Emil Keränen

Reitinhakualgoritmien käyttö videopeleissä

Tietotekniikan kandidaatintutkielma

5. kesäkuuta 2018

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Emil Keränen

Yhteystiedot: keranen.emil@gmail.com

Ohjaaja: Sanna Mönkölä

Työn nimi: Reitinhakualgoritmien käyttö videopeleissä

Title in English: Pathfinding algorithms in video games

Työ: Kandidaatintutkielma

Sivumäärä: 20+0

Tiivistelmä: Reitinhaku on sekä videopeleissä että tekoälyn ja robotiikan puolella hyvin tuttu ongelma. Sen tutkimiseen on käytetty viime vuosina paljon resursseja lisääntyneen tekoälykiinnostuksen vuoksi. Tässä tutkielmassa keskitytään videopeleissä tapahtuvaan reitinhakuun, josta käsitellään lyhyesti sen historiaa, joitain käytettyjä algoritmeja ja pelialueen esitysmenetelmiä sekä vertaillaan niiden tehokkuutta keskenään. Tutkielma on kirjallisuuskatsaus, joka perustuu muihin tieteellisiin teoksiin. Keskeisinä tuloksina tulee esille alueen kolmioinnin sekä kolmiointia hyödyntävän TRA*-algoritmin vahvuudet verrattuna alueen ruudukkoesitykseen ja perinteiseen A*-algoritmiin. Tämän vuoksi pelialueet pitäisi esittää jonain muina kuin ruudukkoina, jotta tehokkaammat algoritmit voivat toimia paremmin.

Avainsanat: reitinhaku, videopeli, polunetsintä, A*-algoritmi, TRA*-algoritmi

Abstract: Pathfinding is a known problem in video games, artificial intelligence (AI) and robotics. In recent years more and more resources have been spent in researching AI and its problems because of growth in interest. In this study I focus on pathfinding in video games, whereof I briefly introduce its history, some algorithms that are used and how the game area is represented, and finally compare their efficiency. The study is a review that depends on other studies. The key results are the benefits of triangulation and TRA*-algorithm in comparison with using a grid map and A*-algorithm. Therefore game areas should be represented as something else than a grid map, so more efficient algorithms can perform better.

Keywords: pathfinding, video, game, A*-algorithm, TRA*-algorithm

Kuviot

Sisältö

1 Johdanto

Reitinhaku tarkoittaa kahden solmun (engl. *node*), alkusolmun ja maalisolmun, välisen reitin etsimistä. Reitinhakua kutsutaan myös polunetsinnäksi (engl. *pathfinding*). Hyväksytyn reitin löytämisen sijasta tärkeämpää voi olla vaatimuksiltaan optimaalisimman polun löytäminen (Mathew & Malathy 2015). Optimaalisuus voi tarkoittaa esimerkiksi reitinhaun nopeutta ja tarkkuutta.

Reitinhaku on videopeleissä hyvin tunnettu ja vaativa ongelma (Cui & Shi 2011). Entistä monimutkaisemmat ympäristöt ja suuremmat, reaaliaikaisesti liikuteltavat joukot ovat tuoneet esille muutamia luotettavia algoritmeja, kuten Dijkstra- ja syvyyshakualgoritmi (Cui & Shi 2011; Mathew & Malathy 2015). Näillä ei kuitenkaan löydetä haettavaa reittiä riittävän nopeasti, minkä vuoksi Hart, Nilsson ja Raphael (1968) esittelivät erään Dijkstra-algoritmin variaation, A*-algoritmin, joka on tämän hetken tunnetuimpia reitinhakualgoritmeja videopeleissä (Cui & Shi 2011).

Uudet, kehittyneemmät algoritmit, sekä aiemmin mainittu A*-algoritmi, perustuvat heuristisuuteen, jolloin kaikkia reittivaihtoehtoja ei lasketa tai käydä läpi, vaan esimerkiksi selvästi huonot vaihtoehdot karsitaan ratkaisujen joukosta. Tämä nopeuttaa algoritmin toimintaa huomattavasti, mutta käytetystä heuristiikasta riippuen ei välttämättä takaa enää kaikista lyhintä reittiä (Cui & Shi 2011; Hart, Nilsson & Raphael 1968).

Tässä tutkielmassa esitellään, kirjallisuuteen pohjautuen, perustasolla Dijkstra- ja A* -algoritmit. Tarkemmin käsitellään alueen kolmiointiin perustuvaa A*-algoritmin muunnosta ja sen hyötyjä videopelien reitinhaussa. Menetelmän englanninkielinen nimi on Triangular Reduction A*, ja tässä tutkielmassa siitä käytetään lyhennettä TRA*.

Aloitan aluksi kertomalla luvussa 2 miten reitinhakuongelma ilmenee videopeleissä, mikä tekee siitä monimutkaisen ja mitä menetelmiä ongelman ratkaisemiseksi tai helpottamiseksi käytetään. Luvussa 3 esittelen lyhyesti Dijkstra- ja A* -algoritmit ja niiden toiminnan käytännön tasolla. Luvussa 4 käsittelen TRA*-algoritmin tarkem-

min läpi sekä esittelen sen toimintaan liittyvän alueen kolmioinnin, jonka jälkeen luvussa 5 kerron sen hyödyistä ja mahdollisesti haitoista verrattuna muihin reitinhakualgoritmeihin. Lopuksi luvussa 6 on yhteenveto tutkielmasta, johon sisältyy myös mahdolliset johtopäätökset aiheesta.

2 Reitinhakuongelma videopeleissä

Videopelien rooli on ollut tärkeä tekoälyn (engl. *Artificial Intelligence, AI*) tutkimisessa ja reaalimaailman simuloinnissa (Botea, Bouzy, Buro, Bauckhage & Nau 2013). Tekoälyllä tarkoitetaan tässä yhteydessä pääasiassa ei-pelaaja-hahmojen (engl. *non-player-character, NPC*) toimintaa, kuten liikkumista ja yleistä päätöksentekoa erilaisissa tilanteissa. Pelihahmon automaattiseen liikkumiseen ja eri reittien valitsemiseen pelimaailmassa tarvitaan reitinhakua, jonka vuoksi se luokitellaan tekoälyksi. Reitinhaku on yksi videopelien tekoälyn tärkeimpiä ja vaativimpia aihealueita, jonka vuoksi se houkuttelee tutkijoita pariinsa. Erityisesti ongelmia tuottaa reitinhaun reaaliaikaisuus (engl. *real-time*), jolloin kaikki toiminta ja reitinlaskenta tapahtuu reaaliajassa. Tämän lisäksi alueella voi olla useita joukkoja, joille täytyy laskea reitit ottaen toiset joukot huomioon. Tämänlaisia pelejä kutsutaan reaaliaikaisiksi strategiapeleiksi (RTS-peleiksi, engl. *real-time strategy games*) (Synnaeve, Ontanon, Uriarte, Richoux, Churchill & Preuss 2013).

RTS on videopelien alalaji, jossa tarkoituksena on kerätä resursseja, rakentaa tukikohta ja koota armeija tuhotakseen vihollisen tukikohta (Synnaeve ym. 2013). Usein vihollisena toimii toiset pelaajat tai tekoäly. RTS-peleille ominaista on suuret, jopa satojen hahmojen suuruiset joukot, joiden kuljettaminen monimutkaisten ympäristöjen läpi osoittautuu todella hankalaksi ja laskennallisesti vaativaksi (Buro & Furtak 2004). Pelitilanteessa komentoja voivat antaa useat pelaajat, jonka seurauksena laskennallinen vaativuus suoraan moninkertaistuu. Uudet ja tehokkaammat reitinhakualgoritmit ovat välttämättömiä nopeatempoisen ja miellyttävän pelikokemuksen turvaamiseksi.

Tavallisesti reitinhakuongelmaa lähestytään algoritmien näkökulmasta, jolloin keksitään uusia ja tehokkaampia algoritmeja tai parannellaan jo tiedossa olevia algoritmeja. Joissain RTS-peleissä hyödynnetään algoritmien ohessa myös ohjautuvaa käyttäytymistä (engl. *steering behavior*), jolla saavutetaan yksilöiden tai joukkojen liikkuminen ympäristössä uskottavammin ja jopa ennakoivammin (Reynolds 1999; Synnaeve ym. 2013). Tämä tarkoittaa käytännössä esimerkiksi sitä, että joukkoa oh-

jattaessa joukon yksiköt asettuvat järjestykseen toisiinsa nähden (kuten kääntyvät samaan suuntaan ja ovat samalla etäisyydellä toisistaan) ja liikkuvat täten yhtenä joukkona (Reynolds 1999). Tämän avulla voidaan mahdollisesti helpottaa reitinhaussa ilmenevää laskentaa, jos usean reitin laskemisen sijaan käytetään joukolle yhtä reittiä (Synnaeve ym. 2013).

Optimaalinen reitinhaku ei kuitenkaan ole pelkän lyhimmän reitin etsimistä, vaan myös ympäristön ottamista huomioon. Esimerkiksi RTS-pelit Age of Empires II (1999) ja Civilization V (2010) ovat saaneet kritiikkiä huonosta reitinhausta, joka ilmenee esimerkiksi hahmojen jumiutumisena puihin, jos joukot käsketään metsän läpi kohteeseen (Cui & Shi 2011). Joukkojen kasvaessa ja metsien tiheytyessä ongelman ilmeneminen lisääntyy.

RTS-peleissä mallinnetaan tyypillisesti kaksiulotteinen (2D) avaruus. Monet nykypelit, kuten ensimmäisen persoonan ammuntapelit (FPS-pelit, engl. *first-person shooter games*) ja massiiviset monen pelaajan verkkopelit (MMO-pelit, engl. *massively multiplayer online games*) sen sijaan käyttävät kolmea ulottuvuutta (3D), jolloin nopeimman reitin löytäminen monimutkaisessa ympäristössä vaikeutuu. Cuin ja Shin (2011) mukaan näissä peleissä yhden hahmon liikkuminen on tärkeämmässä roolissa kuin esimerkiksi RTS-peleissä, joten liikkumisen täytyisi olla mahdollisimman ihmismäistä ja sulavaa.

3 Perinteiset reitinhakualgoritmit

Tavallisesti videopelien reitinhaussa on käytetty Dijkstra- ja A*-algoritmia. Molemmat algoritmit hyödyntävät reitinhaussa alueen ruudukkoesitystä, mutta A*-algoritmi on usein käytetympi sen hyödyntämän heuristiikan vuoksi.

3.1 Dijkstra-algoritmi

Dijkstra-algoritmi on perinteinen lyhimmän reitin ongelman ratkaisu. Sen toiminta perustuu kaikkien reittivaihtoehtojen laskemiseen, jonka vuoksi se löytää varmasti lyhimmän reitin solmujen välillä. Varsinkin nykypeleissä sen käyttö on tosin vähäistä, sillä sen aikavaativuus on $O(n^2)$, jolloin solmujen lukumäärän kasvaessa myös laskenta-aika kasvaa nopeasti. Tämän vuoksi muut vaihtoehdot, kuten A* ja sen muunnelmat, ovat osoittautuneet kannattavemmiksi.

Cuin ja Shin (2011) mukaan Dijkstra-algoritmin haasteena ovat suuret kartat ja alueet, jotka toteutetaan ruudukkoina ja joissa yksi ruutu kuvaa yhtä solmua. Tällöin kahden solmun välisen reitin etsiminen vaatii esimerkiksi 1000x1000 ruudukossa hyvin monen vaihtoehdon laskemisen, joten tehtävä osoittautuu laskennallisesti vaativaksi.

Joissain peleissä kartat voivat olla pienempiä ja yksinkertaisempia, jolloin lyhimmän reitin laskeminen ei ole pahimmassakaan tapauksessa nykytietokoneille kovin haasteellista. Lyhin reitti voi myös olla vaatimuksena esimerkiksi oppimispeleissä, joissa pelaajan tehtävänä on ratkaista kahden pisteen välinen lyhin reitti. Tämänkaltaisissa tilanteissa Dijkstra-algoritmi olisi järkevin vaihtoehto reitinhakuongelmaan.

3.2 A*-algoritmi

A*-algoritmi on nykyisin luultavasti käytetyin reitinhakualgoritmi videopeleissä varsinkin tekoälyn puolella (Cui & Shi 2011; Botea ym. 2013). Sen toiminta perustuu jo aiemmin mainittuun heuristisuuteen, jolloin se ei laske kaikkia mahdollisia reitte-

jä. Tämän vuoksi sen aikavaativuus voi vähentyä huomattavasti verrattuna esimerkiksi Dijkstra-algoritmiin. Jos A*-algoritmin heuristinen osa jätetään suorittamatta, se palautuu Dijkstra-algoritmiksi. A*-algoritmi takaa löytyvän reitin, jos reitti on olemassa. A*-algoritmi alustetaan käytännössä siten, että AUKI-listassa on vain alkusolmu, SULJETTU-lista on tyhjä ja g-,h- ja f-arvot ovat laskettu alkusolmun avulla. A*-algoritmin toimintaperiaate pseudokoodina on seuraava:

```
while AUKI ei ole tyhjä do
  s=solmu AUKI-listasta, jolla on pienin f-arvo
  if s==loppu then
    return
  end if
  poistetaan s AUKI-listasta
  lisätään s SULJETTU-listaan
  for all m on lapsisolmu(s) do
    if m on SULJETTU-listassa then
      jatketaan
    end if
    arvo=g(s)+etäisyys(s,m)
    if m on AUKI-listassa ja arvo<g(m) then
      poistetaan m AUKI-listasta koska uusi reitti on parempi
    end if
    if m on SULJETTU-listassa ja arvo<g(m) then
      poistetaan m SULJETTU-listasta
    end if
    if m ei ole AUKI-listassa eikä SULJETTU-listassa then
      lisätään m AUKI-listaan
      g(m)=arvo
      h(m)=heuristiikka(m,loppu)
      f(m)=g(m)+h(m)
    end if
  end for
```

end while
return virhe

A*-algoritmin pseudokoodissa AUKI-lista sisältää solmut, joissa on käyty mutta joita ei ole vielä laajennettu ja SULJETTU-lista sisältää solmut, joissa on käyty ja jotka on laajennettu. Arvo g(s) on reitti alkusolmusta solmuun s ja h(s) on heuristiikka, joka arvioi lyhimmän reitin solmusta s maalisolmuun. Näistä koottu arvo f(s) arvioi reittiä alkusolmusta maalisolmuun.

A*-algoritmin käytön hyöty riippuu myös jonkin verran siitä, millainen peli on kyseessä ja miten pelimaailma on siinä toteutettu. Jälleen kerran suuret ruudukkokartat osoittautuvat laskennallisesti haastaviksi, mutta esimerkiksi pienentämällä etsimisaluetta A*-algoritmia voidaan nopeuttaa (Cui & Shi 2011). Tämän lisäksi heuristisuutta muuttamalla A*-algoritmin toimintaa voidaan tehostaa, mutta se ei enää välttämättä takaa lyhintä reittiä.

Heuristisuuden muuttaminen on täysin tilanteen mukaista eikä mitään yleistä ratkaisua ole (Cui & Shi 2011; Hart, Nilsson & Raphael 1968). Esimerkkitapauksena voidaan käyttää euklidista etäisyyttä maalisolmuun, joka saadaan laskemalla pisteiden välinen etäisyys, jolloin ainoastaan paremmalta vaikuttavat vaihtoehdot otetaan tarkasteluun (Cui & Shi 2011; Hart, Nilsson & Raphael 1968).

Vaikka A*-algoritmi on laajasti käytetty ja yksi parhaita ratkaisuja reitinhakuongelmaan, sen heikkoutena on muistinkäyttö (Cui & Shi 2011; Mathew & Malathy 2015). Cuin ja Shin (2011) mukaan A*-algoritmin täytyy pitää muistissa jokaisen etsinnän eteneminen, joka vie paljon muistia varsinkin isoissa ympäristöissä.

4 TRA*-algoritmi

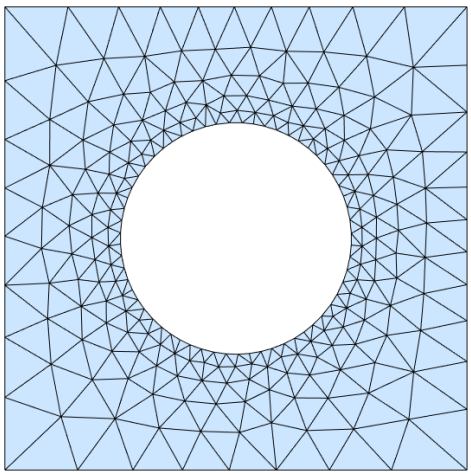
TRA*-algoritmi on Demyen ja Buron (2006) esittelemä algoritmi reitinhakuongelman ratkaisemiseksi. Algoritmin tavoitteena on pyrkiä älykkäämpään ja sitä kautta myös nopeampaan objektin liikkumiseen vähentämällä päätöksentekoa reittivaihtoehtojen välillä ja ottamalla huomioon liikkuvan objektin koon. Tämä saavutetaan käyttämällä alueen ruudukoinnin sijasta kolmiointia, jonka seurauksena esimerkiksi pyöreät objektit voidaan kuvata tarkemmin kuin ruudukoidussa alueessa (Demyen & Buro 2006).

4.1 Kolmiointi

Alueen esittäminen monikulmioina tarjoaa useita eri hyötyjä ruudukkoesitykseen verrattuna, kuten esimerkiksi kaikkien validien reittien säilyttämisen ja esteiden esittämisen juuri sellaisina kuin ne ovat (Demyen & Buro 2006). Alue jaetaan geometrialtaan yksinkertaisiin elementteihin, jotka ovat TRA*-algoritmin tapauksessa kolmioita. Vierekkäiset elementit, joita ovat tässä tapauksessa kolmiot, liittyvät toisiinsa pisteissä, joita kutsutaan solmuiksi. Verkko muodostuu solmuista ja niitä yhdistävistä kaarista, jotka ovat kolmioiden reunaviivoja. Reitinhaku tapahtuu tämänlaisessa verkossa. Kuvioissa 1 ja 2 on ympyränmuotoinen este. Geometrinen approksimointi on onnistunut tarkemmin kolmioilla, joiden koko vaihtelee (Kuvio 1) kuin keskenään samankokoisilla neliöillä toteutettuna (Kuvio 2).

TRA*-algoritmi hyödyntää kolmiointia, jota kutsutaan rajoitetuksi Delaunayn kolmioinniksi (engl. *Constrained Delaunay Triangulation, CDT*) (Kallmann 2005; Demyen & Buro 2006). Tavallisen Delaunayn kolmioinnin tarkoitus on muodostaa kolmiot seuraavanlaisesti:

```
Olkoon P joukko pisteitä tasolla.
Kolmiointi K on Delaunayn kolmiointi pisteistä P, jos
jokaiselle kolmioinnin K reunalle R on olemassa
ympyrä Y seuraavilla ominaisuuksilla:
```



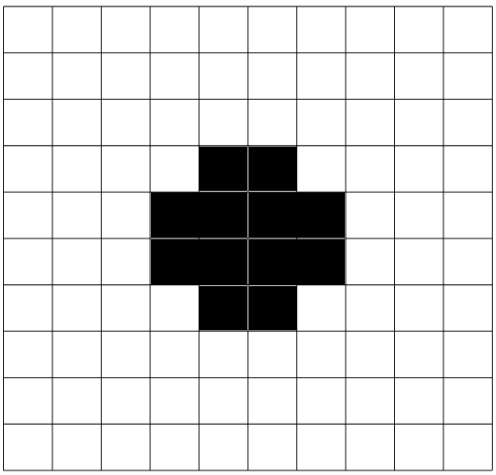
Kuvio 1. Alueen kolmiointi

1. Reunan R kärkipisteet ovat ympyrän Y rajoissa.
2. Mikään muu pisteiden P kärki ei ole ympyrän Y sisällä.

CDT lisää ehtoihin myös esteet, joihin ei saa reitinhaussa mennä (Chew 1989).

Delaunayn kolmiointiin liittyy Voronoin verkko, sillä ne ovat toistensa duaaliverkkoja. Voronoin verkko muodostuu siten, että jokaiselle pisteelle, kutsutaan näitä esimerkiksi emopisteiksi, luodaan oma alueensa, jossa jokainen alueen piste on lähempänä kyseistä emopistettä kuin mitään toista emopistettä. Yhdistämällä emopisteet, jotka jakavat saman reunan, muodostuu Delaunayn kolmiointi (Aurenhammer 1991). Erityisesti reitinhaussa Voronoin verkkoa voidaan hyödyntää suorittamalla etsintä reitinhakualgoritmilla kyseisessä verkossa.

Kallmann (2005) tarkentaa, että kolmioituun alueeseen liittyy kaksi osa-aluetta: vapaat kolmiot eli kolmiot, joiden kautta voidaan kulkea, ja esteet, joita kutsutaan myös rajoitteiksi (engl. *constraints*). Yhtenäiset, vapaat kolmiot muodostavat kanavan (engl. *channel*), josta reitinhakualgoritmi voi etsiä optimaalisimman reitin, jos reitti ylipäätään on olemassa. Kanavan sisältämä reitti ei välttämättä ole kaikista lyhin reitti kyseisellä alueella, mutta varsinkin peleissä kolmioinnin mahdollistama reitinvalinnan nopeus ja alueen muutoksiin sopeutuminen nousevat tärkeimmiksi



Kuvio 2. Alueen ruudukointi

prioriteeteiksi.

CDT:n yhtenä haittana voidaan pitää sen aikavaativuutta, joka on aiemmin käytetyimpien ratkaisujen tapauksessa usein neliöllinen (Kallmann, Bieri & Thalmann 2003). Kallmann, Bieri ja Thalmann (2003) esittelivät ratkaisuna tähän tehokkaamman menetelmän muuttuvan alueen kolmioimiselle, jonka vuoksi saavutetaan entistä paremmat tulokset reitinhaulle.

4.2 Algoritmin toiminta

Alueen kolmioinnin seurauksena syntyneet kolmiot esitetään abstraktissa graafissa solmuina, jossa niitä on helppo käsitellä. TRA*-algoritmissa solmut luokitellaan asteiden avulla nollasta kolmeen, joka kertoo vierekkäisten elementtien lukumäärän (Demyen & Buro 2006). Esimerkiksi kolmio, jonka aste on yksi, on naapurielementti yhden vapaan kolmion kanssa. Loput kaksi naapurielementtiä ovat esteitä. Kolmioiden astejaon avulla graafin vierekkäsiä elementtejä, esimerkiksi yhden asteen kolmioita, saadaan yhdistettyä, jolloin graafin pienentyessä myös reitinhaku helpottuu ja nopeutuu (Demyen & Buro 2006).

Objektin leveyden ottaminen huomioon on kolmioinnin ansiosta yksinkertaista. Kun objektit esitetään ympyröinä, niin osuminen seiniin ja kulmiin voidaan estää lisäämällä jokaiseen kolmion kulmiin objektin säteen mukainen ympyrä, jonka kautta ei voi kulkea (Demyen & Buro 2006). Tällä mahdollistetaan sujuva liikkuminen, jossa objekti ei jää jumiin kulmiin.

Kun lopullinen kolmioitu alue on ympyröineen muodostettu, niin vapaiden kolmioiden muodostaman kanavan läpi täytyy löytää lyhin reitti. Tämä reitti on mahdollista löytää lineaarisessa ajassa Hershbergerin ja Snoeyinkin (1994) esittelemällä suppiloalgoritmilla (engl. *funnel algorithm*).

5 Algoritmien vertailu

Demyen ja Buron (2006) kokeiden mukaan TRA*-algoritmi suoriutui useista reitinhakutehtävistä selvästi paremmin kuin A*- ja Sturtevantin ja Buron (2005) esittelemä PRA*-algoritmi (Partial Refinement A*) (Demyen & Buro 2006; Sturtevant & Buro 2005). Testit suoritettiin Baldur's Gate- ja Warcraft 3 -videopeleissä, joissa molemmissa käytetään ruudukkoesitystä kartasta. Tämän vuoksi alkuperäisillä, ruudukointiin perustuvilla algoritmeilla oli ns. etulyöntiasema uusiin algoritmeihin verrattuna. Siitäkin huolimatta TRA* oli joissain tapauksissa kymmeniä kertoja A*-algoritmia nopeampi pääasiassa kolmioinnin ansiosta, sillä sen avulla etsittävien solmujen lukumäärää saatiin vähennettyä hyvin paljon (Demyen & Buro 2006). Itse kolmiointi ei kuitenkaan vienyt aikaa kuin noin 20 mikrosekuntia kolmiota kohden (Demyen & Buro 2006).

TRA*-algoritmissa oli kuitenkin myös haittapuolia riippuen esitetystä ympäristöstä. Pienten esteiden runsas määrä, kuten Warcraft 3 -pelissä puukasaumat, aiheuttivat monia, lähes turhia laskutoimituksia. Reittivaihtoehtoja tuli paljon lisää, mutta näiden ero oli käytännössä hyvin pieni (Demyen & Buro 2006). Demyen ja Buro (2006) pyrki korjaamaan tilannetta muokkaamalla algoritmia hieman, jolloin ”kolmiota ei laajennettu kahdesti ennenkuin ensimmäinen reitti oli löydetty”. Näin ensimmäisten reittien löytämisestä tuli paljon nopeampaa.

6 Yhteenveto

Tässä työssä tarkastelin videopelien reitinhaun kannalta tärkeitä algoritmeja, joita ovat Dijkstra-, A*- ja TRA*-algoritmi, sekä vertailin niiden suurimpia hyötyjä ja haittoja keskenään. Työn tärkeimpänä havaintona tuli pelialueen tai -kartan mallintamisen hyödyt, jotka tulivat esille kolmioidun alueen ja sitä hyödyntävien algoritmien tapauksissa. Ruudukkoesityksen avulla alueen geometrian mallintaminen on kolmiointia epätarkempaa, ja kirjallisuudessa ruudukkoesitykseen pohjautuva reitinhaku osoittautuikin alueen kolmiointiin perustuvaa reitinhakua huonommaksi vaihtoehdoksi.

Videopelien reaaliaikaisuus, suuret alueet ja jopa sadat liikuteltavat hahmot pakottavat priorisoimaan ajankäytön suhteen, jolloin optimaalisin reitti ei välttämättä ole kaikkein nopein koko reittiavaruudessa. Tämän vuoksi perinteiset reitinhakualgoritmit vaativat uudentyyppisten alueen mallintamis- ja verkottamistekniikoiden hyödyntämistä, jotta ne pystyisivät kilpailemaan kehittyneempien algoritmien kanssa. Tulevaisuudessa tärkeimpänä tutkimuskohteena ei välttämättä ole enää algoritmien toiminnan muokkaaminen, vaan pelialueen optimaalisin esitystapa. Tässä tapauksessa kolmiointi on osoittautunut toimivaksi ratkaisuksi, mutta jokin muu tapa saattaa olla vielä tehokkaampi.

Kirjallisuutta

Aurenhammer, F. 1991. *Voronoi diagrams—a survey of a fundamental geometric data structure*. ACM Computing Surveys - Volume 23, Issue 3, s. 345–405. ACM.

Botea, A., Bouzy, B., Buro, M., Bauckhage, C. & Nau, D. 2013. *Pathfinding in Games*. Dagstuhl Follow-Ups - Volume 6, s. 21–31. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Buro, M. & Furtak, T. M. 2004. *RTS games and real-time AI research*. Proceedings of the Behavior Representation in Modeling and Simulation Conference (BRIMS) - Volume 6370. BRIMS.

Chew, P., L. 1989. *Constrained delaunay triangulations*. Algorithmica - Volume 4, Issue 1, s. 97–108.

Cui, X. & Shi, H. 2011. *A*-based pathfinding in modern computer games*. International Journal of Computer Science and Network Security - Volume 11, s. 125–130. IJCSNS.

Demyen, D. & Buro, M. 2006. *Efficient triangulation-based pathfinding*. AAAI'06 Proceedings of the 21st national conference on Artificial intelligence - Volume 1, s. 942–947. AAAI.

Hart, P., E., Nilsson, N., J. & Raphael, B. 1968. *A formal basis for the heuristic determination of minimum cost paths*. IEEE transactions on Systems Science and Cybernetics - Volume 4, Issue 2, s. 100–107. IEEE.

Hershberger, J. & Snoeyink, J. 1994. *Computing minimum length paths of a given homotopy class*. Computational Geometry - Volume 4, Issue 2, s. 63–97.

Kallmann, M. 2005. *Path planning in triangulations*. Proceedings of the IJCAI workshop on reasoning, representation, and learning in computer games, s. 49–54. IJCAI.

Kallmann, M., Bieri, H. & Thalmann, D. 2003. *Fully Dynamic Constrained Delaunay Triangulations*. Geometric Modeling for Scientific Visualization, s. 241–257. Springer Berlin Heidelberg.

Mathew, G., E. & Malathy, G. 2015. *Direction based heuristic for pathfinding in video games*. 2015 2nd International Conference on Electronics and Communication Sys-

tems (ICECS), s. 1651–1657. ICECS.

Reynolds, C., W. 1999. *Steering behaviors for autonomous characters*. Game developers conference, s. 763–782.

Sturtevant, N. & Buro, M. 2005. *Partial pathfinding using map abstraction and refinement*. Proceeding AAAI'05 Proceedings of the 20th national conference on Artificial intelligence - Volume 3, s. 1392–1397. AAAI.

Synnaeve, G., Ontanon, S., Uriarte, A., Richoux, F., Churchill, D. & Preuss, M. 2013. *A survey of real-time strategy game ai research and competition in starcraft*. IEEE Transactions on Computational Intelligence and AI in games - Volume 5, s. 293–311. IEEE.