

# **Rinnakkainen reitinhaku tietokonepeleissä**

Santeri Martikainen

Helsinki 22.10.2016

HELSINGIN YLIOPISTO  
Tietojenkäsittelytieteen laitos

Tiedekunta – Fakultet – Faculty	Laitos – Institution – Department
Matemaattis-luonnontieteellinen tiedekunta	Tietojenkäsittelytieteen laitos
Tekijä – Författare – Author	
Santeri Martikainen	
Työn nimi – Arbetets titel – Title	
Rinnakkainen reitinhaku tietokonepeleissä	
Oppiaine – Läroämne – Subject	
Tietojenkäsittelytiede	
Työn laji – Arbetets art – Level	Aika – Datum – Month and year
	22.10.2016
Sivumäärä – Sidoantal – Number of pages	
10 sivua + 1 liitesivu	
Tiivistelmä – Referat – Abstract	
Avainsanat – Nyckelord – Keywords	
aine	
Säilytyspaikka – Förvaringställe – Where deposited	

# Sisältö

1 Johdanto	1
2 Reitinhaku	1
3 Rinnakkainen reitinhaku	2
3.1 Local Repair A*	
3.2 Cooperative A*	
3.3 Hierarchical Cooperative A*	
3.4 Windowed Hierarchical Cooperative A*	
3.5 Conflict-based Search	
Lähteet	5

# 1 Johdanto

Tässä aineessa keskitytään tarkastelemaan reitinhakua tietokonepelien saralla ja siinä erityisesti algoritmeja jotka on suunniteltu tilanteeseen, missä monta eri toimijaa (agent) jakaa saman tilan ja joille kaikille on löydettävä reitti kohteeseensa joillakin reunaehdoilla.

Reitinhaussa on pohjimmiltaan kyse mahdollisimman suoran ja/tai nopean polun löytämisestä kahden pisteen välillä jossakin topologiassa. Pelimaailman topologia, ts. kenttä, alue, jolla pelaaja ja mahdolliset ei-pelaajahahmot voivat liikkua, voi olla ulkoasultaan hyvin abstrakti ja pelkistetty, tai toisaalta vaikuttaa hyvinkin realistiselta maastolta puineen, vesistöineen, rakennuksineen ja niin edelleen. Rakenteellisesti kenttä kuitenkin muodostuu tyypillisesti verkosta  $G = (V, E)$ , missä  $V$  (vertex) edustaa solmuja ja  $E$  (edge) solmuja yhdistäviä kaaria. Kenttä voidaan muodostaa myös soluista, käytännössä polygoneista, jotka ovat joko kolmioita, neliöitä tai kuusikulmioita. Edellämainitut muodot ovat ainoat polygonit, joita yhdistelemällä voidaan jokin alue kattaa ilman väliin jääviä aukkoja. Liikkuminen voi tapahtua joko solmusta/solusta toiseen, tai sitten nämä voivat edustaa kiintopisteitä joiden välittömässä läheisyydessä voidaan liikkua ilman erillistä reitinhakua. Osa näistä voi toimia esteenä joko pysyvästi, tai väliaikaisesti esimerkiksi jonkun toisen toimijan ollessa siinä. Molemmissa tapauksissa täytyy luonnollisesti löytää reitti esteen ympäri. Topologia voi hyvin olla dynaaminen, eli sen rakenne saattaa muuttua, jopa koko ajan, mikä asettaa omat haasteensa reitinhauille[1].

# 2 Reitinhaku

Realismia tavoittelevassa pelimaailmassa esteitä ovat luonnollisesti kaikki sellaiset objektit, jotka olisivat reaalimaailmassakin esteitä, kuten puut tai seinät. Siten pelissä liikkuva hahmo ei usein voi kulkea kahden pisteen välillä suoraan, vaan esteiden ympäri on löydettävä jokin reitti, jonka löytämiseen sovelletaan reitinsintäalgoritmeja.

Reitinsintä kokonaisuutena jakaantuu kahteen vaiheeseen: Toimintaympäristöstä muodostetaan ensin yksinkertaistettu malli, minkä jälkeen sitä käydään läpi jollakin algoritmilla halutun reitin löytämiseksi. Tunnetuin näistä algoritmeista on nimeltään  $A^*$  (eli A-star tai A-tähti), josta on kehitetty lukuisia eri variantteja eri toimintaympäristöjä silmälläpitäen[1].

$A^*$  toteuttaa niin sanottua paras ensin-tyyliä, jossa jokaisen solmun tai solun kohdalla pyritään ensiksi etenemään suoraan kohti maalia. Jos tiellä on jokin este, algoritmi pyrkii kiertämään sen pyrkimällä viereisiin soluihin ja sieltä jälleen maalia kohti kunnes joko kohteeseen päästään, tai reittiä ei voida löytää.

Videopeleissä on yleensä useita, jopa satoja tai tuhansia, eri toimijoita, joille on löydettävä reitit kohteisiinsa. Tällainen monen toimijan rinnakkainen reitinhaku (multi-agent pathfinding, MAPF) tuo yksittäisen toimijan reitinhakuun verrattuna uusia ongelmia, kuten sallitaanko reittien risteäminen, voiko kaksi tai useampi toimija olla samaan aikaan samassa paikassa ja tuleeko liikkumista porrastaa odottamalla että reitti edessä vapautuu[2]. Näiden ongelmien ratkaisemiseen on kehitetty useita eri algoritmeja.

### 3 Rinnakkainen reitinhaku

#### 3.1 Local Repair A\*

LRA\* on itse asiassa yleistermi joukolle A\*-pohjaisia algoritmeja, jotka jakavat saman toimintaperiaatteen: Jokainen toimija etsii reitin A\*:lla ja seuraa sitä siihen asti kunnes siirtyminen seuraavaan solmuun saisi aikaan yhteentörmäyksen jonkun toisen toimijan kanssa, eli solmu johon pitäisi siirtyä on varattu. Tällöin tehdään uusi A\*-haku ja jatketaan niin kauan kunnes on tultu maaliin.

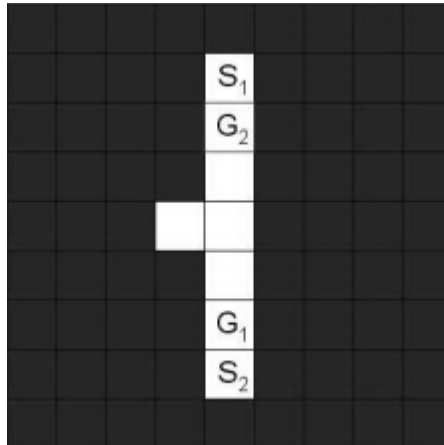
Syklit ovat tässä menetelmässä sekä mahdollisia että yleisiä, joten ongelmaa on pyritty ratkaisemaan lisäämällä nk. kohinaa etäisyysheuristiikkaan joka kerta kun yhteentörmäys havaitaan [4]. Nyt siis jos jatkuvasti kohdataan esteitä jossakin solmussa tai jollakin alueella, niin algoritmin laskeman kustannuksen sinne etenemisestä pitäisi ennenpitkää nousta niin suureksi, että toimija hakeutuu ongelma-alueen ympäri tai etsii kokonaan uuden reitin.

Tällainen lähestymistapa johtaa ruuhkatilanteissa helposti epä-älylliseltä näyttävään poukkoiluun, ja sen myötä prosessoriajan hävikkiin kun jokaisen yhteentörmäyksen yhteydessä reitti pitää laskea uudestaan.

#### 3.2 Cooperative A\*

CA\* pyrkii estämään yhteentörmäykset ennalta ottamalla aikaulottuvuuden huomioon reittejä suunniteltaessa. Jokaiselle toimijalle lasketaan reitti A\*-algoritmillä ja suunnitellun reitin solut merkitään taulukkoon, esimerkiksi kolmiulotteiseen hajautustauluun, missä kaksi ensimmäistä alkioita merkitsevät x- ja y-koordinaatteja, ja kolmas alkio aikaa milloin kyseinen solmu on tämän toimijan käytössä. Nyt seuraavien toimijoiden reittejä laskettaessa voidaan verrata suunniteltuja askeleita edellämainittuun taulukkoon ja odottamalla havaituissa törmäystilanteissa halutun reittisolmun vapautumista.

Tämän algoritmin heikkoutena on kyvyttömyys ratkaista joitakin verrattaen yksinkertaisia tilanteita. Kuvassa 1 nähdään tilanne, missä toimijat S1 ja S2 pyrkivät vastaavasti maaleihin G1 ja G2. Vaikka intuitiivisesti nähdään, että jommankumman toimijan olisi mahdollista joko väistää sivulle tai vaihtaa paikkoja törmätessään ja siten ratkaista kohtaamisongelma, ei Cooperative A\* pysty tällaiseen ratkaisuun, vaan toimijat eivät koskaan pääse maaliin.



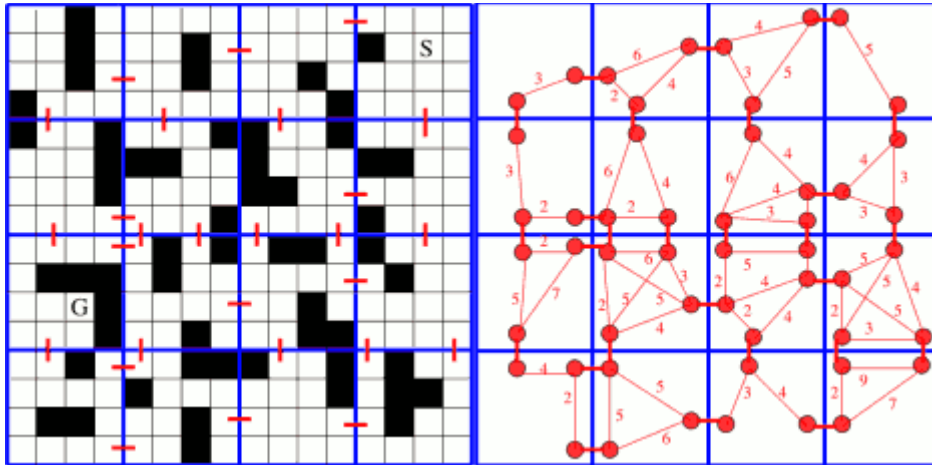
*Kuva 1: Esimerkki kohtaamis-ongelmasta, jossa toimijat  $S_1$  ja  $S_2$  voivat päästä maalisolmuihin  $G_1$  ja  $G_2$  vain jos jompikumpi väistää sivulle[4].*

### 3.3 Hierarchical Cooperative A\*

R.C.Holte et al. esitteli vuonna 1996 hierarkkisen A\*-reitinetsintäalgoritmin HA\*[8] ratkaisemaan CA\*-algoritmin ongelmia. Siinä varsinaisen topologian rinnalla käytetään toista, abstraktia, topologiaa auttamaan A\*:a löytämään kohteeseen. Alkuperäisen topologian solmut ryhmitetään halutun abstraktioetäisyyden perusteella isommiksi abstraktiosolmuiksi, ja tätä jatketaan rekursiivisesti niin kauan, kunnes koko topologiasta on muodostettu yksittäinen abstraktiosolmu. Varsinaisen topologian rinnalla on nyt siis monitasoinen abstraktiohierarkia, jonka alimmalla tasolla on alkuperäinen topologia. Tätä monitasoista hierarkiaa käytetään heuristiikan apuna ja liikkumalla siinä tasolta toiselle sen mukaan miten tarkkaa jakoa esimerkiksi esteiden ympärillä tarvitaan, ja käyttämällä sen antamia etäisyyksiä reitinhakualgoritmin apuna. Etäisyydet kohteeseen lasketaan vain tarvittaessa dynaamisesti muuttuvasta ympäristöstä johtuen[4].

HCA\* eli hierarkkinen yhteistoiminnallinen A\*-reitinetsintäalgoritmi puolestaan käyttää yksinkertaistettua hierarkiaa, joka koostuu vain yhdestä topologian kaksikulotteisesta abstraktiosta (kuva 2) jättäen huomiotta niin aikaulottuvuuden, solmujen varaustaulun, kuin muut toimijat. Kun reitti on laskettava uudelleen esimerkiksi jonkun toisen toimijan tullessa eteen, pyritään säästämään resursseja käyttämällä käänteistä jatkettavaa A\*-hakua RRA\* (Reverse Resumable A\*) abstraktiotasolla. Siinä missä alkuperäinen reitti haettiin lähtöpisteestä maaliin, RRA\* etsii reitin maalista haluttuun solmuun, kuten toimijaa lähimpään abstraktiotason solmuun[4]. Mikäli tässä vaiheessa optimaalisen reitin varrella on muita toimijoita, tulee lasketusta reitistä näiden väistämisen vuoksi luonnollisesti pidempi, aivan kuten alkuperäisen reitinhaunkin suhteen.

Ongelmana tämän algoritmin käytössä on reitinhaun lopettamisen määrittelemine, sillä vaikka jokin toimija olisi jo tullut päämääräänsä, sen täytyy mahdollisesti vielä väistää jotain toista toimijaa ja hakeutua tämän jälkeen uudestaan maaliin[9]. Niinikään toimijoiden vuorjärjestyksellä on väliä: Staattinen vuorottelu voi johtaa siihen, ettei reittiä maaliin koskaan löydetä jonkun toimijoiden sulkiessa toisiltaan tien. Tämä voidaan välttää antamalla toimijoille erilaiset prioriteetit jo alun alkaen, tai sitten sitten korkeampi prioriteetti voidaan antaa halutuille toimijoille vuorotellen lyhyeksi aikaa[4]. Tässä, kuten aiemmissakin yhteistoiminnallisissa reitinhakualgoritmeissa, on resurssien käytön suhteen ongelmana potentiaalisesti turhaan tehty työ.



Kuva 2: Vasemmalla olevasta ruudukosta on muodostettu hierarkiamallin mukainen abstraktio. Punaiset viivat vasemmalla vastaavat abstraktion solmupareja, ja kirjaimet S ja G viittaavat lähtö- ja maalitiloihin. (<http://aigamedev.com/open/review/near-optimal-hierarchical-pathfinding/>)

### 3.4 Windowed Hierarchical Cooperative A\*

WHCA\* eroaa HCA\*:sta siinä, että konkreettisen topologian tasolla reittejä ei lasketa maaliin asti, vaan reitinhaku on rajoitettu johonkin ennaltamääritellyyn syvyyteen. Jotta toimijat saadaan hakeutumaan varmasti oikeaan suuntaan, lasketaan reitti abstraktiotasolla sen sijaan maaliin asti[4][9]. Kun reittiä on kuljettu johonkin ennaltamääritellyyn raja-arvoon asti, kuten puolet aiemmin lasketusta reitistä, lasketaan uusi osittainen reitti ja niin edelleen. ”Windowed” tarkoittaa tässä siis aikaikkunaa tai kehystä, mihin asti konkreettinen reitti on nähtävillä ja mitä siirretään aina tarpeen mukaan. Resurssien käyttöä voidaan myös tasata antamalla toimijoille erikokoiset ikkunat, niin että taakka reittien laskemisesta jakautuu mahdollisimman tasaisesti ajan suhteen. Kuten HCA\*, myös WHCA\* hyödyntää RRA\*:ta ja hyödyntää edellisen ikkunan aikana tehtyä hakua, mikä luonnollisesti tarkoittaa toimijakohtaista kirjanpitoa läpikäydyistä solmuista.

### 3.5 Conflict-based Search

## Lähteet

- [1] Algfoor, Z.A.; Sunar, M.S.; Kolivand, H.  
A Comprehensive Study on Pathfinding Techniques for Robotics and Video Games  
International Journal of Computer Games Technology Volume 2015, Article ID 736138
- [2] Erdem, E.; Kisa, D.G.; Oztok, U.; Schüller, P.  
A General Formal Framework for Pathfinding Problems with Multiple Agents  
AAAI, 2013
- [3] Standley, T. S.  
Finding optimal solutions to cooperative pathfinding problems.  
In Proc. of AAAI, 2010
- [4] Silver, D.  
Cooperative Pathfinding  
In Proc. of AIIDE, 117-122, 2005
- [5] Cui, X.; Shi, H.  
A\*-based Pathfinding in Modern Computer Games  
IJCSNS International Journal of Computer Science and Network Security, VOL.11 No.1, 2011
- [7] Stout, Bryan  
Smart Moves: Intelligent Pathfinding  
Game Developer Magazine, July 1997
- [8] Holte, R.C.; Perez, M.B.; Zimmer, R.M.; MacDonald, A.J.  
Hierarchical A\*: Searching Abstraction Hierarchies Efficiently  
AAAI/IAAI, Vol. 1, 1996
- [9] Botea, A.; Bouzy, B.; Buro, M.; Bauckhage, C.; Nau, D.  
Pathfinding in Games  
Artificial and Computational Intelligence in Games, ss. 21–31, 2013
- [??] Sharon, G.; Stern, R.; Felner, A.; Sturtevant, N.R.  
Conflict-based search for optimal multi-agent pathfinding  
Artificial Intelligence, 2015