Dijkstran algoritmi

Miska Sainkangas

Ville Rantala

Harjoitustyö

Joulukuu 2016

Tieto- ja viestintätekniikka

Tekniikan ja liikenteen ala

Sisällysluettelo

[1 Johdanto 2](#_Toc468654621)

[2 Teoria 2](#_Toc468654622)

[3 Toteutus 2](#_Toc468654623)

[3.1 Käyttöohjeet 4](#_Toc468654624)

[3.1.1 Editori 4](#_Toc468654625)

[3.1.2 Reitinhaku 4](#_Toc468654626)

[3.2 Koodi 5](#_Toc468654627)

[4 Testaus 7](#_Toc468654628)

[4.1 Skenaario 1: Reikä 8](#_Toc468654629)

[4.2 Skenaario 2: Metsän ohi 10](#_Toc468654630)

[4.3 Skenaario 3: Etelään 12](#_Toc468654631)

[4.4 Skenaario 4: Saarelle 14](#_Toc468654632)

[4.5 Skenaario 5: Järvien ympäri 16](#_Toc468654633)

[4.6 Skenaario 6: Metsän läpi 18](#_Toc468654634)

[4.7 Skenaario 7: Suora reitti esteiden ohi 20](#_Toc468654635)

[4.8 Skenaario 8: Tyhjä tasanko 22](#_Toc468654636)

[4.9 Skenaario 9: Sokkelo 24](#_Toc468654637)

[5 Yhteenveto 26](#_Toc468654638)

[6 Lähteet 26](#_Toc468654639)

# Johdanto

Harjoitustyöllä on seuraavat tavoitteet:

1. Toteuttaa Dijkstran reitinhakualgoritmi ja saada se toimimaan virheettömästi.
2. Luoda graafinen ympäristö, jolla algoritmia voidaan testata ja demonstroida.
3. Optimoida algoritmin toimintaa.
4. Testata algoritmin tehokkuutta useassa eri kontekstissa.

# Teoria

Dijkstran algoritmin alkuperäisenä tarkoituksena on hakea lyhin reitti kohden solmun välillä painotetussa verkossa, mutta sitä voidaan käyttää myös viritetyn puun luomiseen samaisesta verkosta. Algoritmin tehokkuus riippuu pitkälti verkon solmujen ja linkkien määrästä. Alkuperäisen algoritmin tehokkuuden huonoimmaksi tapaukseksi on laskettu O(V2), jossa V on solmujen määrä. Algoritmia voidaan kuitenkin optimoida prioriteettijonon avulla, jolloin huonoin tapaus saadaan kaavalla O(E + V log V). (Dijkstra’s algorithm -- N.d).

Algoritmi toimii karkealla tasolla seuraavasti:

1. Lisätään alkupiste avoimeksi listaksi kutsuttuun listaan.
2. Järjestetään avoin lista polun hinnan mukaan.
3. Valitaan ja poistetaan halvin solmu avoimesta listasta.
4. Merkataan valittu solmu suljetuksi.
5. Jos valittu solmu on sama kuin päätepiste, lopetusehto on saavutettu.
6. Käydään läpi valitun solmun jokainen naapurisolmu.
7. Jos naapuri ei ole suljettu…
   1. Lasketaan polun hinta valitusta solmusta naapuriin lisäämällä naapuriin johtavan linkin painoarvo valitun solmun polun hintaan.
   2. Jos naapuri ei ole avoimella listalla, asetetaan naapurin polun hinnaksi juuri laskettu arvo, merkataan valittu solmu naapurin tulosuunnaksi ja lisätään naapuri avoimelle listalle.
   3. Jos naapurille on jo laskettu polun hinta, verrataan sitä uuteen laskettuun arvoon. Jos uusi arvo on pienempi, päivitetään se naapurille uudeksi polun hinnaksi ja asetetaan valittu solmu uudeksi tulosuunnaksi.
8. Niin kauan kuin avoimella listalla riittää solmuja ja lopetusehtoa ei ole saavutettu, iteroidaan uudestaan kohdasta 4 alkaen.

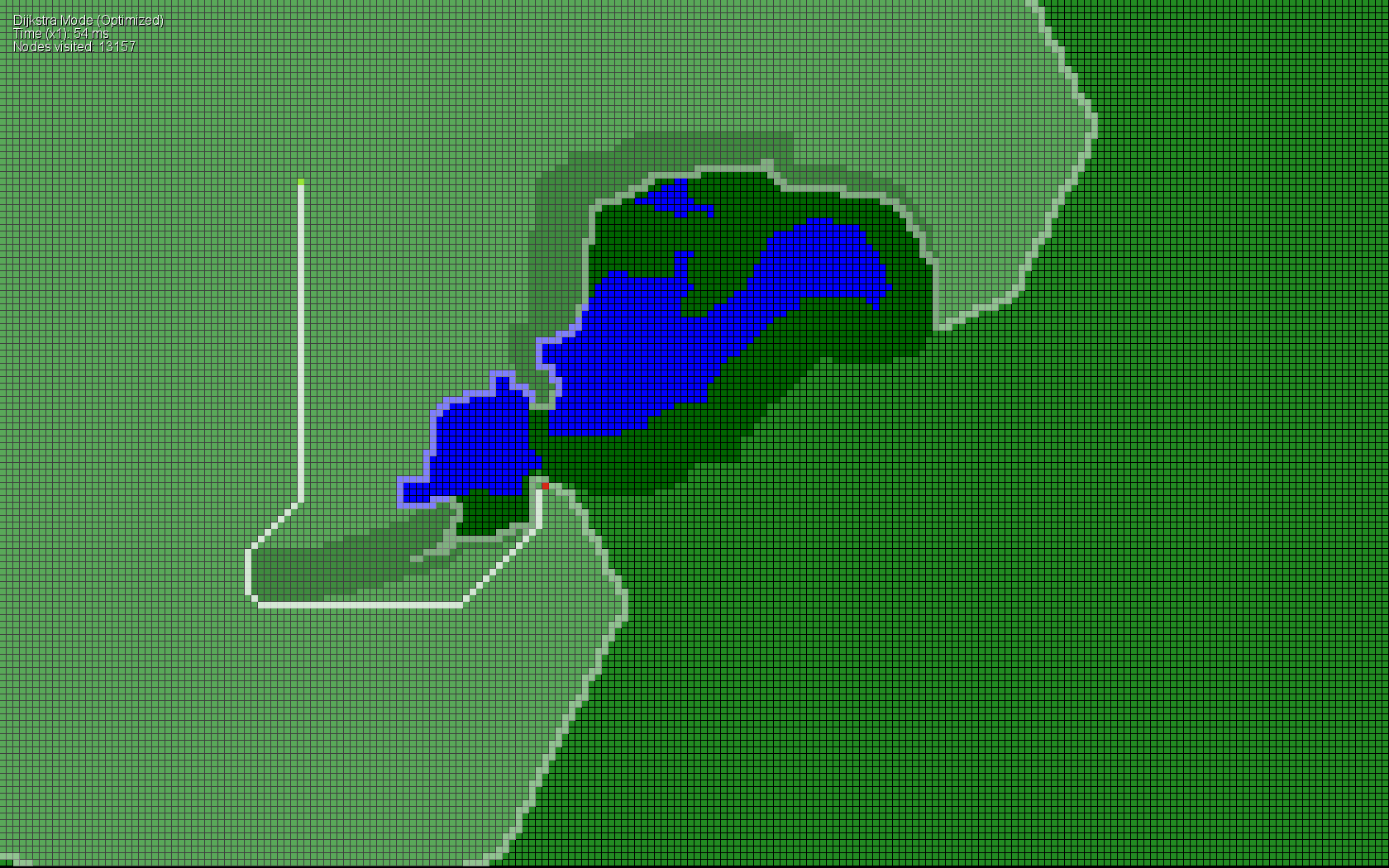
# Toteutus

Ohjelma toteutettiin C# -kielellä. Ohjelmassa on Dijkstran (lähes) alkuperäinen algoritmi ja siitä optimoitu versio, joka käyttää hyödyksi binääripuuta. Puhtaan Dijkstran algoritmin lisäksi toteutettiin sen yleistys, eli A\* algoritmi, vertailun vuoksi. Molempien algoritmien optimointia varten toteutettiin myös tietorakenne, joka tunnetaan nimellä binary heap, eli binäärikeko.

Ohjelmalla on graafinen käyttöliittymä, jolla voidaan valita lähdepiste, kohdepiste ja muuttaa solmujen painoarvoja. Solmujen määrä riippuu näytön koosta. Painoarvoina voi olla joko ruoho (1), vesi (10) tai metsä (1000). Pisteiden ollessa valittuna ohjelma laskee algoritmia käyttäen lyhimmän reitin niiden välillä ja näyttää käyttöliittymässä tuloksen.



Kuva 2 Solmuille annettu arvoiksi ruohoa, vettä ja metsää.



Kuva 3 Vaaleanvihreä piste on lähtöpiste ja punainen kohdepiste. Käydyt solmut näkyvät vaaleana alueena.



Kuva 4 Laskennan nopeus sekä kuinka monessa solmussa on käyty.

## Käyttöohjeet

Ohjelmassa on 5 eri moodia, joiden välillä voi siirtyä käyttämällä nuolinäppäimiä.

### Editori

Editoriin voi siirtyä painamalla numeronäppäintä 1. Editori toimii hieman piirto-ohjelmien tavoin. Hiiren vasemmalla näppäimellä voi maalata kartalle valittua maastotyyppiä. Maastotyypin voi valita pikanäppäimillä Q (vesi) ja W (metsä.) Hiiren oikea näppäin maalaa kartalle ruohoa. Pikanäppäimillä A, S, D, F voi muokata siveltimen kokoa.

### Reitinhaku

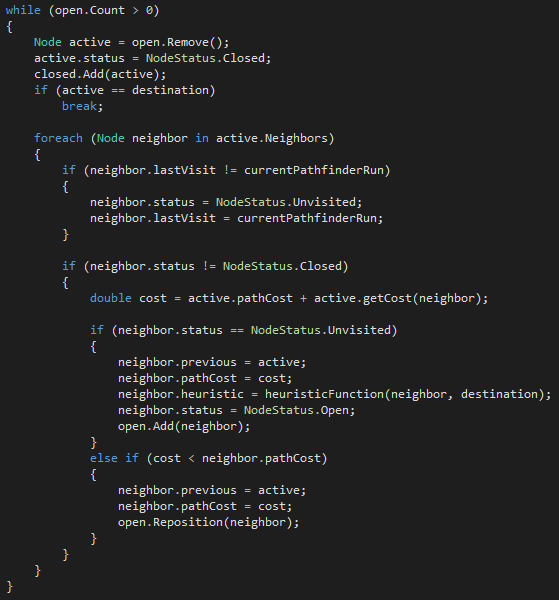
Reitinhaku-moodiin voi siirtyä painamalla numeronäppäimiä 2, 3, 4 tai 5. 2 on Dijkstra-moodi. 3 on optimoitu versio Dijkstra-moodista. 4 on A\*-moodi. 5 puolestaan on A\*:n optimoitu versio.

Näppäimet ovat kaikille reitinhakumoodeille yhteisiä. Hiiren vasemmalla valitaan lähtöpiste, oikealla kohde. Näppäimillä Q, W, E ja R voidaan hidastaa algoritmeja tapauksissa, joissa reitinhaku on liian nopeaa tarkkoja mittauksia varten. Näppäimiä A ja S käytetään piirtotavan valintaan. A piirtää näytölle koko reitin, sekä suljetun ja avoimen listan. S piirtää suljettua listaa askel kerrallaan, jonka avulla voidaan visualisoida algoritmin etenemistä. Z, X, C ja V hidastavat tai nopeuttavat S-moodin etenemisnopeutta.

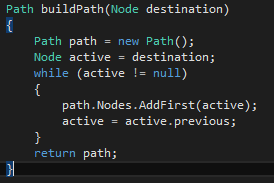
## Koodi

Molemmista algoritmeista on toteutettu 2 versiota, joilla on vain yksi suuri ero. Ensimmäisessä versiossa avoimena listana toimii tavallinen C# List-luokan olio, joka järjestetään uudelleen aina ennen uuden aktiivisen solmun valintaa. Toisessa versiossa käytetään binäärikekoa, joka hoitaa sisäisesti sen, että halvin solmu on aina ensimmäisenä.

Ohjelman koodista suuri osa ohjaa käyttöliittymän toimintaa. Itse algoritmille relevantti koodi löytyy tiedostoista Node.cs ja BinaryHeap.cs.



Kuva 5 Algoritmin toteutus

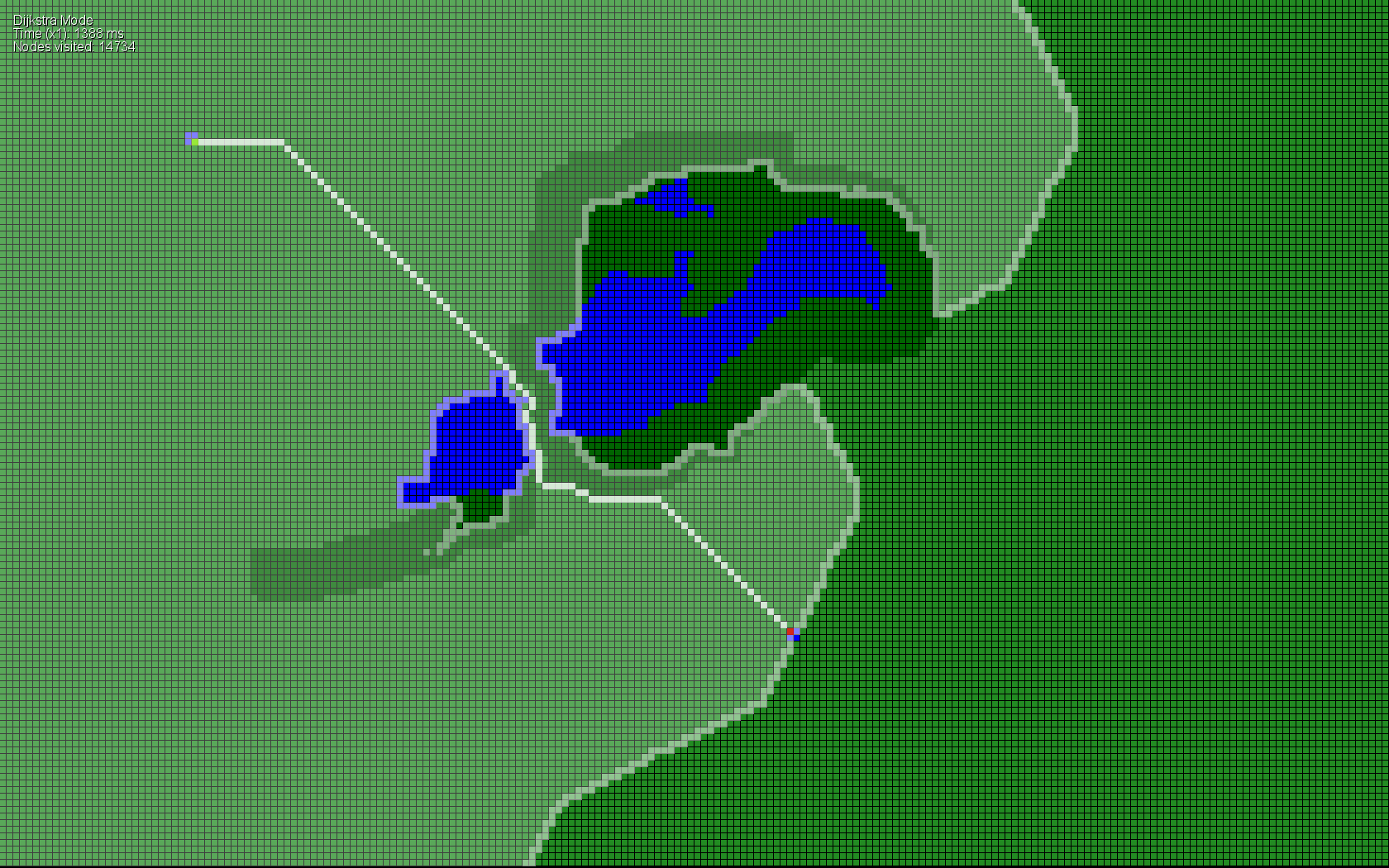


Kuva 6 Funktio, joka rakentaa polun kohteeseen

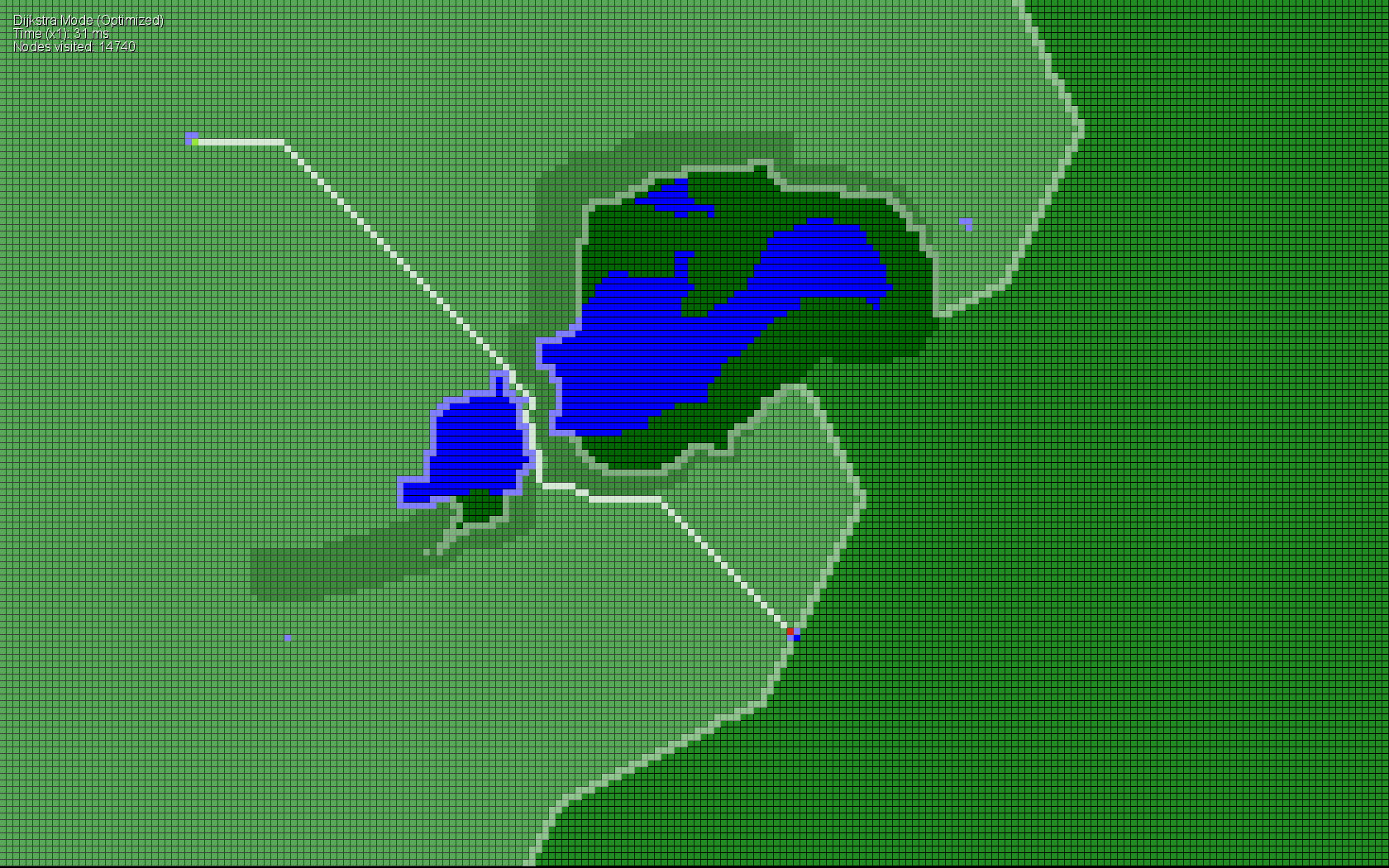
# Testaus

Testaus toteutettiin luomalla manuaalisesti erilaisia skenaarioita ja testaamalla niissä molempien algoritmien molempien versioiden tehokkuutta. Tehokkuuden mittauksessa otettiin huomioon sekä kulunut aika, että läpikäytyjen solmujen määrä.

## Skenaario 1: Reikä



Kuva 7 Dijkstra: 1388 ms, 14734 solmua



Kuva 8 Dijkstra optimoitu: 31 ms, 14740 solmua

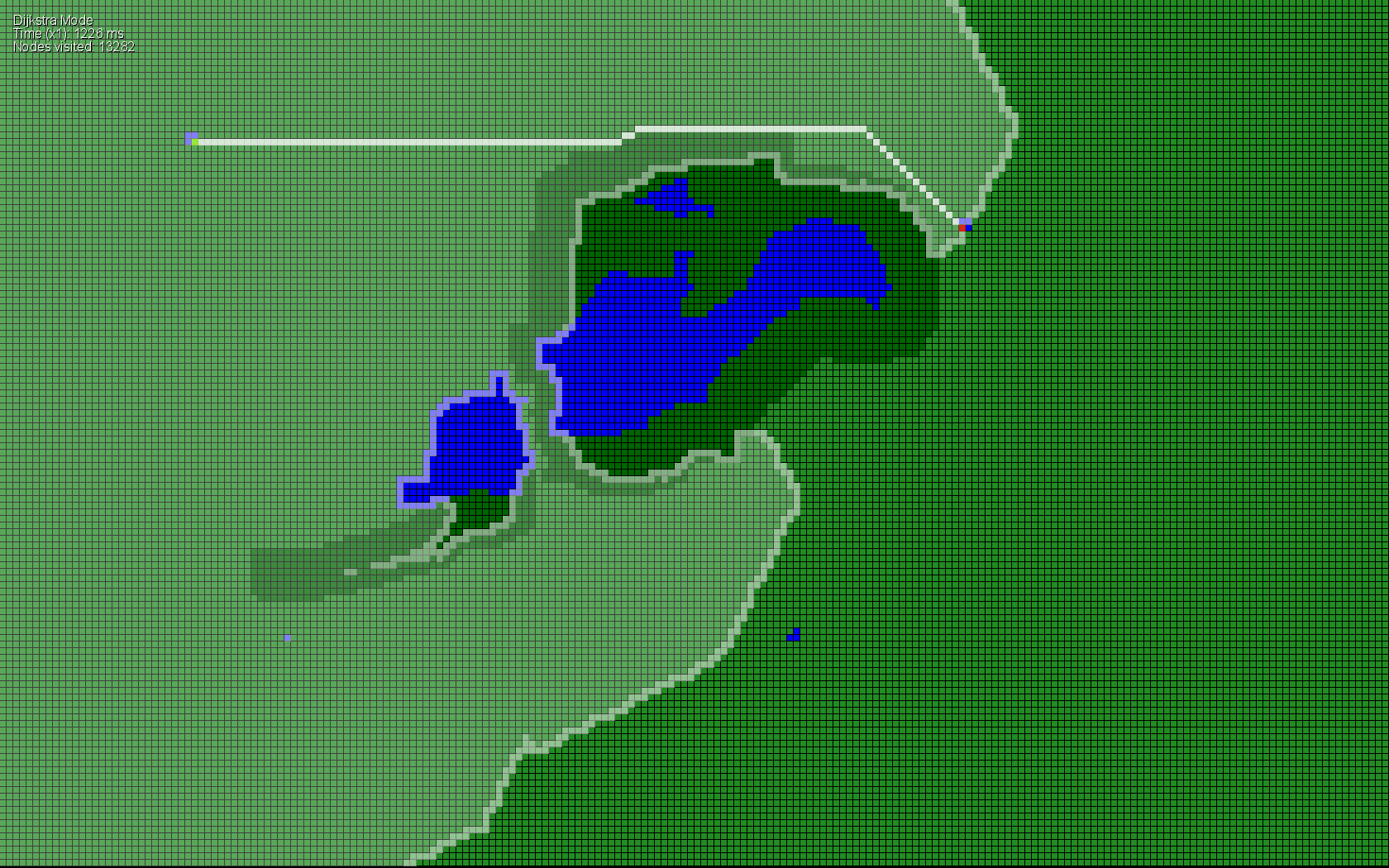


Kuva 9 A\*: 81 ms, 1967 solmua

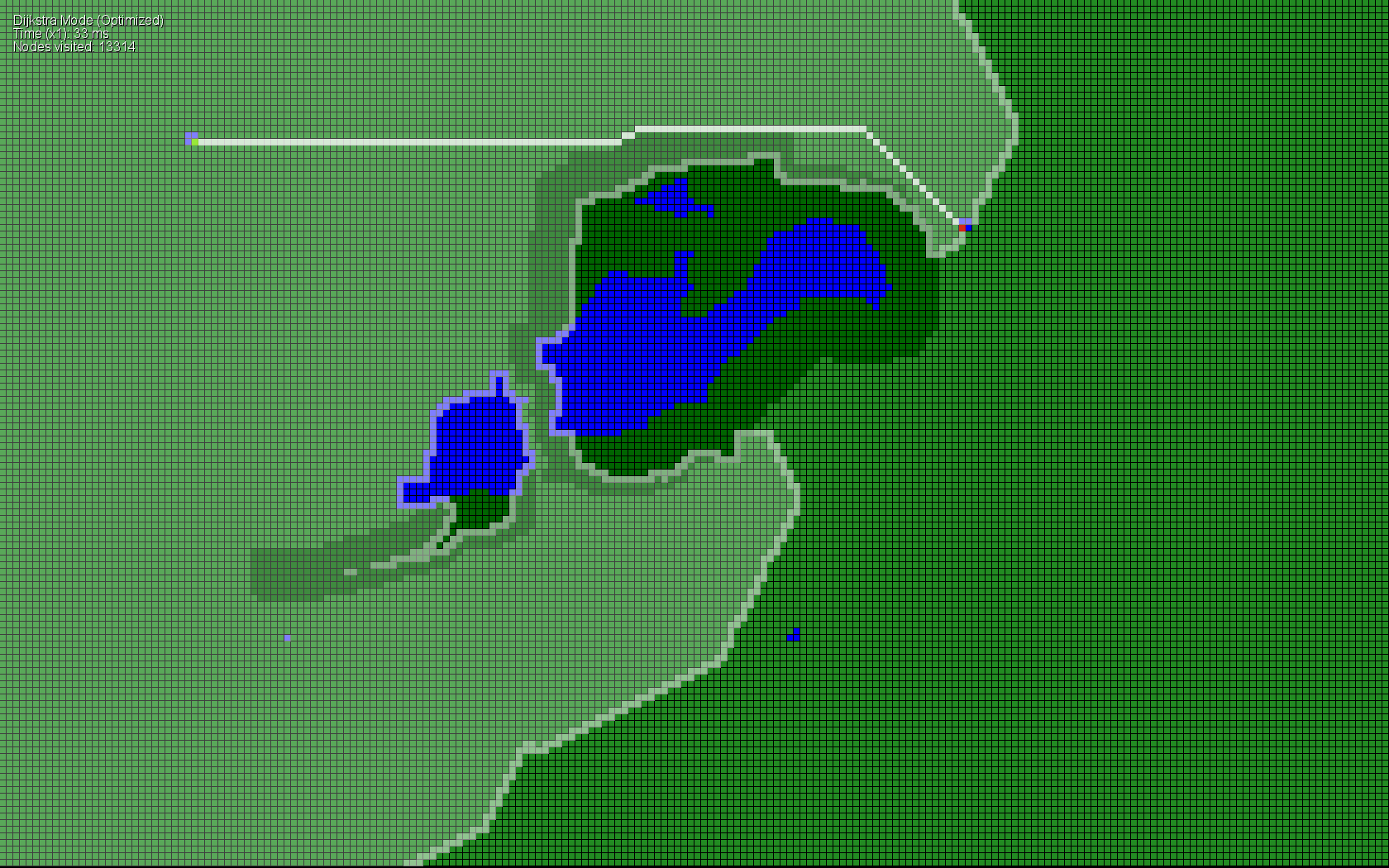


Kuva 10 A\* optimoitu 7 ms, 1638 solmua

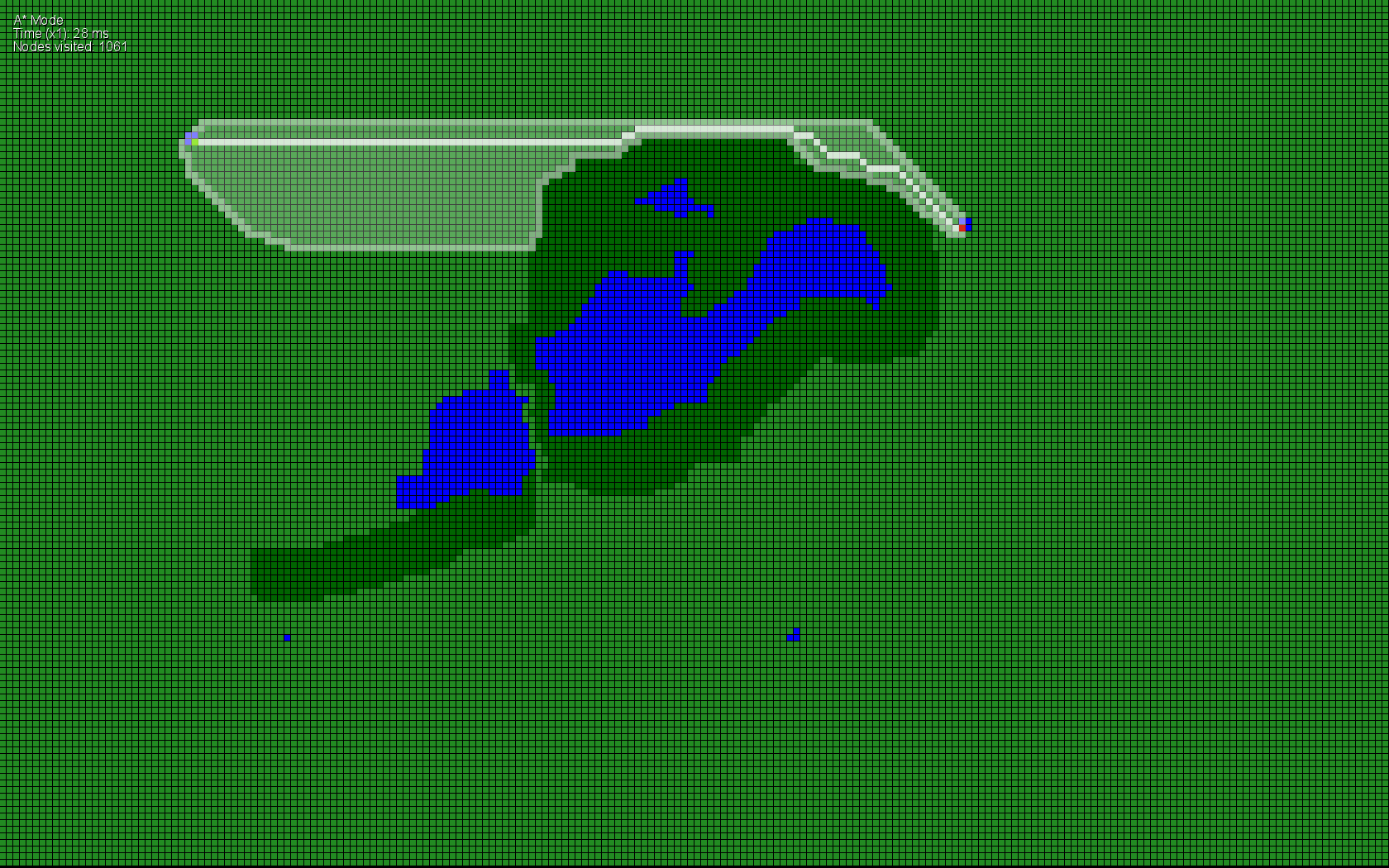
## Skenaario 2: Metsän ohi



Kuva 11 Djikstra: 1226 ms, 13282 solmua



Kuva 12 Djikstra optimoitu: 33 ms, 13314 solmua

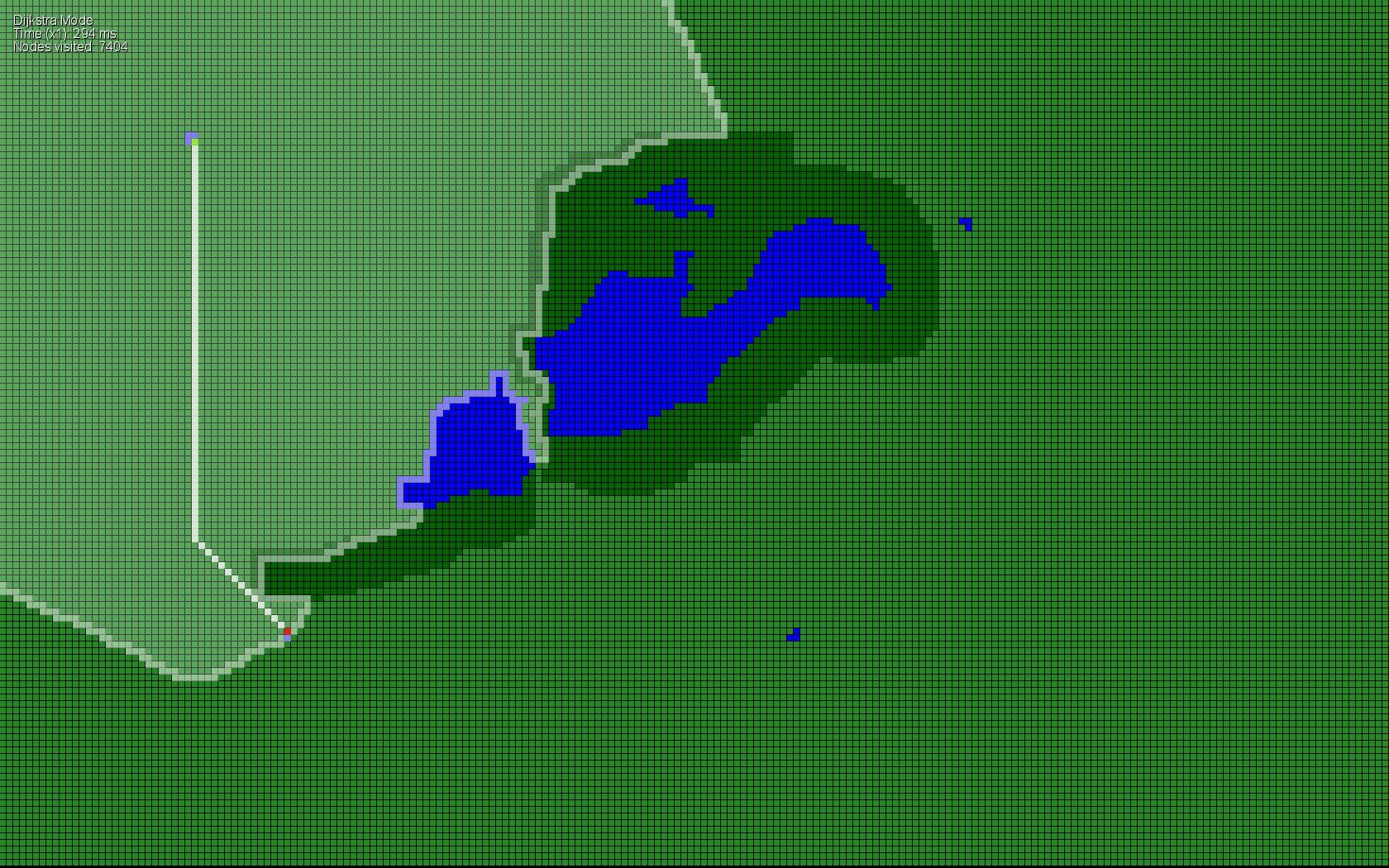


Kuva 13 A\*: 28 ms, 1061 solmua



Kuva 14 A\* optimoitu: 2 ms, 923 solmua

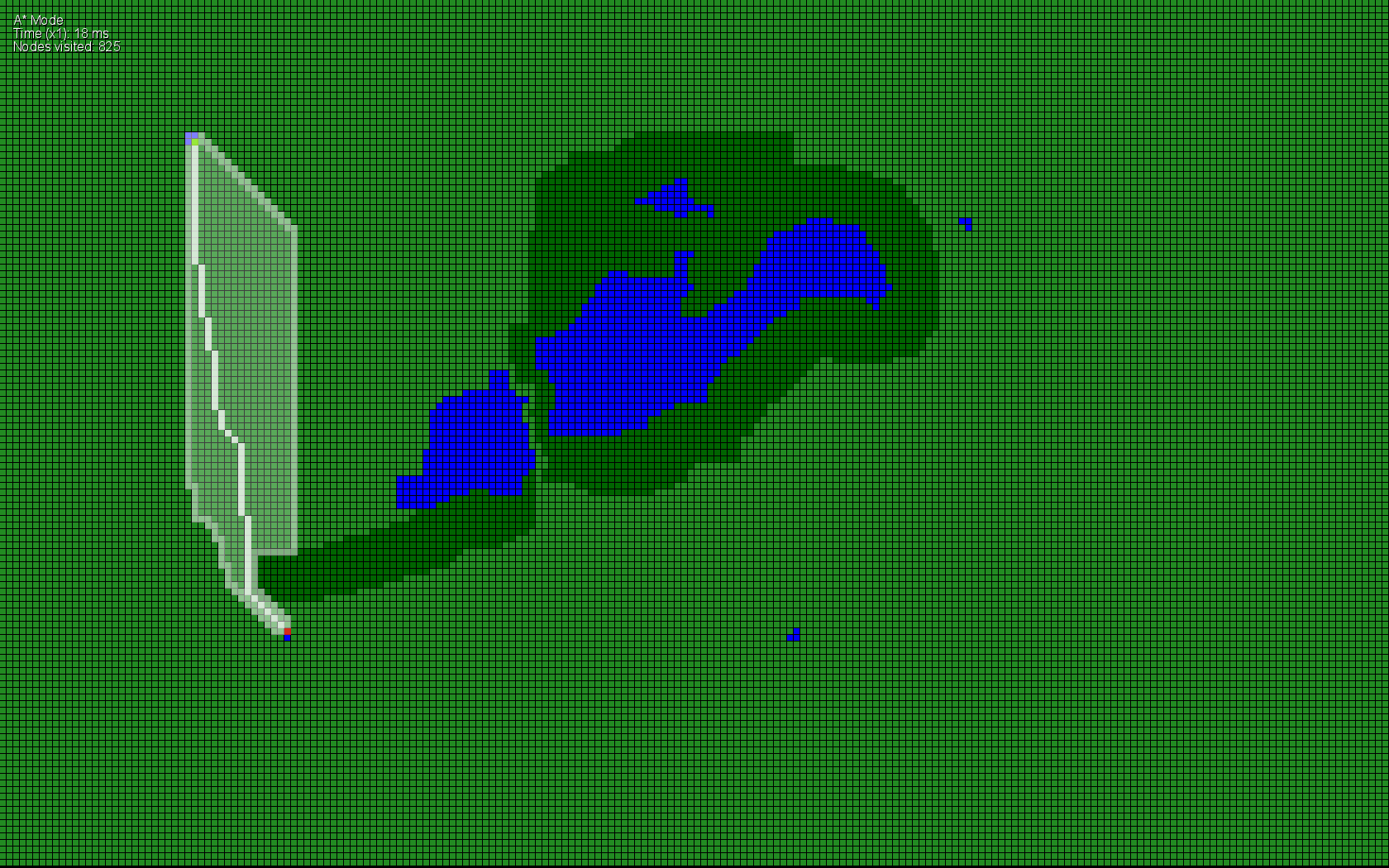
## Skenaario 3: Etelään



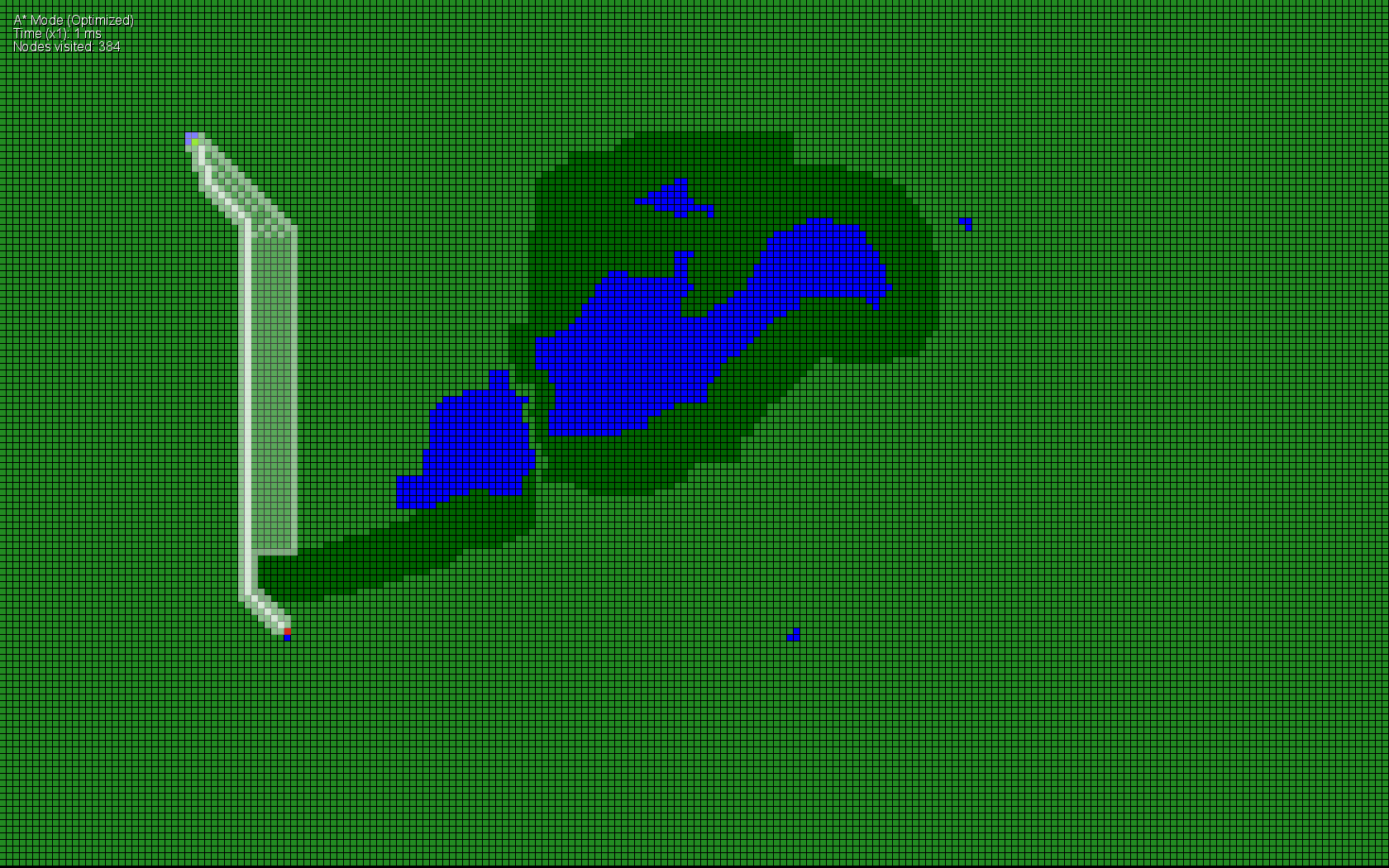
Kuva 15 Dijkstra: 294 ms, 7404 solmua



Kuva 16 Dijkstra optimoitu: 13 ms, 7416 solmua

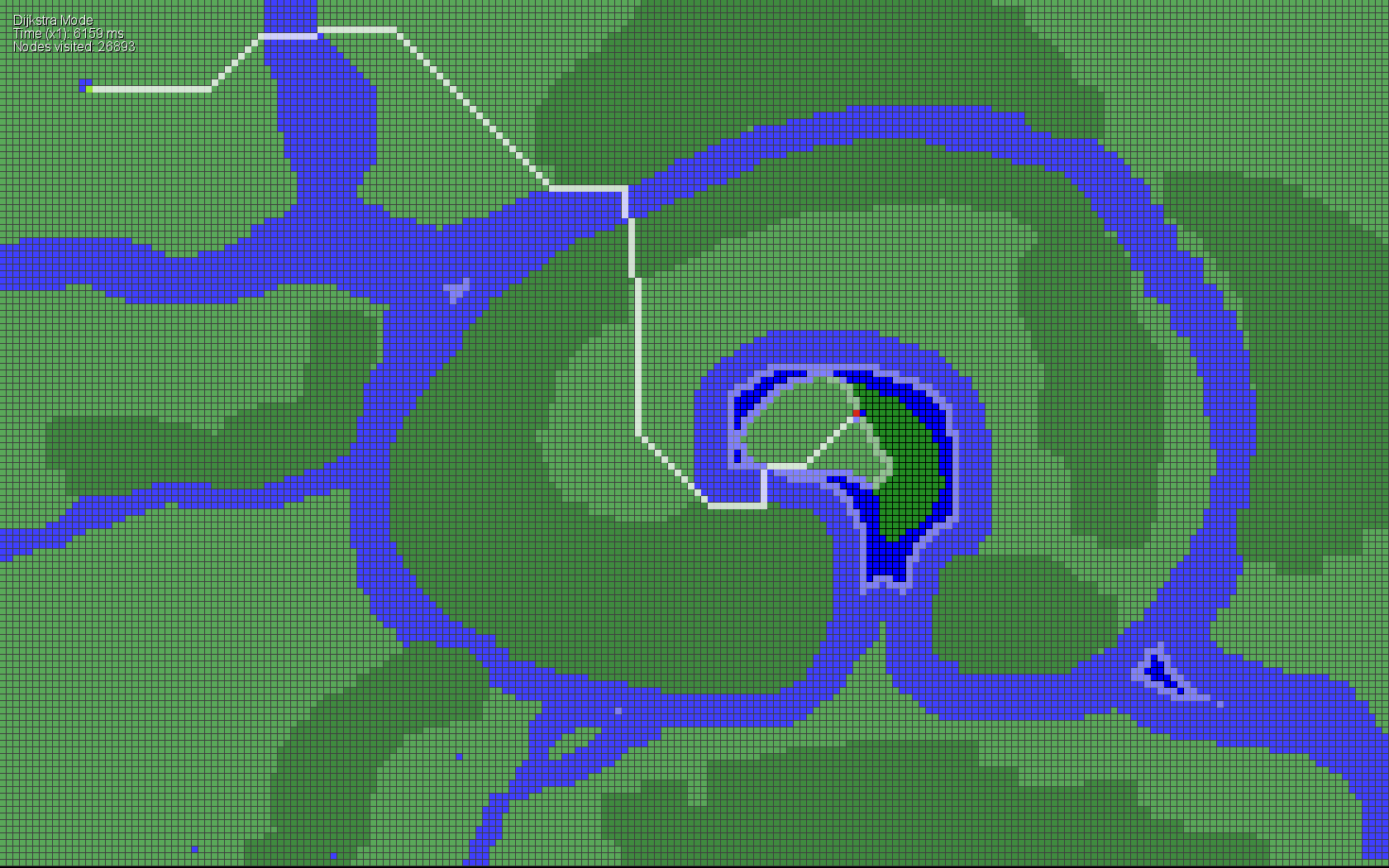


Kuva 17 A\*: 18 ms, 825 solmua

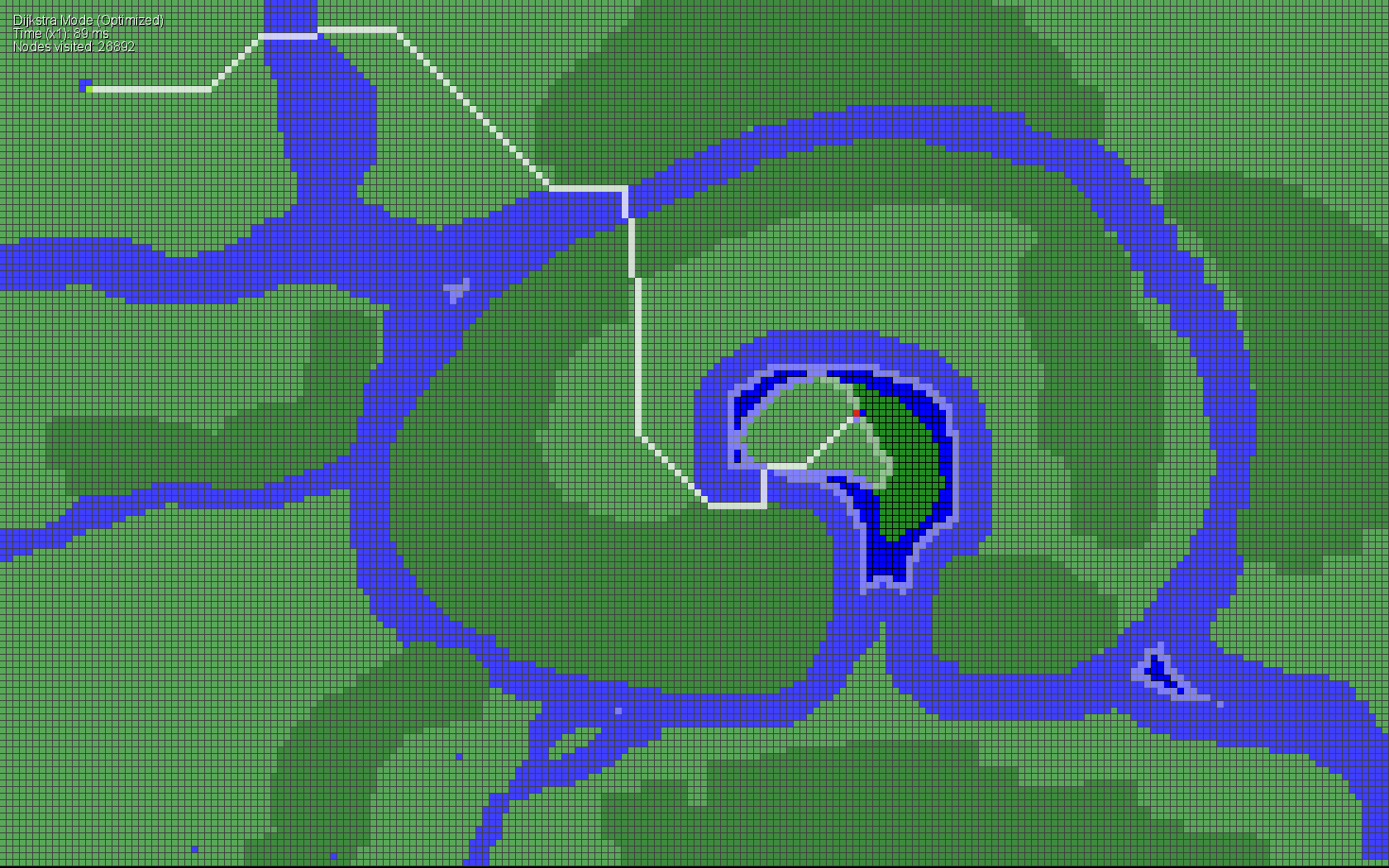


Kuva 18 A\* optimoitu: 1ms, 384 solmua

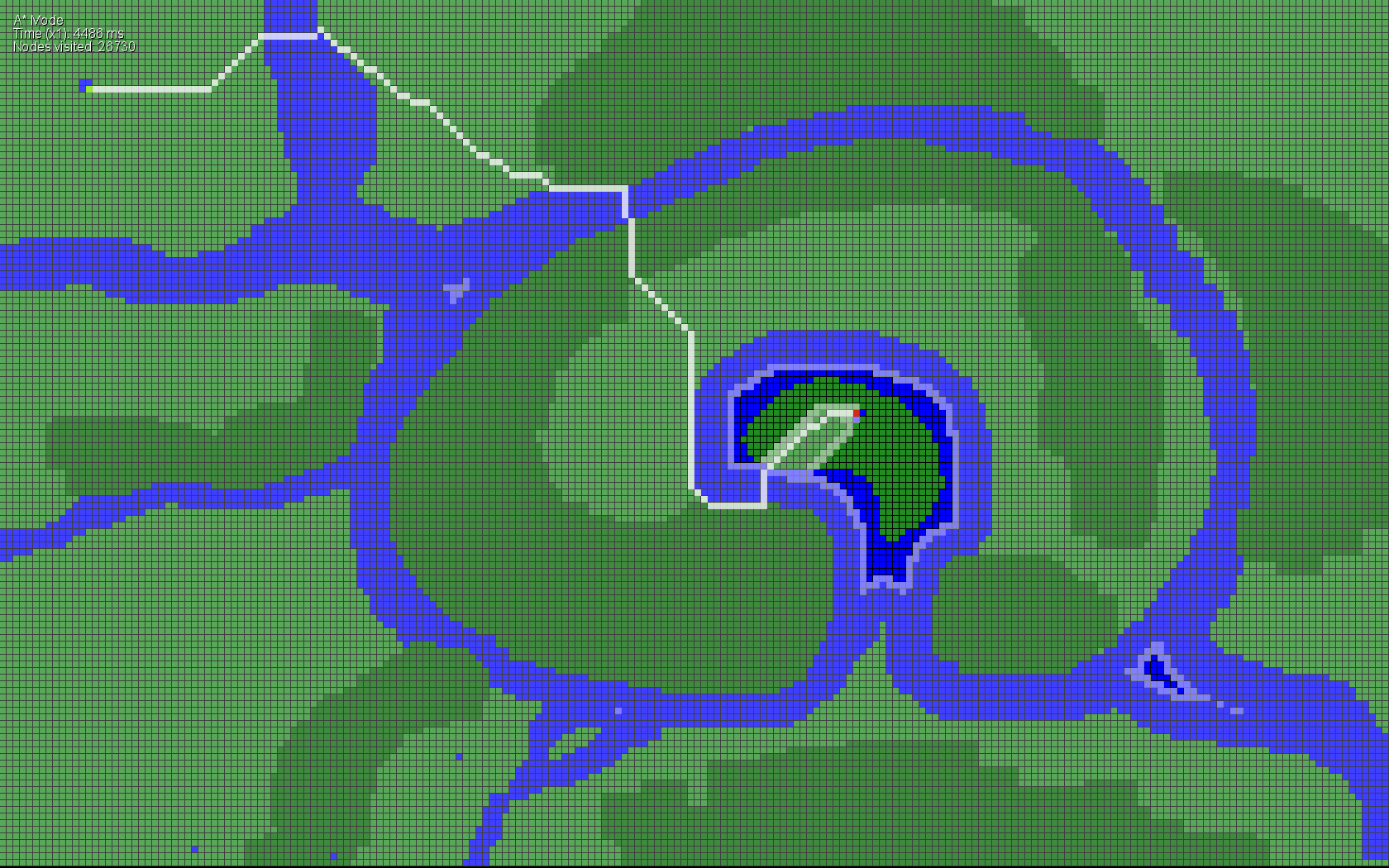
## Skenaario 4: Saarelle



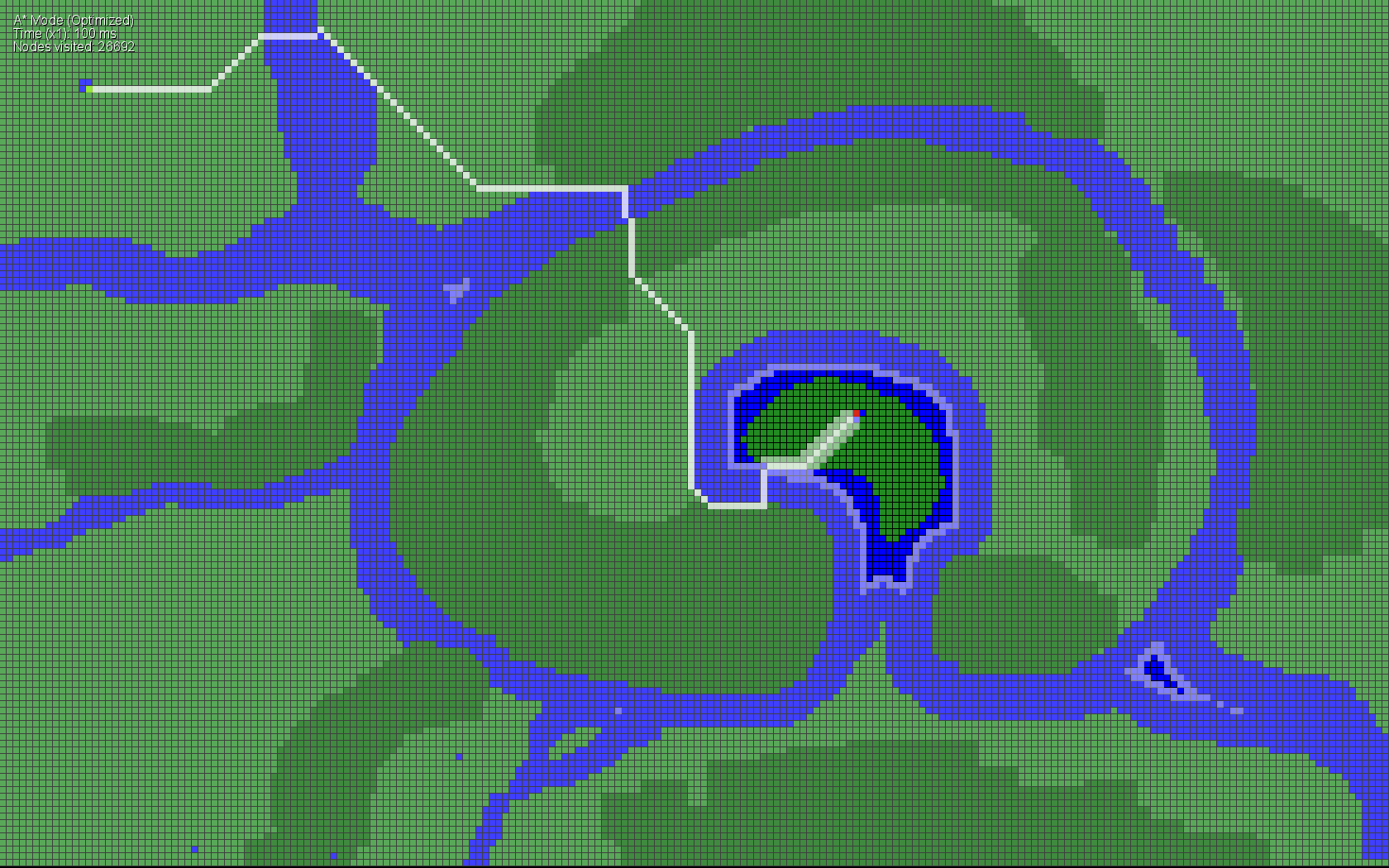
Kuva 19 Dijkstra: 6159 ms, 26893 solmua



Kuva 20 Dijkstra optimoitu: 89 ms, 26892 solmua

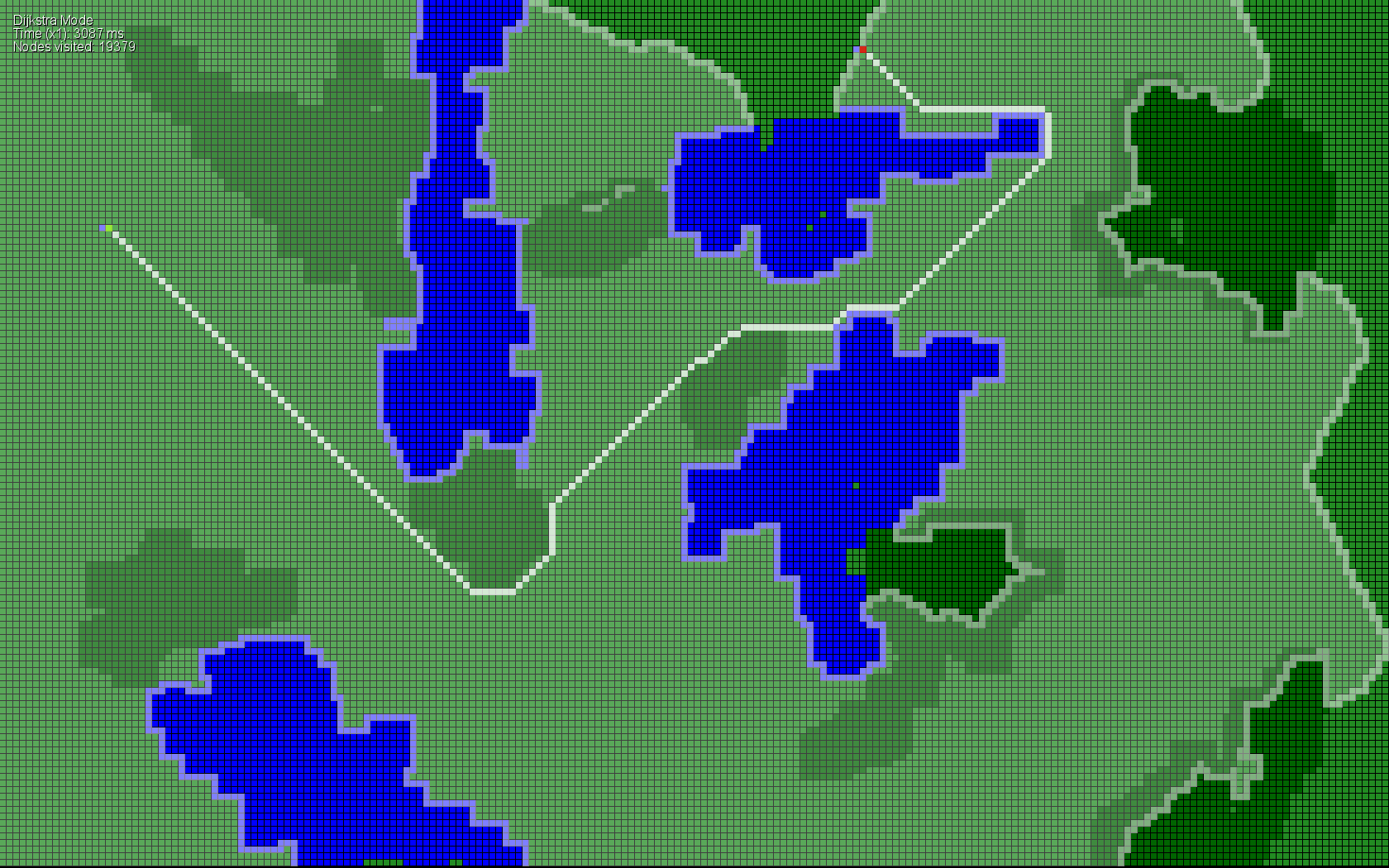


Kuva 21 A\*: 4486 ms, 26730 solmua

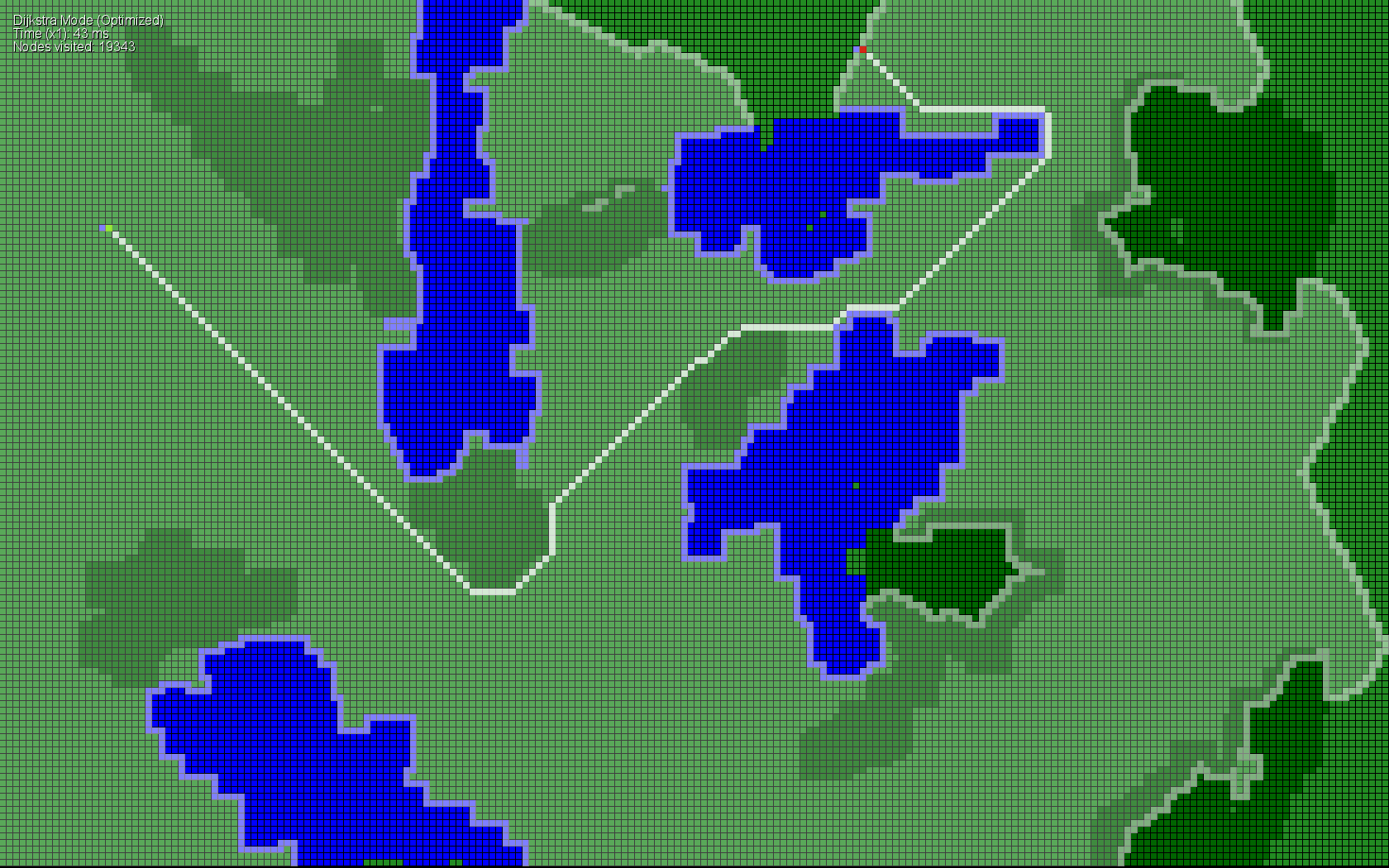


Kuva 22 A\* optimoitu: 100 ms, 26692 solmua

## Skenaario 5: Järvien ympäri



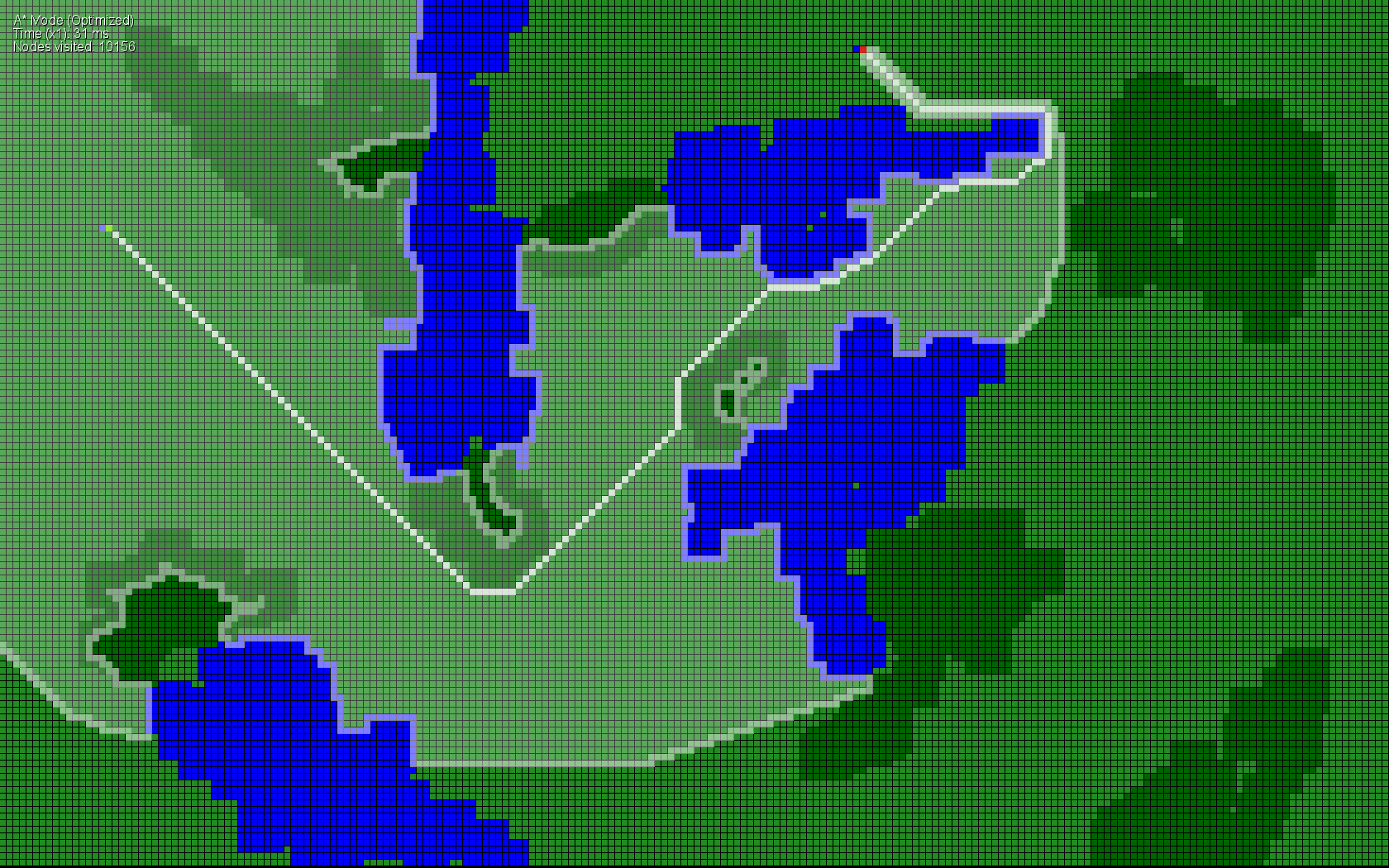
Kuva 23 Dijkstra: 3087 ms, 19379 solmua



Kuva 24 Dijkstra optimoitu: 43 ms, 19343 solmua

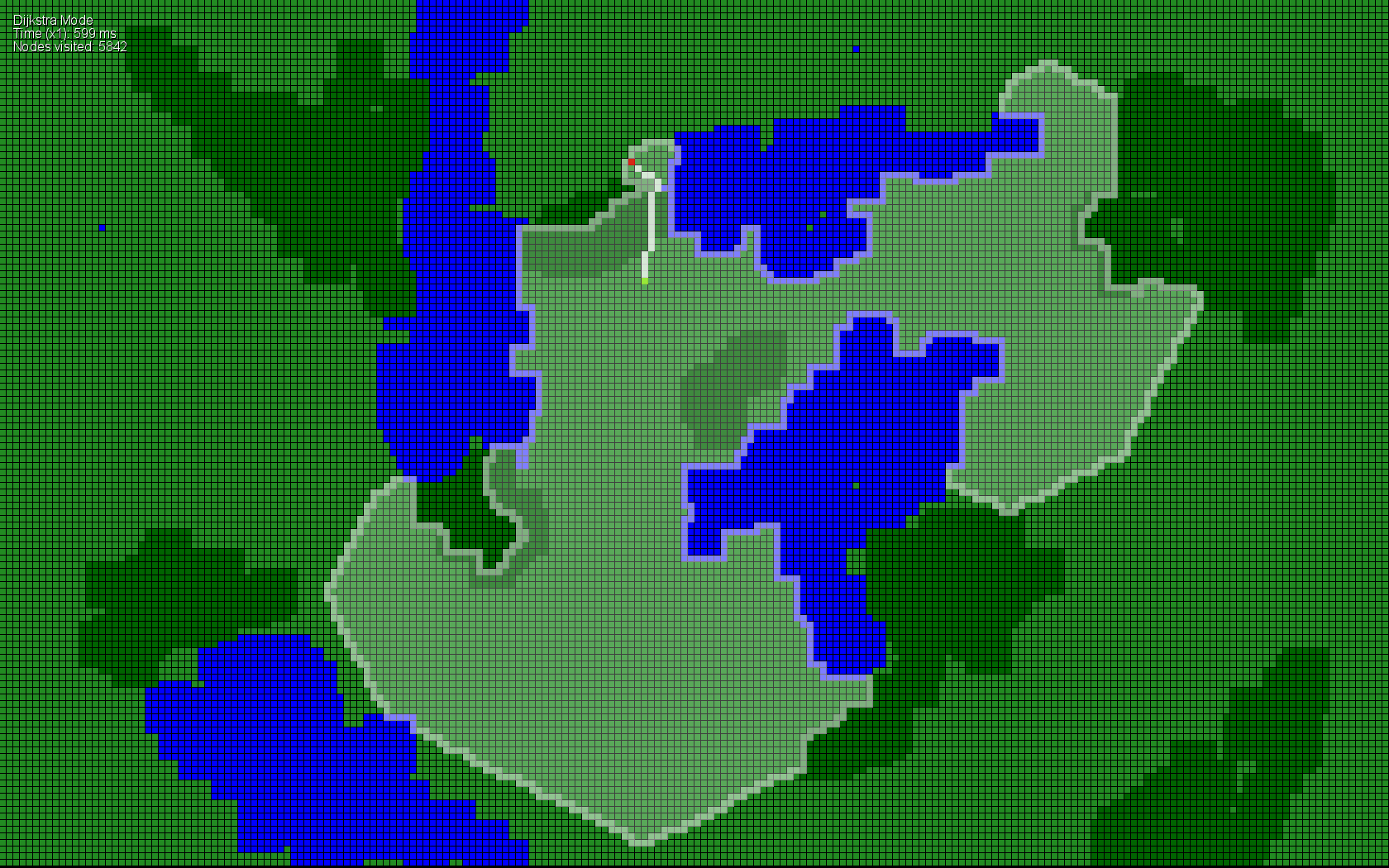


Kuva 25 A\*: 1132 ms, 10246 solmua

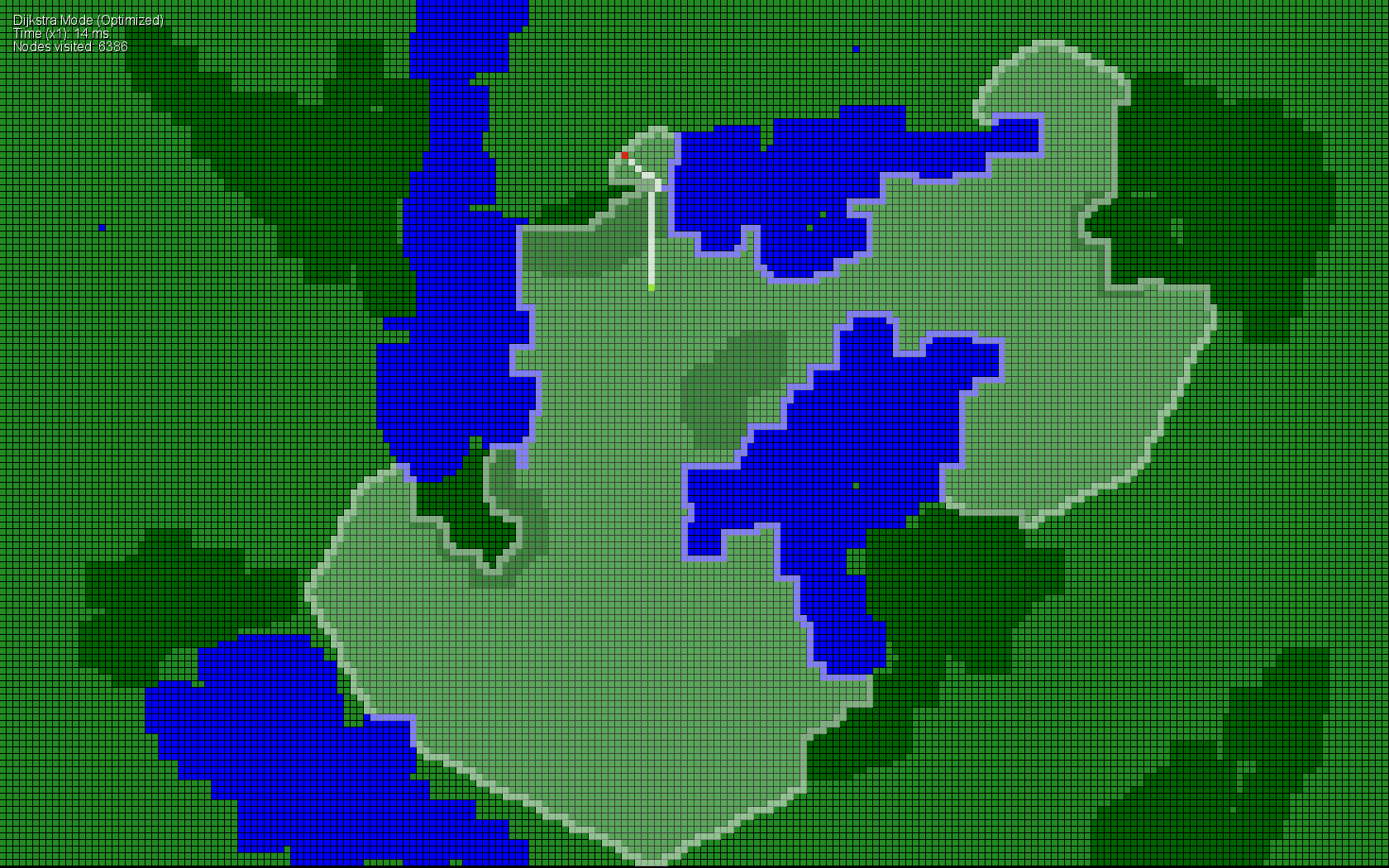


Kuva 26 A\* optimoitu: 31 ms, 10156 solmua

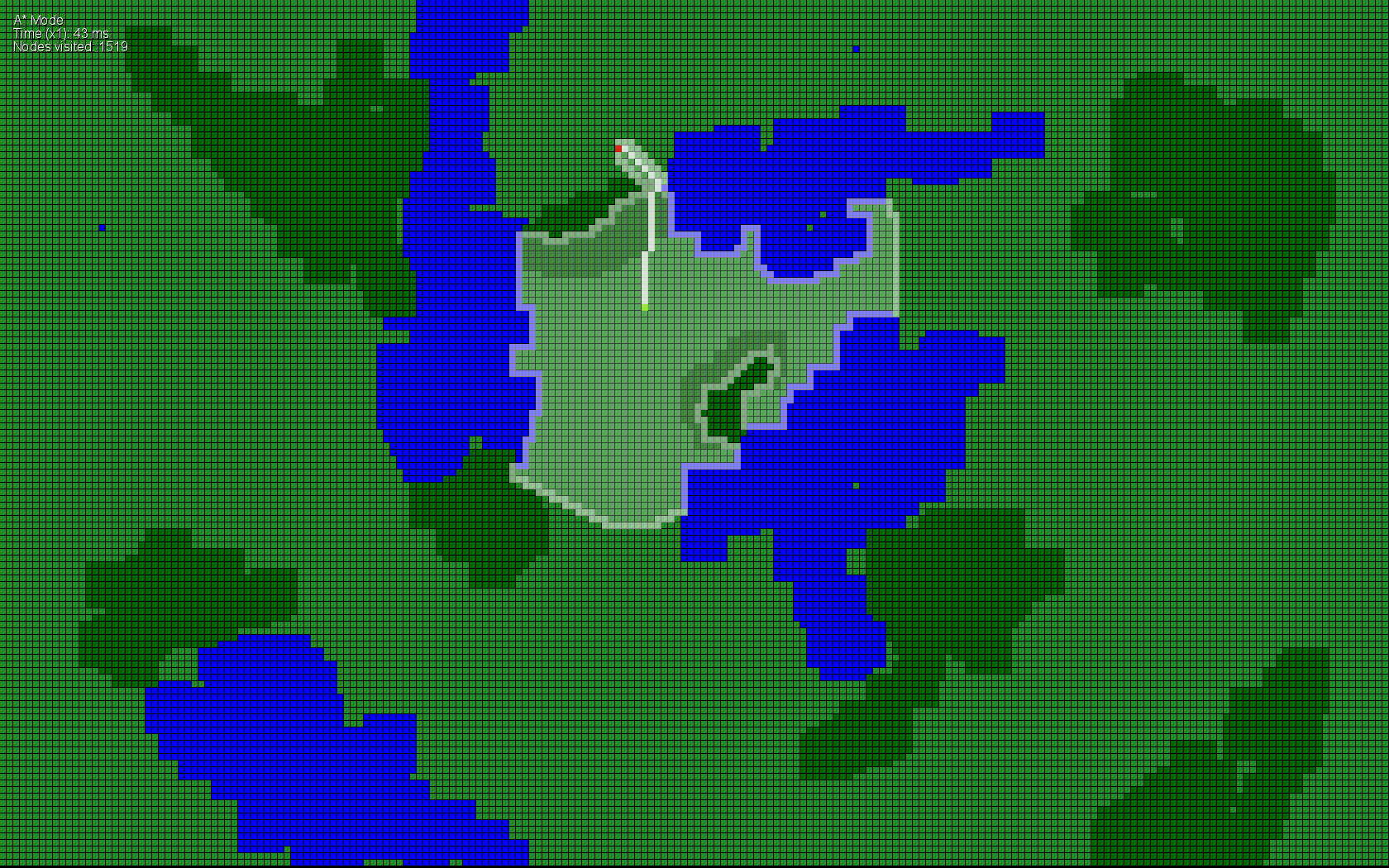
## Skenaario 6: Metsän läpi



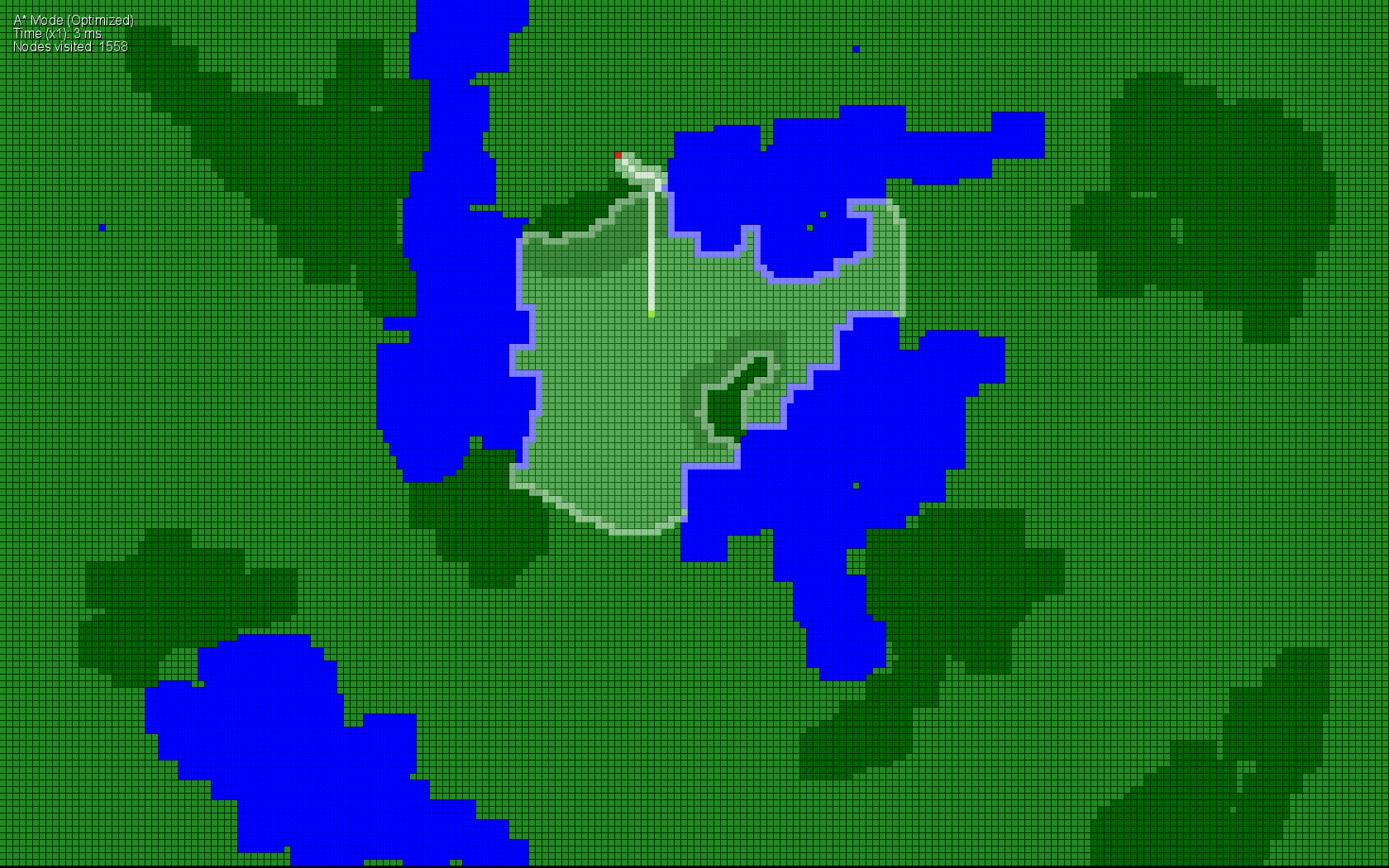
Kuva 27 Dijkstra: 599 ms, 5842 solmua



Kuva 28 Dijkstra optimoitu: 14 ms, 6386 solmua



Kuva 29 A\*: 43 ms, 1519 solmua



Kuva 30 A\* optimoitu: 3 ms, 1558 solmua

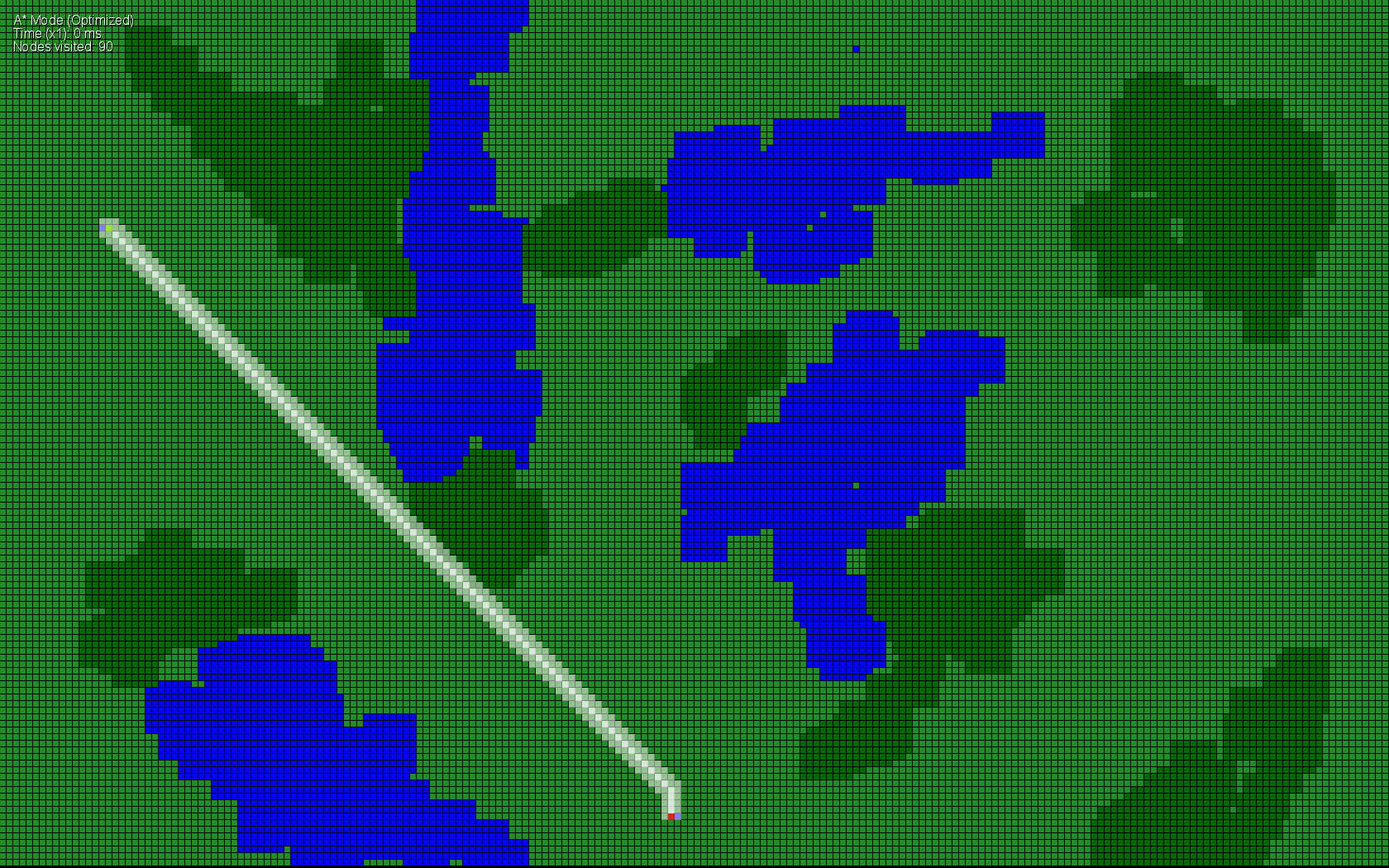
## Skenaario 7: Suora reitti esteiden ohi



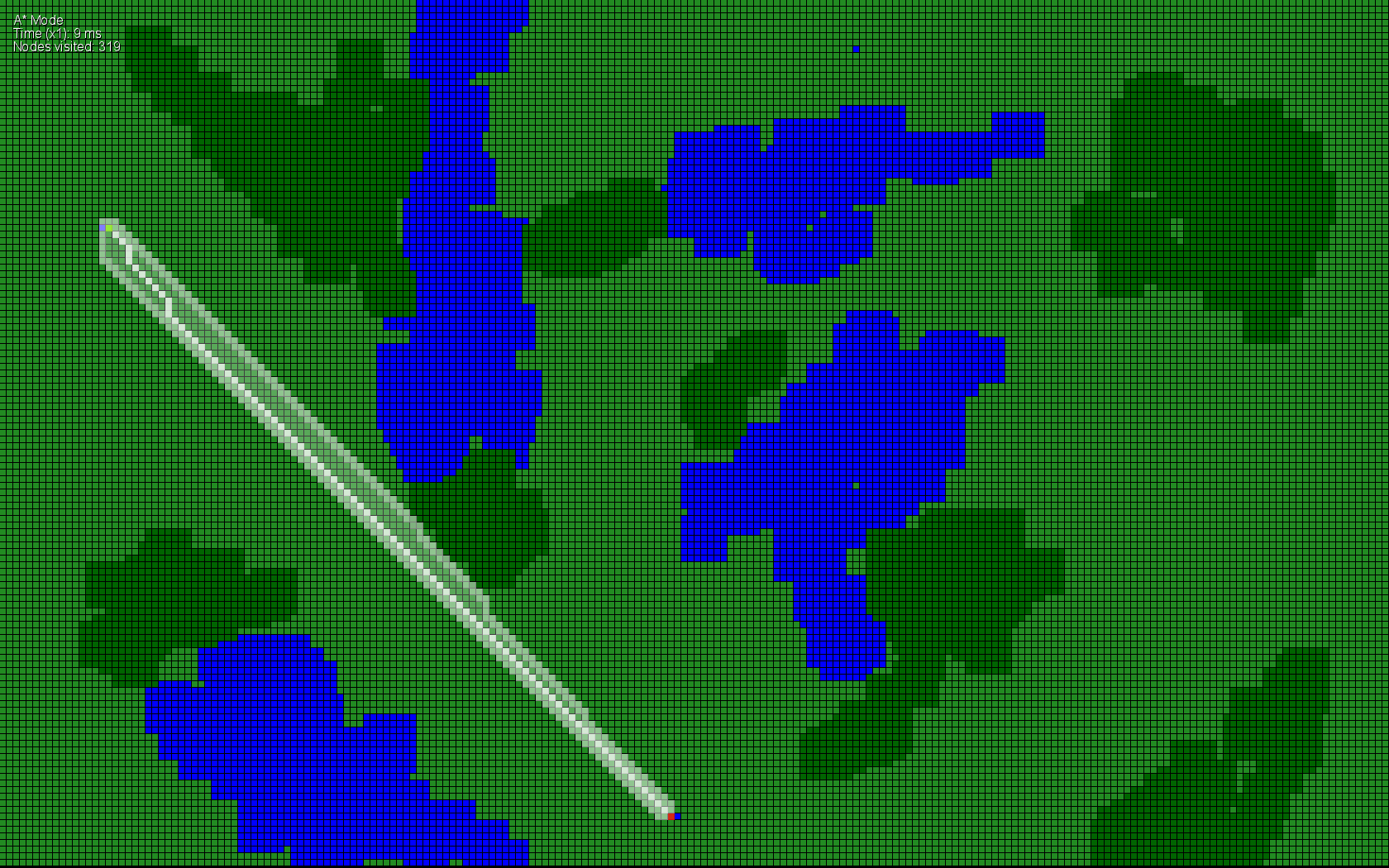
Kuva 31 Dijkstra: 703 ms, 9686 solmua



Kuva 32 Dijkstra optimoitu: 18 ms, 9681 solmua

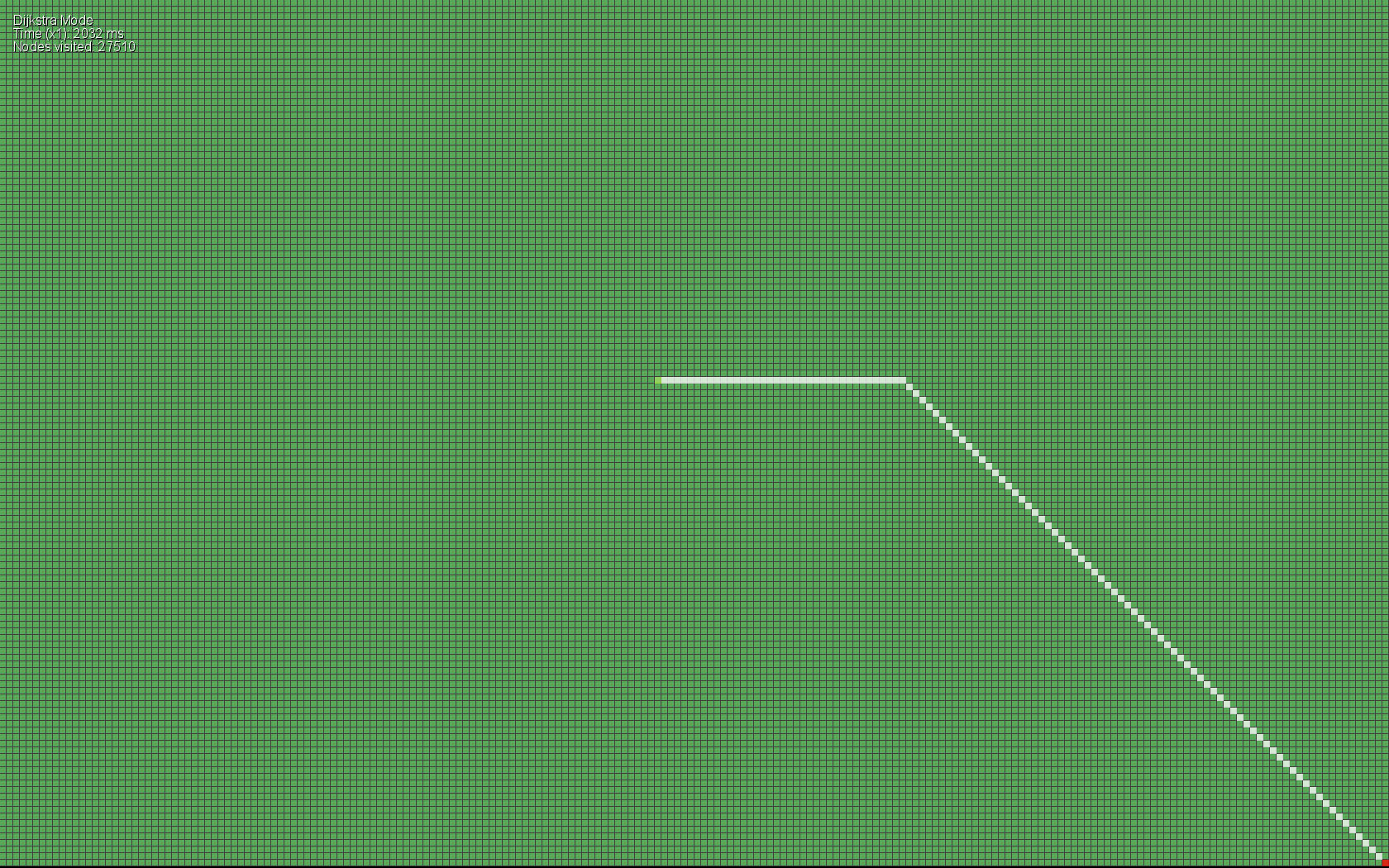


Kuva 33 A\*: 9 ms, 319 solmua

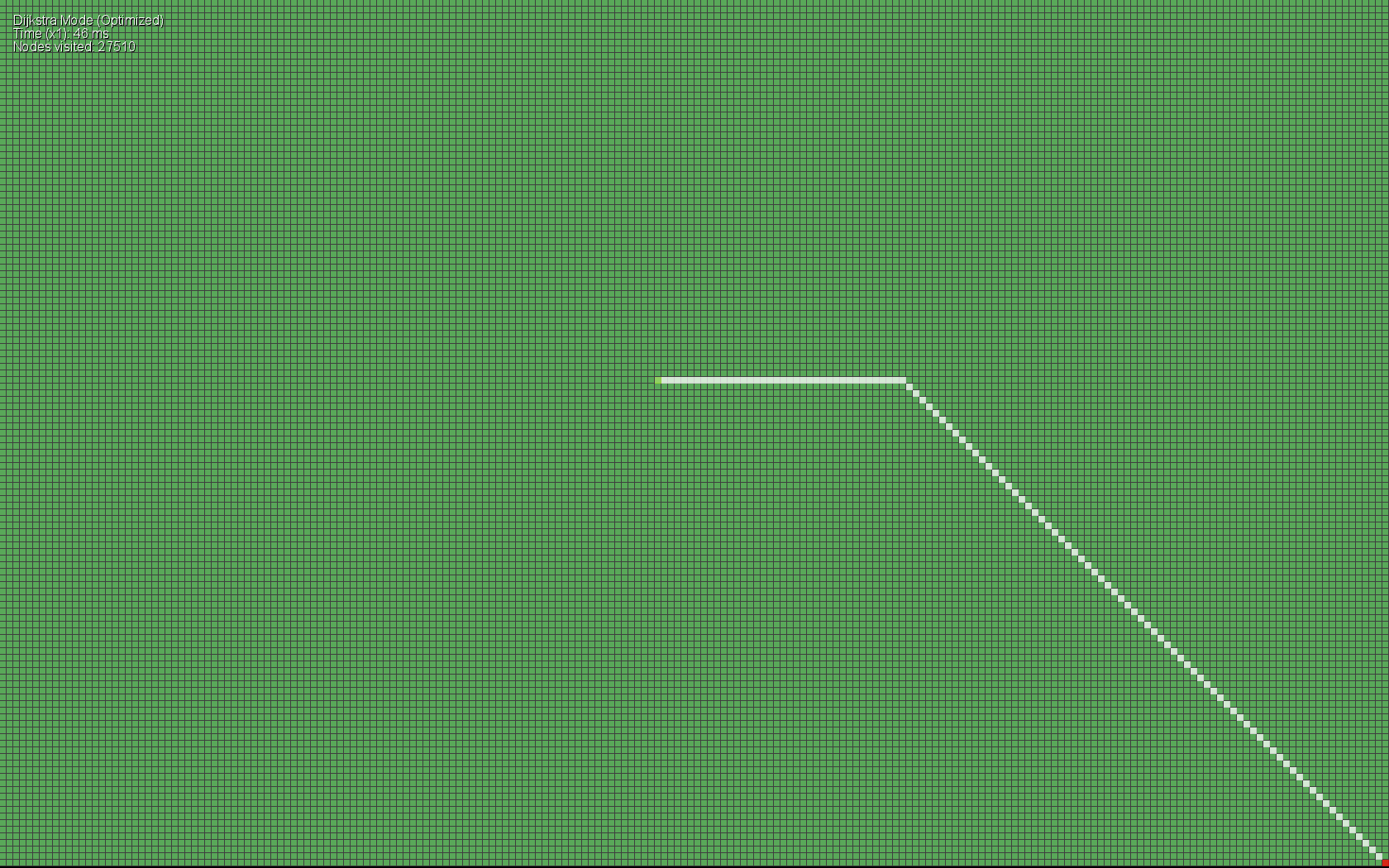


Kuva 34 A\* optimoitu: 0 ms, 90 solmua

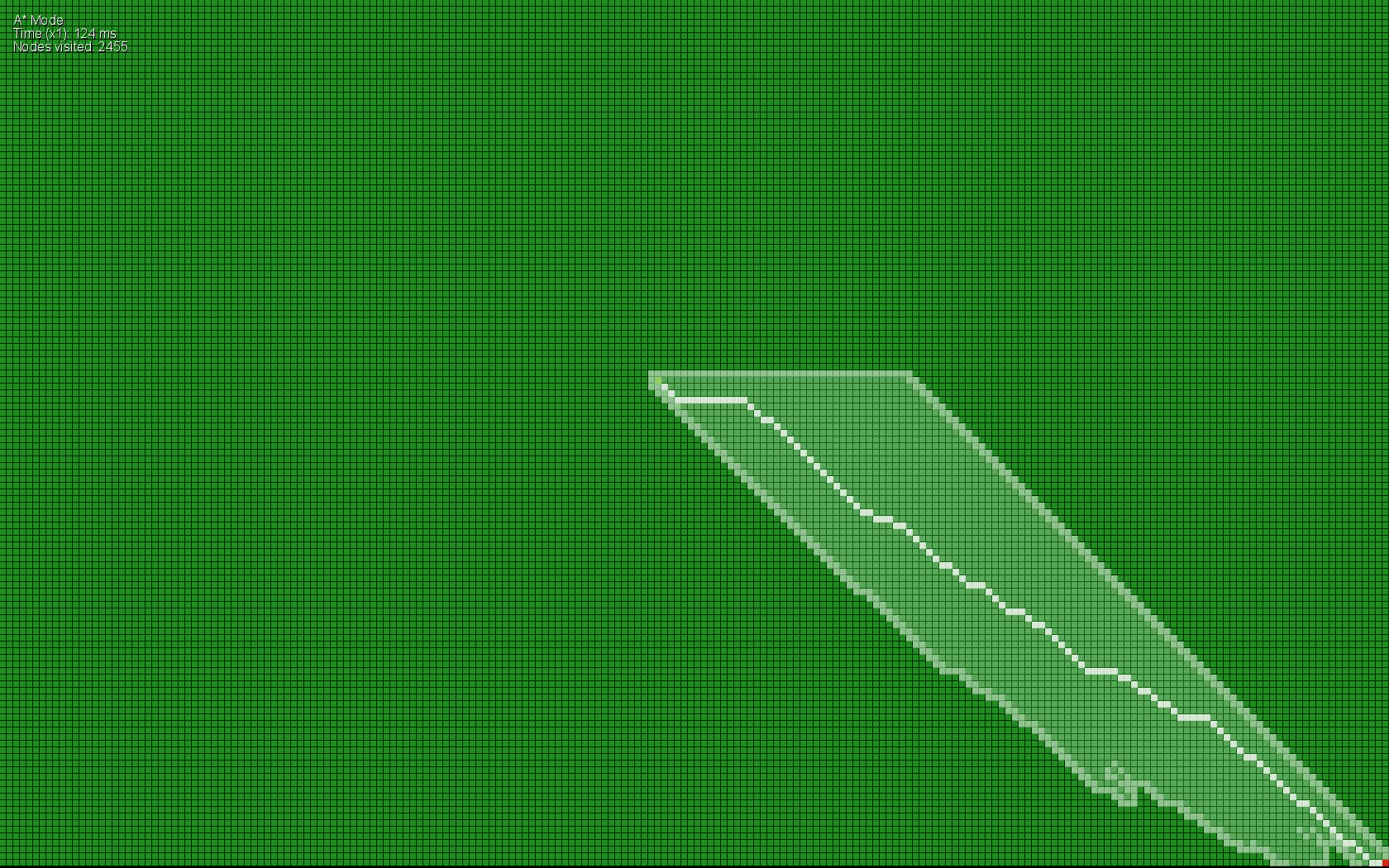
## Skenaario 8: Tyhjä tasanko



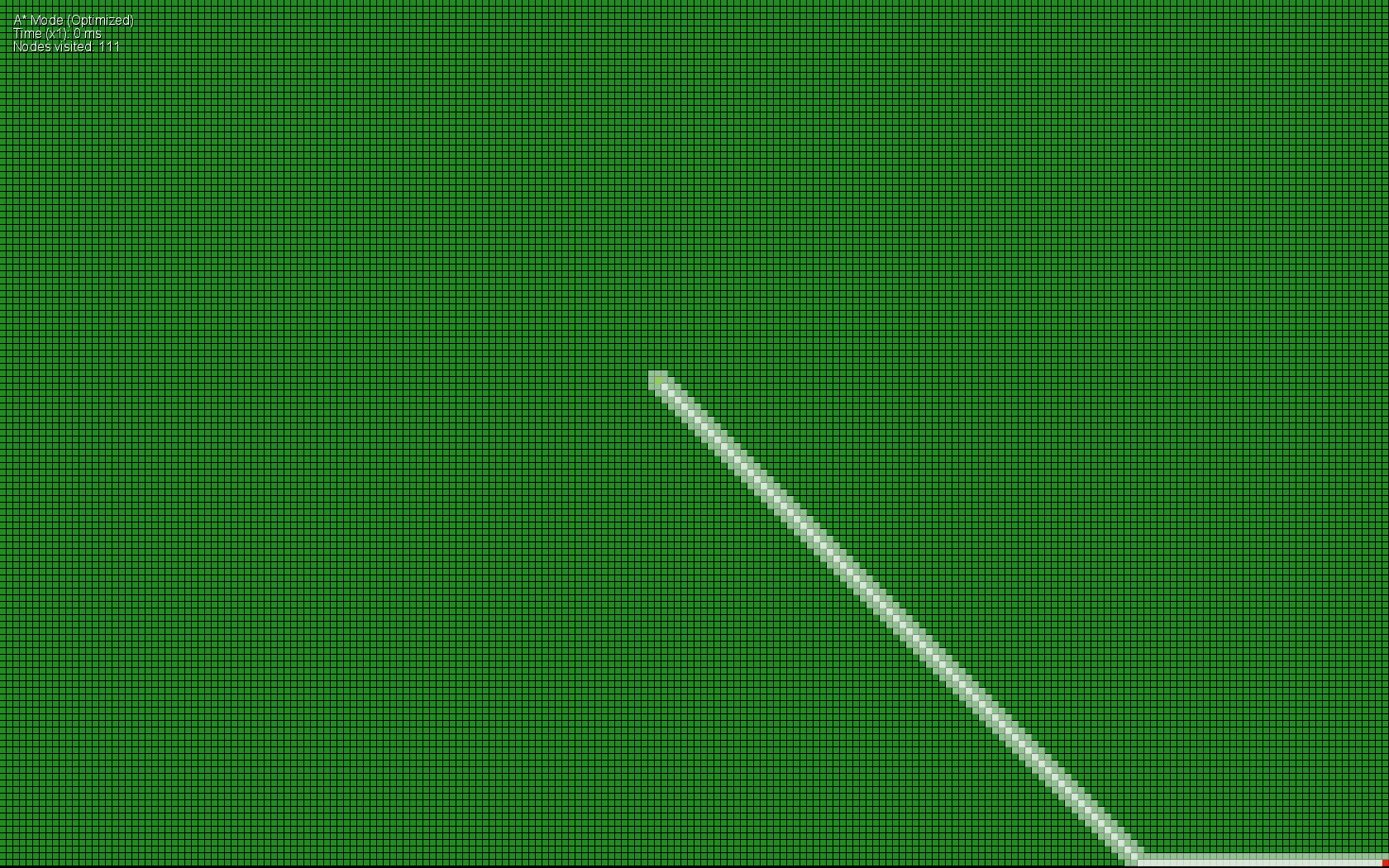
Kuva 35 Dijkstra: 2032 ms, 27510 solmua



Kuva 36 Dijkstra optimoitu: 46 ms, 27510 solmua

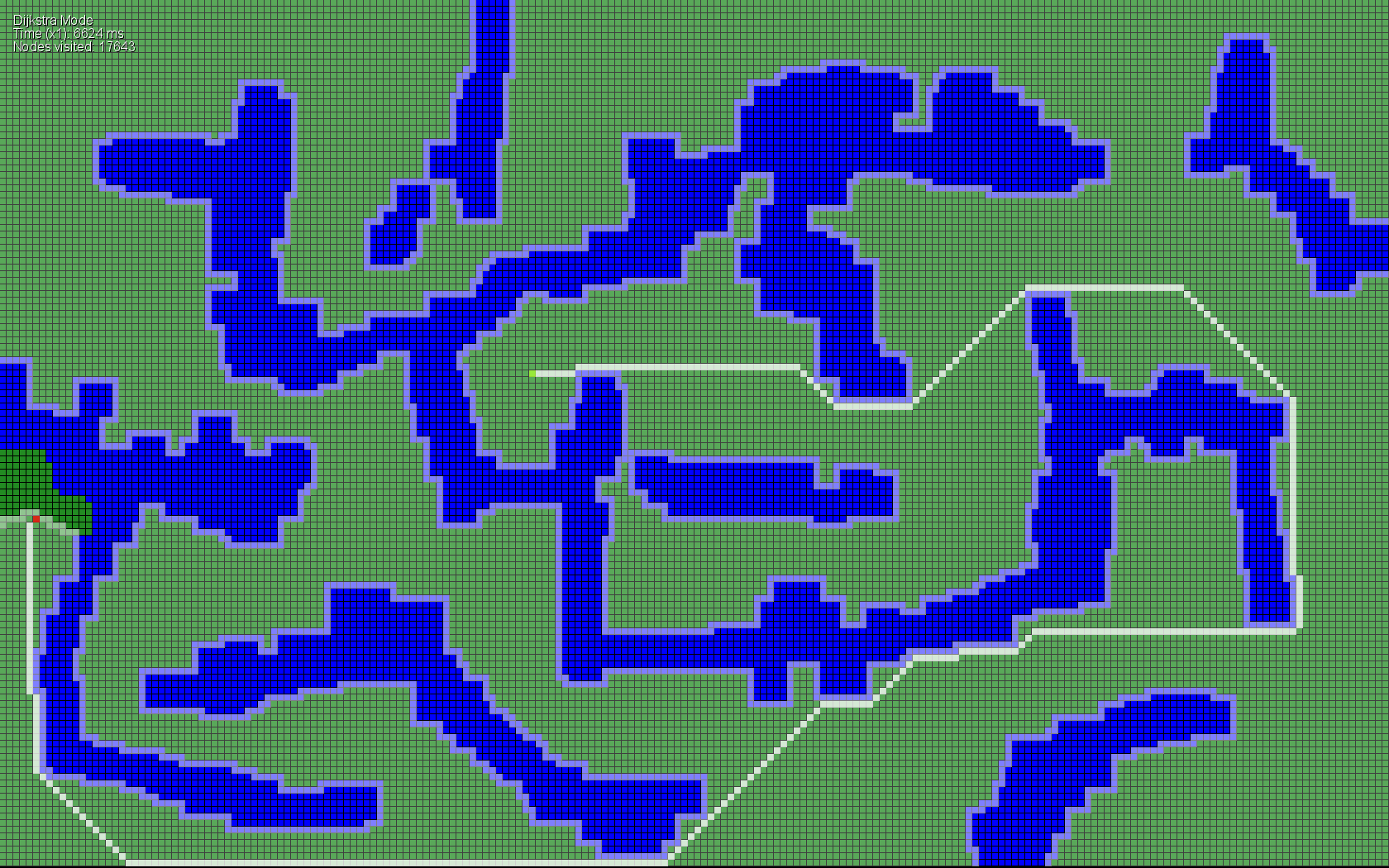


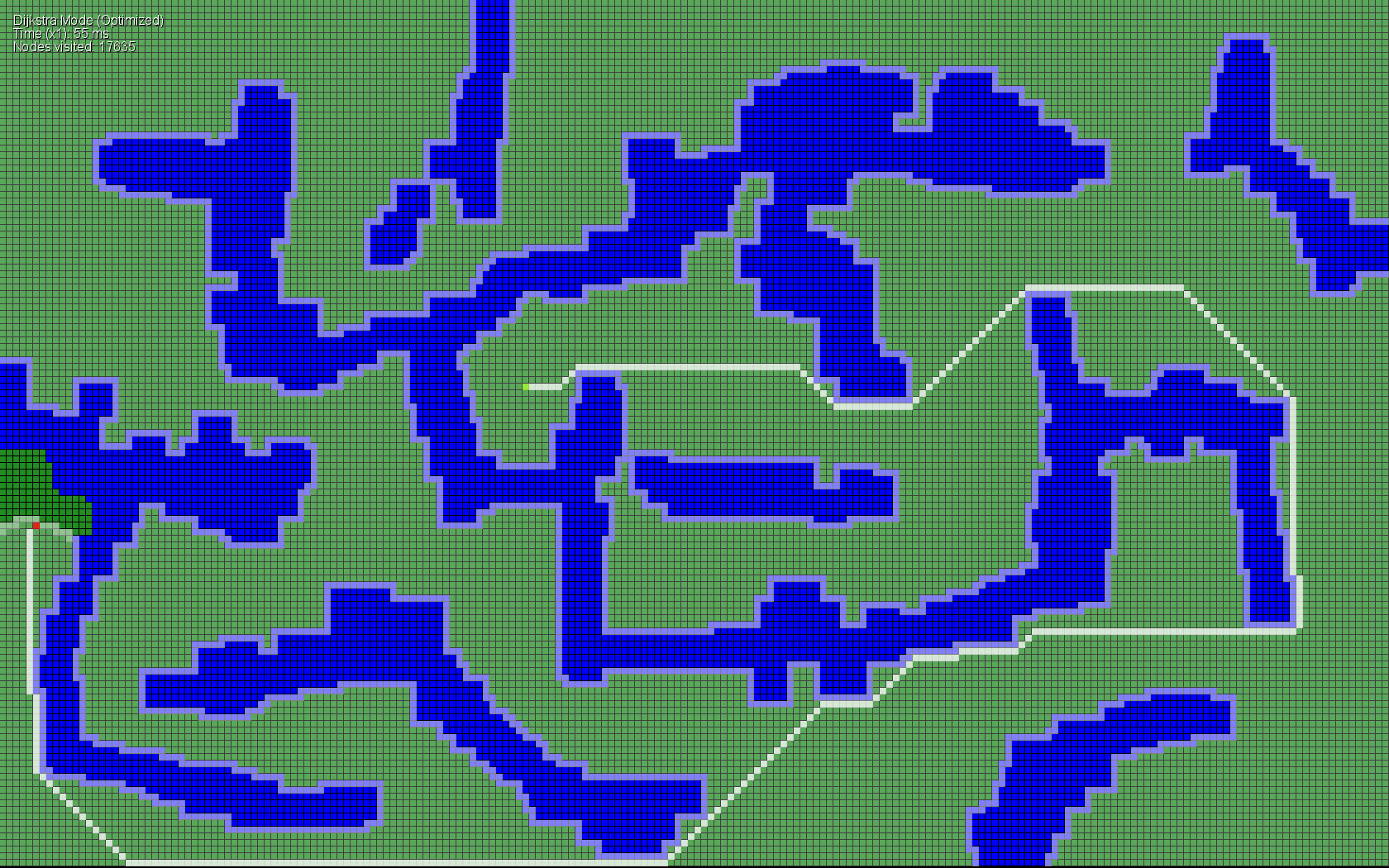
Kuva 37 A\*: 124 ms, 2455 solmua



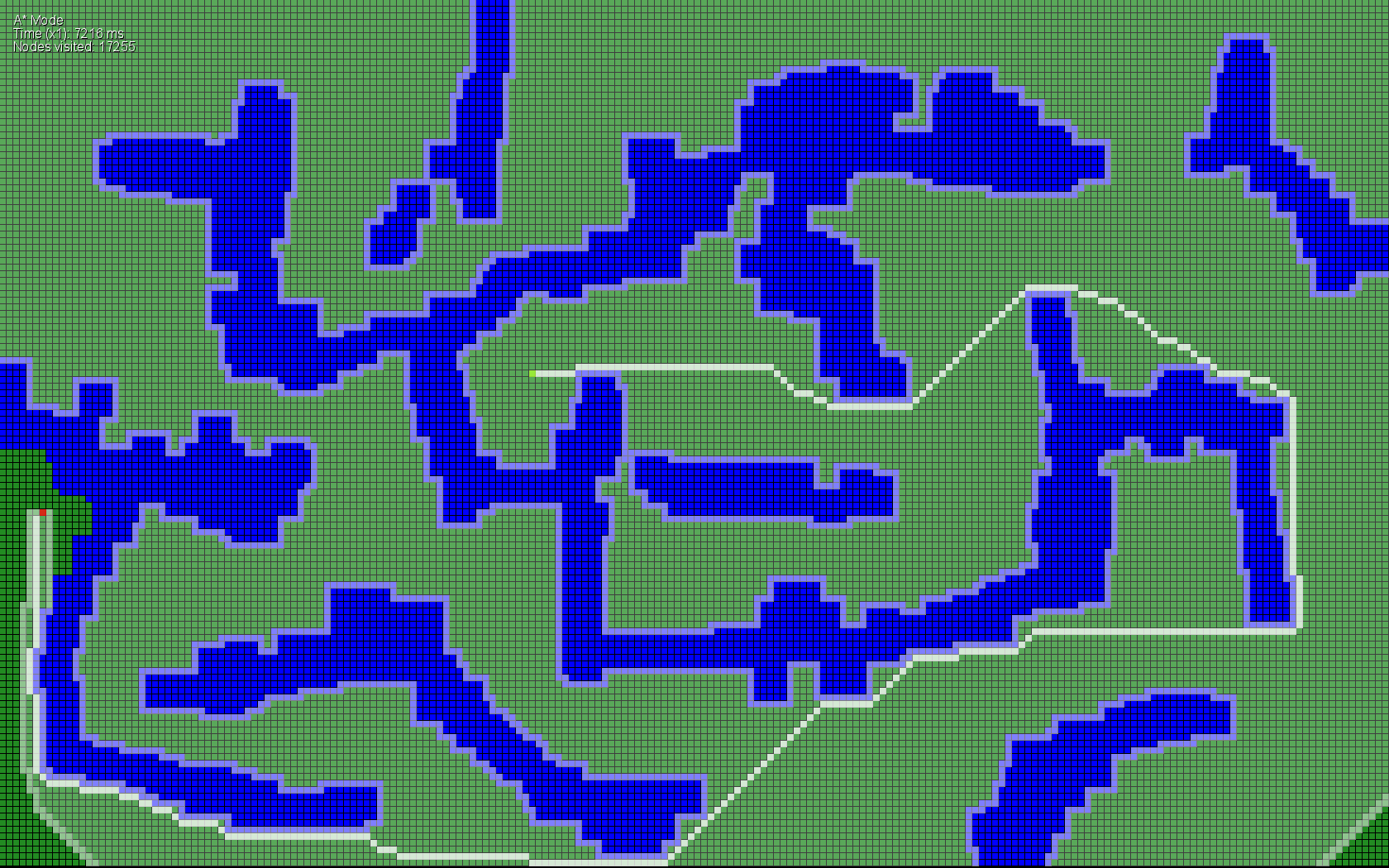
Kuva 38 A\* optimoitu: 0 ms, 111 solmua

## Skenaario 9: Sokkelo

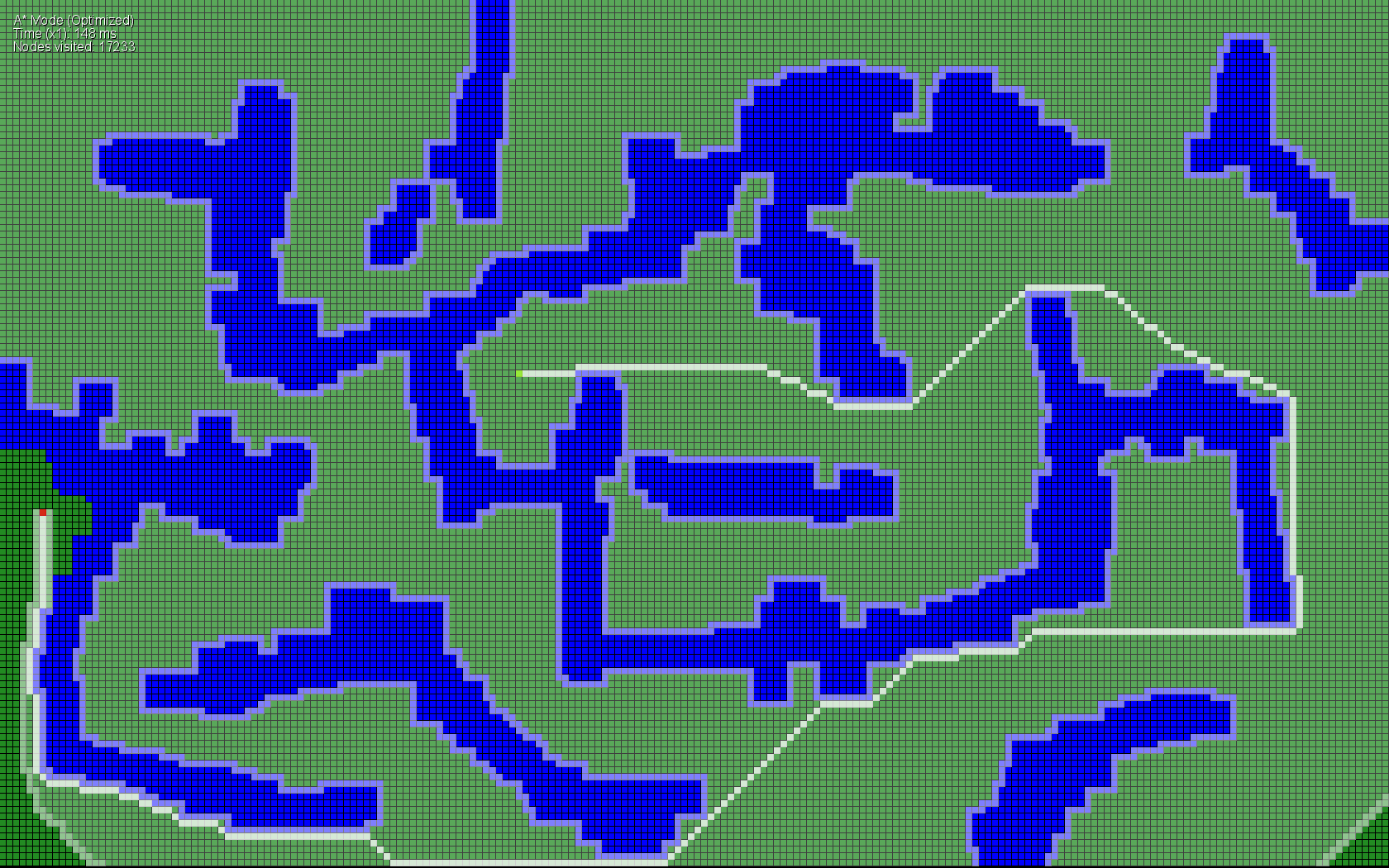


Kuva 39 Dijkstra: 6624 ms, 17643 solmua

Kuva 40 Dijkstra optimoitu: 55 ms, 17635 solmua



Kuva 41 A\*: 7216 ms, 17255 solmua



Kuva 42 A\* optimoitu: 148 ms, 17233 solmua

# Yhteenveto

Harjoitustyössä on saavutettu asetetut tavoitteet, varsinkin teknisessä puolella, mutta ei ongelmitta. Virheiden löytäminen ja korjaus vie huomattavan määrän aikaa, mutta lopulta algoritmit saatiin toimimaan teorian mukaisesti.

Algoritmien tehokkuus riippuu vahvasti maaston muodosta, joten algoritmin tehokkuudelle on vaikea saada yleistä kaavaa ja mahdollisten kombinaatioiden määrä tekee empiirisen mittauksenkin hankalaksi. Testien ja intuition avulla voi kuitenkin päätellä joitakin yleistyksiä. A\* algoritmi esimerkiksi pärjää hyvin avoimessa maastossa, jossa on vähän esteitä. Dijkstran algoritmi puolestaan loistaa sokkeloisessa maastossa. Tämä siksi, että A\* voi juuttua esteisiin, joka johtaa siihen, että se käy läpi lähes saman määrän solmuja, mikä vie sen edun verrattuna Dijkstran algoritmiin. Koska A\* joutuu laskemaan heuristiikan arvoja, se tässä tapauksessa tekee enemmän töitä verrattuna Dijkstran algoritmiin.

# Lähteet

Dijkstra’s algorithm. N.d. Artikkeli Wikipedian sivustolla. Viitattu 4.12.2016. <https://en.wikipedia.org/wiki/Dijkstra's_algorithm>