Dijkstran algoritmi

Miska Sainkangas

Ville Rantala

Harjoitustyö

11 2016

Tieto- ja viestintätekniikka

Tekniikan ja liikenteen ala

Sisällysluettelo

[1 Johdanto 2](#_Toc468459025)

[2 Visualisoinnin määritys ja tavoitteet 2](#_Toc468459026)

[3 Toteutus 3](#_Toc468459027)

[4 Testiaineiston generointi, testaus 5](#_Toc468459028)

[4.1 Skenaario 1: Reikä 5](#_Toc468459029)

[4.2 Skenaario 2: Metsän ohi 7](#_Toc468459030)

[4.3 Skenaario 3: Etelään 9](#_Toc468459031)

[4.4 Skenaario 4: Saarelle 11](#_Toc468459032)

[4.5 Skenaario 5: Järven ympäri 13](#_Toc468459033)

[4.6 Skenaario 6: Metsän läpi 15](#_Toc468459034)

[4.7 Skenaario 7: Suora reitti esteiden ohi 17](#_Toc468459035)

[4.8 Skenaario 8: Tyhjä tasanko 19](#_Toc468459036)

[4.9 Skenaario 9: Sokkelo 21](#_Toc468459037)

[5 Yhteenveto 23](#_Toc468459038)

[6 Lähteet 23](#_Toc468459039)

# Johdanto

Tämän harjoitustyön tarkoituksena on tutkia reititysalgoritmeihin kuuluvaa Dijkstran algoritmia ja analysoida sen toimintaa niin teoriassa kuin käytännössä.

# Visualisoinnin määritys ja tavoitteet

Kirjoitetaan ohjelma, jonka tarkoitus on demonstroida Dijkstran algoritmin toimintaa. Tavoitteena on saada ohjelma toimimaan tehokkaasti ja luotettavasti. Ohjelmasta tehdään graafinen, jotta Dijkstran toiminta voidaan nähdä havainnollisesti.

Visualisoinnista tulee näkyä algoritmin toiminta vaihe vaiheelta. Aluksi lasketaan teoreettinen laskennallinen vaativuus, sitten käynnistetään ohjelma ja mitataan algoritmien suorituksien käyttämä aika.

Algoritmin paras tapaus on se, että lähde ja kohde ovat samat, jolloin algoritmi päättyy vakioajassa O (1), mutta tämä ei ole kovin yleinen eikä käytännöllinen tapaus.

Algoritmin huonoin tapaus lasketaan kaavalla: O((E+V) \* log(V))

Kuva 1 X- akseli: Linkkien määrä, Z -akseli: Solmujen määrä, Y -akseli: Aika.

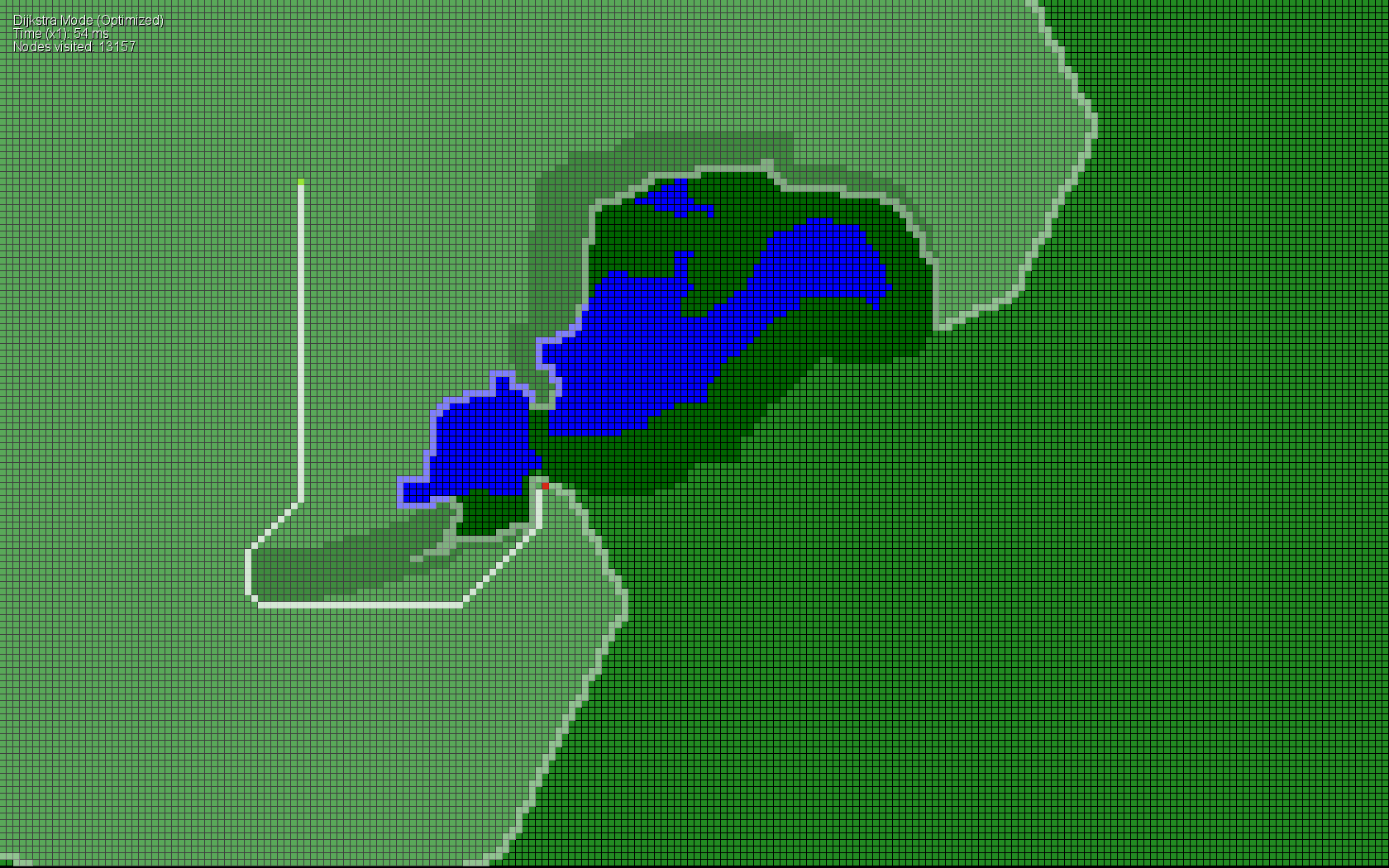
# Toteutus

Ohjelma toteutettiin C# -kielellä. Ohjelmassa on perinteinen Dijkstran algoritmi ja siitä selvästi optimoitu versio, joka käyttää hyödyksi binääripuuta. Dijkstran algoritmin lisäksi toteutettiin A\* algoritmi.

Ohjelmalla on graafinen käyttöliittymä, jolla voidaan syöttää lähdepiste, kohdepiste ja muuttaa solmujen arvoja. Solmujen määrä riippuu näytön koosta ja niiden arvoina voi olla, joko ruoho (1), vesi (10) tai metsä (1000). Pisteiden ollessa valittuna ohjelma laskee taustalla lyhimmän reitin algoritmin avulla ja näyttää käyttöliittymässä tuloksen. Algoritmin lopetusehto on lopettaa toimintansa, kun lyhin reitti alkupisteestä kohdepisteeseen on löydetty.



Kuva 2 Solmuille annettu arvoiksi ruohoa, vettä ja metsää.



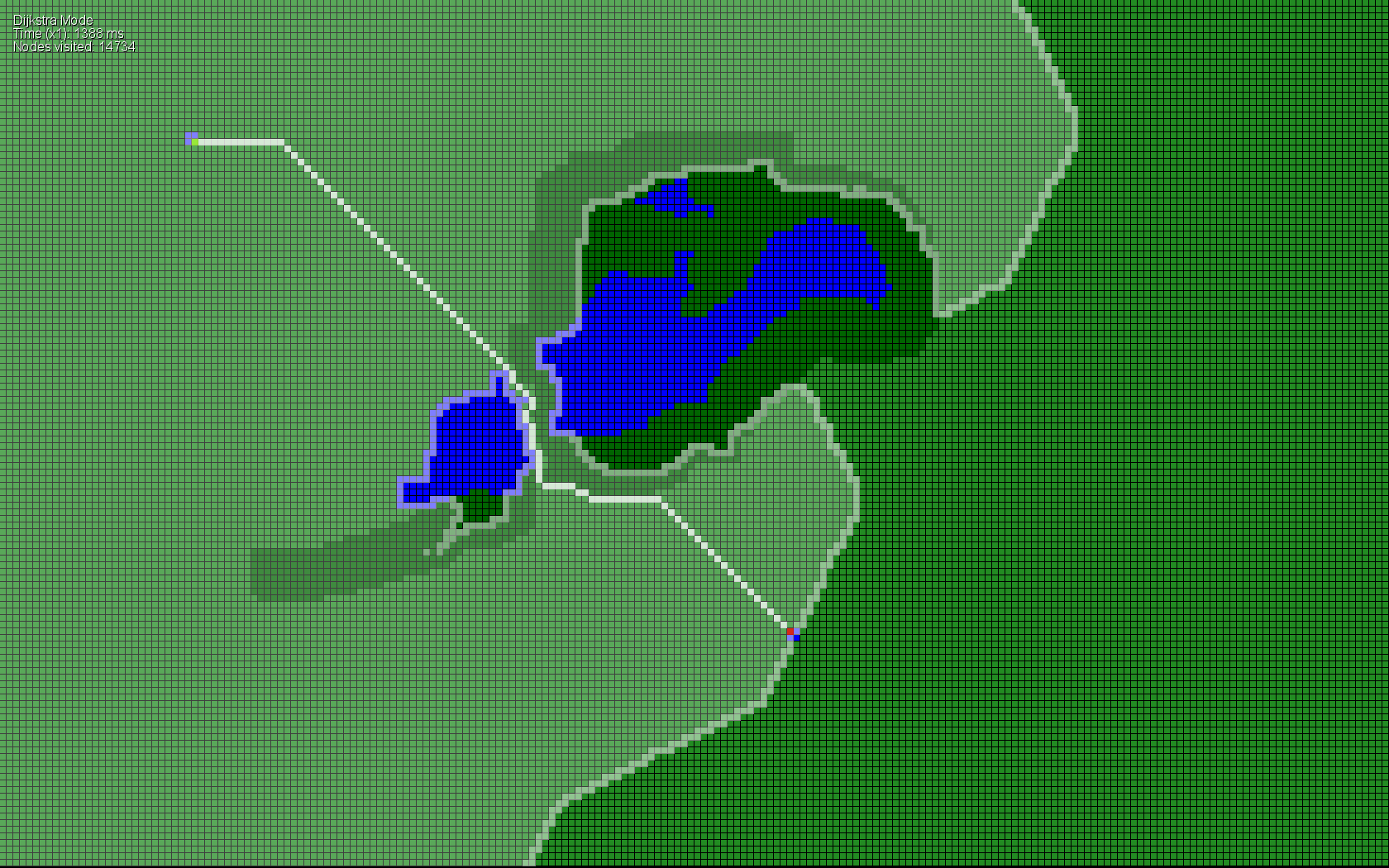
Kuva 3 Vaaleanvihreä piste on lähtöpiste ja punainen kohdepiste. Käydyt solmut näkyvät vaaleana alueena.



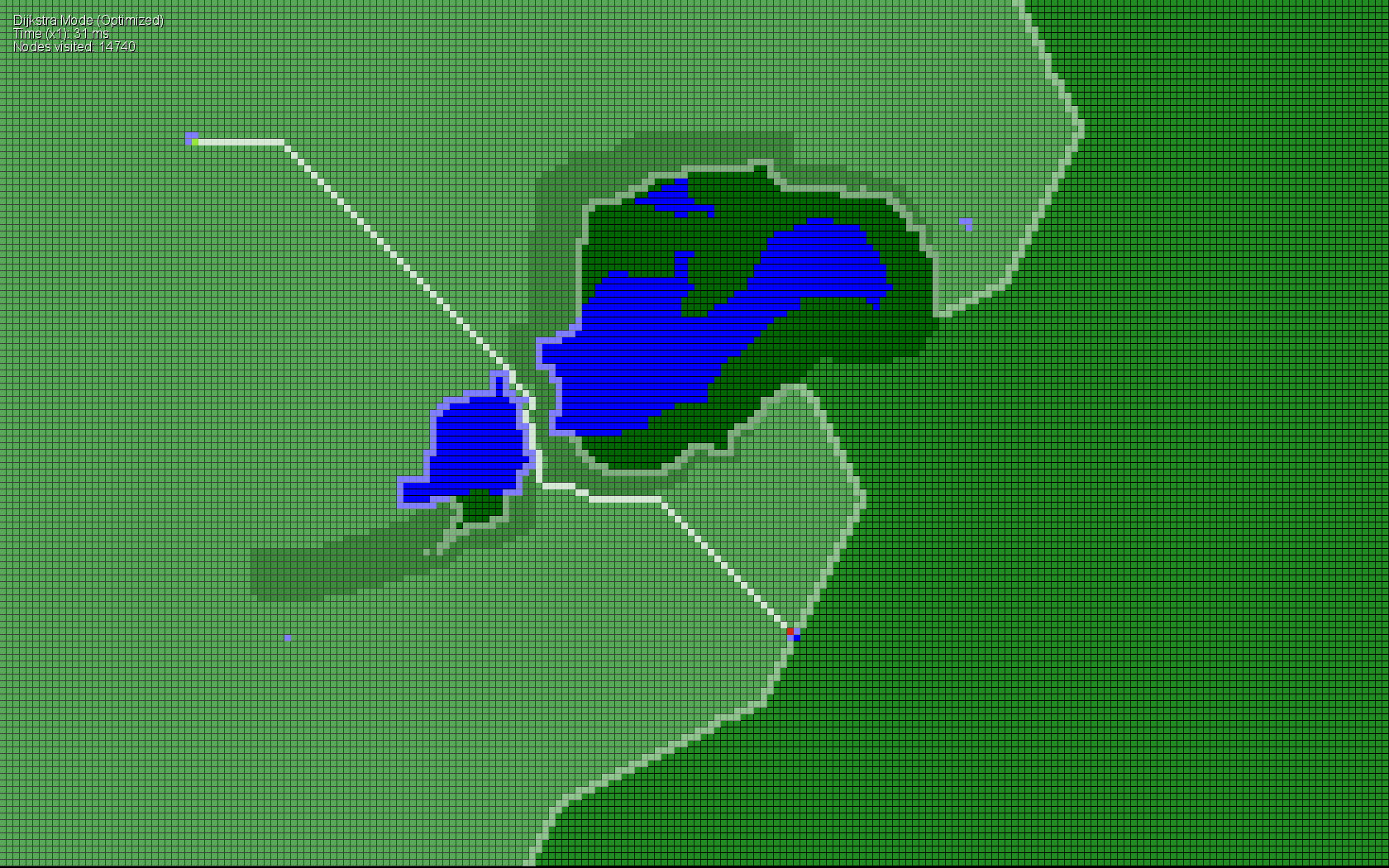
Kuva 4 Laskennan nopeus sekä kuinka monessa solmussa ollaan käyty.

# Testiaineiston generointi, testaus

## Skenaario 1: Reikä



Kuva 5 Dijkstra: 1388 ms, 14734 solmua



Kuva 6 Dijkstra optimoitu: 31 ms, 14740 solmua

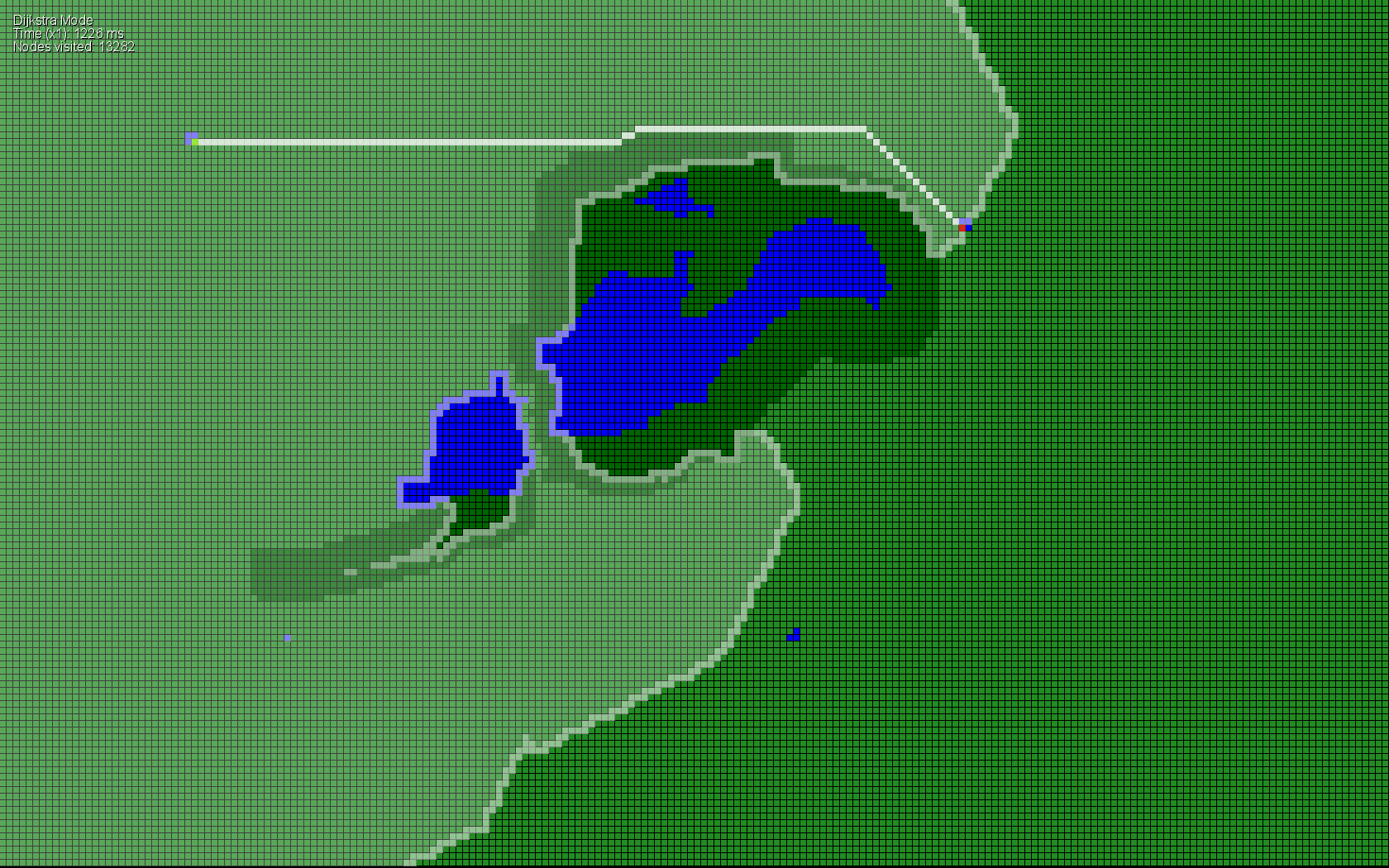


Kuva 7 A\*: 81 ms, 1967 solmua

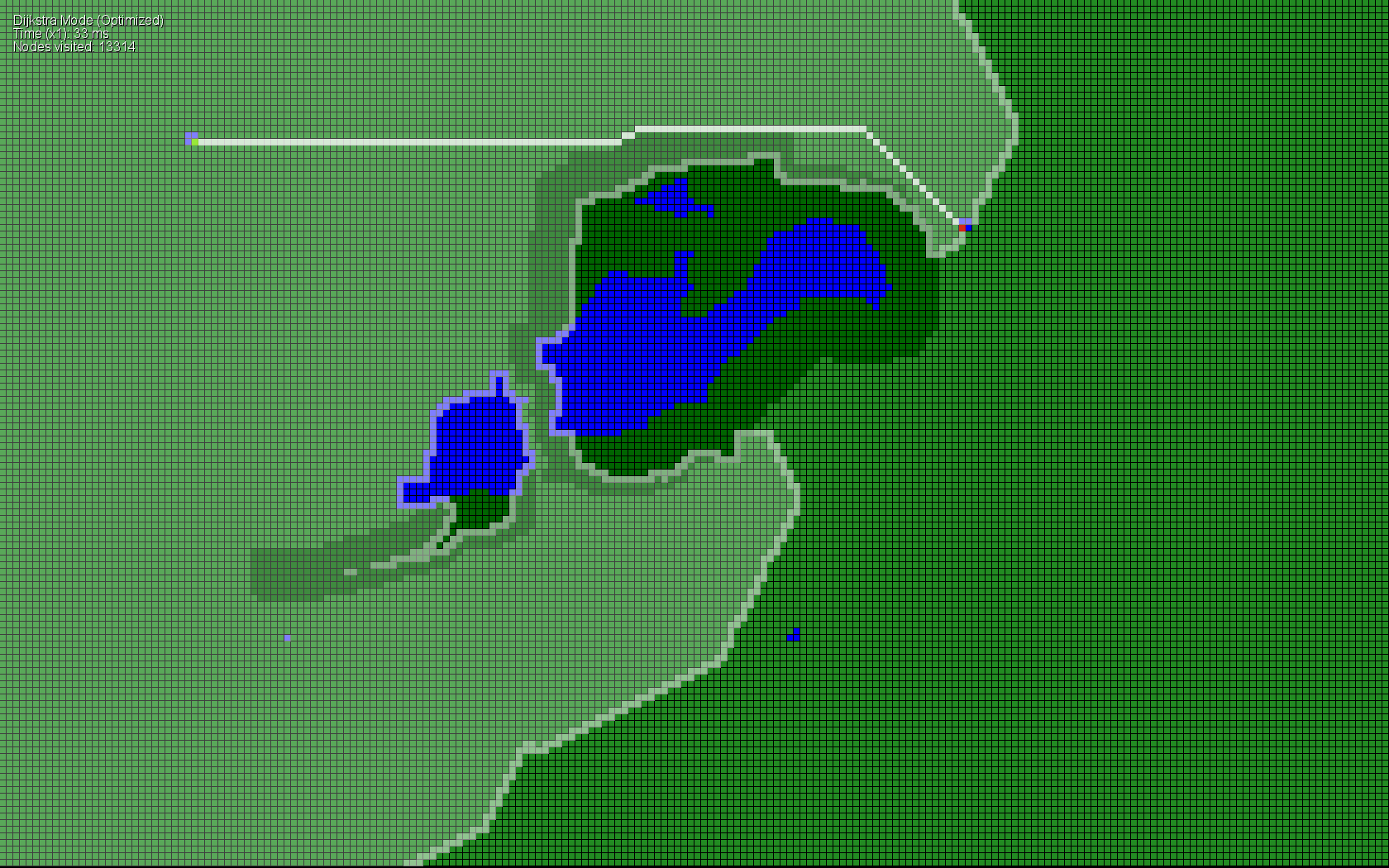


Kuva 8 A\* optimoitu 7 ms, 1638 solmua

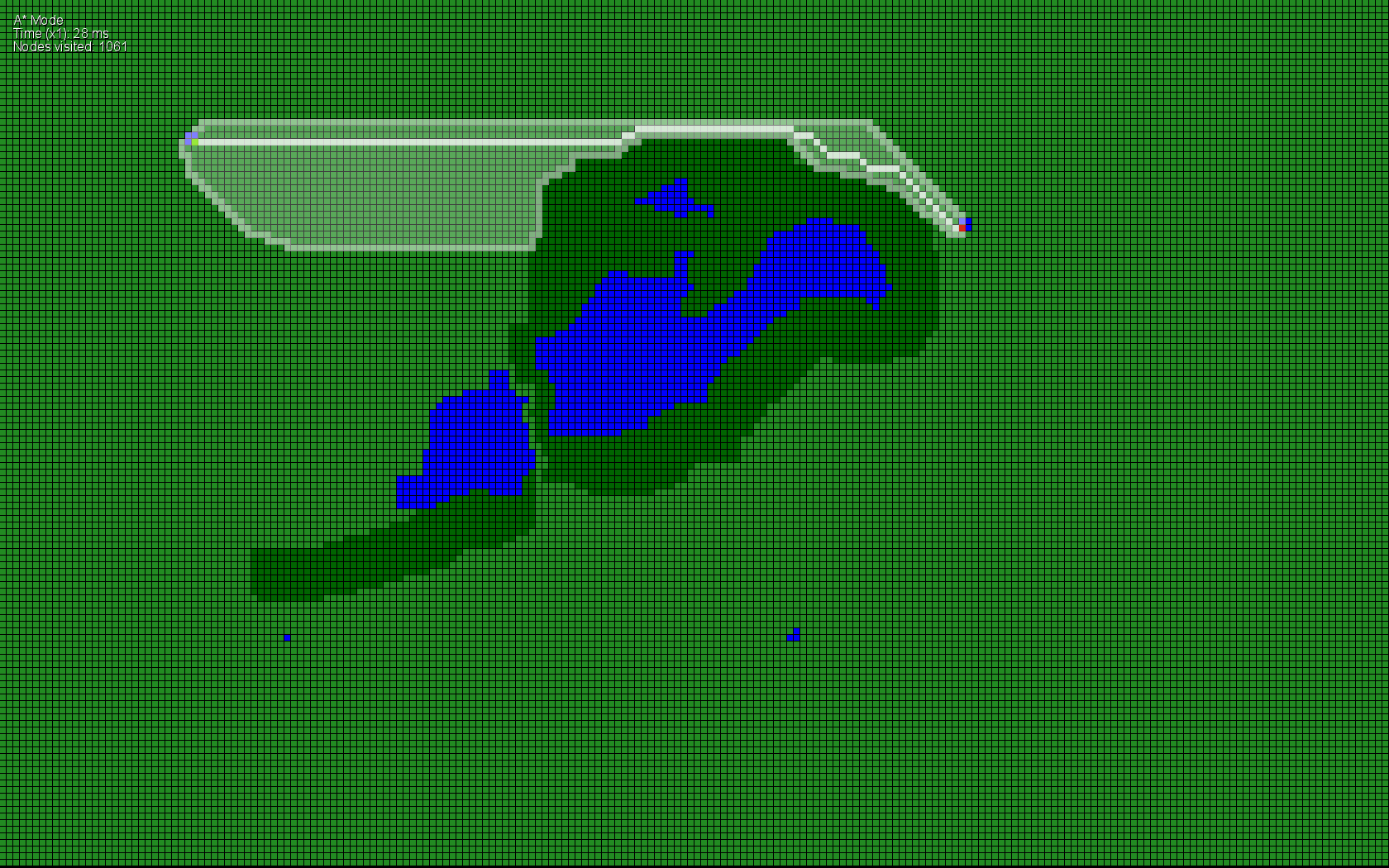
## Skenaario 2: Metsän ohi



Kuva 9 Djikstra: 1226 ms, 13282 solmua



Kuva 10 Djikstra optimoitu: 33 ms, 13314 solmua

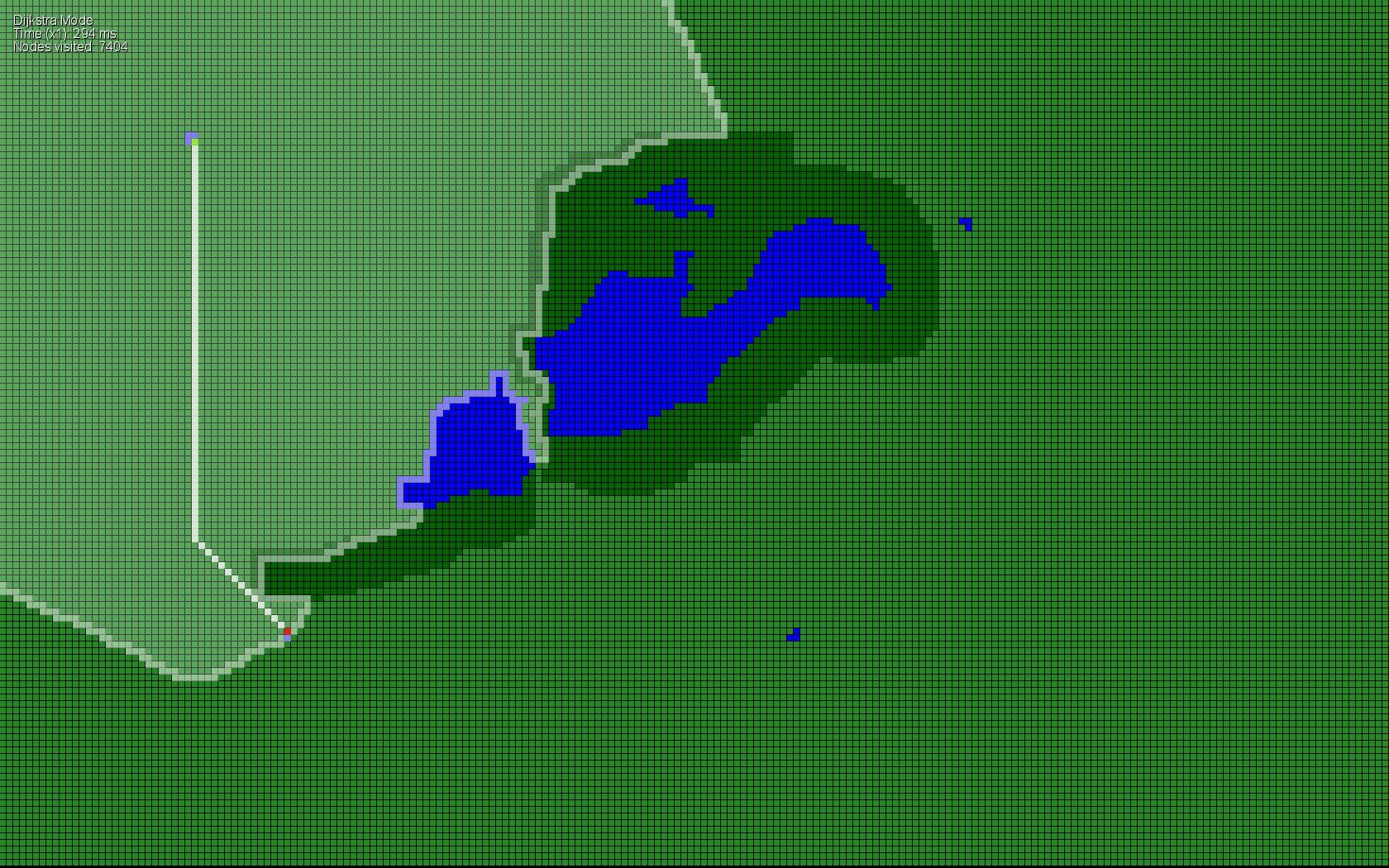


Kuva 11 A\*: 28 ms, 1061 solmua



Kuva 12 A\* optimoitu: 2 ms, 923 solmua

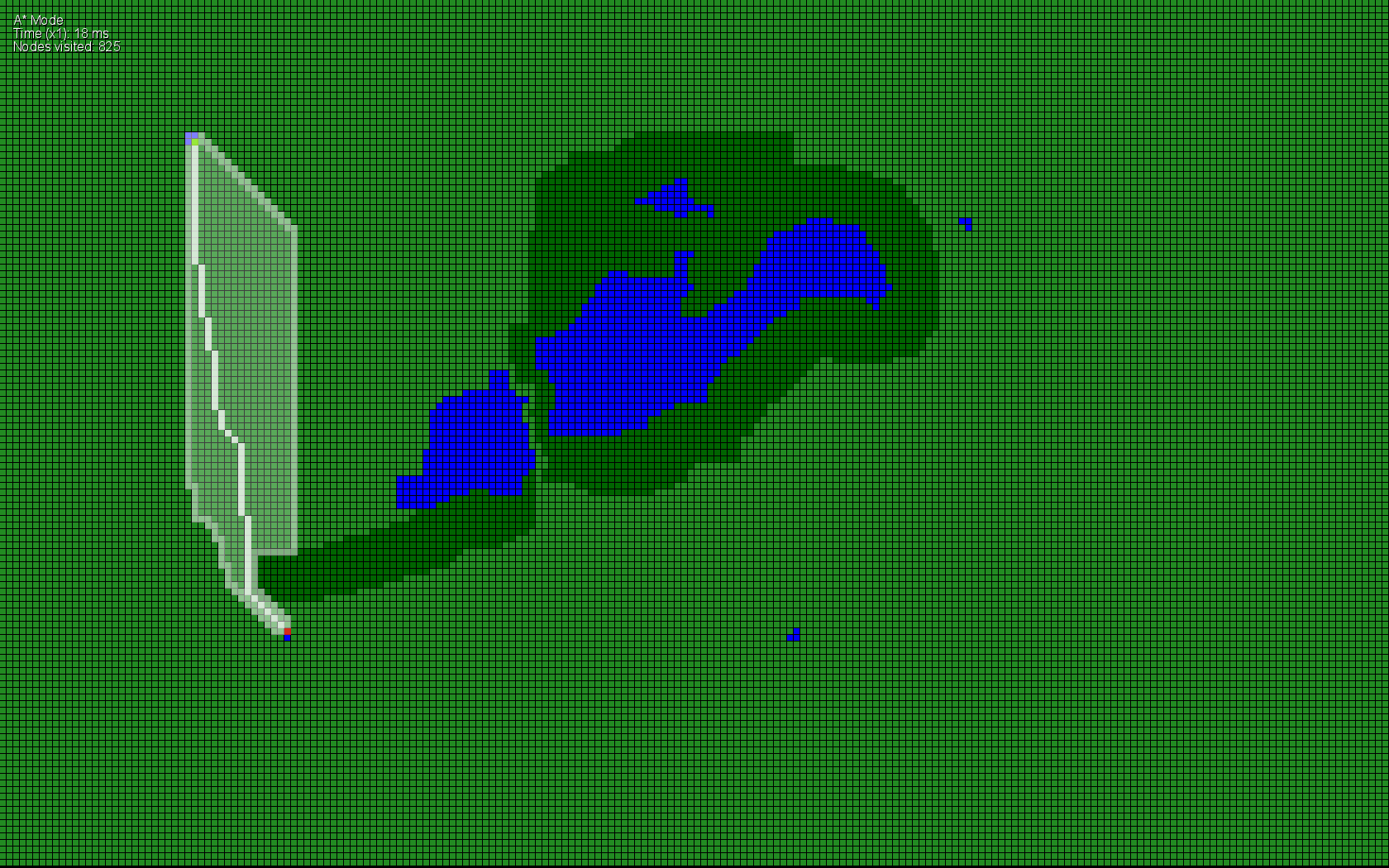
## Skenaario 3: Etelään



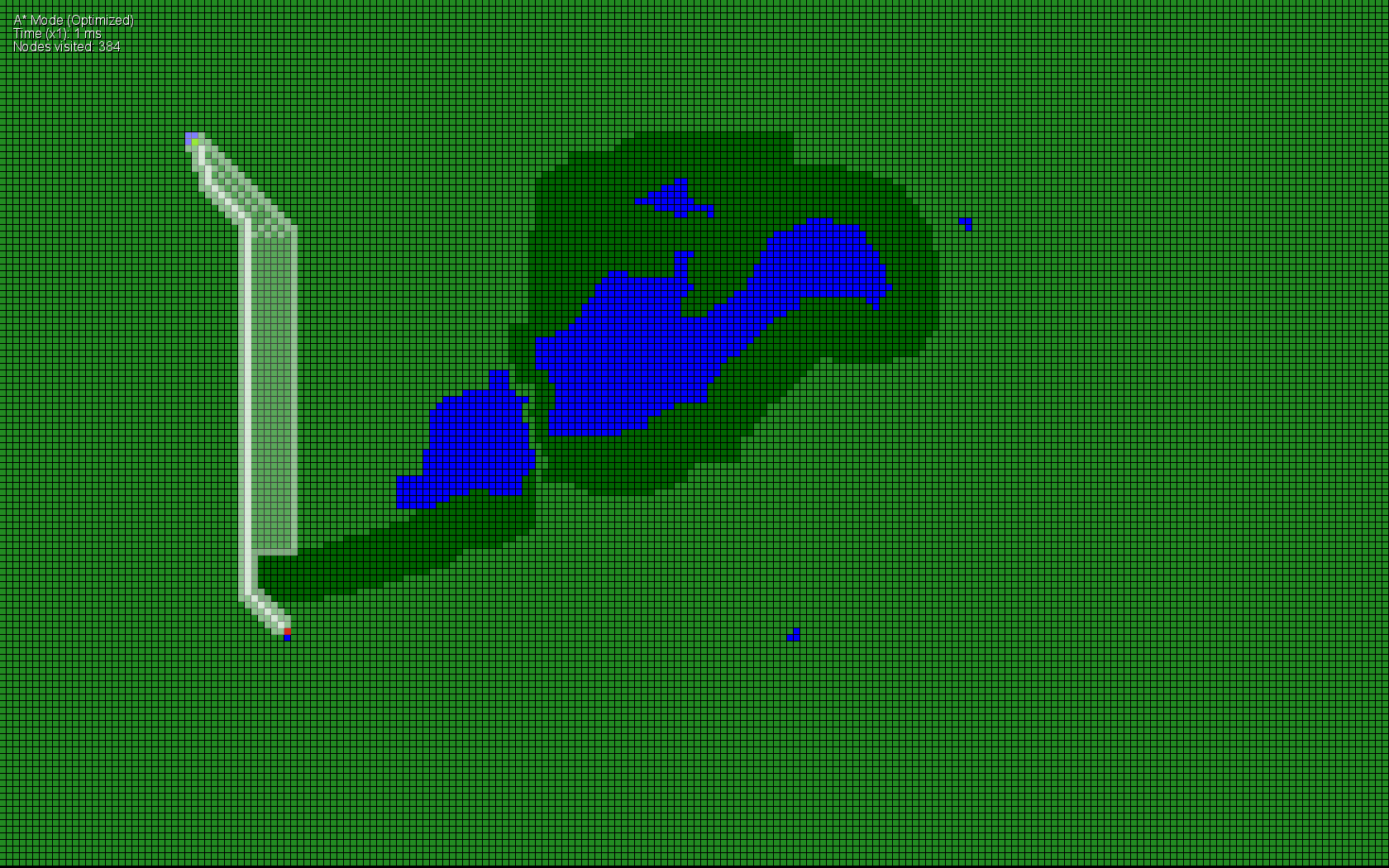
Kuva 13 Dijkstra: 294 ms, 7404 solmua



Kuva 14 Dijkstra optimoitu: 13 ms, 7416 solmua

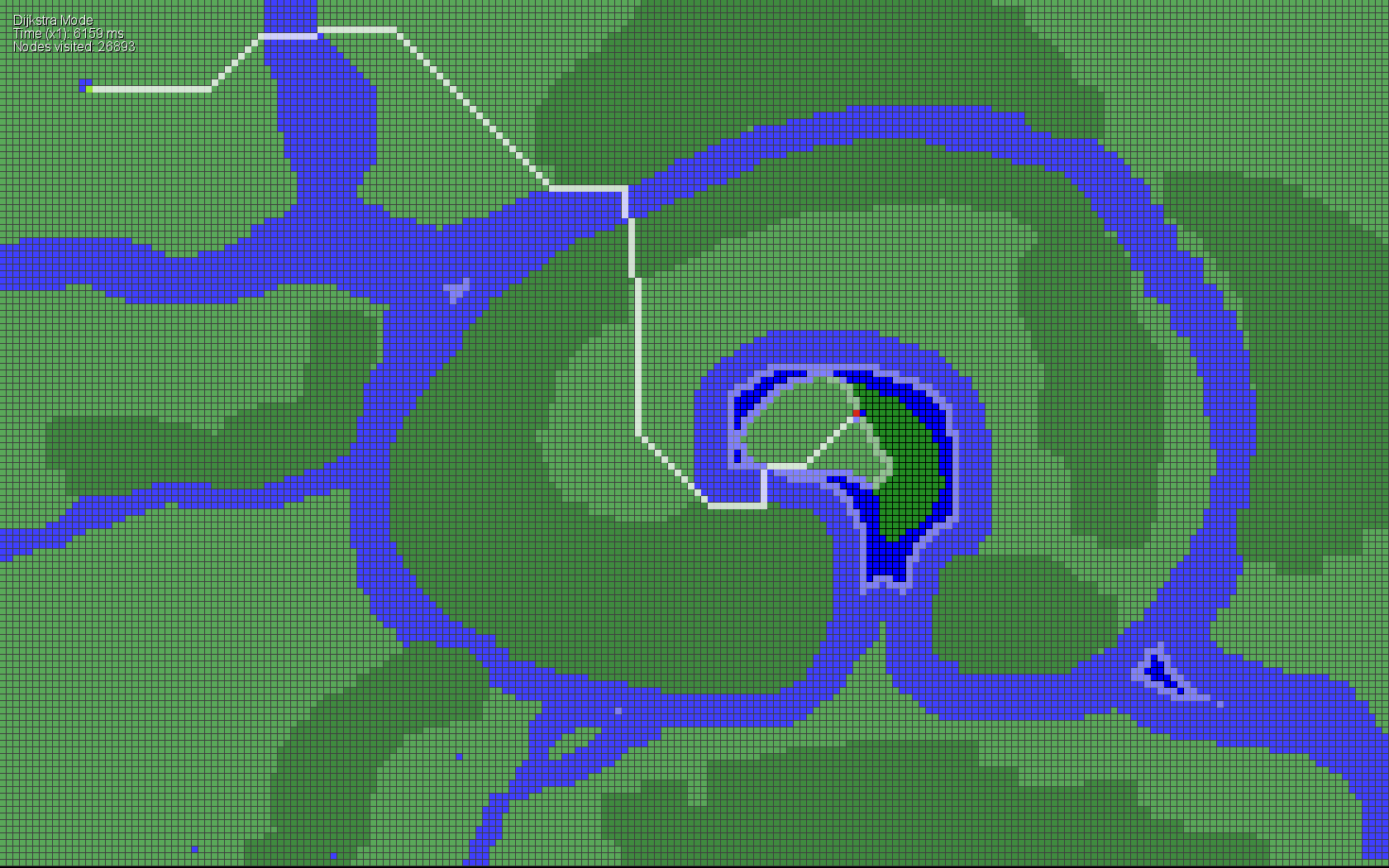


Kuva 15 A\*: 18 ms, 825 solmua

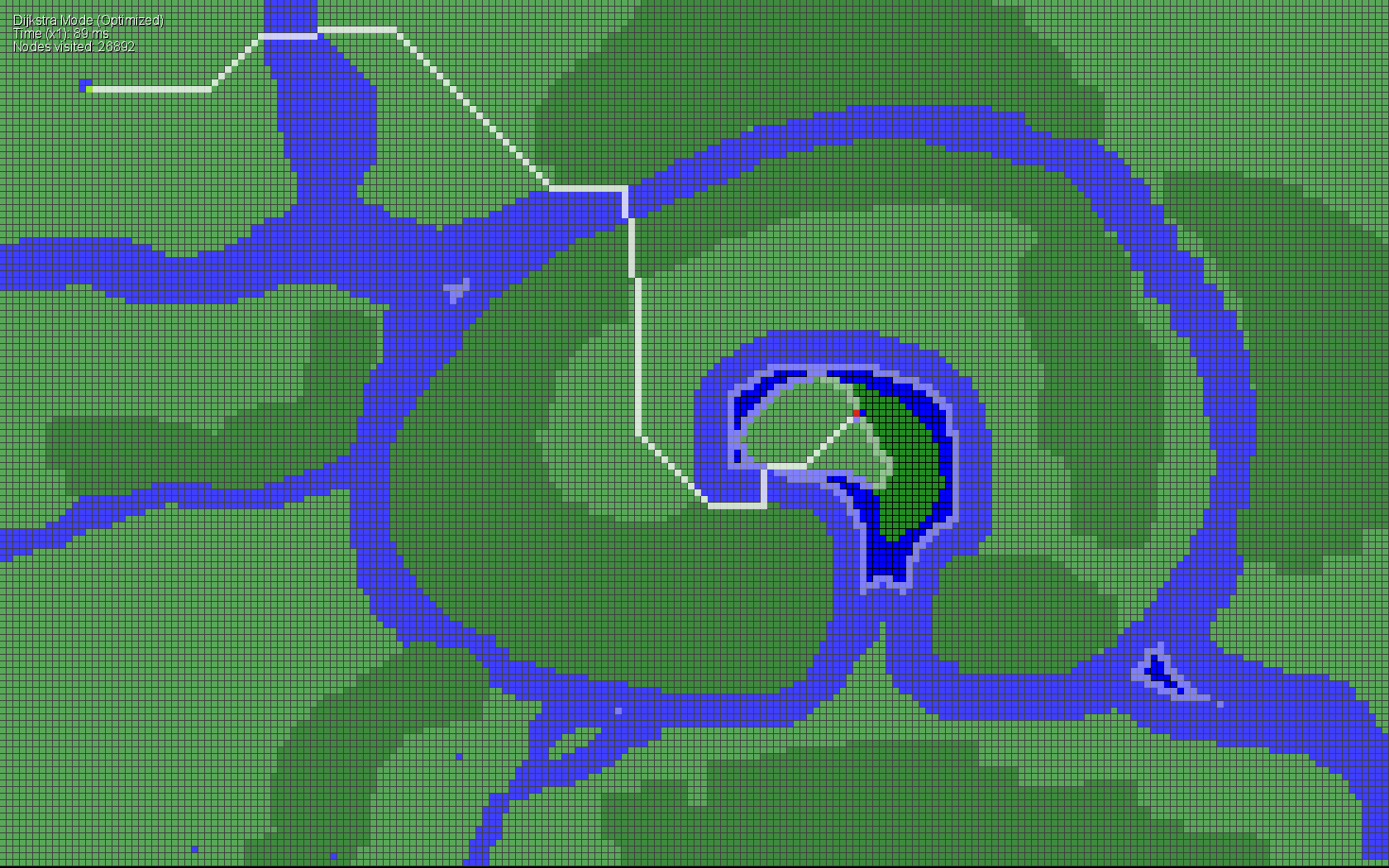


Kuva 16 A\* optimoitu: 1ms, 384 solmua

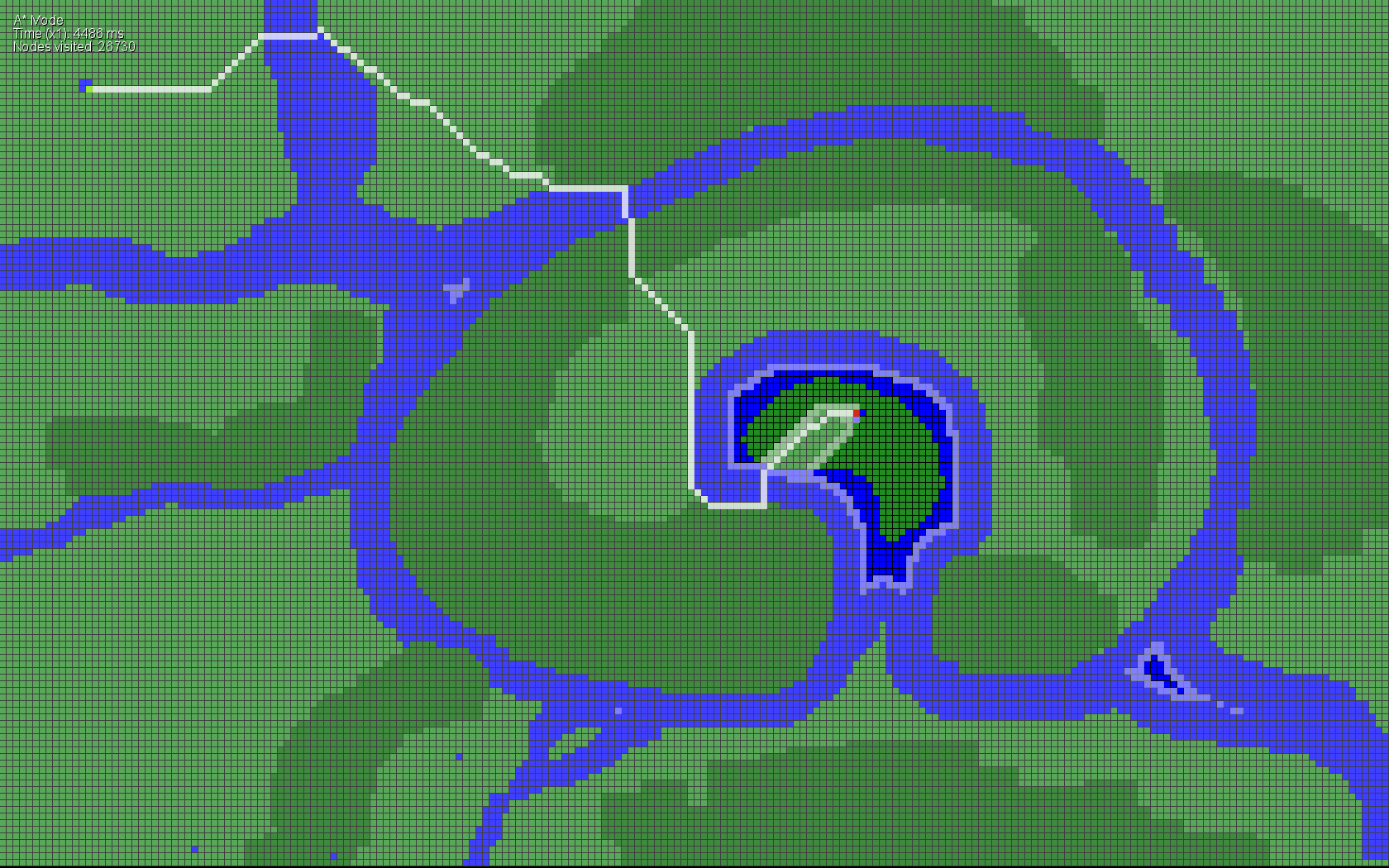
## Skenaario 4: Saarelle



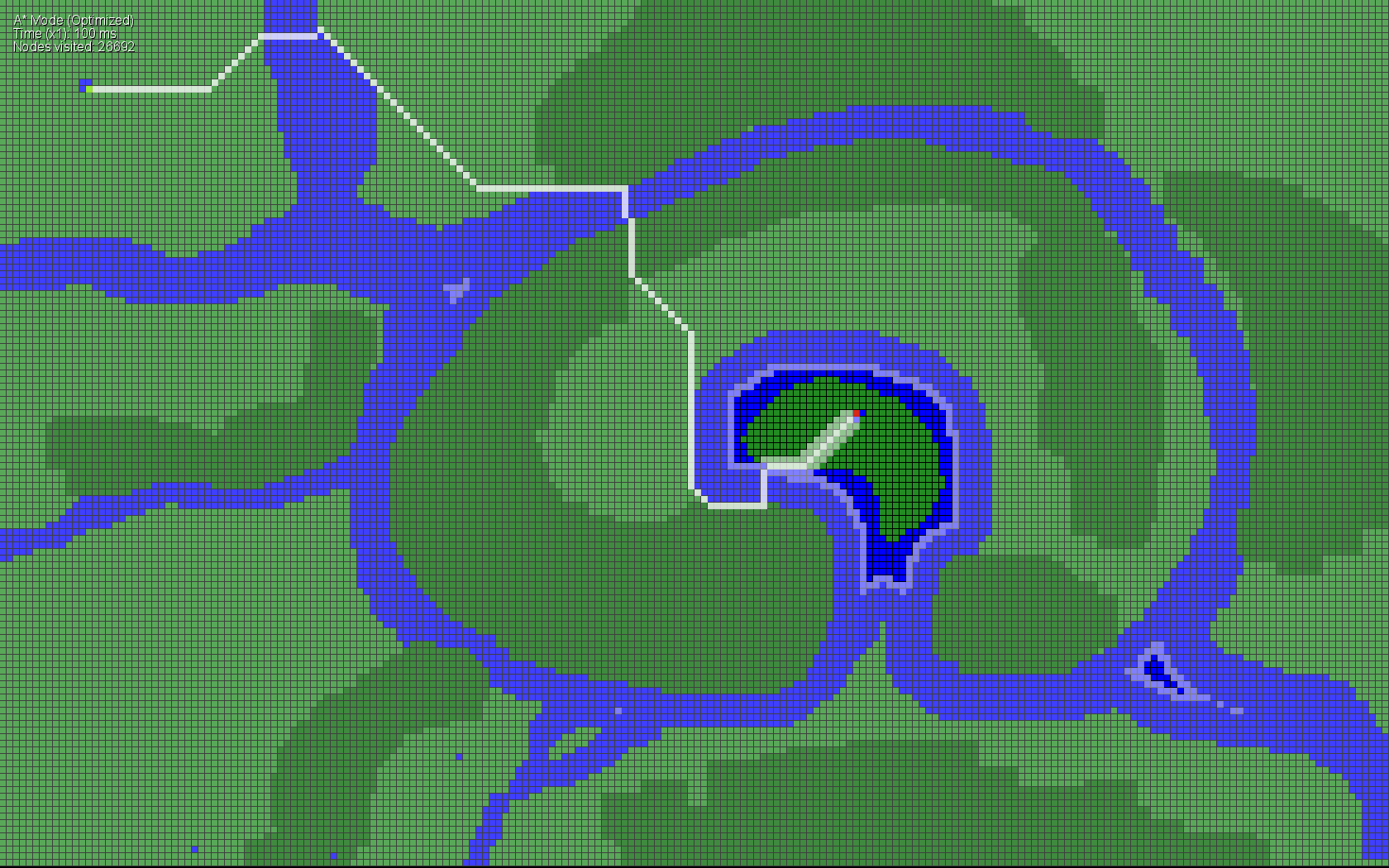
Kuva 17 Dijkstra: 6159 ms, 26893 solmua



Kuva 18 Dijkstra optimoitu: 89 ms, 26892 solmua

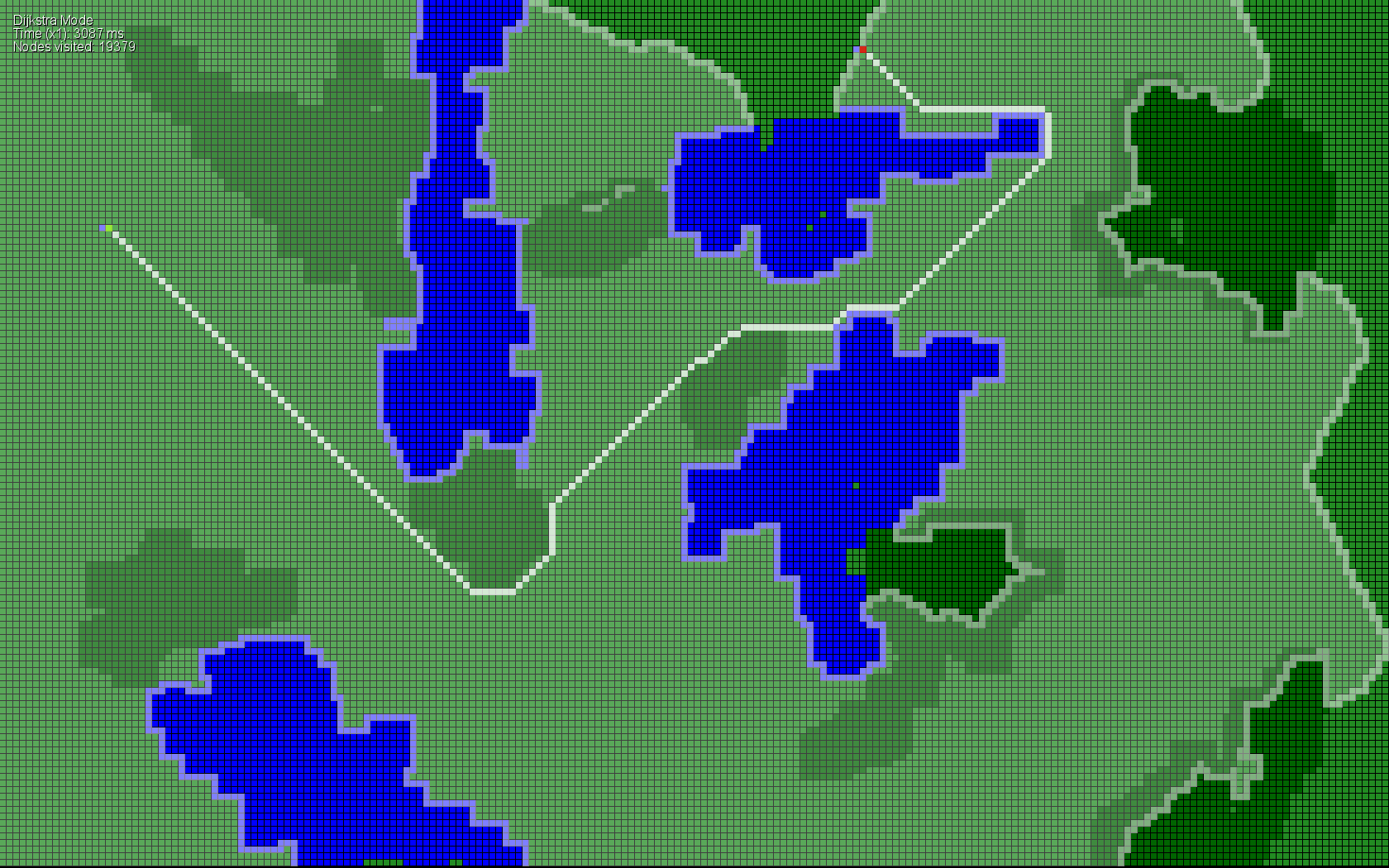


Kuva 19 A\*: 4486 ms, 26730 solmua

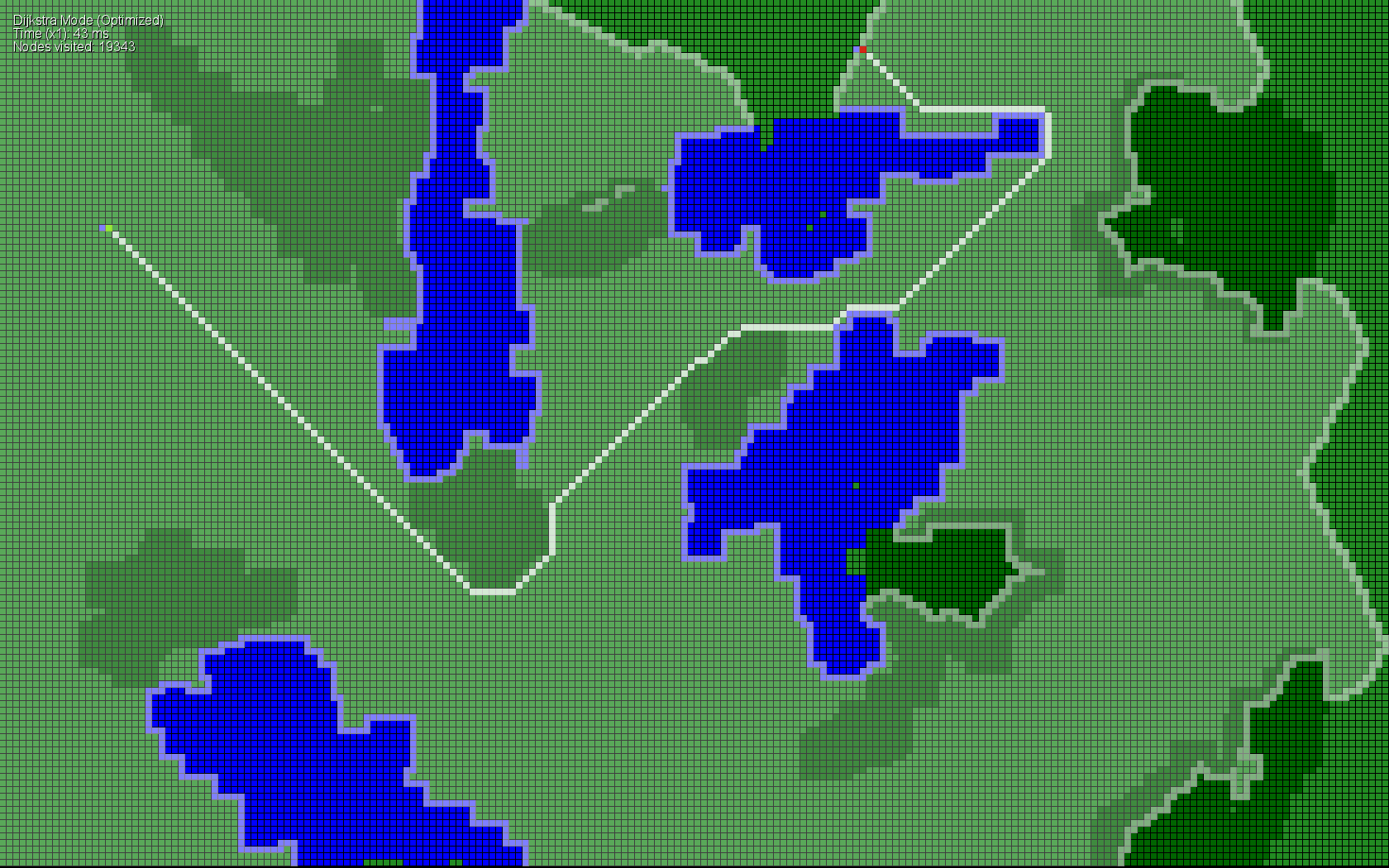


Kuva 20 A\* optimoitu: 100 ms, 26692 solmua

## Skenaario 5: Järven ympäri



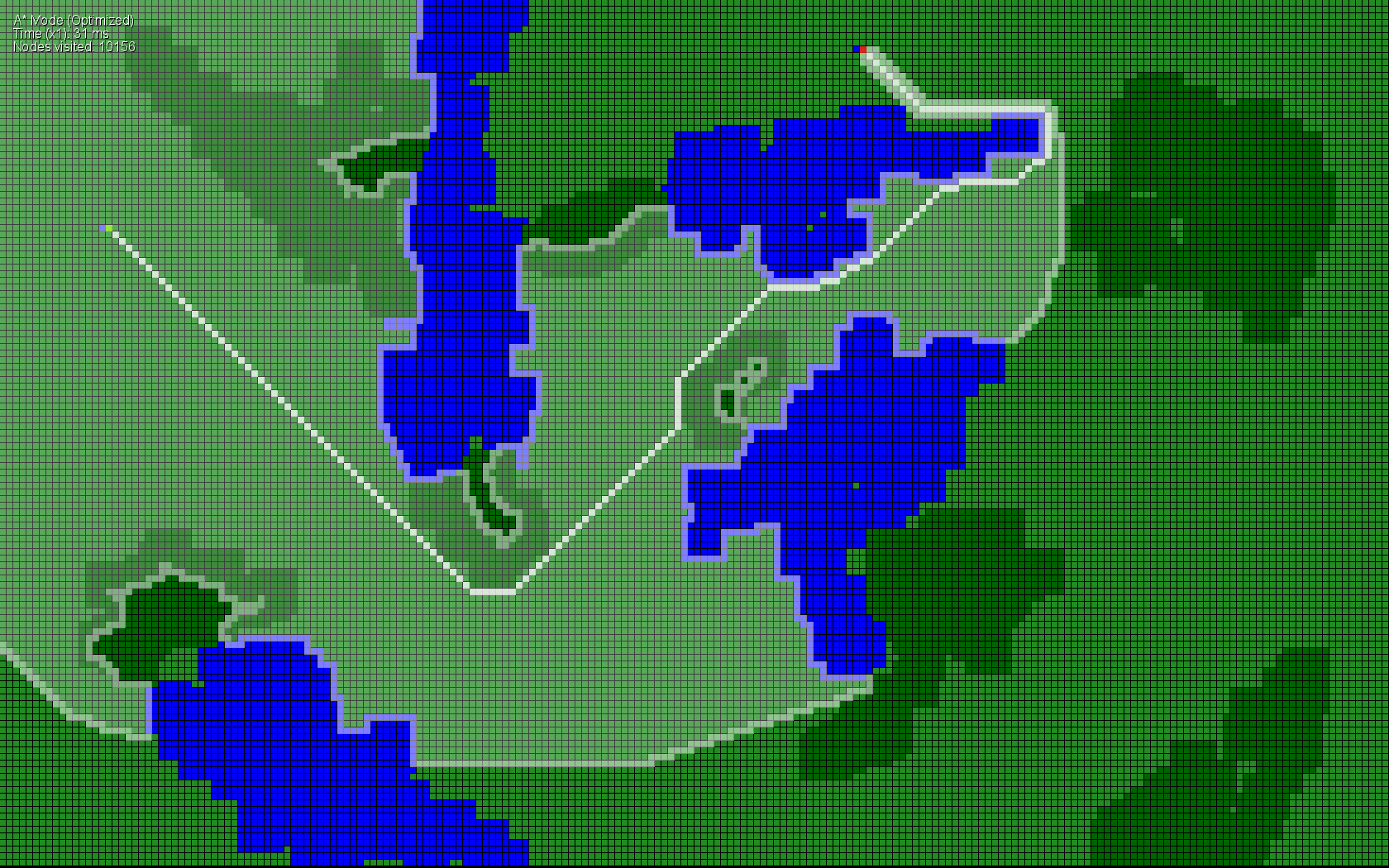
Kuva 21 Dijkstra: 3087 ms, 19379 solmua



Kuva 22 Dijkstra optimoitu: 43 ms, 19343 solmua

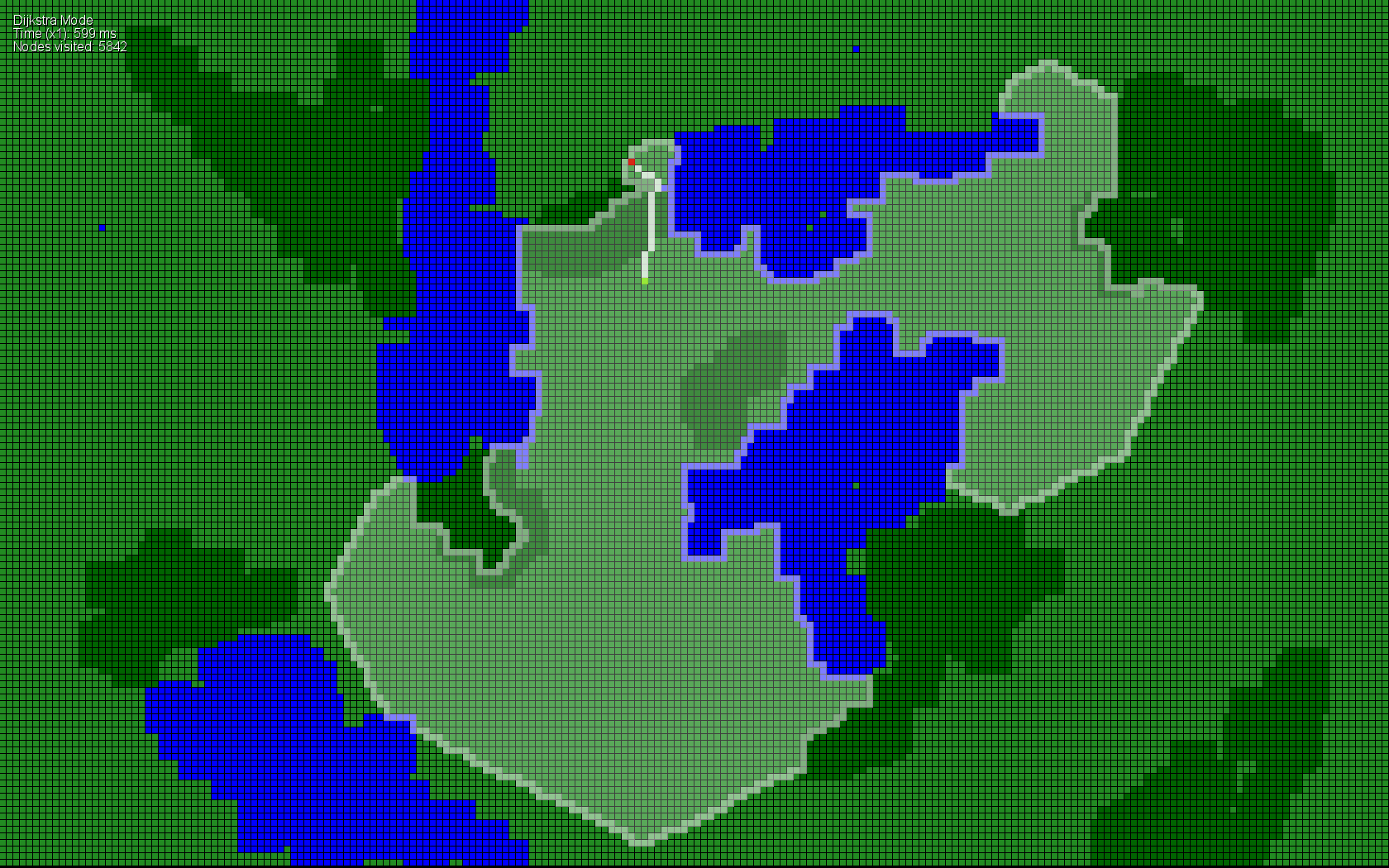


Kuva 23 A\*: 1132 ms, 10246 solmua

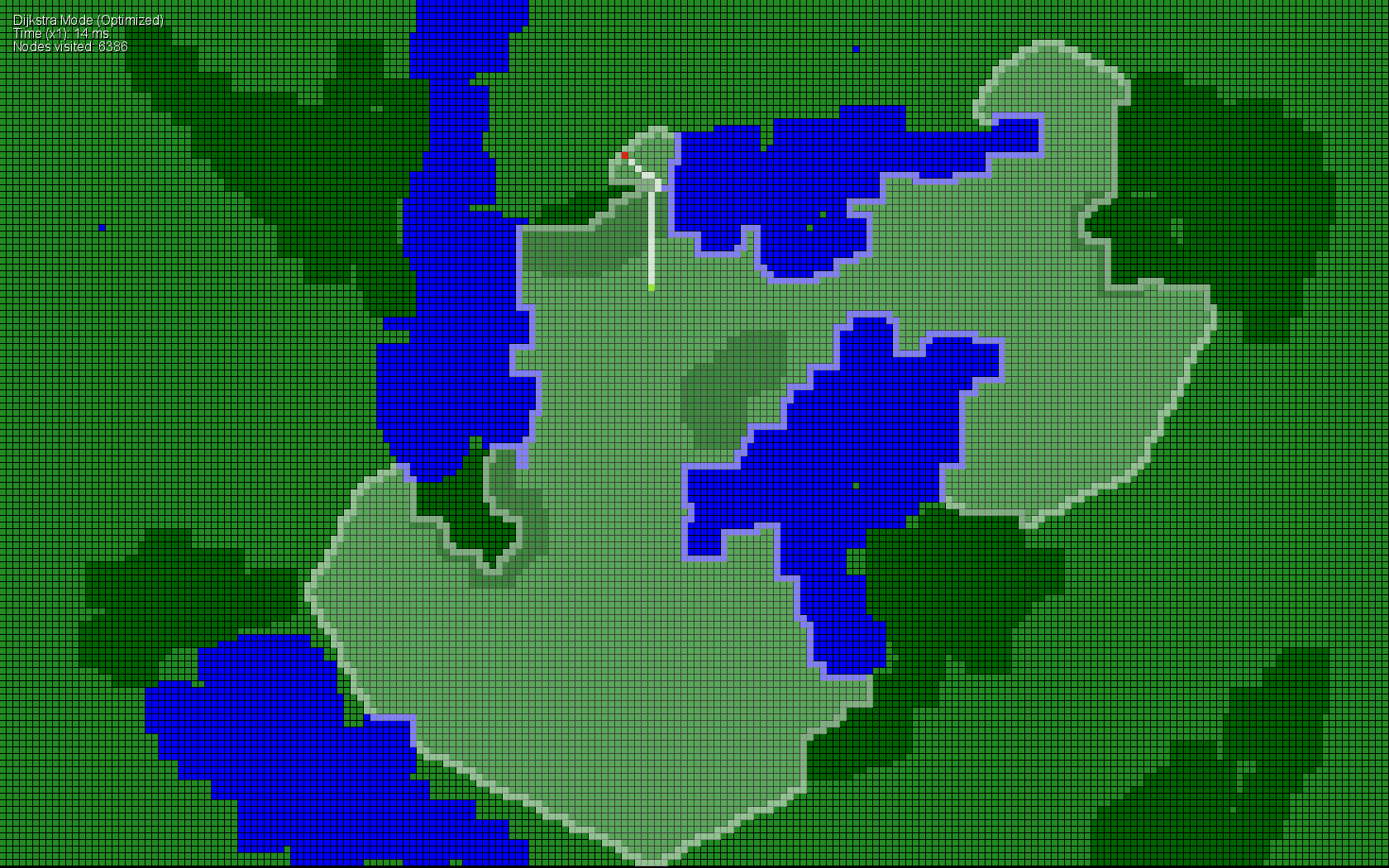


Kuva 24 A\* optimoitu: 31 ms, 10156 solmua

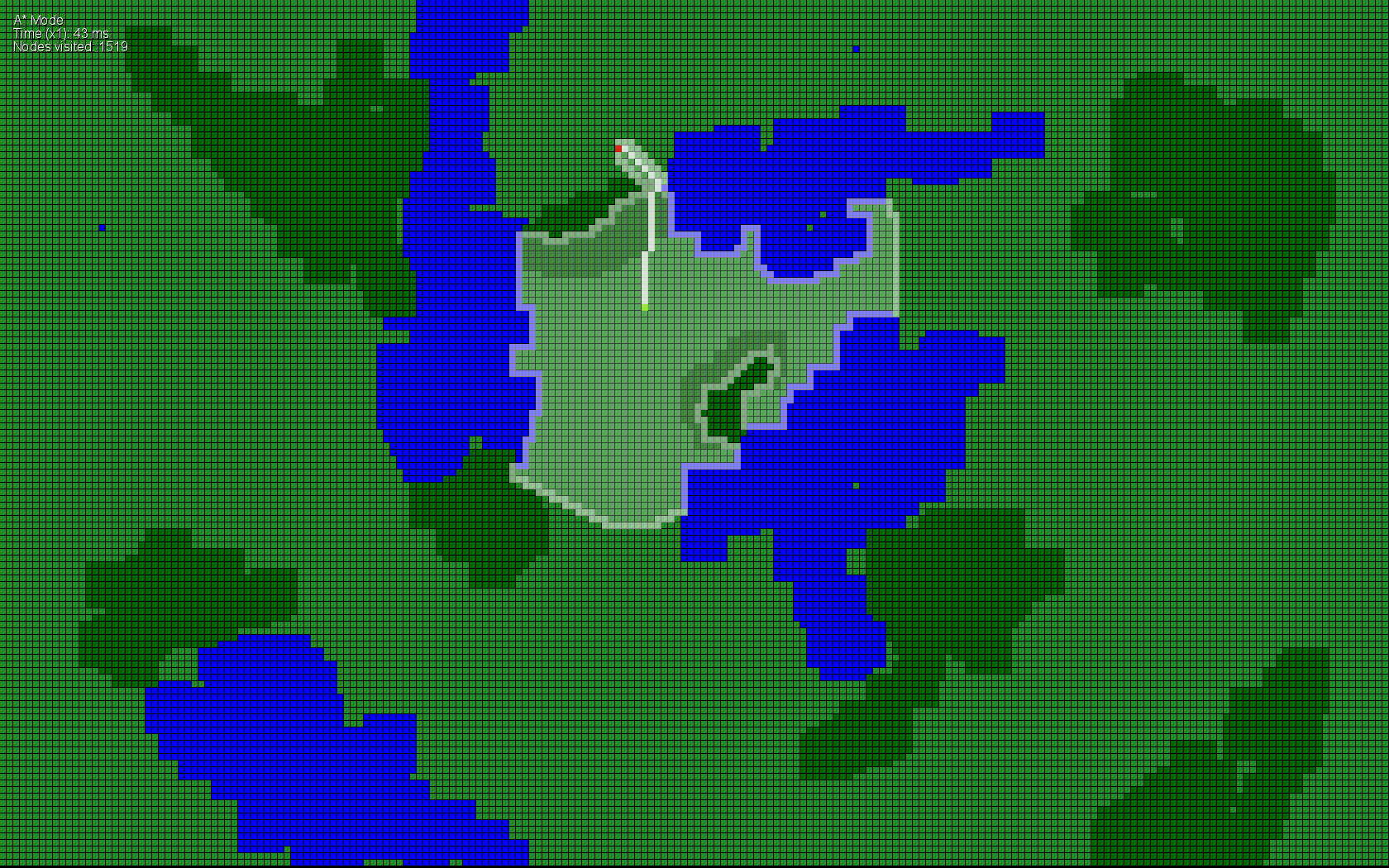
## Skenaario 6: Metsän läpi



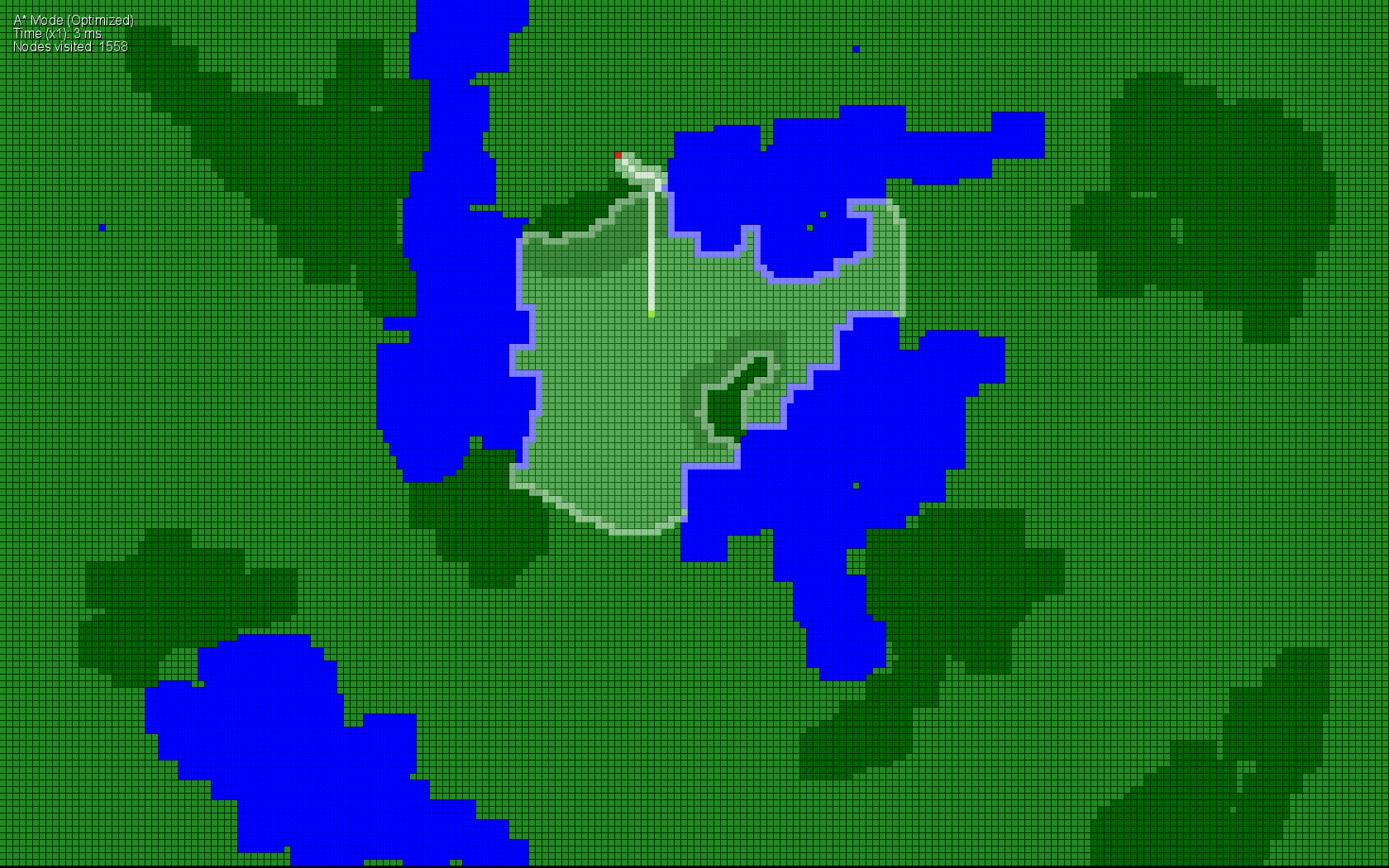
Kuva 25 Dijkstra: 599 ms, 5842 solmua



Kuva 26 Dijkstra optimoitu: 14 ms, 6386 solmua



Kuva 27 A\*: 43 ms, 1519 solmua



Kuva 28 A\* optimoitu: 3 ms, 1558 solmua

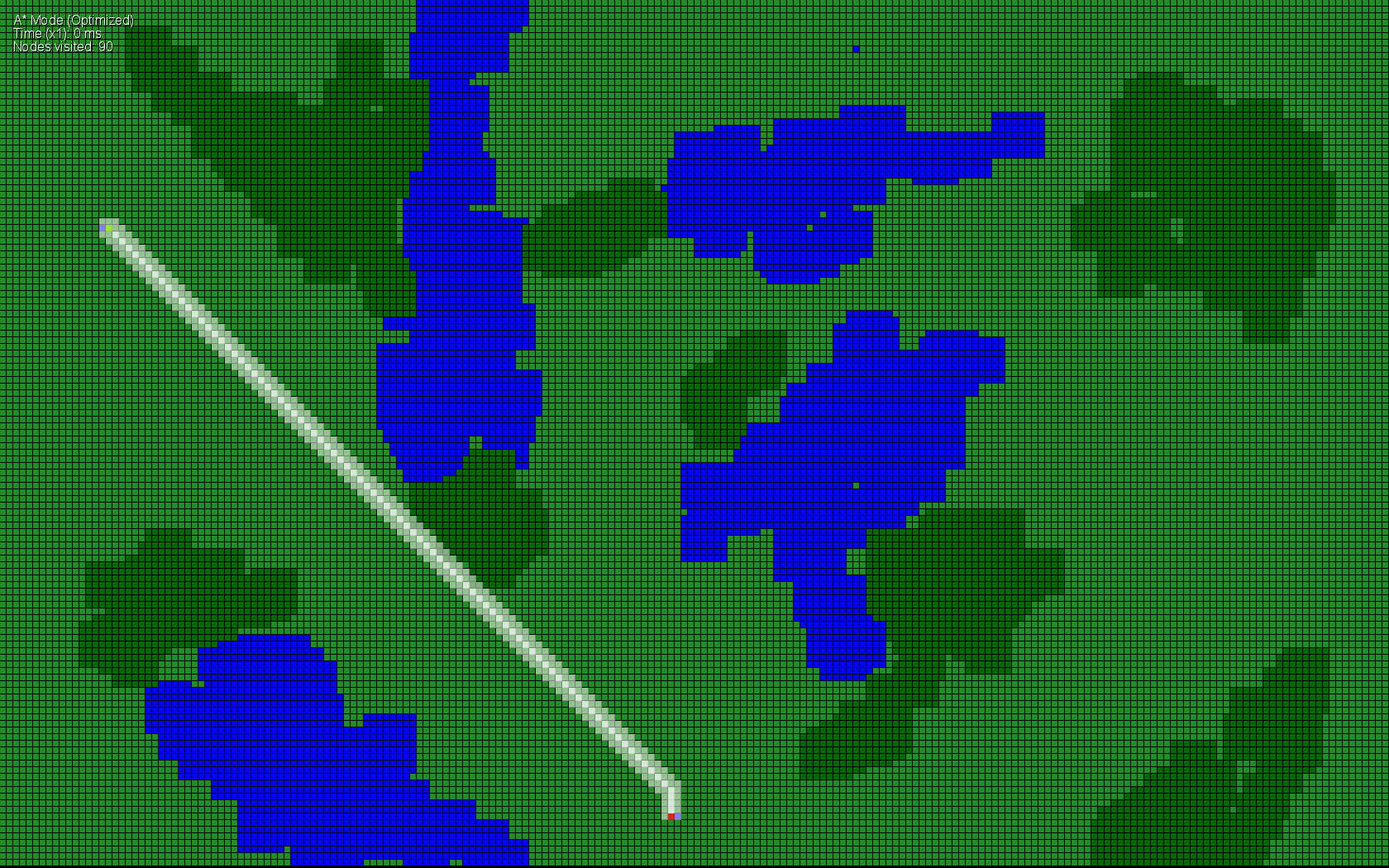
## Skenaario 7: Suora reitti esteiden ohi



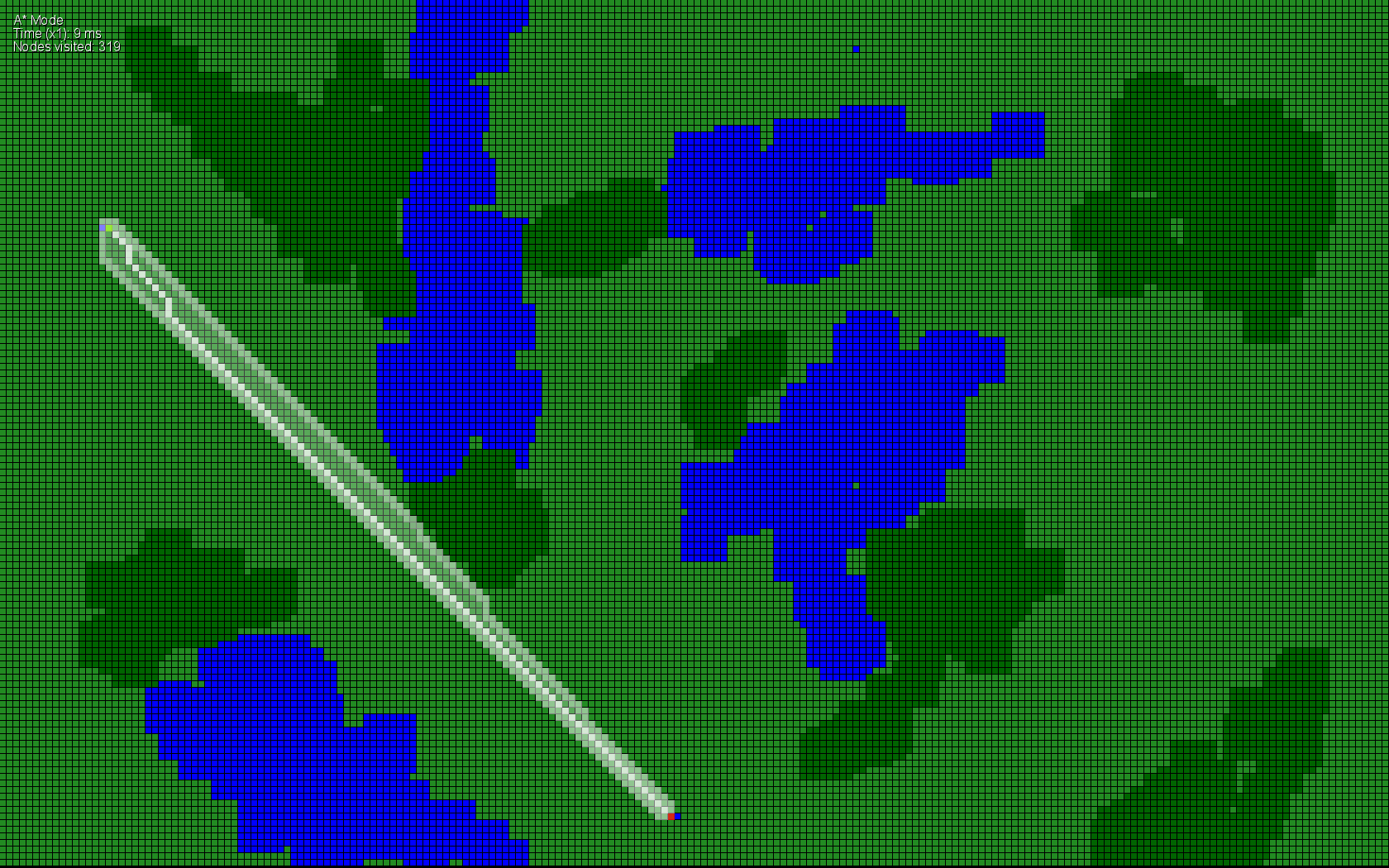
Kuva 29 Dijkstra: 703 ms, 9686 solmua



Kuva 30 Dijkstra optimoitu: 18 ms, 9681 solmua

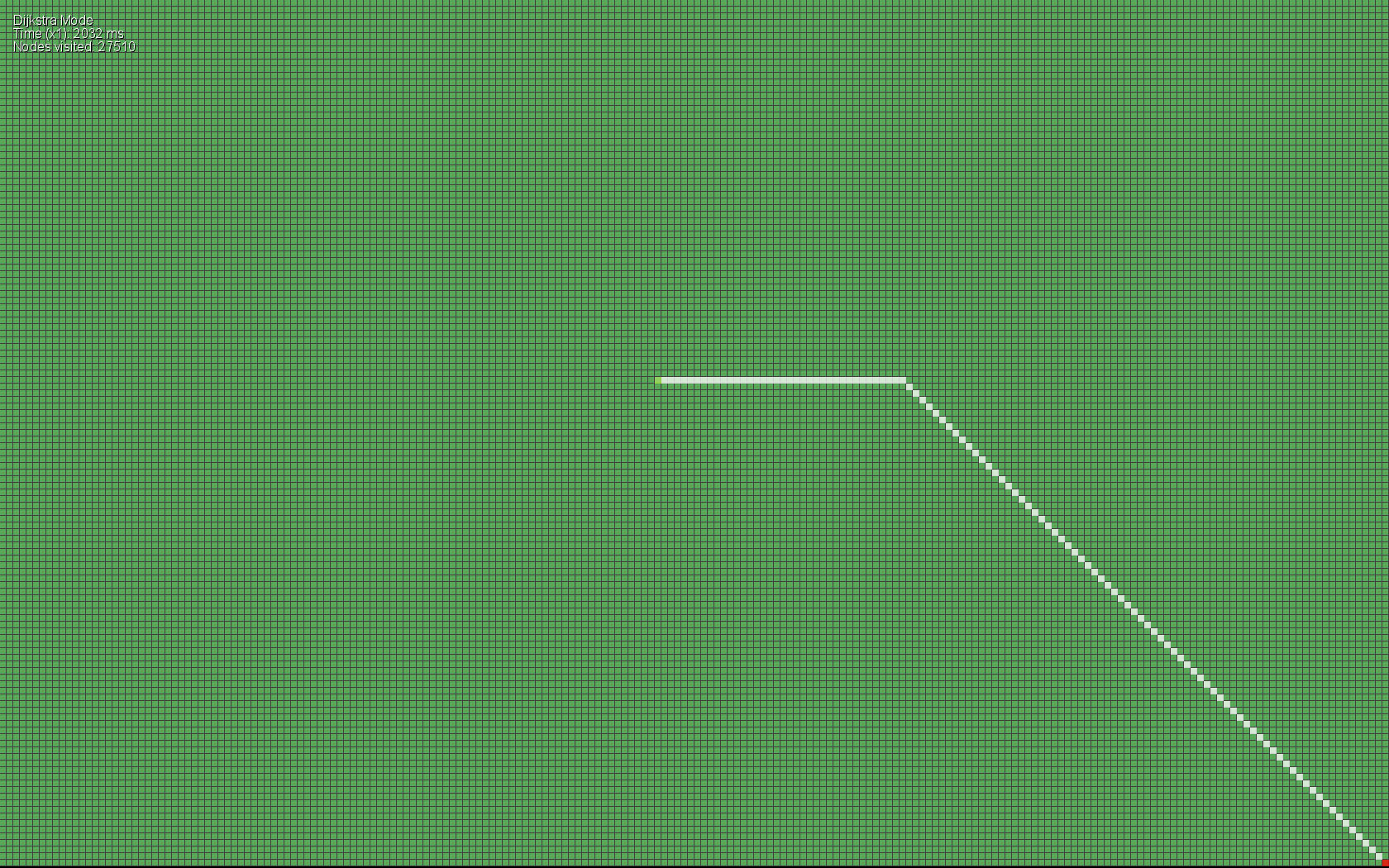


Kuva 31 A\*: 9 ms, 319 solmua

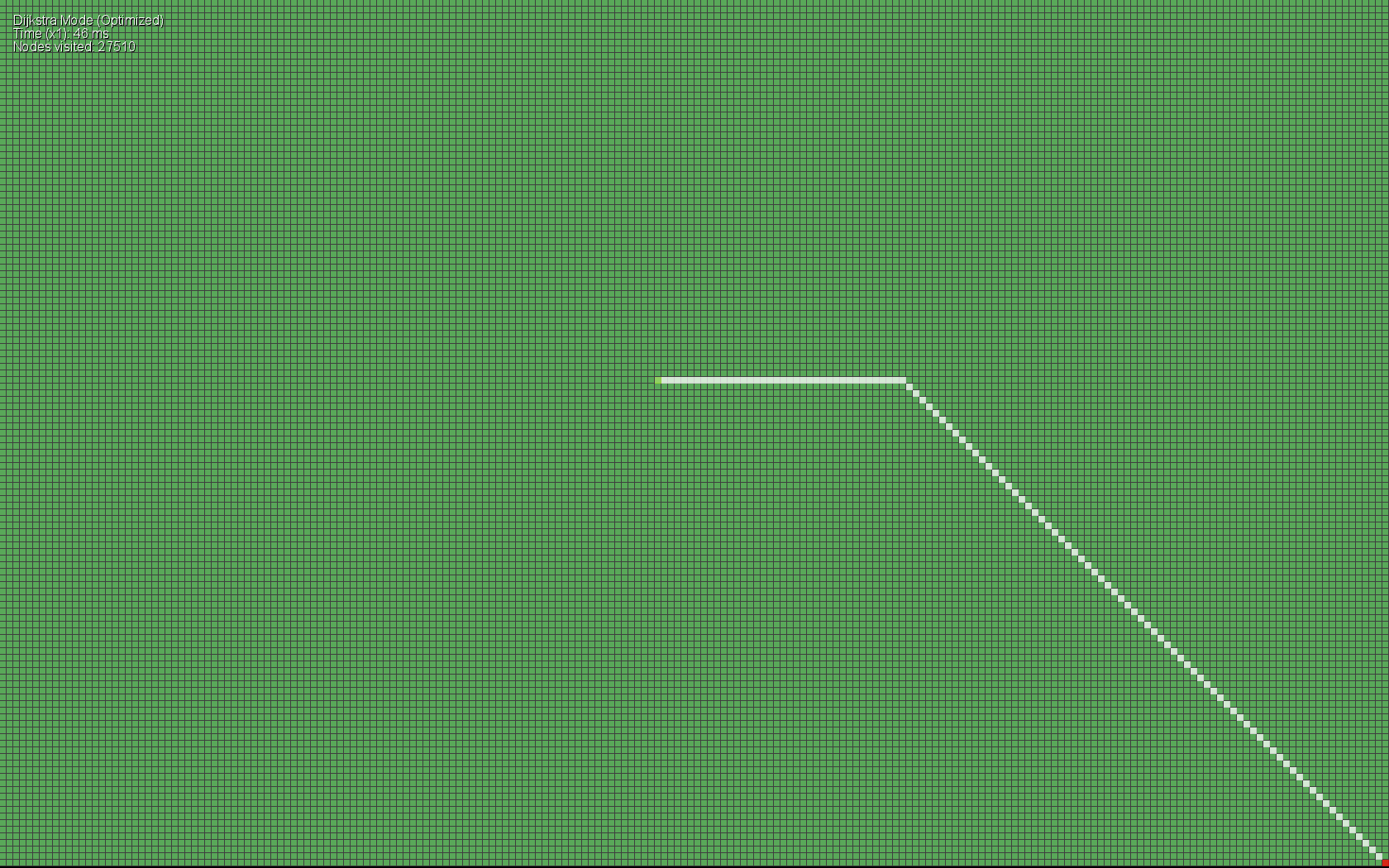


Kuva 32 A\* optimoitu: 0 ms, 90 solmua

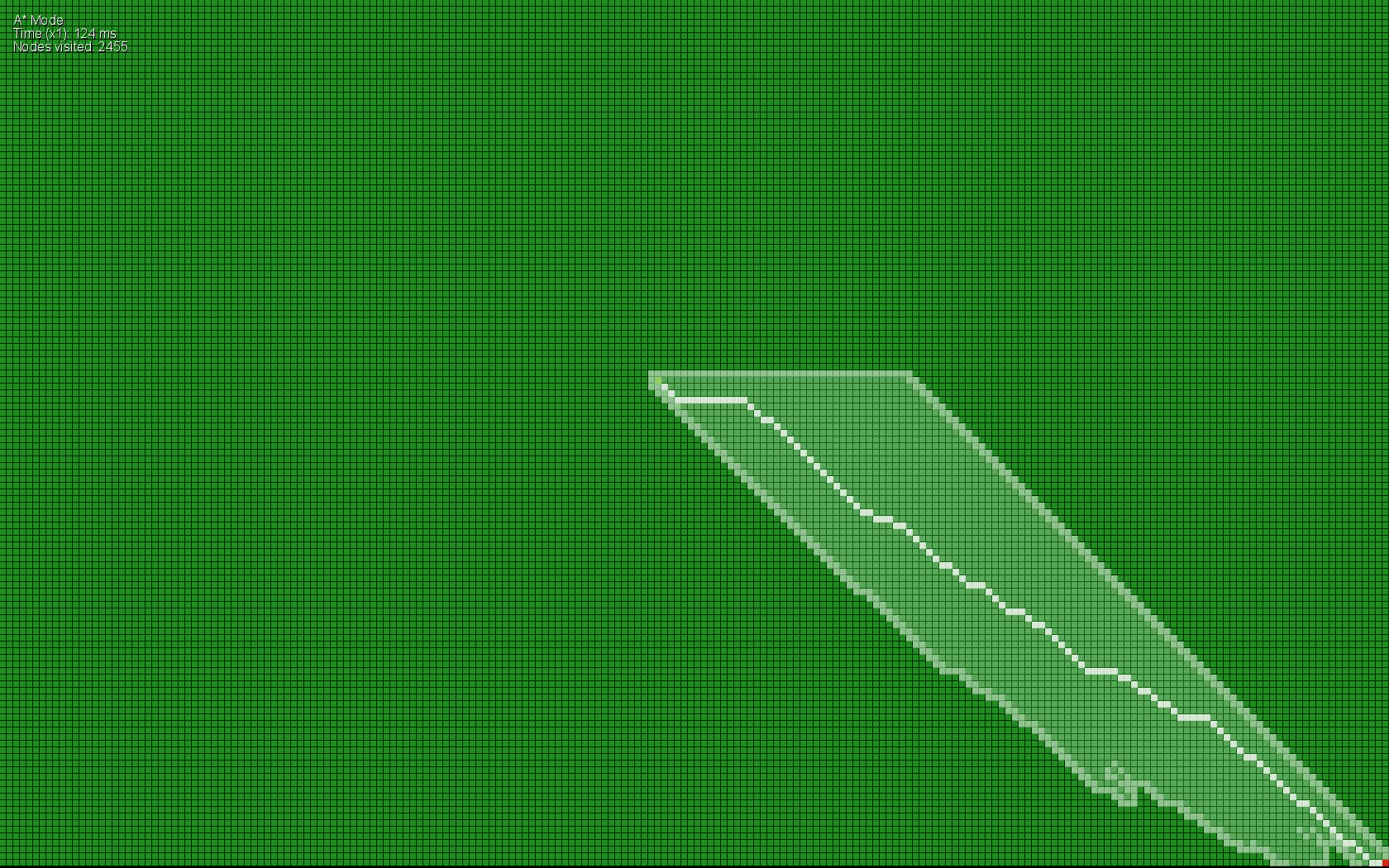
## Skenaario 8: Tyhjä tasanko



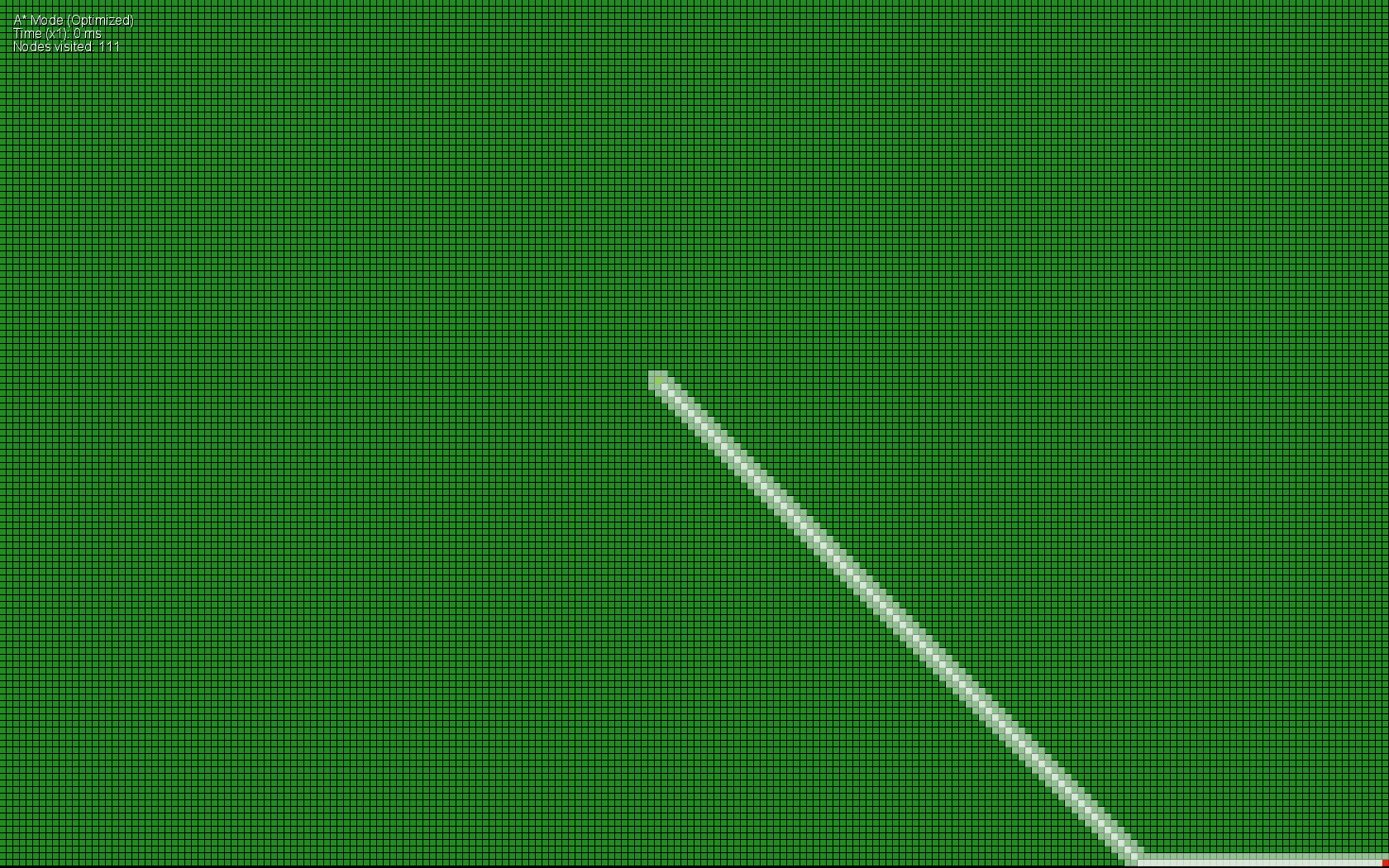
Kuva 33 Dijkstra: 2032 ms, 27510 solmua



Kuva 34 Dijkstra optimoitu: 46 ms, 27510 solmua

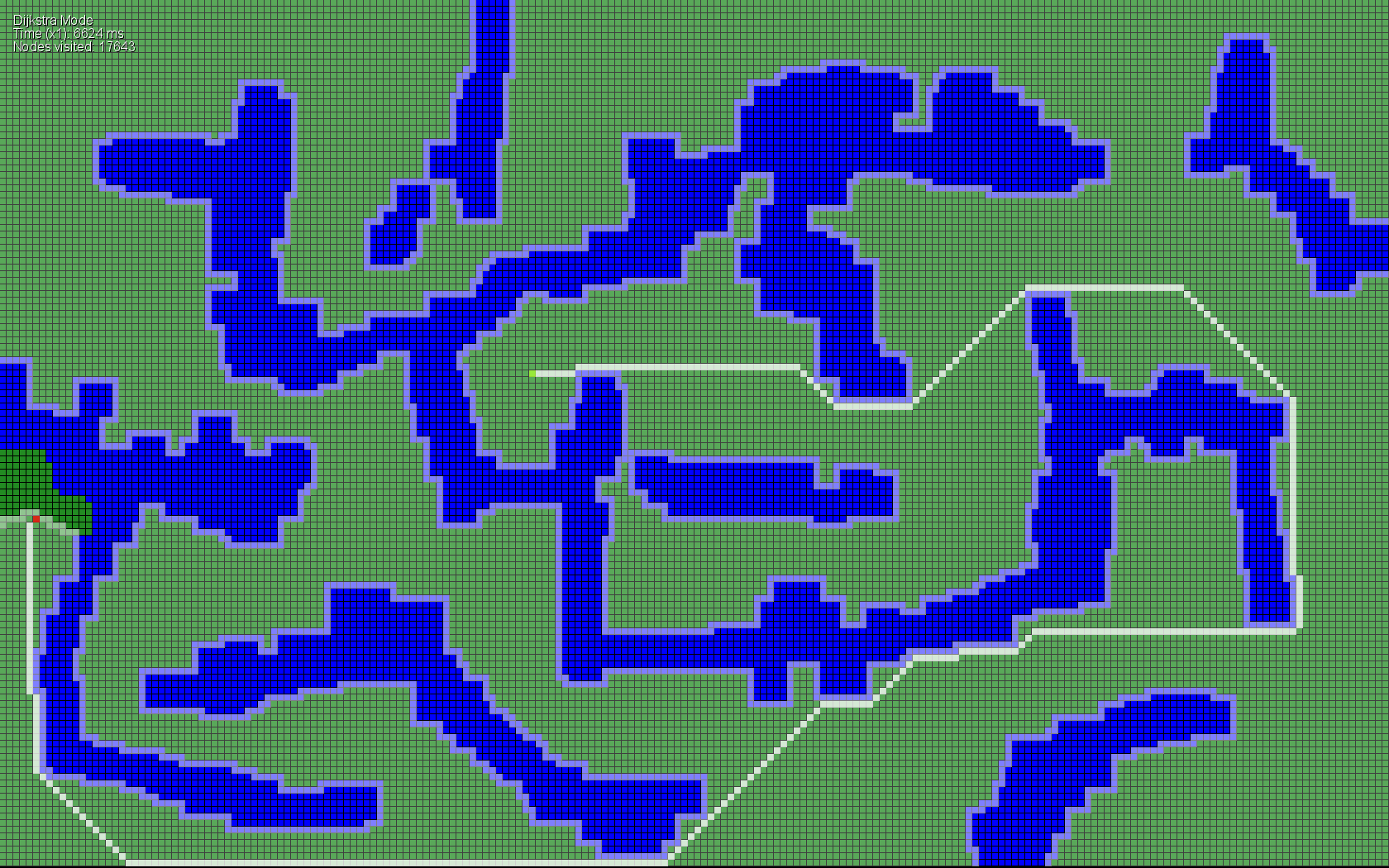


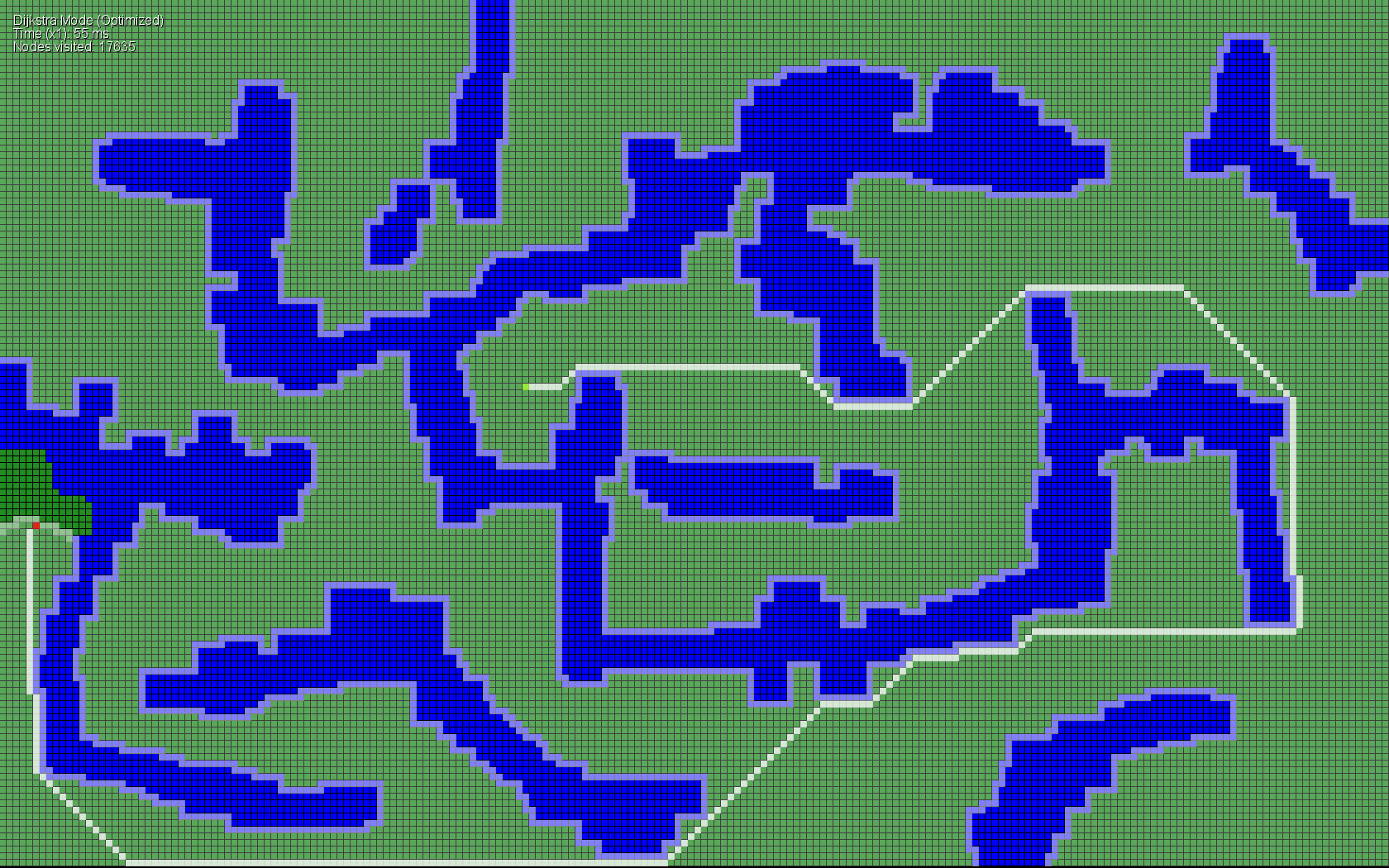
Kuva 35 A\*: 124 ms, 2455 solmua



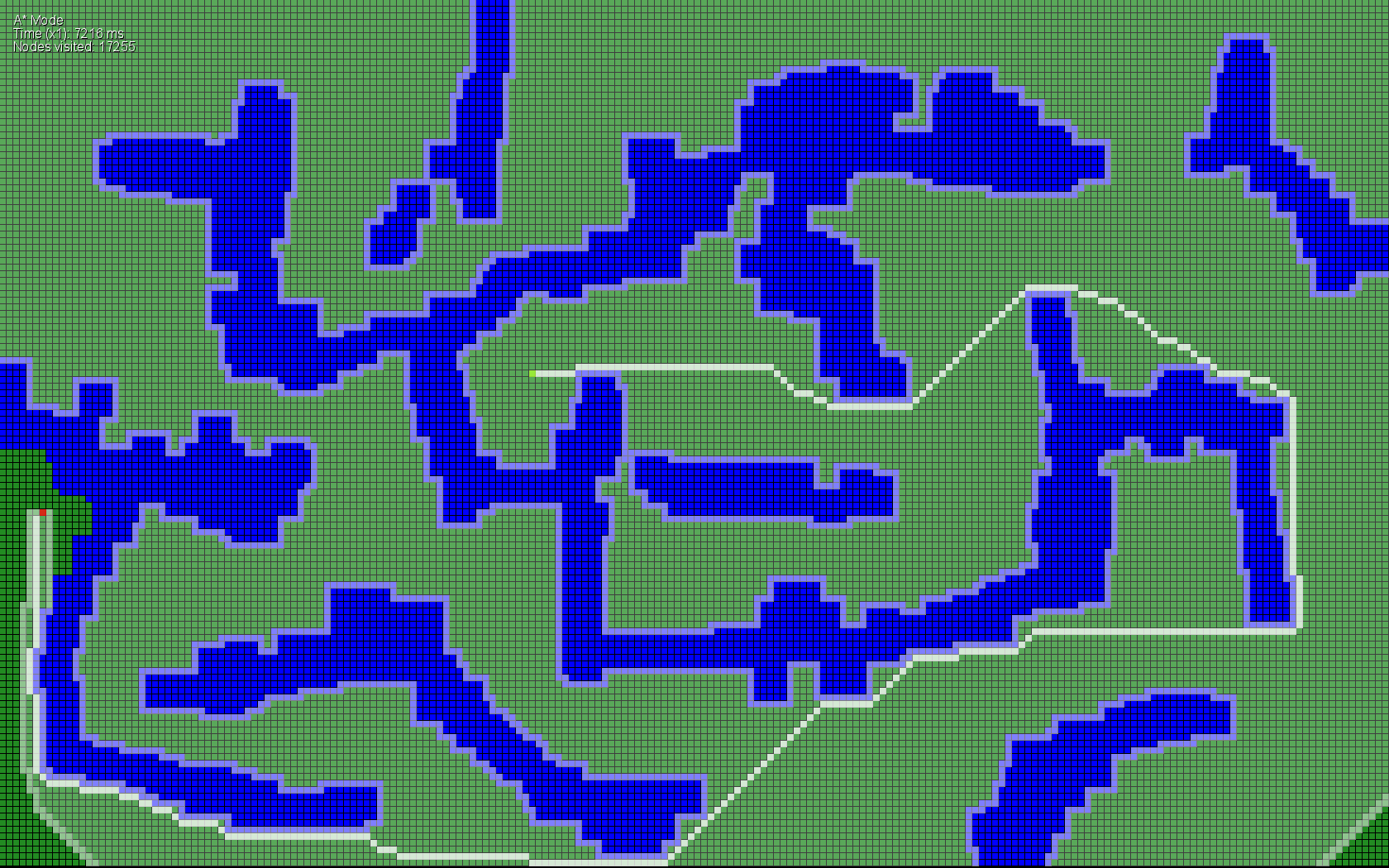
Kuva 36 A\* optimoitu: 0 ms, 111 solmua

## Skenaario 9: Sokkelo

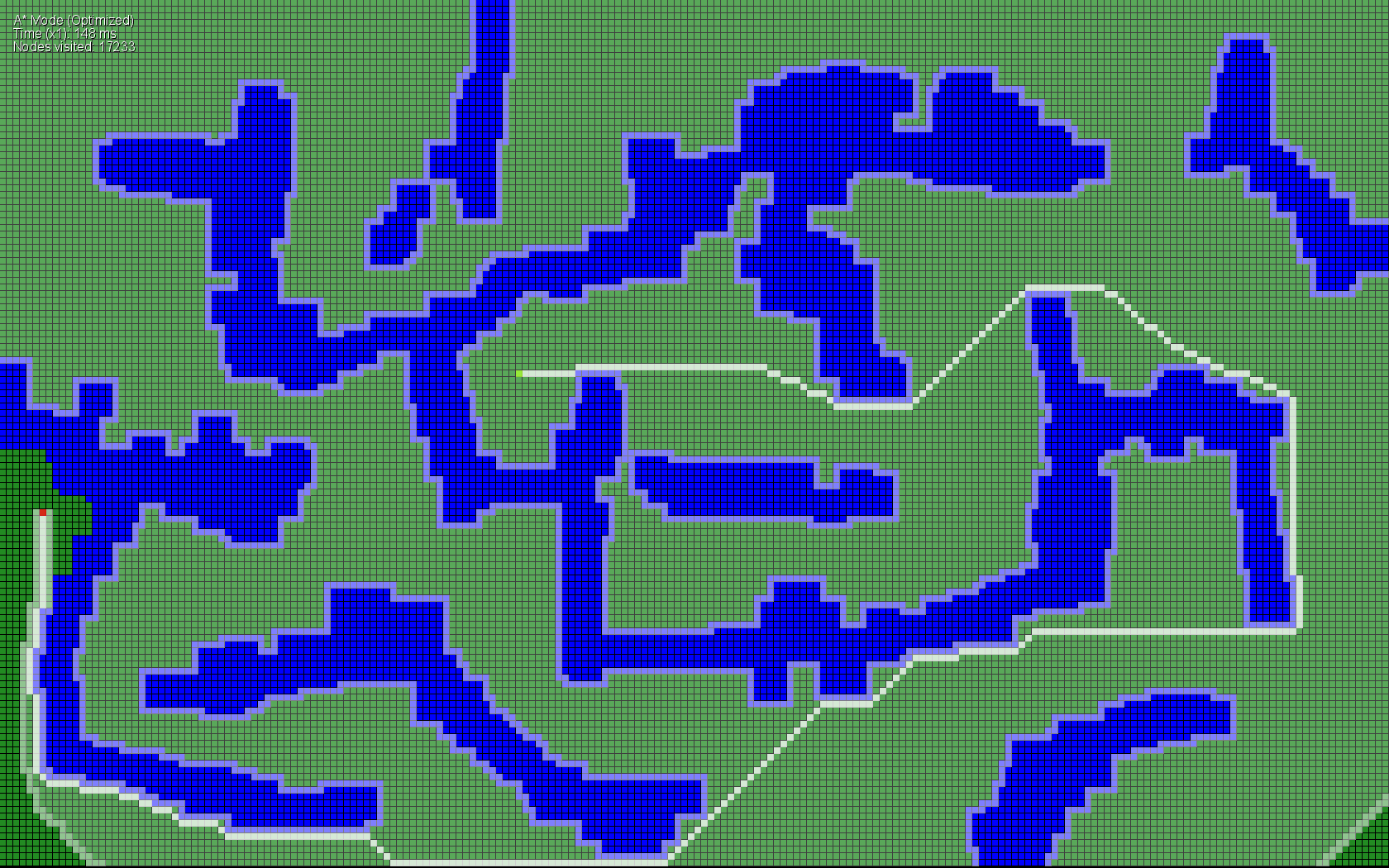


Kuva 37 Dijkstra: 6624 ms, 17643 solmua

Kuva 38 Dijkstra optimoitu: 55 ms, 17635 solmua



Kuva 39 A\*: 7216 ms, 17255 solmua



Kuva 40 A\* optimoitu: 148 ms, 17233 solmua

# Yhteenveto

Algoritmien tehokkuus riippuu vahvasti ympäröivästä maastosta. A\* pärjää hyvin avoimessa maastossa, jossa on vähän esteitä. Dijkstran algoritmi taas sokkeloisessa maastossa. 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 Dijkstraan.

# Lähteet

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