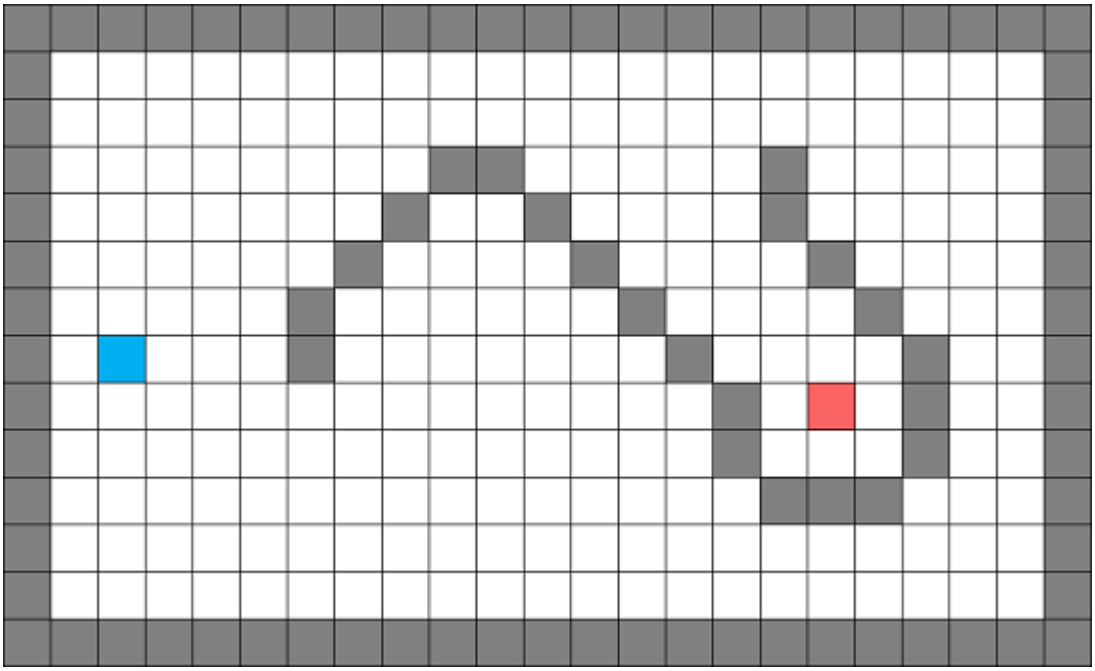
Ohjelma 1. *A*-algoritmin toiminta. Parametreina ovat alkupiste P ja loppupiste Q.*

A*-algoritmille on saatavilla vaihtoehtoinen toteutus, kun käsitellään naapurisolmuja, jotka löytyvät suljettu-listasta. Sen sijaan, että solmun käsittely ohitetaan kokonaan, kuten ohjelman 1 riveillä 21 ja 22, voidaan solmu vielä käsitellä. Tämä tarkoittaa sitä, että suljettu-listassa oleville solmuille ei välttämättä ole löydetty vielä optimaalisinta polkua alkusolmusta, minkä ansiosta saatetaan puolestaan löytää parempi polku maalisolmuun. Solmun uudelleenkäsittelyä ei kuitenkaan tarvitse huomioida tilanteissa, missä heuristiikka on jatkuvasti hyväksyttävä, koska tällöin kaikille suljettu-listassa

oleville solmuille on ominaista, että niille ollaan jo löydetty optimaalisin polku alkusolmusta [10].

5. ONGELMA: RUUDUKKO

Verkkoja käytetään usein edustamaan erilaisia alueita, kuten karttoja, kenttiä ja pelialueita. Niitä käytetään muun muassa lautapeleissä, kuten shakissa, sekä strategiapeleissä. Ruudukko on verkoista tavallisin, sillä se on toteutukseltaan yksinkertainen ja noudattaa karteesista koordinaatistoa, joka on ihmisille yleisesti hyvin tuttu. [13]

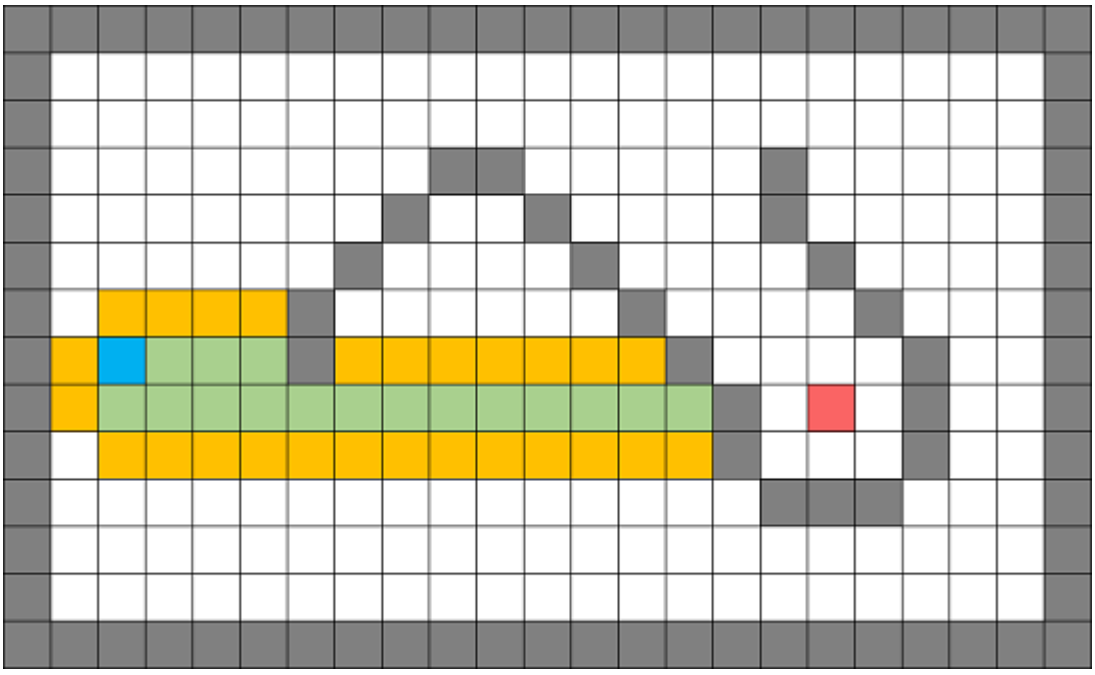


Kuva 6. *Ruudukkokartta, jossa heuristiikkoja testataan. Sininen ruutu kuvaa alkusolmua ja punainen ruutu maalisolmua.*

Tässä työssä A*-algoritmia käytetään arvioimaan kahden solmun välistä etäisyyttä kuvan 6 kaksiulotteisessa ruudukossa, jossa voidaan liikkua kardinaalisissa suunnissa (*four-way movement*). Poissulkien aloitus- ja maaliruudun, ruudukossa on joko niin sanottuja lattioita eli ruutuja, joissa voidaan kulkea, tai sitten esteitä, joiden läpi ei voida kulkea. Ruudukko on siis painottamaton, eli kaikki ruudut, joissa voidaan liikkua, omaavat saman painon. Vaihtoehtoisesti ruudukko voisi sisältää hidasteita, kuten vaikkapa metsiä tai vuoria joiden läpi voidaan kulkea, mutta joiden paino olisi suurempi kuin tavallisilla ruuduilla.

6. TULOKSET JA YHTEENVETO

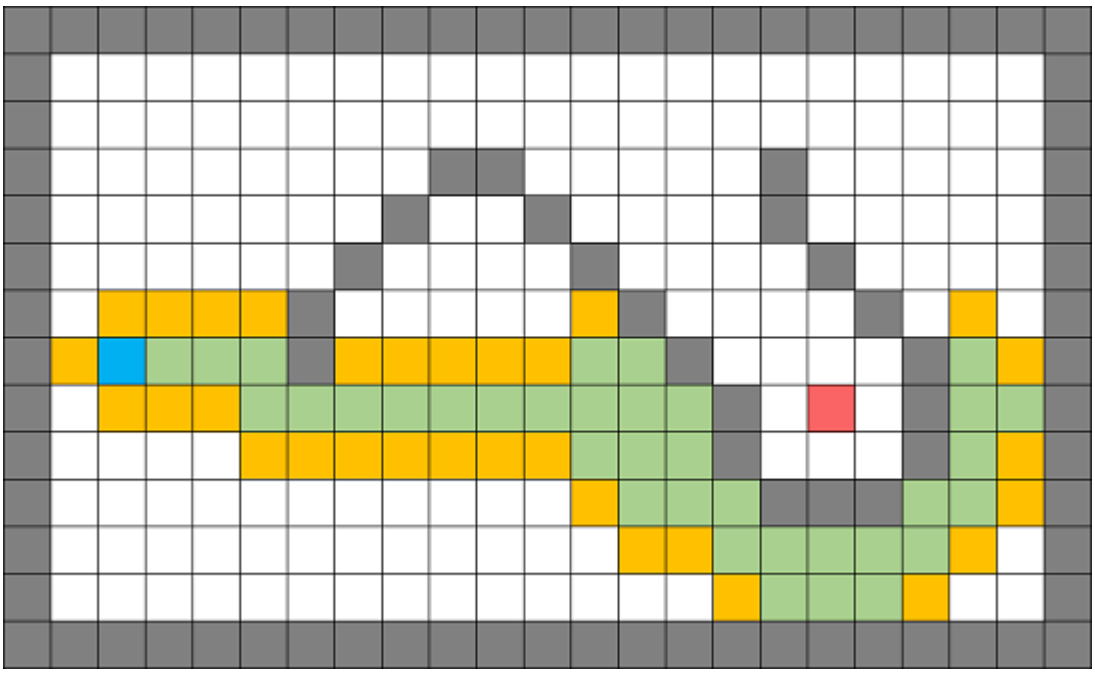
Työssä esitettävät tulokset ja havainnot on saatu JavaScript-ohjelmalla käyttäen PathFinding.js –reitinhakukirjastoa [14]. Koska työssä käytetyn kirjaston A*-algoritmin implementaatio on deterministinen, ei testejä tarvitse suorittaa jokaiselle tapaukselle kuin vain kerran. Algoritmin deterministisyydellä viitataan siihen, että algoritmi saa samalla syötteellä saman tuloksen jokaisella suorituskerralla [1]. Heuristiikan painottamisessa käytetään kokonaislukuja arvoväliltä 1-10. Painottamisessa ei ole syytä käyttää pienempää painotusta, sillä A*-algoritmi löytää aina optimaalisen ratkaisun painotuksen ollessa 1 [1, 10]. Tämän lisäksi suuremmilla painotuksilla ei ole erityisen poikkeavia vaikutuksia A*-algoritmin käyttäytymiseen, joten työssä käytetty suurin painotus on rajattu arvoon 10. Tässä työssä operaatioiksi lasketaan jokaisen solmun avaaminen ja sulkeminen. Lopuksi työssä käytetään luvun 3 käsittelemiä heuristiikkoja erilaisilla painotuksilla luvussa 5 esitetyn kuvan 6 karttaa käyttäen.



Kuva 7. *A*-algoritmin suorittamat haut reitinhaun alussa käyttäen heuristiikkana Manhattan-etäisyyttä painotuksella 1. Vihreät ruudut edustavat käsiteltyjä, suljettuja solmuja ja oranssit ruudut ovat avoimia, käsiteltävissä olevia solmuja.*

A*-algoritmi käyttäytyy alussa heuristiikasta ja sen painotuksesta riippumatta hyvin samankaltaisesti verkossa. Ensimmäisten operaatioiden aikana se käy laajentamaan alkusolmusta oikealle ja yrittää edetä aluksi muurin alapuolelta kohti maalia. Algoritmi hakeutuu ensimmäisen esteen kohdalla ensisijaisesti alas, sillä maalisolmu on alkusolmua matalammalla y-koordinaatilla ja täten heuristiikka priorisoi haun edistämistä

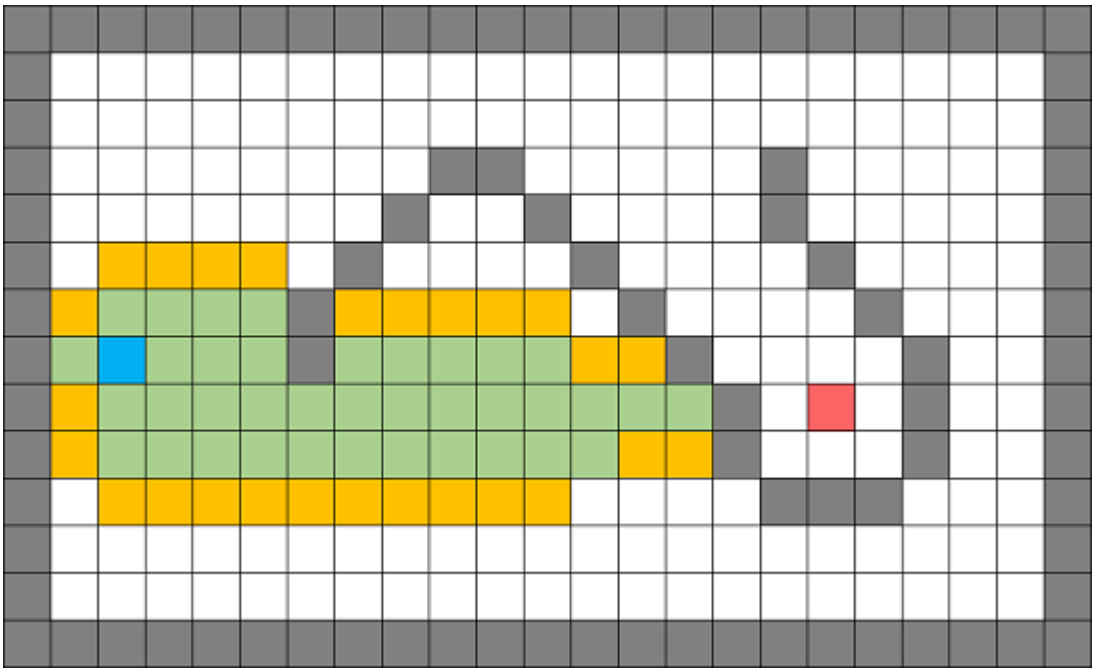
siihen suuntaan kuvan 7 kaltaisesti. Vasta kun algoritmi saavuttaa maalin edessä olevan muurin, alkaa haku laajeta sivuille päin etsien polkua muurin ohi.



Kuva 8. *A*-algoritmin suorittamat haut noin 100:n operaation jälkeen käyttäen Manhattan-etäisyyttä painotuksella 10.*

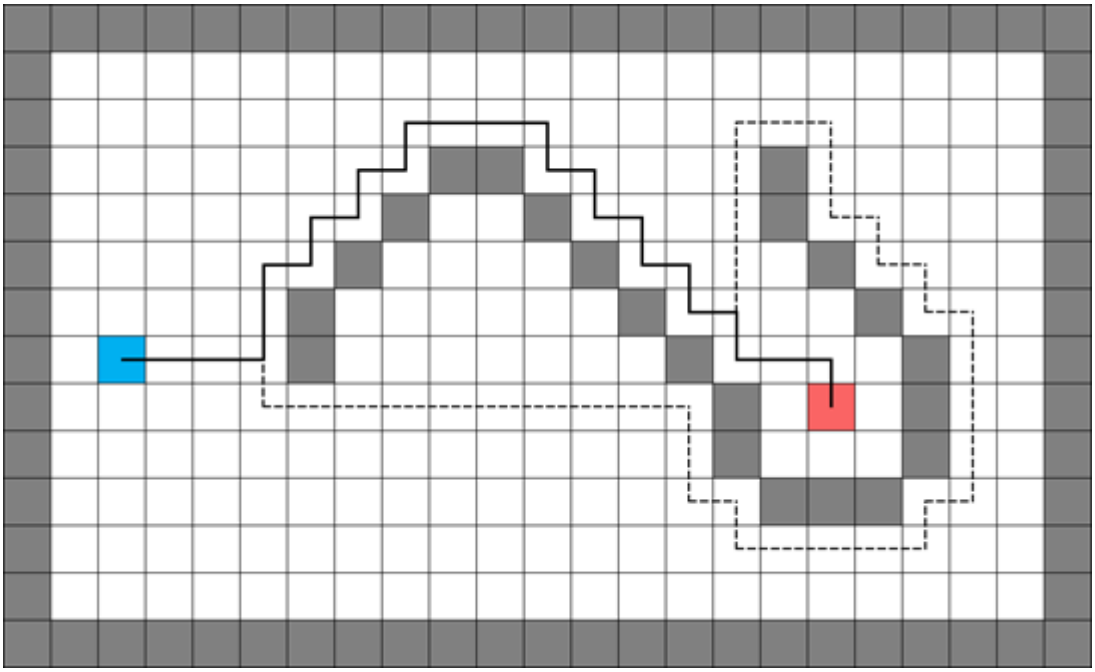
Heuristiikan vaikutus reitinhakualgoritmin käyttäytymiseen tulee ilmi, kun algoritmi kohtaa esteen, jota se ei pysty ohittamaan välittömästi. Mitä vahvempi painotus heuristiikalla on, sitä aggressiivisemmin hakualgoritmi pyrkii etsimään polkua niiden solmujen kautta, jotka ovat lähimpänä tavoiteltua maalisolmua. Kuvan 8 mukaisesti tarpeeksi suurella heuristiikalla tämä käytös johtaa siihen, että hakualgoritmi ei koe muurin yläkautta kulkevaa reittiä merkitykselliseksi, vaan se ennemminkin pyrkii etsimään polkua maaliin muurin alapuolelta. Haun edetessä algoritmi saattaa lopulta ongelmallisesti löytää maalisolmuun polun, joka hyvin epäoptimaalisesti kiertää alateitse koko muurin.

Kuvasta 9 voidaan nähdä, että pienemmällä painotuksella algoritmi laajentaa hakualuetta huomattavasti kauempaa esteestä kuin edellä esitetty voimakkaasti painotettu heuristiikka. Pienellä heuristiikalla reitinhakualgoritmi huomioi kaukaisempia vaihtoehtoja reilummin, jonka takia esteelle saatetaan löytää tehokas kiertotie varhaisessa vaiheessa. Näillä heuristiikoilla haun pinta-ala laajenee kuitenkin hyvin nopeasti esteeseen törmätessä, joka hidastaa haun etenemistä, mikäli kiertotietä ei löydetä välittömästi.



Kuva 9. *A*-algoritmin suorittamat haut noin 100:n operaation jälkeen käyttäen euklidista etäisyyttä painotuksella 1.*

Suuresta heuristiikasta poiketen pienempi heuristiikka huomioi muurin yläkautta kiertämisen varhaisessa vaiheessa. Vaikka pienellä heuristiikalla saatetaan käyttää ensisijaisesti enemmän operaatioita esteen ohittamiseen, lopputuloksena saadaan kuitenkin lyhyempi polku maalisolmuun.



Kuva 10. *Esitys mahdollisista reiteistä maaliin. Katkoviivoilla esitetty polku on poluista pidempi, joka voidaan saada heuristiikkaa vahvasti painottamalla.*

Kuvassa 10 esitetään, että maalisolmuun on kaksi eri reittiä. Yksi tapa on kulkea alkusolmusta suoraan muurin yläpuolelta maalisolmuun, tarjoten nopeimman polun maaliin. Tämä lyhyt polku seuraa kuvassa yhtenäistä viivaa ja sen pituus on 26 ruutua. Vaihtoehtoinen, pidempi polku maaliin käy kuvassa olevia katkoviivoja pitkin seinämän ympäri huomattavasti pidempää reittiä pitkin maaliin. Tämän pitkän polun pituus on puolestaan 44 ruutua. Reitinhakualgoritmin palauttamat polut voivat kulkea hieman eri reittiä pitkin kuvan 10 polkuihin nähden. Algoritmi voisi esimerkiksi antaa lyhyelle polulle reitin, joka ei kulje kuvan kaltaisesti muuria pitkin maalisolmua lähestyessä. Polut ovat tästä huolimatta kuitenkin yhtä pitkät, ja tämä polun muodostuminen riippuu pitkälti algoritmin toteutuksesta. Taulukkoon 1 on kerätty kunkin heuristiikan saaman polun pituus eri painotuksin arvoväliltä 1–10.

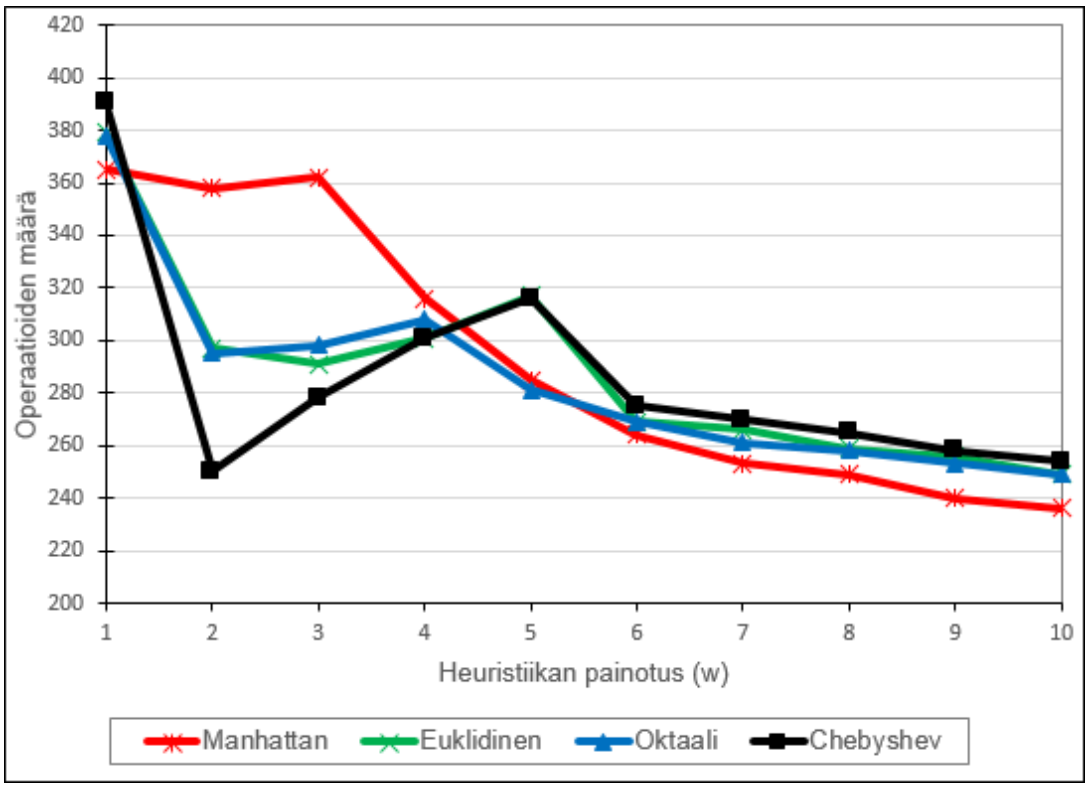
Taulukko 1 *Saadun polun muutos painotuksen kasvaessa.*

Painotus	Manhattan	Euklidinen	Oktaali	Chebyshev
1	lyhyt	lyhyt	lyhyt	lyhyt
2	lyhyt	lyhyt	lyhyt	lyhyt
3	lyhyt	lyhyt	lyhyt	lyhyt
4	pitkä	lyhyt	lyhyt	lyhyt
5	pitkä	lyhyt	pitkä	lyhyt
6	pitkä	pitkä	pitkä	pitkä
7	pitkä	pitkä	pitkä	pitkä
8	pitkä	pitkä	pitkä	pitkä
9	pitkä	pitkä	pitkä	pitkä
10	pitkä	pitkä	pitkä	pitkä

Mielenkiintoisena havaintona testitapauksen kaltaisille nelisuuntaisille ruudukoille suunnattu Manhattan-etäisyys pystyi sietämään painottamista vähiten. Tämä luultavasti johtuu kuitenkin pitkälti siitä, että Manhattan-etäisyys antaa verratuista heuristiikoista kaikkein pessimistisimmän eli pisimmän arvion. Painottamisen myötä Manhattan-etäisyydellä laskettu heuristiikka vääristää todellisen etäisyyden maaliin suuremmin kuin muut heuristiikat. Vastaavasti Euklidinen ja Chebyshev-etäisyydet pystyivät sietämään painottamista eniten. Nämä heuristiikat ovat heuristiikoiltaan optimistisia, jolloin painottaminen ei alussa vaikuttanut niin vahvasti kuin vaikkapa Manhattan-etäisyydellä.

A*-algoritmin suorituskykyä ei yksin kuvaa sen saaman reitin pituus. Tässä työssä yksi toinen tärkeä suorituskyvyn mittari reitinhakualgoritmin tehokkuudelle on algoritmin suorittamien operaatioiden määrä tuloksen aikaansaamiseksi. Tämä ei siis huomioi saadun reitin pituutta ollenkaan, vaan se huomioi vain haun aikana suoritetut operaatiot.

Kaikkien eri heuristiikkojen ja painotusten yhdistelmien operaatioiden määrät ovat esitetty kuvassa 11.

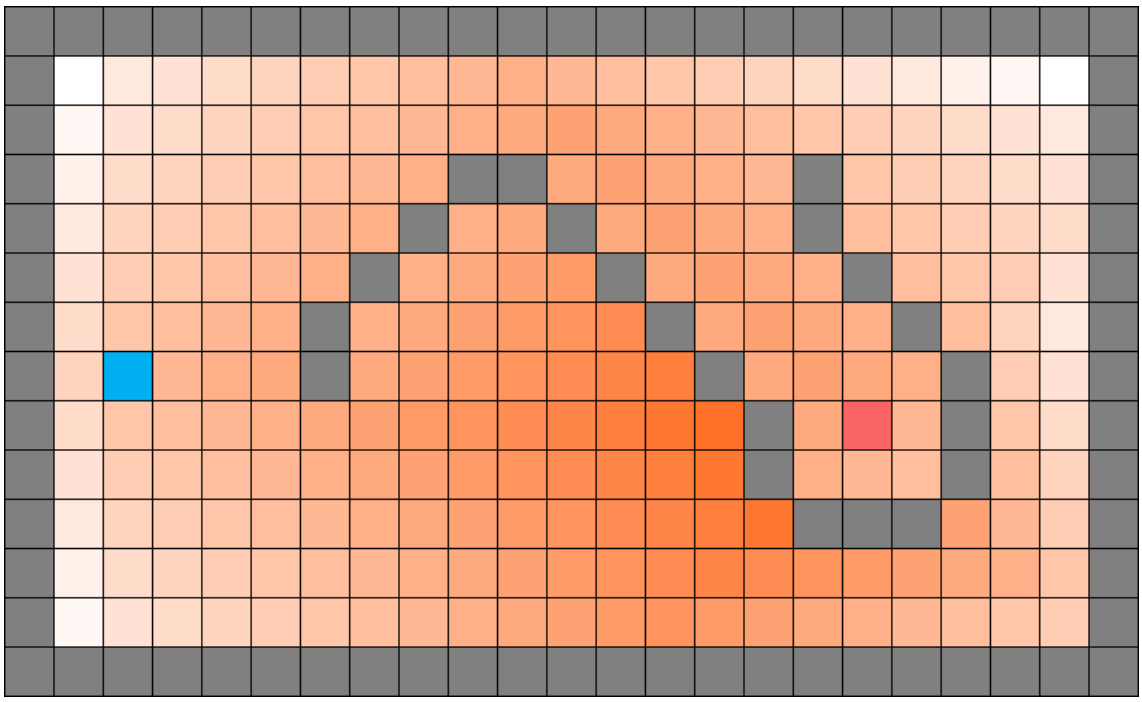


Kuva 11. *Kuvaaja A*-algoritmin suorittamien operaatioiden määrästä eri heuristiikoilla ja painotuksilla.*

Taulukkoa 1 sekä kuvaa 11 tarkkailemalla voidaan havaita, että kun heuristiikan painottamisen myötä saadun polun pituus kasvaa lyhyestä polusta pitkään polkuun, algoritmin suorittamien operaatioiden määrä laskee merkittävästi. Näistä ilmiöistä näkyvin on Manhattan-etäisyys. Sen suorittamien operaatioiden määrä väheni painotusten 3 ja 4 välillä yli 40:llä ylittäessään lyhyen ja pitkän polun välisen kynnyksen. Tämän kynnyksen ylityksen jälkeen operaatioiden määrä vähenee maltillisesti painotuksen kasvaessa. Tämä johtuu pitkälti siitä, että heuristiikka tekee reitinhakualgoritmin käyttäytymisestä jatkuvasti ahneempaa.

Yksittäisistä painotuksen arvoista hyvin mielenkiintoinen on painoarvo 2. Algoritmin suorittamien operaatioiden määrä tällä painotuksella on hyvin pieni kaikilla heuristiikoilla suhteessa muihin painotuksiin. Etenkin Chebyshev-etäisyydellä saadaan operaatioiden määrää vähennettyä merkittävän pieneksi. Kuvassa 12 kuvataan A*-algoritmin funktion $f(n)$ arvoja verkossa edellä mainitun Chebyshev-etäisyyden kanssa painotuksella 2. Tästä voidaan huomata, heuristiikka soveltuu erinomaisen hyvin työn verkkoon. Ensisijaisten hakujen jälkeen algoritmi käsittelee mahdollisuuden liikkua muurin

yläpuolelta hyvin varhaisessa vaiheessa, mikä johtaa vähäisiin vaadittuihin operaatioihin.



Kuva 12. *A*-algoritmin funktion f(n) arvo, kun heuristiikkana käytetään Chebyshev-etäisyyttä painotuksella 2. Tummempi väri kuvaa pienempää funktion arvoa.*

Yksi operaatioiden määrään vaikuttava tekijä on ruudukon ulkoreunoilla oleva ympyröivä seinämä. Tämä ei välittömästi vaikuta erityisen merkittävältä, sillä algoritmin saama polku ei tule kuitenkaan kulkemaan niin kaukaa seinämiä pitkin koskaan. Nämä seinämät kuitenkin vaikuttavat haun aikana suoritettujen operaatioiden määrään huomattavasti. Erityisesti kun suurilla heuristiikoilla hakiessa törmätään maalin ympärillä olevaan muuriin, saattaa haku lähteä helposti laajenemaan pois maalisolmusta. Seinämän on siis ennen kaikkea tarkoitus rajoittaa reitinhakualgoritmin haun laajenemista liian kauas pois kentän keskeltä.

Vaikka tulokset voivat antaa ymmärtää yhden heuristiikan olevan parempi kuin toisen, on toisaalta huomioitava, että tulokset eivät ole yksiselitteisiä. Heuristiikan valintaan vaikuttavat monet tekijät, mutta ennen kaikkea valintaan vaikuttaa se, mitä verkosta halutaan löytää. Joku voisi haluta optimaalisen polun kahden pisteen välille yli kaiken, toinen haluaa taas ihan minkä tahansa reitin, kunhan se saadaan mahdollisimman nopeasti. Mikäli tavoitteena on saada optimaalinen polku kahden pisteen välille, ei heuristiikkaa tulisi painottaa yhtään. Heuristiikan valintaan voi vaikuttaa myös verkon rakenne. Jos vaikkapa tiedetään, että ruudukossa voidaan liikkua myös diagonaalisissa suunnissa, oktaali ja Chebyshev-etäisyydet olisivat erittäin johdonmukaisia valintoja heuristiikaksi muihin heuristiikkoihin nähden diagonaalisten suuntien toteutuksien

ansiosta. Lisäksi jos käsiteltävästä verkosta tiedetään joitain ominaisuuksia etukäteen, kuten vaikkapa sen esteiden tiheyden, voidaan heuristiikkaa muokata tehostamaan algoritmin toimintaa entisestään. Jatkotutkimusta ajatellen testattavana verkkona voitaisiin käyttää isompaa karttaa, sillä heuristiikan painottamisen positiiviset piirteet eivät korostuneet näin pienessä verkossa kovin vahvasti.

LÄHTEET

[1] S. Edelkamp, S. Schrödl, Heuristic Search: Theory and Applications, Elsevier Science & Technology, 2012, 865 p.

[2] G. Pólya, How To Solve It, Princeton, Princeton University Press, 1957. 253p.

[3] O. Torikka, A*-algoritmi ja siihen pohjautuvat muistirajoitetut heuristiset reitinhakualgoritmit, Itä-Suomen Yliopisto, helmikuu 2015. 61p.

[4] R. E. Tarjan, Data structures and network algorithms, Society for Industrial and Applied Mathematics, 1983. 131p.

[5] E.W. Dijkstra, A Note on Two Problems in Connexion Graphs, 1959: A Note on Two Problems in Connexion Graphs. Saatavissa: http://www-m3.ma.tum.de/foswiki/pub/MN0506/WebHome/dijkstra.pdf

[6] A. Patel, Introduction to A*: A* Comparison. Saatavissa: http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html#the-a-star-algorithm, Haettu: 18.04.2019

[7] R. Dechter, J. Pearl, Generalized Best-First Search Strategies and the Optimality of A*, Jul 1985. 505–536p.

[8] A. Patel, Heuristics, From Amit's Thoughts on Pathfinding. Saatavissa: http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html, Haettu: 17.01.2019

[9] P. E. Hart, N. J. Nilsson, B. Raphael, A formal basis for the heuristic determination of minimum cost paths, IEEE Transactions on Systems Science and Cybernetics, Volume: 4, Issue: 2, Jul 1968. 100–107p.

[10] J. T. Thayer, W. Ruml, Faster Than Weighted A*: An Optimistic Approach to Bounded Suboptimal Search, AAAI Press, Menlo Park, 2008

[11] G. Upton, I. Cook, Dictionary of Statistics, Oxford University Press, 2014 Saatavissa: http://www.oxfordreference.com/view/10.1093/acref/9780199679188.001.0001/acref-9780199679188

[12] N. Sturtevant, A* Tie Breaking. Saatavissa: https://movingai.com/astar.html, Haettu: 31.03.2019

[13] A. Patel, Grid math: Square, Hexagon, Triangle, 2006. Saatavissa: http://www-cs-students.stanford.edu/~amitp/game-programming/grids/, Haettu: 16.04.2019

[14] X. Xu, PathFinding.js. Saatavissa: https://github.com/qiao/PathFinding.js/, Haettu: 12.03.2019