Heuristiset reitinhakualgoritmit videopeleissä: vertailu ja käyttökohteet

Turun yliopisto
Tietotekniikan laitos
TkK-tutkielma
Ohjelmistotekniikka
Toukokuu 2025
Jaakko Pönnelin

TURUN YLIOPISTO
Tietotekniikan laitos

Jaakko Pönnelin: Heuristiset reitinhakualgoritmit videopeleissä: vertailu ja käyttökohteet

TkK-tutkielma, 28 s.
Ohjelmistotekniikka
Toukokuu 2025

Monissa videopeleissä käytetään reitinhakualgoritmeja liikuttamaan hahmoja, jotka eivät ole pelaajan ohjaamia. Aikaisemmin reitinhakualgoritmit etsivät maalin sokkona, mutta nykyään varsinkin videopeleihin käytetään heuristisia reitinhakualgoritmeja, joilla on tietoa maalin sijainnista. Heuristiset reitinhakualgoritmit ovat lähes aina nopeampia, resurssitehokkaampia ja yleensä löytää lyhyimmän mahdollisen reitin. Siitä huolimatta pelinkehittäjät saattavat valita vanhan algoritmin uuden sijaan, koska ne esiintyvät useammin kirjallisuudessa. Tässä tutkielmassa esitetään eri heuristisia reitinhakualgoritmeja, joita kirjallisuudessa esiintyy. Näiden reitinhakualgoritmien toiminta sekä vahvuudet ja heikkoudet selitetään. Lisäksi näille reitinhakualgoritmeille annetaan esimerkkejä käyttökohdemahdollisuuksista. Tutkielmassa myös lajitellaan heuristiset reitinhakualgoritmit toimintaperiaatteiden mukaan. Esitetyt luokat ovat helposti ymmärrettäviä, jotta pelinkehittäjät voivat löytää nopeasti ja helposti omaan peliin soveltuvan reitinhakualgoritmin. Esitettyjen luokkien reitinhakualgoritmeilla on samankaltainen käyttökohde, johon konkreettisena esimerkkinä annetaan videopeli, jossa reitinhakualgoritmia voitaisiin käyttää.

Asiasanat: heuristinen, informoitu, reitinhaku, algoritmi, videopelit, pelinkehittäjä

UNIVERSITY OF TURKU
Department of Computing

Jaakko Pönnelin: Heuristiset reitinhakualgoritmit videopeleissä: vertailu ja käyttökohteet

Bachelor's Thesis, 28 p.
Software Engineering
May 2025

Many video games use pathfinding algorithms to move characters that are not controlled by any player. In the past, pathfinding algorithms used blind search to find the goal, but nowadays heuristic pathfinding algorithms, which have some information about the position of the goal, are commonly used in video games. Heuristic pathfinding algorithms are almost always faster, more resource efficient, and generally find the shortest possible path. Despite this, game developers may choose an older algorithm instead of a newer one, because they show up more in literature. This thesis presents various heuristic pathfinding algorithms found in literature. The operation, as well as strengths and weaknesses of these pathfinding algorithms are explained. In addition, examples of potential use cases for these algorithms are provided. The thesis also categorizes heuristic pathfinding algorithms based on their operating principles. The presented categories are easy to understand, to let game developers quickly and easily find a suitable pathfinding algorithm for their game. The pathfinding algorithms in the presented categories share a similar use case, and a video game where the algorithm could be used is provided as a concrete example.

Keywords: heuristic, informed, pathfinding, algorithm, video games, developer

Sisällys

1 Johdanto

Monissa videopeleissä esiintyy useita eri hahmoja. Pelaaja saattaa ohjata yhtä tai useampaa hahmoa, mutta pelaaja ei välttämättä ohjaa kaikkia hahmoja itse. Hahmot, joita pelaaja ei itse ohjaa ovat NPC-hahmoja (engl. non-playable character). Moninpeleissä hahmoja saattaa ohjata toiset pelaajat, mutta NPC-hahmoja, joita kukaan ei liikuta manuaalisesti, täytyy kuitenkin liikuttaa jollain muulla tavalla. Tähän käytetään algoritmeja. Reitinhakualgoritmi on erikoistunut löytämään reitti annetusta lähtöpaikasta maaliin, ja ne pyrkivät löytämään lyhimmän reitin käyttämällä mahdollisimman vähän aikaa ja tietokoneen resursseja [1]. Yhtä ainoaa parasta reitinhakualgoritmia ei ole, vaan algoritmi täytyy valita tarpeiden mukaan.

Eri peleissä voidaan vaatia hyvin erilaisia algoritmeja. Esimerkiksi FPS-pelit (engl. first-person shooter) ovat hyvin eri kaltaisia RPG-pelien (engl. role-playing game) kanssa. Esimerkiksi kolmiulotteisissa peleissä täytyy ottaa huomioon korkeuserot, joita kaksiulotteisissa peleissä ei tarvita ja muuttuvassa kartassa ei ole hyötyä käyttää algoritmia, joka laskee reitin alussa, mutta ei päivitä sitä. Staattiselle kartalle tällainen algoritmi sopii hyvin ja reitti voidaan jopa laskea etukäteen. Samassa pelissä voidaan myös käyttää useita eri reitinhakualgoritmeja eri tarkoituksiin.

Reitinhakualgoritmeja voidaan lajitella monin eri tavoin. Eräs tapa lajitella ne on jakaa ne epäinformoituihin (engl. uninformed), informoituihin tai heuristisiin (engl. informed/heuristic) sekä metaheuristisiin (engl. metaheuristic). Epäinformoidut reitinhakualgoritmit eivät tiedä mitään maalin sijainnista. Tästä syystä ne ovat usein

hitaampia kuin heuristiset algoritmit, joilla on tietoa maalin suunnasta. [1] Heuristiset algoritmit saattavat löytää vain paikallisen minimin, minkä takia kehitettiin metaheuristiset algoritmit. Metaheuristiset algoritmit ovat samankaltaisia heuristisen algoritmien kanssa, mutta toimivat ympäristöstä huolimatta. [2] Ne vaativat kuitenkin enemmän aikaa ja resursseja, minkä takia videopeleissä yleensä suositaan heuristisia reitinhakualgoritmeja [1].

Uusia parempia reitinhakualgoritmeja kehitetään koko ajan, minkä takia algoritmit vanhenevat. Osa reitinhakualgoritmeista, kuten A*, yleistyivät ja niitä käytetään edelleen videopeleissä, vaikka uudempia ja parempia reitinhakualgoritmeja on saatavilla [3], [4]. Heuristisia reitinhakualgoritmeja on paljon, mutta siitä huolimatta vain harva niistä on tunnettu. Monissa kirjallisuuskatsauksissa, kuten [1], [5] ja [6], vertaillaan epäinformoituja algoritmeja heuristisen algoritmien kanssa, etenkin A*:n kanssa. Näissä kirjallisuuskatsauksissa päädytään samaan lopputulokseen: heuristiset reitinhakualgoritmit ovat parempia lähes joka tilanteessa. Kirjallisuudessa ei ole paljon vertailtu heuristisia reitinhakualgoritmeja keskenään, minkä takia pelinkehittäjien on vaikea löytää omaan peliinsä sopivaa reitinhakualgoritmia.

Tässä kirjallisuuskatsauksessa tarkastellaan eri heuristisia reitinhakualgoritmeja ja vertaillaan niiden tarkkuutta, nopeutta ja resurssien käyttöä. Suuri osa heuristisista reitinhakualgoritmeista perustuu A*-algoritmiin, joten siihen perustuvia algoritmeja käsitellään paljon. Eri käyttötarkoituksiin annetaan sopivia reitinhakualgoritmeja, minkä perusteella pelinkehittäjät voivat valita omaan peliinsä sopivan algoritmin tai etsiä kyseisestä algoritmista sopivampaa tai päivitettyä versiota. Työn tutkimuskysymykset ovat:

1. Mitä videopeleihin soveltuvia heuristisia reitinhakualgoritmeja akateemisessa kirjallisuudessa esiintyy?

2. Mihin käyttötarkoituksiin eri heuristiset reitinhakualgoritmit soveltuvat videopeleissä?

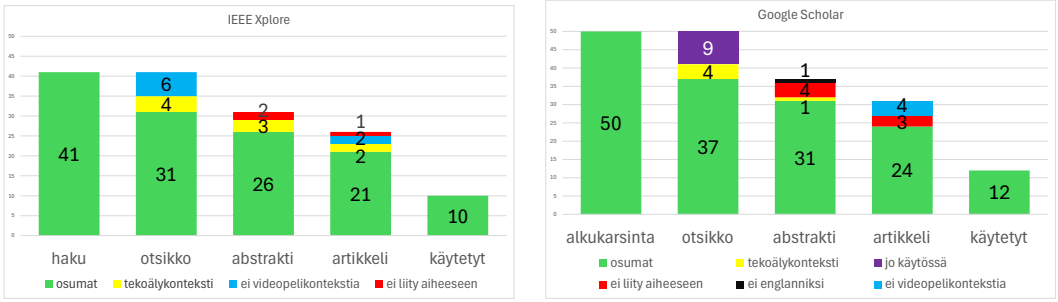
1.1 Hakumenetelmät

Aineistohaku on tehty IEEE Xplore sekä Google Scholar -metatietokannoissa. IEEE Xplore on IEEE:n hallitsema tietokanta, joka koostuu yli 6 miljoonasta tieteellisestä tieto- ja sähkötekniikan aineistosta [7]. Google Scholar on Googlen omistama hakukone tutkimusaineistoille ja se pyrkii antamaan osuvimmat aineistot ensiksi. [8] Aineisto haettiin molemmissa tietokannoissa hakulausekkeella: heuristic AND pathfinding AND algorithm* AND "video game*". Google Scholarissa osumia löytyi 1580, mikä on liikaa tähän tutkielmaan. Google Scholar antaa ensiksi osuvimmat haut, minkä takia siitä valittiin ensimmäiset 50 osumaa. IEEE Xplore -tietokannassa ei tarvinnut tehdä tällaisia toimenpiteitä, sillä hakusanalla saatiin 41 osumaa.

Näiden rajausten jälkeen käytiin läpi kaikki lähdeaineistot ja niistä karsittiin pois aineistot, jotka eivät sopineet tutkielman aiheeseen (Kuva 1.1). Ensiksi aineistoja karsittiin otsikon perusteella. IEEE Xplore -tietokannassa karsittiin 4 aineistoa, jotka liittyivät tekoälyyn ja 6 aineistoa, joissa ei ollut videopelikontekstia. Google Scholar -tietokannassa oli 4 tekoälyyn liittyvää aineistoa sekä 9 aineistoa, jotka löytyivät jo IEEE Xplore -tietokannasta.

Tiivistelmän perusteella karsittiin IEEE Xplore -tietokannassa 3 tekoälyyn perustuvaa aineistoa sekä 2 tutkielman aiheen ulkopuolella olevaa aineistoa. Google Scholar -tietokannassa taas karsittiin 1 tekoälyyn liittyvä aineisto, 4 aiheen ulkopuolella olevaa aineistoa sekä 1 aineisto, joka ei ollut englanninkielinen.

Lopulta koko tekstin perusteella karsittiin IEEE Xplore -tietokannasta 1 aiheen ulkopuolella oleva aineosta, 2 aineistoa, joissa ei ollut videopelikontekstia sekä 2 tekoälykontekstia. Google Scholar -tietokannassa oli 3 aiheen ulkopuolella olevaa aineistoa sekä 4 aineistoa, joissa ei ollut videopelikontekstia. Tämän jälkeen osumia oli 45 kappaletta, joista tutkielmassa käytettiin 22.



(a) Lähteiden haku sekä rajaus IEEE Xplore -metatietokannassa.

(b) Lähteiden haku sekä rajaus Google Scholar -metatietokannassa.

Kuva 1.1: Lähteiden haut ja rajaukset käytetyissä metatietokannoissa.

1.2 Tutkielman rakenne

Luvussa 2 esitellään eri heuristisia reitinhakualgoritmeja. Ensiksi esitellään suosittu A*-algoritmi ja kerrotaan sen toiminnasta. Sen jälkeen esitellään siihen perustuvia algoritmeja.

Luvussa 3 käsitellään eri peligenrejä ja käyttökohteita, missä reitinhakualgoritmeja tarvitaan ja jokaiselle pelille ja käyttökohteelle annetaan reitinhakualgoritmeja, jota siinä voisi mahdollisesti käyttää. Kaikki valinnat perustellaan algoritmien hyötyjen ja haittojen perusteella, jotta pelinkehittäjät voivat valita sopivan algoritmin videopeleilleen.

Luvussa 4 on pohdinta tutkielmasta. Luvussa analysoidaan tutkielman kontribuutioita sekä implikaatioita sekä tarkastellaan tutkielman rajoitukset sekä jatkotutkimus. Lopuksi luvussa on yhteenveto ja johtopäätelmät.

2 Heuristiset reitinhakualgoritmit videopeleissä

Eräs hyvin tunnettu heuristinen algoritmi on P. Hartin ehdottama A* (lausutaan A tähti), joka löytää lyhimmän reitin kahden ruudun välillä. Algoritmi perustuu epäinformoituun Dijkstran algoritmiin, mutta A*-algoritmilla on tieto, missä maali on. [9] A*-algoritmia on vertailtu paljon epäinformoituihin algoritmeihin, kuten Dijkstraan ja BFS-algoritmiin (engl. breadth first search). Lähes jokaisessa testissä epäinformoituja algoritmeja vastaan A* löysi reitin nopeammin, kuin muut algoritmit. A*:n löytämä reitti on myös yhtä lyhyt tai lyhyempi kuin vertailtavien algoritmien. A*:n heuristiikan suorittamisesta johtuen pienissä kartoissa A* saattoi olla hitaampi, mutta vain vähän. [5], [10] Usein videopeleissä vaaditaan suurempia karttoja, joten tämä ei ole yleensä ongelma. Koska monet heuristiset algoritmit perustuvat A*-algoritmiin, on tärkeää ymmärtää, miten se toimii. Muiden algoritmien toimintaa ei käsitellä yhtä tarkasti, mutta niitä vertaillaan A*-algoritmiin.

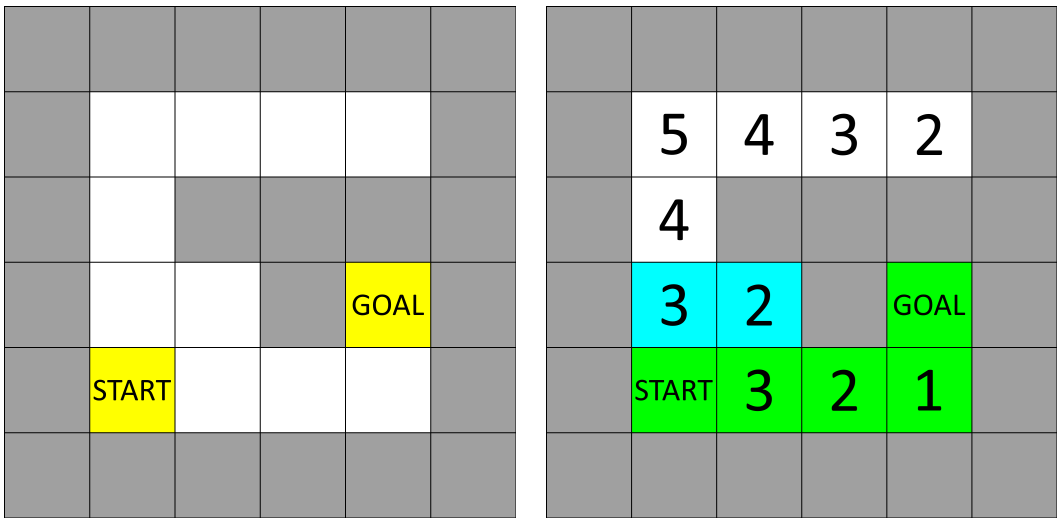
Algoritmit on lajiteltu kolmeen kategoriaan: A*-, D*- ja Theta*-perusteisiin algoritmeihin. D*- ja Theta* perustuvat myös A*:een, mutta niillä on selvästi eri tarkoitus. D* toimii dynaamisessa kartassa ja Theta* voi kulkea missä tahansa kulmassa. Monet algoritmit häivyttävät näiden kategorioiden rajoja, minkä takia ne on lajiteltu yksinkertaisesti alkuperäisen algoritmin perusteella.

2.1 A*- ja siihen perustuvat algoritmit

A*-algoritmi etsii annetun lähtö- ja maalisolmun perusteella lyhimmän reitin niiden välillä, jos sellainen on [3]. Ohjelmalistaus 1 esittää A*-algoritmin pseudokoodia, jossa on esitetty tarkasti sen toiminta. Ohjelmalle annetaan lähtösolmu, joka lisätään open-listaan. Tämän jälkeen algoritmi tarkastaa solmun viereiset solmut ja valitsee näistä parhaimman vaihtoehdon funktion $f(n) = g(n) + h(n)$ mukaisesti, jossa $f(n)$ on kumulatiivinen etäisyys sekä hinta, $g(n)$ on hinta lähdöstä solmuun n ja $h(n)$ on arvioitu hinta solmusta n maaliin [11]. P. Hart on todistanut, että A* löytää lyhimmän reitin aina, jos $h(n)$ arvon hinta on yhtä halpa tai halvempi kuin oikeasti halvin reitti maaliin. Lisäksi P. Hart on todistanut, että A* käyttää kyseistä heuristista funktiota optimaalisesti, eli vähempää solmuja ei voi avata kyseisellä heuristiikalla. Heuristiikkaa täytyy siis muuttaa, jos halutaan nopeampi tai resurssitehokkaampi algoritmi. [3]

Ohjelmalistaus 1 A*-algoritmin pseudokoodi. [12], [3]

```
add start node to open list
while open list is not empty:
    find node with lowest f value in open list
    set it as current node
    move current node to closed list from open list
    if current node is goal:
        calculate path and stop execution
    end if
    for each neighbor of current node not in closed list:
        if neighbor not in open list:
            add neighbor to open list
            set current node as its parent
            calculate its f, g and h values
        else if neighbor in open list and new path is cheaper:
            change neighbor's parent to current node
            recalculate its f, g and h values
        end if
    end for
end while
```



(a) A*-algoritmille annettu kartta.

(b) A*-algoritmin heuristiikka, avaamat solmut sekä lopullinen reitti.

Kuva 2.1: A*-algoritmin toiminta. Kuvat on tehnyt tutkielman kirjoittaja.

Kuva 2.1 esittää A*-algoritmia, joka voi liikkua vain vaaka- ja pystysuoraan. Valkoisella merkityt ruudut ovat vapaita ruutuja ja harmaalla merkityt ruudut ovat puolestaan seiniä. Lähtö- ja maaliruudut ovat merkitty keltaisella. Sinivihreät sekä vihreät ruudut ovat A*:n avaamat solmut. Ensimmäiseksi A* avaa ruudut, joissa etäisyys maaliin on kolme. Tämän jälkeen uusista solmuista avataan aina lähempänä maalia olevat solmut, kunnes saavutaan maaliin. Vihreät ruudut esittävät reittiä, jonka algoritmi löytää.

Koska A* löytää optimaalisen reitin nopeasti, soveltuu se hyvin moniin erilaisiin tarkoituksiin. Ruudukolla A* voi liikkua vain vaaka- ja pystysuoraan tai sen lisäksi viistoon [13], mutta A*:eä voidaan käyttää myös navigointiverkoissa (engl. Navigation Mesh). Navigointiverkot käyttävät kuperia monikulmioita kartan esittämiseen ruudukon sijaan, jolloin hahmot voivat liikkua monikulmion kärkien välillä vapaasti osumatta seiniin. [14] Vaikka A* soveltuu monipuolisesti eri käyttötarkoituksiin, on olemassa parempia A*:een perustuvia eri käyttötarkoituksiin erikoistuneita algoritmeja, joita käsitellään myöhemmin tutkielmassa.

Suuntaan perustuva heuristiikka

Suuntaan perustuva heuristiikka (engl. Direction based heuristic) on A*-algoritmiin perustuva algoritmi, jonka G. Mathew on esittänyt. Solmujen heuristiikka on etäisyyden sijaan vain tieto maalin suunnasta. A* tarkistaa aina kaikki solmun naapurit, mutta suunnan avulla voidaan tarkistaa vain maaliin suunnassa olevat solmut. Algoritmi löytää lyhimmän reitin A*:n tapaan, mutta nopeammin ja avaa vähemmän solmuja kuin A*. G. Mathew tutki suuntaan perustuvaa heuristiikkaa vain suorakulmaisessa ruudukossa, mutta algoritmia voidaan helposti soveltaa muihinkin ruudukkoihin. [15]

Suuntaan perustuvaa heuristiikkaa voidaan soveltaa samoin kuin A*:eä, mutta vain ruudukkomaisissa kartoissa. Navigointiverkkoihin se ei välttämättä sovellu, sillä navigointiverkossa maalin suunta ei ole yhtä selvä, kuin ruudukossa. Suuntaan perustuva heuristiikka käyttää vähemmän resursseja, kuin A* ja sopii suurempiin karttoihin. [15]

CentA*-algoritmi

CentA*-algoritmi (engl. Centroid A*) on A*:n kanssa hyvin samanlainen algoritmi. Ennen reitinhakua CentA* tarkistaa lähtö- ja maalisolmun viereiset solmut ja valitsee näistä parhaimmalta vaikuttavat solmut uusiksi lähtö- ja maalisolmuiksi. Tämän jälkeen algoritmit käyttää A*:eä reitinhakuun uusista solmuista. CentA* on hieman nopeampi ja resurssitehokkaampi kuin A*, mutta ei paljon. [16]

CentA* soveltuu hyvin samoihin käyttötarkoituksiin, kuin A* ja suuntaan perustuva heuristiikka. Vaikka suuntaan perustuva heuristiikka on luultavasti parempi kuin CentA*, centA*:eä voidaan soveltaa helposti myös navigointiverkoissa toisin kuin suuntaan perustuvaa heuristiikkaa. [15], [16]

TBA*-algoritmi

TBA*-algoritmin (engl. Time Bounded A*) löytämä reitti on samankaltainen A*:n kanssa. Molemmat algoritmit etsivät reitin aloitussolmusta loppusolmuun, mutta TBA* eroaa A*:stä siten, että TBA* liikkuu samalla, kun reitinhaku on kesken. Tietyin väliajoin, kun reitinhaku on edennyt useamman solmun, TBA* pysäyttää haun. Sen jälkeen NPC-hahmo liikkuu lähimpänä maalia olevaan solmuun, jonka algoritmi on siihen mennessä löytänyt. TBA* on siis reaaliaikainen algoritmi, eli liikkuu reaaliajassa huolimatta siitä, kuinka suuri kartta on. [11]

TBA* on hyödyllinen tilanteissa, jossa NPC:n oletetaan liikkuvan heti. Pienissä kartoissa eroa A*:een ei välttämättä huomaa, mutta suurissa kartoissa A*:llä kestää kauemmin laskea reitti ja lähteä liikkeelle. Tietokoneen resurssien käyttö on myös TBA*:n vahvuus, sillä reitinhaku keskeytetään väliajoin, mikä vähentää avattujen solmujen määrää. Tämä voi olla tärkeää, jos peli on muuten vaativa resurssien suhteen. [11]

LRTA*-perusteiset algoritmit

LRTA*-algoritmi (engl. Learning Real-Time A*) on reaaliaikainen heuristinen reitinhakualgoritmi, joka oppii reitinhaun aikana. Samoin kuin muuta reaaliaikaiset reitinhakualgoritmit, LRTA* ei etsi koko reittiä maaliin, vaan pysäyttää etsinnän ja kulkee parhaimmalta vaikuttavaan solmuun. LRTA* laskee heuristiikkaa kahdella tavalla. Ensimmäinen on etäisyys ja hinta, kuten A*, mutta toinen on käyttämällä uusia hintoja jo käydyistä solmuista. [17] Se muuttaa vain yhden solmun hintaa per liike, minkä takia se ei ole hyödyllinen videopeleissä. LRTA*:n ongelma on hankaus (engl. scrubbing), eli samojen solmujen välillä kulkeminen edestakaisin, mikä ei ole suotavaa reitinhakualgoritmeille. [18].

LRTA*:stä kehitettiin myöhemmin LSS-LRTA*-algoritmi (engl. Local Search Space Learning Real-Time A*). LSS-LRTA* parantaa alkuperäistä kahdella tavalla.

LSS-LRTA* oppii paljon enemmän, eli päivittää solmujen hintoja. LRTA* päivittää vain käydyn solmun hintaa, mutta LSS-LRTA* päivittää kaikkien avattujen solmujen hinnat. LSS-LRTA* liikkuu myös monta solmua reitinhakujen välissä, toisin kuin LRTA*, mikä nopeuttaa algoritmia. LSS-LRTA* löytää paljon lyhyempiä reittejä kuin LRTA* ja on jopa kilpailukykyinen D* Lite -algoritmia vastaan. E. Burns et al. paransivat LSS-LRTA*:n reitinhakuaikaa useammalla eri tavalla. He osoittivat, että LSS-LRTA* löytää maalin nopeammin, jos se tekee vain yhden liikkeen per suunnitteluvaihe. Lisäksi he osoittivat, että A*:n harhojen poisto (engl. de-biasing) vähentää maaliin pääsemisen aikaa huomattavasti. [17]

LRTA*:stä on myös kehitetty D LRTA* algoritmi, joka yritti poistaa oppimisvaiheen LRTA*:stä. V. Bulitko et al. kehittivät D LRTA*:n pohjalta kNN LRTA*-algoritmin (engl. k Nearest Neighbor Learning Real-Time A*), jonka tarkoitus oli yksinkertaistaa ja nopeuttaa D LRTA*-algoritmia. Algoritmissa on esiprosessointivaihe ja varsinainen reitinhaku. Algoritmi pyrkii myös välietappien (engl. subgoal) avulla poistaa hankauksen. Esiprosessointivaiheessa lasketaan tietokanta välietapeista lähimpien naapureiden perusteella. Varsinaisessa reitinhaussa kNN LRTA* etsii tietokannasta samankaltaisia reittejä ja sen perusteella valitsee mitä reitinhakualgoritmia käyttää. Algoritmi voi käyttää A*:eä tai mäenkiipeämisalgoritmia (engl. Hill-Climbing algorithm), joka kulkee aina parasta mahdollista solmua kohden, riippumatta onko se nopein reitti. Algoritmia voidaan myös käyttää liikuttamaan useaa hahmoa. [18]

LRTA*:een perustuvat reitinhakualgoritmit soveltuvat muiden reaaliaikaisten algoritmien kanssa tilanteisiin, jossa halutaan hahmon liikkuvan heti. LSS-LRTA* ja kNN LRTA*-algoritmeja ei ole vertailtu keskenään ja ne ovat esitetty eri metriikoilla, joten ei ole selvää kumpi näistä on parempi, mutta kNN LRTA* on parempi, jos sitä sovelletaan usean hahmon liikuttamiseen [18].

MA-HEHA*-algoritmi

Y. Yiu et al. ovat kehittäneet MA-HEHA*-algoritmin (engl. Multi-Agent Hierarchical Evolutionary Heuristic A*) heidän aikaisemman HEHA*-algoritmin perusteella. MA-HEHA* on erikoistunut liikuttamaan useaa hahmoa. S koostuu esiprosessointivaiheesta sekä varsinaisesta reitinhakuvaiheesta. Esiprosessoinnissa algoritmi klusteroi karttaa ja kategorisoi klusterit eri tasoihin laskemalla, kuinka mahdollisesti hahmojen välillä tapahtuu törmäyksiä kyseisessä klusterissa. Varsinaisessa reitinhaussa hahmot etsivät ensin reitin abstraktoidusta kartasta ja sen jälkeen etsivät varsinaisen polun käyttäen A*:eä. Kauimpana maalista olevilla hahmoilla on etuajo-oikeus, mikä vähentää hahmojen liikkellä olevaa aikaa. [19]

MA-HEHA* on A*:stä erikoistunut malli, joka pystyy liikuttamaan useita hahmoja samaan aikaan. MA-HEHA* osaa myös ottaa huomioon erilaiset moniagenttireitinhakuongelmat (engl. multi-agent pathfinding problem) huomioon ja löytää optimaaliset reitit A*:n avulla hyvällä todennäköisyydellä. Sadasta hahmosta noin 60-90 prosenttia löytää optimaalisen reitin. [19]

BMAA*-algoritmi

D. Sigurdson et al. kehittämä BMAA*-algoritmi (engl. Bounded Multi Agent A*) on reaaliaikainen reitinhakualgoritmi, joka voi liikuttaa monia hahmoja samaan aikaan. Jokainen hahmo etsii reittiä eteenpäin tietyn määrän solmuja TBA*:n tyylisesti. BMAA* käyttää kuitenkin RTAA*-algoritmia (engl. Real-Time Adaptive A*). Algoritmissa voidaan valita, kuinka paljon eteenpäin hahmot etsivät ennen liikkumista. BMAA*:n reitit ovat pidempiä, kuin aikaisempien algoritmien, mutta useampi hahmo löytää maaliinsa nopeammin. [12]

BMAA* on hyödyllinen, kun halutaan vähentää tietokoneen resursseja ja halutaan hahmojen lähtevän liikkeelle heti riippumatta kartan koosta. Koska BMAA* on erikoistunut usean hahmon siirtoon, se sopii peleihin, jossa on paljon NPC-hahmoja,

joiden täytyy liikkua reaaliaikaisesti yhtä aikaa. Algoritmi on myös hyvä, jos kartta muuttuu dynaamisesti tai hahmojen maalin sijainti muuttuu reitinhaun aikana. BMAA* pystyy toimimaan satojen ja jopa tuhansien hahmojen kanssa. Tuhannesta hahmosta BMAA* onnistuu löytämään noin 80 prosentille reitin. Algoritmille umpikujat ovat ongelma. [12]

KM-A* ja HKMA-algoritmit

A*-algoritmi toimii hyvin pienistä kartoista suuriin, mutta todella suurissa kartoissa avattavia solmuja on paljon, minkä takia A*:stä on kehitetty KM-A* -algoritmi (engl. K-Means A*). KM-A* jakaa reitinhaun kahteen osaan. Ensin suuri kartta jaetaan pienempiin osiin esikäsittelyssä käyttäen k-keskiarvo klusterointia (engl. k-means clustering). K-keskiarvo klusterointi pyrkii jakamaan avarat ja kapeat alueet erilleen, jolloin klusterin sisäinen reitinhaku on yksinkertaisempaa. Tämän jälkeen A*-algoritmi valitsee reitin klustereiden välillä ja lopuksi tarkan reitin jokaisen klusterin sisällä. [20]

C. Chen et al. kehittivät HKMA-algoritmin (engl. Hierarchical K-Means A*), joka parantaa alkuperäisen algoritmin klusterointimetodia. KM-A* tarvitsee kolme syötettä: kartan tiedot, pelinkehittäjän valitsema k-arvo ja satunnaisesti valitut alkuklustereiden keskipisteet. HKMA sen sijaan tarvitsee vain kartan tiedot ja laskee muuten kaiken itse. Klusterointi kestää muutaman minuutin kauemmin, mutta reitinhakualgoritmi on huomattavasti parempi. HKMA löytää reitin nopeammin ja reitti on myös lyhyempi ja lisäksi se käyttää vähemmän tietokoneen muistia. [20]

Suurissa kartoissa reitinhakualgoritmien isoin ongelma on tietokoneen muistin ja prosessorin käyttö. Klusterointi auttaa vähentämään suurissa kartoissa resurssien käyttöä, joten KM-A* ja HKMA toimivat parhaiten, kun pelissä on vain suuria karttoja. Pienissä kartoissa klusterointi aiheuttaa turhaa esiprosessointia ja voi vaikuttaa reitinhaun kestoon sekä löydetyn reitin pituuteen negatiivisesti.

DEC-A*-algoritmi

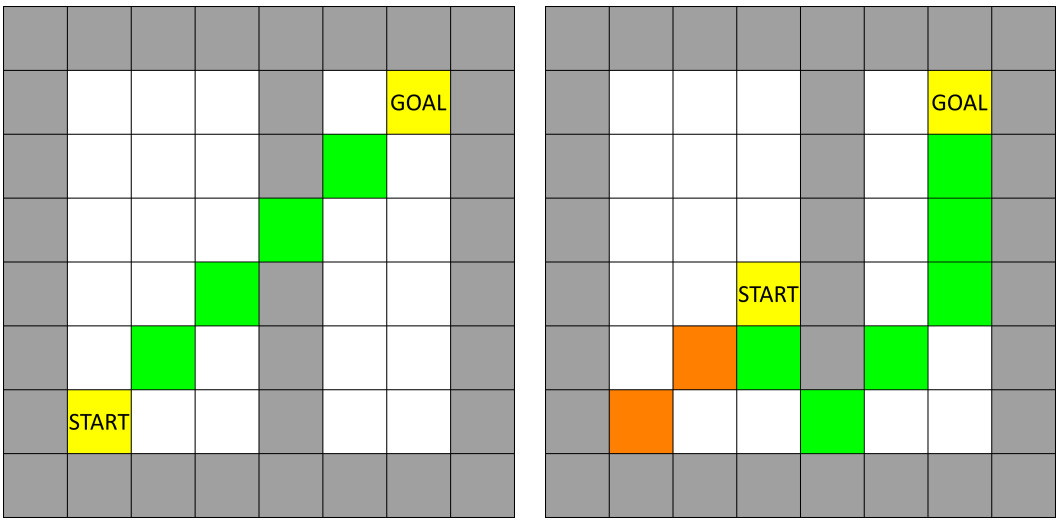
DEC-A*-algoritmim (eng. Decentralized A*) kehitti M. Falou et al. ja se on A*:stä parannettu versio. DEC-A* löytää yhtä lyhyen reitin kuin A*, mutta nopeammin suurissa kartoissa. Algoritmista on esitetty kaksi versiota, mutta molemmat versiot toimivat samalla periaatteella. DEC-A* pilkkoo kartan yhtä suuriin osakarttoihin, joiden suuruus voidaan valita. Jokaisessa osakartassa suoritetaan yhtä aikaa A*-algoritmi. Algoritmi pyrkii siirtymään toiseen osakarttaan ja poistaa lyhyemmät reitit etsinnästä. Toisen version ero on se, että klusteroivien algoritmien tapaan DEC-A* esiprosessoi reittiä. DEC-A* ei kuitenkaan tee klusterointia etukäteen, vaan laskee jokaisen osakartan lähtösolmun naapureista parhaimmat solmut. [21]

DEC-A* toimii hyvin suurissa kartoissa, mutta ei pienissä. Algoritmi myös löytää optimaalisen reitin aivan kuten A*:kin. M. Falou et al. olivat tutkineet algoritmia vain ruudukossa, jossa voidaan liikkua neljään suuntaan, joten algoritmin toiminta muissa tapauksissa ei ole selvää. [21]

Hiipivä reitinhakualgoritmi

M. Mendonça et al. kehittivät algoritmin, jonka ainoa tavoite ei ole etsiä lyhintä reittiä. Hiipivä reitinhaku (engl. stealth-based pathfinding) yrittää pysyä piilossa vastustajilta. Algoritmi toimii suurentamalla solmujen hintoja, jotka ovat lähellä vastustajan näkökenttää tai jotka poistuvat suojaiselta alueelta. Algoritmi voi liikkua myös eri vauhtia, jos sen lähellä ei ole kuulolla olevia vastustajia. Algoritmi toimii navigointiverkoissa käyttäen A*-algoritmia reitinhakuun. [22]

Hiipivä reitinhakualgoritmi on sopiva peleihin, joissa hiipiminen on isossa roolissa. Algoritmi toimii hyvin yhdellä vastustajalla, mutta vastustajia lisätessä sen onnistumisprosentti vähenee nopeasti. Algoritmi toimii reaaliajassa, joten suuretkaan kartat eivät ole ongelma hiipivälle reitinhakualgoritmille. [22]



(a) D*-algoritmin nykyinen reitti.

(b) D*-algoritmin uusi reitti maaston muuttuessa.

Kuva 2.2: D*:n kulkema reitti muuttuvassa maastossa. Kuvat on tehnyt tutkielman kirjoittaja.

2.2 D*- ja siihen perustuvat algoritmit

Stentz et al. kehittivät D*-algoritmin (engl. dynamic A*), jota voidaan käyttää tilanteissa, joissa maasto voi vaihtua dynaamisesti. D* algoritmin ei tarvitse tietää tarkasti, esteiden sijaintia tai edes koko maastoa. Algoritmi toimii lähes samalla tavalla kuin A*, mutta solmuilla on avatun ja suljetun tilan lisäksi uusi (engl. new), korotettu (engl. raise) ja laskettu (engl. lower) tilat. [1] Algoritmi aloittaa reitinhaun uudestaan, kun huomaa vanhan solmun hinnan muuttuvan. Myöhemmin Koenig et al. kehittivät D* Lite -algoritmin, joka on vähintään yhtä hyvä kuin D*. D* Lite ei aloita reitinhakua kokonaan uudestaan, vaan laskee vain muuttuneet solmut uudestaan. [23]

Kuva 2.2 esittää D*-algoritmin toimintaa muuttuvassa maastossa. Vihreät ruudut esittävät nykyistä reittiä ja oranssit ruudut esittävät jo kuljettua reittiä. Maaston muuttuessa algoritmi laskee reitin uudelleen.

D* Lite soveltuu peleihin, jossa kartta voi muuttua dynaamisesti. D* Lite löytää optimaalisen polun ja päivittää sitä tarvittaessa, jos solmujen hinta muuttuu. [23]

MOD* Lite -algoritmi

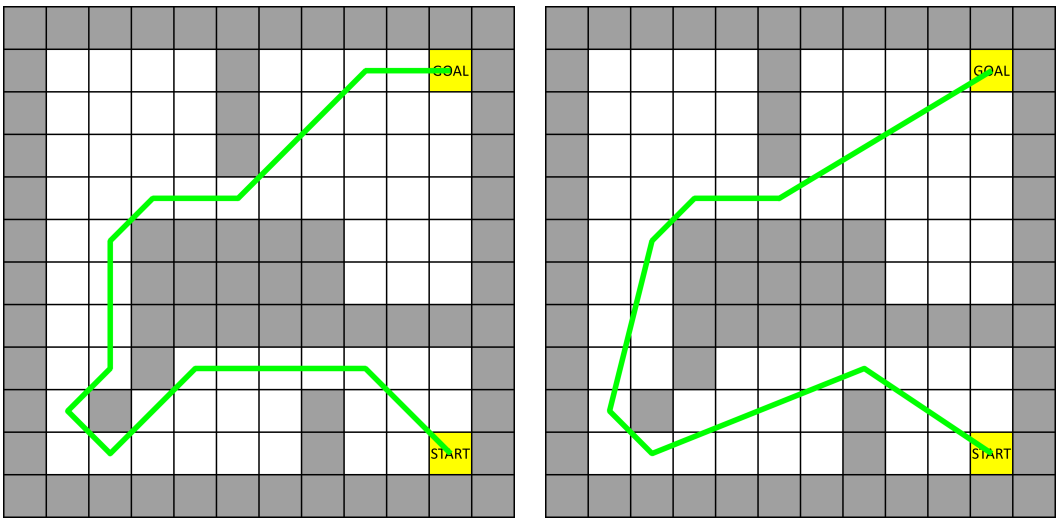
T. Oral et al. kehittivät D* Lite -algoritmin perusteella MOD*-algoritmin (engl. Multi-objective D*). MOD*:n ainoa tavoite ei ole lyhimmän reitin etsiminen. Lyhimmän reitin lisäksi MOD* pyrkii etsimään reitin, joka altistuu uhkille mahdollisimman vähän. Uhka on pelinkehittäjän valitsema alue, joka muuttaa solmujen hintoja halutulla tavalla. Uhka voi olla esimerkiksi NPC-hahmon näkökenttä tai juoksuhiekka. Uhkia voi olla eri tyyppisiä ja eri hintaisia, mutta ne eivät voi olla vuorovaikutuksessa keskenään. [23]

MOD* on hyvä valinta peleihin, joissa kartassa on erilaisia esteitä tai muita alueita, joita NPC-hahmojen halutaan välttää. MOD* löytää optimaalisen reitin tarpeeksi nopeasti. [23]

2.3 Theta*- ja siihen perustuvat algoritmit

A*-algoritmin jälkeen siitä muokattiin D* algoritmi ja myöhemmin Theta*-algoritmi, jotta hahmon ei tarvitse liikkua vain pääilmansuuntien mukaisesti. Theta* on A*-algoritmista muokattu versio, joka ei ole rajoittunut liikkumaan vain viereisten solmujen välillä. Theta* etsii samalla tavalla seuraavan solmun kuin A*, mutta sen jälkeen tarkastaa yksi kerrallaan ovatko vanhemmat solmut uuden solmun näkökentässä. Reitti muutetaan vanhasta solmusta uuteen, jos esteitä ei niiden välillä ole. [24]

Kuvassa 2.3 on esitetty A*:n ja Theta*:n löytämät reitit samassa kartassa. Kuvassa A* voi liikkua 45 asteen kulmassa. Valkoisella merkityt ruudut ovat vapaita ja harmaalla merkityt ovat seiniä. Keltaisella merkityt ruudut ovat alku- ja loppuruutu ja vihreä viiva on algoritmin löytämä reitti. Kuvasta voidaan päätellä, että Theta* on aina nopeampi tai yhtä nopea kuin A*. Theta*:llä reitinhaku kestää kuitenkin kauemmin kuin A*, koska se tekee enemmän laskutoimituksia. Nopeuden ero



(a) A*-algoritmin löytämä reitti. (b) Theta*-algoritmin löytämä reitti

Kuva 2.3: A*:n ja Theta*:n löytämien reittien vertailu samassa kartassa. Kuvat on tehnyt tutkielman kirjoittaja.

suurenee, kun kartan kokoa kasvatetaan. [13]

Theta* on hitaampi kuin A*, mutta se löytää lyhyemmän reitin, koska se ei ole rajoittunut 90 tai 45 asteen kulmiin. Tämän takia se soveltuu hyvin, kun halutaan lyhin mahdollisin reitti. Theta* ei suoraan sovellu navigointiverkkoihin, mutta adaptoitu versio voi toimia hyvin, sillä se ei ole rajoittunut liikkumaan solmujen reunoilla. Theta* ei toimi dynaamisissa kartoissa. [13]

C-Theta*-algoritmi

P. Mendonca et al. ovat ehdottaneet C-Theta*-algoritmia, joka jakaa kartan klustereihin. Klusterit arvioidaan niiden tiheyden perusteella joko matala- tai korkeatiheisiin klustereihin. Korkeatiheisissä klustereissa on enemmän seiniä, joten ne hyötyvät vähemmän Theta*-algoritmin tekemistä näkökenttätesteistä ja sen takia niissä käytetään A*-algoritmia. Theta*-algoritmia taas käytetään matalatiheissä klustereissa, joissa siitä on todennäköisemmin hyötyä. Näiden muutosten avulla C-Theta*-algoritmi on keskiarvolta noin 20 prosenttia nopeampi kuin Theta*, mutta löytää noin prosenttia pidemmän reitin. [13]

C-Theta* soveltuu samoihin käyttötapauksiin kuin Theta*, jos reitin ei tarvitse olla ehdottomasti lyhin. C-Theta* toimii kuitenkin paremmin kuin Theta*, kun kartan kokoa kasvatetaan. Samoin kuin Theta*, C-Theta* toimii vain staattisissa kartoissa, joten dynaamisissa kartoissa täytyy käyttää jotakin muuta algoritmia. [13]

3 Heurististen algoritmien soveltuvuus videopeleihin

Videopelit vaativat paljon erilaisia reitinhakualgoritmeja. Heuristiset reitinhakualgoritmit soveltuvat hyvin videopeleihin, sillä ne käyttävät vähemmän tietokoneen resursseja ja löytävät reitin nopeammin kuin epäinformoidut reitinhakualgoritmit [1]. Algoritmeja on paljon ja niiden nimistä ei aina selviä, mitä varten algoritmi on kehitetty. Tämän takia aikaisemmin esitetyt heuristiset reitinhakualgoritmit jaetaan tässä luvussa viiteen eri kategoriaan. Monilla reitinhakualgoritmeilla on päällekkäisyyksiä useiden kategorioiden kanssa, joten ne esitetään jokaisessa kategoriassa, mihin ne soveltuvat. Kategorioiden sisällä eri reitinhakualgoritmeilla voi myös olla eri käyttökohteita ja niille esitetään videopelejä, jossa niitä on käytetty tai voitaisiin käyttää.

Ensimmäisessä kategoriassa on tiettyä resurssia optimoivat algoritmit. Resurssi voi olla esimerkiksi reitin pituus tai tietokoneen muisti. Toisessa kategoriassa on dynaamisessa maastossa toimivat algoritmit. Nämä algoritmit löytävät reitin, vaikka kartta muuttuisi reitinhaun aikana. Kolmannessa kategoriassa on reaaliaikaiset algoritmit, jotka eivät odota, että reitti on valmis, vaan lähtevät liikkeelle heti. Neljännessä kategoriassa on algoritmit, jotka voivat liikuttaa useaa hahmoa samaan aikaan. Viidennessä kategoriassa on algoritmeja, jotka ovat erikoistuneet johonkin muuhun tarkoitukseen kuin neljä edellistä kategoriaa.

3.1 Eri resurssia optimoivat algoritmit

A* luotiin, koska epäinformoidut reitinhakualgoritmit käyttivät paljon tietokoneen muistia ja olivat usein hitaita. A* on käyttää heuristiikkaa, minkä takia se on hyvä vaihtoehto, jos ei tiedä tarkemmin mitä tarkalleen etsii. A* toimii hyvin, jos kartta ei ole suuri, mutta suurissa kartoissa A*:n reitinhaku kestää kauan.

Klusteroivat algoritmit nopeuttavat reitinhakua suurissa kartoissa. Näitä ovat KM-A*, HKMA, DEC-A* ja C-Theta*. HKMA on KM-A*:sta parannettu versio, joka parantaa nopeutta, reitin pituutta ja resurssien käyttöä. HKMA ei tarvitse yhtä monta syötettä kuin KM-A*, joten sitä on myös helpompi käyttää. HKMA toimii hyvin ainakin 50×50 ruudun kokoisiin karttoihin. [20] DEC-A* pyrkii myös nopeuttamaan reitinhakua suurissa kartoissa. DEC-A* ei kuitenkaan toimi erityisen hyvin pienissä kartoissa, vaan se alkaa olemaan nopea vasta yli 100×100 kokoisissa kartoissa. [21]

C-Theta* toimii hyvin suurissa kartoissa, mutta se pyrkii optimoimaan myös reitin pituuden. Se toimii ainakin 100×100-kokoisissa kartoissa hyvin. Algoritmit ei välttämättä käytä aina Theta*-algoritmia reitinhakuun, vaan saattaa käyttää A*:eä joissakin tilanteissa vähentääkseen reitinhaun suoritukseen kestävää aikaa. Theta* on taas erinomainen valinta, jos halutaan optimoida reitin pituus täysin. Theta* vie jopa tuplasti enemmän aikaa kuin A* 100×100-ruudukossa. [13]

Kuvassa 3.1 on kuvankaappaus tornipuolustuspelistä (engl. tower defense) Emberward, jossa vastustajat yrittävät päästä keskellä olevaan nuotioon ja pelaaja yrittää estää niitä. Pelaaja voi asettaa seiniä hidastaakseen vastustajia ja vastustajat puolestaan yrittävät löytää nopeimman reitin, mihin voitaisiin käyttää Theta* tai C-Theta*-algoritmeja. Emberward-pelissä reitti on rajoitettu 45 asteen kulmaan, joten siinä ei luultavasti ole käytetty Theta*-algoritmia.

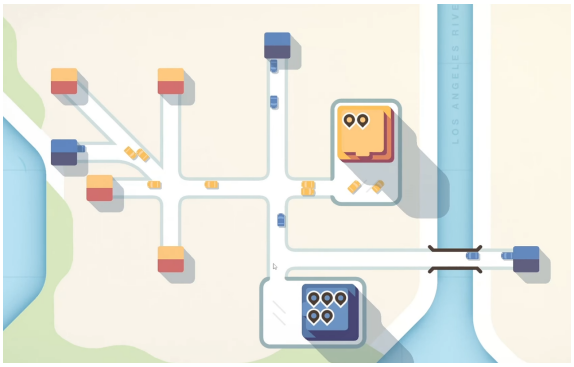
Tietokoneen muistia optimoivat algoritmit ovat usein myös nopeampia, koska niiden ei tarvitse tehdä yhtä paljon laskutoimituksia. CentA*-algoritmi vähentää

Kuva 3.1: Kuvankaappaus Emberward-pelistä, johon on merkitty vastustajien kulkema reitti punaisella. Vastustajat tulevat monista suunnista kohti kuvan keskiötä.

muistin käyttöä ja reitin nopeutta, mutta suuntaan perustuva heuristiikka vaikuttaa tekevän sen paremmin, sillä se ei avaa yhtä paljon solmuja, kuin etäisyyteen perustuva heuristiikka. Suuntaan perustuva heuristiikka ei kuitenkaan sovellu navigointiverkkoon, sillä se vaatii tiedon suunnasta. [15] Reaaliaikaiset reitinhakualgoritmit vähentävät tietokoneen muistin käyttöä, mistä kerrotaan tarkemmin kohdassa 3.3.

3.2 Dynaamisessa maastossa toimivat algoritmit

Monissa videopeleissä staattiset reitinhakualgoritmit eivät toimi. Usein kartassa voi tapahtua muutoksia, jotka täytyy ottaa huomioon reitinhaun aikana. D* luotiin tämän takia ja se yksinkertaisesti vain aloittaa reitinhaun uudelleen, kun reitti muuttuu. Myöhemmin D* Lite -algoritmi paransi tätä muuttamalla vain osaa reitistä. Dynaamiset reitinhakualgoritmit soveltuvat hyvin tilanteisiin, joissa halutaan lyhin mahdollinen reitti dynaamisessa maastossa. Dynaamiset reitinhakualgoritmit eivät ole suosittuja, koska reaaliaikaiset reitinhakualgoritmit ovat usein nopeampia



Kuva 3.2: Kuvankaappaus Mini Motorways -pelistä, jossa autojen täytyy dynaamisesti löytää lyhin reitti kodin ja työpaikan välillä. Pelaaja voi lisätä ja poistaa teitä milloin vain.

ja käyttävät vähemmän muistia. Dynaamisilla reitinhakualgoritmeilla on kuitenkin yksi ominaisuus, minkä perusteella ne voivat olla hyvä valinta. Ne eivät jää jumiin helposti eli hankaa samojen solmujen välillä, sillä ne laskevat koko reitin alusta loppuun. Hankaaminen on usein ongelma reaaliaikaisille reitinhakualgoritmeille. [18]

Dynaamisia reitinhakualgoritmeja ovat D* Lite, MOD* ja MA-HEHA*. D* Lite soveltuu peleihin, jossa kartta voi muuttua hieman. Esimerkiksi liikkuvat alustat ja seinät toimivat hyvin dynaamisten reitinhakualgoritmien kanssa. MOD*-algoritmi voi muuttaa solmujen hintaa dynaamisesti, joten se soveltuu hyvin esimerkiksi peliin, jossa NPC-hahmo liikkuu eri vauhtia eri maastossa. MA-HEHA* soveltuu puolestaan dynaamisiin karttoihin, jossa käytetään useaa NPC-hahmoa.

Kuvassa 3.2 on kuvankaappaus Mini Motorways -strategiapelistä. Pelissä pelaaja rakentaa teitä eri väristen autojen kotien ja yhtiöiden välillä. Teillä voi olla useita risteyksiä ja pelaaja voi dynaamisesti muuttaa teitä. Autojen täytyy siis dynaamisesti laskea nopein mahdollinen reitti. Autoja on paljon, joten tähän voitaisiin käyttää MA-HEHA*-algoritmia.

3.3 Reaaliaikaiset algoritmit

Reaaliaikaiset reitinhakualgoritmit ovat samankaltaisia dynaamisten algoritmien kanssa siten, että molemmat toimivat dynaamisessa ympäristössä. Reaaliaikaisille algoritmeille reitin pituus ei ole kuitenkaan tärkein vaan se, että hahmo lähtee liikkelle heti reaaliajassa. Reaaliaikaiset reitinhakualgoritmit pysäyttävät reitinhaun tietyin väliajoin ja lähtevät liikkeelle parhaimpaa löydettyä solmua kohti. Nämä algoritmit eivät avaa montaa solmua, koska ne eivät hae koko reittiä yhdellä kertaa, vaan tekee monta lyhyttä hakua. Tällöin ei tarvitse käyttää paljon tietokoneen muistia ylläpitämään solmujen tietoja, minkä takia reaaliaikaiset algoritmit käyttävät vähän tietokoneen resursseja.

Tutkielmassa olevista reaaliaikaisista reitinhakualgoritmeista nopein ja resurssitehokkain on luultavasti kNN LRTA*. Sitä on suoraan vertailtu TBA*:een ja kNN LRTA* oli parempi jokaisessa metriikassa yhtä lukuun ottamatta. TBA* on nopeampi silloin, kun kNN LRTA*:n tietokannassa on paljon välietappeja tallennettuna. LSS-LRTA* on myös mahdollisesti sopiva samoissa tilanteissa, mutta ei ole selvää kumpi näistä on parempi.

Usean hahmon liikuttamiseen reaaliaikaisista algoritmeista soveltuu BMAA* ja kNN LRTA*, mutta näitä vertaillaan tarkemmin kohdassa 3.4.

Kuvassa 3.3 on kuvankaappaus MMORPG-pelistä (engl. massively multiplayer online role-playing game) Old School RuneScape. Pelissä pelaajat ohjaavat omaa NPC-hahmoa, jonka täytyy löytää reitti pelaajan haluamaan kohtaan. Tähän voitaisiin käyttää kNN LRTA* -algoritmia, koska kaikki pelaajat voivat käyttää samaa tietokantaa, mikä nopeuttaa reitinhakua huomattavasti. [18]

Kuva 3.3: Kuvankaappaus Old School RuneScape -pelistä, jossa näkyy monta pelaajien hallitsemaa NPC-hahmoa.

3.4 Useaa hahmoa liikuttavat algoritmit

Monissa videopeleissä on paljon NPC-hahmoja, joiden täytyy liikkua samaan aikaan. Monet heuristiset reitinhakualgoritmit eivät osaa ottaa toisten hahmojen liikettä huomioon ja sen takia on kehitetty algoritmeja, jotka ohjaavat useaa hahmoa samaan aikaan. Dynaamiset ja reaaliaikaiset algoritmit saattavat toimia, mutta luultavasti vievät enemmän aikaa ja resursseja, kuin usean hahmon liikuttamiseen erikoistuneet reitinhakualgoritmit.

MA-HEHA* on dynaaminen reitinhakualgoritmi, joka on erikoistunut monen hahmon liikuttamiseen. BMAA* on kuitenkin paljon nopeampi, mutta BMAA*:llä on ongelmia, jotka D. Sigurdson et al. ovat maininneet. BMAA* saattaa liikuttaa hahmoja umpikujaan ja hahmo voi jäädä jumiin myös, jos yhden hahmon leveällä käytävällä on toinen hahmo maalissaan [12]. Molemmat algoritmit pystyvät työntämään toisia hahmoja, jos pelinkehittäjä haluaa, mutta BMAA* ei voi työntää jo maalissa olevia hahmoja. MA-HEHA*:stä ei ole mainittu selkeitä ongelmia, mutta on selvästi hitaampi, minkä takia BMAA* on luultavasti parempi vaihtoehto. [12], [19] Kolmas algoritmi, joka ei suoraan sovellu usean hahmon liikuttamiseen, mutta on siihen erittäin hyvä, on kNN LRTA*. Sen käyttämä tietokanta voi auttaa muistin käytön vähentämisessä, sillä kaikki hahmot voivat etsiä tietokannasta sopivan välie-

Kuva 3.4: Kuvankaappaus League of Legends -pelin pienoiskartasta, jossa on ympyröitynä monta NPC-hahmoa, jotka liikkuvat yhtä aikaa.

tapin. Algoritmit on nopea ja löytää lyhyet reitit, mutta sen esiprosessointivaihe kestää kauan. Käyttämällä useaa hahmoa, kNN LRTA*:n heikkous lievenee. [18]

Kuvassa 3.4 on kuvankaappaus MOBA-pelin (engl. Multiplayer Online Battle Arena) League of Legends pienoiskartasta. Pienoiskartassa on punaisella ympyröity monien NPC-hahmojen rykelmät, jotka kaikki liikkuvat yhtä aikaa. MOBA-peleissä onkin usein paljon NPC-hahmoja, joihin useaa hahmoa liikuttavat reitinhakualgoritmit soveltuvat hyvin.

D. Churchill kehitti robotin, joka pelaa RTS-peliä (engl. Real-Time Strategy). Robotin täytyy ohjata useaa hahmoa yhtä aikaa reaaliajassa, minkä takia D. Churchill käytti reitinhakuun useaa hahmoa liikuttavaa algoritmia, joka rekursiivisesti abstraktoi karttaa pienempiin osiin. [4]

Kuva 3.5: Kuvankaappaus The Elderscrolls V: Skyrim -pelistä, jossa on ympyröitynä NPC-hahmo hiipimässä kohti vastustajaa.

3.5 Muihin käyttötarkoituksiin soveltuvia algoritmeja

On olemassa reitinhakualgoritmeja, jotka pyrkivät johonkin muuhun tavoitteeseen kuin edellä mainitut kategoriat. Jotkin heuristiset reitinhakualgoritmit on kehitetty käyttämään muuta heuristiikkaa kuin vain etäisyyttä maalista. Tällaisilla algoritmeilla on usea tavoite, eikä vain lyhimmän reitin löytäminen nopeasti.

MOD*-algoritmi on usean tavoitteen algoritmi, joka voi dynaamisesti muuttaa solmujen hintaa. Se soveltuu hyvin peleihin, jossa maasto muuttuu ja pelinkehittäjä haluaa NPC-hahmojen välttävän eri alueita. MOD*-algoritmi toimii myös hiipimiseen, mikä on olennainen osa The Elder Scrolls V: Skyrim -peliä. Hiipiminen tarkoittaa vastustajilta piilossa pysyttelyä, mikä voidaan ohjelmoida antamalla vastustajien näkökentällä oleville solmuille suuremmat hinnat. Algoritmi kasvattaa vastustajien näkökentän lähellä olevien solmujen hintoja, minkä avulla NPC-hahmot eivät liiku tälle alueelle, ellei muita vaihtoehtoja ole.

Kuvassa 3.5 on kuvankaappaus The Elder Scrolls V: Skyrim -pelistä. Pelissä voi hankkia seuraajia, jotka osaavat hiipiä tarvittaessa. Kuvassa on ympyröity punaisella seuraaja, joka hiipii kohti vastustajaa. MOD*:n ja hiipivän reitinhakualgoritmin kaltaiset algoritmit soveltuisivat hyvin tällaiseen peliin.

4 Pohdinta

Tässä tutkielmassa esiteltiin ja tutkittiin eri heuristisia reitinhakualgoritmeja. Heuristiset reitinhakualgoritmit eroavat toisistaan osittain paljonkin. Tutkielmalla on teoreettisia kontribuutioita ja käytännön implikaatioita pelinkehittäjille, pelaajille sekä reitinhakualgoritmien kehittäjille.

4.1 Teoreettiset Kontribuutiot ja käytännön implikaatiot

Kirjallisuudessa esiintyy paljon tutkimuksia yksittäisistä reitinhakualgoritmeista. Eri reitinhakualgoritmeja vertailevia tutkimuksia löytyy myös paljon, mutta kaikki tässä työssä löytyneistä aineistoista vertailivat epäinformoituja reitinhakualgoritmeja heuristisiten reitinhakualgoritmien kanssa. Useissa aineistoissa A* oli ainoa heuristinen reitinhakualgoritmi, jota tutkittiin.

Tämä tutkielma antaa hyvän yleiskatsauksen eri heuristisiten reitinhakualgoritmien toiminnasta. Lisäksi tutkielma antaa usean teoreettisen luokittelun heuristisille reitinhakualgoritmeille. Luvussa 2 heuristiset reitinhakualgoritmit lajitellaan A*, D*, ja Theta* -perusteisiin algoritmeihin sen perusteella, minkä reitinhakualgoritmin pohjalta algoritmi on kehitetty. Monet algoritmit kuitenkin häivyttävät näiden luokkien rajoja, minkä takia suurin tutkielman kontribuutio on luvussa 3 tehty luokittelu. Heuristiset reitinhakualgoritmit ovat luokiteltu helposti ymmärret-

täviin kategorioihin, jotka ovat hyödyllisiä sopivan reitinhakualgoritmin etsimisessä videopeleihin.

Tämä tutkielma on tarkoitettu ensisijaisesti pelinkehittäjille, jotta he voivat valita sopivan reitinhakualgoritmin videopeliinsä. Tutkielmalla on kuitenkin käytännön implikaatioita myös videopelien pelaajille sekä reitinhakualgoritmien kehittäjille.

Pelien kehittäjät voivat helpommin ja nopeammin löytää sopivan algoritmin, joka toimii paremmin kuin tunnetut algoritmit, kuten A*. Koska heuristisia reitinhakualgoritmeja on paljon ja niitä on kehitetty eri käyttötarkoituksiin, on tämän työn perusteella helppo valita tarpeeseen sopiva algoritmi.

Pelien pelaajat taas hyötyvät, koska sopivammat reitinhakualgoritmit ovat nopeampia ja kuluttavat vähemmän resursseja. Tämä parantaa pelin suorituskykyä, mikä mahdollistaa mukavamman pelikokemuksen ja korkeammat grafiikat huonommallakin laitteella. NPC-hahmot voivat myös löytää lyhyempiä ja loogisempia reitejä, mikä voi vähentää pelaajien turhautumista.

Reitinhakualgoritmien kehittäjät puolestaan voivat valita algoritmin, jota he voivat lähteä tutkimaan ja kehittämään sekä ymmärtää minkälaisia reitinhakualgoritmeja videopeleissä tarvitaan.

4.2 Tutkielman rajoitukset ja jatkotutkimus

Tutkielmassa keskityttiin heuristisiin reitinhakualgoritmeihin. Epäinformoidut ja metaheuristiset reitinhakualgoritmit mainittiin, mutta niitä ei tutkittu millään tavalla. Heuristiset reitinhakualgoritmit ovat lähes aina parempia kuin epäinformoidut, mutta metaheuristiset reitinhakualgoritmit voivat soveltua videopeleihin hyvin. Algoritmeja kehitetään koko ajan lisää, joten kaikkia ei voitu analysoida tässä tutkielmassa. Lisäksi Google Scholar -tietokannasta valittiin 50 ensimmäistä osumaa, minkä takia yli tuhat aineistoa karsittiin pois. On siis mahdollista, että tutkielmassa ei ole esitetty kakikista nopeimpia tai vähiten resursseja käyttäviä algoritmeja.

Tulevaisuudessa metaheurististen reitinhakualgoritmien vertailu voisi olla hyödyllistä. Vaikka metaheuristiset reitinhakualgoritmit käyttävät enemmän tietokoneen resursseja, voi niillä olla käyttöä videopeleissä. Heuristisia reitinhakualgoritmeja on paljon, joita ei tässä tutkielmassa käsitelty niiden suuresta määrästä johtuen. Uusia ja päivitettyjä heuristisia reitinhakualgoritmeja voitaisiin myös vertailla. Eri algoritmien tarkkuutta, nopeutta ja resurssien käyttöä voidaan myös vertailla, sillä eri aineistot käyttävät eri metriikoita, minkä takia niitä ei voitu vertailla keskenään tässä utkielmassa.

4.3 Yhteenveto ja johtopäätelmät

Tässä kirjallisuuskatsauksessa esiteltiin eri kirjallisuudessa esiintyviä heuristisia reitinhakualgoritmeja videopelikontekstissa. Heuristisen reitinhakualgoritmien toiminta sekä vahvuudet ja heikkoudet kerrottiin, mikäli ne oli mainittu alkuperäisessä aineistossa. Kaikki algoritmit luokiteltiin viiteen eri luokkaan niiden toimintaperiaatteiden mukaan ja luokalle tyypilliset piirteet kerrottiin. Luokista valittiin paras algoritmi eri käyttötarkoituksiin ja esiteltiin jokin videopeli, jossa kyseistä algoritmia voitaisiin käyttää.

Heuristiset reitinhakualgoritmit soveltuvat videopeleihin hyvin johtuen niiden nopeudesta ja resurssitehokkuudesta. Tämän tutkielman avulla videopelien kehittäjät voivat helposti löytää sopivan heuristisen reitinhakualgoritmin pelilleen. Reitinhakualgoritmit on lajiteltu toiminnan perusteella, joka auttaa löytämään omaan peliin soveltuvan algoritmin. Lisäksi esimerkit videopeleistä eri kategorioissa auttavat pelinkehittäjiä varmistumaan, että kyseinen algoritmi soveltuu heidän käyttötarkoituksiinsa. Jos sopivaa algoritmia ei löydy, tutkielma antaa vihiä, minkälaista algoritmia kannattaa lähteä etsimään.

Lähdeluettelo

[1] S. R. Lawande, G. Jasmine, J. Anbarasi ja L. I. Izhar, "A systematic review and analysis of intelligence-based pathfinding algorithms in the field of video games", *Applied Sciences*, vol. 12, nro 11, s. 5499, 2022.

[2] A. N. Sabri, N. H. M. Radzi ja A. A. Samah, "A study on Bee algorithm and A* algorithm for pathfinding in games", teoksessa *2018 IEEE Symposium on Computer Applications & Industrial Electronics (ISCAIE)*, IEEE, 2018, s. 224–229.

[3] X. Cui ja H. Shi, "A*-based pathfinding in modern computer games", *International Journal of Computer Science and Network Security*, vol. 11, nro 1, s. 125–130, 2011.

[4] D. G. Churchill, "Heuristic search techniques for real-time strategy games", 2016.

[5] U. A. S. Iskandar, N. M. Diah, M. Ismail ja A. Abdullah, "Comparing the efficiency of Pathfinding Algorithms for NPCs in platform games", *Journal of Positive School Psychology*, vol. 6, nro 3, s. 8434–8441, 2022.

[6] N. Salem, H. Haneya, H. Balbaid ja M. Asrar, "Exploring the Maze: A Comparative Study of Path Finding Algorithms for PAC-Man Game", teoksessa *2024 21st Learning and Technology Conference (L&T)*, IEEE, 2024, s. 92–97.

[7] IEEE, *IEEE Xplore Digital Library*, `https://ieeexplore.ieee.org/Xplorehelp/overview-of-ieee-xplore/about-ieee-xplore`, 26.11.2024, 2024.

[8] Google, *Google Scholar*, `https://scholar.google.com/intl/fi/scholar/about.html`, 26.11.2024, 2024.

[9] H. Jin, W. Wei ja L. Ziyan, "Multi-agent pathfinding system implemented on XNA", teoksessa *2012 Fourth International Conference on Computational Intelligence and Communication Networks*, IEEE, 2012, s. 651–655.

[10] S. H. Permana, K. Y. Bintoro, B. Arifitama, A. Syahputra et al., "Comparative analysis of pathfinding algorithms a*, dijkstra, and bfs on maze runner game", *IJISTECH (International J. Inf. Syst. Technol*, vol. 1, nro 2, s. 1, 2018.

[11] F. H. Putra, S. M. Nasution ja R. A. Nugrahaeni, "Comparison of A* Algorithm and Time Bounded A Algorithm on Maze Chase Game NPC", teoksessa *2019 International Seminar on Research of Information Technology and Intelligent Systems (ISRITI)*, IEEE, 2019, s. 79–84.

[12] D. Sigurdson, V. Bulitko, W. Yeoh, C. Hernández ja S. Koenig, "Multi-agent pathfinding with real-time heuristic search", teoksessa *2018 IEEE conference on computational intelligence and games (CIG)*, IEEE, 2018, s. 1–8.

[13] P. Mendonca ja S. Goodwin, "C-theta*: Cluster based path-planning on grids", teoksessa *2015 International Conference on Computational Science and Computational Intelligence (CSCI)*, IEEE, 2015, s. 605–608.

[14] X. Cui ja H. Shi, "Direction oriented pathfinding in video games", *International Journal of Artificial Intelligence & Applications*, vol. 2, nro 4, s. 1, 2011.

[15] G. E. Mathew, "Direction based heuristic for pathfinding in video games", *Procedia Computer Science*, vol. 47, s. 262–271, 2015.

[16] J. Akshya, V. Mehra, M. Sundarrajan, P. T. Sri ja R. Sathish, ”Optimizing Real-Time Path Planning for NPC Navigation: Leveraging CentA* Algorithm to Enhance Efficiency and Adaptability”, teoksessa *2023 First International Conference on Advances in Electrical, Electronics and Computational Intelligence (ICAEECI)*, IEEE, 2023, s. 1–6.

[17] E. Burns, S. Kiesel ja W. Ruml, ”Experimental real-time heuristic search results in a video game”, teoksessa *Proceedings of the International Symposium on Combinatorial Search*, vol. 4, 2013, s. 47–54.

[18] V. Bulitko, Y. Björnsson ja R. Lawrence, ”Case-based subgoaling in real-time heuristic search for video game pathfinding”, *Journal of Artificial Intelligence Research*, vol. 39, s. 269–300, 2010.

[19] Y. F. Yiu ja R. Mahapatra, ”Multi-agent pathfinding with hierarchical evolutionary hueristic a”, teoksessa *2020 IEEE Third International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)*, IEEE, 2020, s. 9–16.

[20] C. Chen, Y. Li ja T.-S. Li, ”KM-A pathfinding algorithm based on hierarchical clustering and strengthened DB Index criteria”, teoksessa *2011 International Conference on Machine Learning and Cybernetics*, IEEE, vol. 4, 2011, s. 1571–1576.

[21] M. El Falou, M. Bouzid ja A. I. Mouaddib, ”Dec-a*: A decentralized multiagent pathfinding algorithm”, teoksessa *2012 IEEE 24th International Conference on Tools with Artificial Intelligence*, IEEE, vol. 1, 2012, s. 516–523.

[22] M. R. Mendonça, H. S. Bernardino ja R. F. Neto, ”Stealthy path planning using navigation meshes”, teoksessa *2015 Brazilian Conference on Intelligent Systems (BRACIS)*, IEEE, 2015, s. 31–36.

[23] T. Oral ja F. Polat, "A multi-objective incremental path planning algorithm for mobile agents", teoksessa *2012 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology*, IEEE, vol. 2, 2012, s. 401–408.

[24] S. L. Pardede, F. R. Athallah, Y. N. Huda ja F. D. Zain, "A review of path-finding in game development", *CEPAT] Journal of Computer Engineering: Progress, Application and Technology*, vol. 1, nro 01, s. 47, 2022.