Canonical Orderings on Grids

Nathan R. Sturtevant
University of Denver

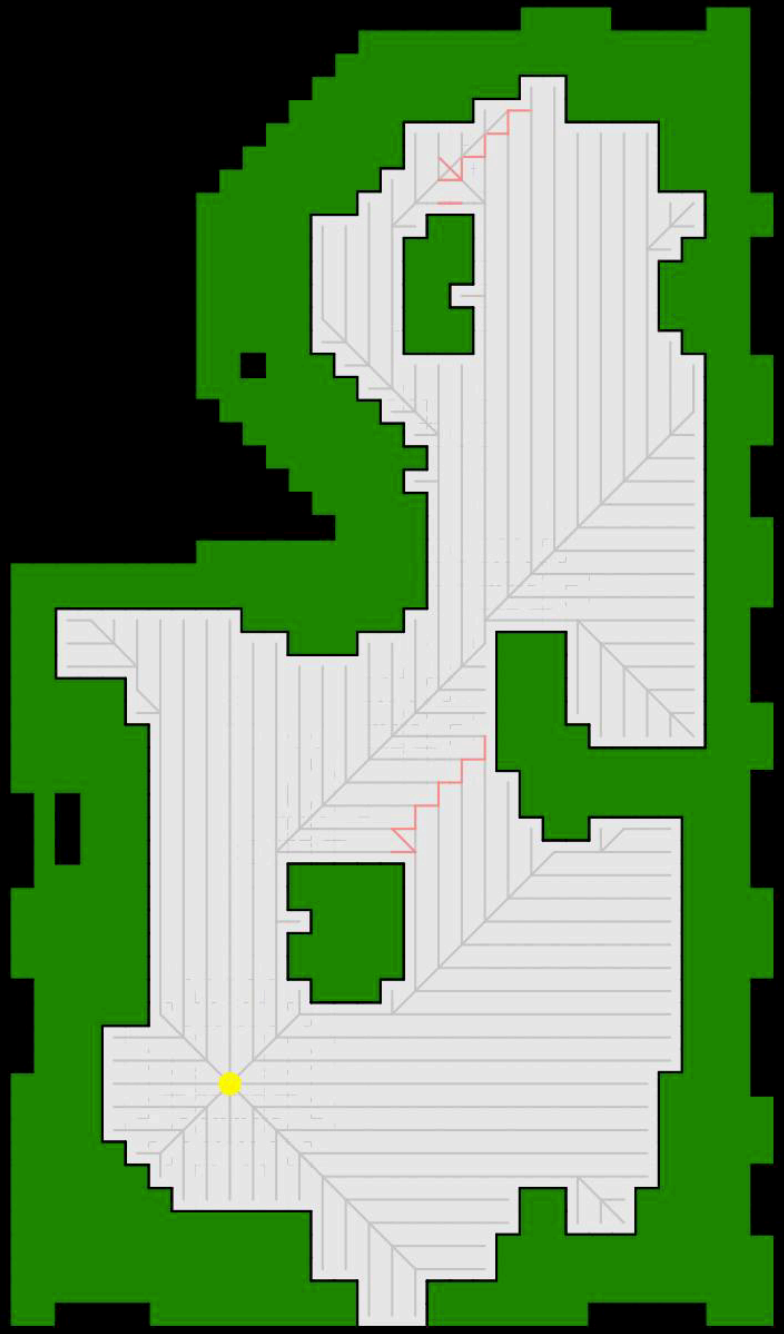
Steve Rabin
DigiPen Institute of Technology

Moving AI lab

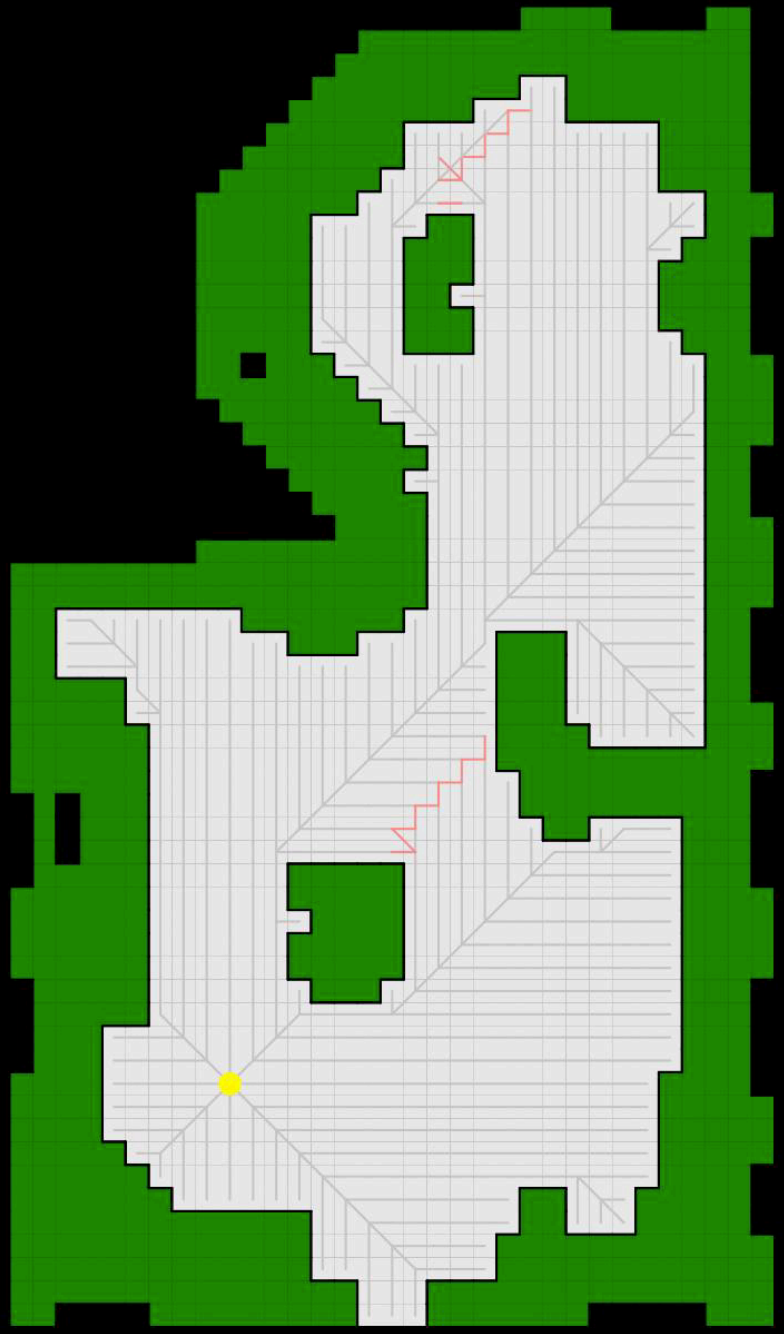
UNIVERSITY of DENVER
DANIEL FELIX RITCHIE SCHOOL OF ENGINEERING & COMPUTER SCIENCE

DigiPen
INSTITUTE OF TECHNOLOGY

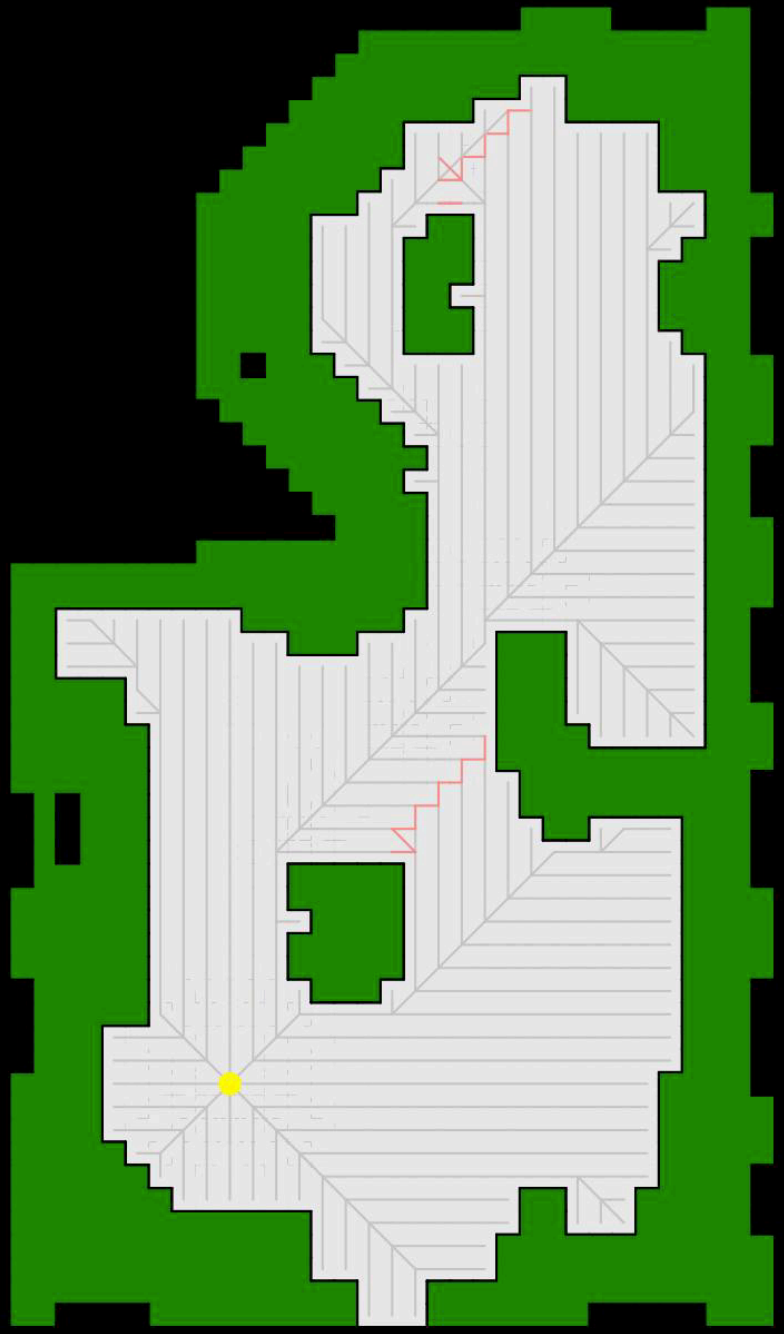
A*



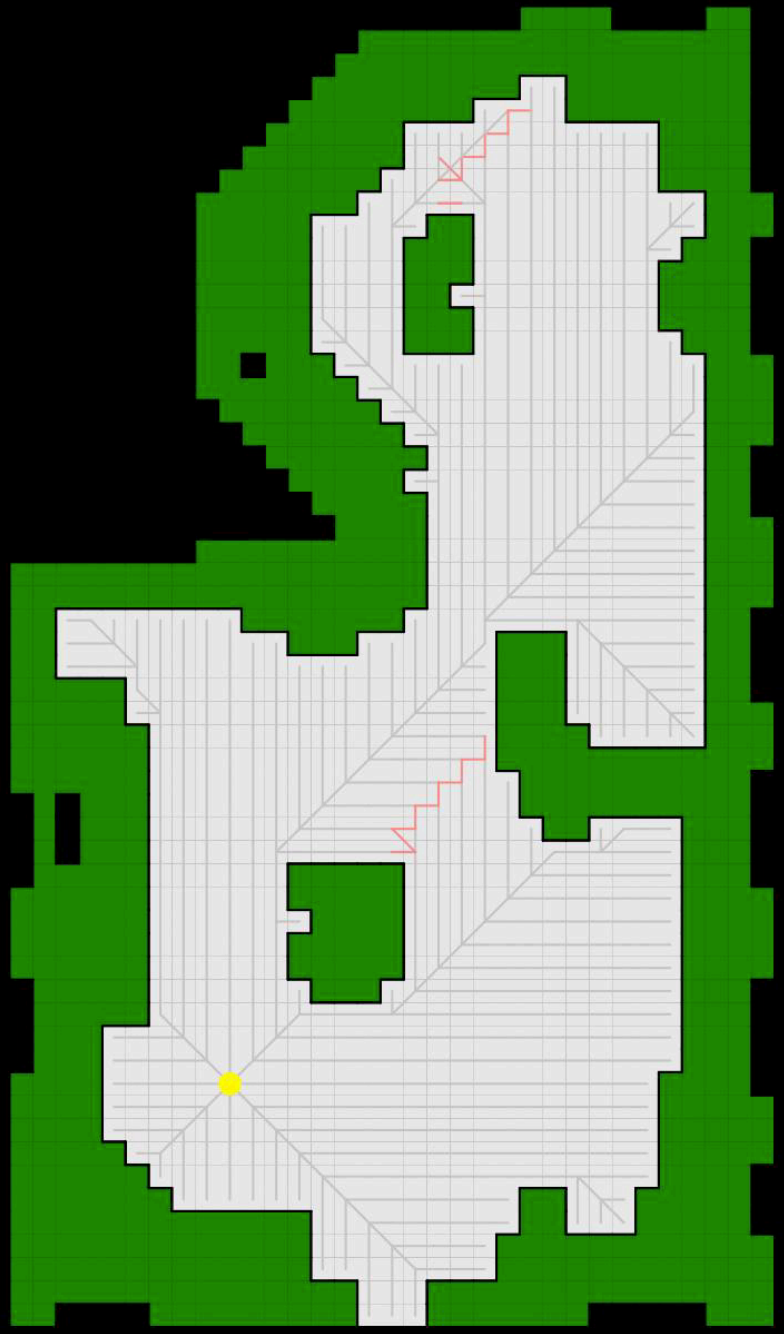
Jump Point Search



A*



Jump Point Search



Contributions

- Jump Point Search (JPS)
  - Harabor and Grastien: 2011; 2014

Contributions

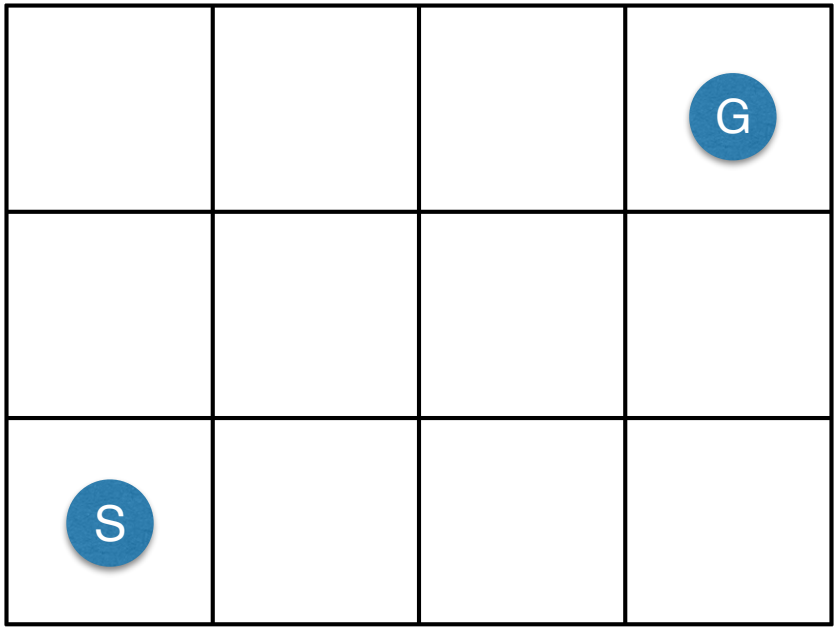
- Jump Point Search (JPS)
  - Harabor and Grastien: 2011; 2014
- Decompose JPS:
  - Canonical ordering of states
  - Jumping policy
  - Best first search

Contributions

- Jump Point Search (JPS)
  - Harabor and Grastien: 2011; 2014
- Decompose JPS:
  - Canonical ordering of states
  - Jumping policy
  - Best first search
- Construct new algorithms
  - Canonical A*, Canonical Dijkstra
  - Bounded JPS, Weighted JPS

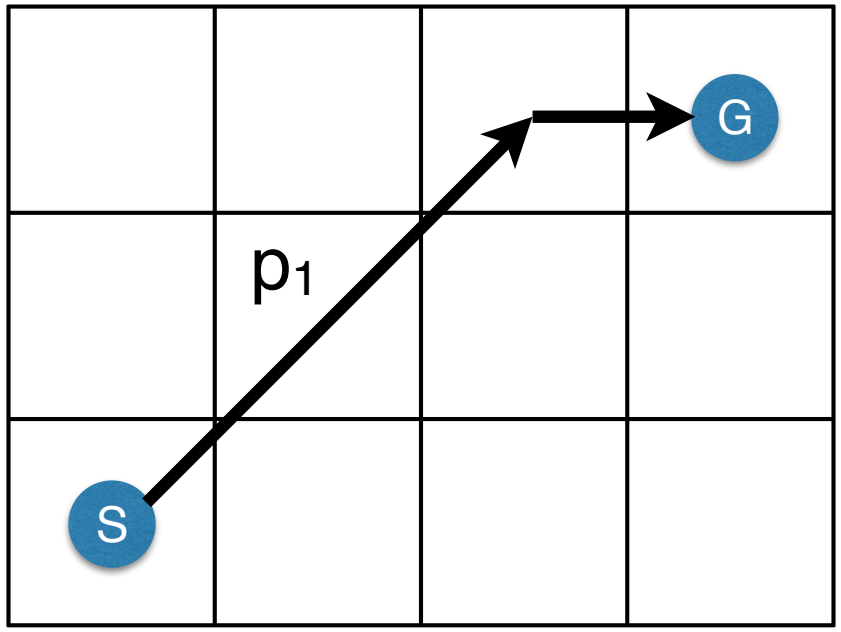
Canonical ordering of paths

- Order all **optimal** paths:
  - Path p_1 is preferred over path p_2 if
    - p_1 has diagonal actions prior to p_2



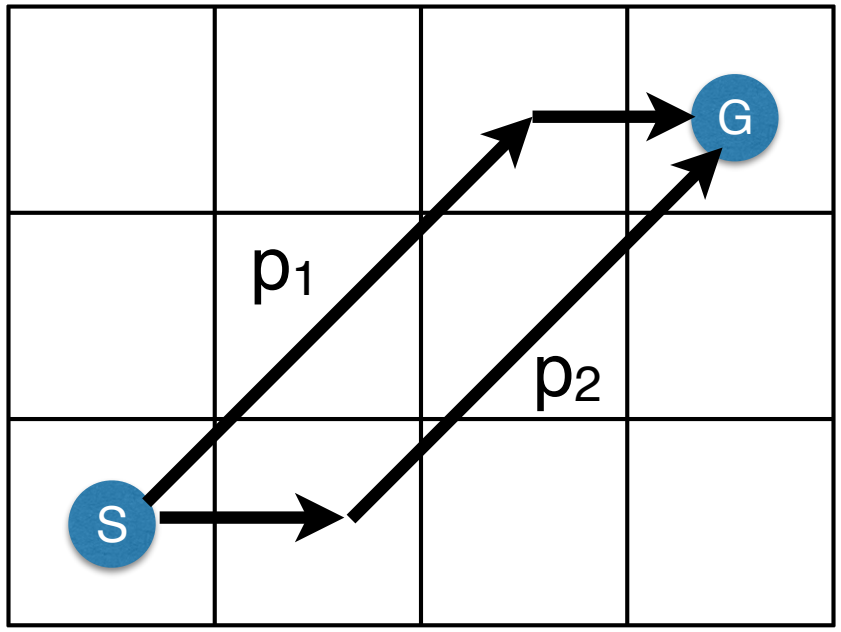
Canonical ordering of paths

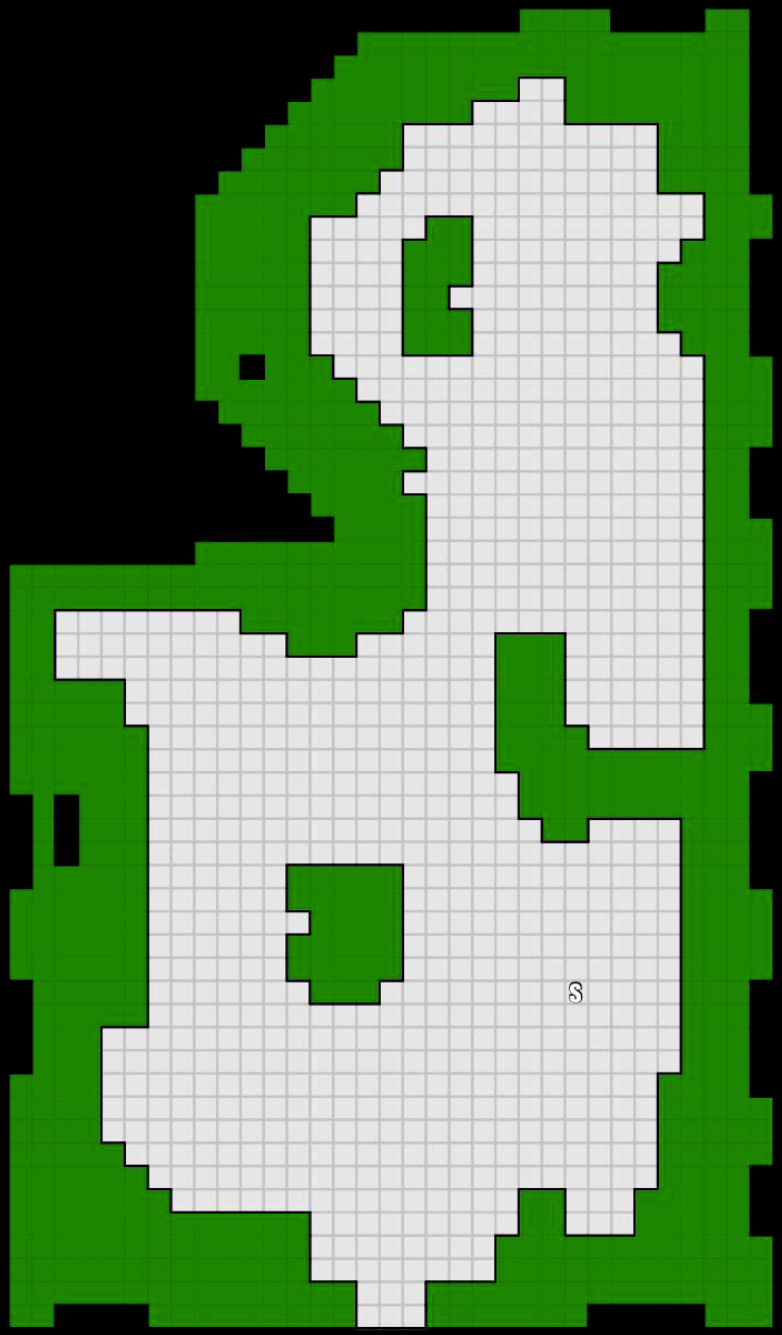
- Order all **optimal** paths:
  - Path p_1 is preferred over path p_2 if
    - p_1 has diagonal actions prior to p_2

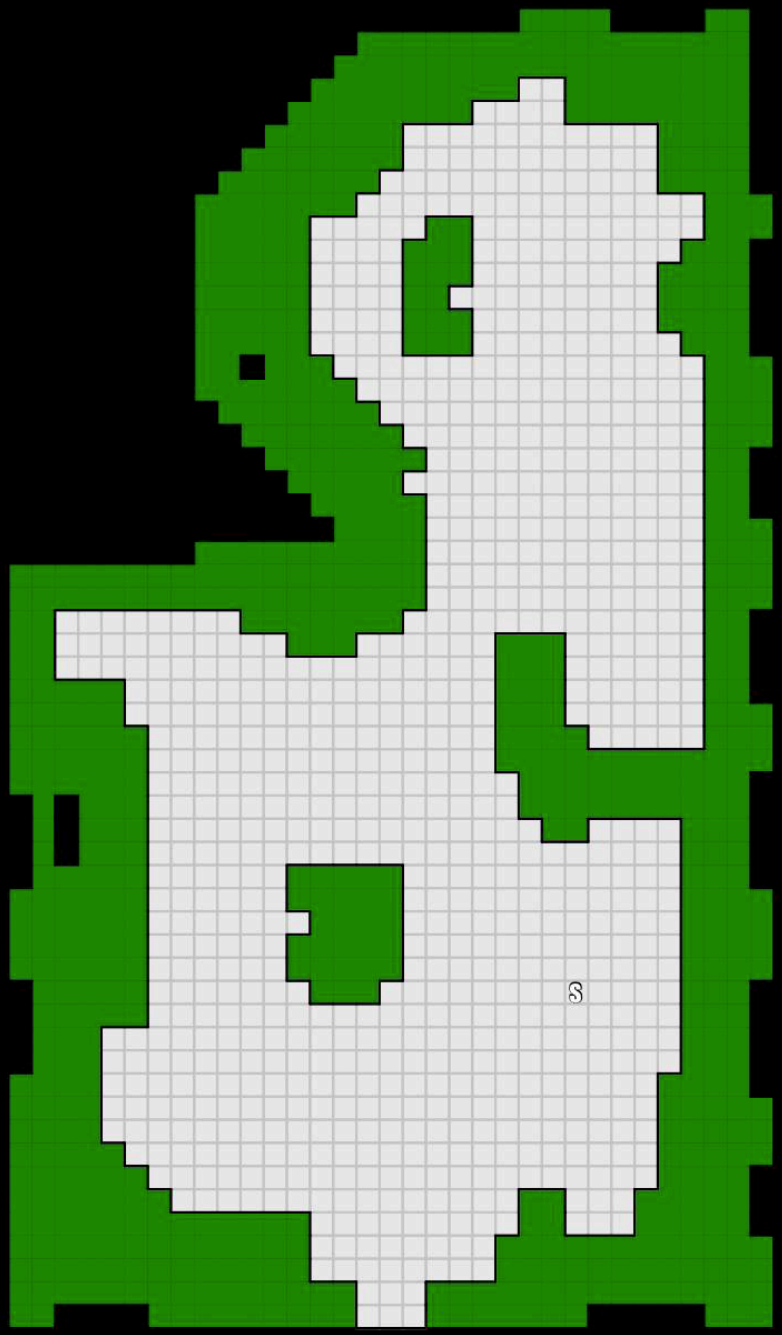


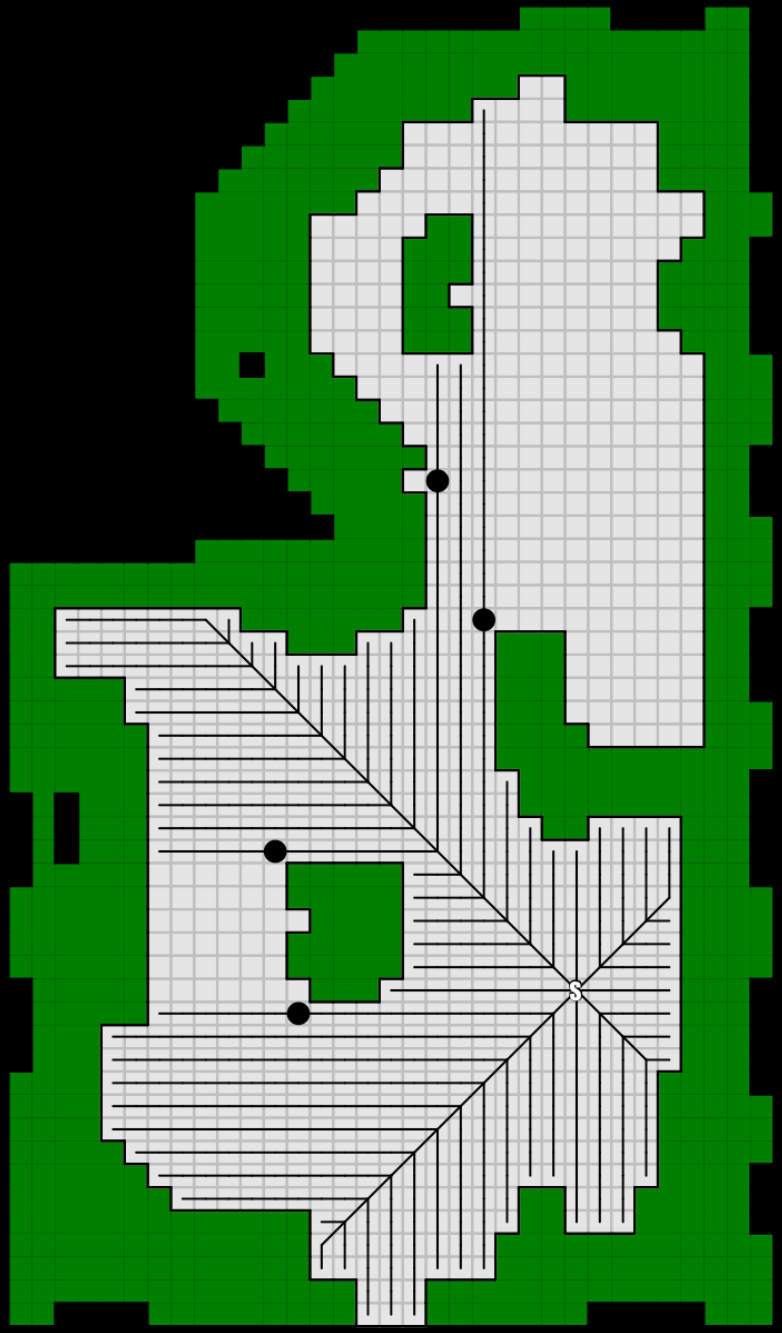
Canonical ordering of paths

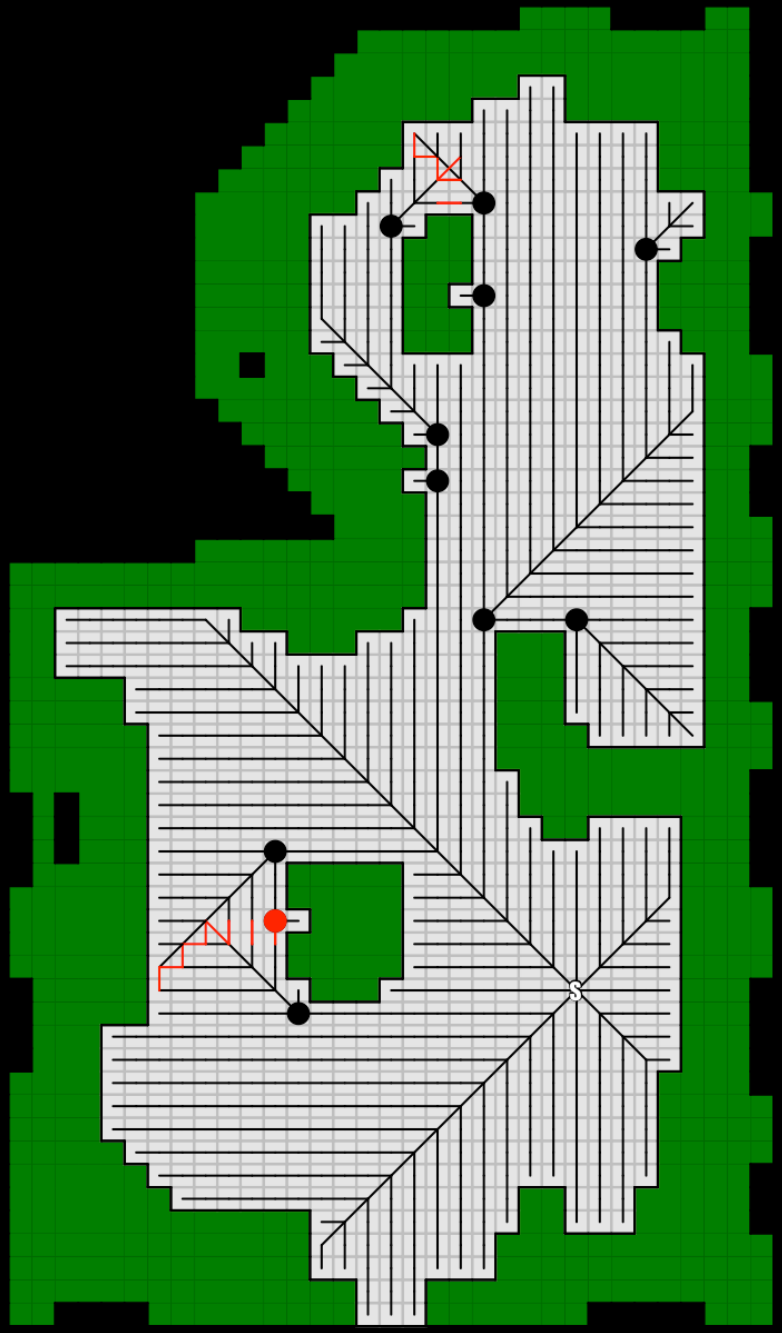
- Order all **optimal** paths:
  - Path p_1 is preferred over path p_2 if
    - p_1 has diagonal actions prior to p_2

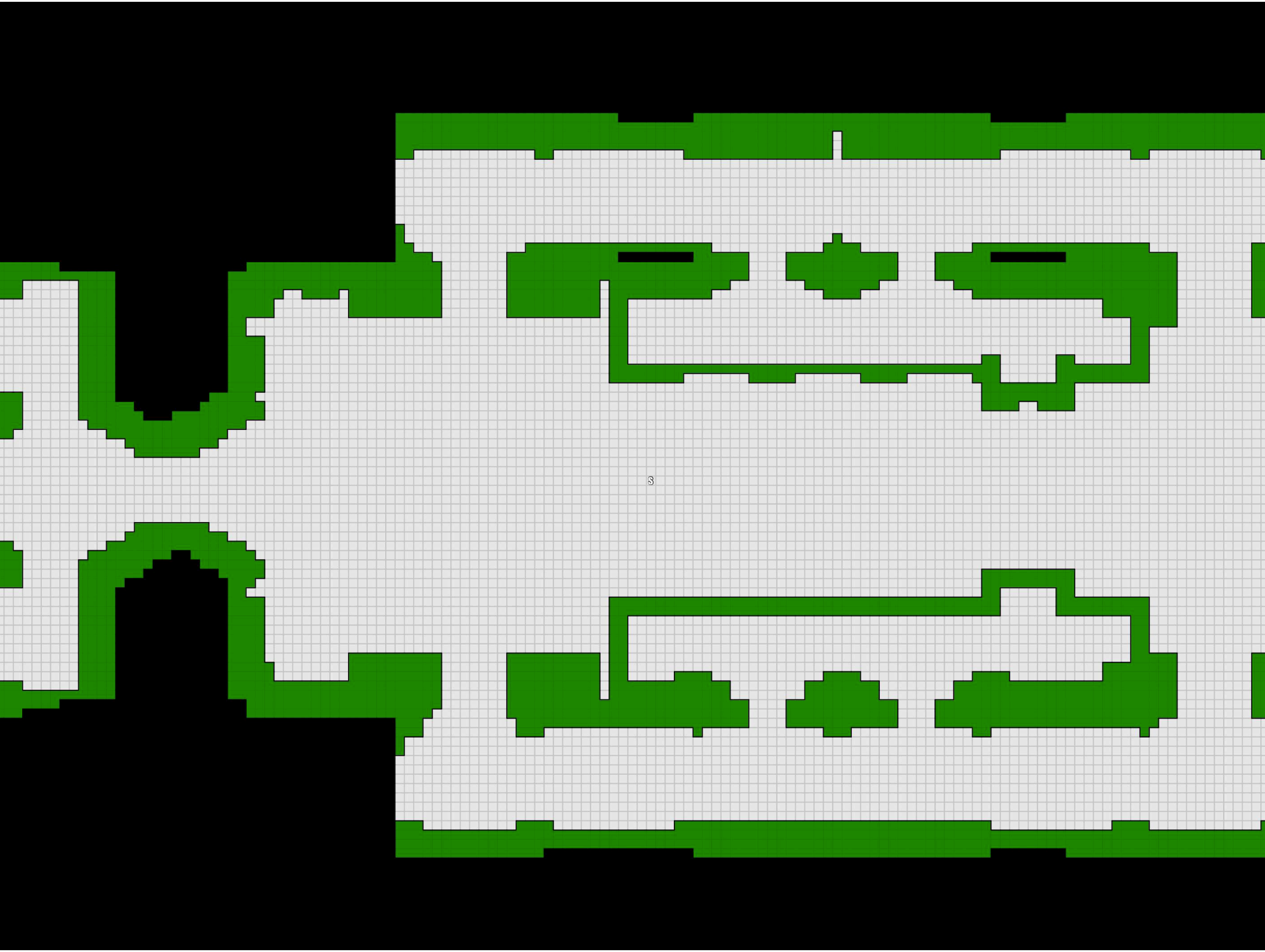


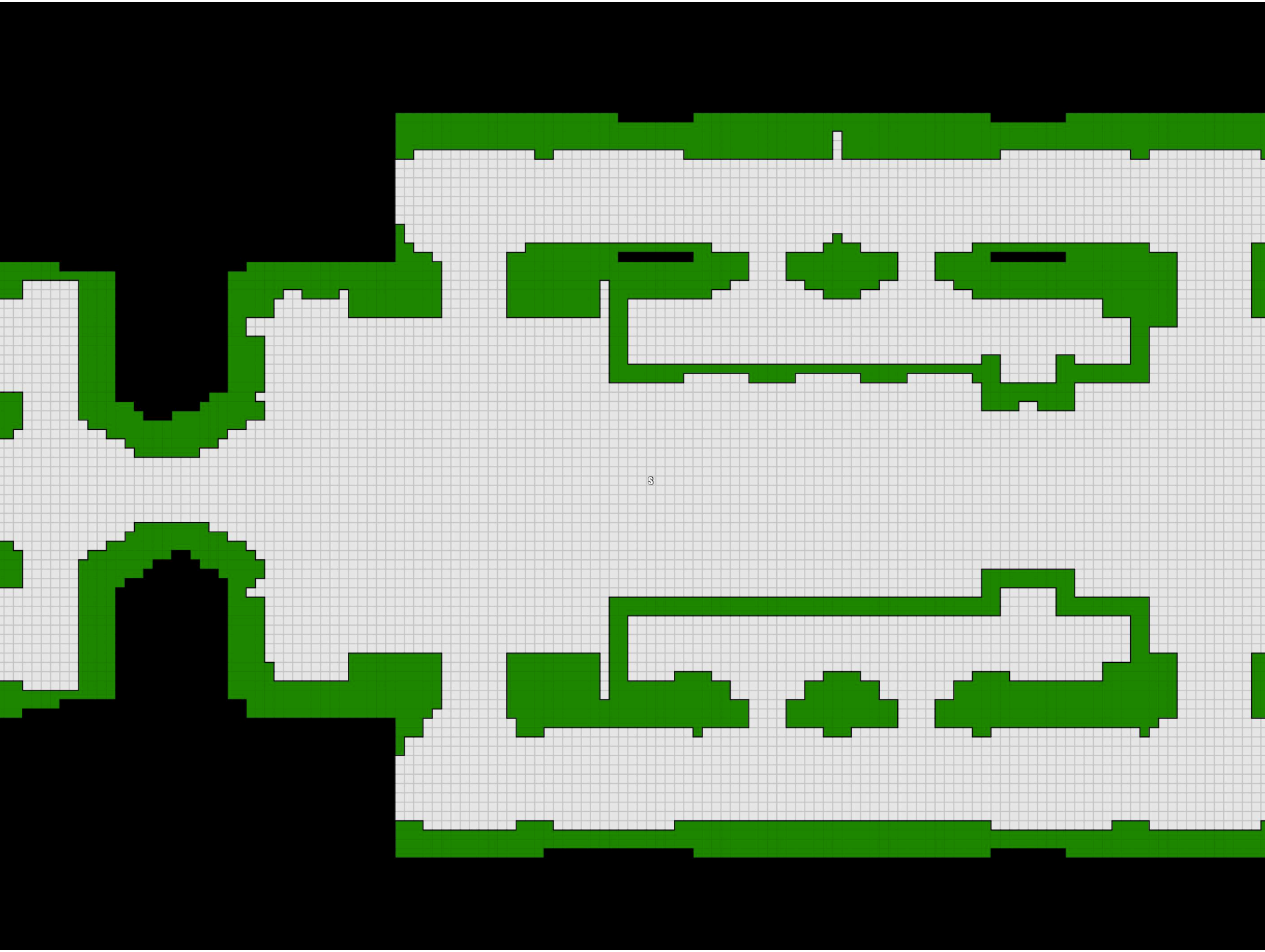


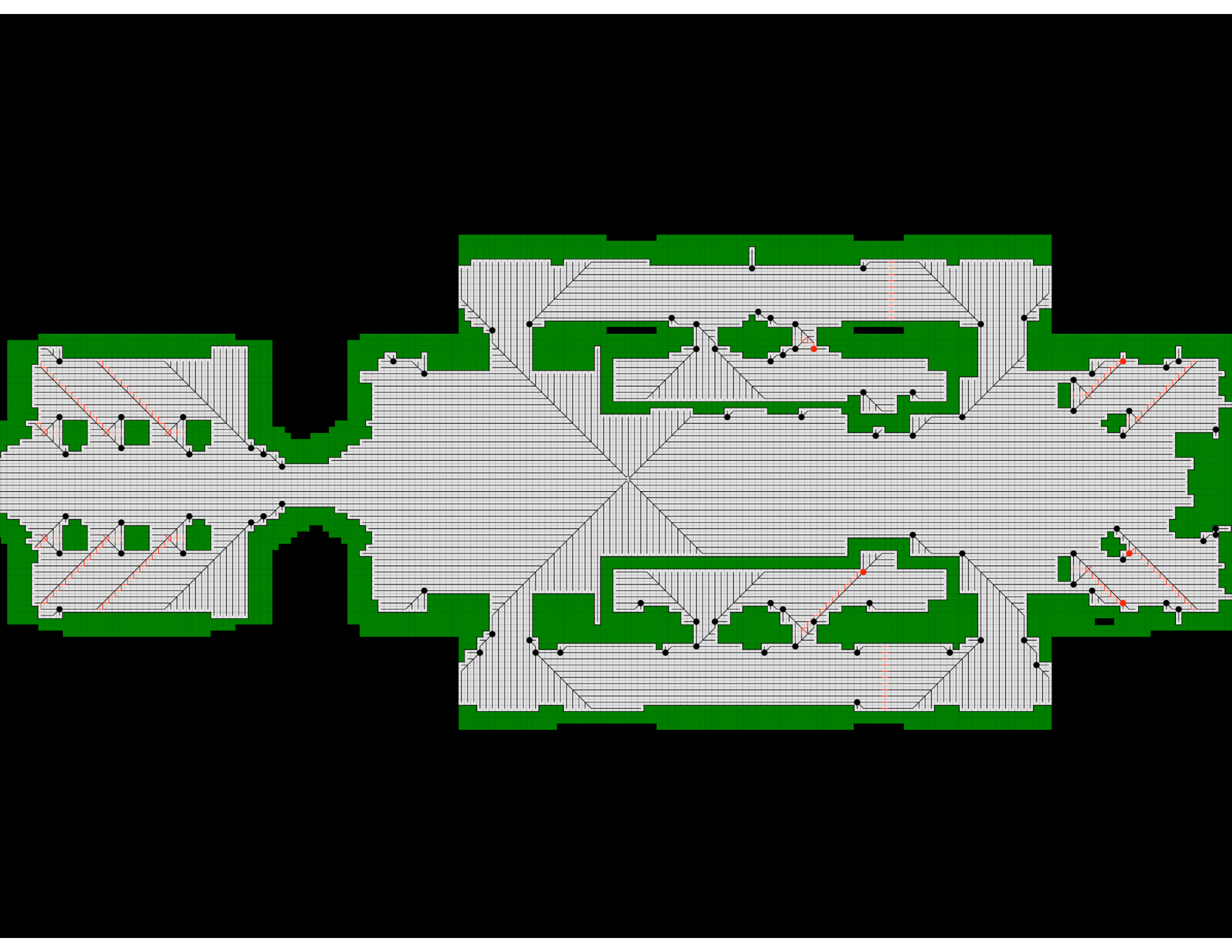



S


S

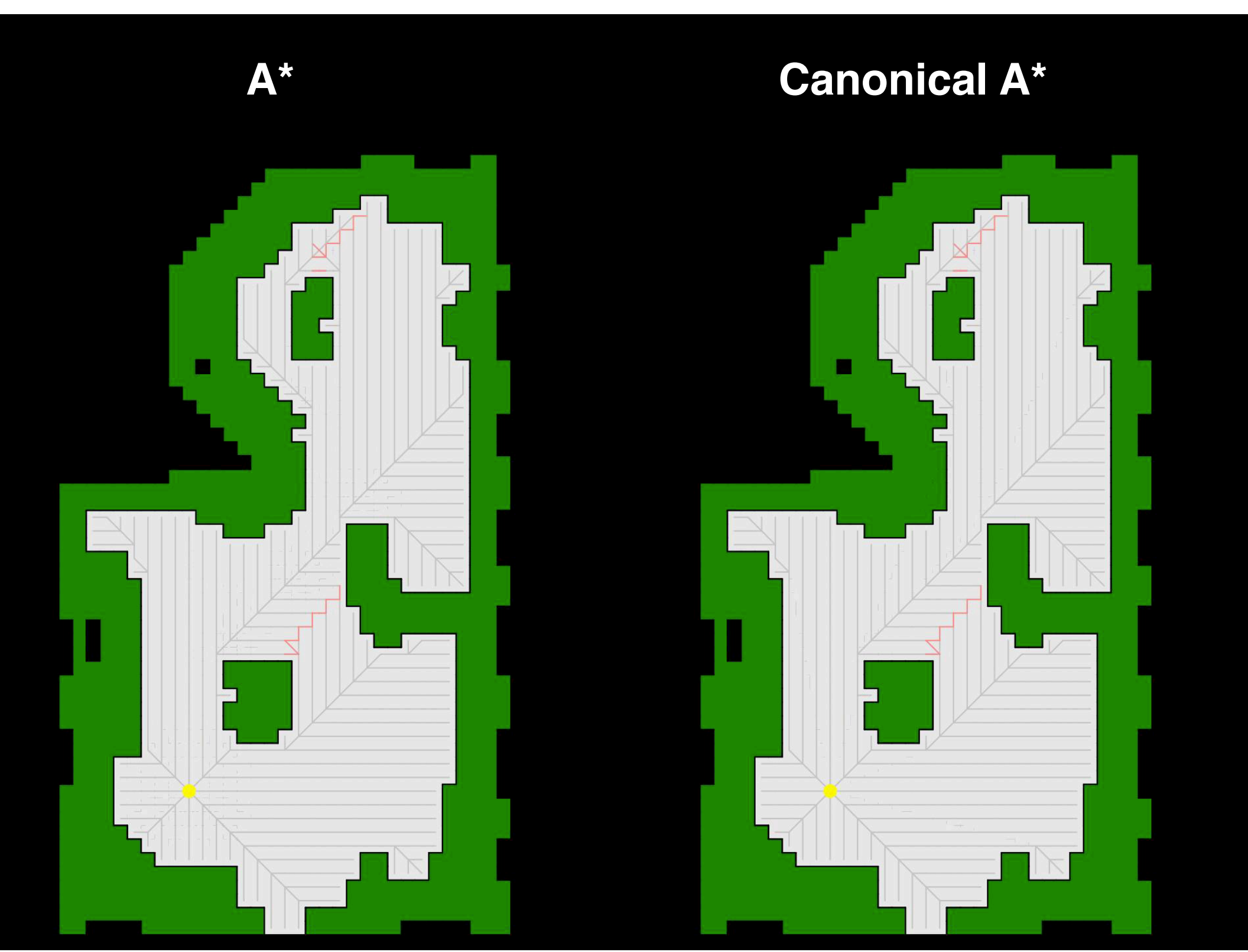


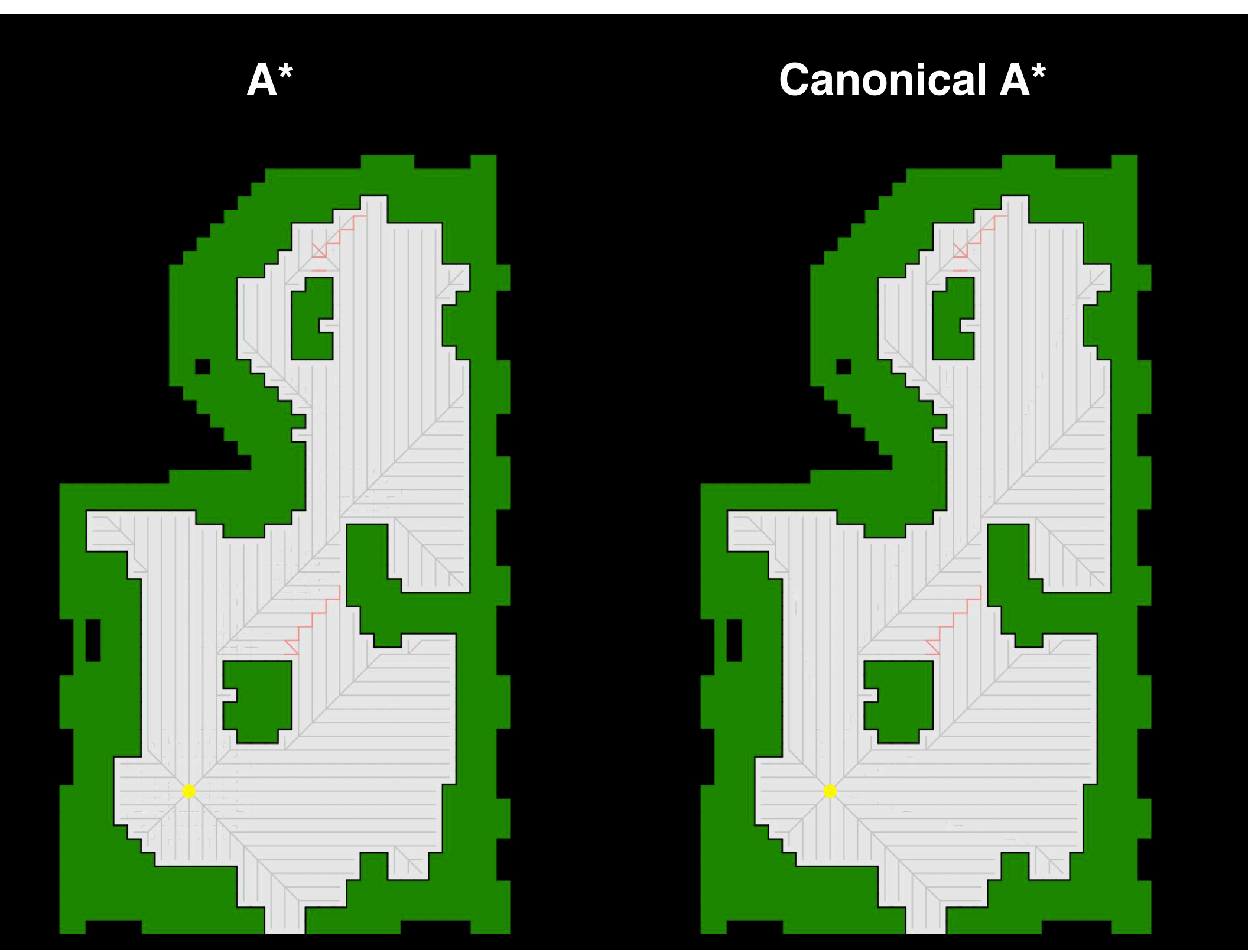


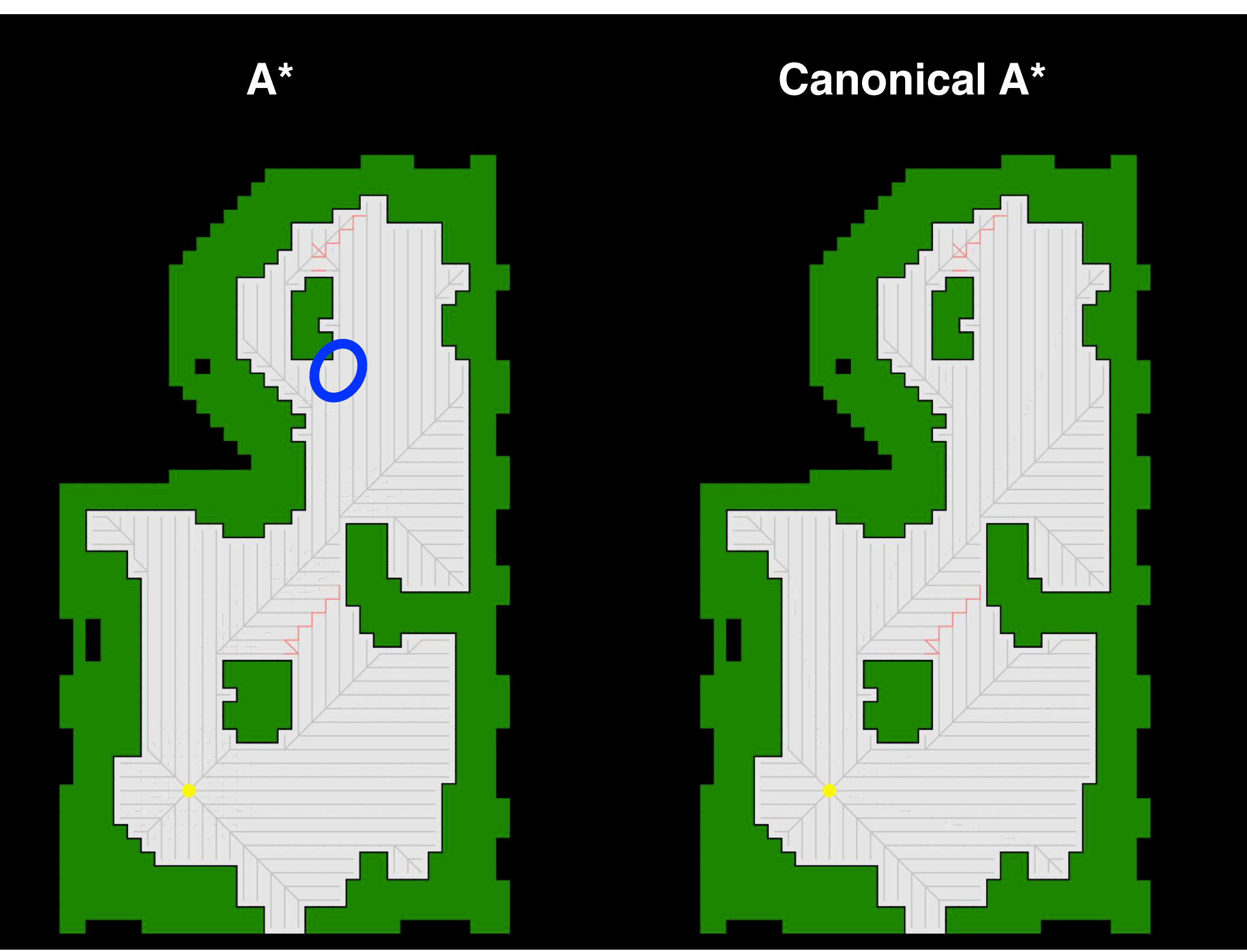


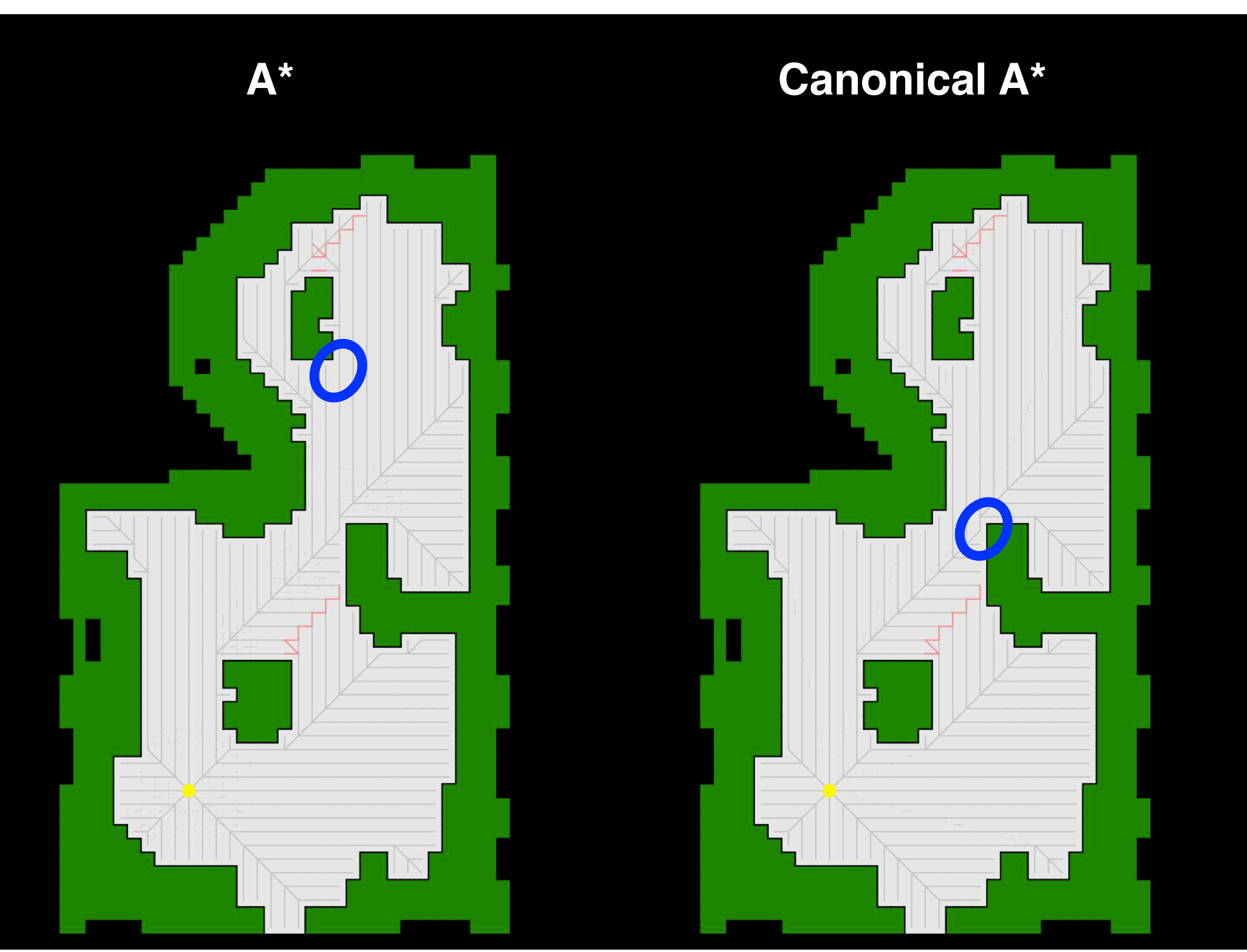
Algorithm #1: Canonical A*

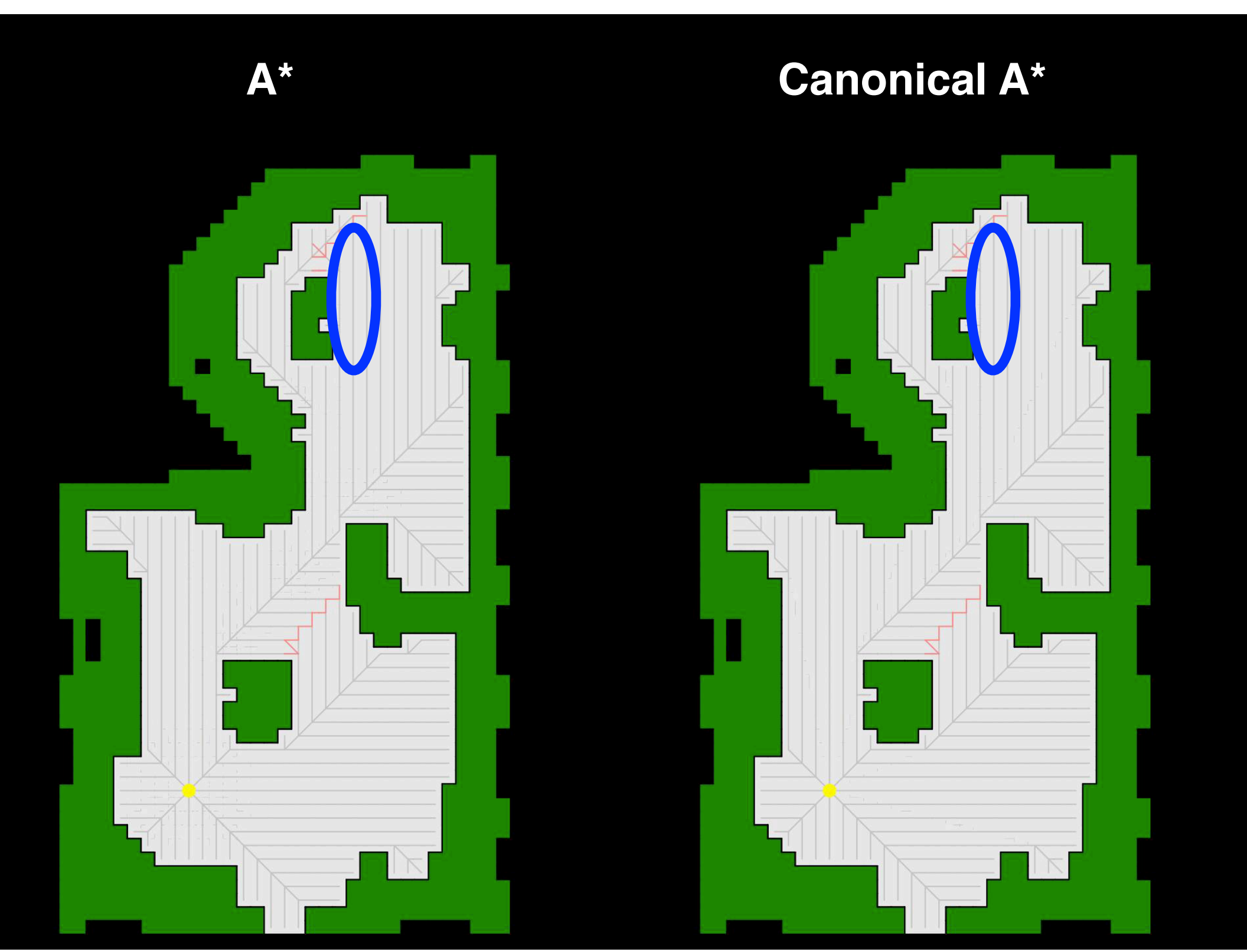
- Run regular A* using the canonical ordering
- Looks the same as A* on the full graph
- Slightly different node expansions
  - Tie-breaking at the goal
- Far fewer generations


A*
Canonical A*


A*
Canonical A*


A*
Canonical A*


A*
Canonical A*


A*
Canonical A*

Experimental Results

	A*	CA*	JPS
Time (ms)	5.600	2.566	1.982

Apply in state spaces where generations are expensive!

Experimental Results

	A*	CA*	JPS
Time (ms)	5.600	2.566	1.982
Expansions	13,295	13,302	229

Apply in state spaces where generations are expensive!

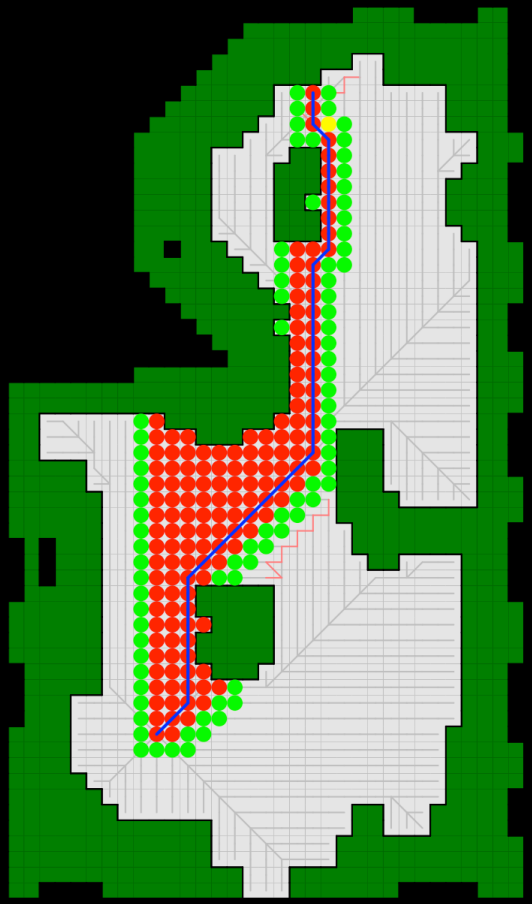
Experimental Results

	A*	CA*	JPS
Time (ms)	5.600	2.566	1.982
Expansions	13,295	13,302	229
Generations	99,483	13,654	61,282

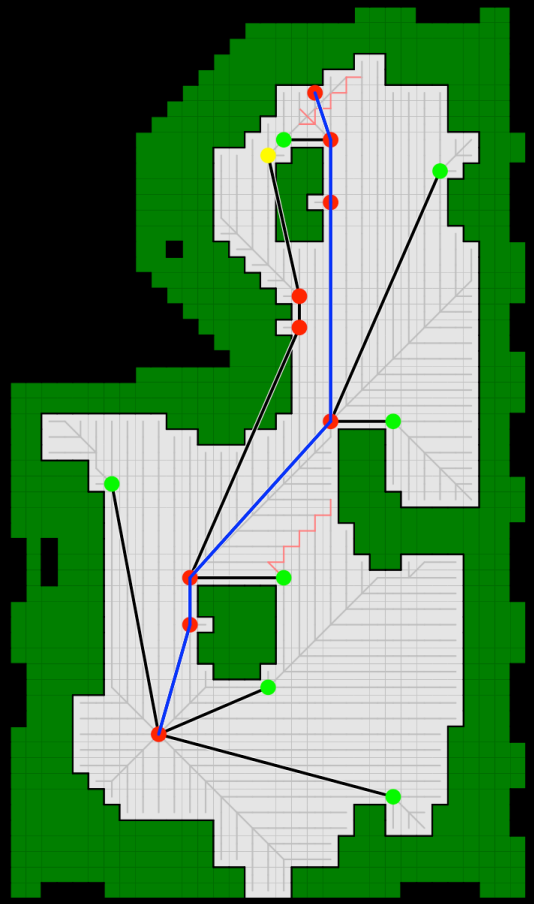
Apply in state spaces where generations are expensive!

JPS Generations

A*/CA*

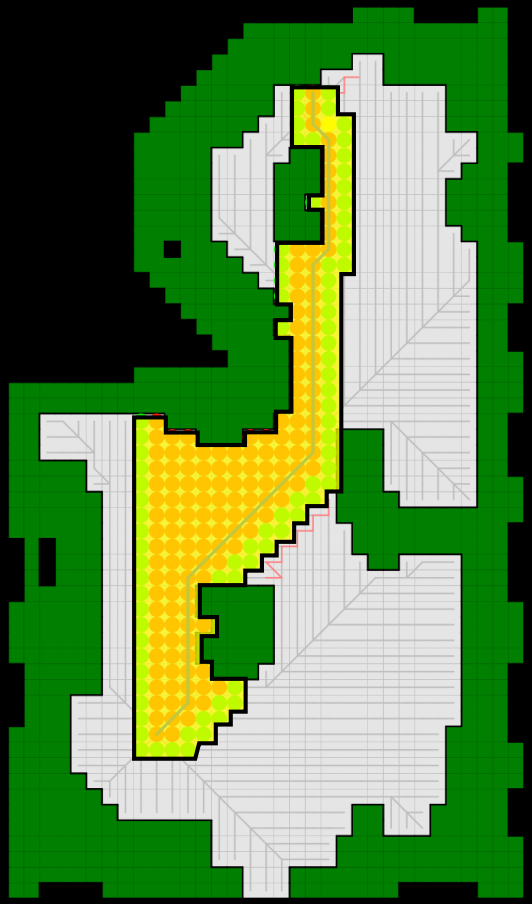


JPS

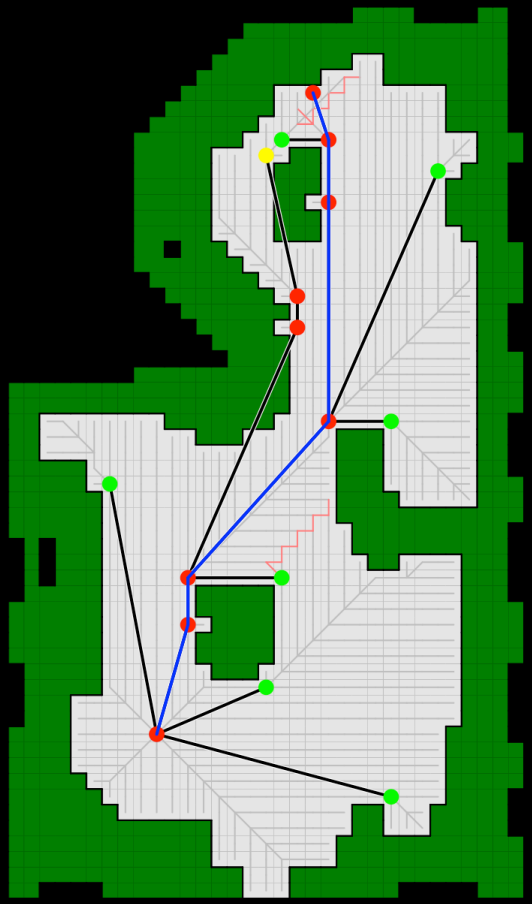


JPS Generations

A*/CA*

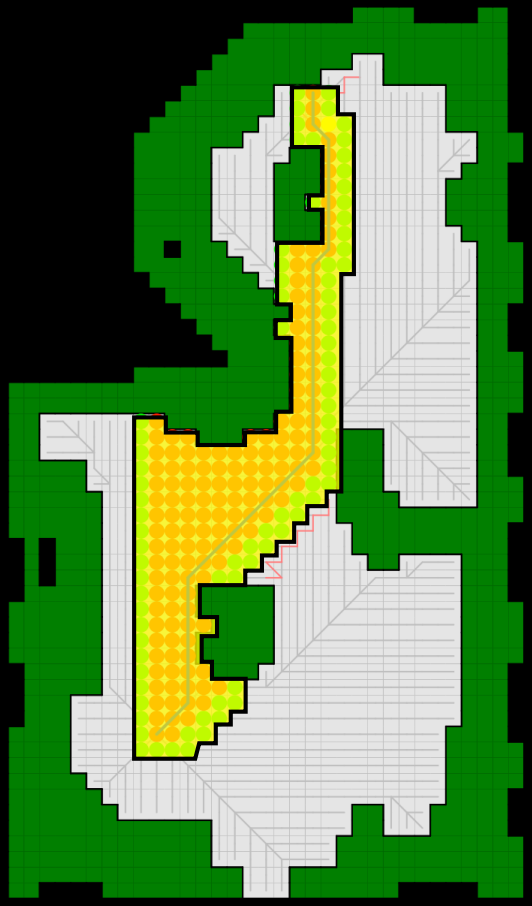


JPS

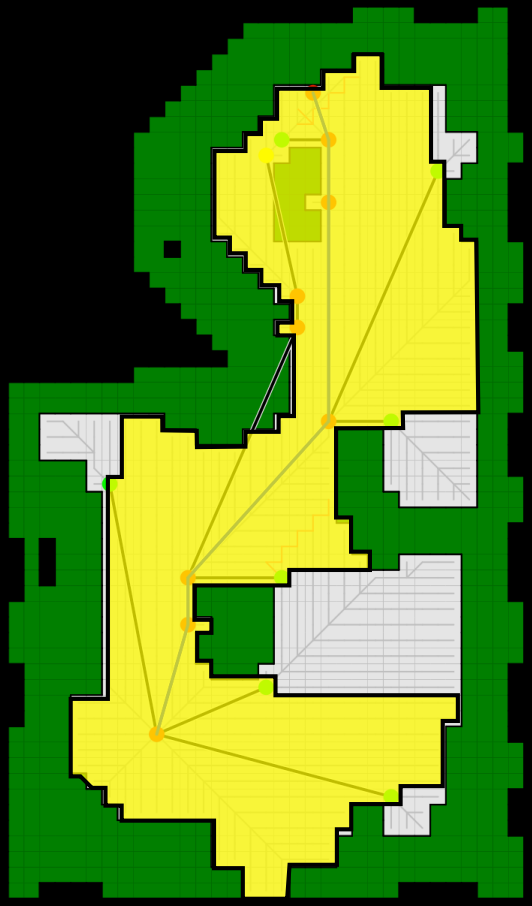


JPS Generations

A*/CA*

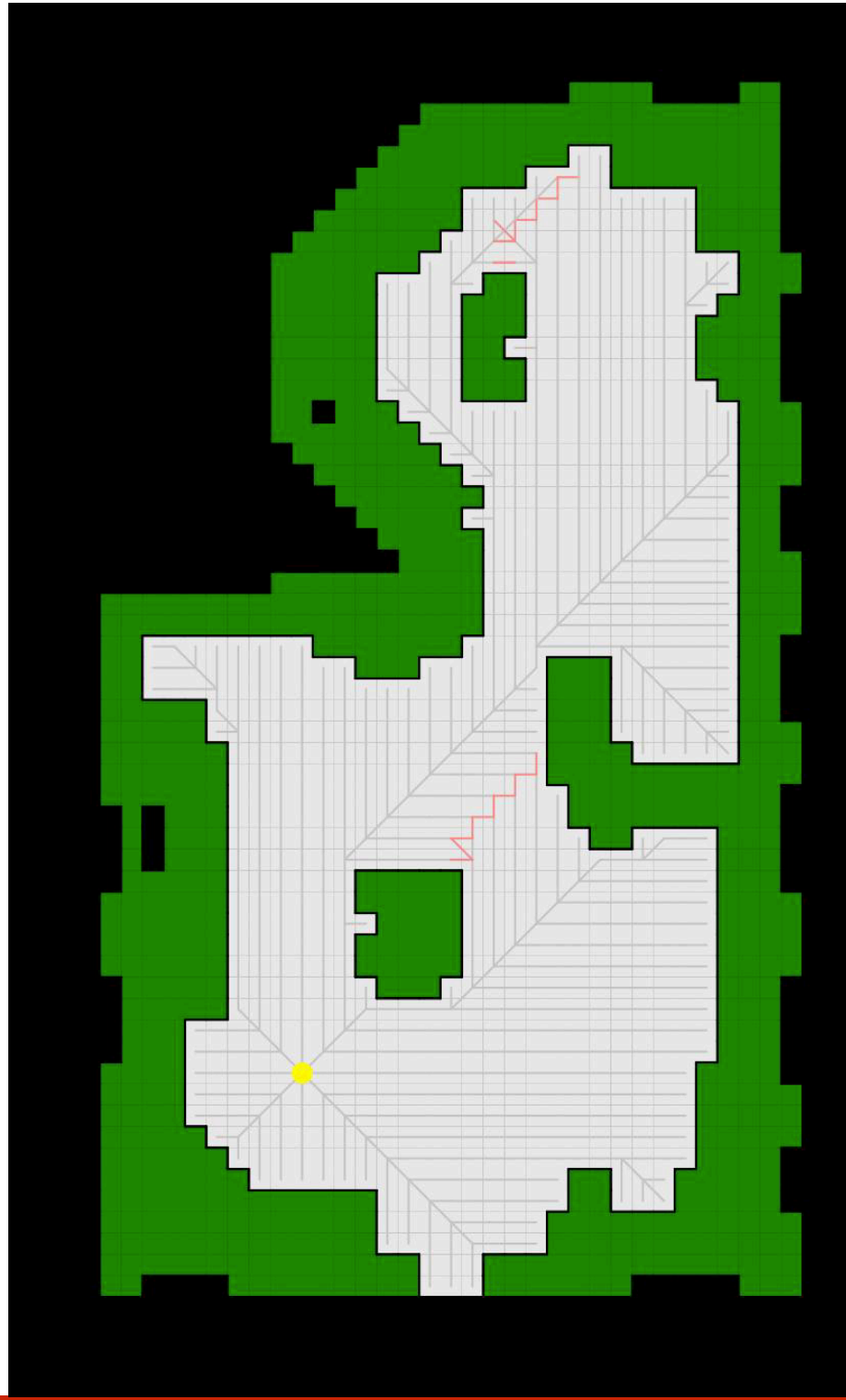


JPS



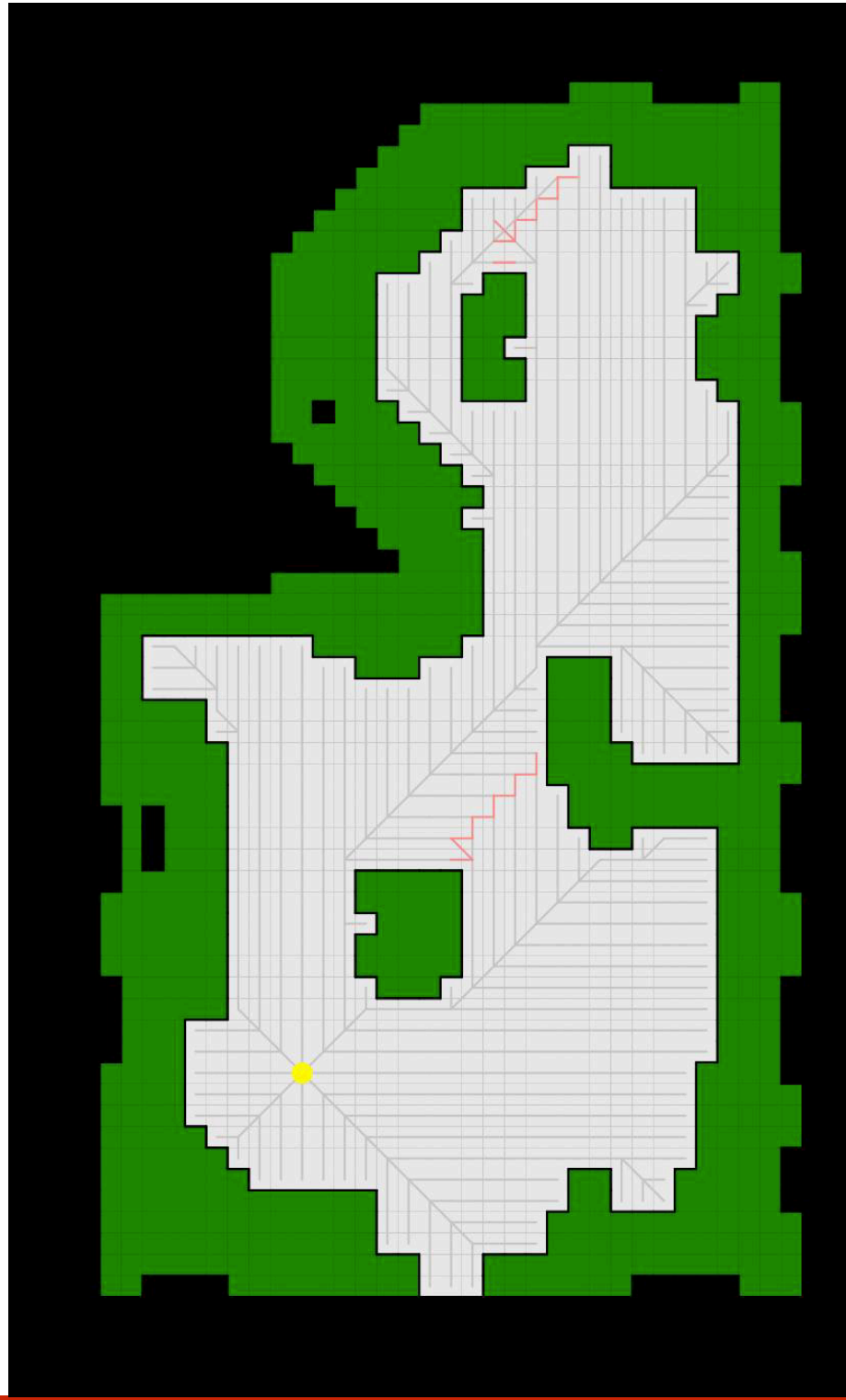
Jumping Policy

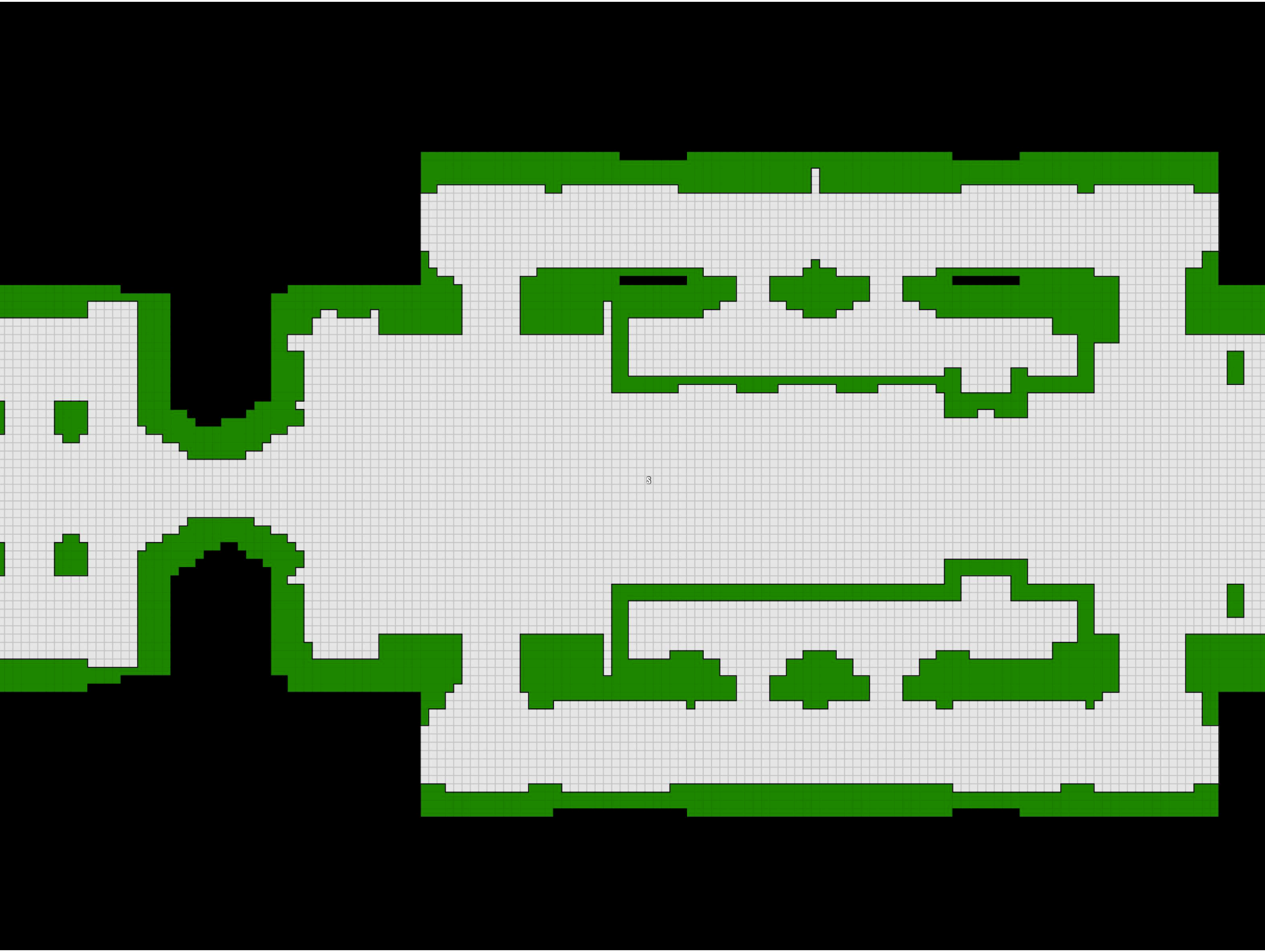
- Continually generate successors until you reach:
  - A jump point (open)
  - The goal (open)
  - *A wall (discard)*

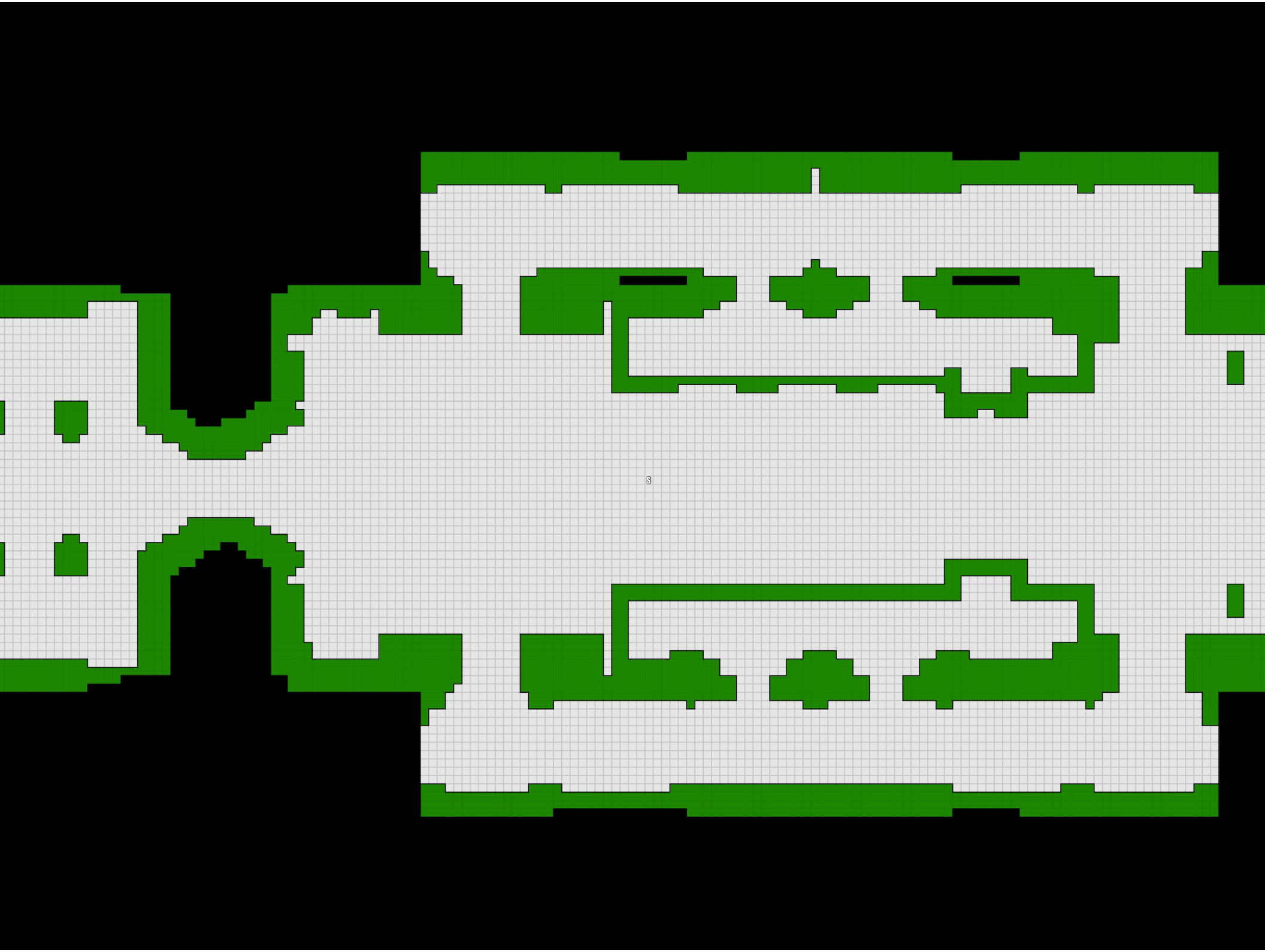


Jumping Policy

- Continually generate successors until you reach:
  - A jump point (open)
  - The goal (open)
  - *A wall (discard)*

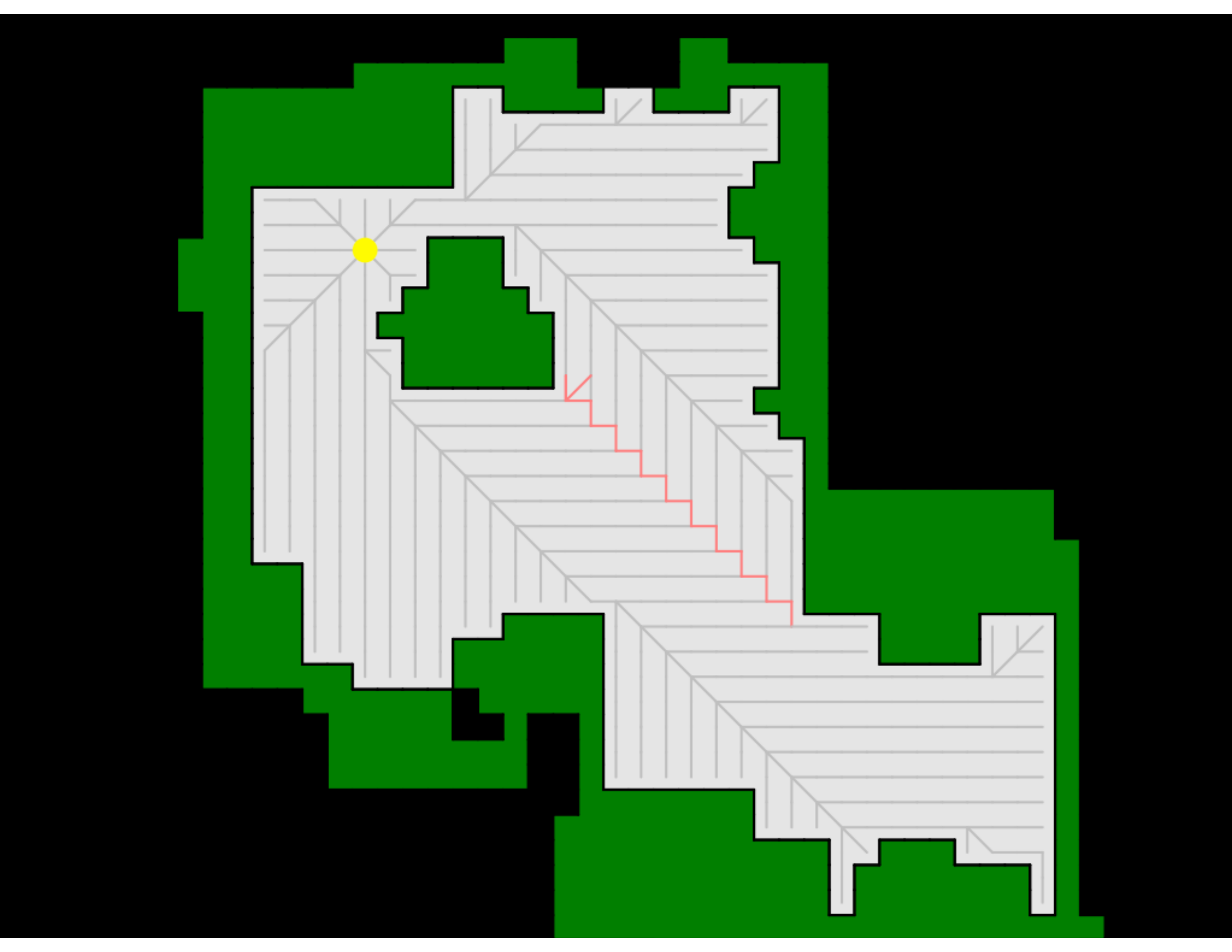


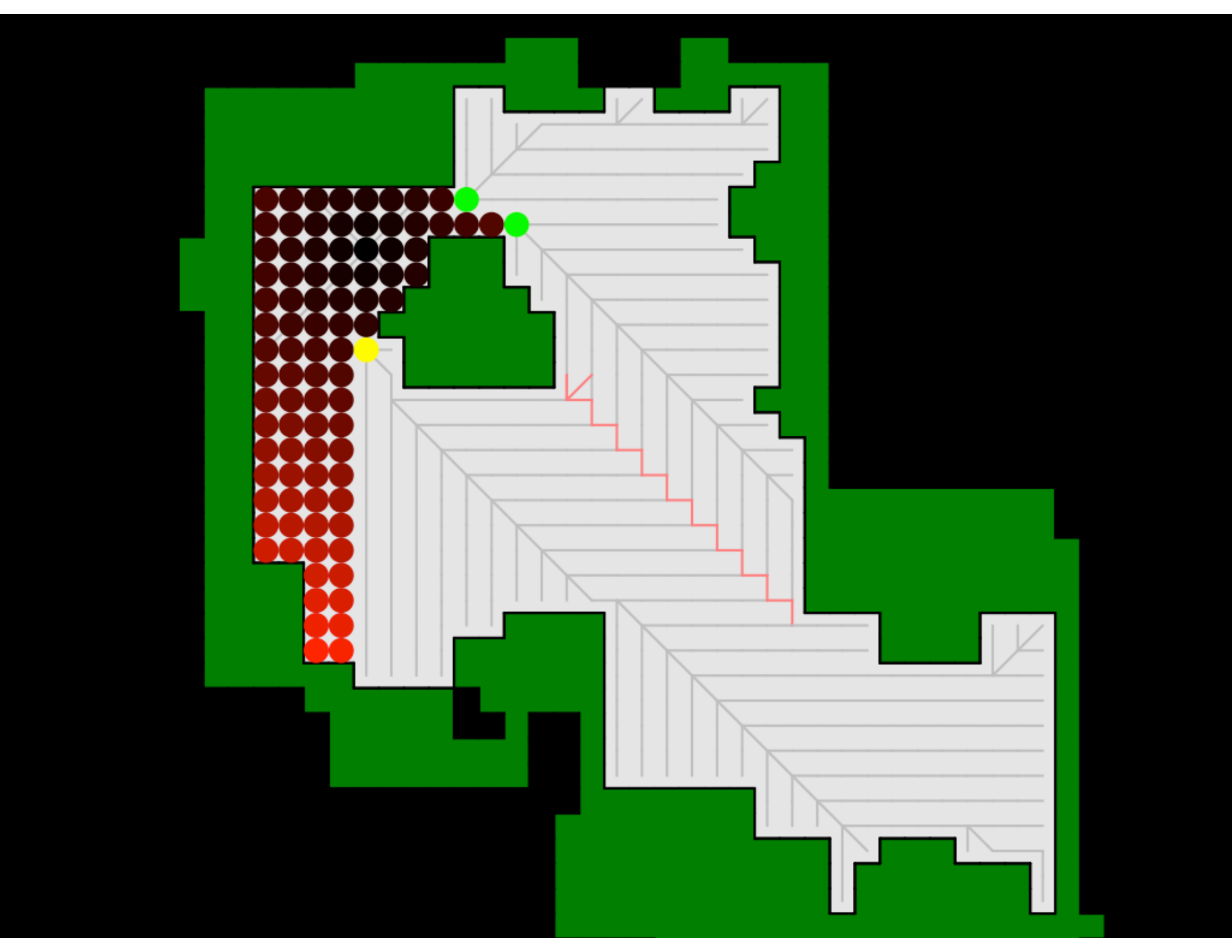


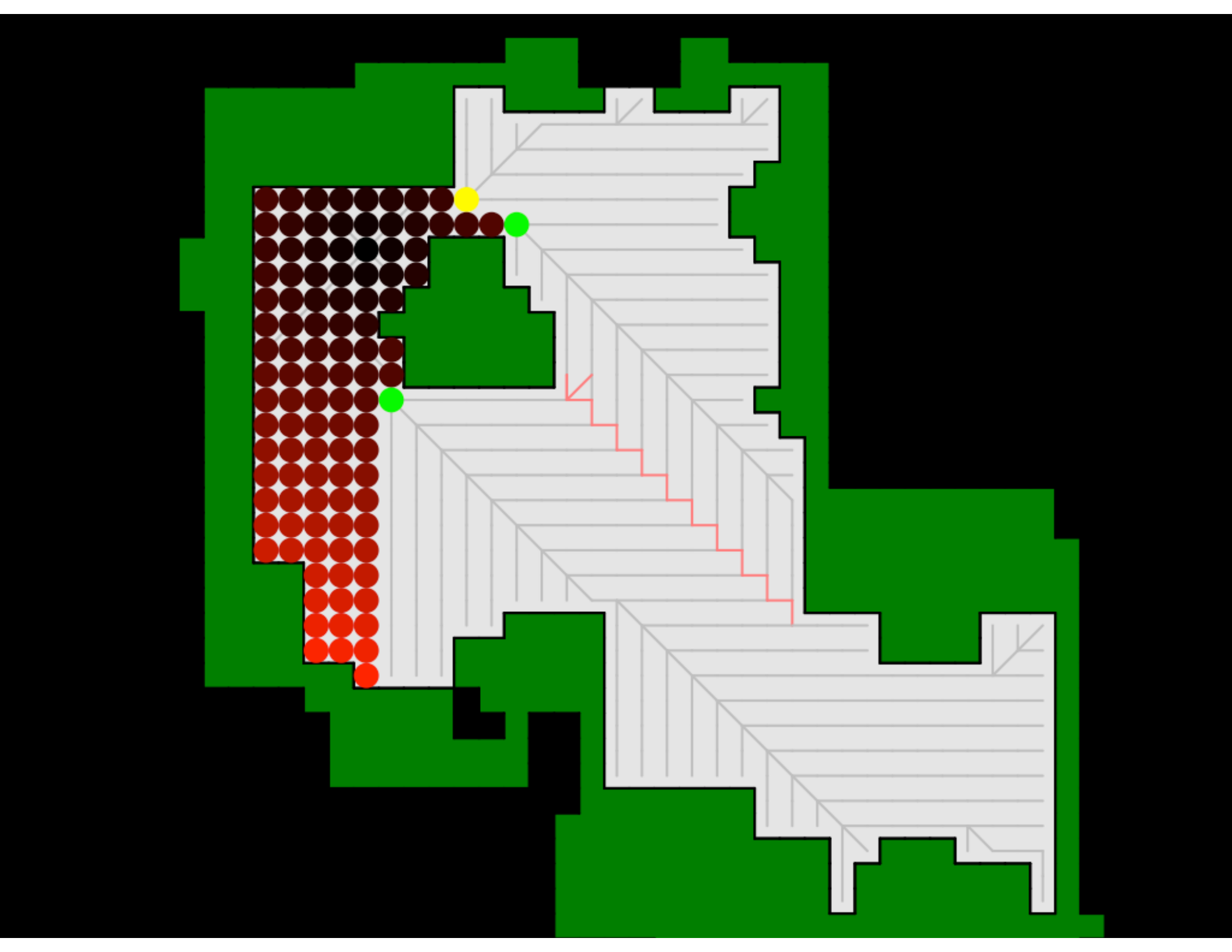


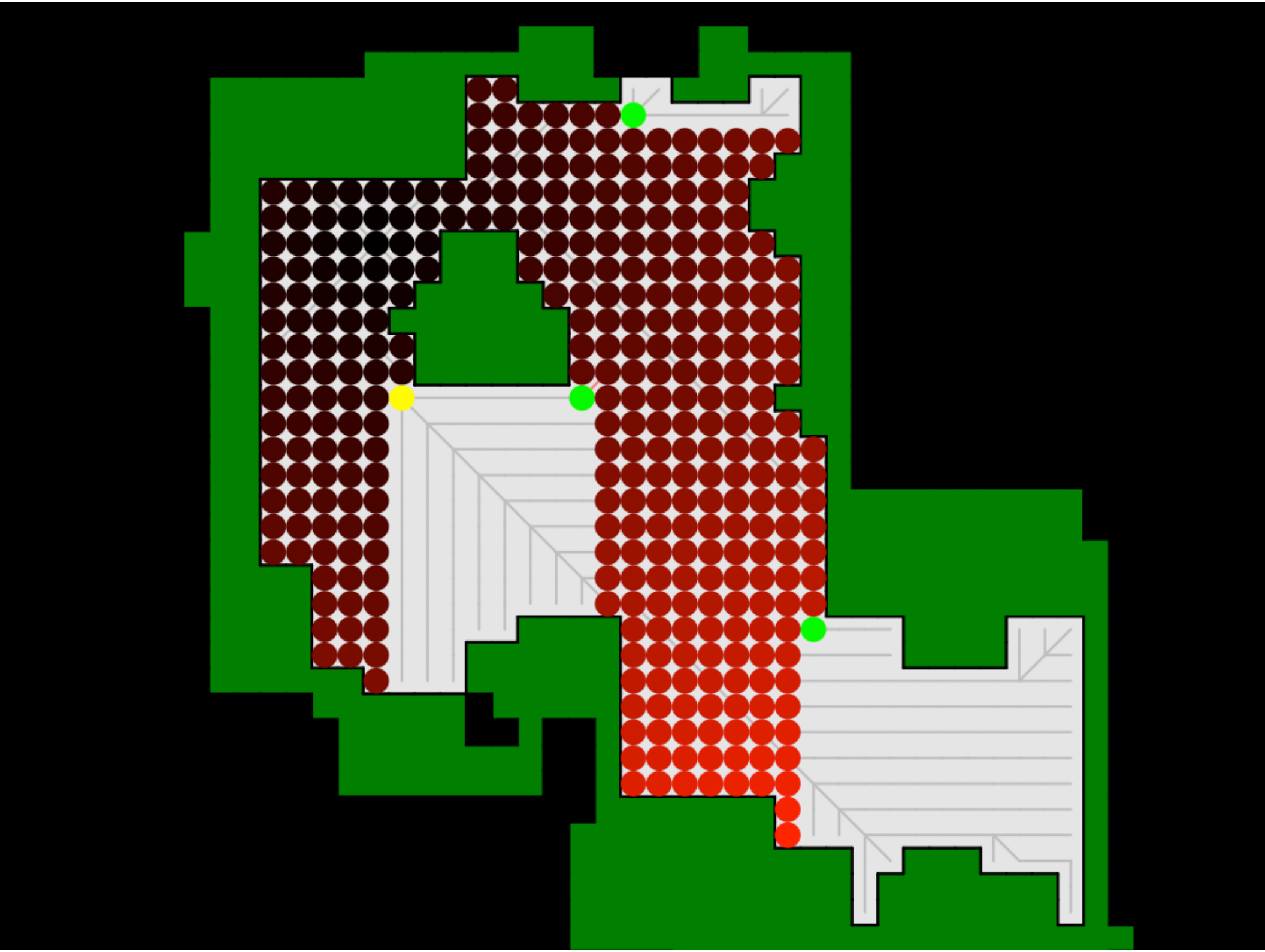
Algorithm #2: Canonical Dijkstra

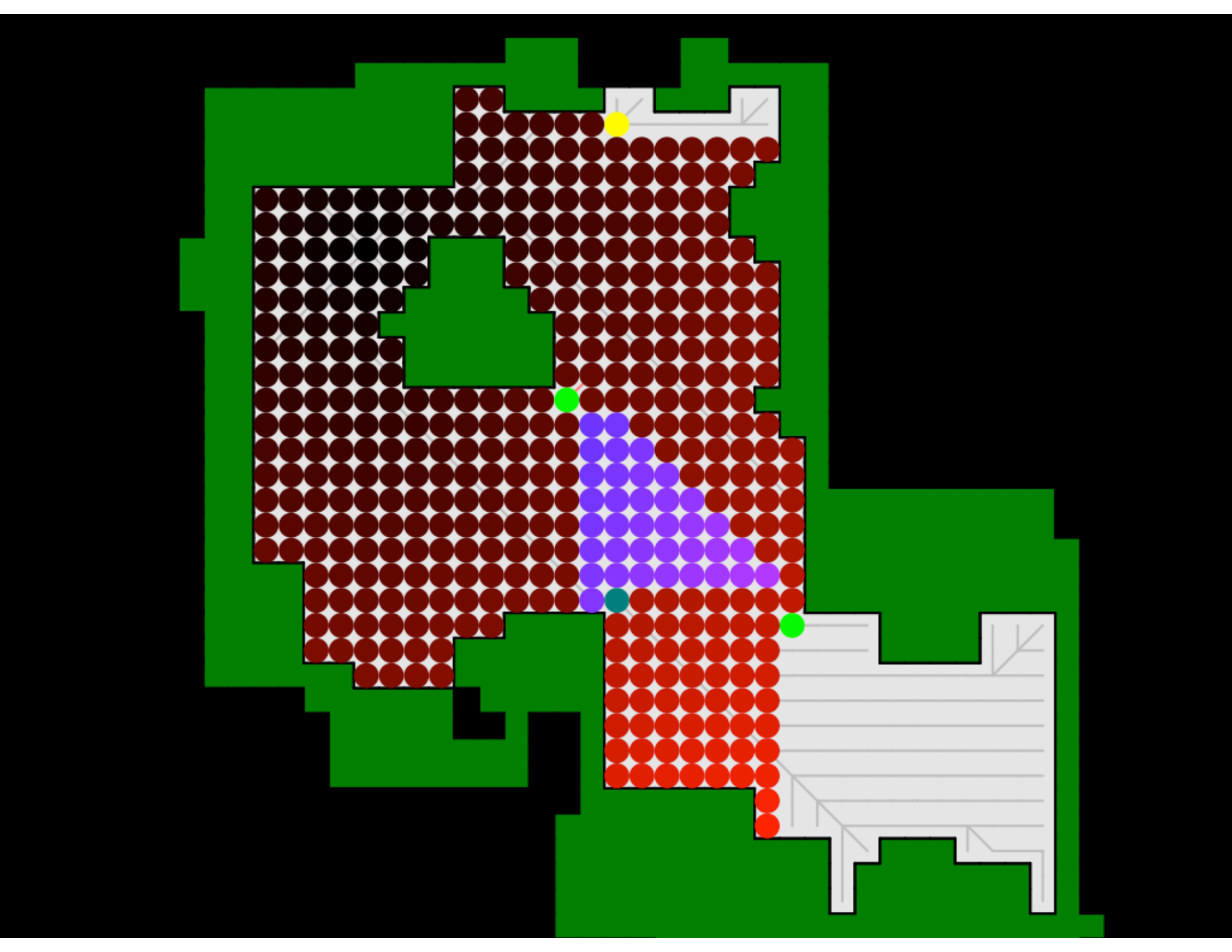
- For fast Single-Source Shortest Path Computation
  - Use Canonical Ordering with Dijkstra
  - When we jump over states, write their g-cost to closed

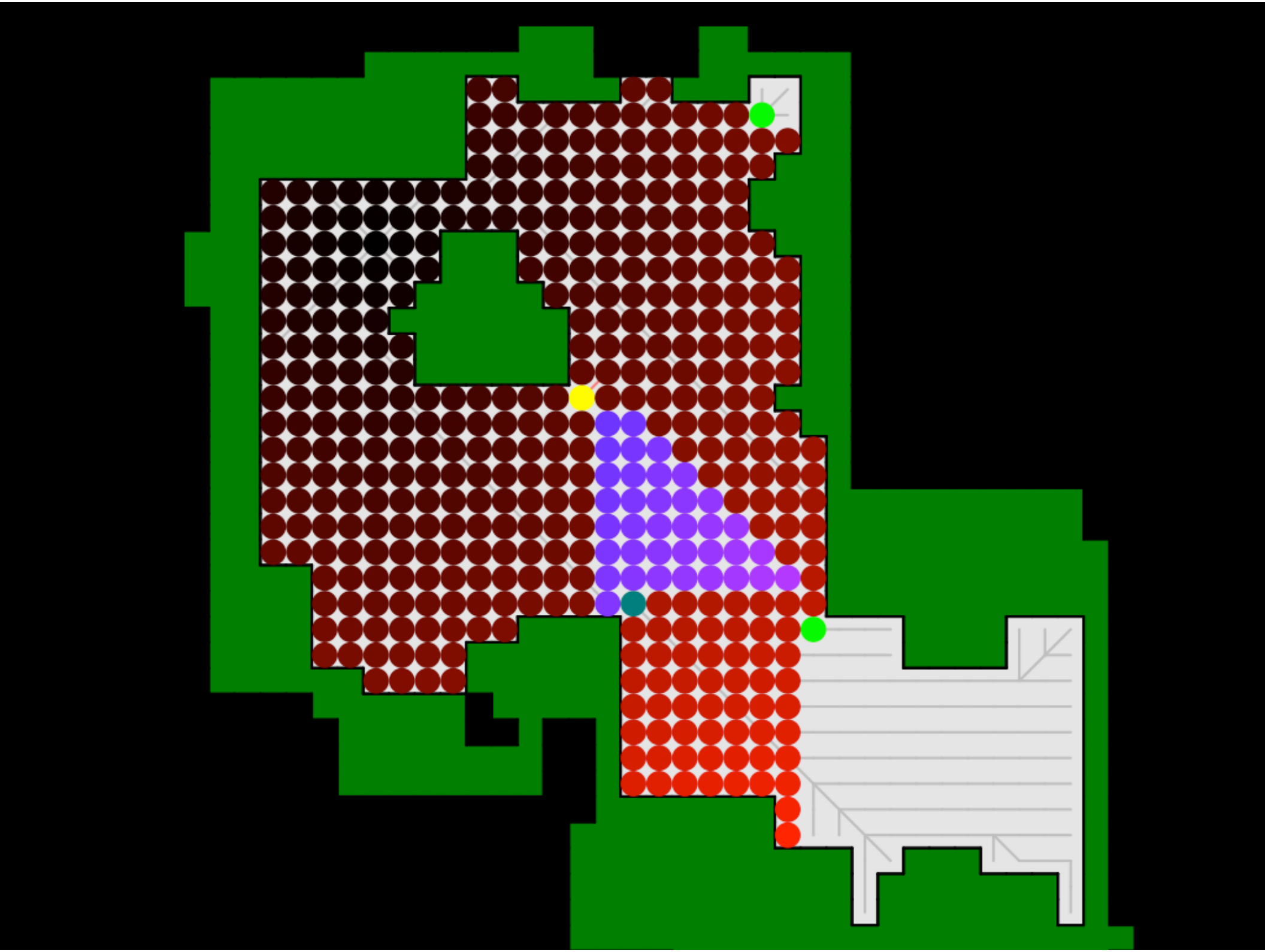


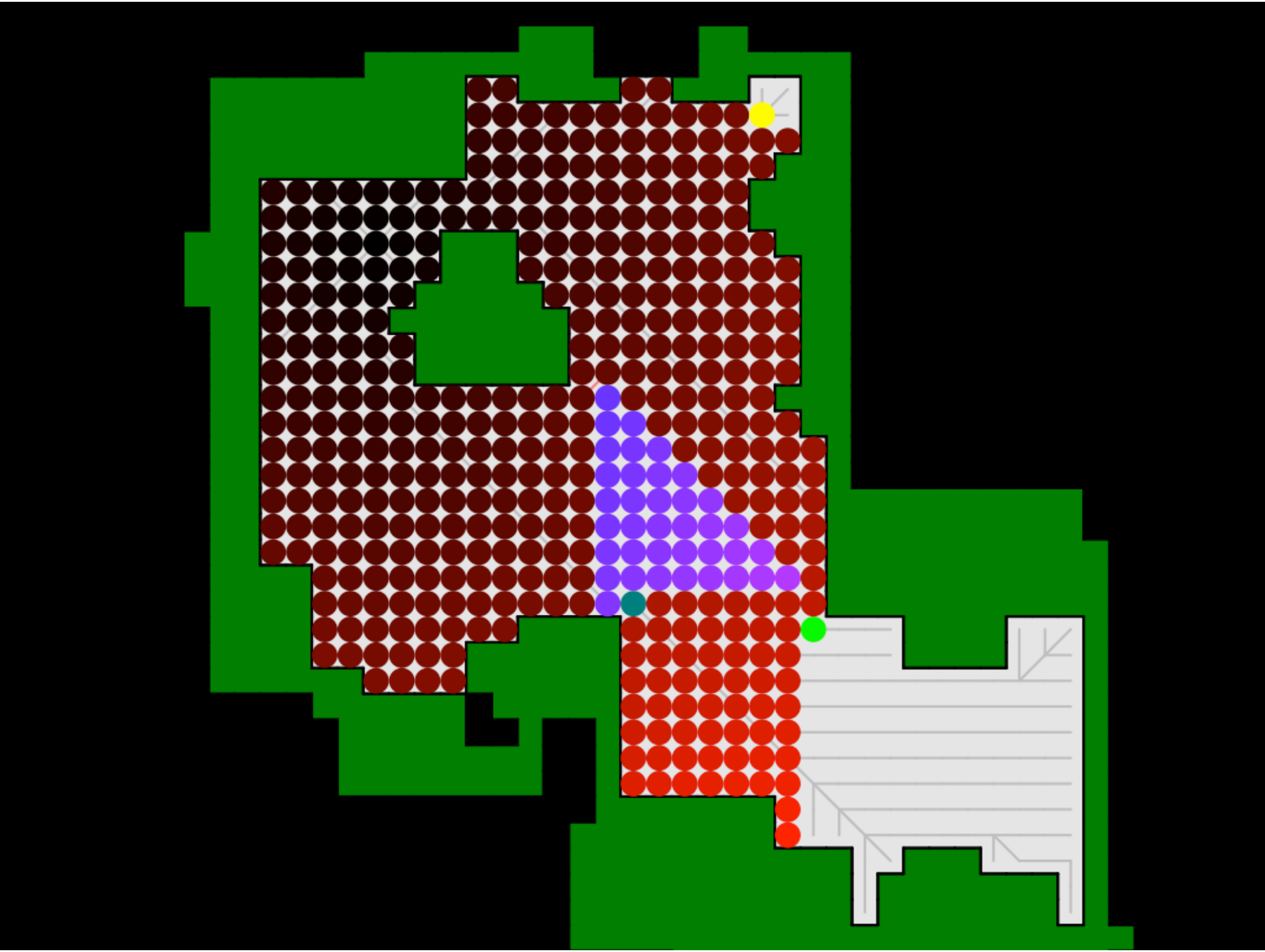


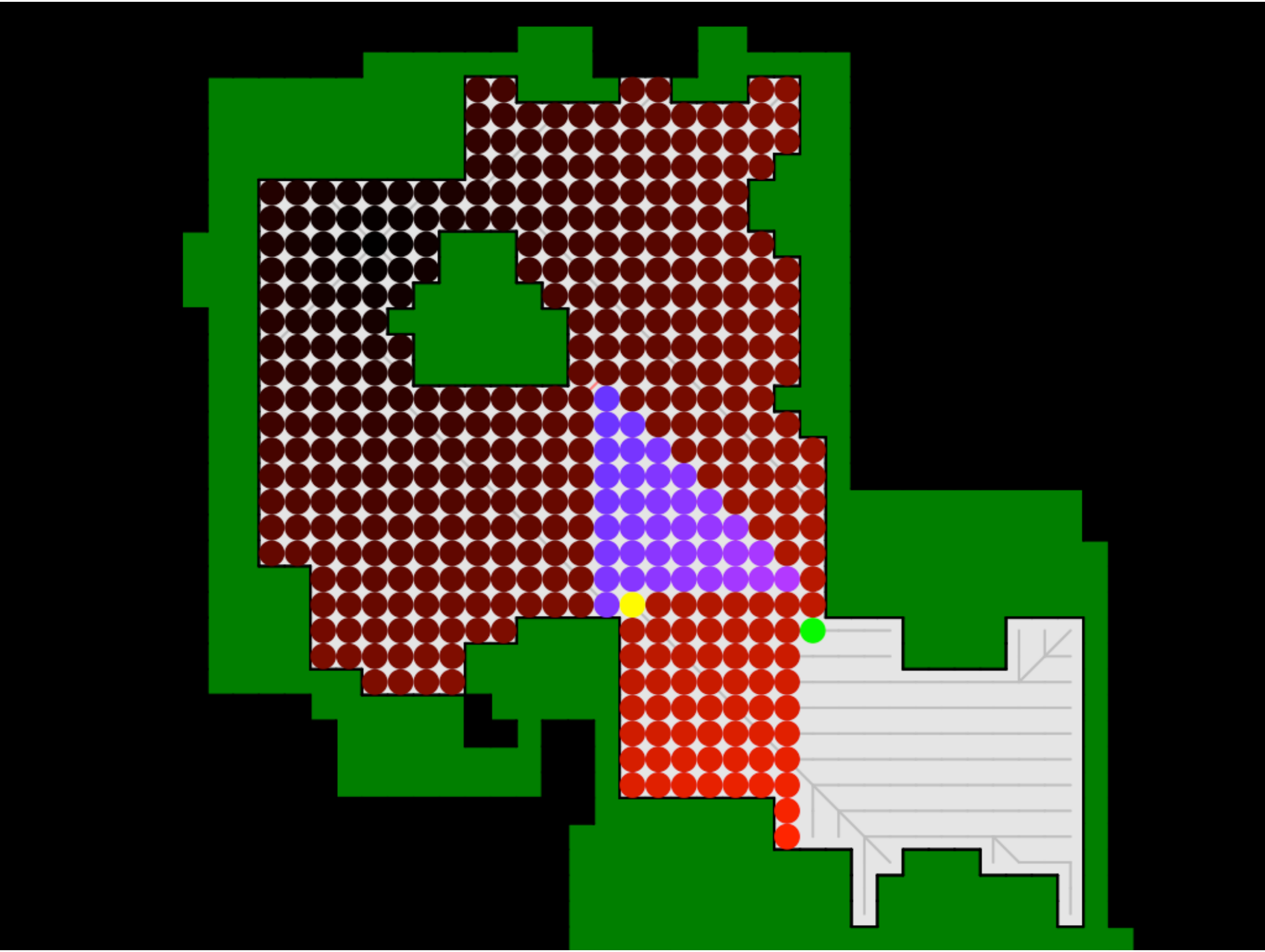


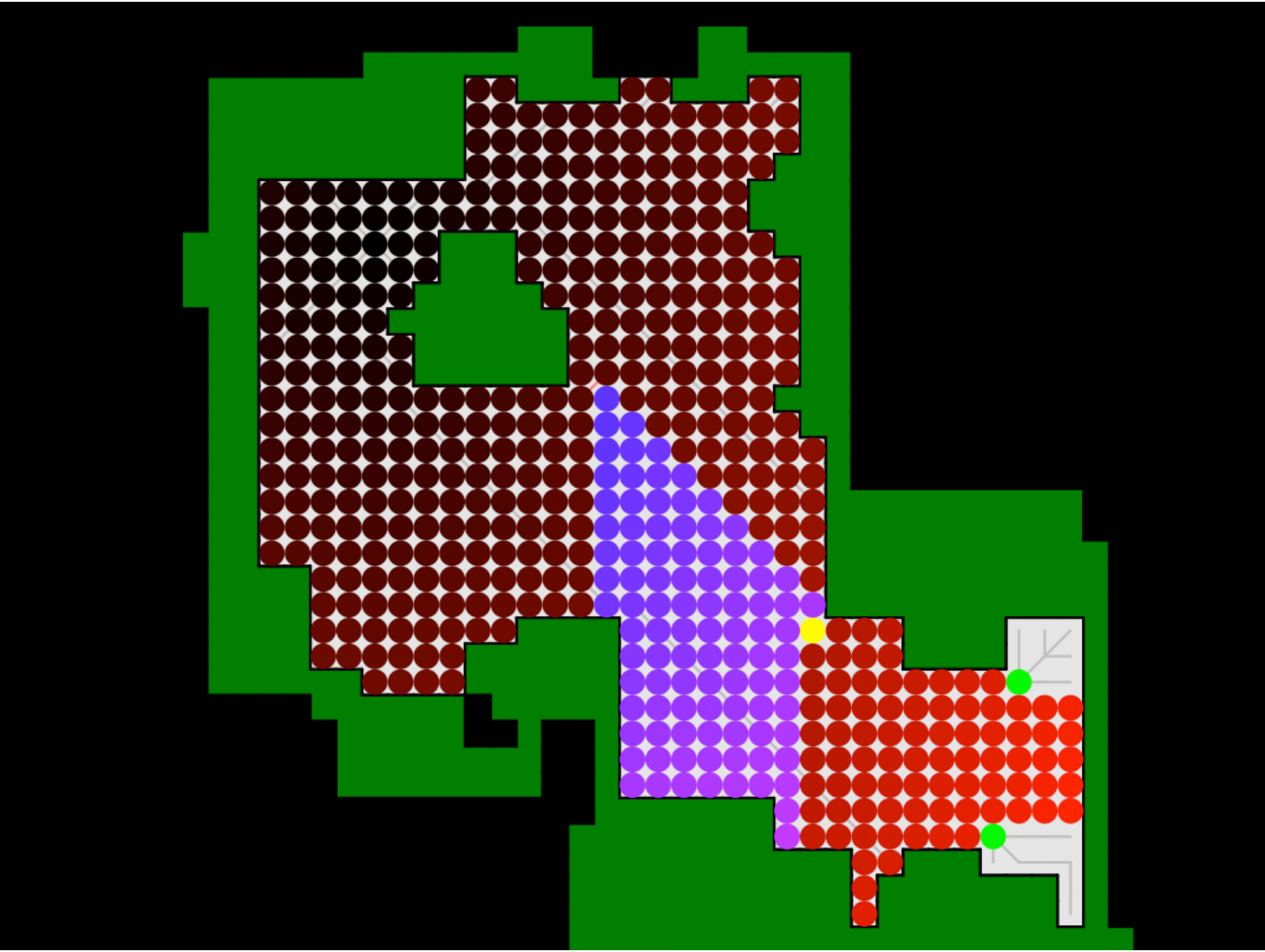


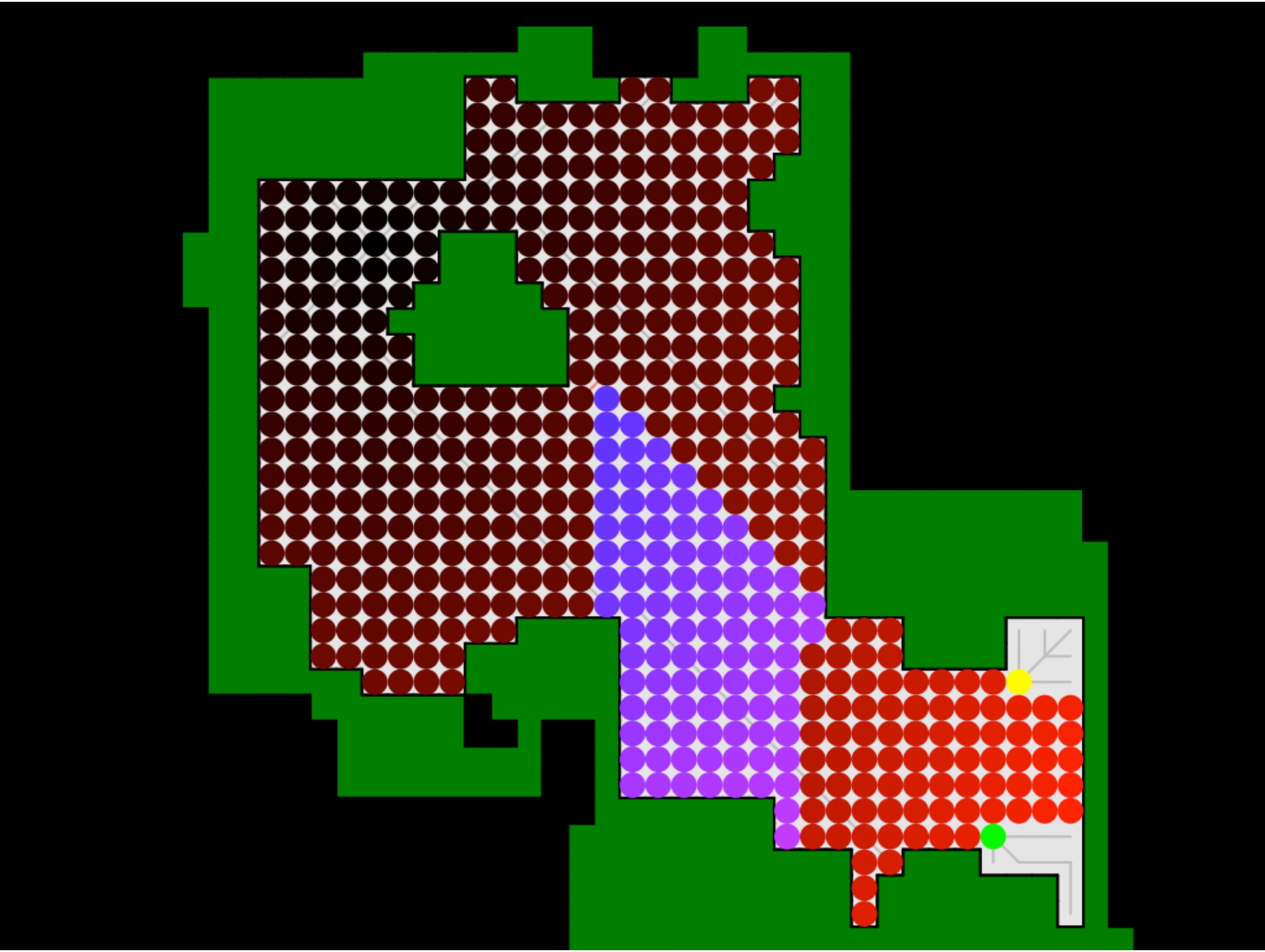


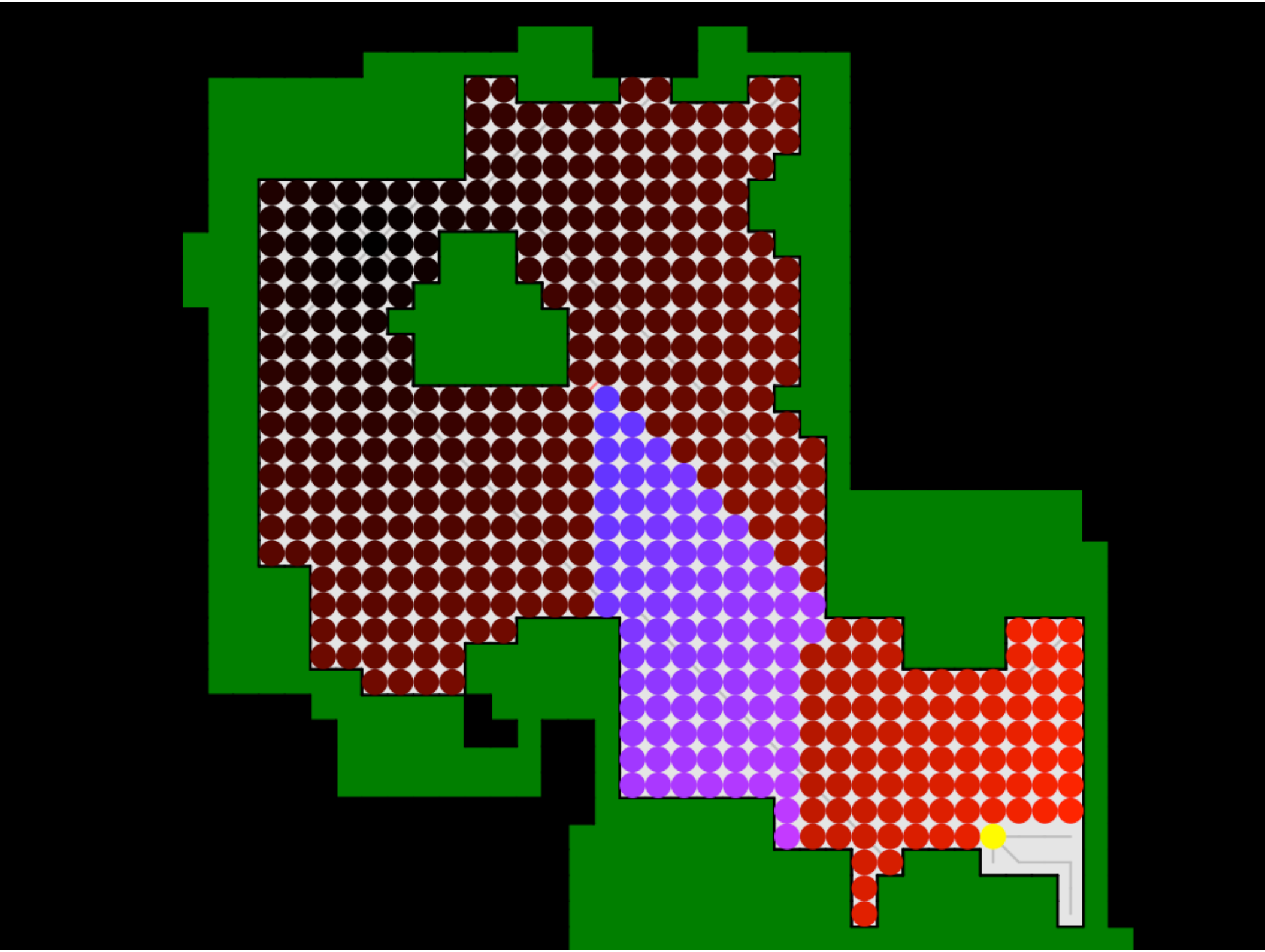


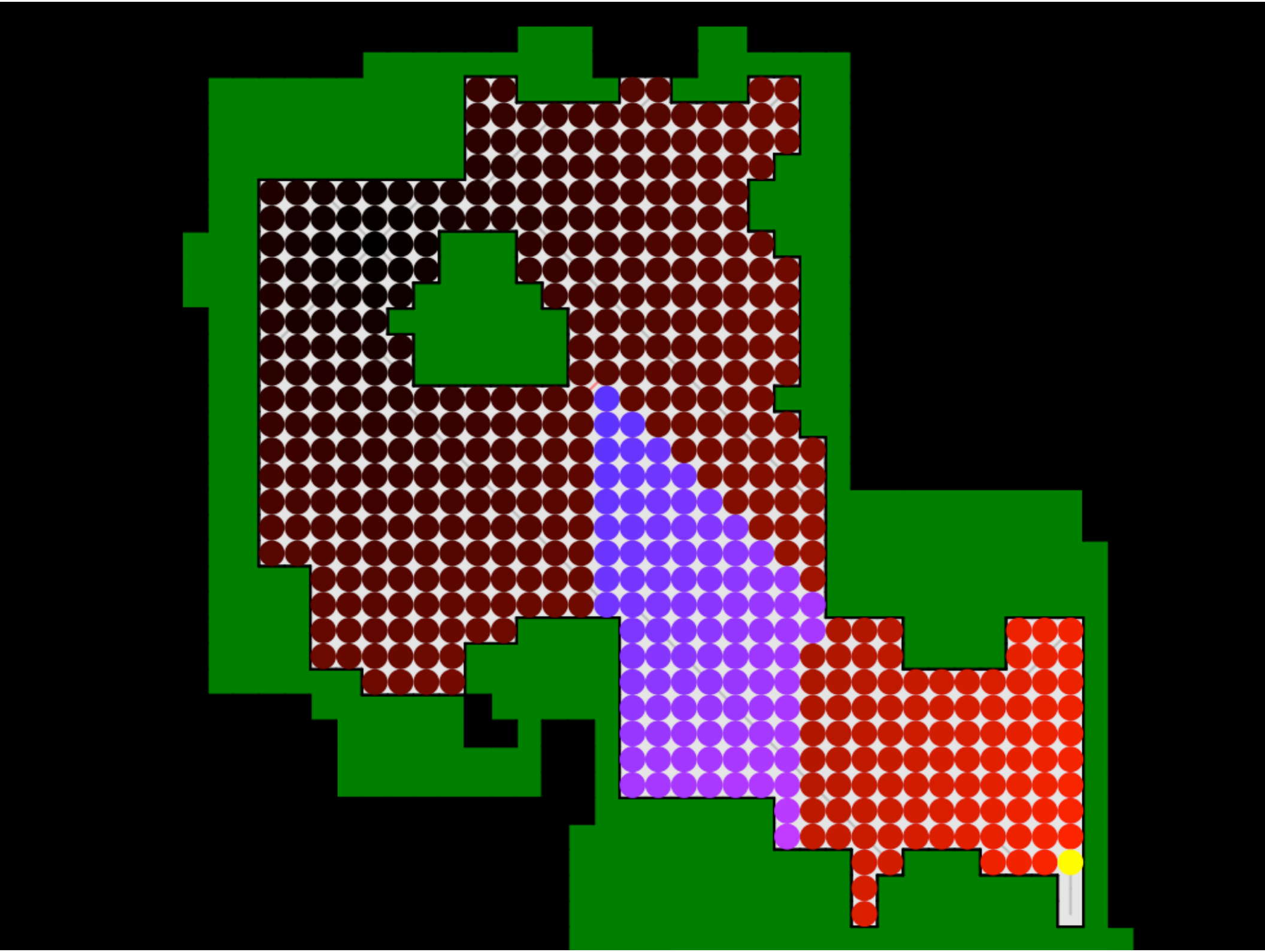


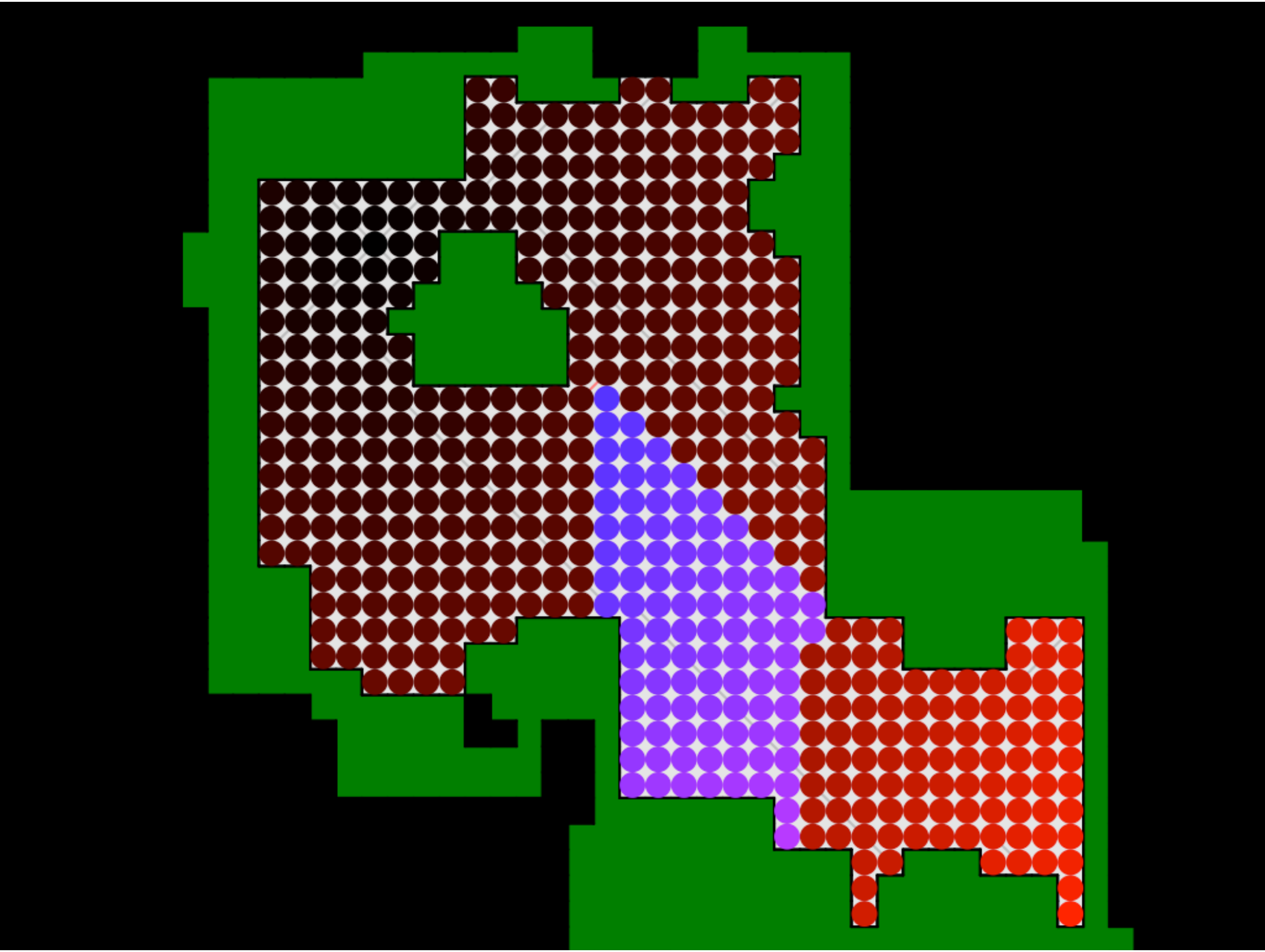












Experimental Results

Map	Dijkstra	CA*	Canonical Dijkstra
Dragon Age	1.0	2.1	3.4

Speedup Factor

Experimental Results

Map	Dijkstra	CA*	Canonical Dijkstra
Dragon Age	1.0	2.1	3.4
Random	1.0	1.7	2.5

Speedup Factor

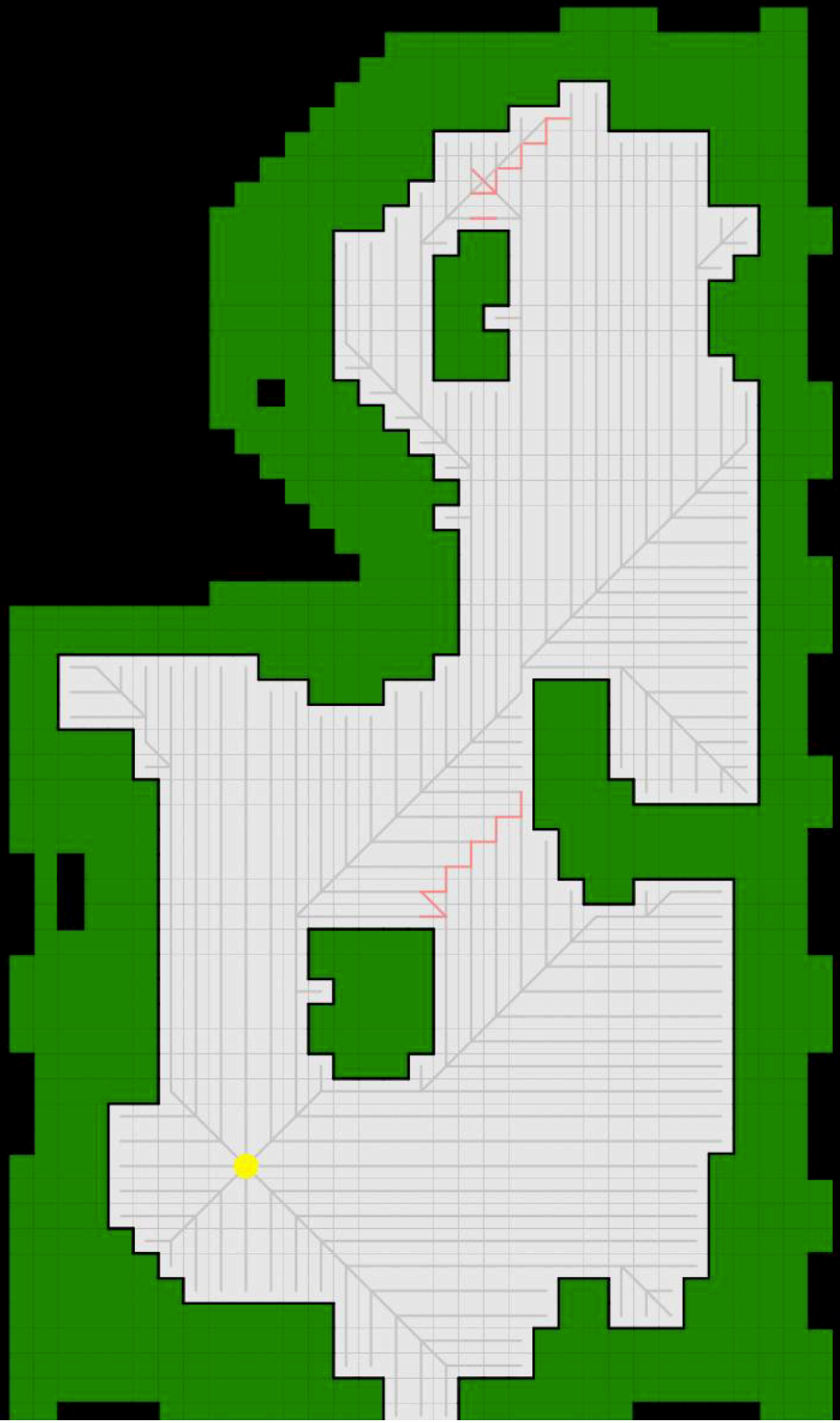
Experimental Results

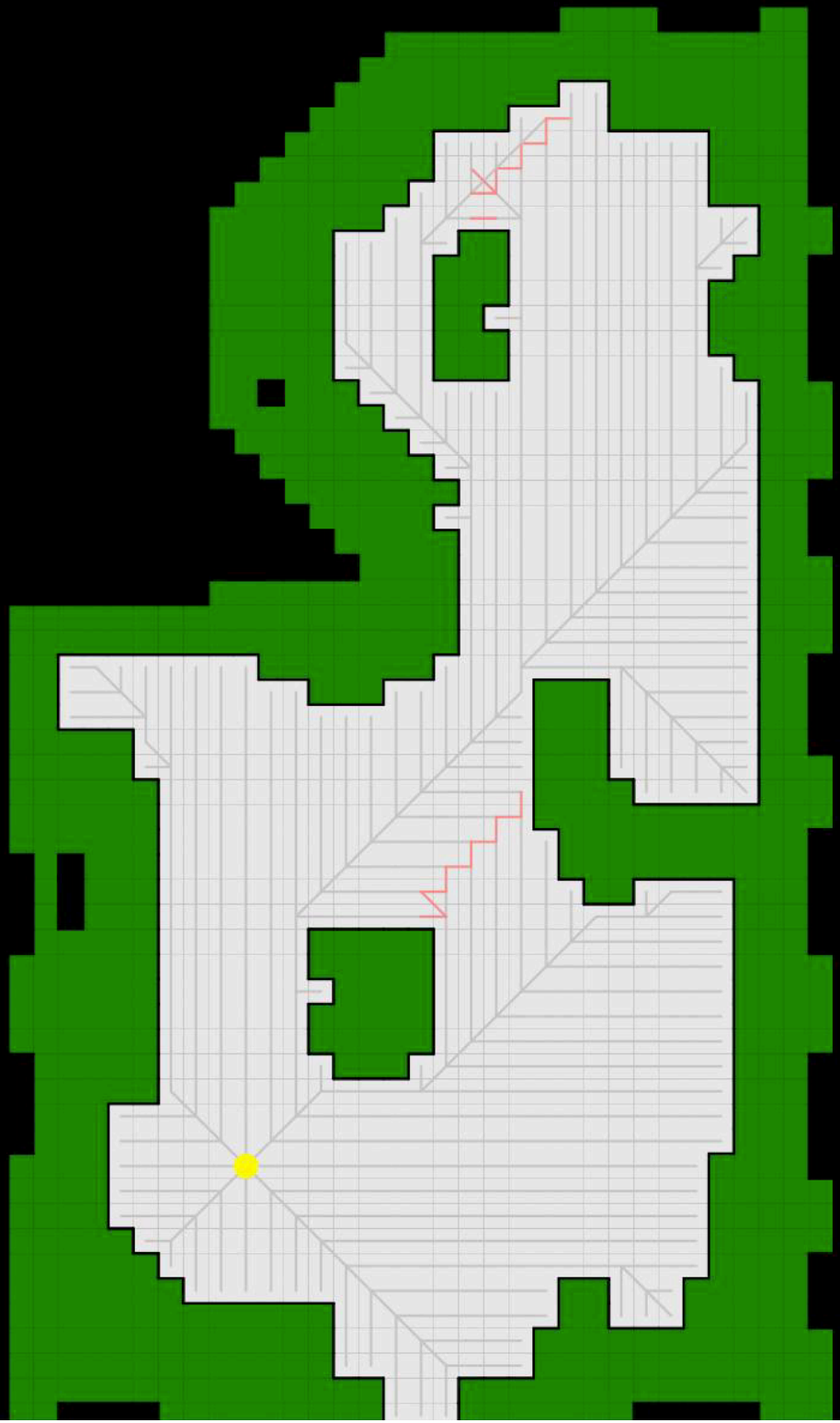
Map	Dijkstra	CA*	Canonical Dijkstra
Dragon Age	1.0	2.1	3.4
Random	1.0	1.7	2.5
Starcraft	1.0	2.2	4.0

Speedup Factor

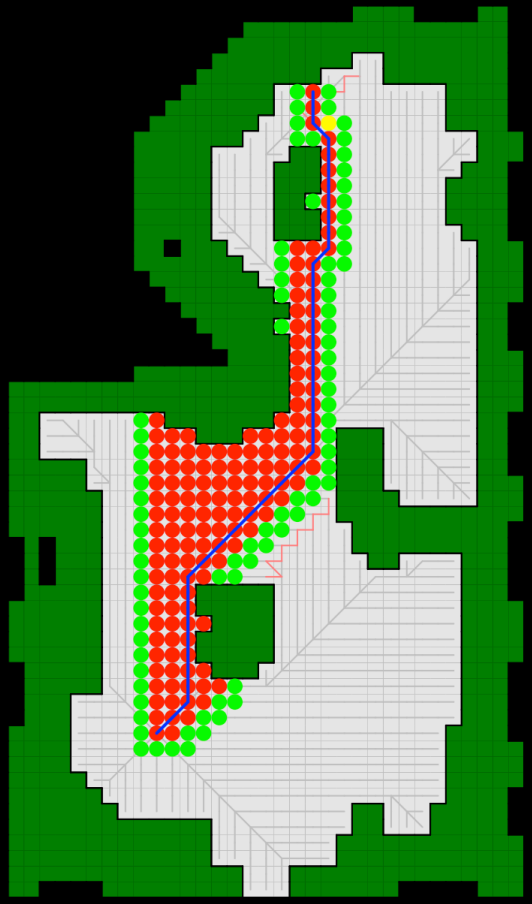
Algorithm #3: Bounded JPS

- Limit the length of jumps
- Parameterization between Canonical A* and JPS
  - No jumping is Canonical A*
  - Infinite jumping is JPS

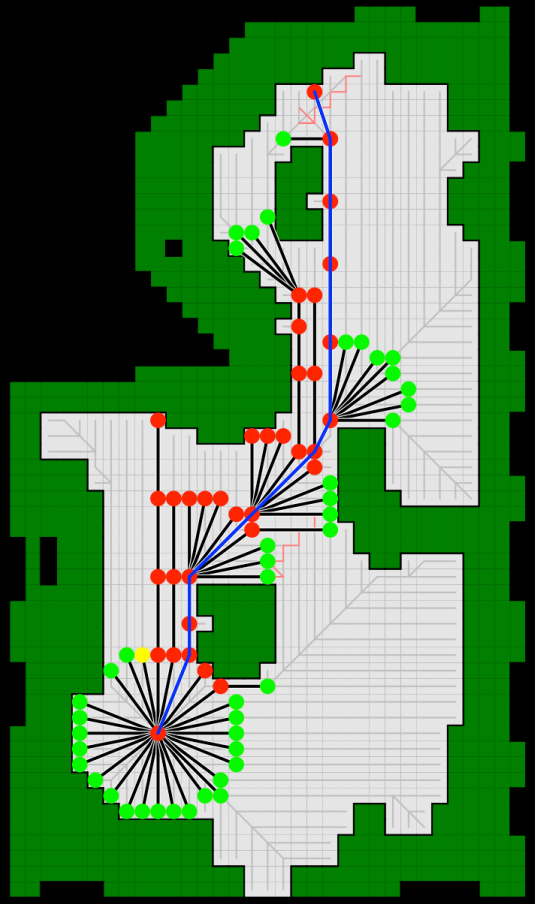




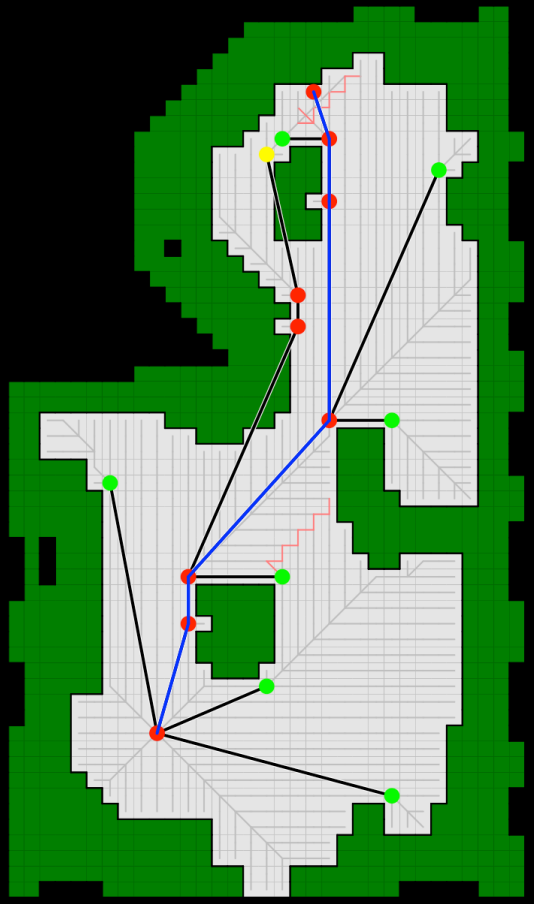
A*



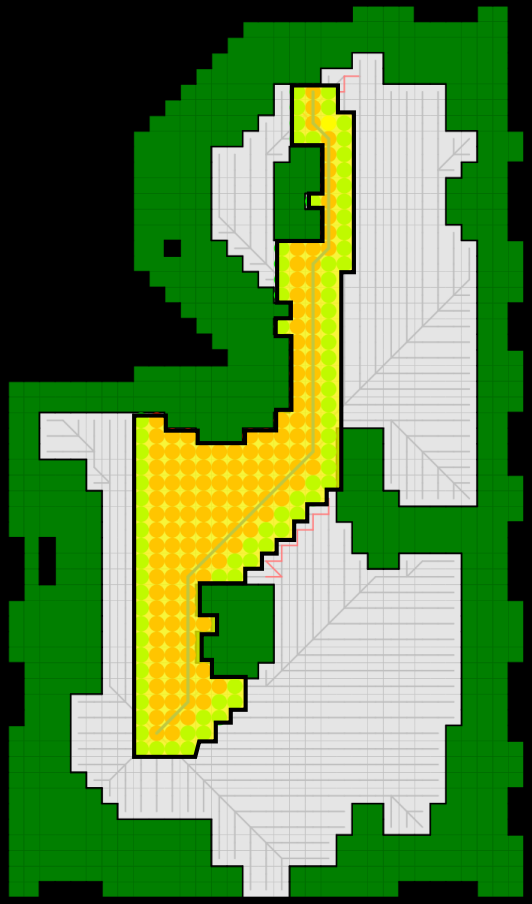
BJPS(4)



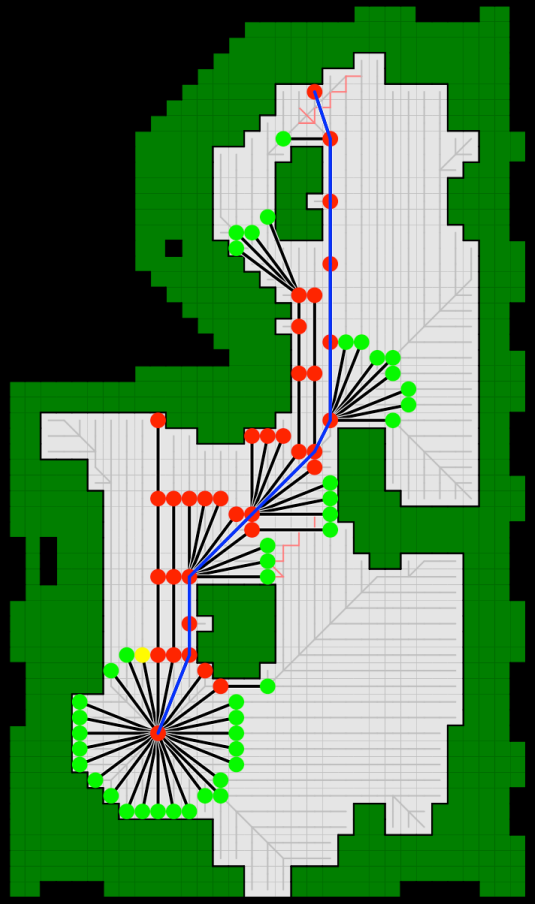
JPS



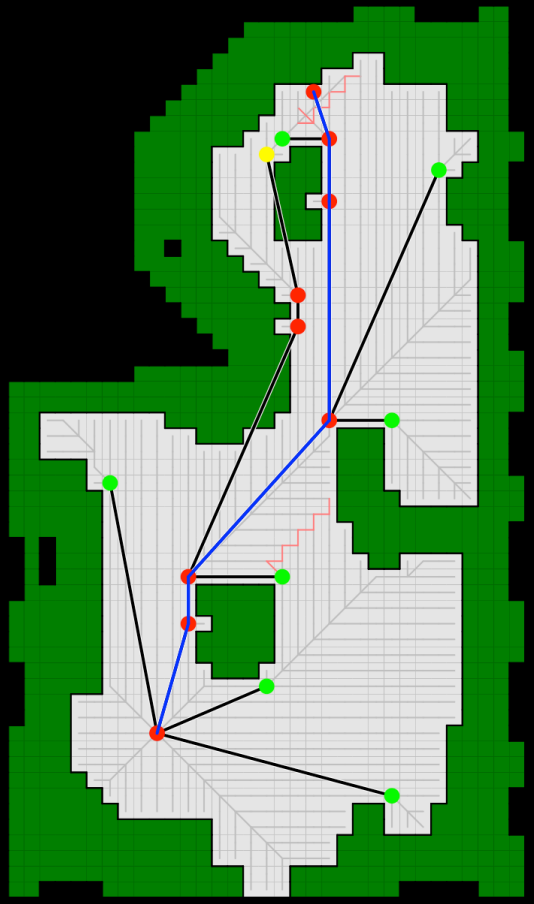
A*



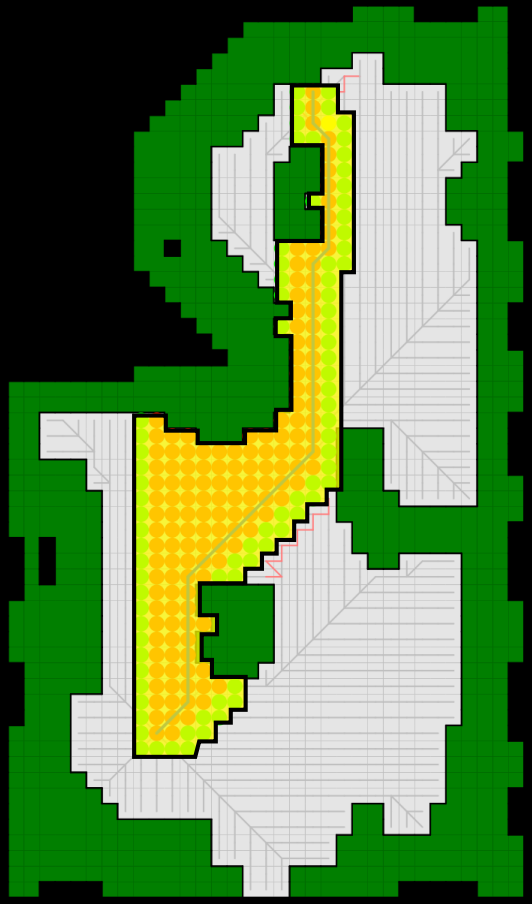
BJPS(4)



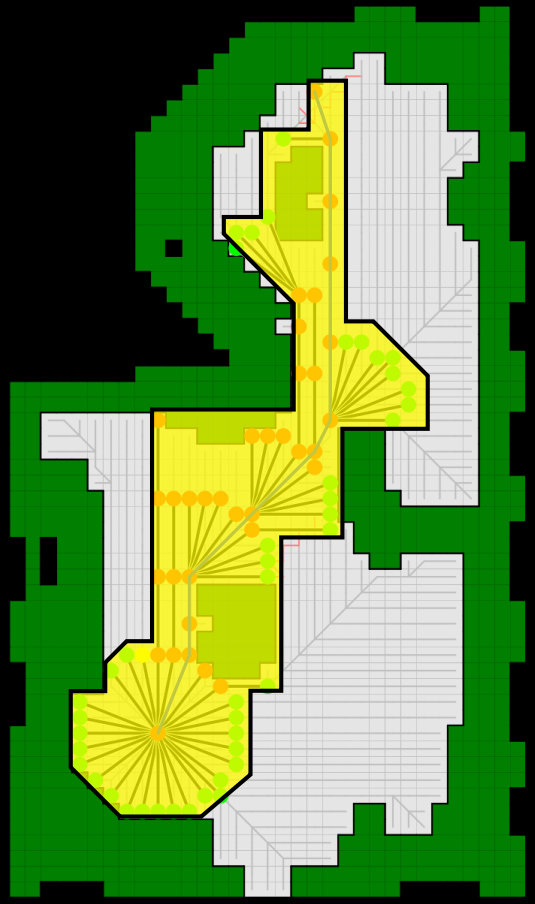
JPS



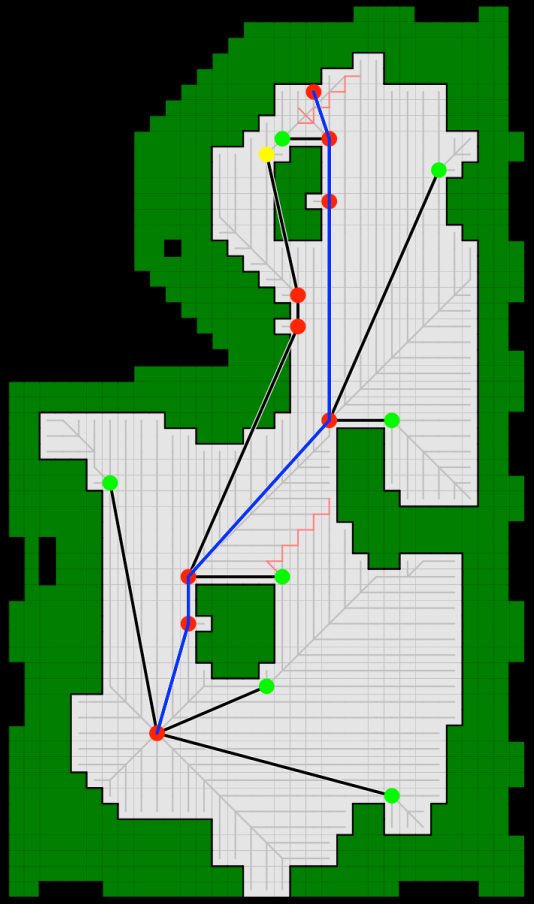
A*



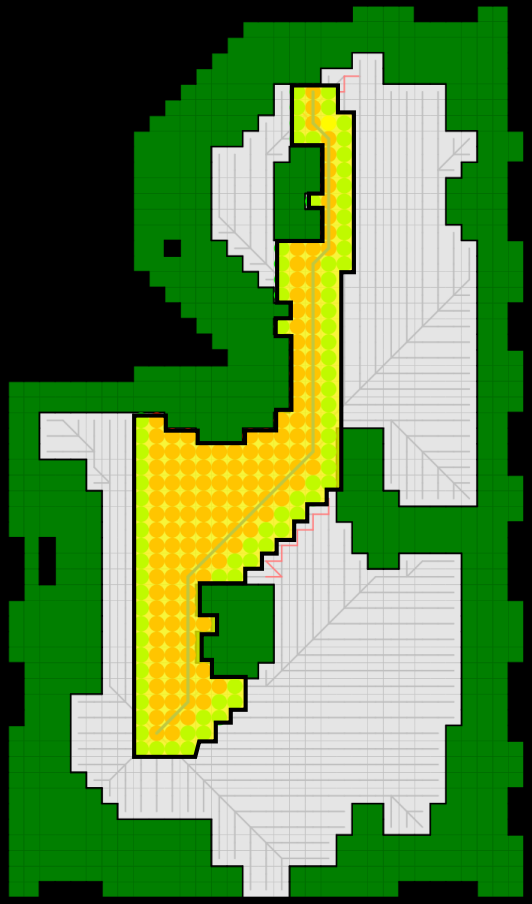
BJPS(4)



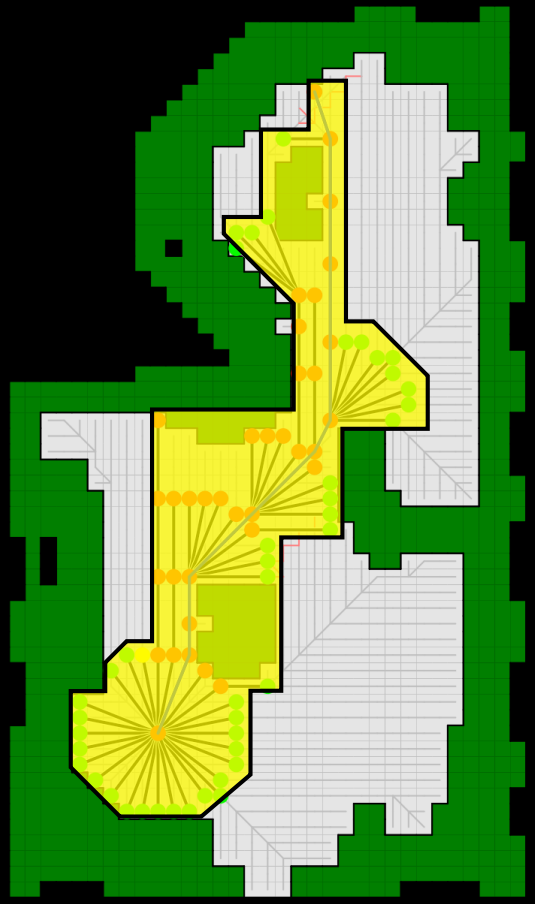
JPS



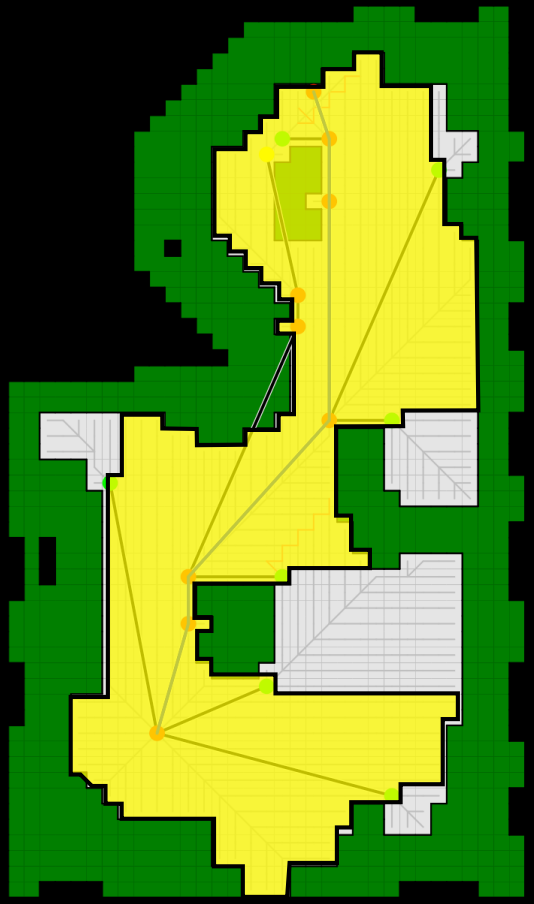
A*



BJPS(4)



JPS



Experimental Results

	CA*	BJPS(4)	JPS
Time (ms)	2.59	1.50	1.98

Experimental Results

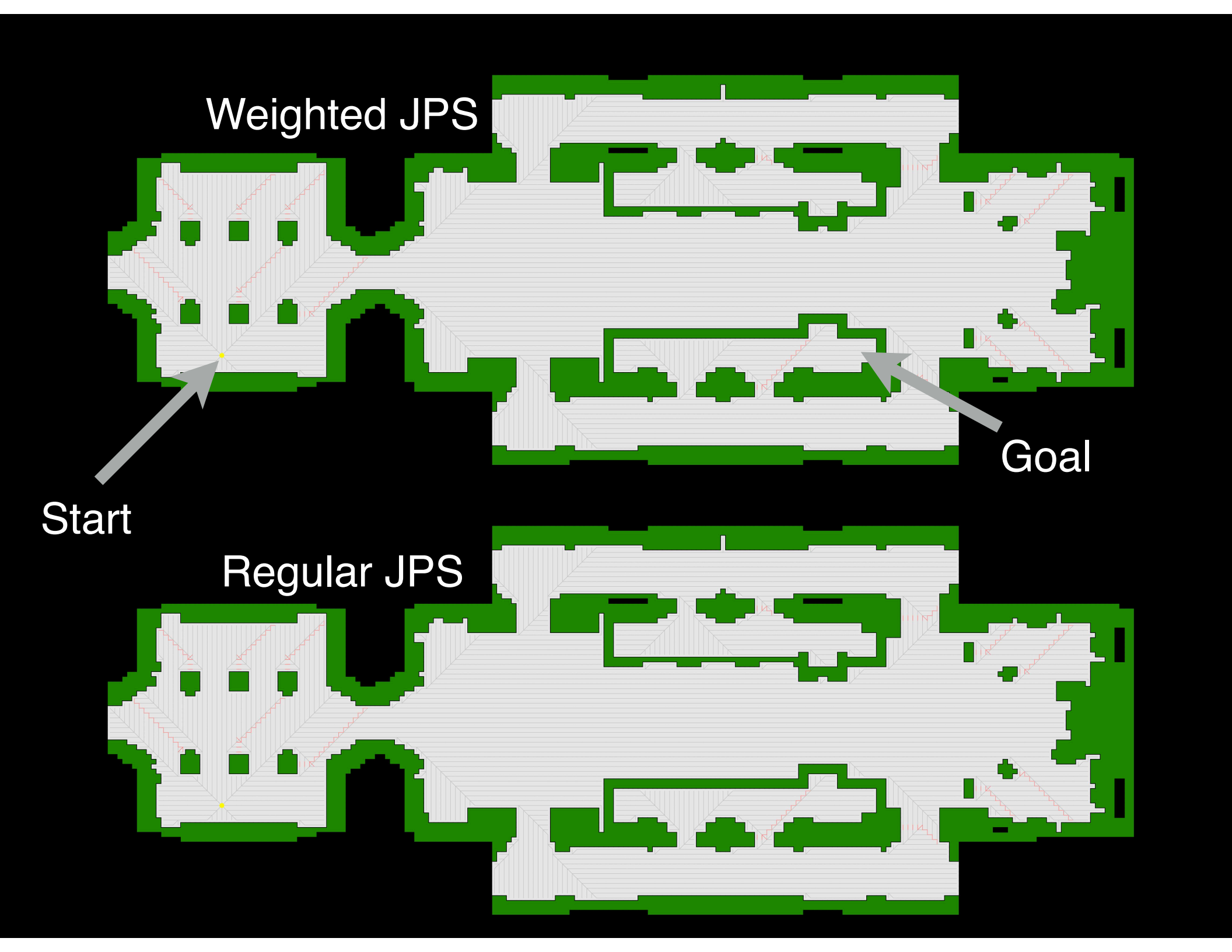
	CA*	BJPS(4)	JPS
Time (ms)	2.59	1.50	1.98
Expansions	13,301	4,091	229

Experimental Results

	CA*	BJPS(4)	JPS
Time (ms)	2.59	1.50	1.98
Expansions	13,301	4,091	229
Generations	13,654	22,363	61,281

Best first search

- We can use suboptimal algorithms (eg Weighted A*) to search with JPS
- JPS only puts onto OPEN:
  - Jump points
  - The goal


Weighted JPS
Start
Goal
Regular JPS


Weighted JPS
Regular JPS

Conclusions

- By decomposing JPS we:
  - Gain a better understanding of JPS
  - Are able to introduce new algorithms using the ideas of JPS
    - Bounded JPS
    - Canonical A*, Canonical Dijkstra
    - Weighted JPS

- http://www.movingai.com/