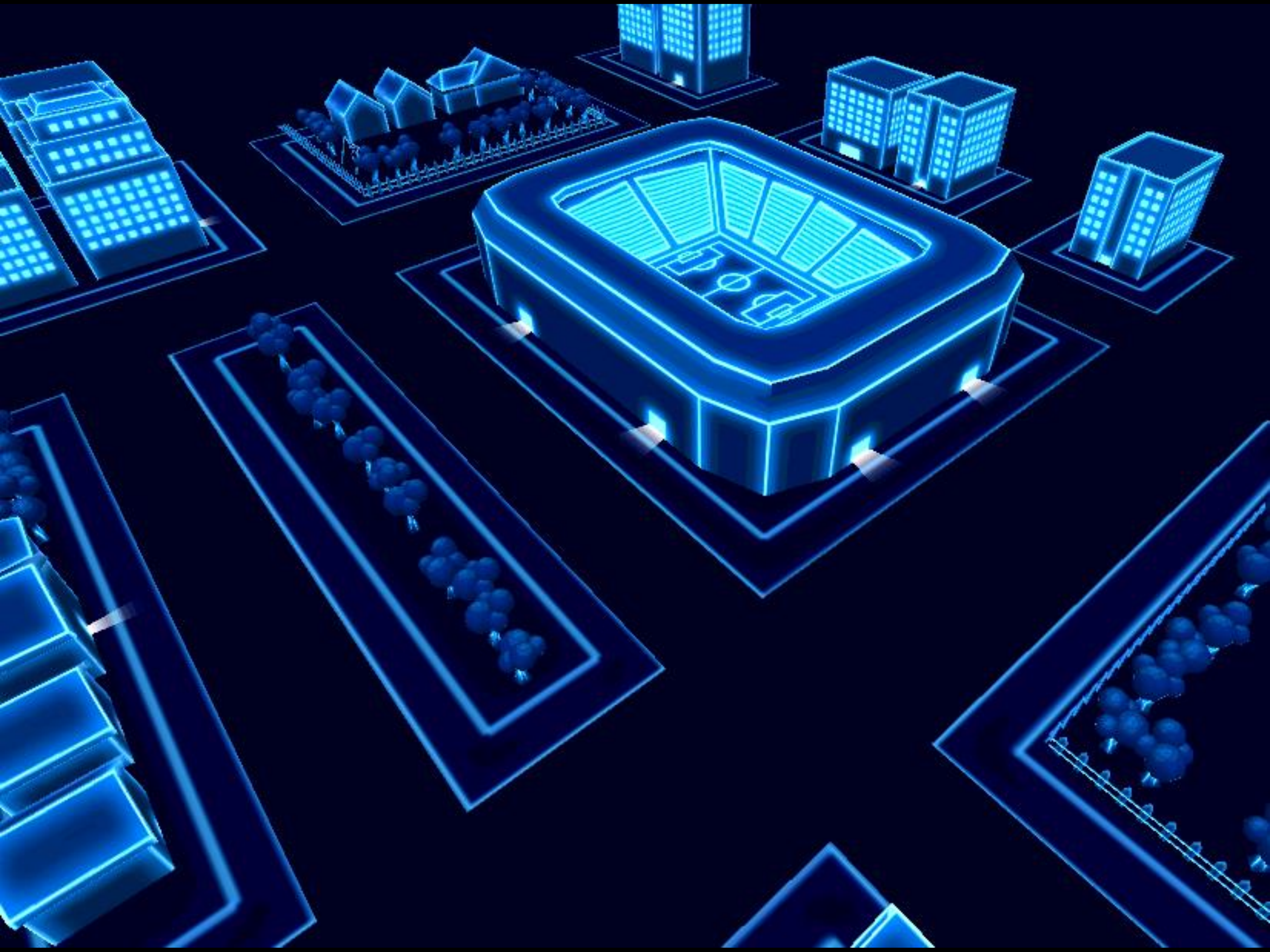
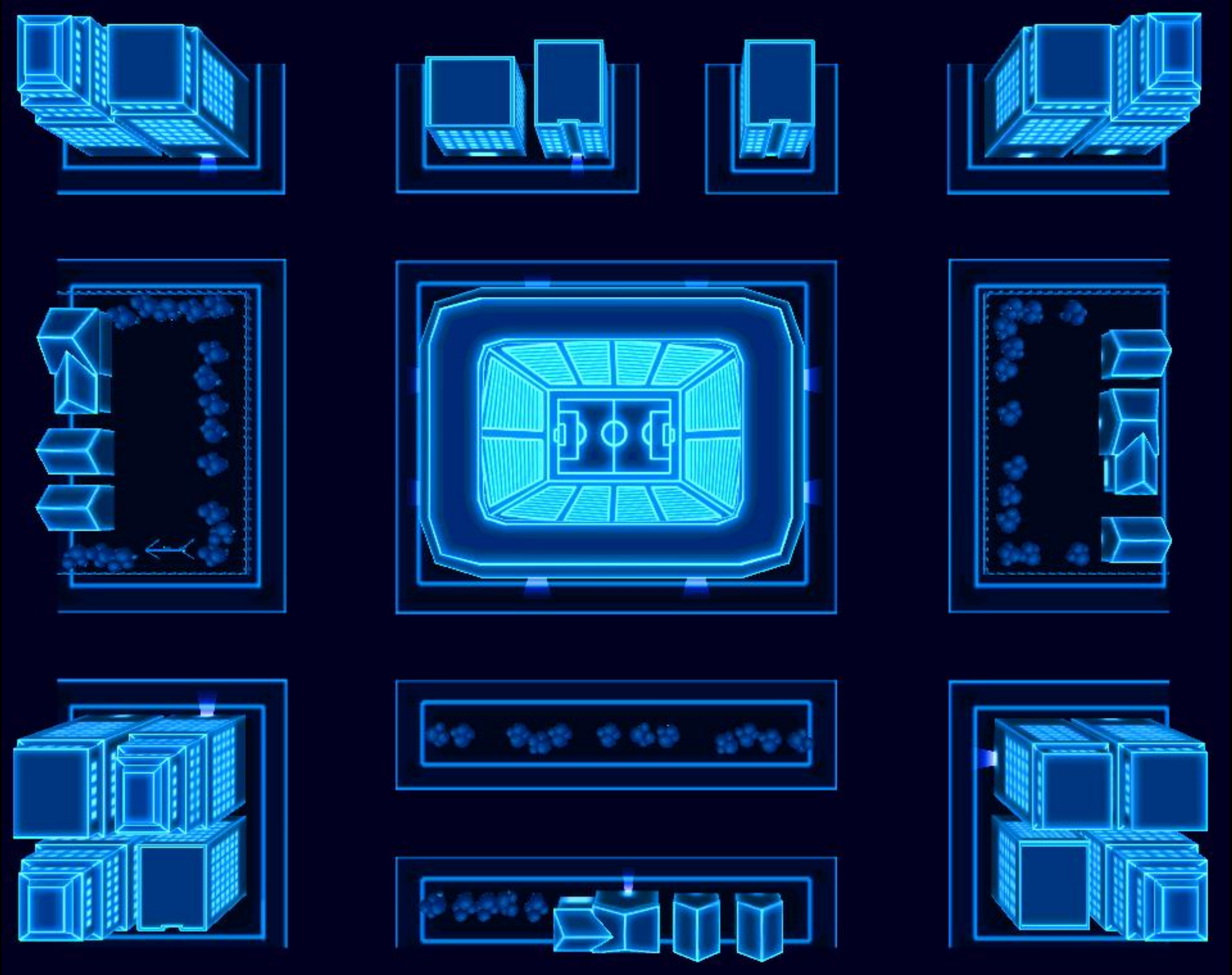
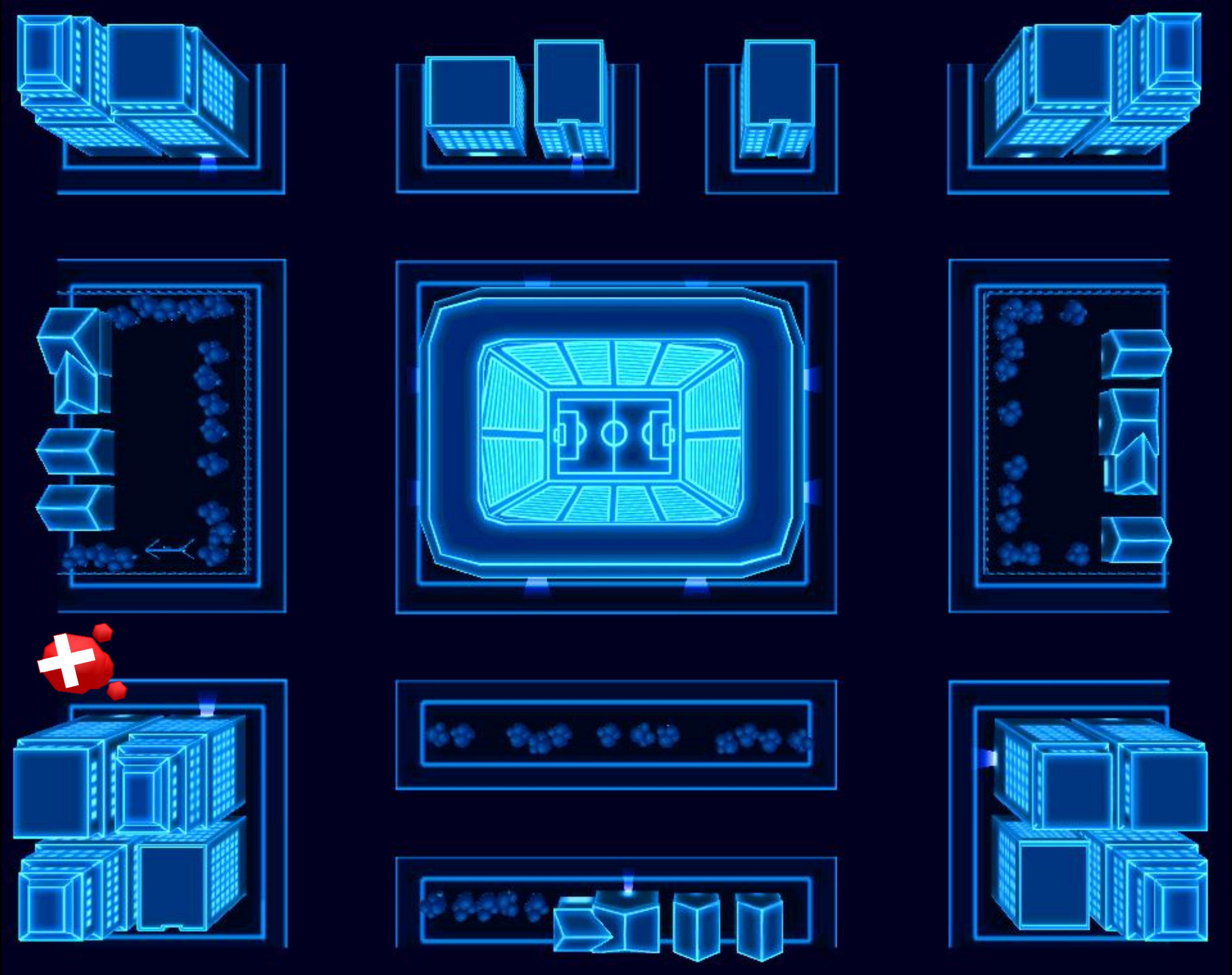
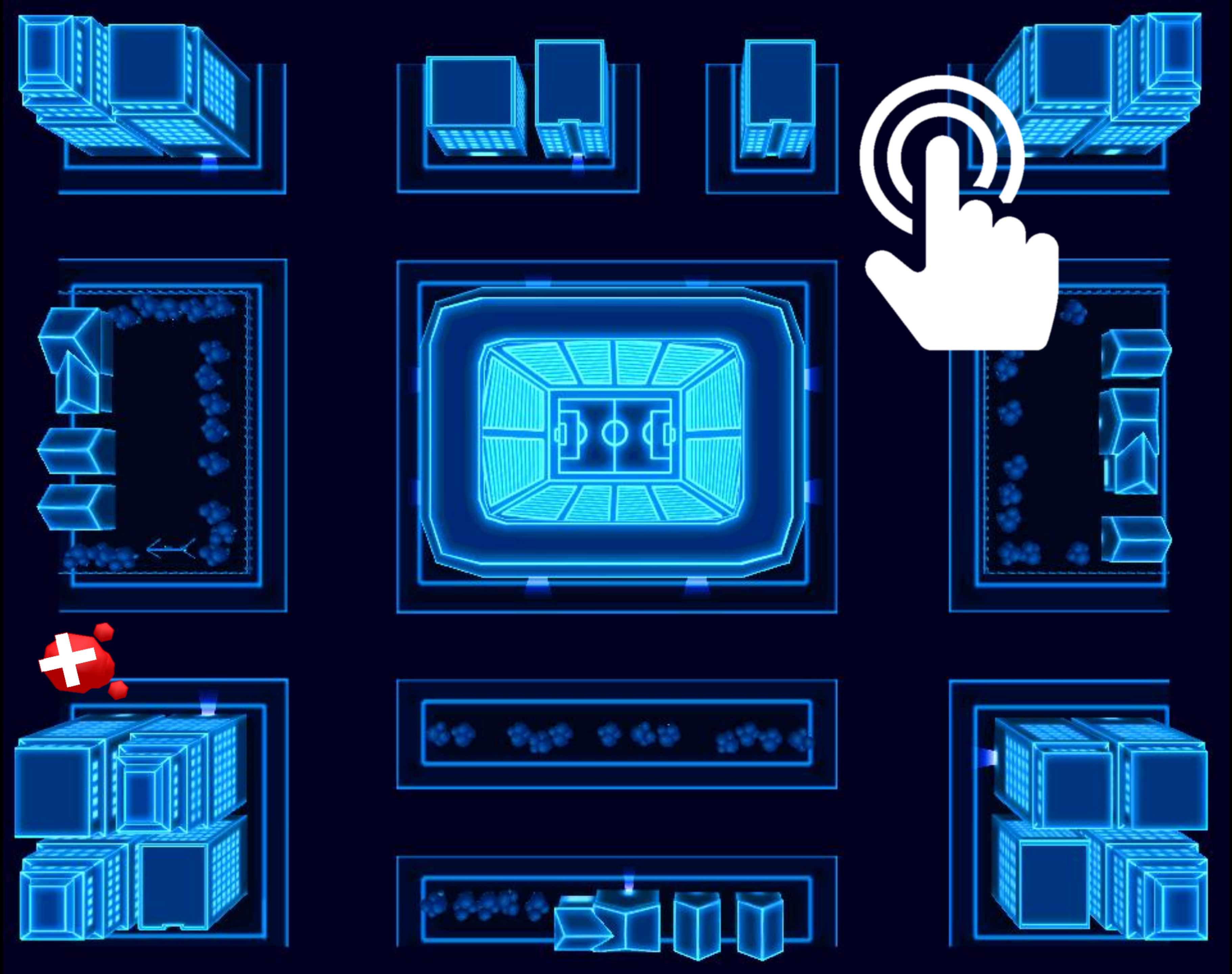
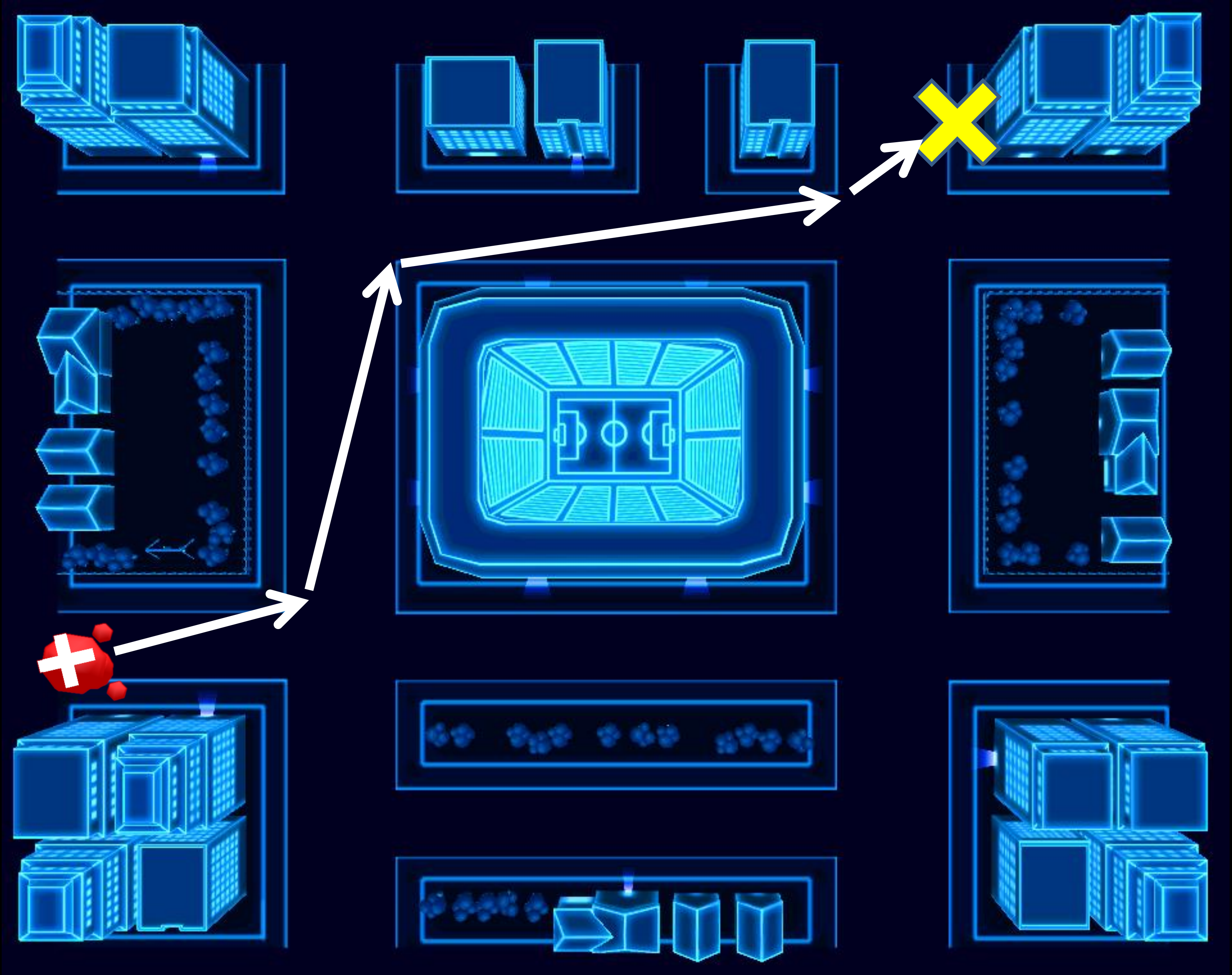
# Algorithms in Games: Any-Angle Pathfinding in Open Spaces

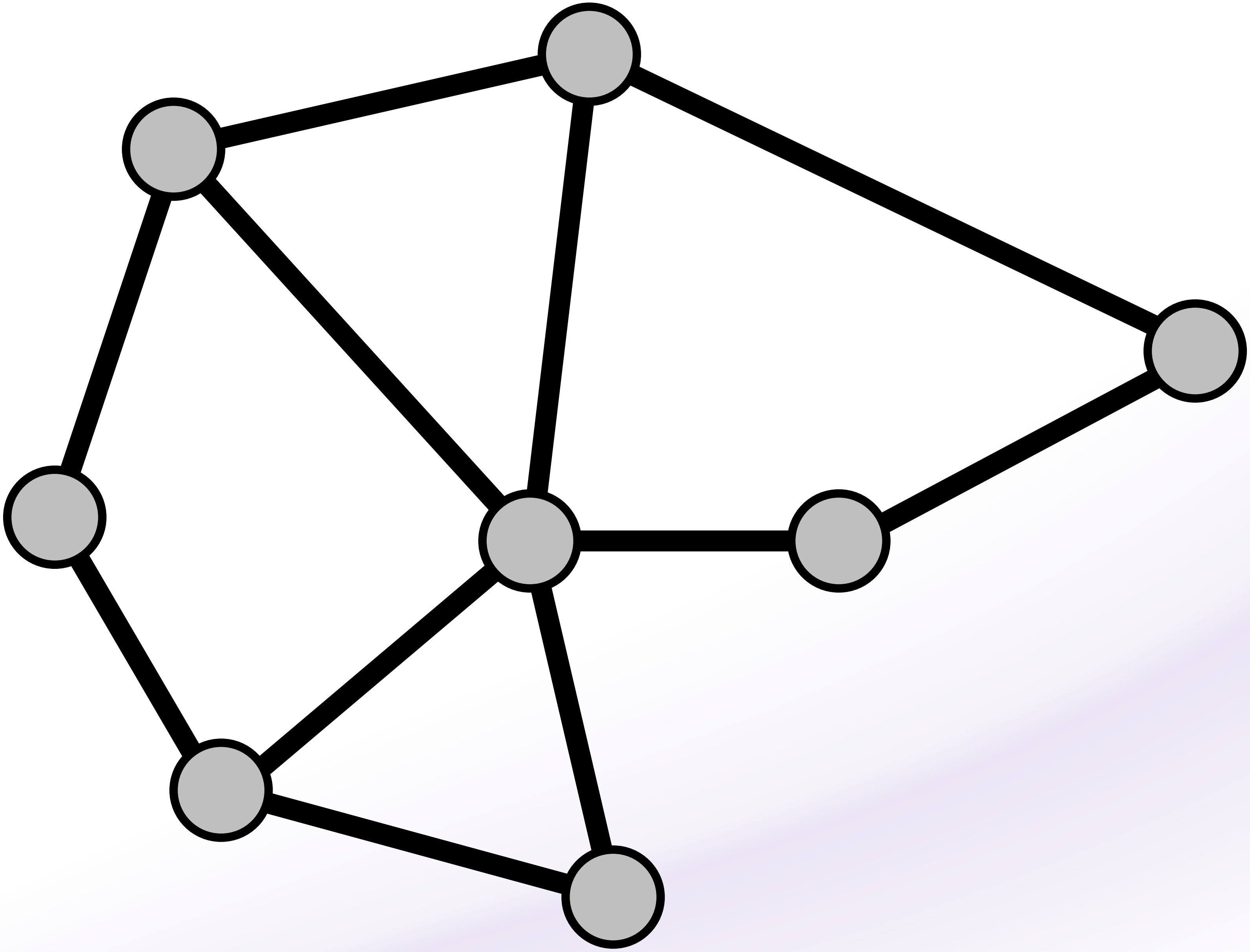


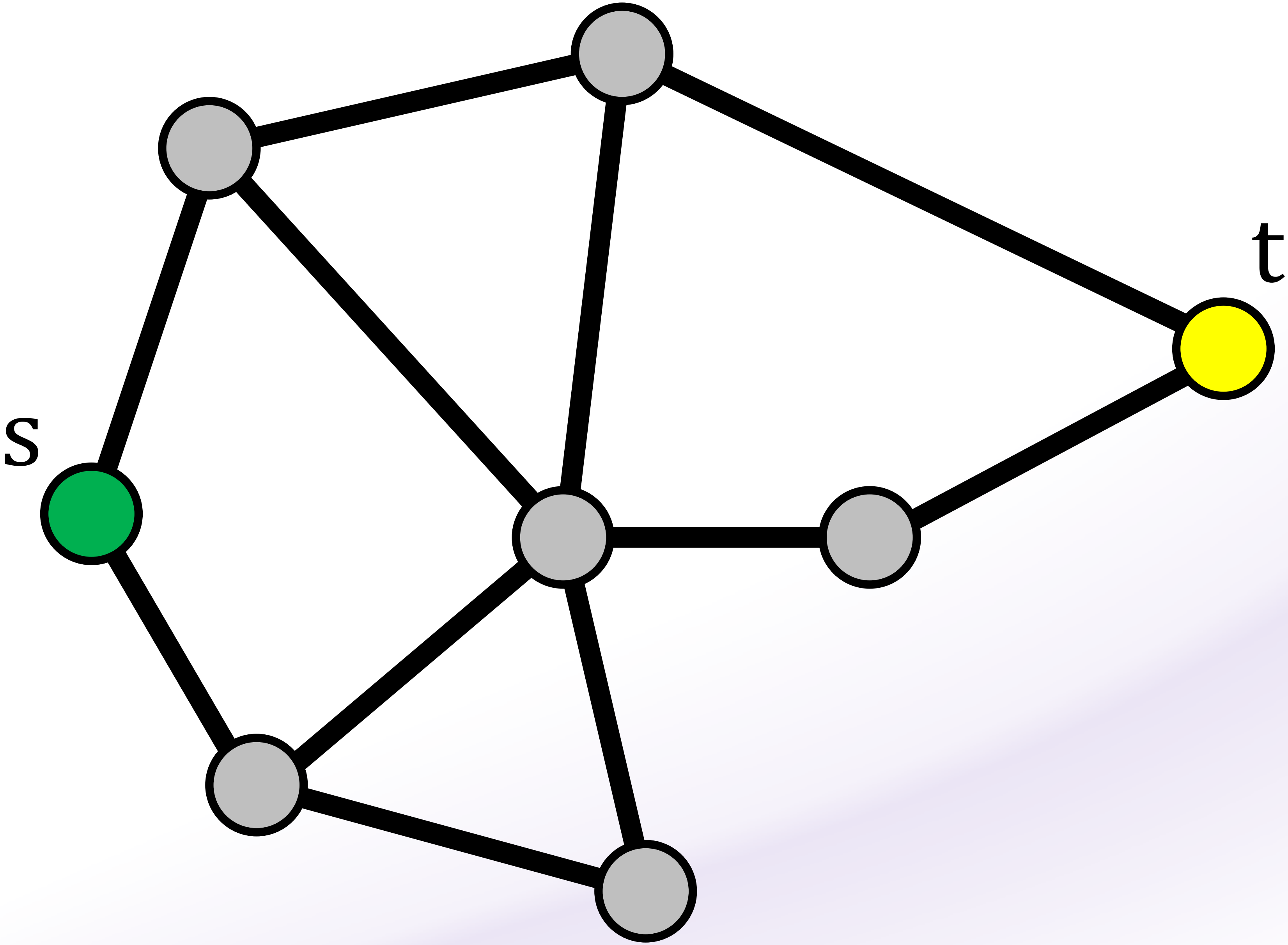


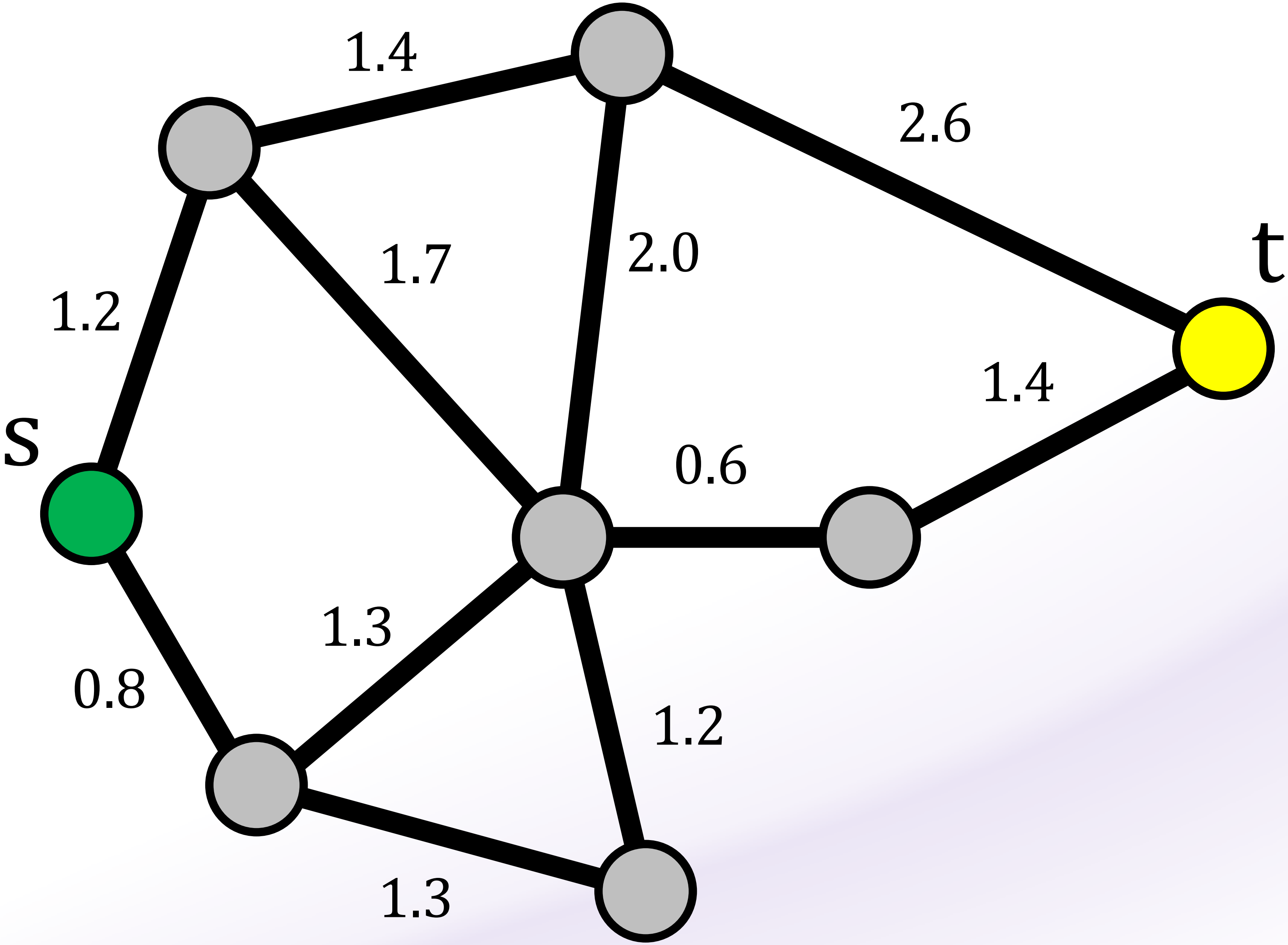


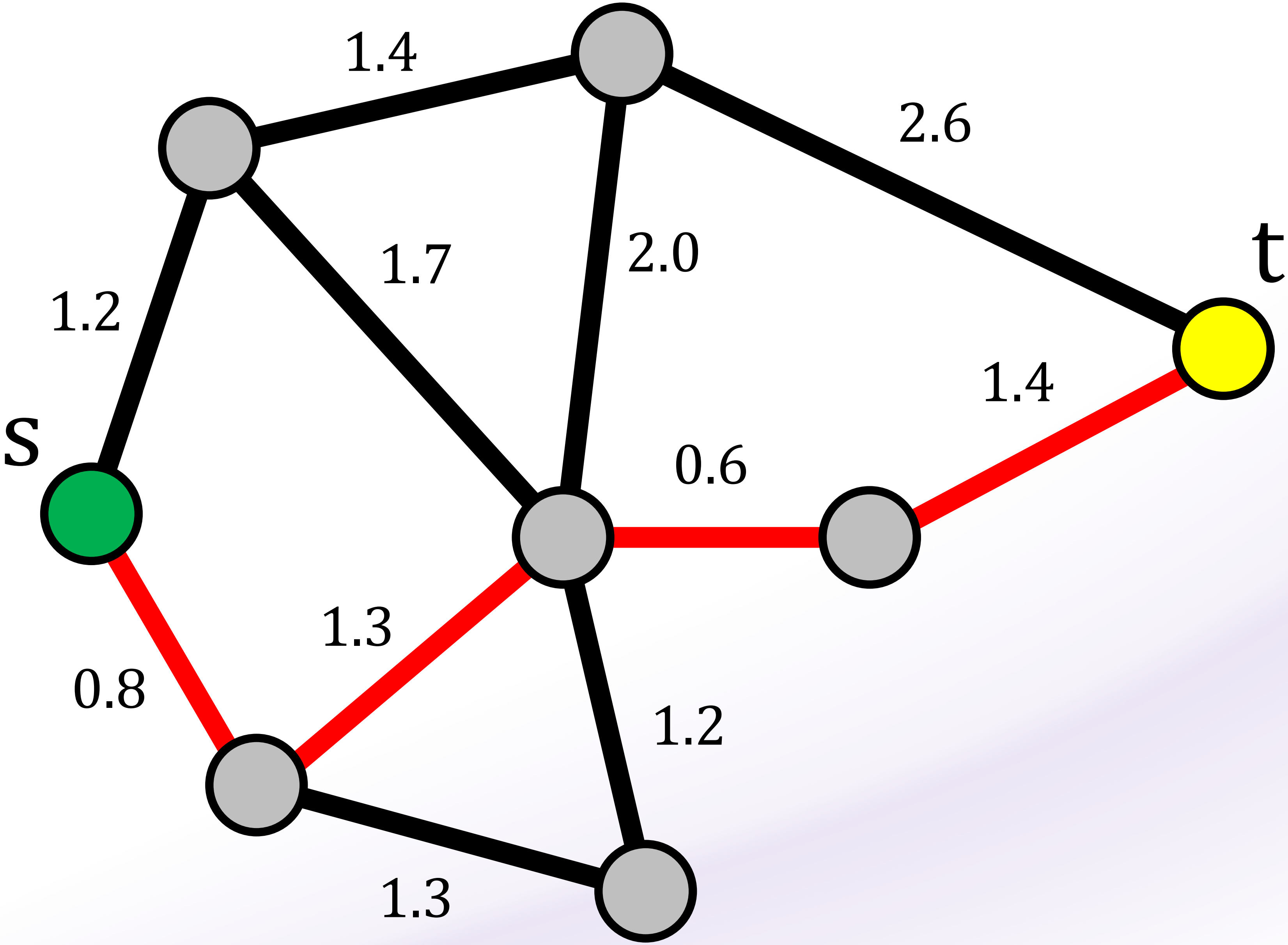


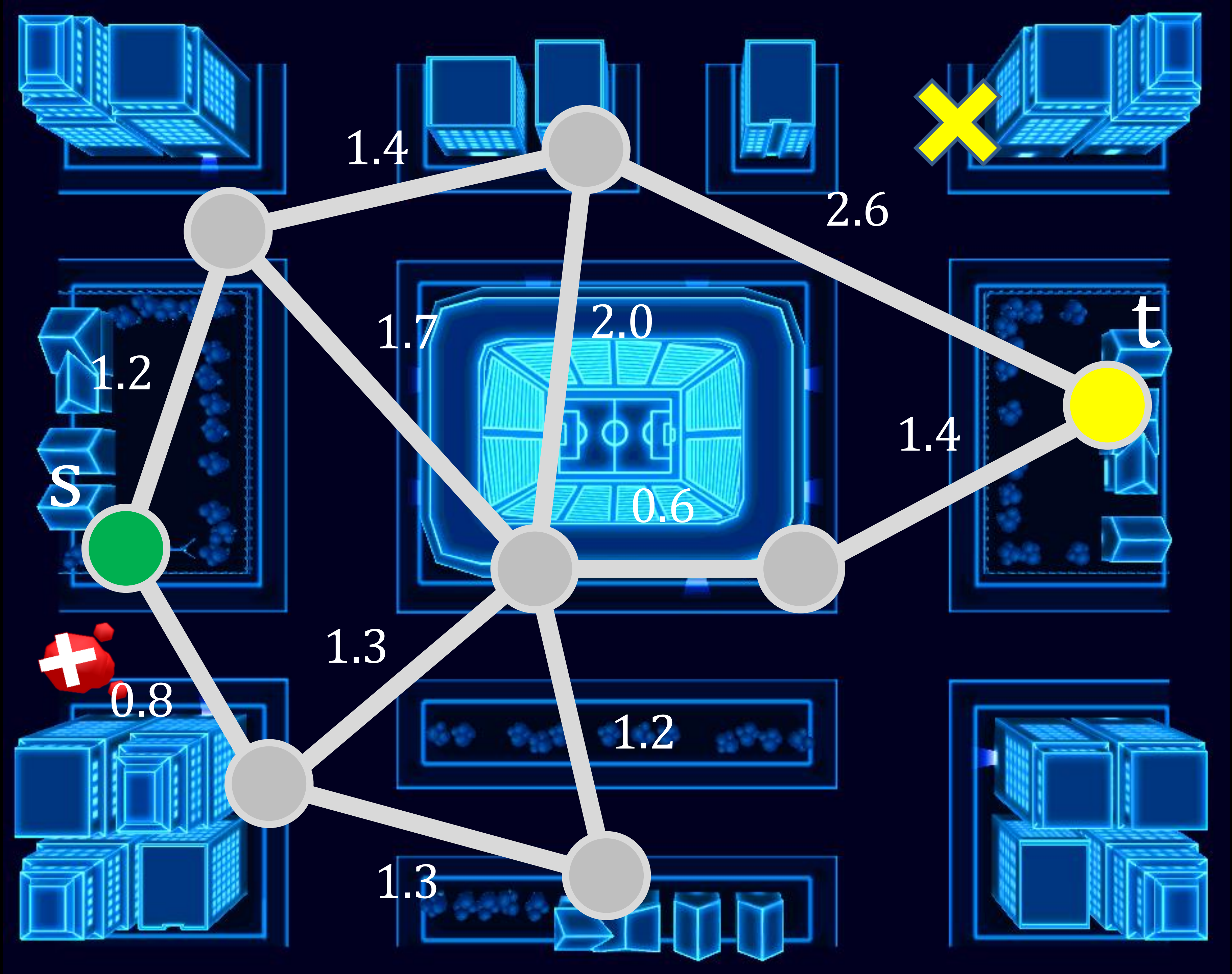


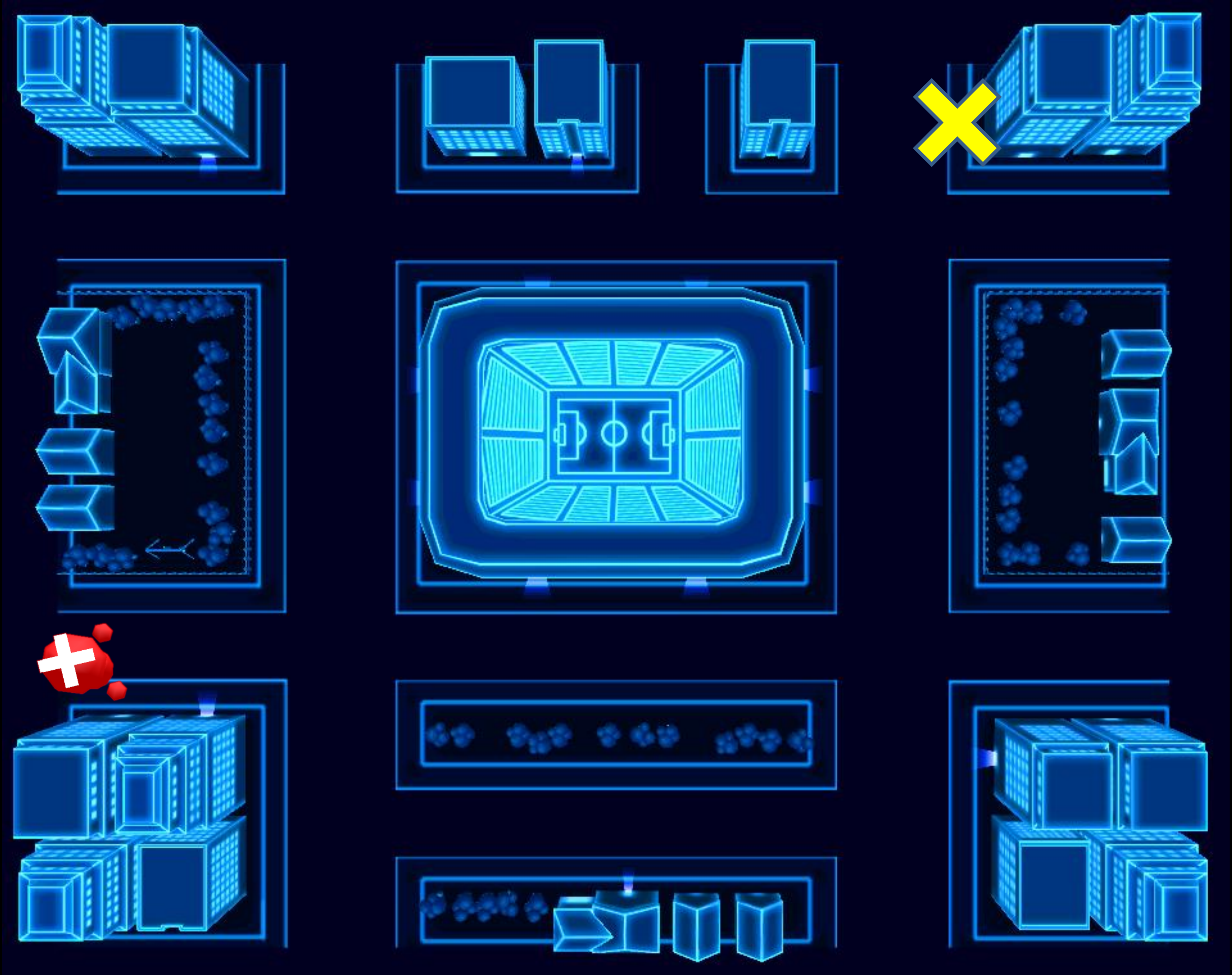



s
t

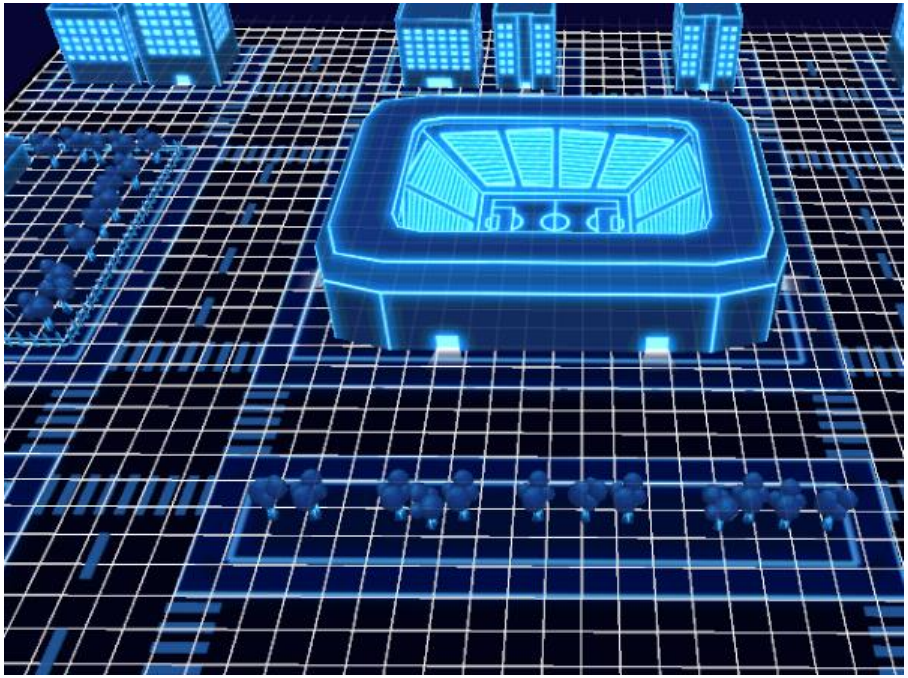

1.4
2.6
t
1.7
2.0
1.2
1.4
s
0.6
1.3
0.8
1.2
1.3


1.4
2.6
t
1.2
1.7
2.0
1.4
s
0.6
1.3
0.8
1.2
1.3

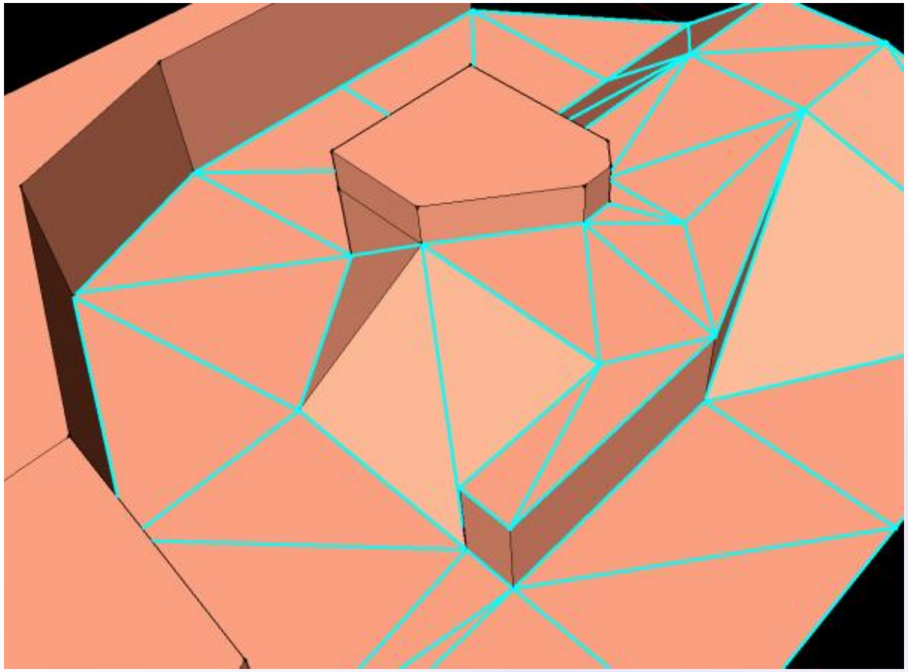

1.4
2.6
1.7
2.0
t
1.2
1.4
s
0.6
1.3
0.8
1.2
1.3



**Grids**

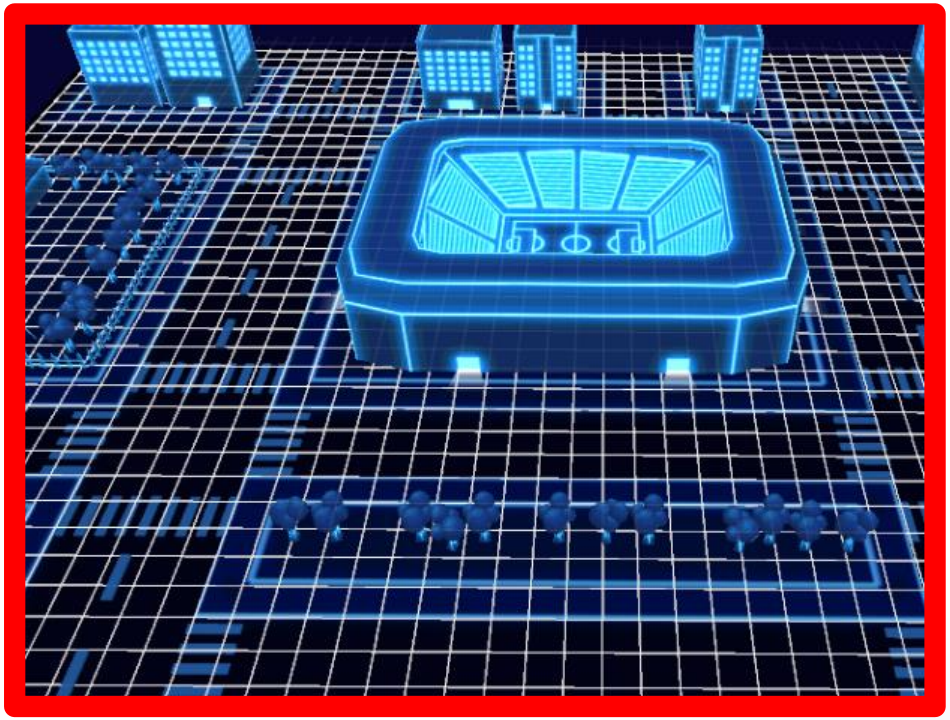


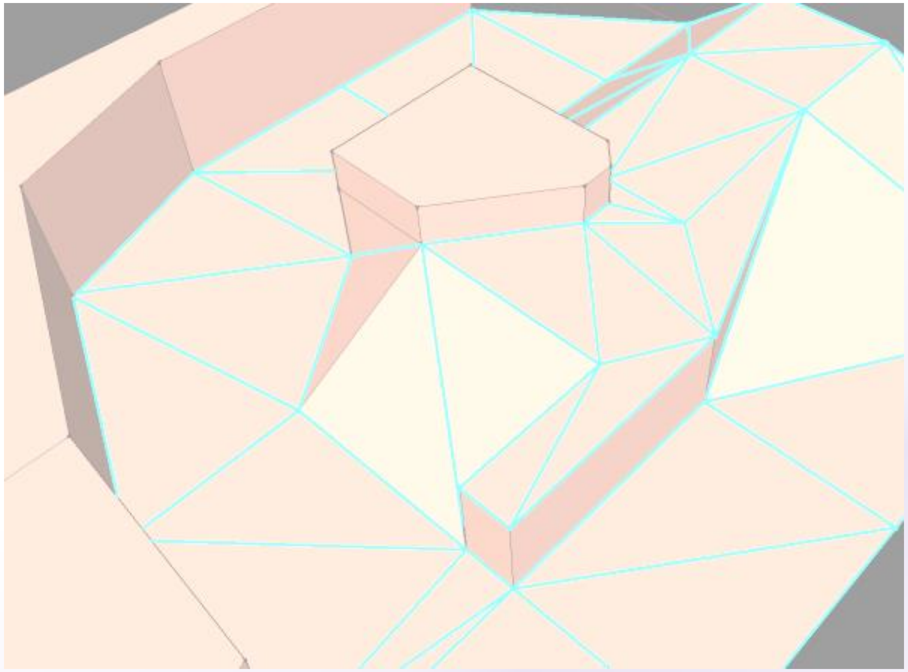
**Navigation Mesh**

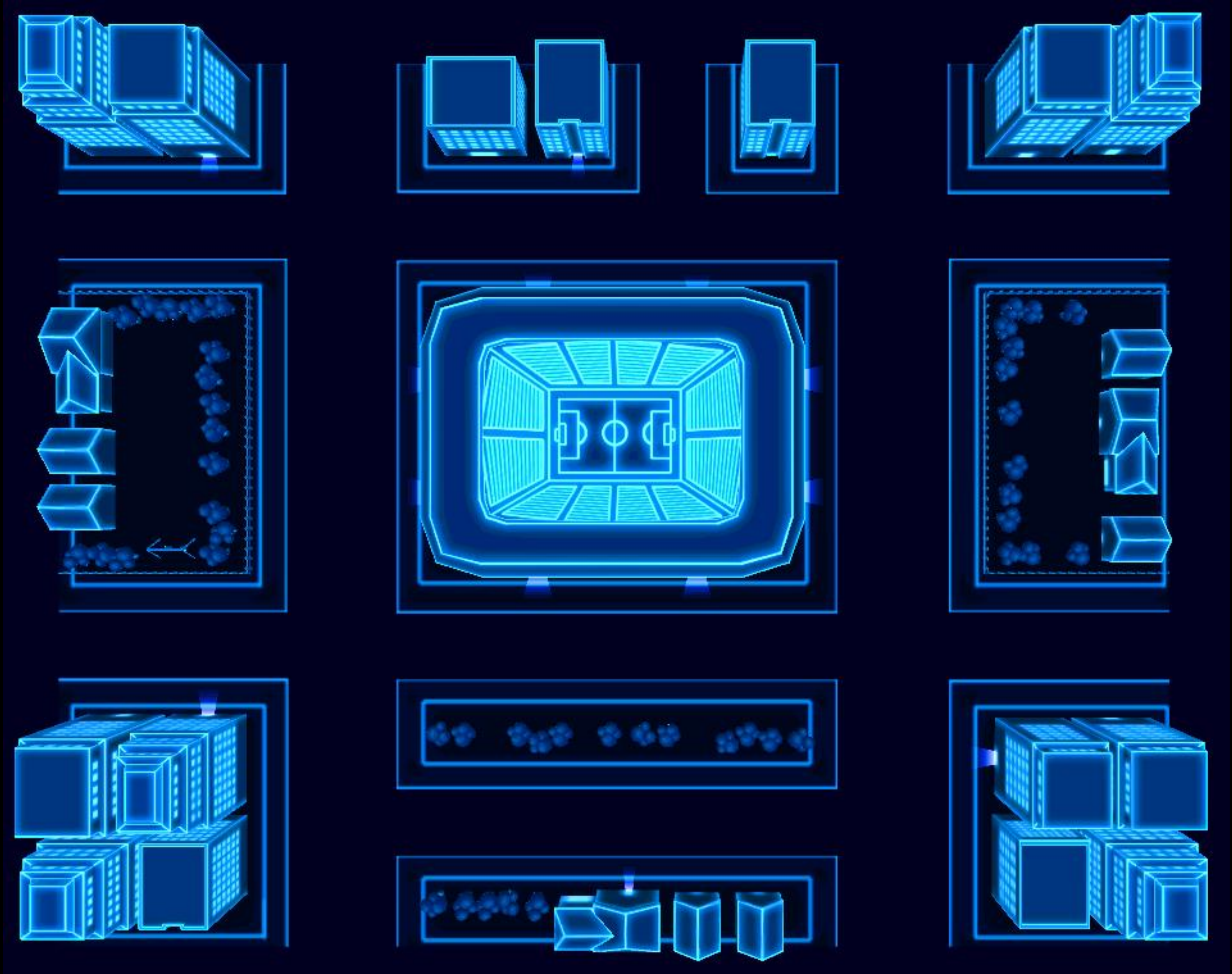


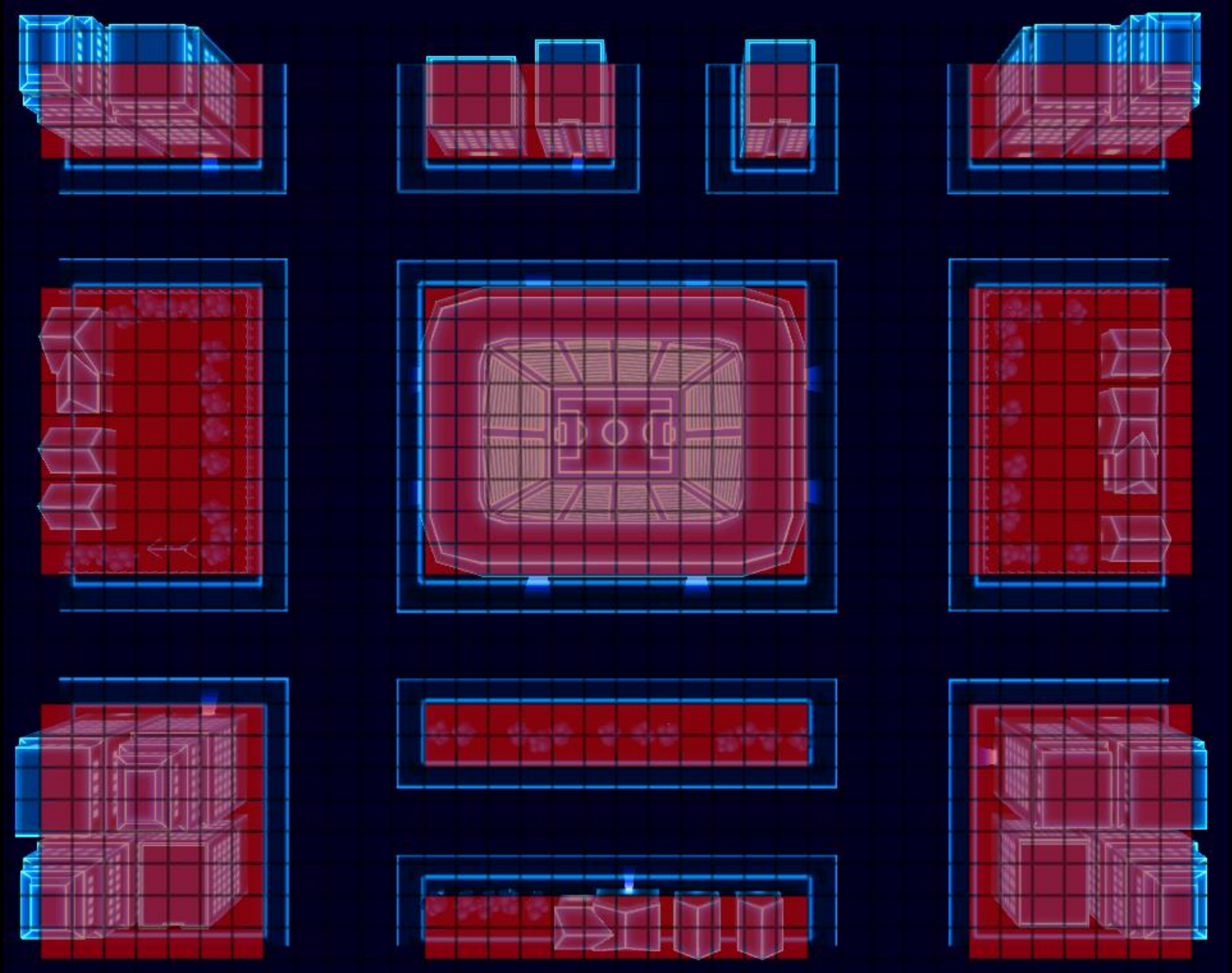
**Grids**

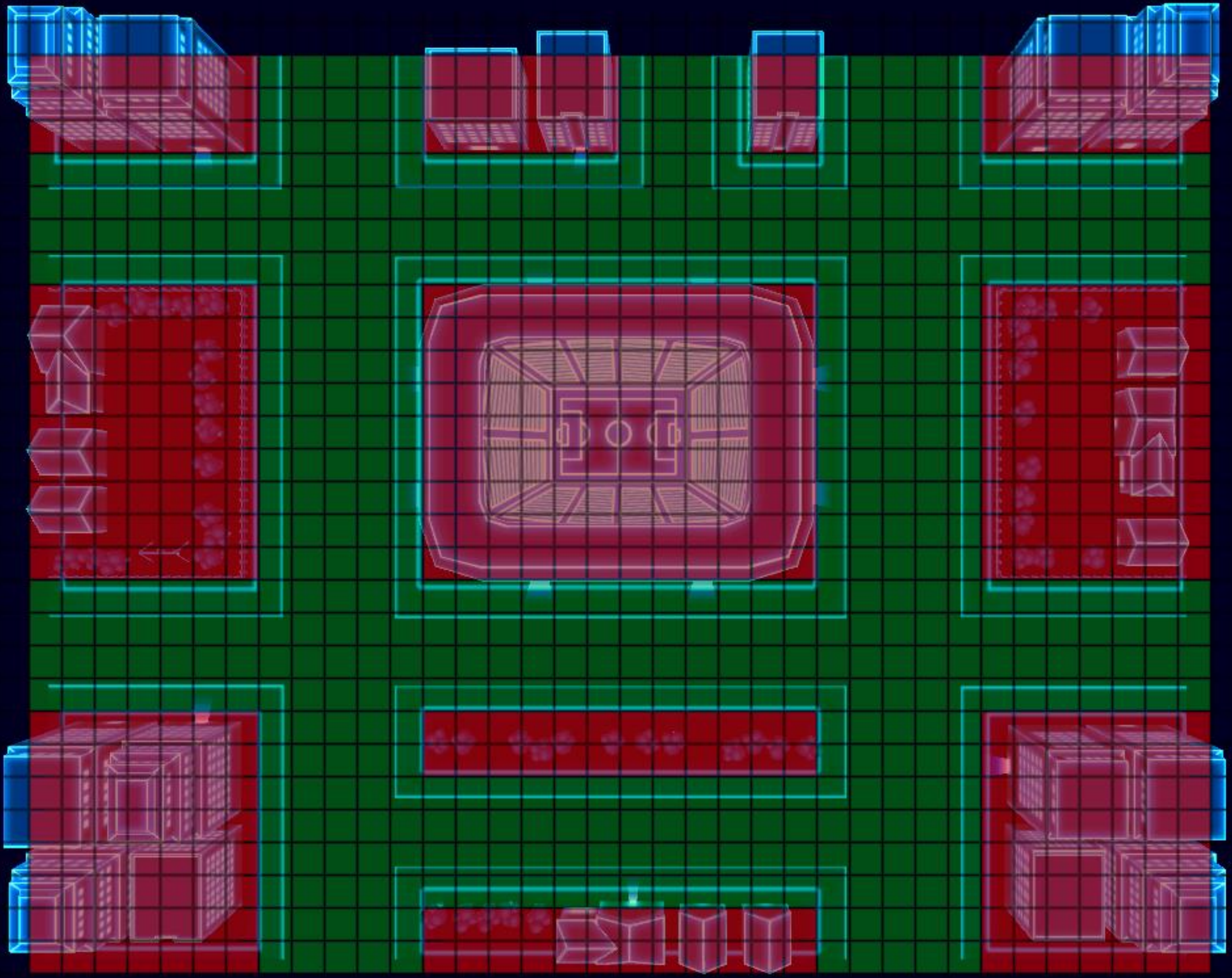
**Navigation Mesh**

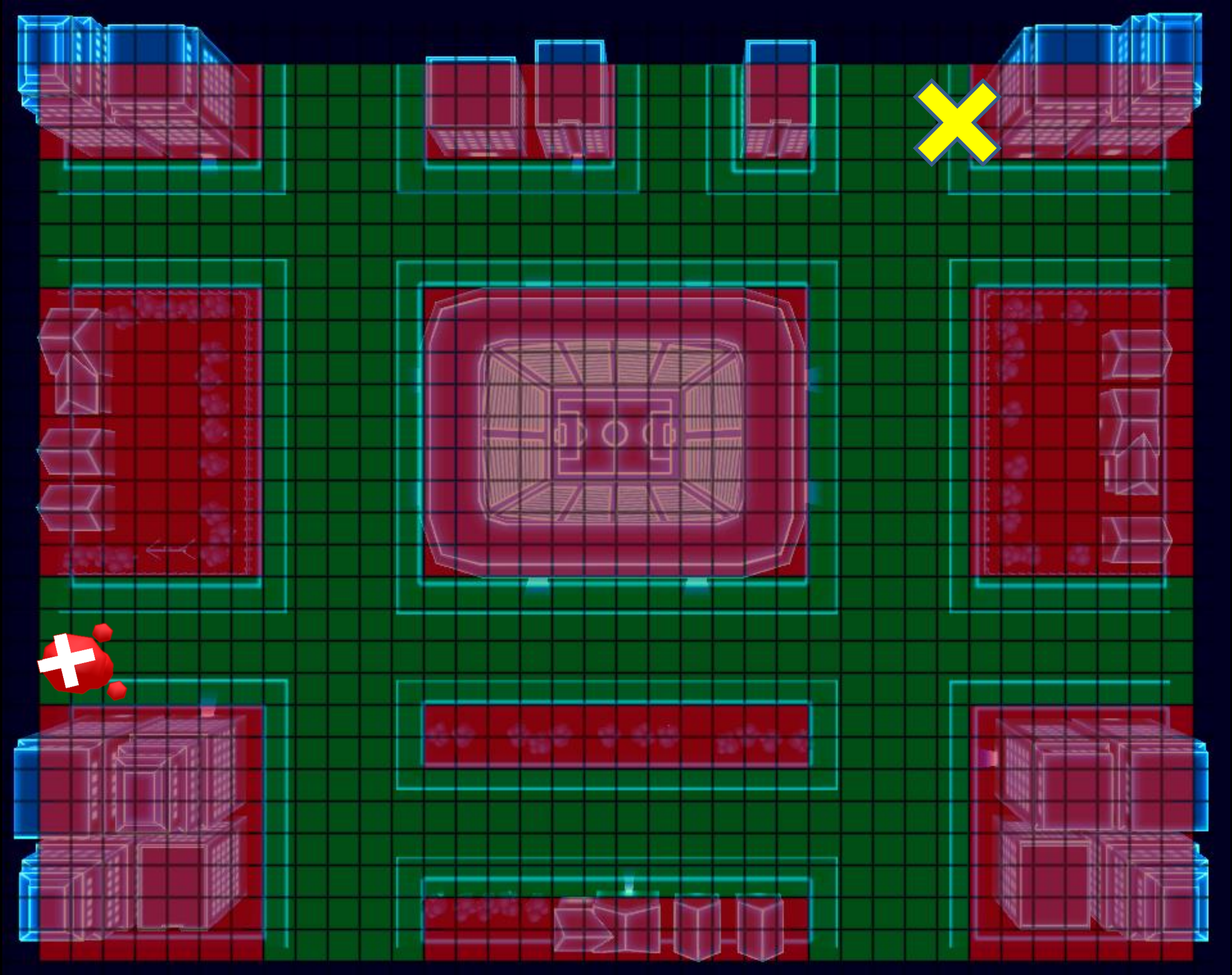












# How do you do pathfinding in games?

# How do you do pathfinding in games?

**Basic answer:**

**A* Algorithm**

# A* Algorithm
## with
# 8-directional paths

### and a quick refresher of A*,
### (or dijkstra’s algorithm, if you don’t know what A* is)

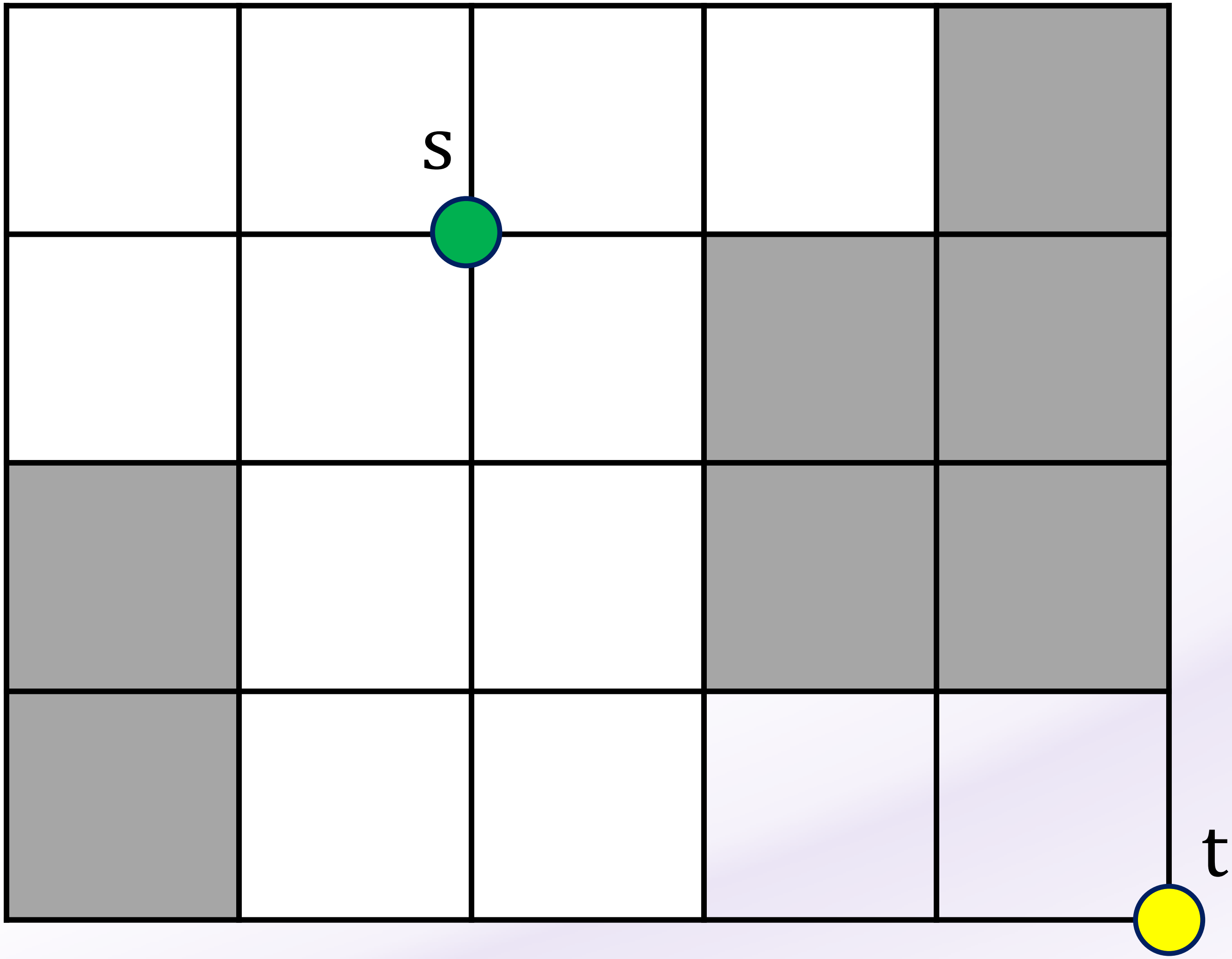
# What is A*?

# What is A*?

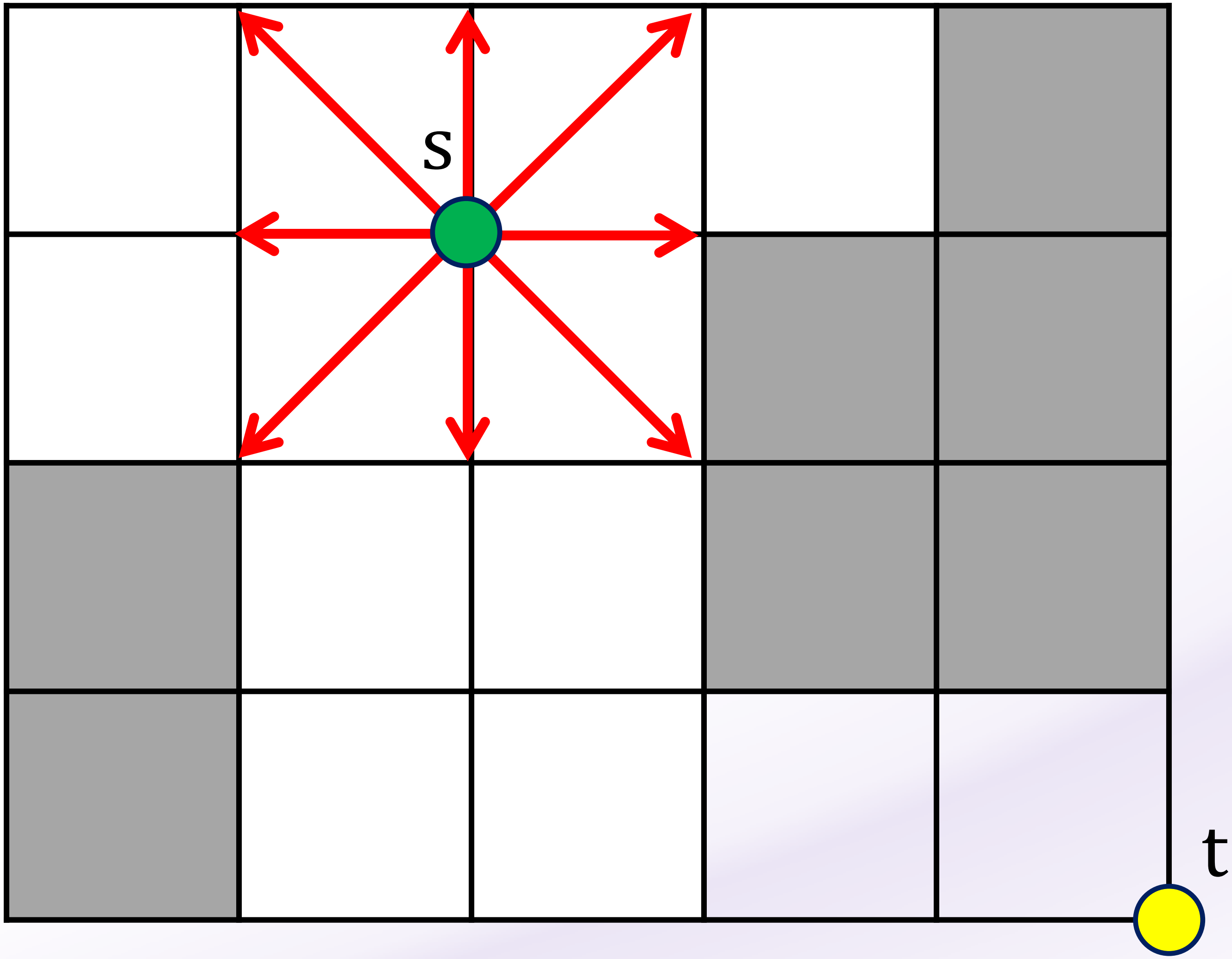
**Dijkstra + heuristic**

**= A* Algorithm**

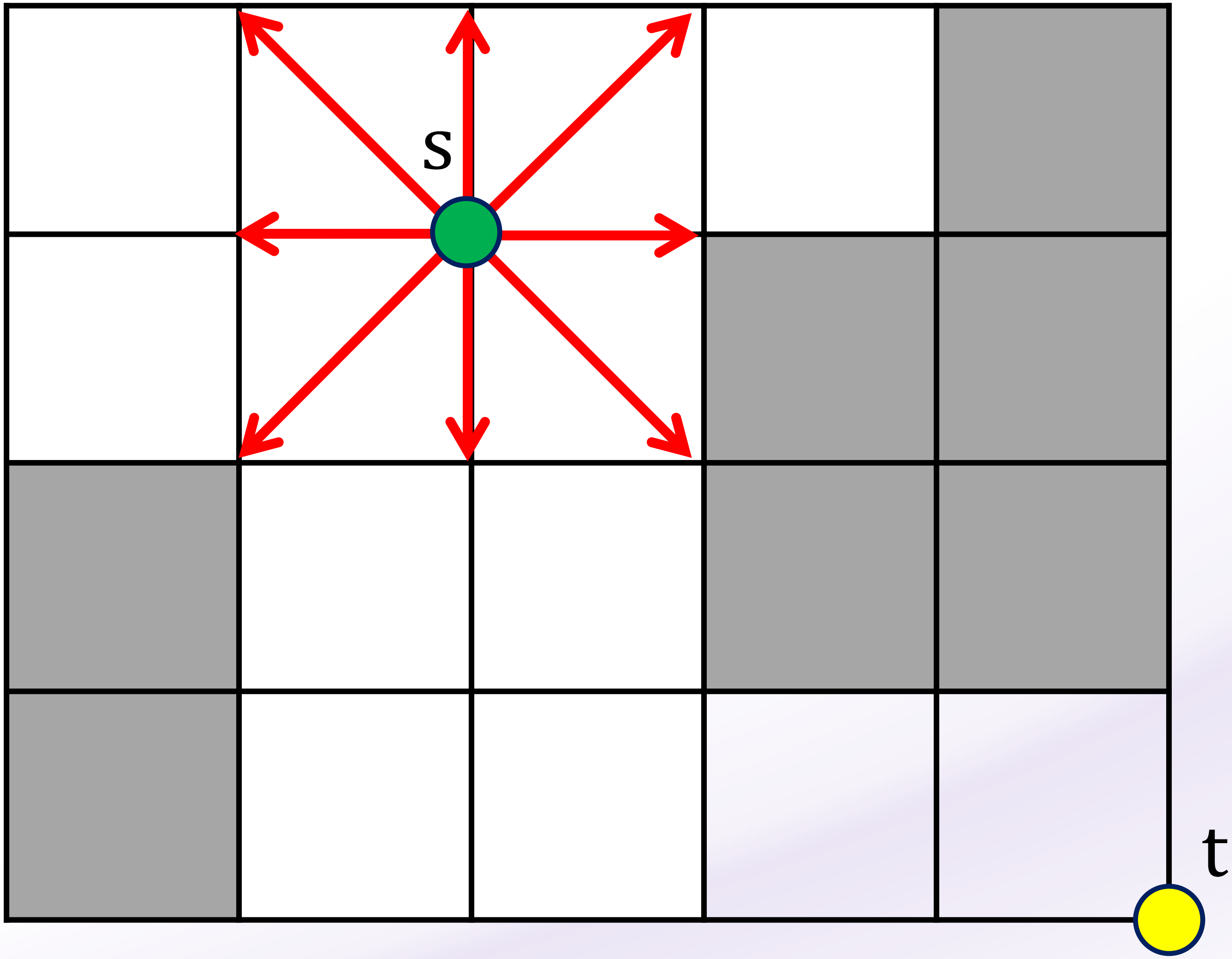
# 8-directional movement



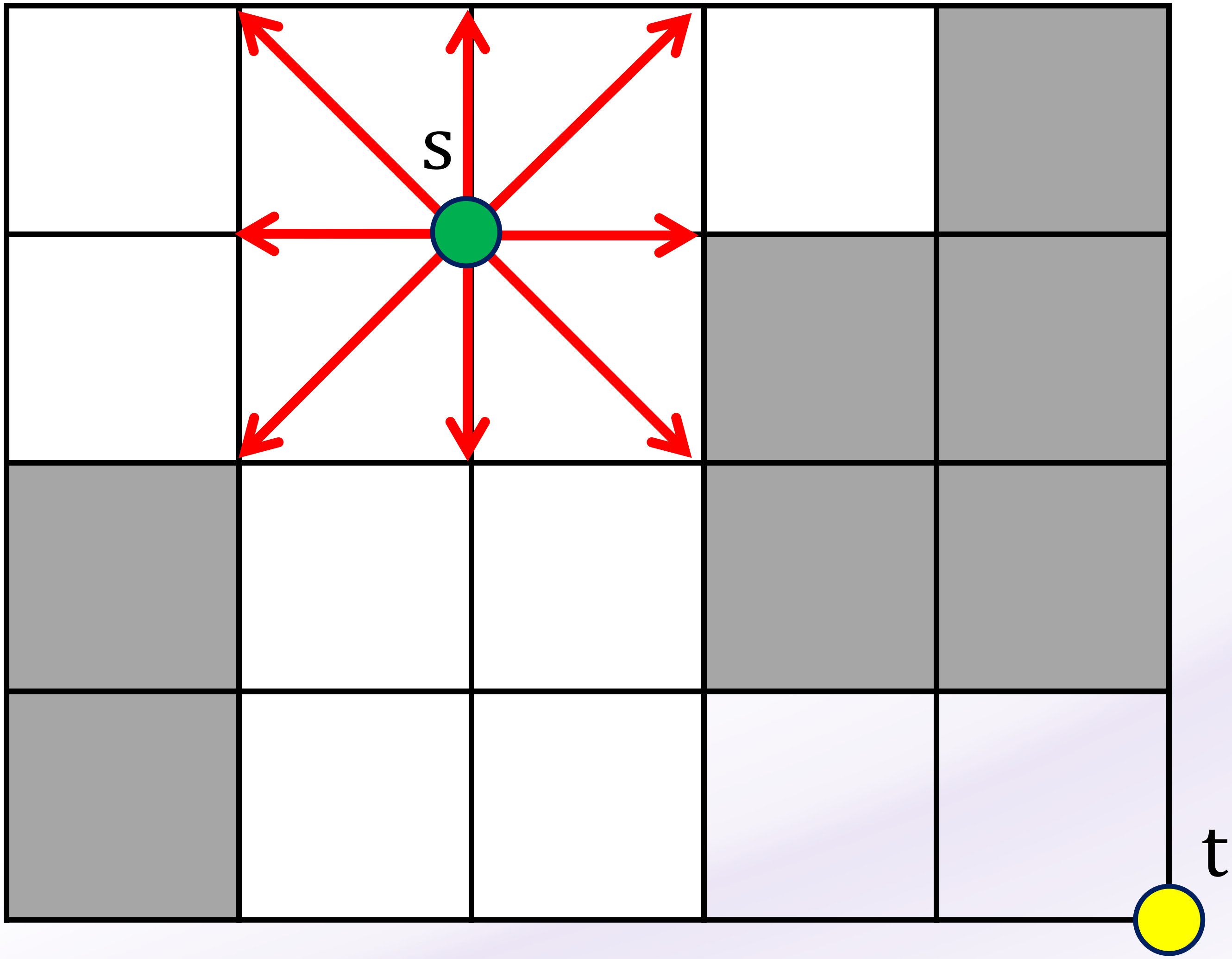
# 8-directional movement



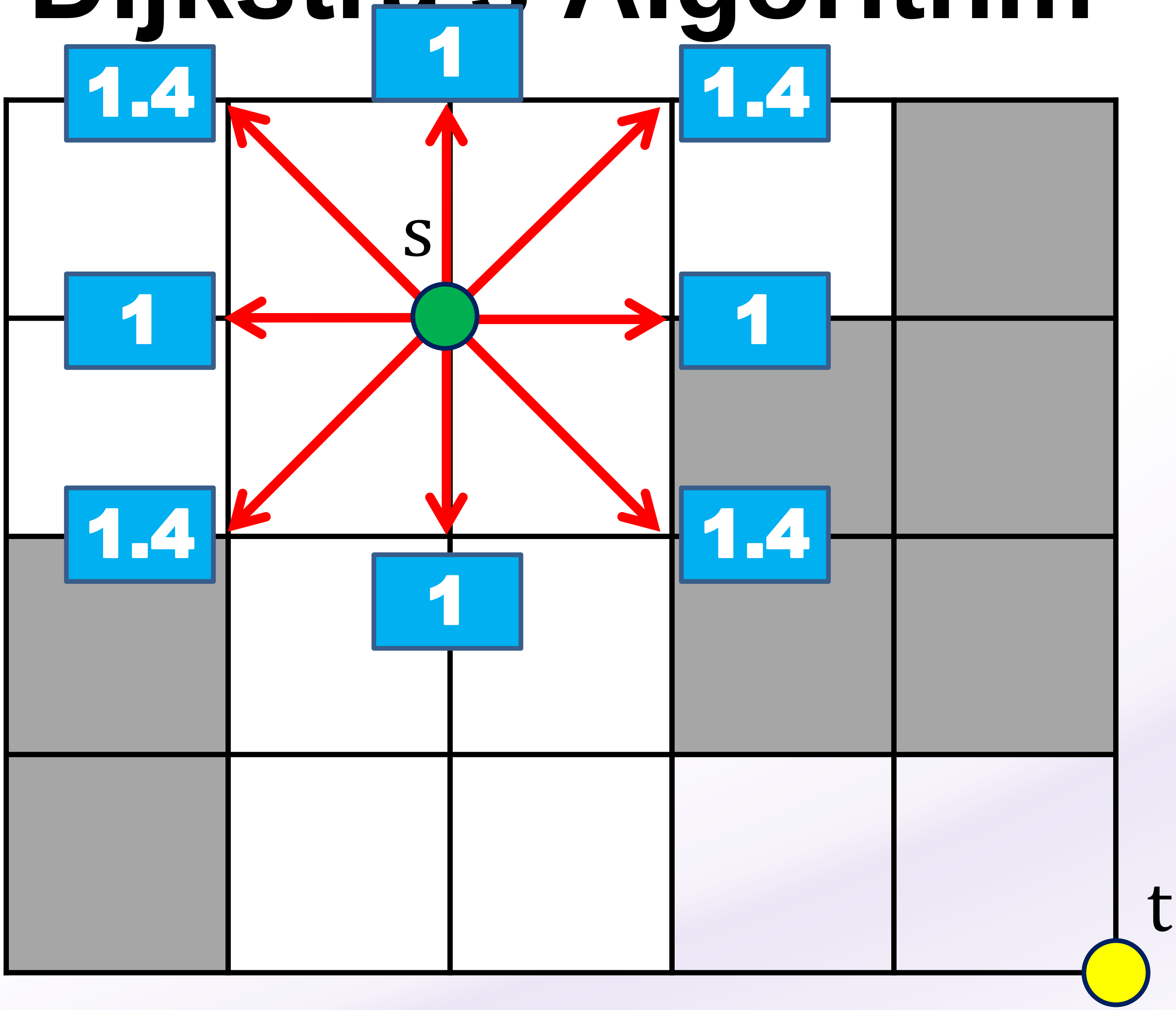
# Which vertex to explore next?



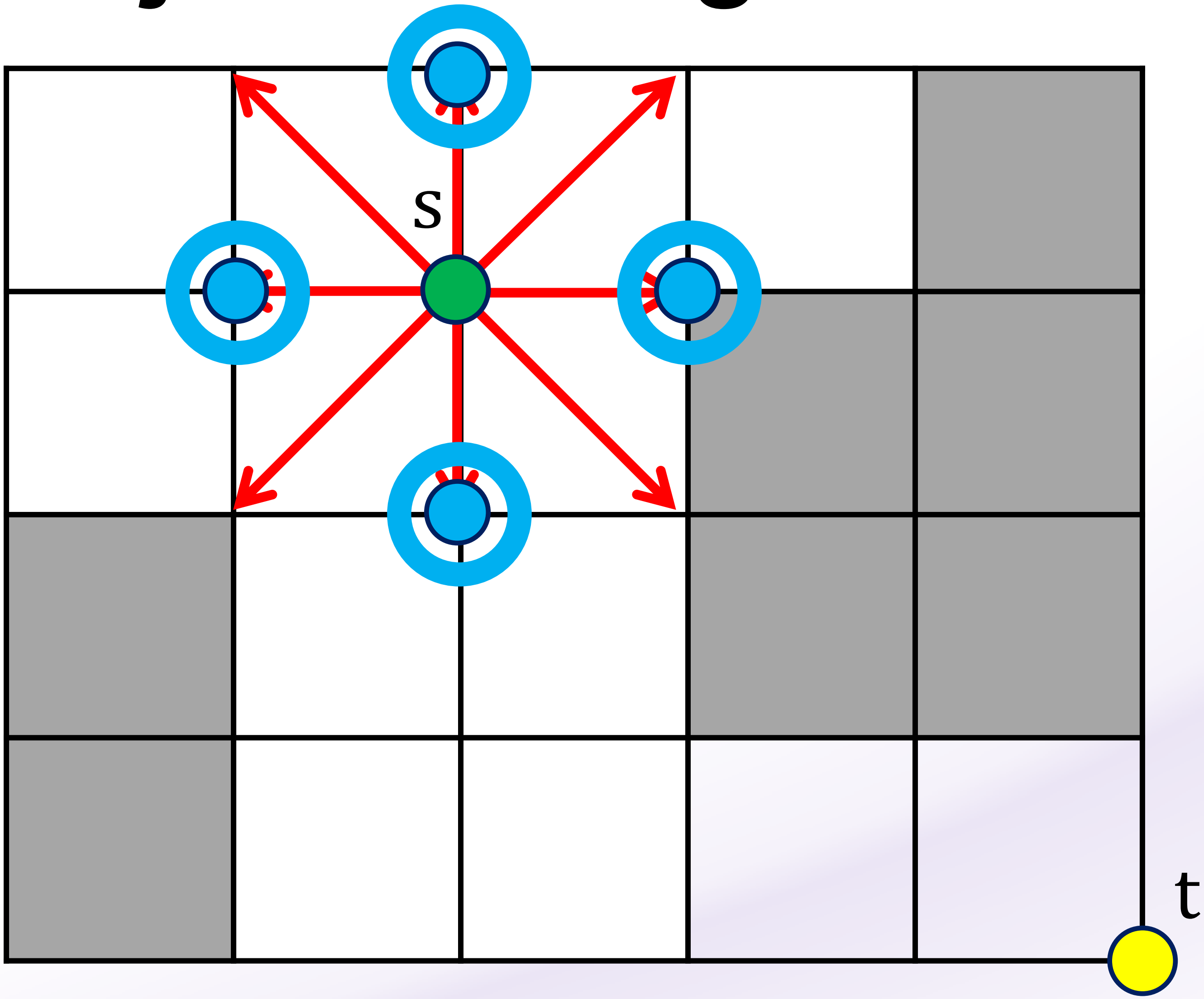
# Dijkstra's Algorithm



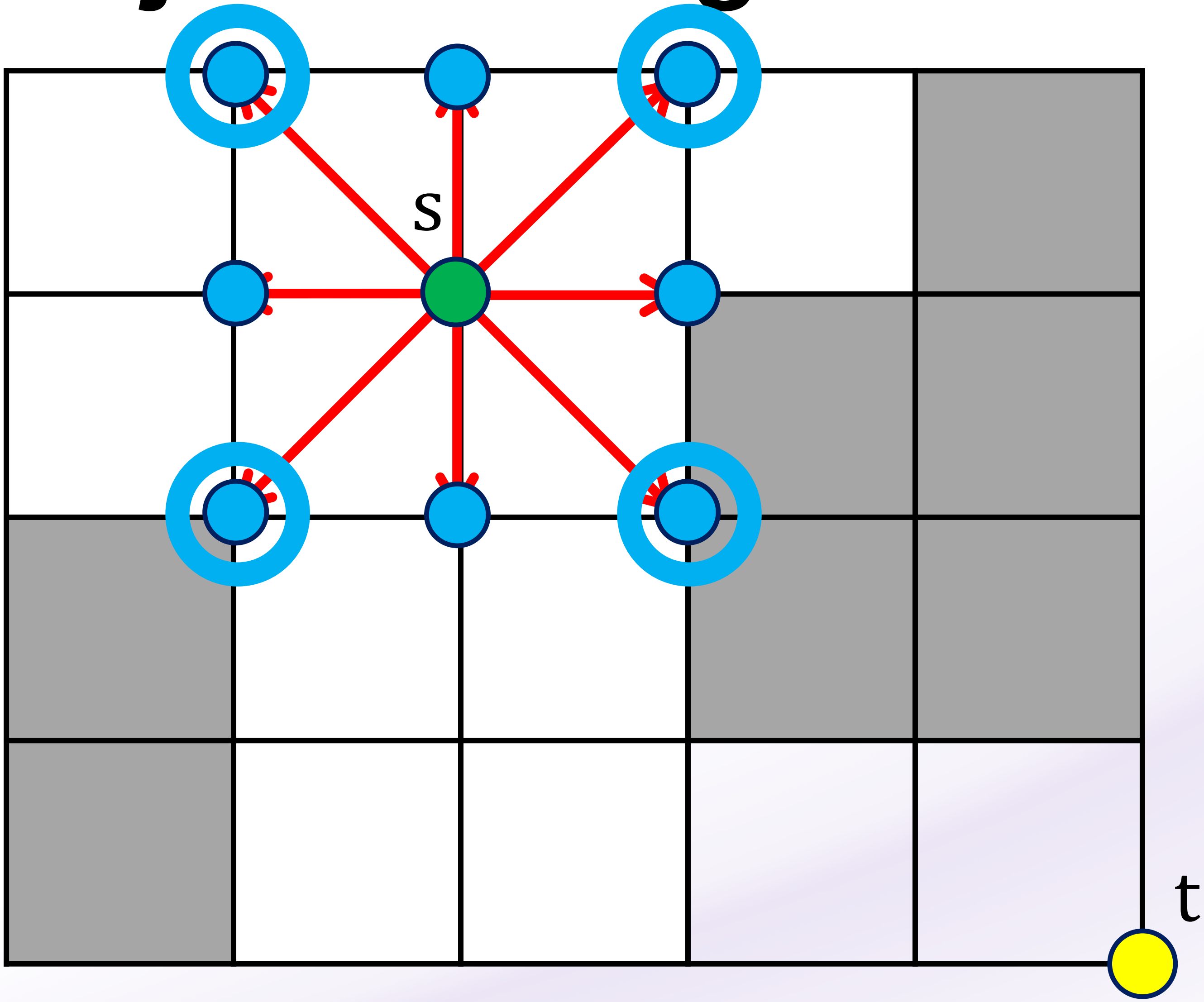
# Dijkstra’s Algorithm



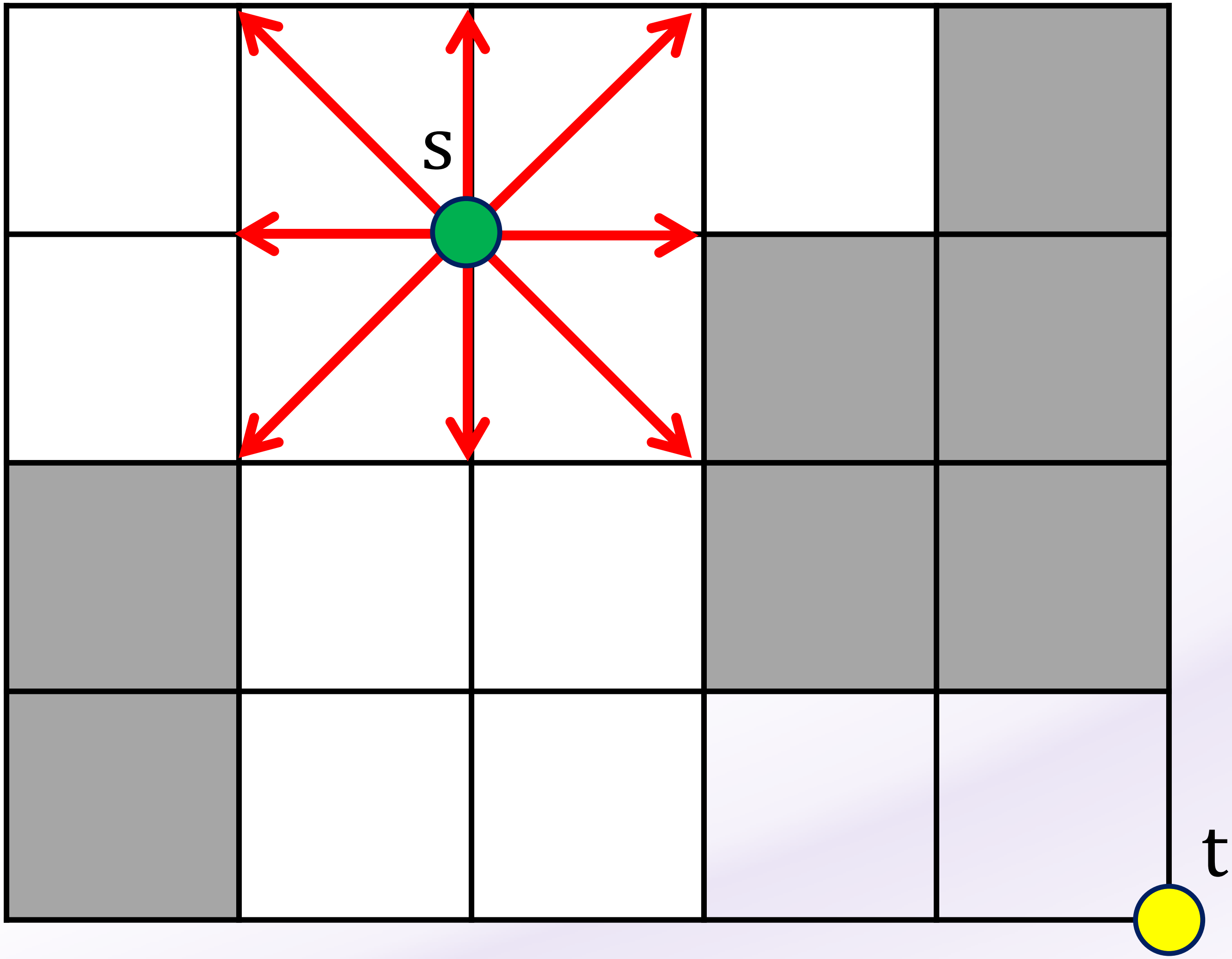
# Dijkstra’s Algorithm



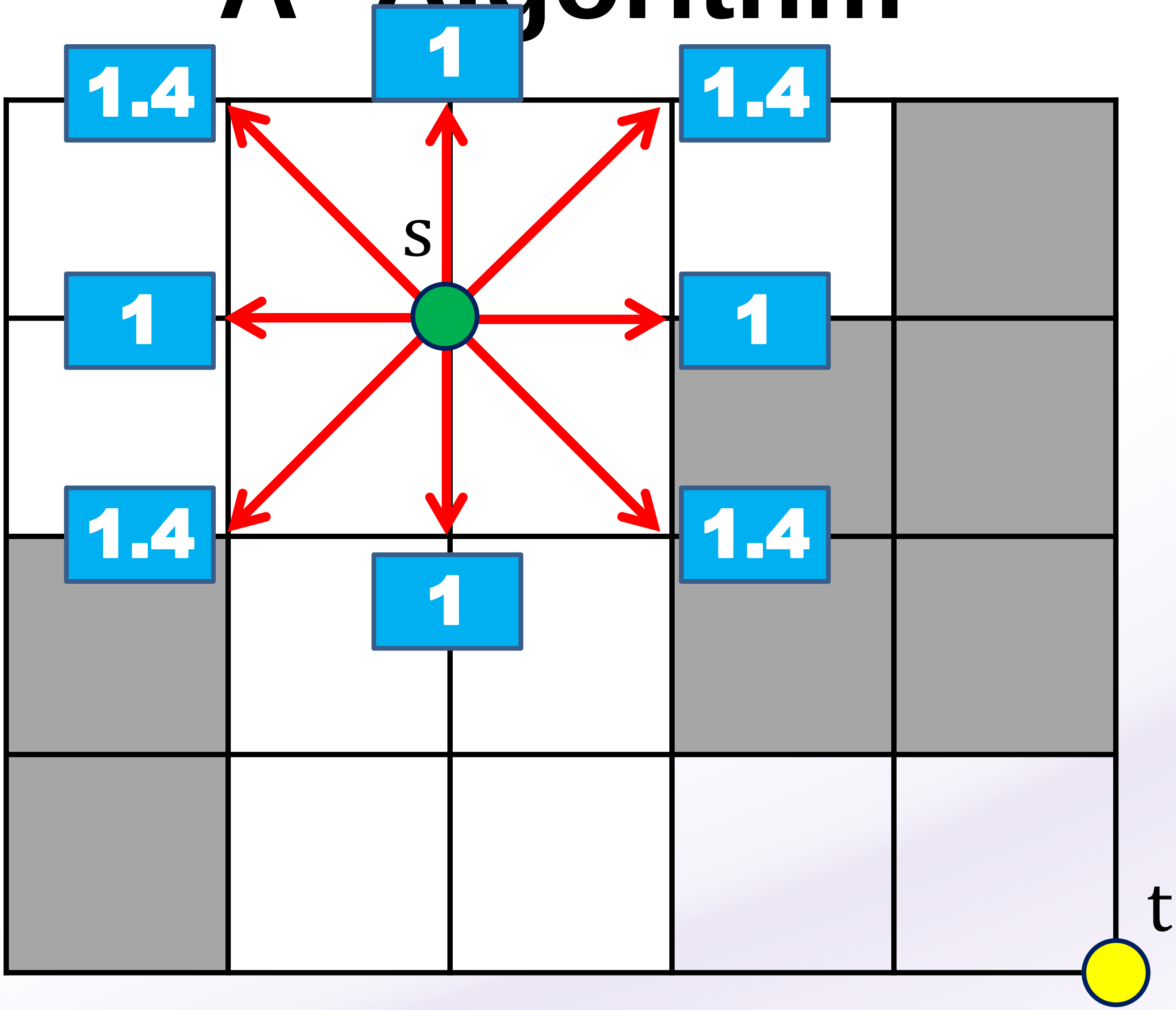
# Dijkstra’s Algorithm



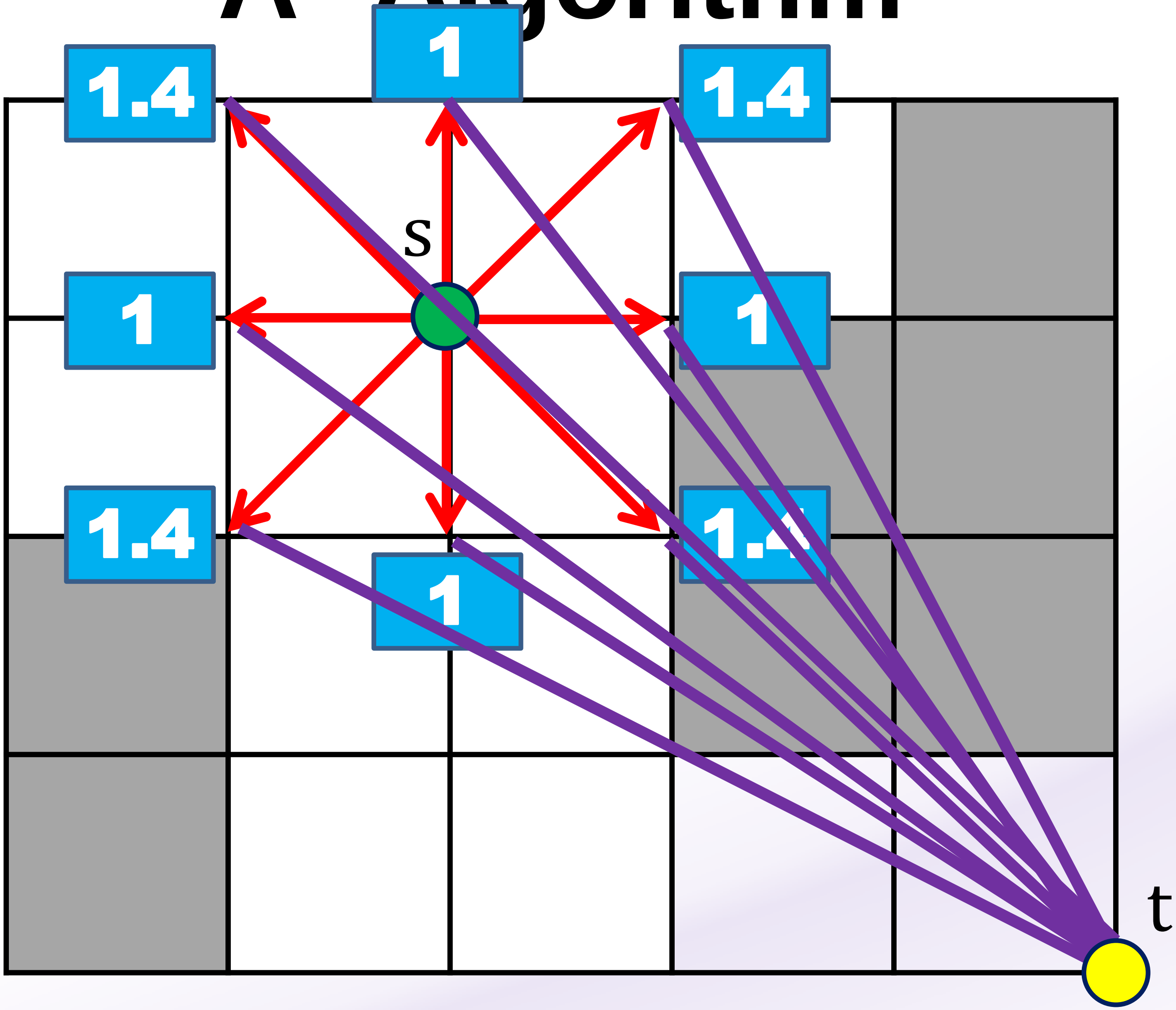
# A* Algorithm



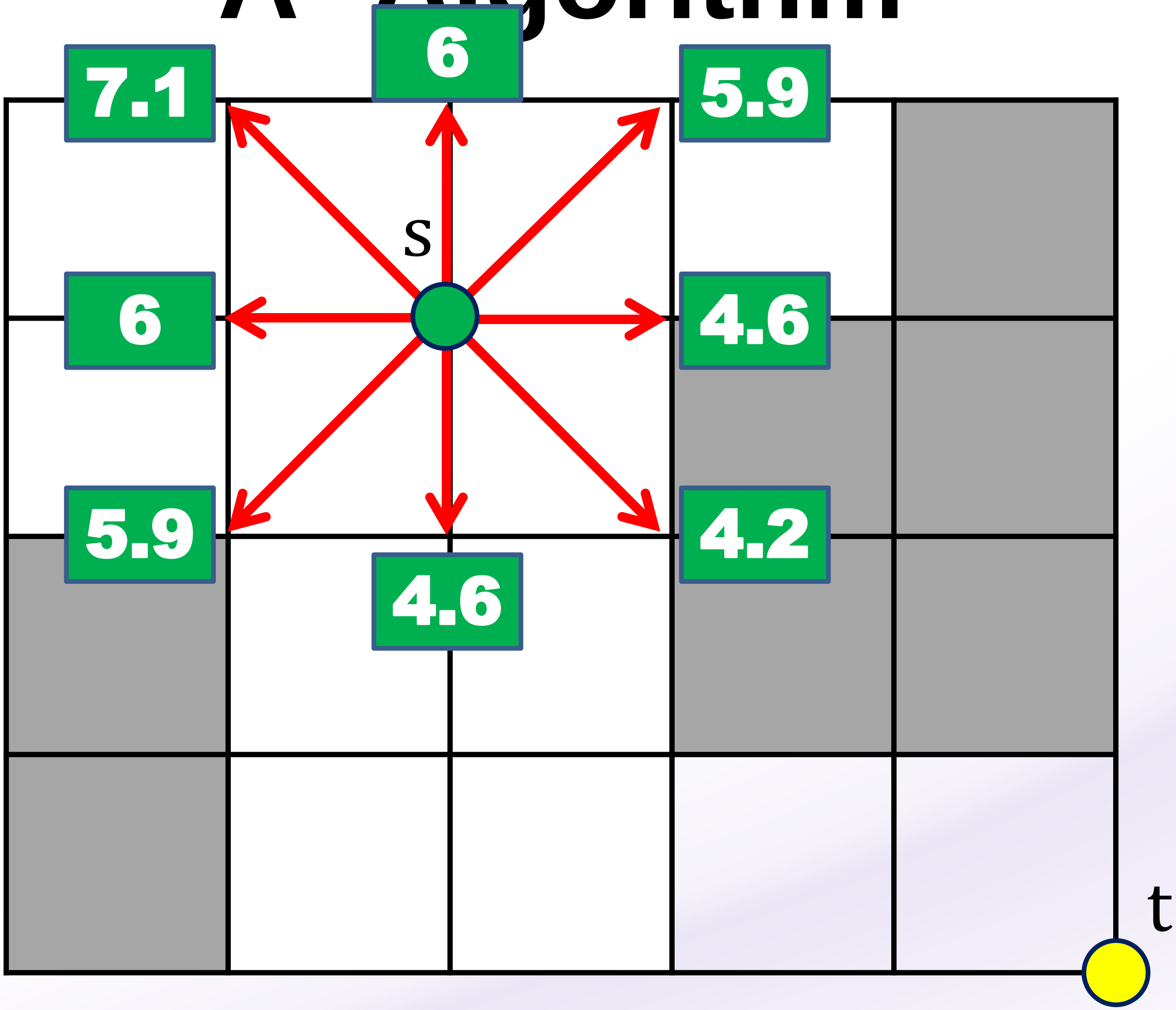
# A* Algorithm



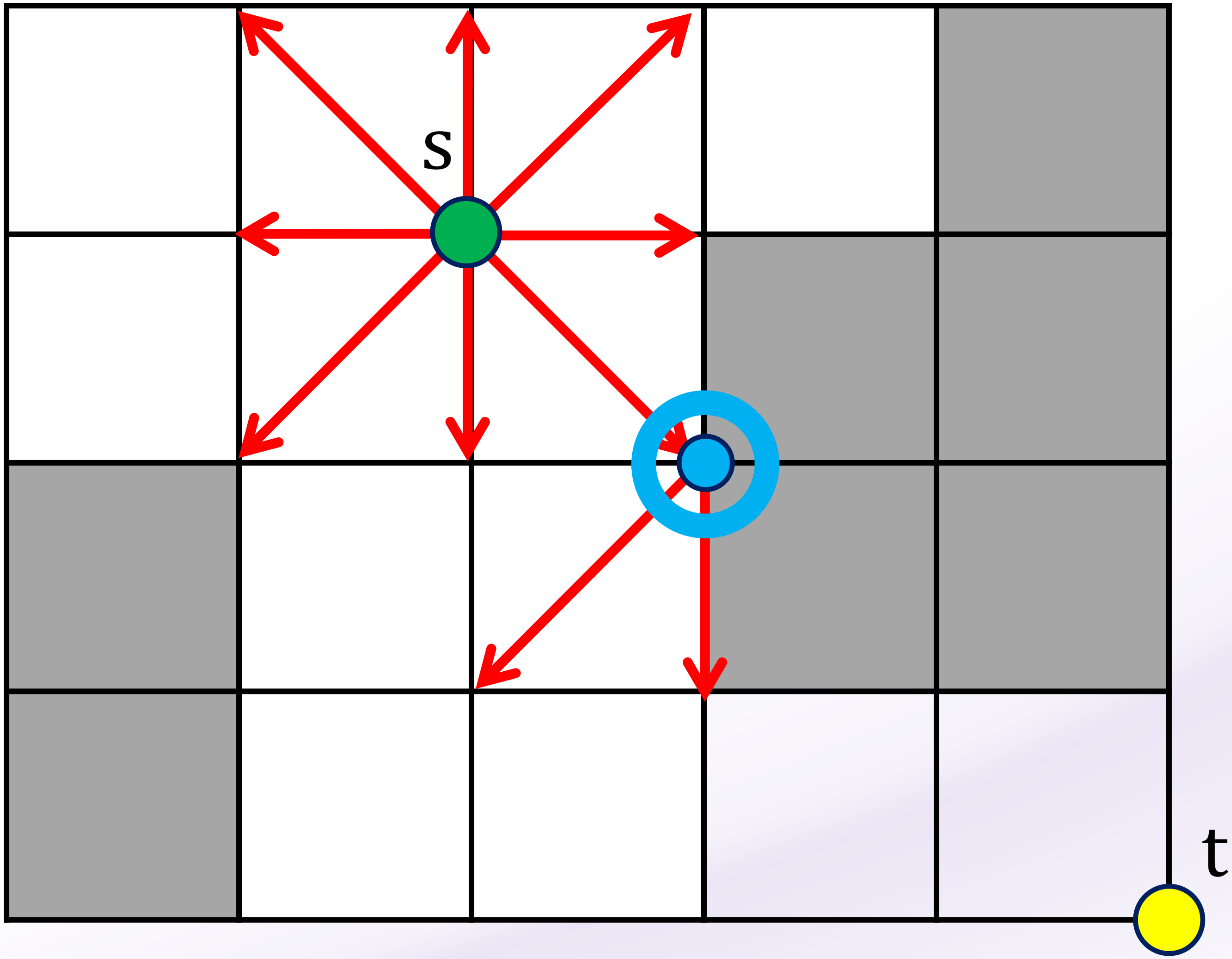
# A* Algorithm



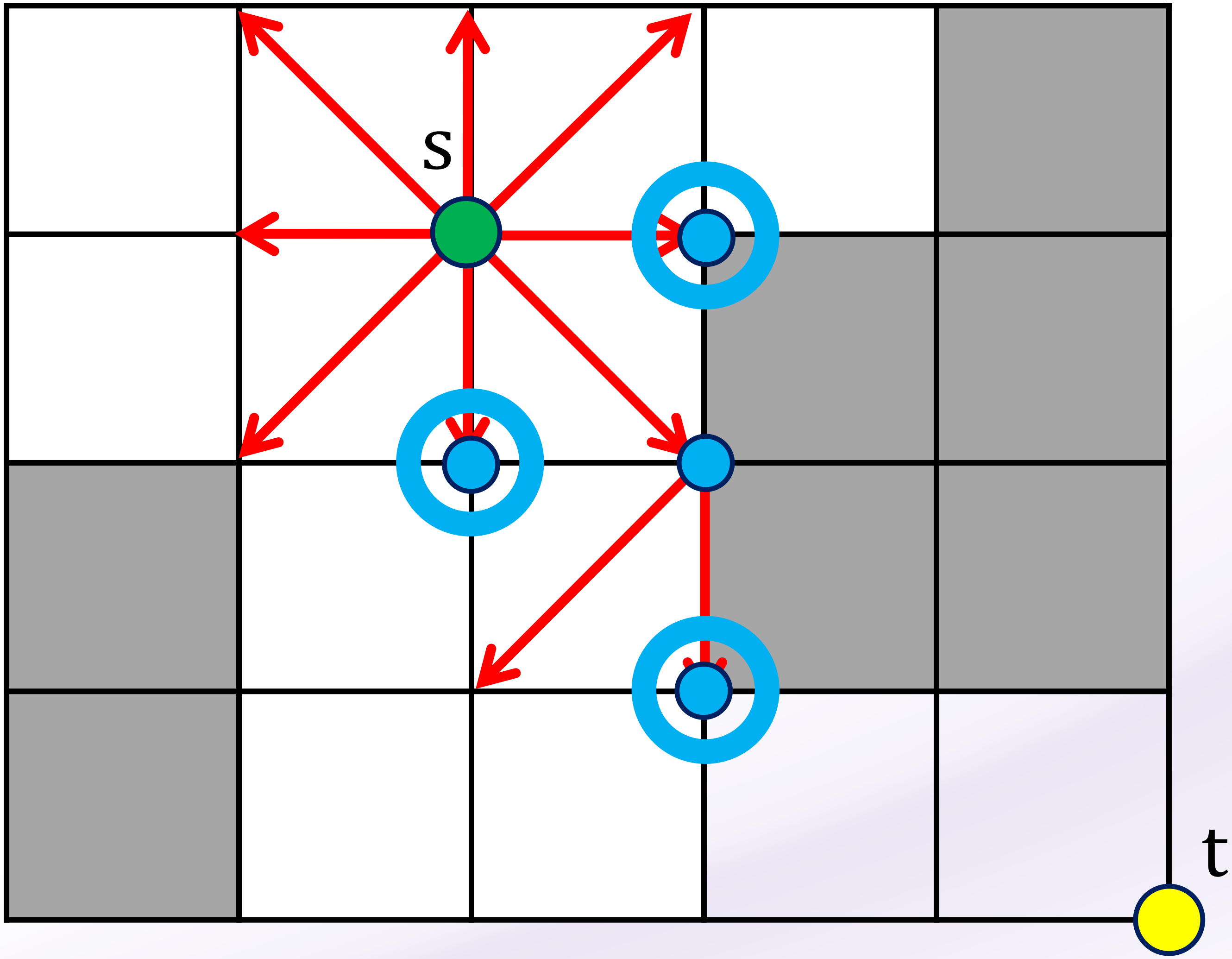
# A* Algorithm



# A* Algorithm



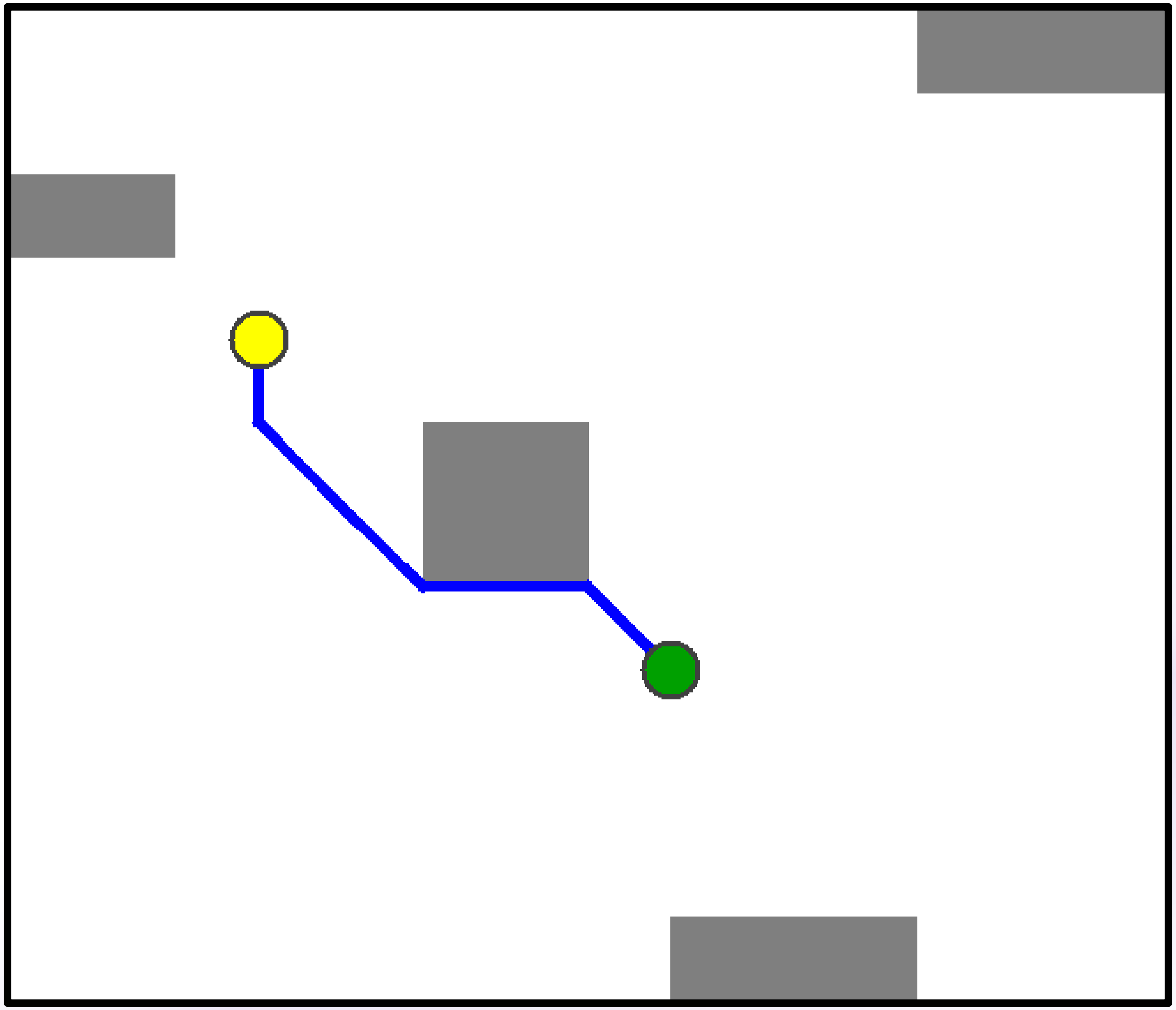
# A* Algorithm



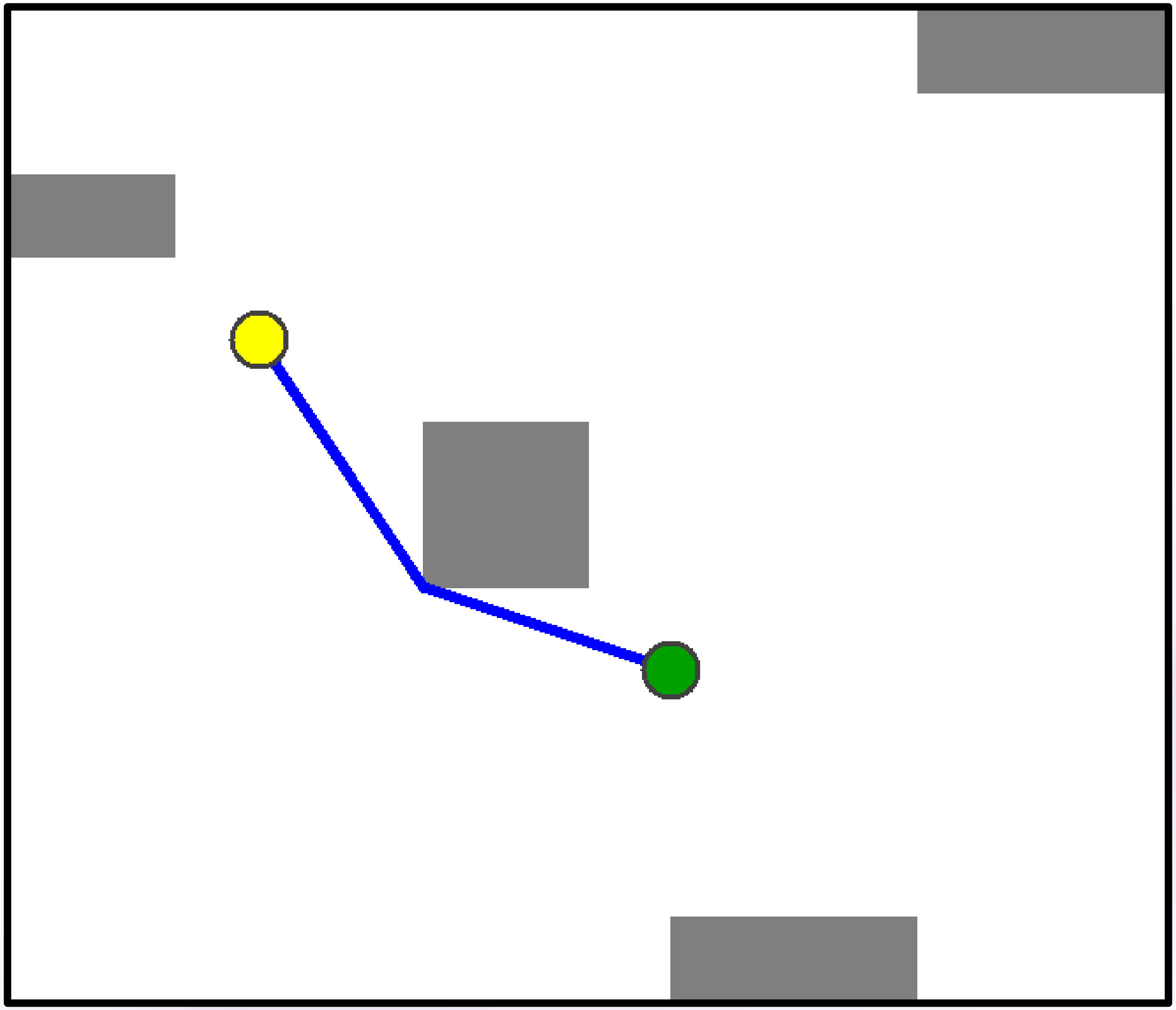
# Dijkstra vs A* Demo

Dijkstra_Demo
Astar_Demo

# It works,
# but isn’t exactly ideal.



# We'd prefer Something like this.



# Any-Angle Pathfinding Algorithms

**Theta***

**Visibility Graphs**

# Any-Angle Pathfinding Algorithms

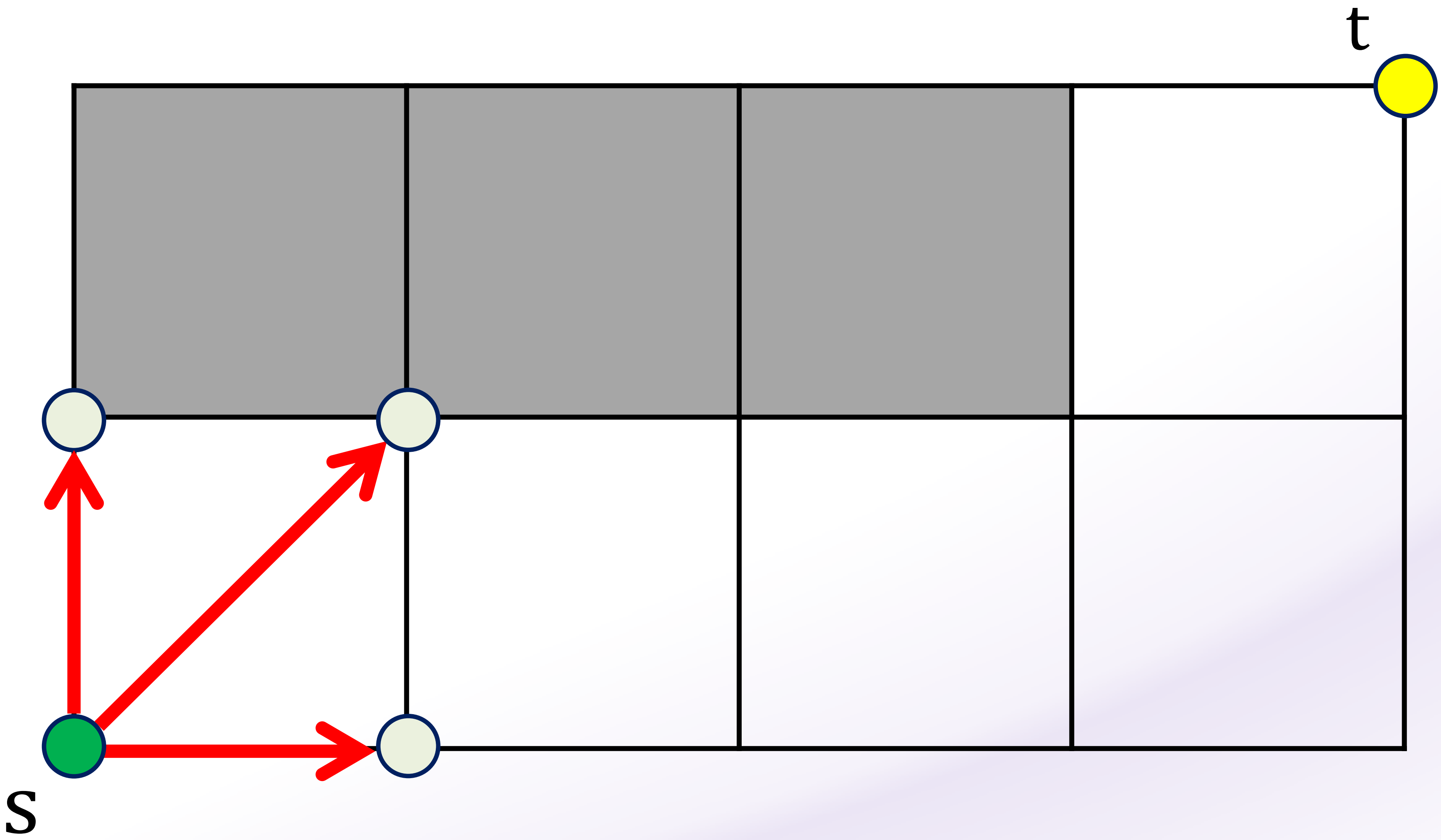
Theta*

Visibility Graphs

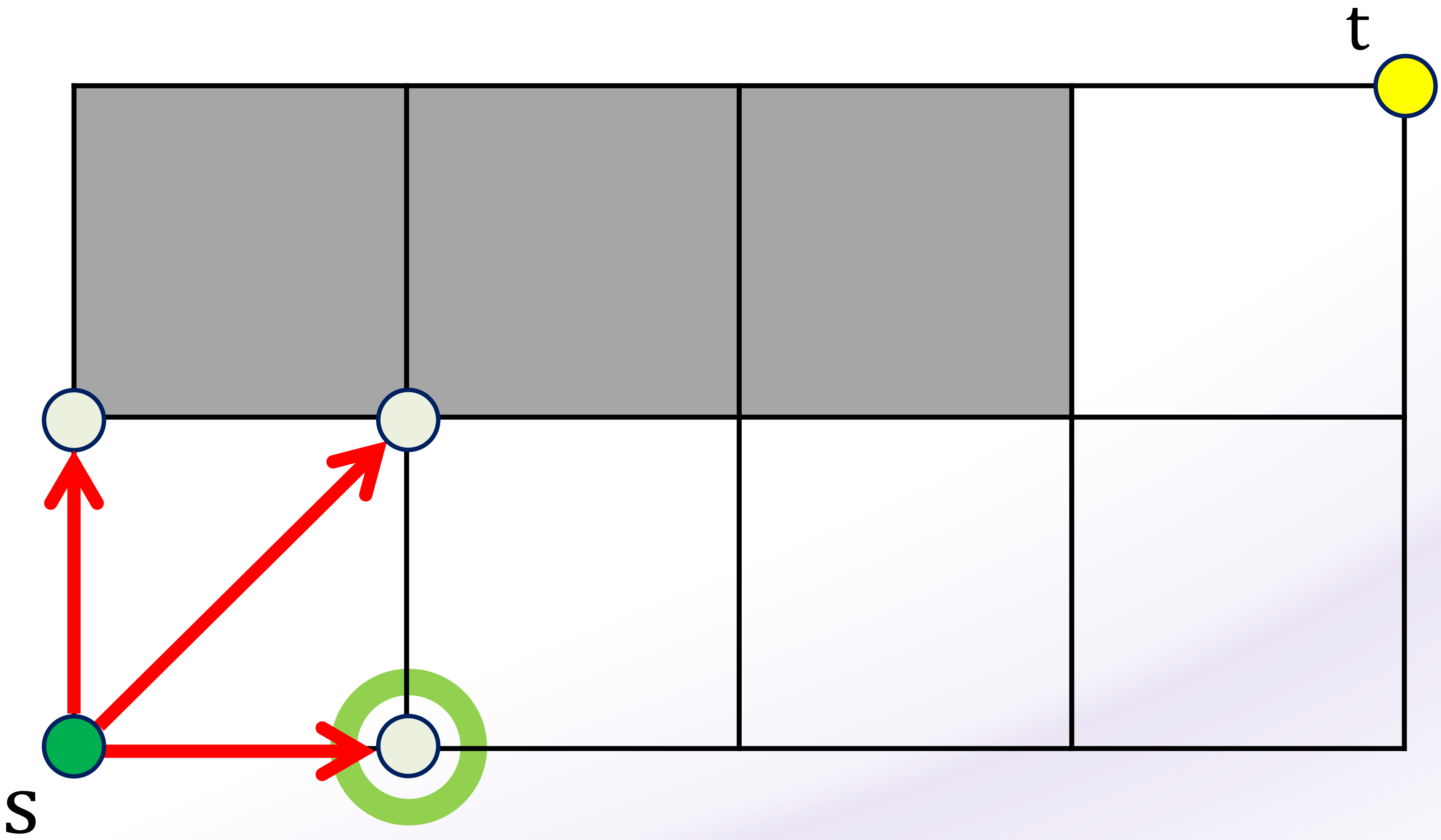
# Theta* Algorithm

## Similar to A*,
## but with one little change

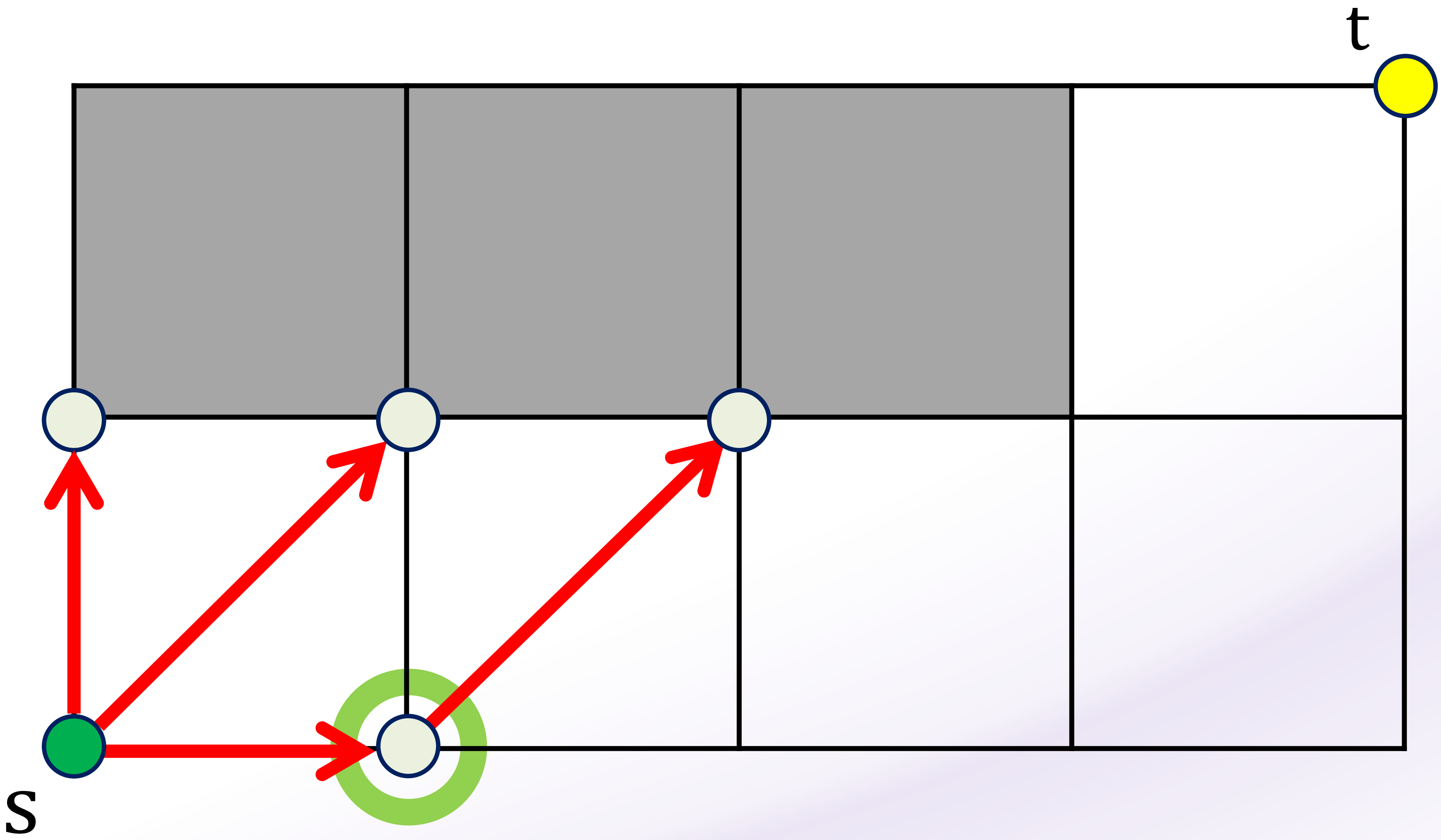
# Theta* Algorithm



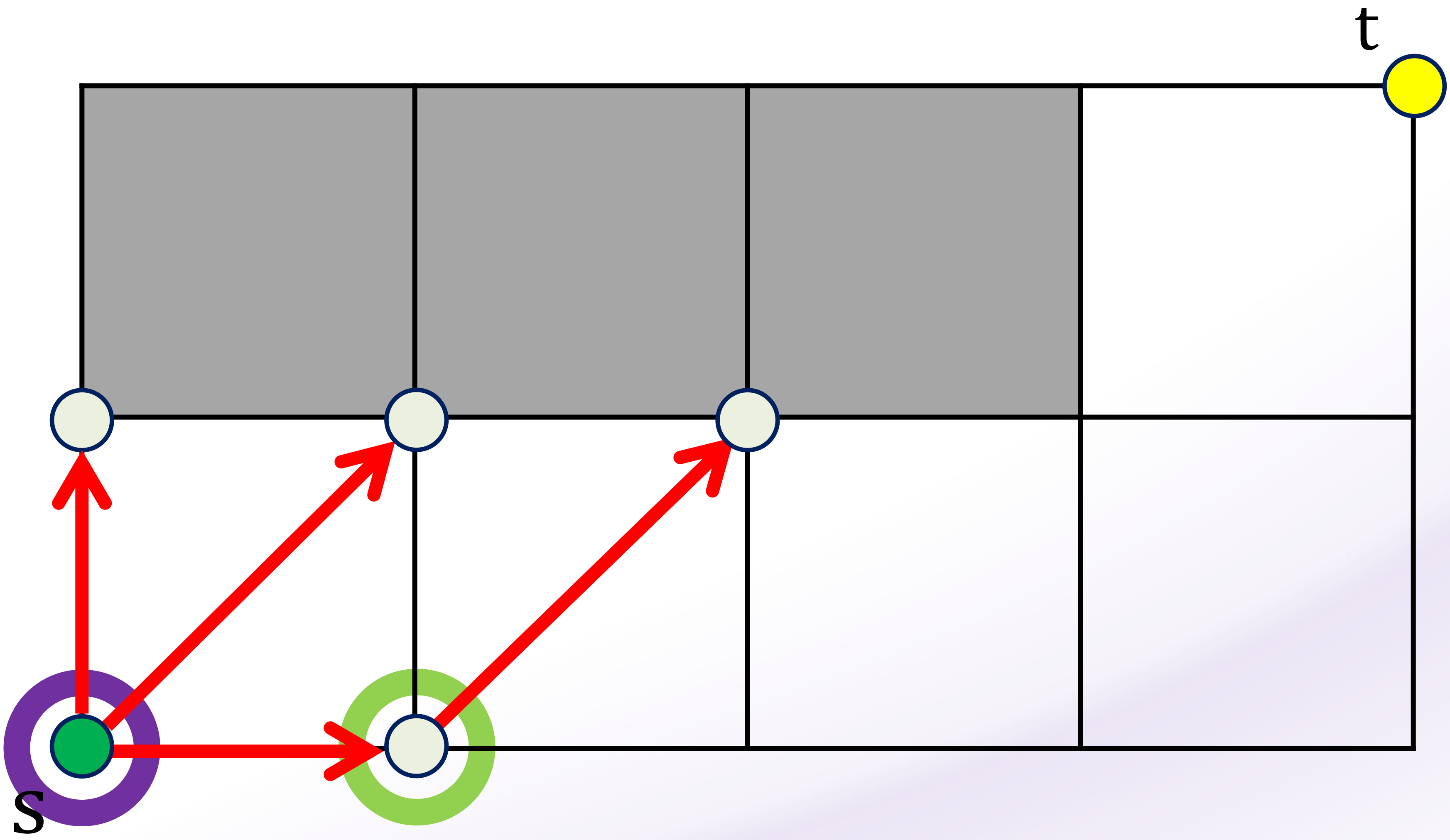
# Theta* Algorithm



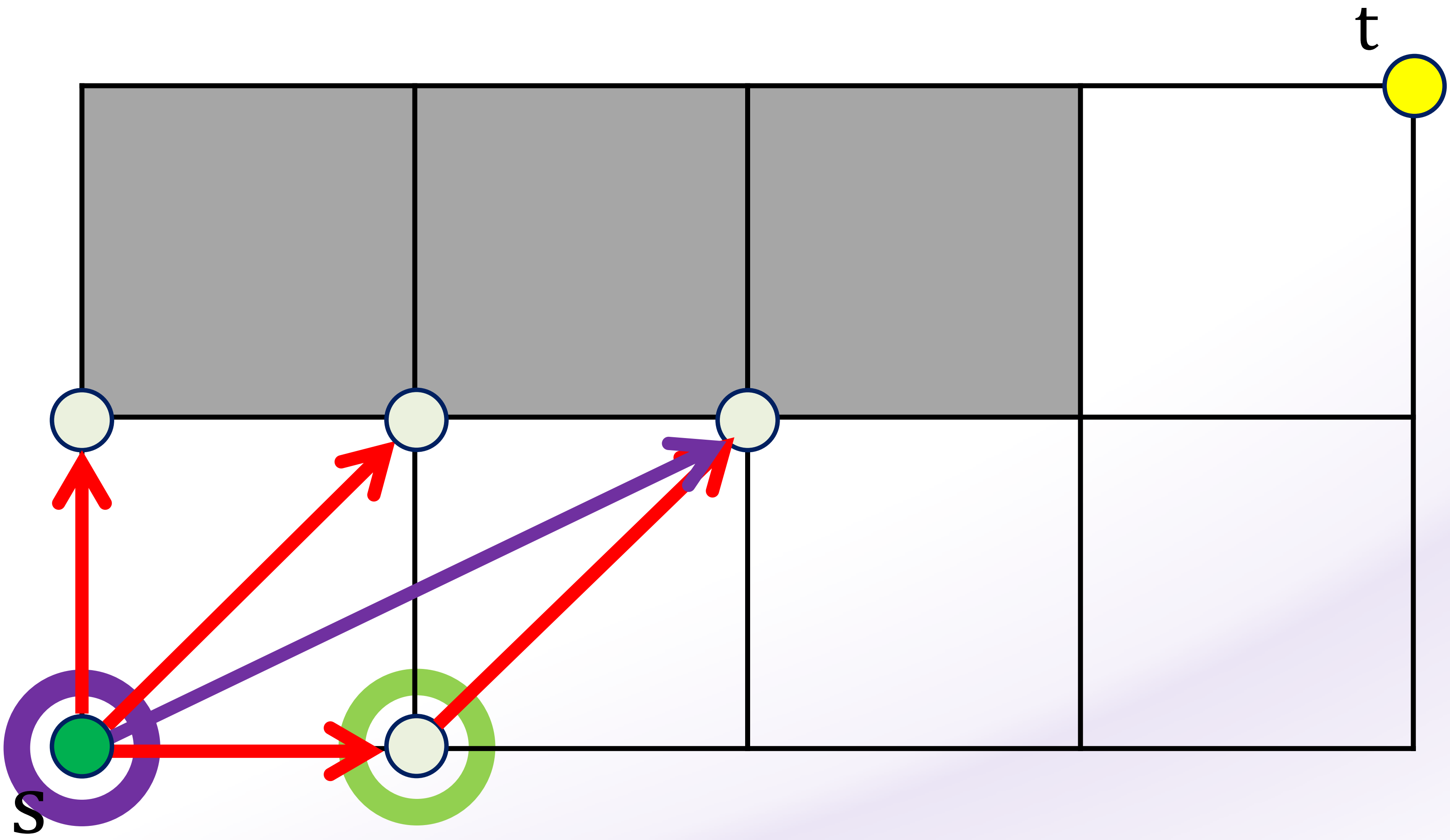
# Theta* Algorithm



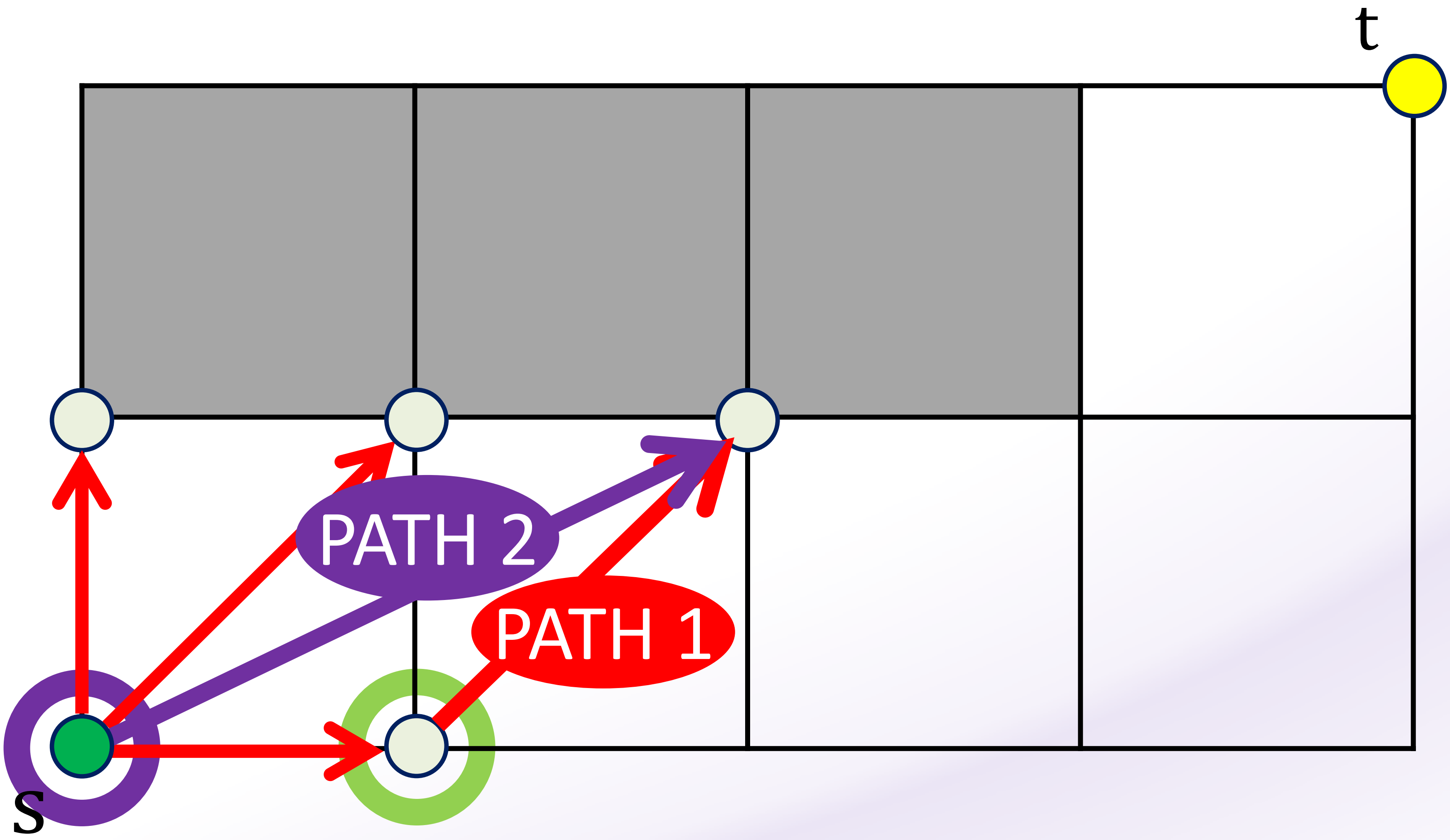
# Theta* Algorithm



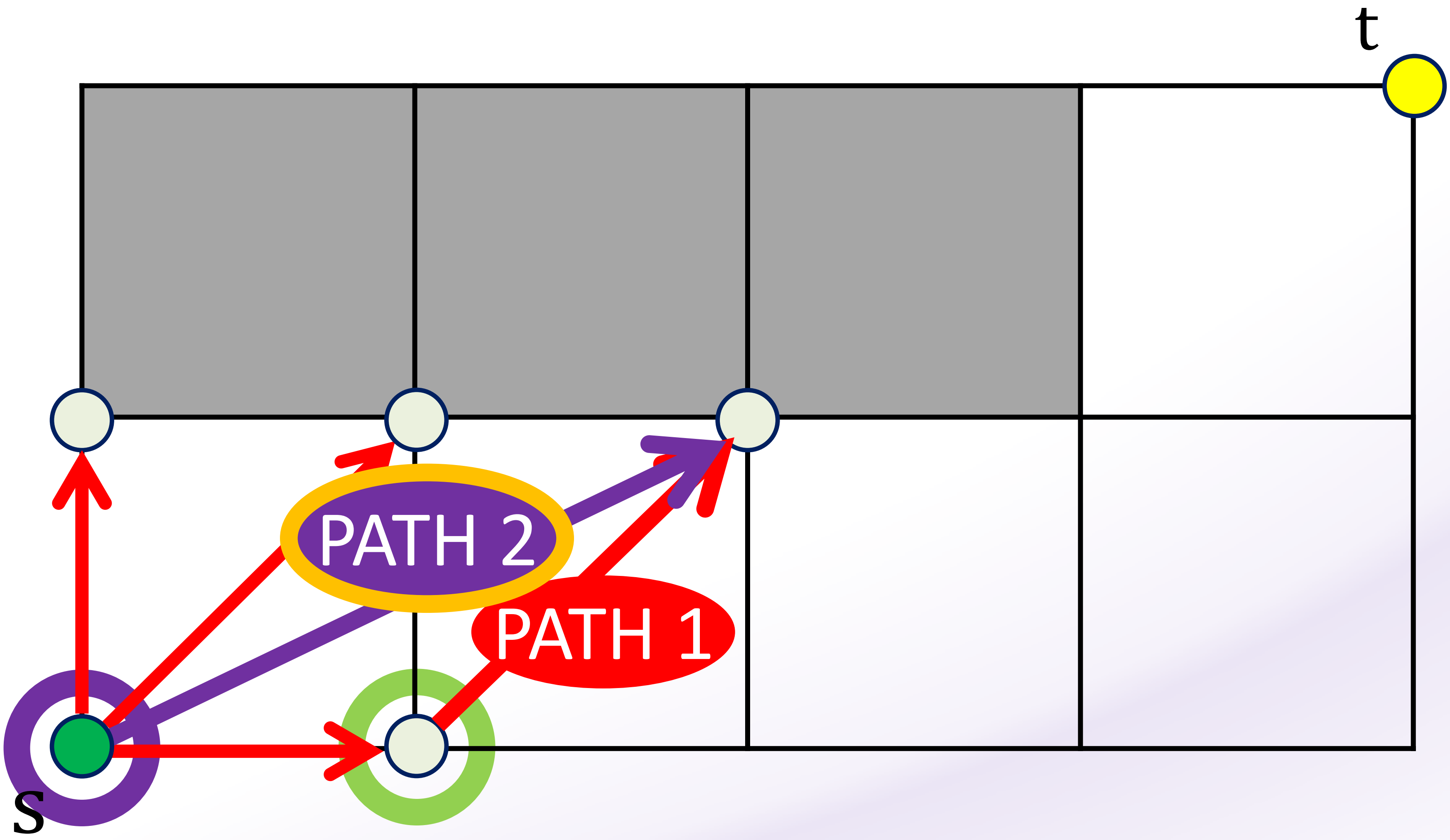
# Theta* Algorithm



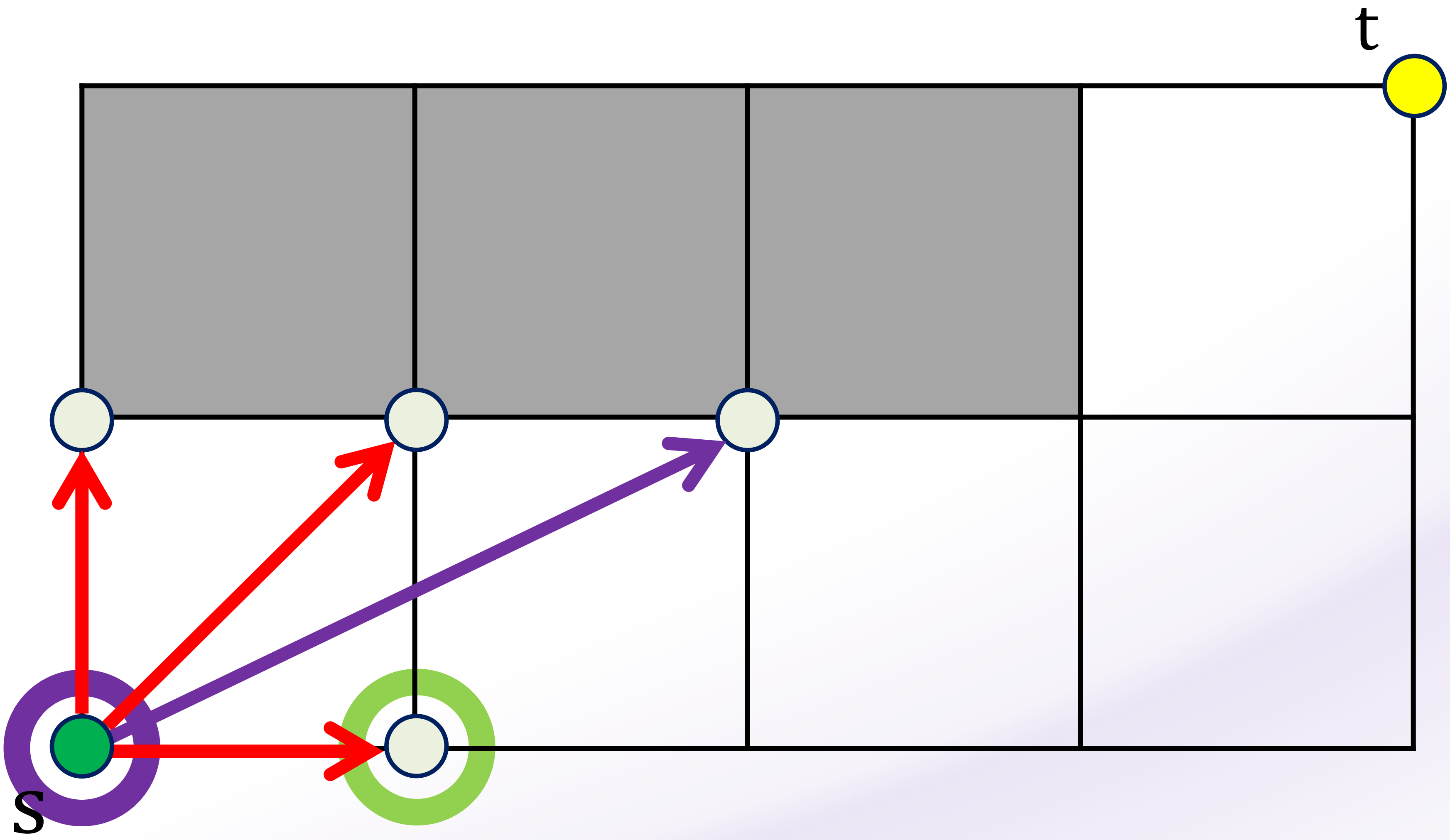
# Theta* Algorithm



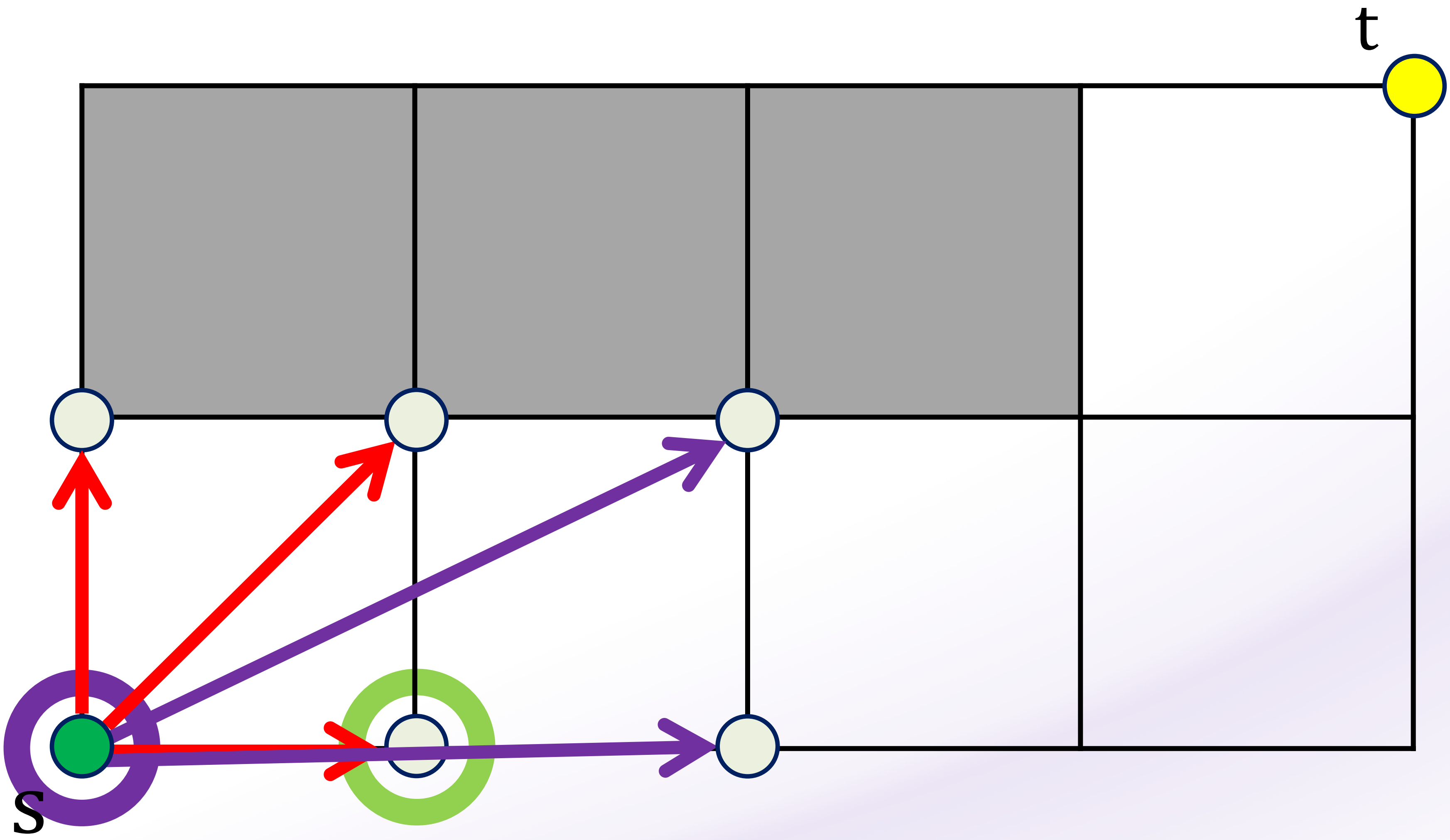
# Theta* Algorithm



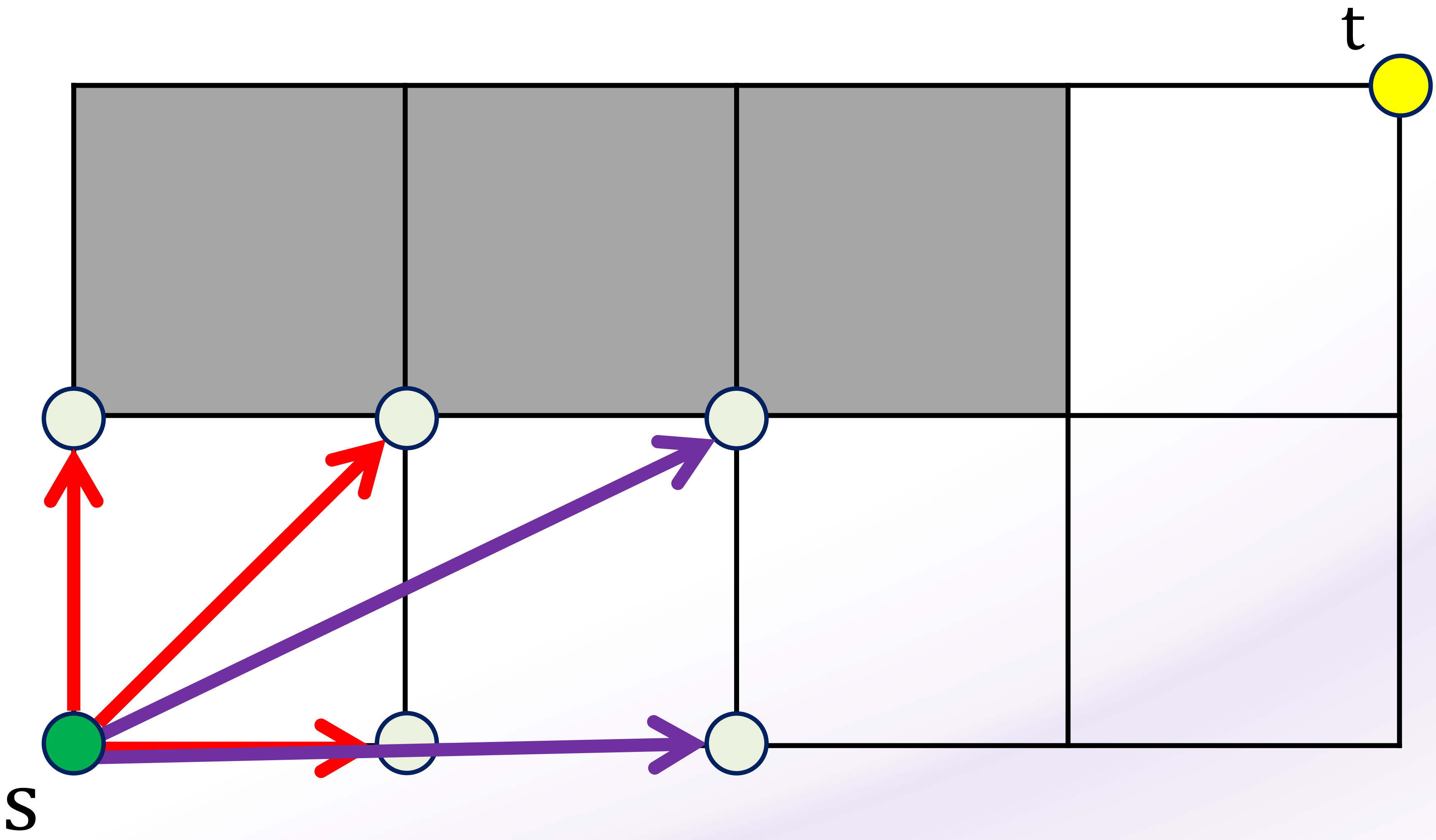
# Theta* Algorithm



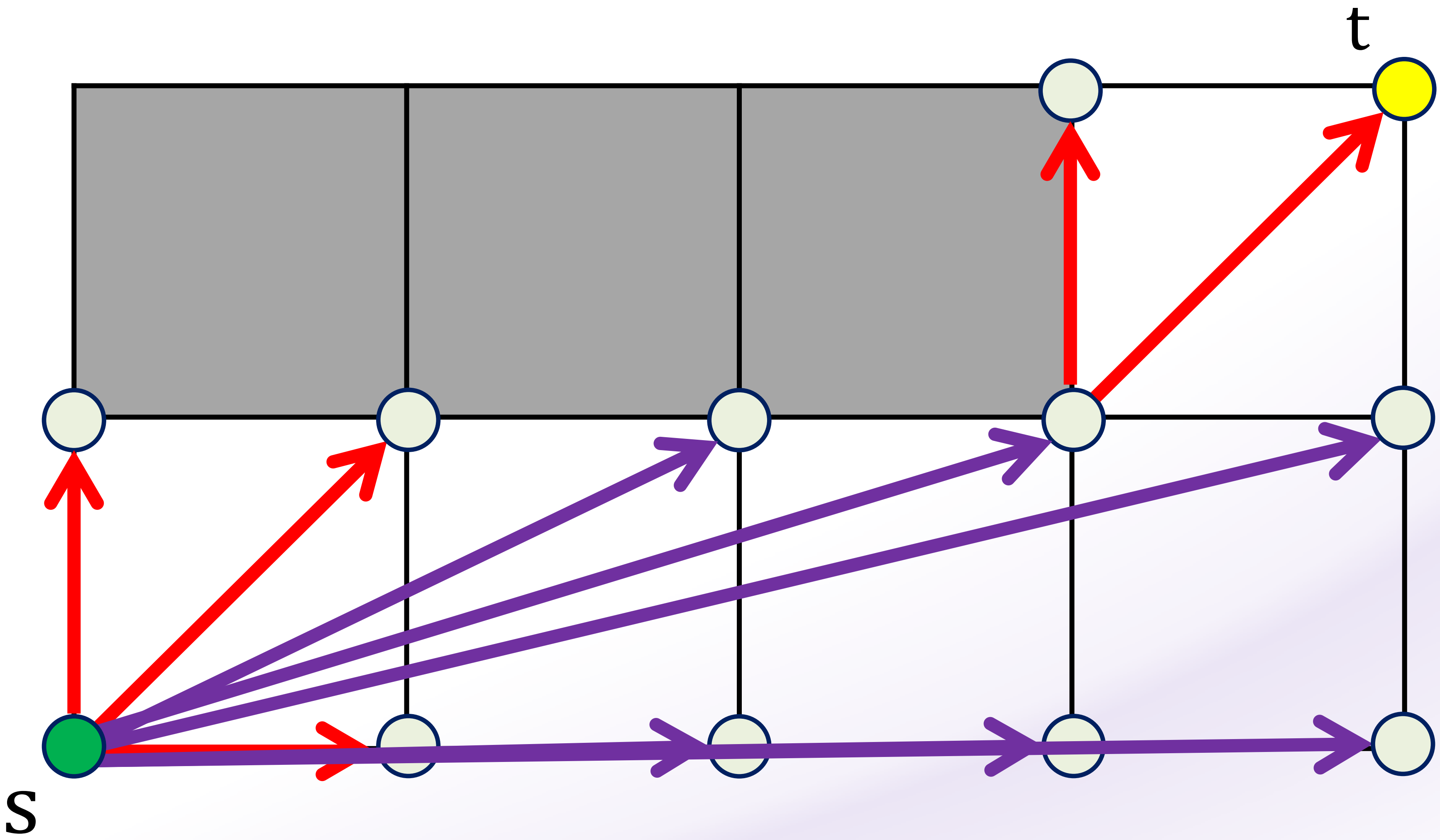
# Theta* Algorithm



# Theta* Algorithm

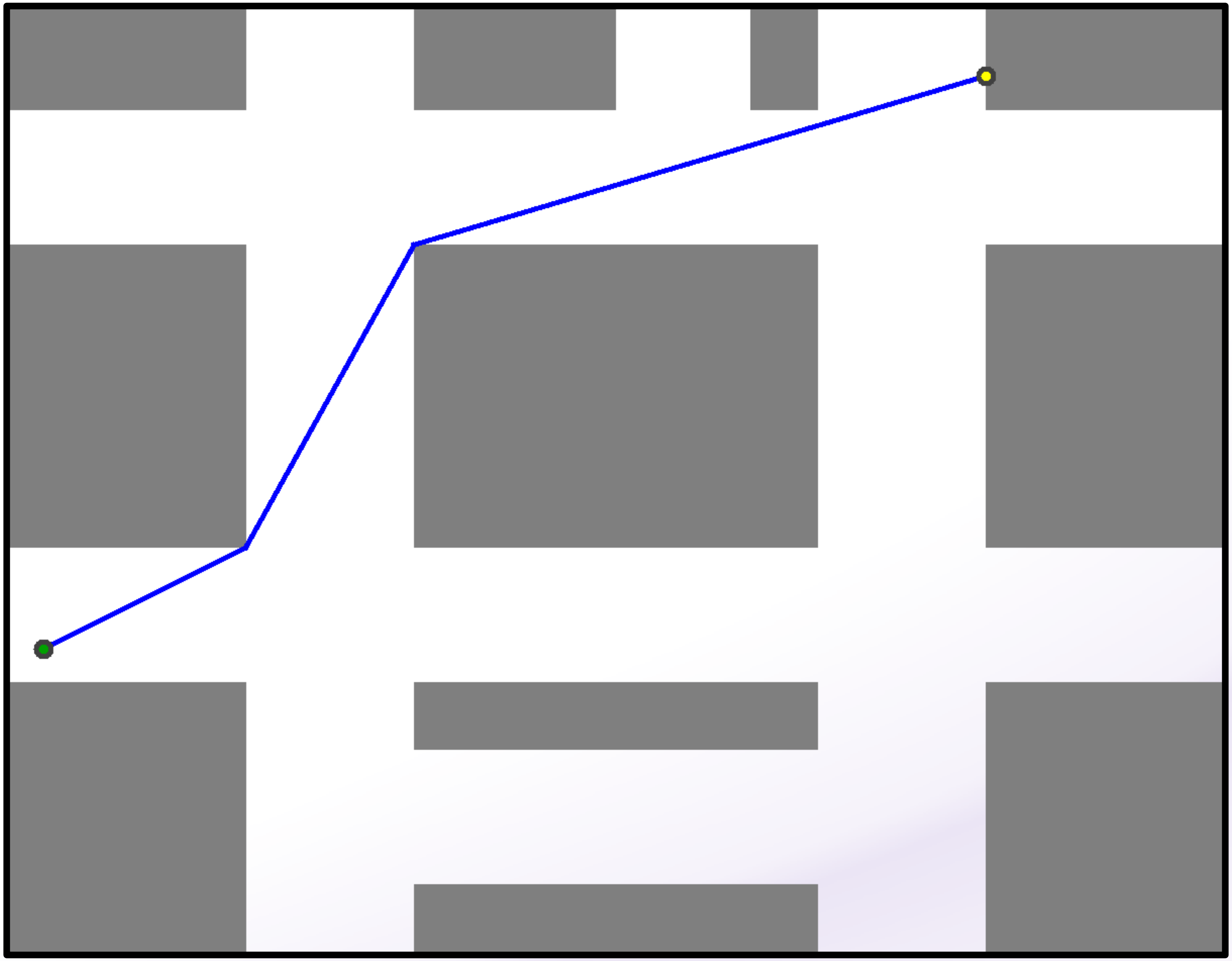


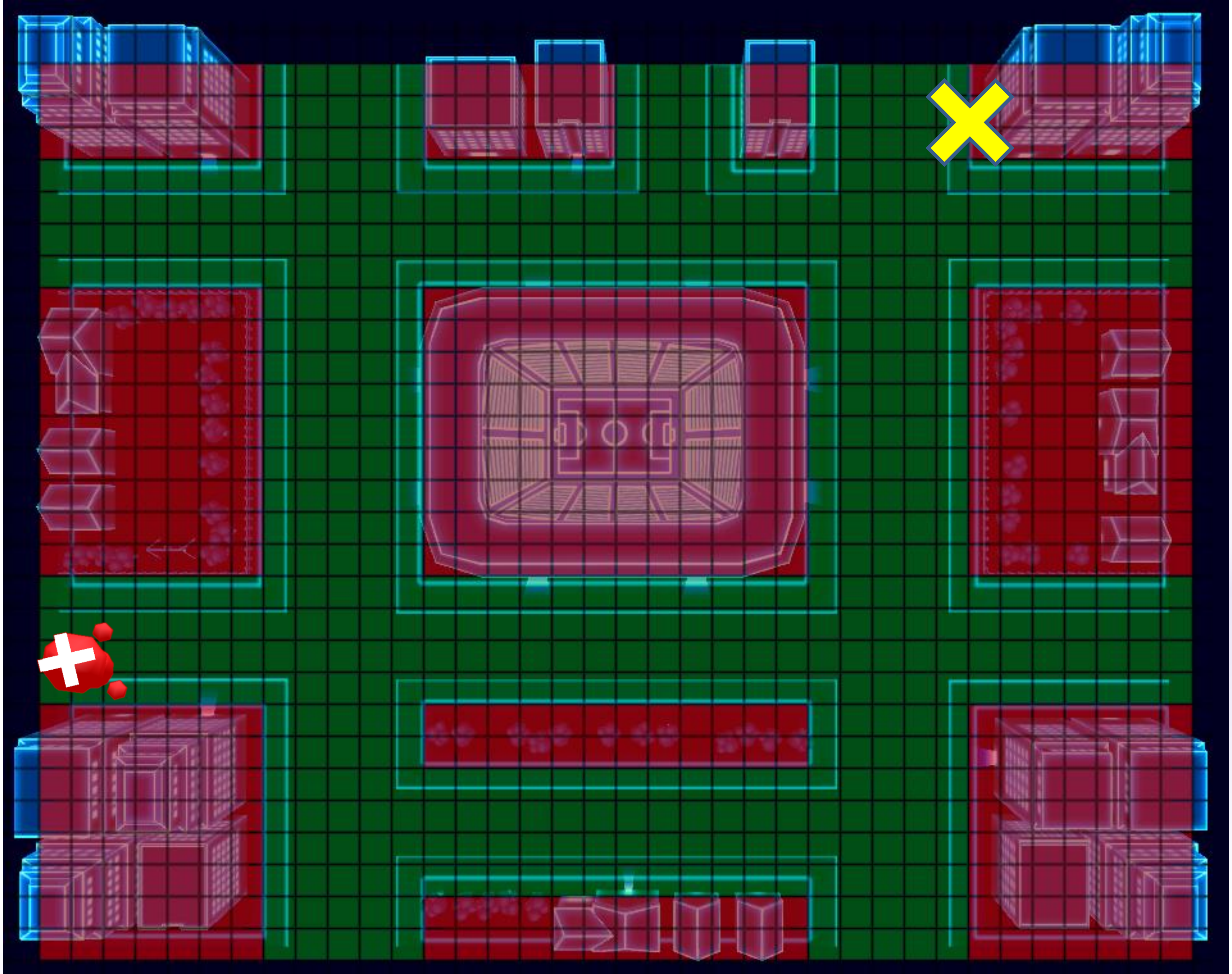
# Theta* Algorithm

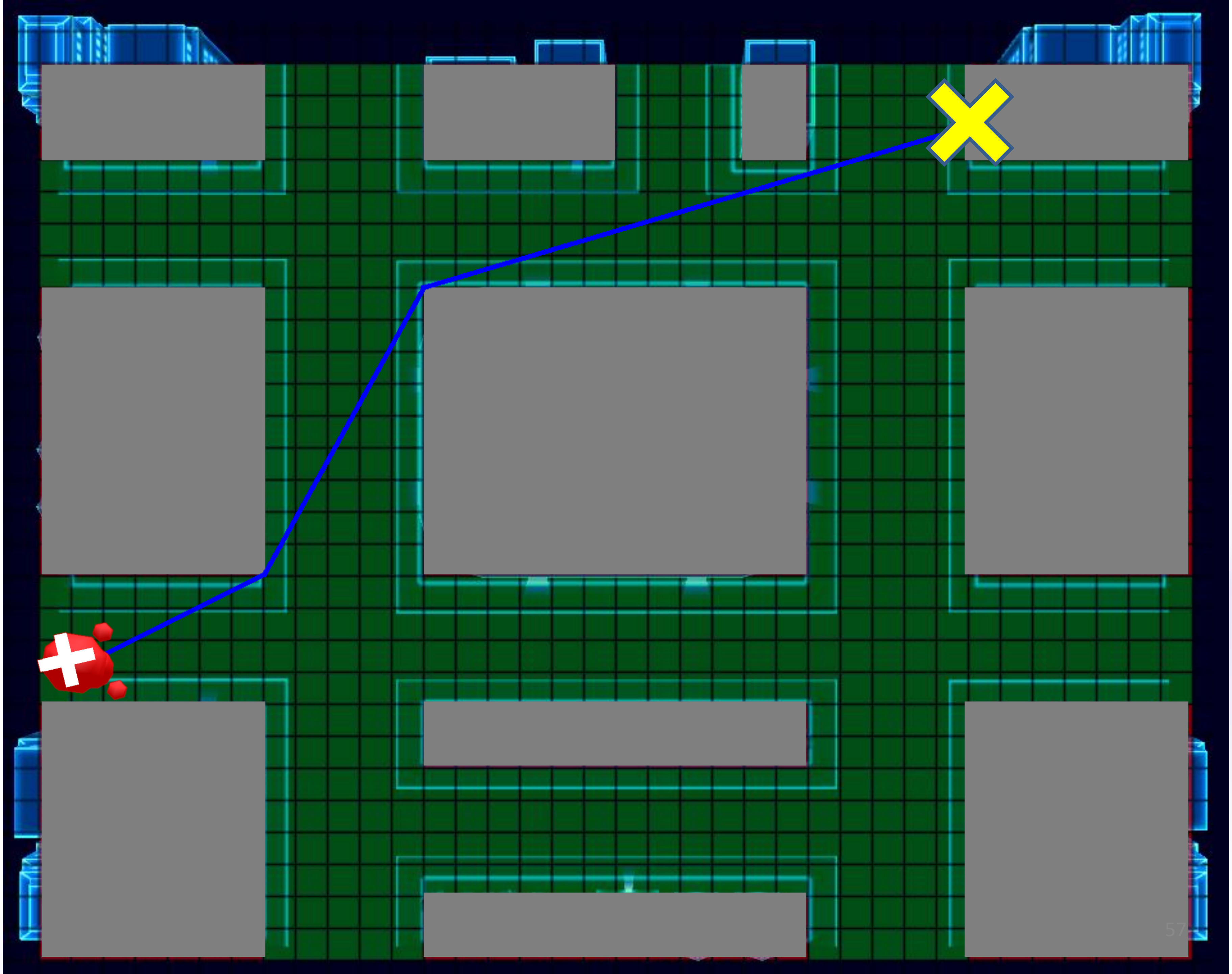


# Theta* Algorithm Demo

ThetaStar_Demo
ThetaStar_Big

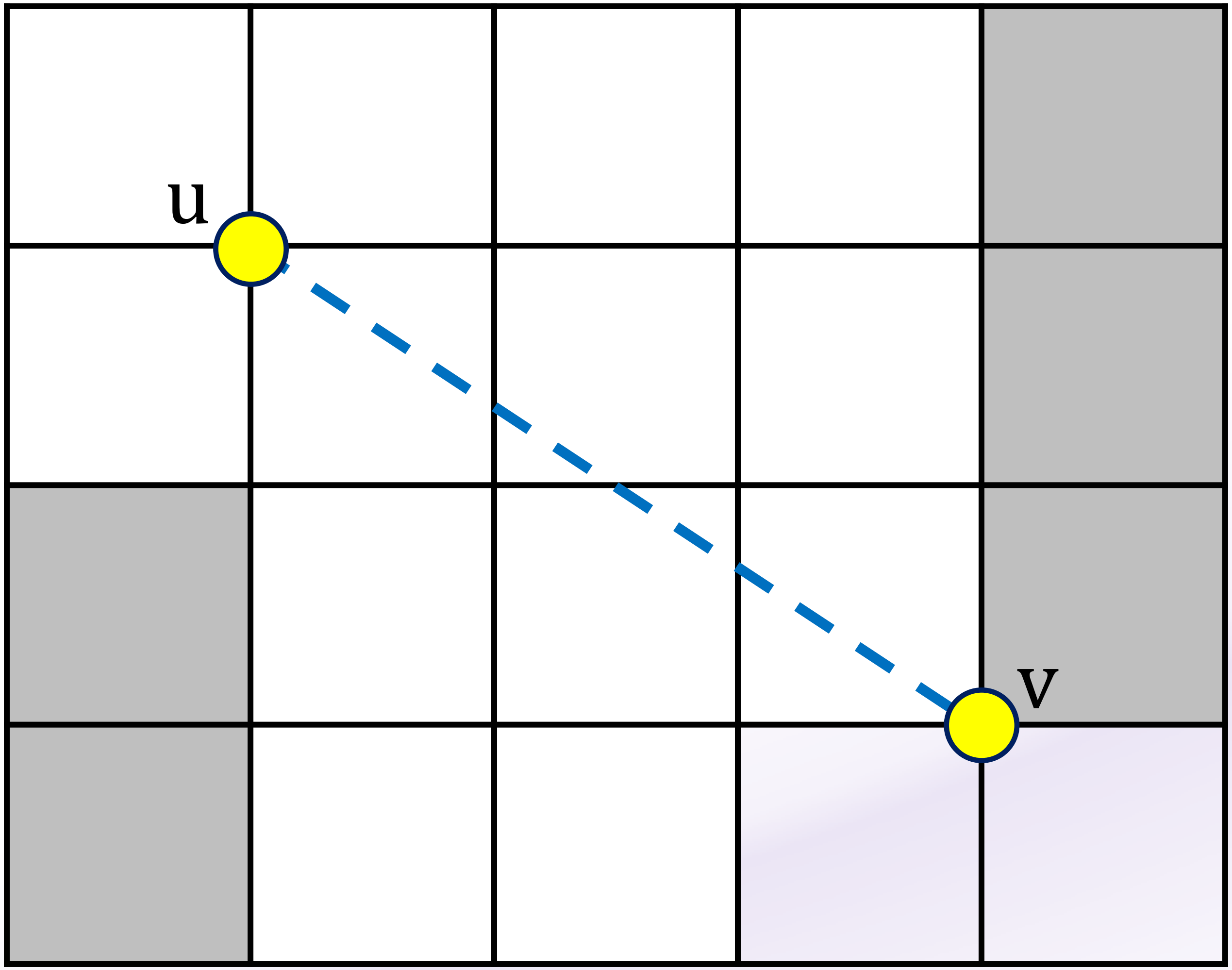




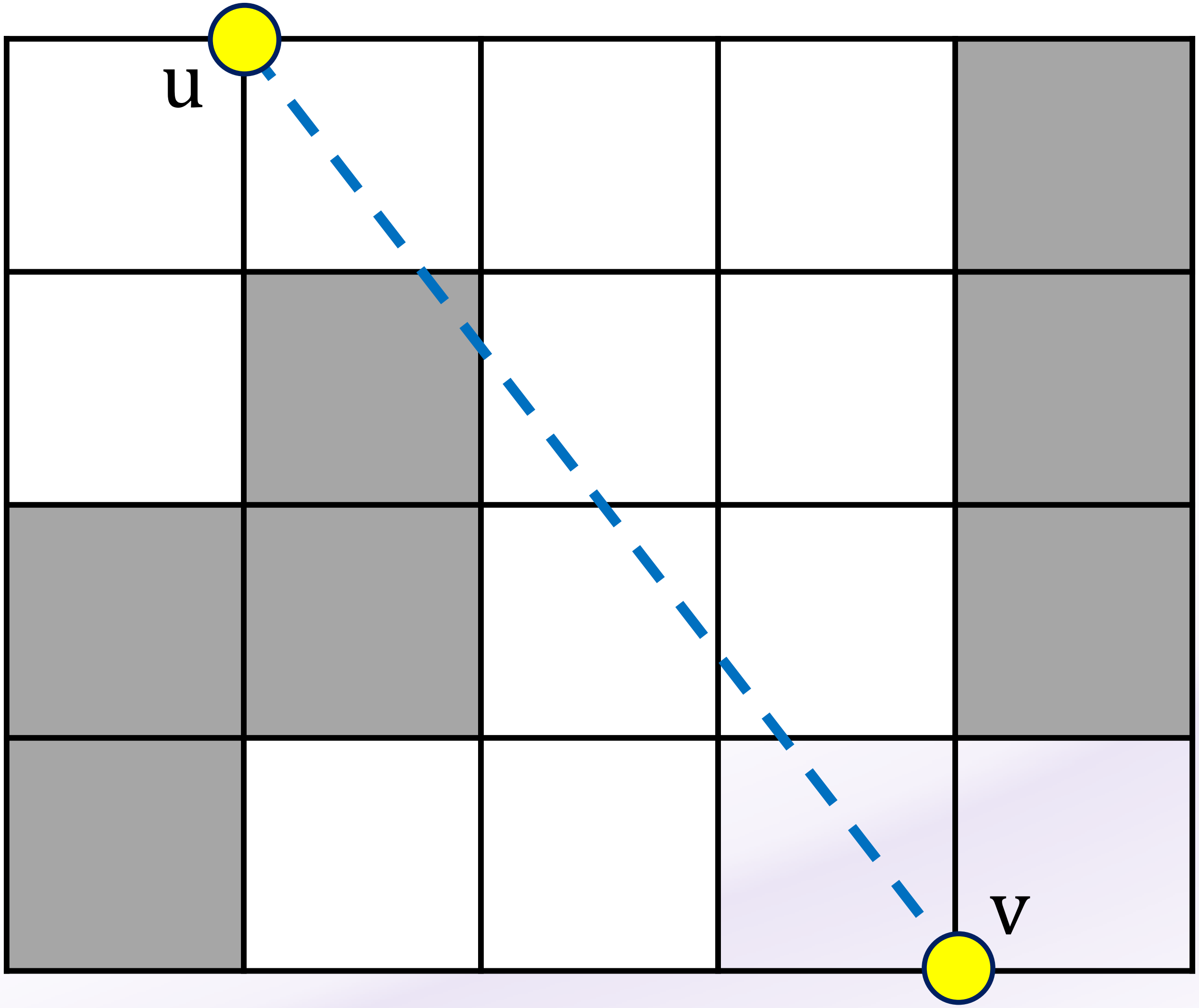


# How do I check Line-of-Sight?

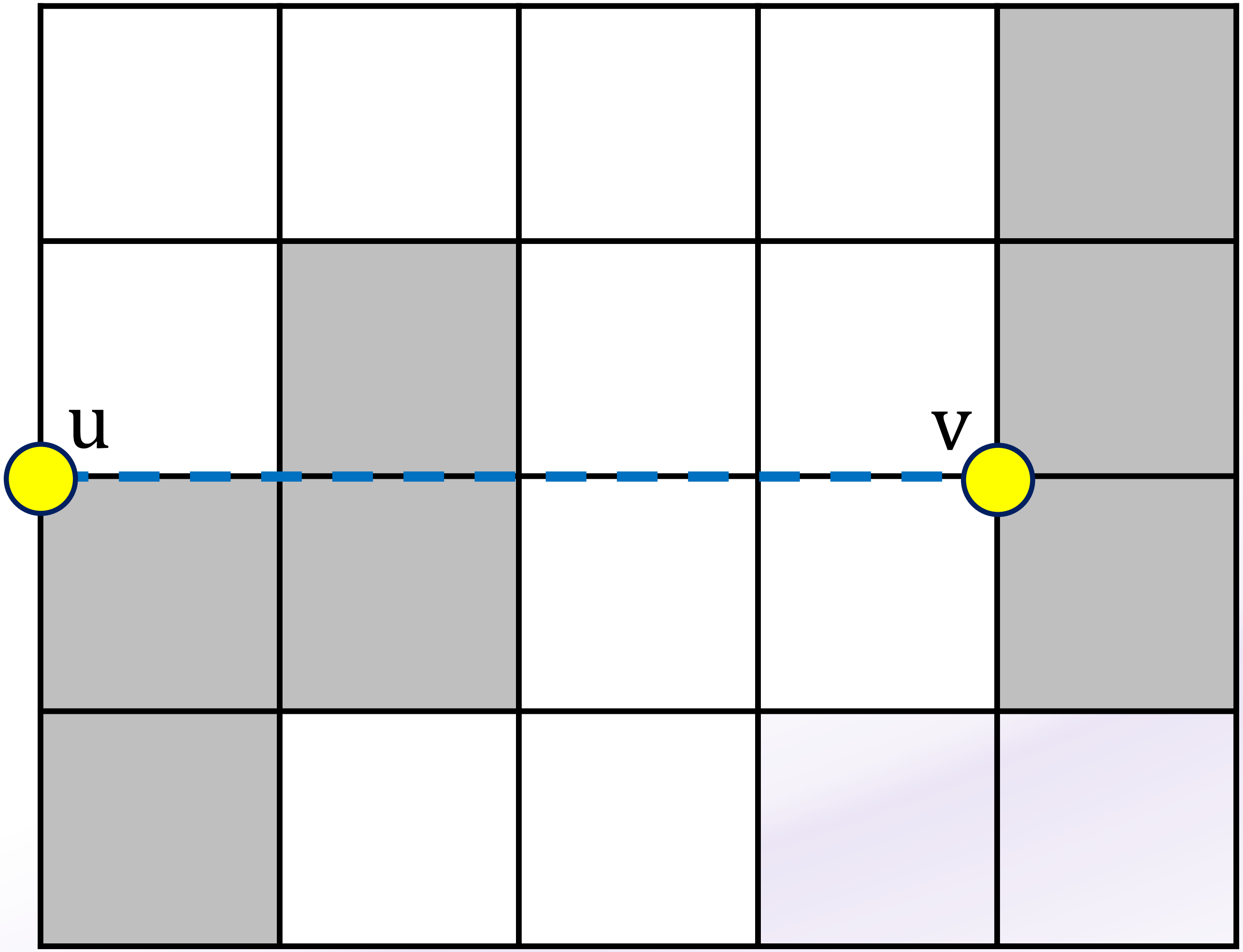
# Example: Has Line-of-Sight!



# Example: No Line-of-Sight!

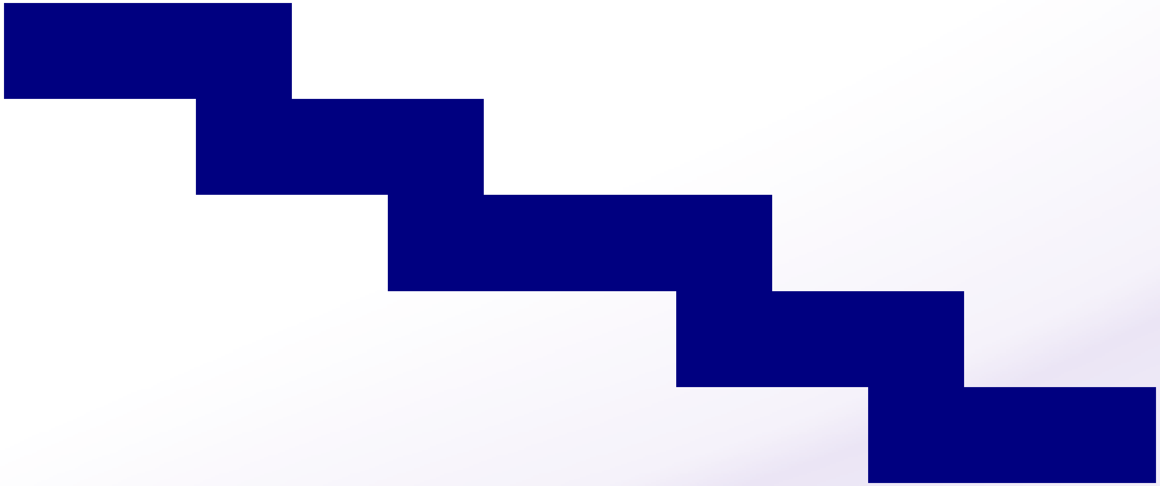


# Example: No Line-of-Sight!

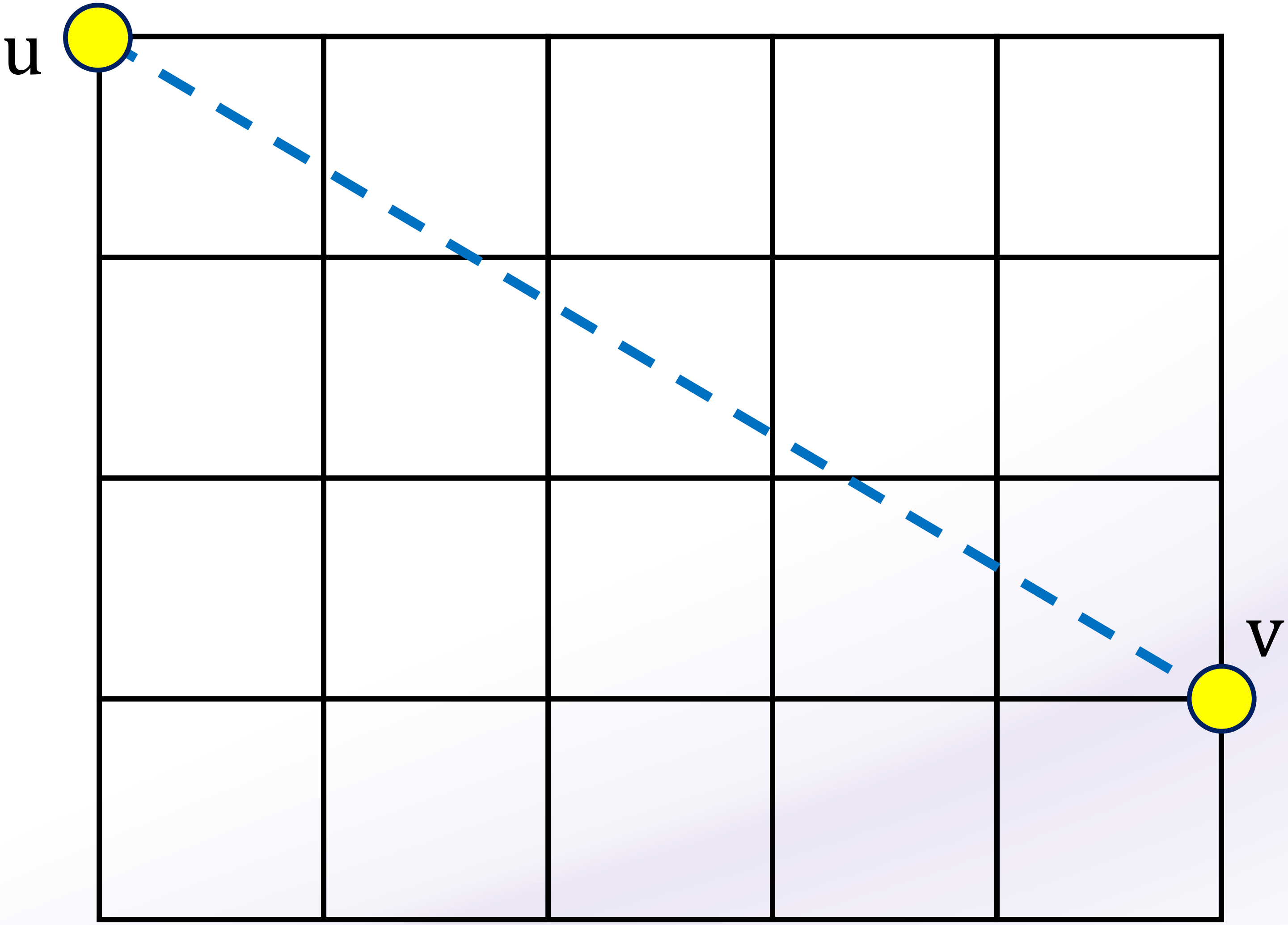


# Bresenham's Line-Drawing Algorithm

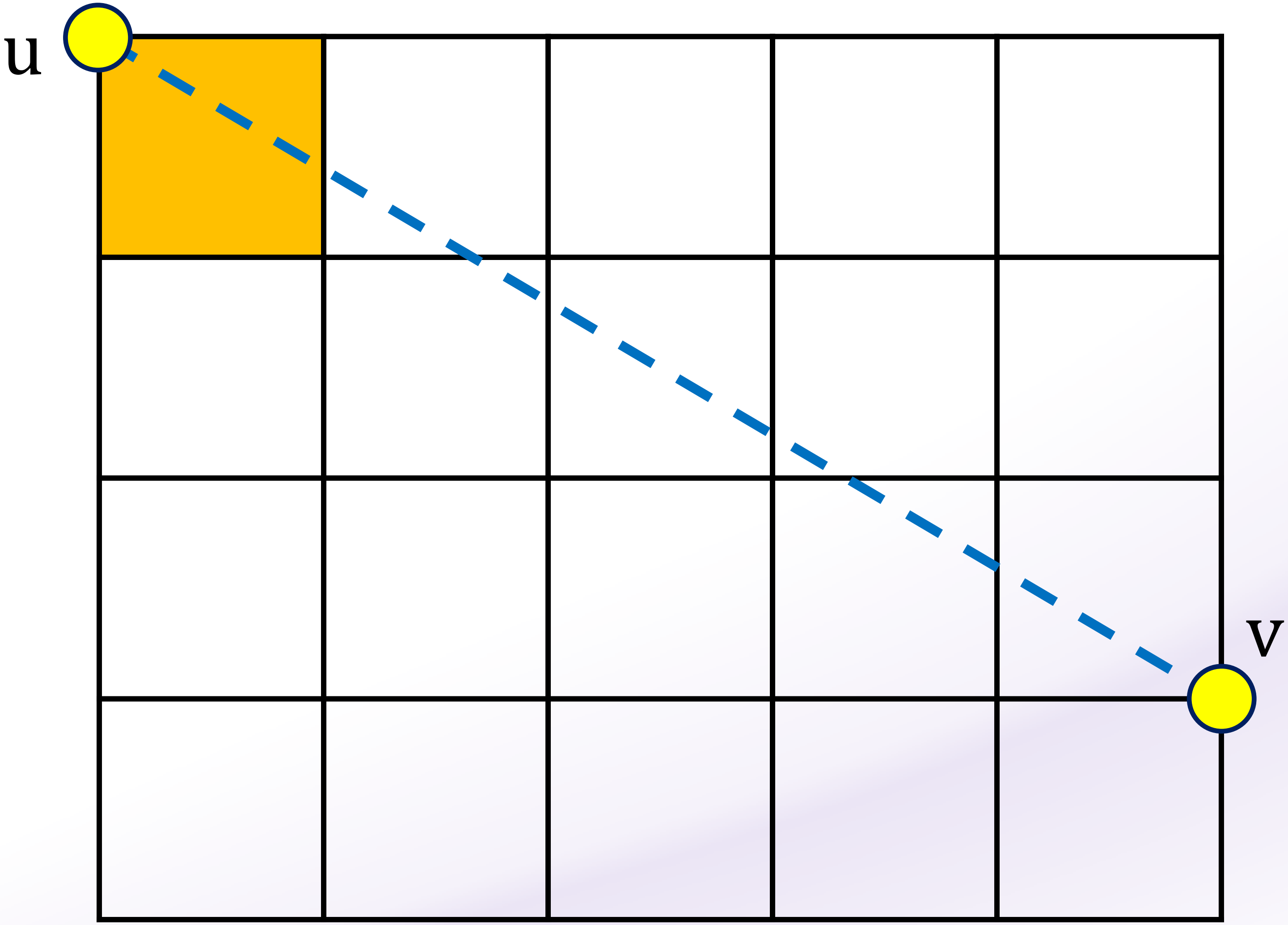
# Bresenham's Line-Drawing Algorithm



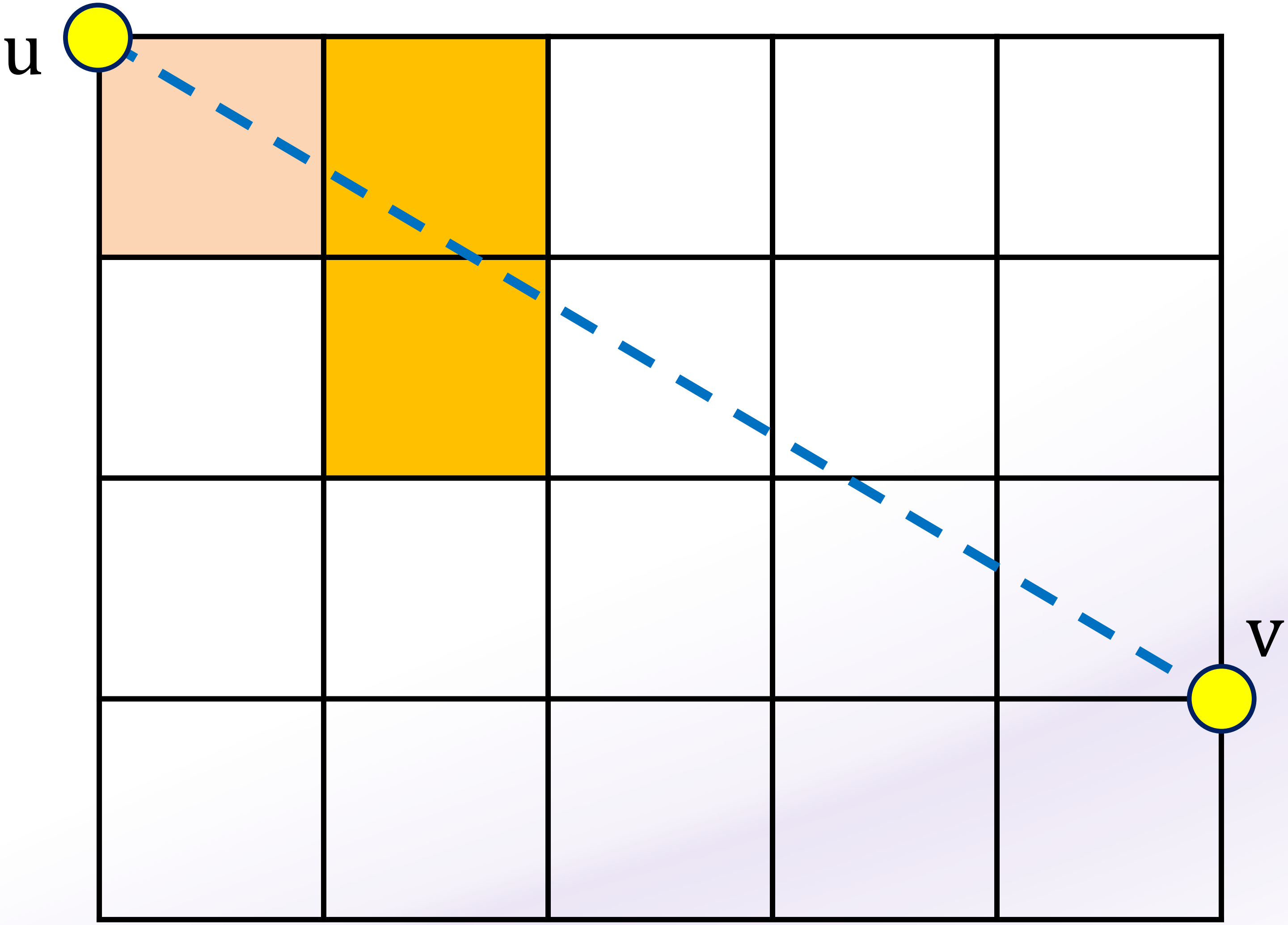
# Bressenham's line-drawing algorithm
# Check grid squares in this order:



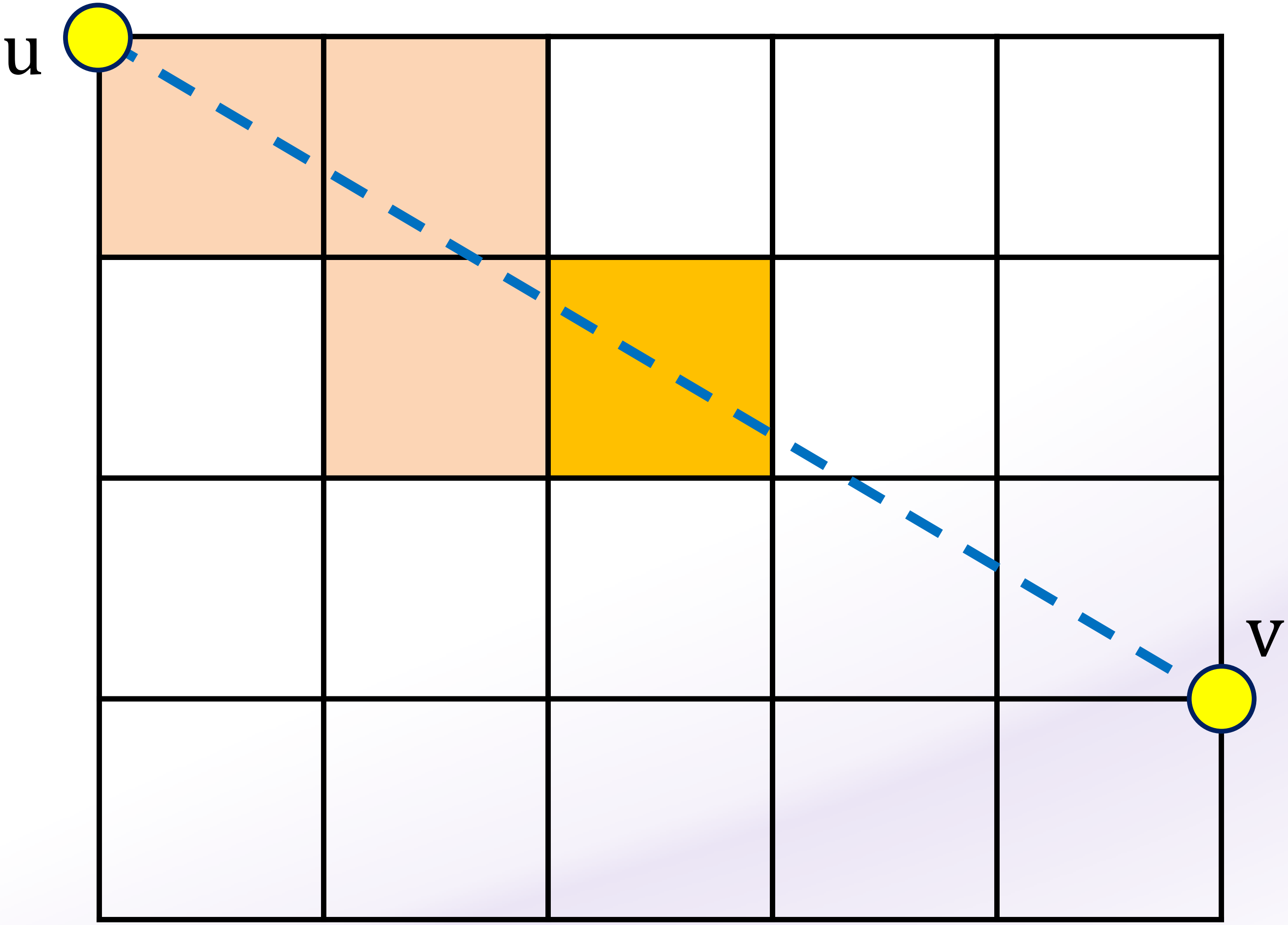
# Bressenham’s line-drawing algorithm
# Check grid squares in this order:



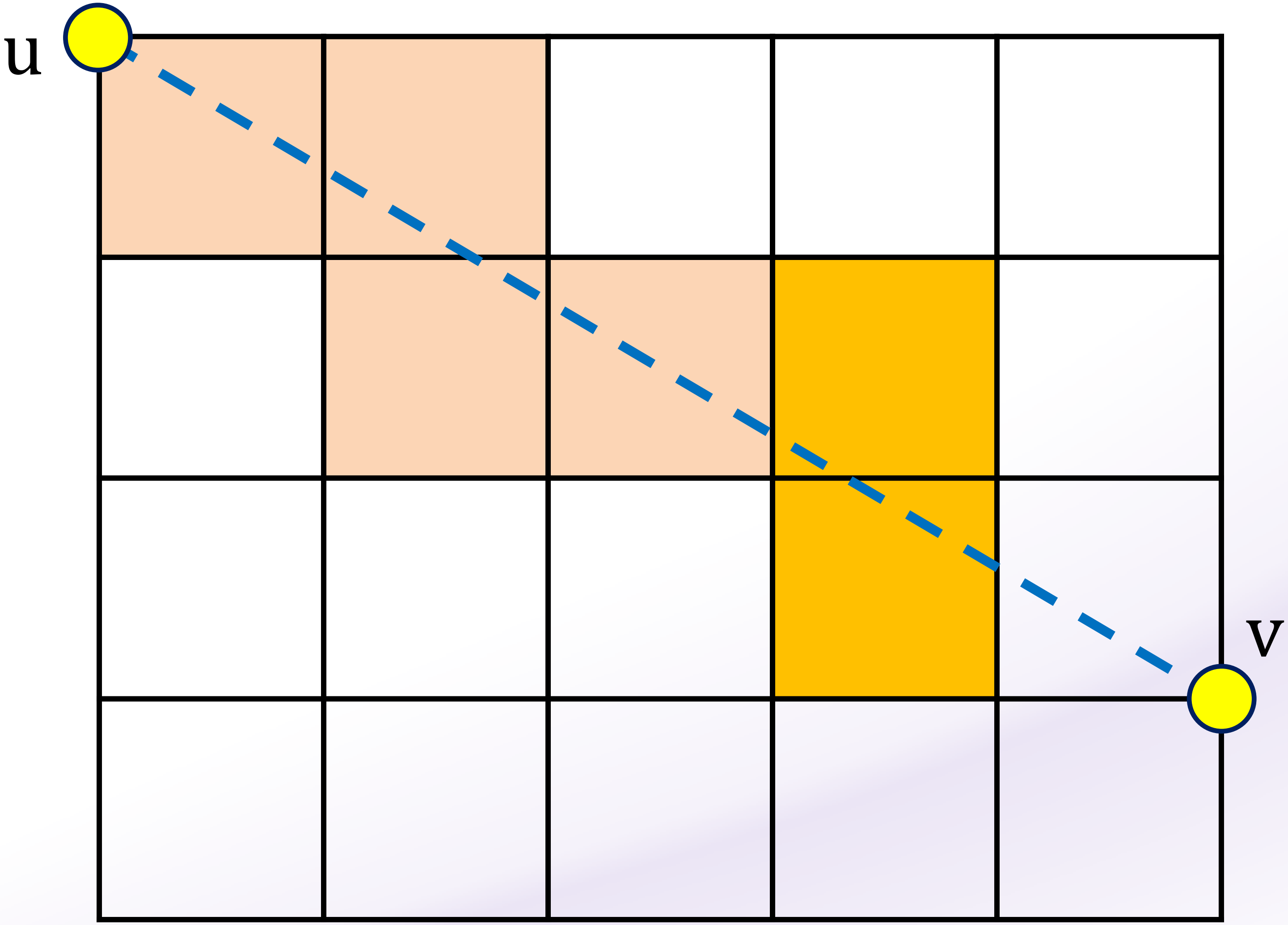
# Bressenham's line-drawing algorithm
# Check grid squares in this order:



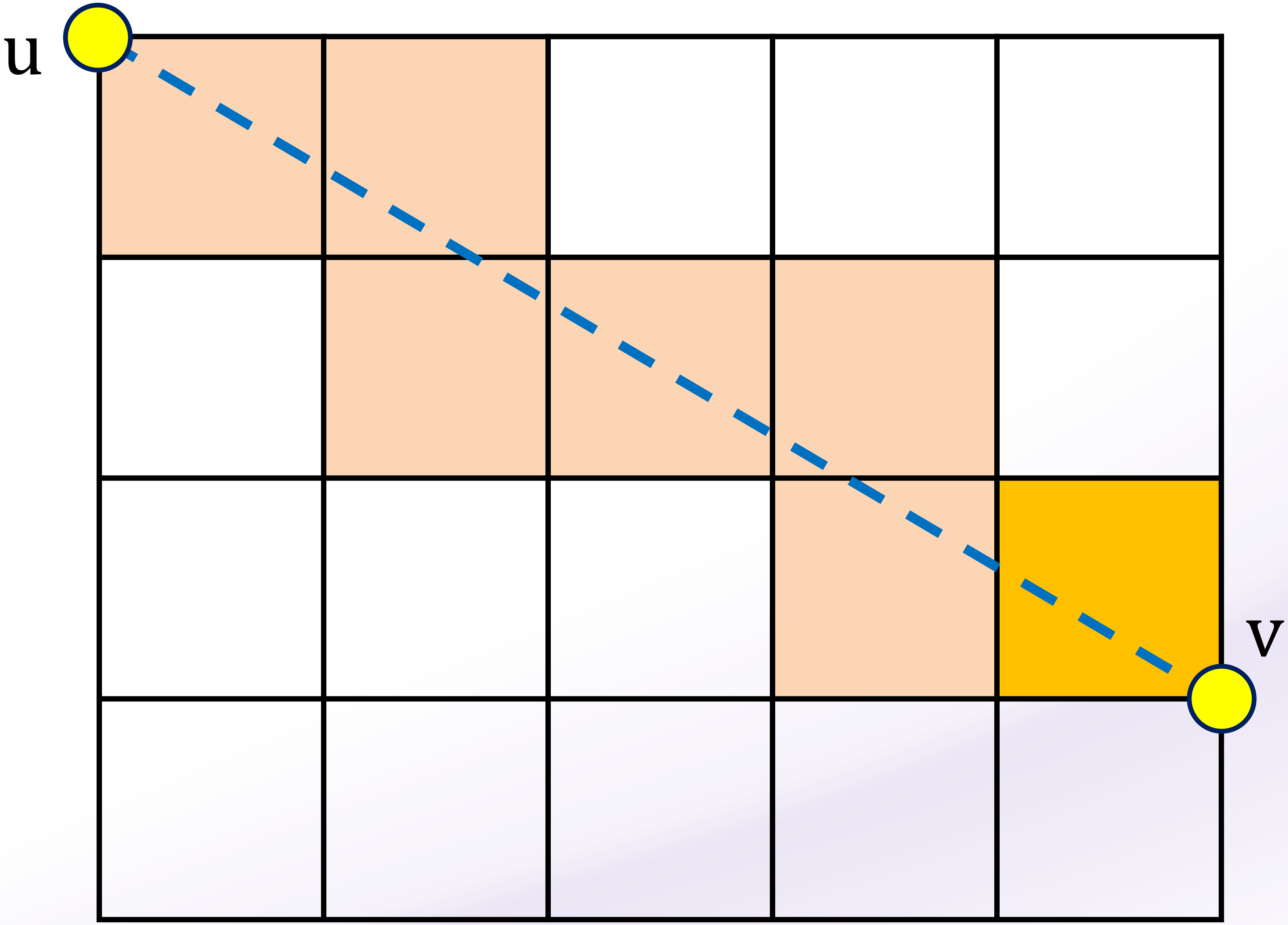
# Bressenham’s line-drawing algorithm
# Check grid squares in this order:



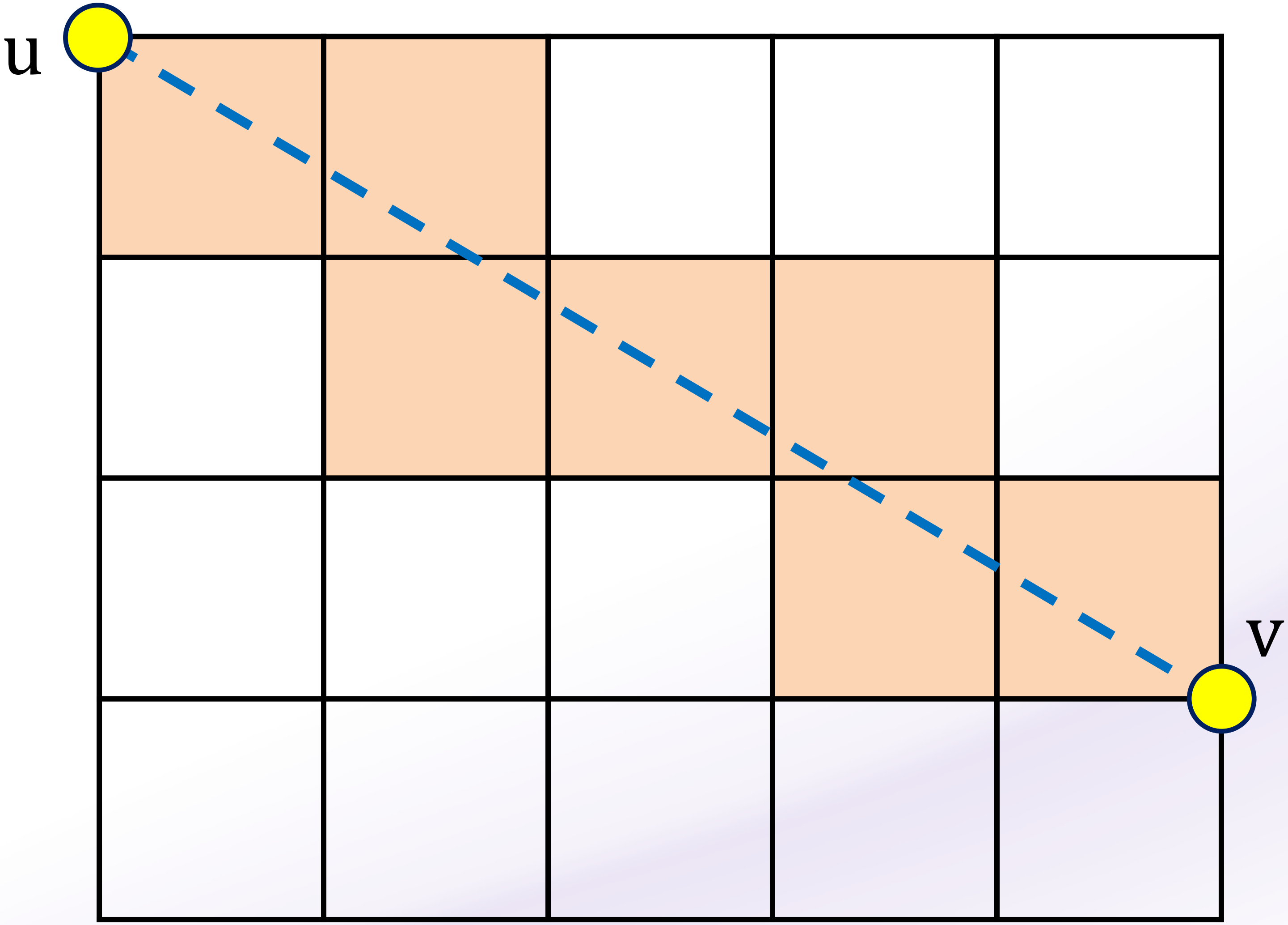
# Bressenham’s line-drawing algorithm
# Check grid squares in this order:

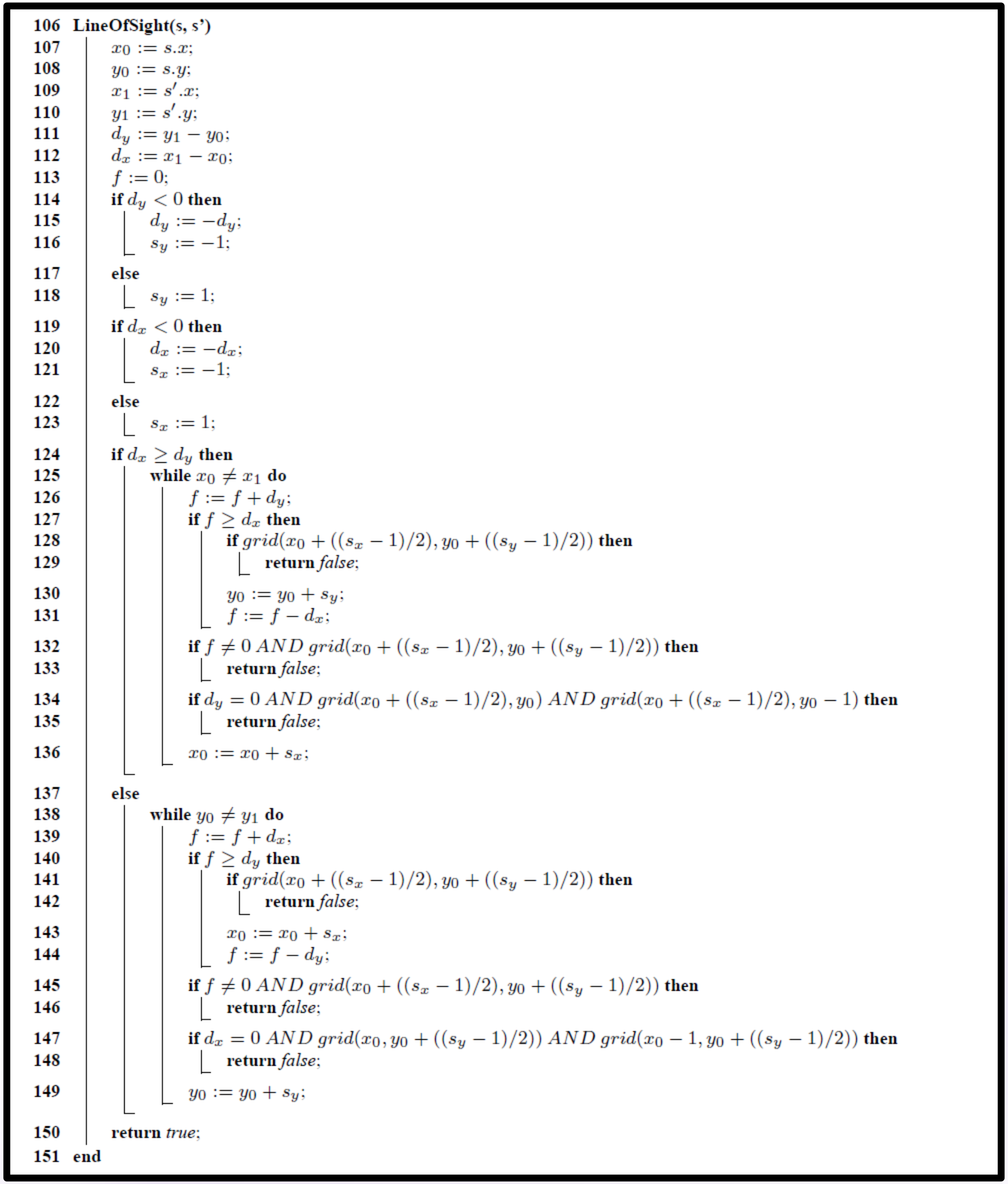


# Bressenham’s line-drawing algorithm
# Check grid squares in this order:



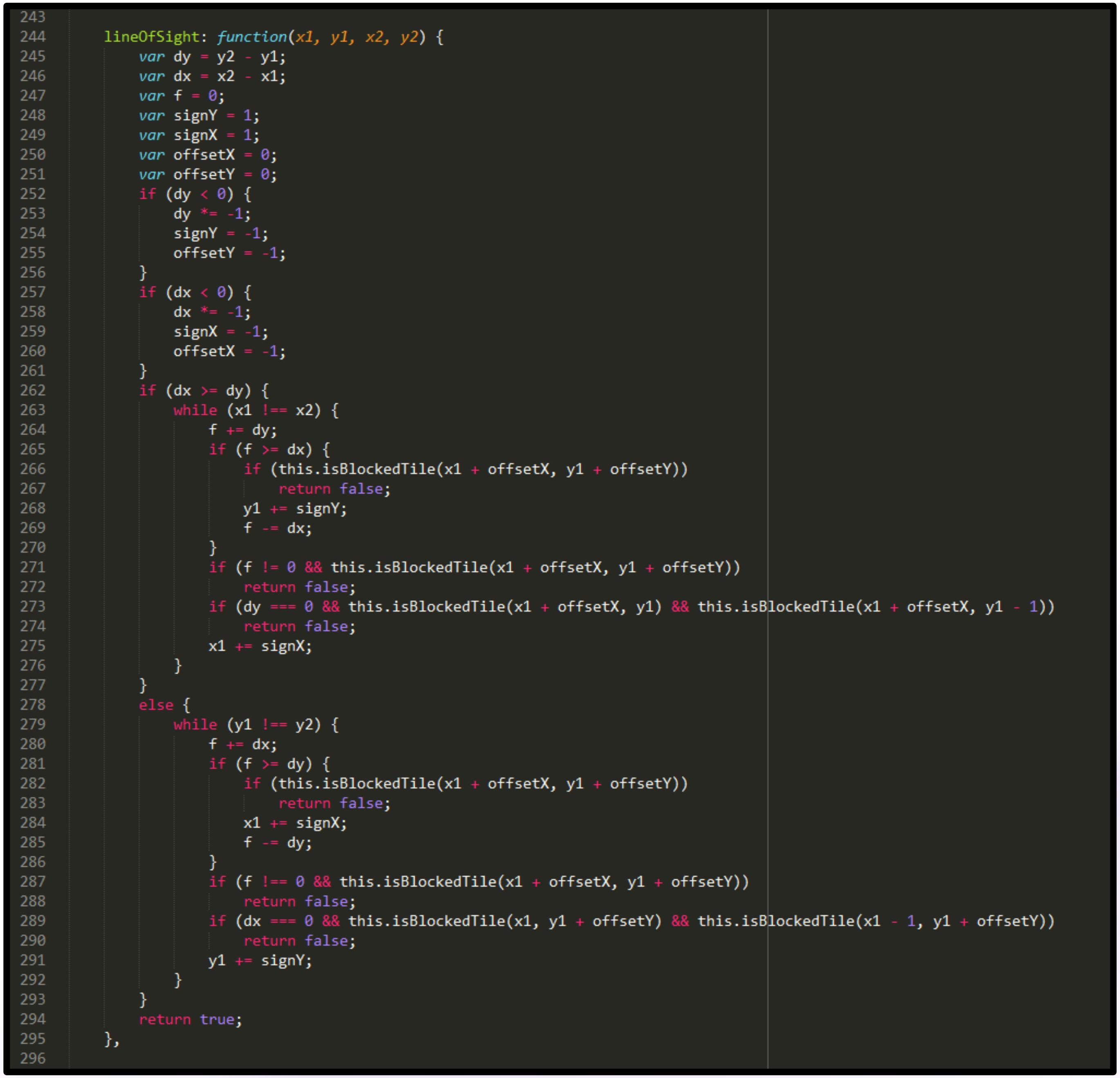
# There is Line of Sight if and only if none of these squares are blocked.





```
**LineOfSight(s, s')**
    $x_0 := s.x$;
    $y_0 := s.y$;
    $x_1 := s'.x$;
    $y_1 := s'.y$;
    $d_y := y_1 - y_0$;
    $d_x := x_1 - x_0$;
    $f := 0$;
    if $d_y < 0$ then
        $d_y := -d_y$;
        $s_y := -1$;
    else
        $s_y := 1$;
    if $d_x < 0$ then
        $d_x := -d_x$;
        $s_x := -1$;
    else
        $s_x := 1$;
    if $d_x \geq d_y$ then
        while $x_0 \neq x_1$ do
            $f := f + d_y$;
            if $f \geq d_x$ then
                if $grid(x_0 + ((s_x - 1)/2), y_0 + ((s_y - 1)/2))$ then
                    return false;
                $y_0 := y_0 + s_y$;
                $f := f - d_x$;
            if $f \neq 0$ AND $grid(x_0 + ((s_x - 1)/2), y_0 + ((s_y - 1)/2))$ then
                return false;
            if $d_y = 0$ AND $grid(x_0 + ((s_x - 1)/2), y_0)$ AND $grid(x_0 + ((s_x - 1)/2), y_0 - 1)$ then
                return false;
            $x_0 := x_0 + s_x$;
    else
        while $y_0 \neq y_1$ do
            $f := f + d_x$;
            if $f \geq d_y$ then
                if $grid(x_0 + ((s_x - 1)/2), y_0 + ((s_y - 1)/2))$ then
                    return false;
                $x_0 := x_0 + s_x$;
                $f := f - d_y$;
            if $f \neq 0$ AND $grid(x_0 + ((s_x - 1)/2), y_0 + ((s_y - 1)/2))$ then
                return false;
            if $d_x = 0$ AND $grid(x_0, y_0 + ((s_y - 1)/2))$ AND $grid(x_0 - 1, y_0 + ((s_y - 1)/2))$ then
                return false;
            $y_0 := y_0 + s_y$;
    return true;
end
```

```java
/**
 * @return true iff there is line-of-sight from (x1,y1) to (x2,y2).
 */
public boolean lineOfSight(int x1, int y1, int x2, int y2) {
    int dy = y2 - y1;
    int dx = x2 - x1;

    int f = 0;

    int signY = 1;
    int signX = 1;
    int offsetX = 0;
    int offsetY = 0;

    if (dy < 0) {
        dy *= -1;
        signY = -1;
        offsetY = -1;
    }
    if (dx < 0) {
        dx *= -1;
        signX = -1;
        offsetX = -1;
    }

    if (dx >= dy) {
        while (x1 != x2) {
            f += dy;
            if (f >= dx) {
                if (isBlocked(x1 + offsetX, y1 + offsetY))
                    return false;
                y1 += signY;
                f -= dx;
            }
            if (f != 0 && isBlocked(x1 + offsetX, y1 + offsetY))
                return false;
            if (dy == 0 && isBlocked(x1 + offsetX, y1) && isBlocked(x1 + offsetX, y1 - 1))
                return false;

            x1 += signX;
        }
    }
    else {
        while (y1 != y2) {
            f += dx;
            if (f >= dy) {
                if (isBlocked(x1 + offsetX, y1 + offsetY))
                    return false;
                x1 += signX;
                f -= dy;
            }
            if (f != 0 && isBlocked(x1 + offsetX, y1 + offsetY))
                return false;
            if (dx == 0 && isBlocked(x1, y1 + offsetY) && isBlocked(x1 - 1, y1 + offsetY))
                return false;

            y1 += signY;
        }
    }
    return true;
}
```

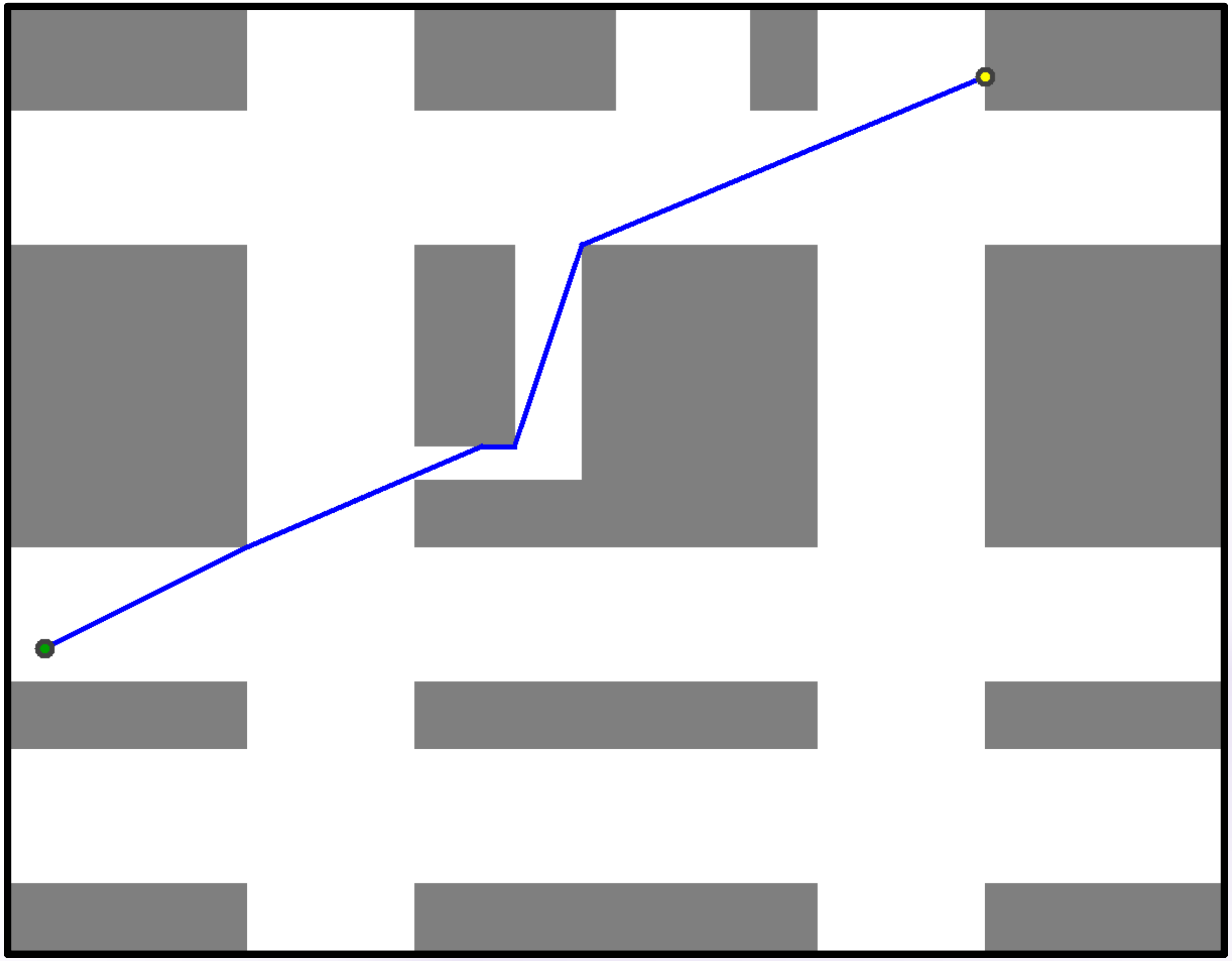


```
    lineOfSight: function(x1, y1, x2, y2) {
        var dy = y2 - y1;
        var dx = x2 - x1;
        var f = 0;
        var signY = 1;
        var signX = 1;
        var offsetX = 0;
        var offsetY = 0;
        if (dy < 0) {
            dy *= -1;
            signY = -1;
            offsetY = -1;
        }
        if (dx < 0) {
            dx *= -1;
            signX = -1;
            offsetX = -1;
        }
        if (dx >= dy) {
            while (x1 !== x2) {
                f += dy;
                if (f >= dx) {
                    if (this.isBlockedTile(x1 + offsetX, y1 + offsetY))
                        return false;
                    y1 += signY;
                    f -= dx;
                }
                if (f != 0 && this.isBlockedTile(x1 + offsetX, y1 + offsetY))
                    return false;
                if (dy === 0 && this.isBlockedTile(x1 + offsetX, y1) && this.isBlockedTile(x1 + offsetX, y1 - 1))
                    return false;
                x1 += signX;
            }
        }
        else {
            while (y1 !== y2) {
                f += dx;
                if (f >= dy) {
                    if (this.isBlockedTile(x1 + offsetX, y1 + offsetY))
                        return false;
                    x1 += signX;
                    f -= dy;
                }
                if (f !== 0 && this.isBlockedTile(x1 + offsetX, y1 + offsetY))
                    return false;
                if (dx === 0 && this.isBlockedTile(x1, y1 + offsetY) && this.isBlockedTile(x1 - 1, y1 + offsetY))
                    return false;
                y1 += signY;
            }
        }
        return true;
    },
```

# Theta*
# is not Optimal
# Demo

ThetaStar_Hard

# Theta* is not optimal!



# Any-Angle Pathfinding Algorithms

**Theta***

**Visibility Graphs**

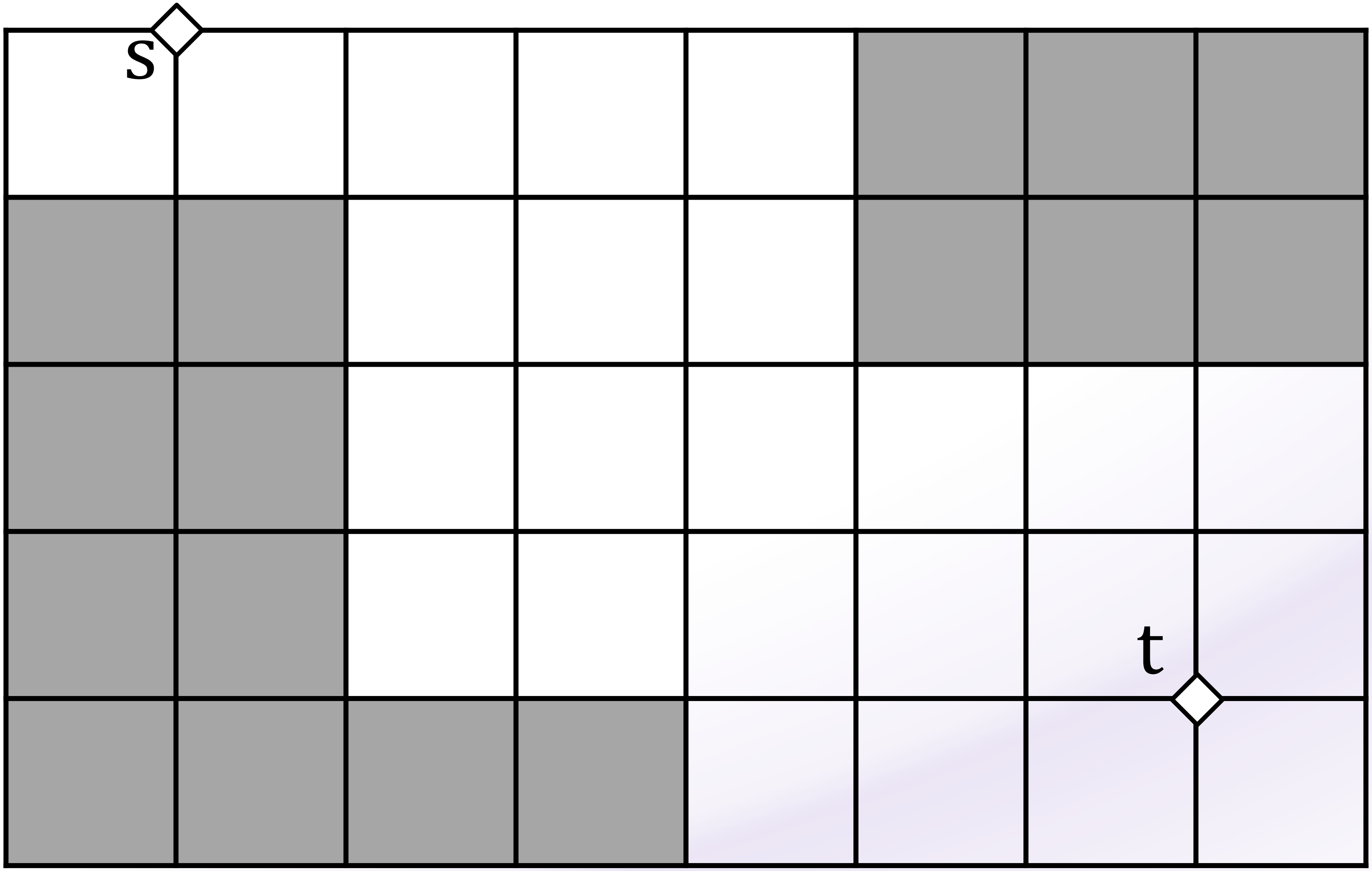
# Visibility Graph Algorithm

## Running A* on Visibility Graphs

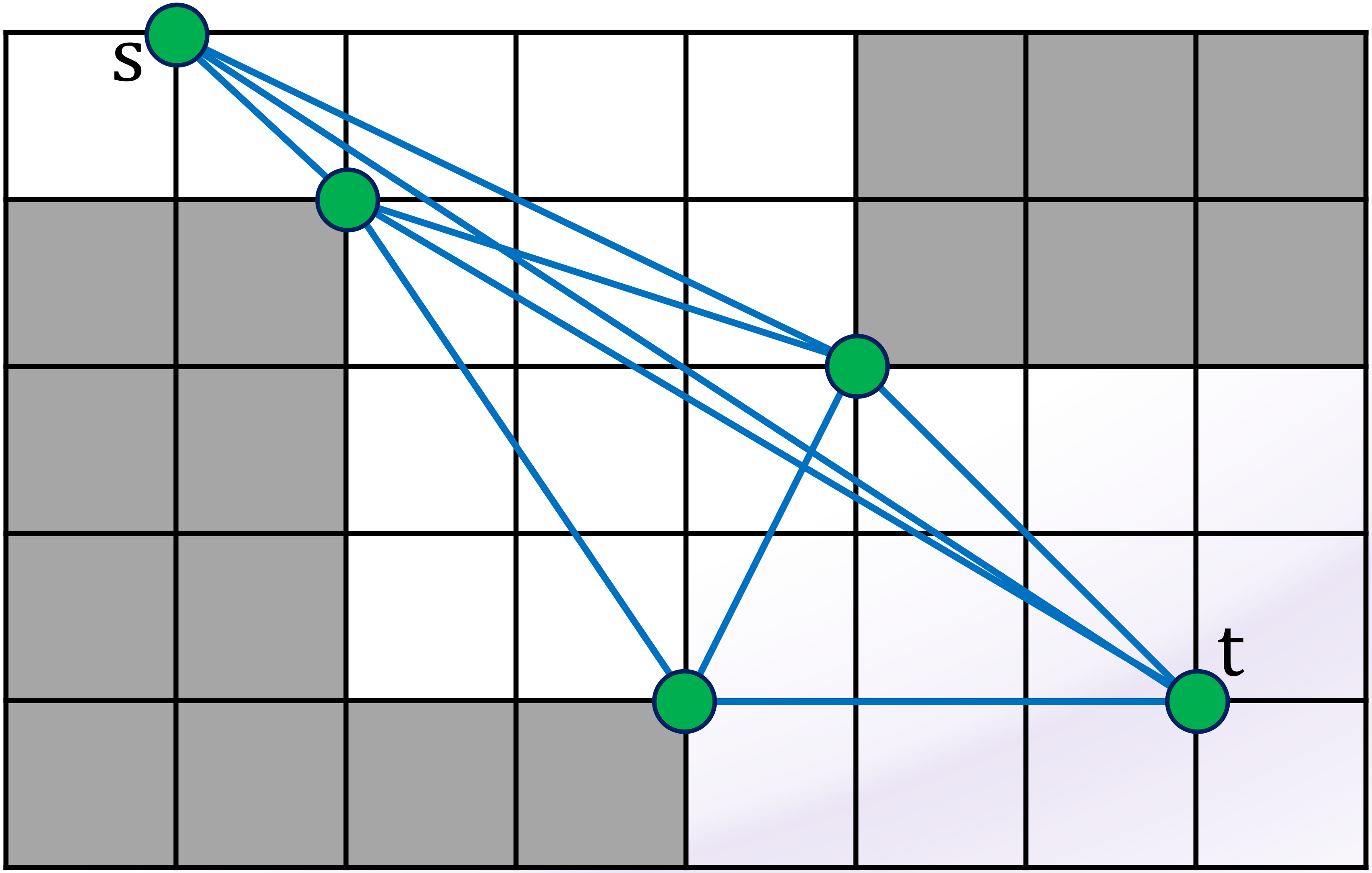
# Two Steps

1) Build Visibility Graph
2) Run A* on it

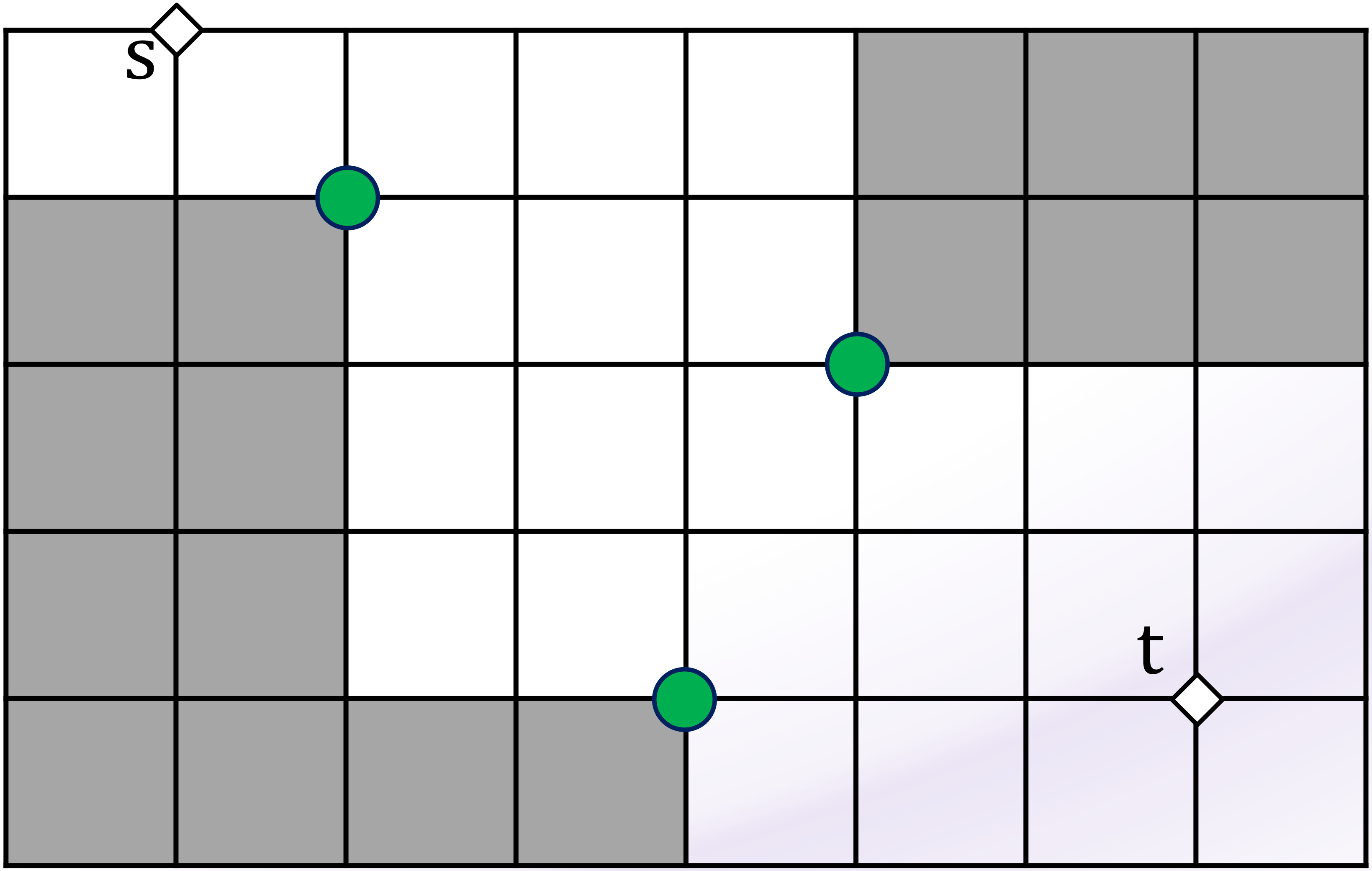
# Visibility Graph



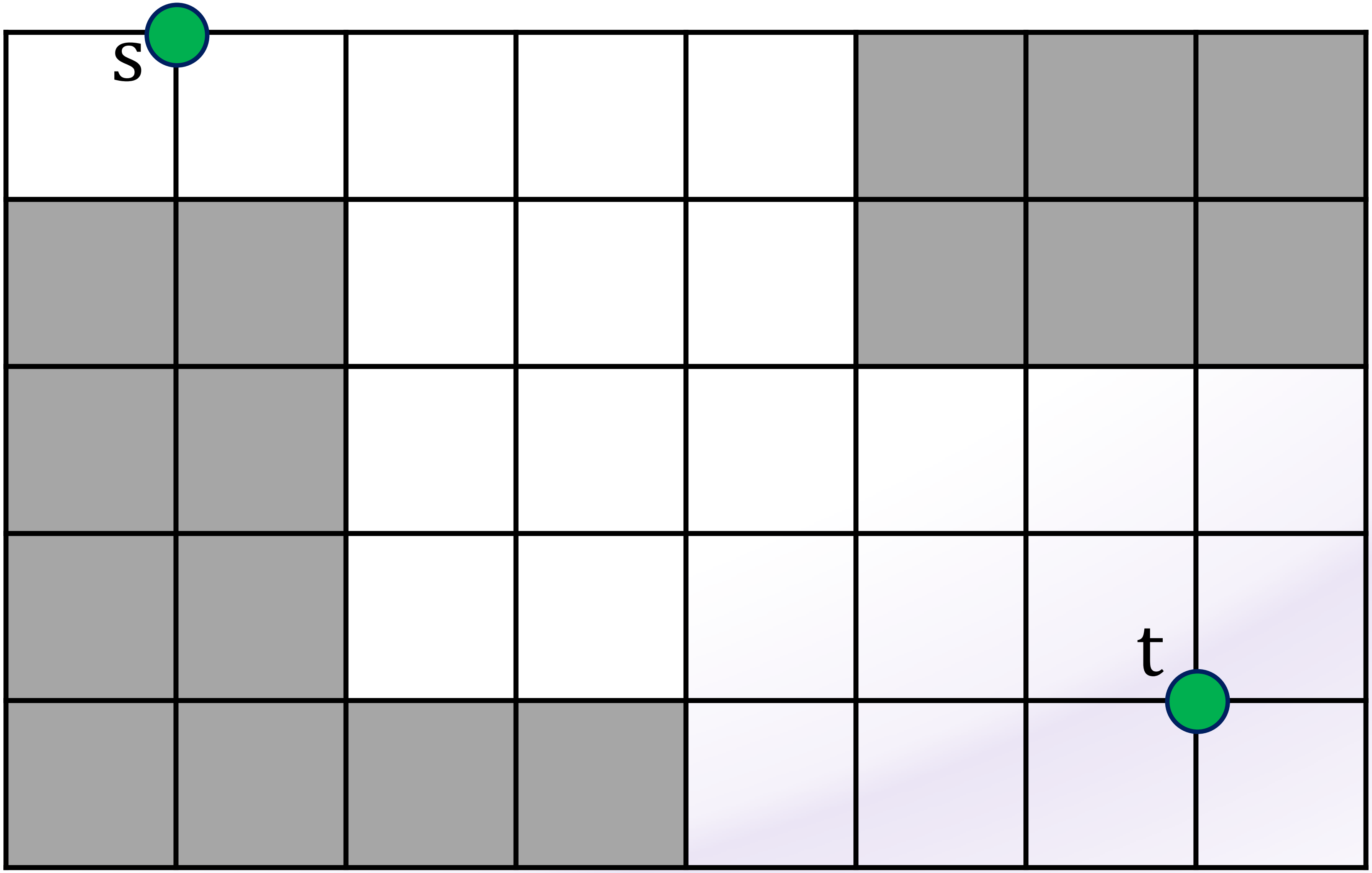
# Visibility Graph



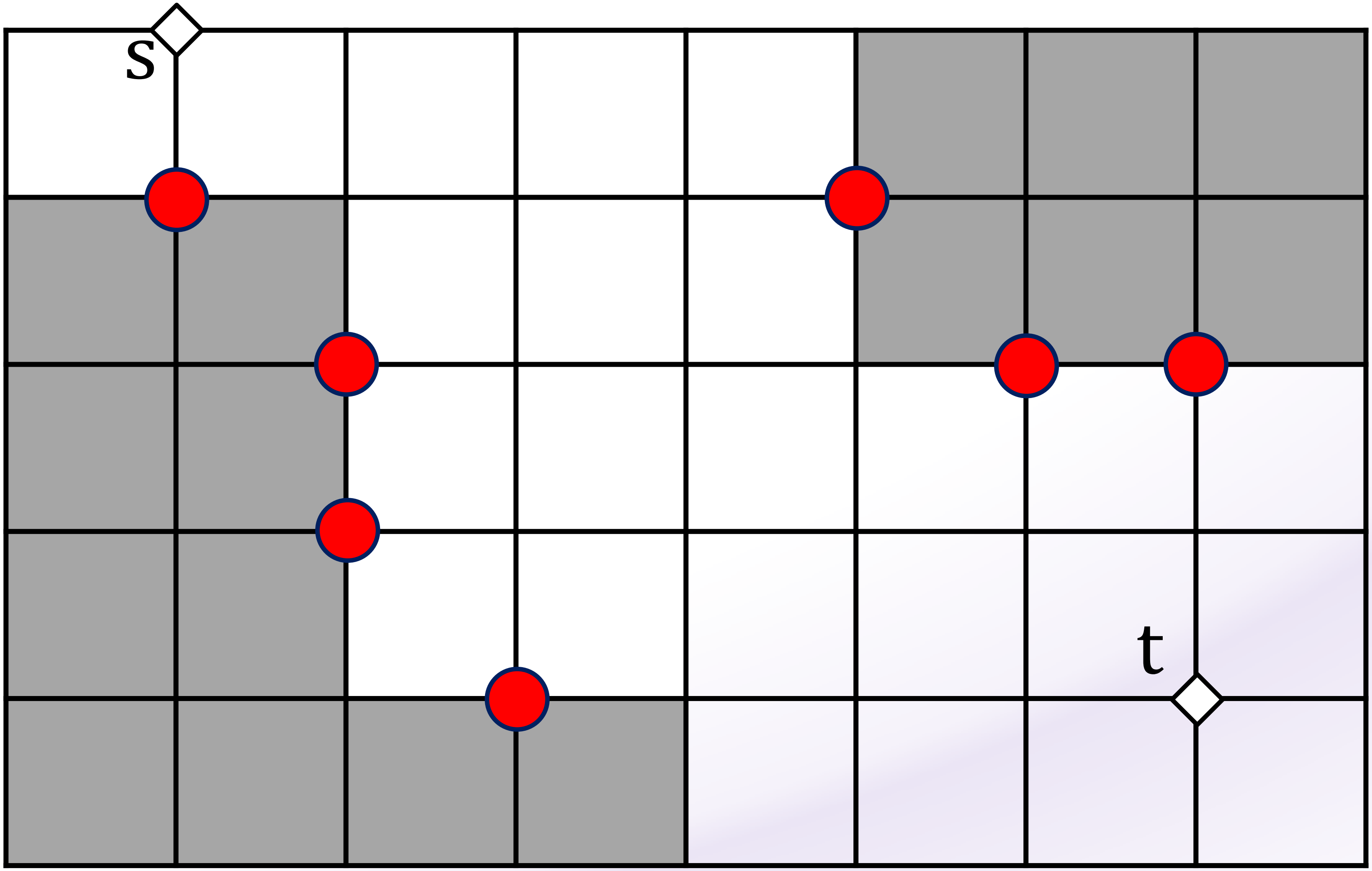
# Outer Corners of Walls: YES



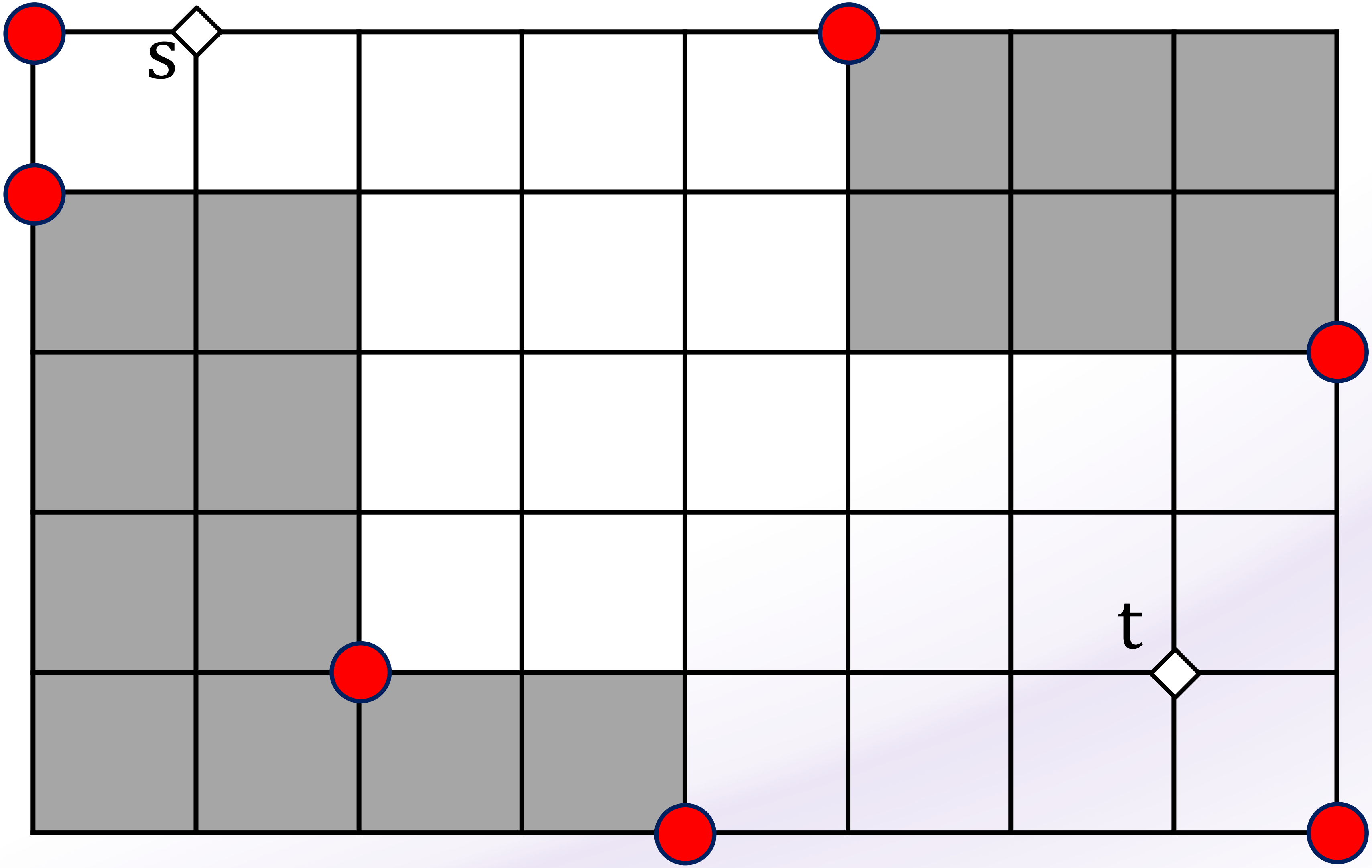
# Start and End Vertex: YES



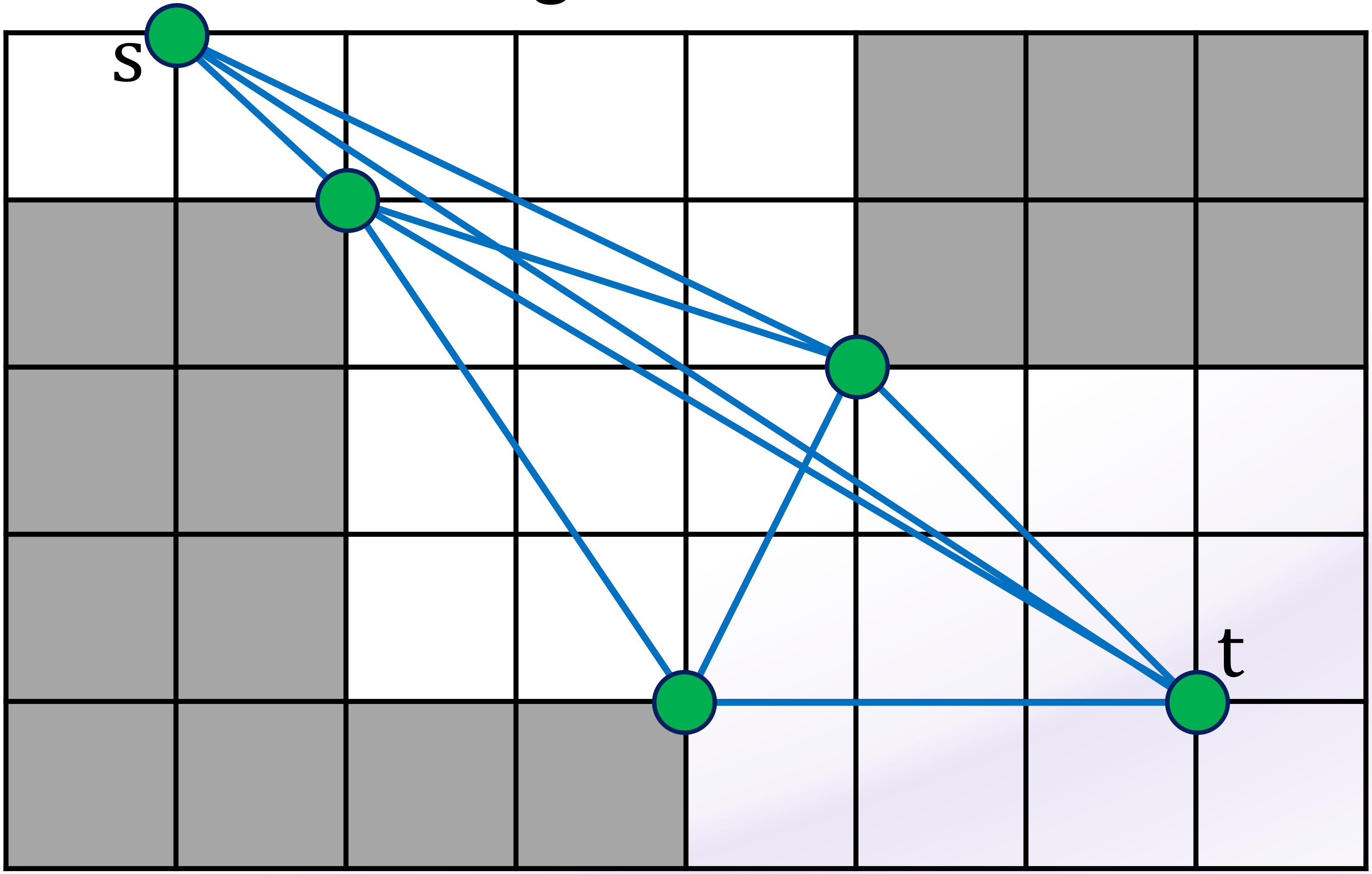
# Sides of Walls: NO



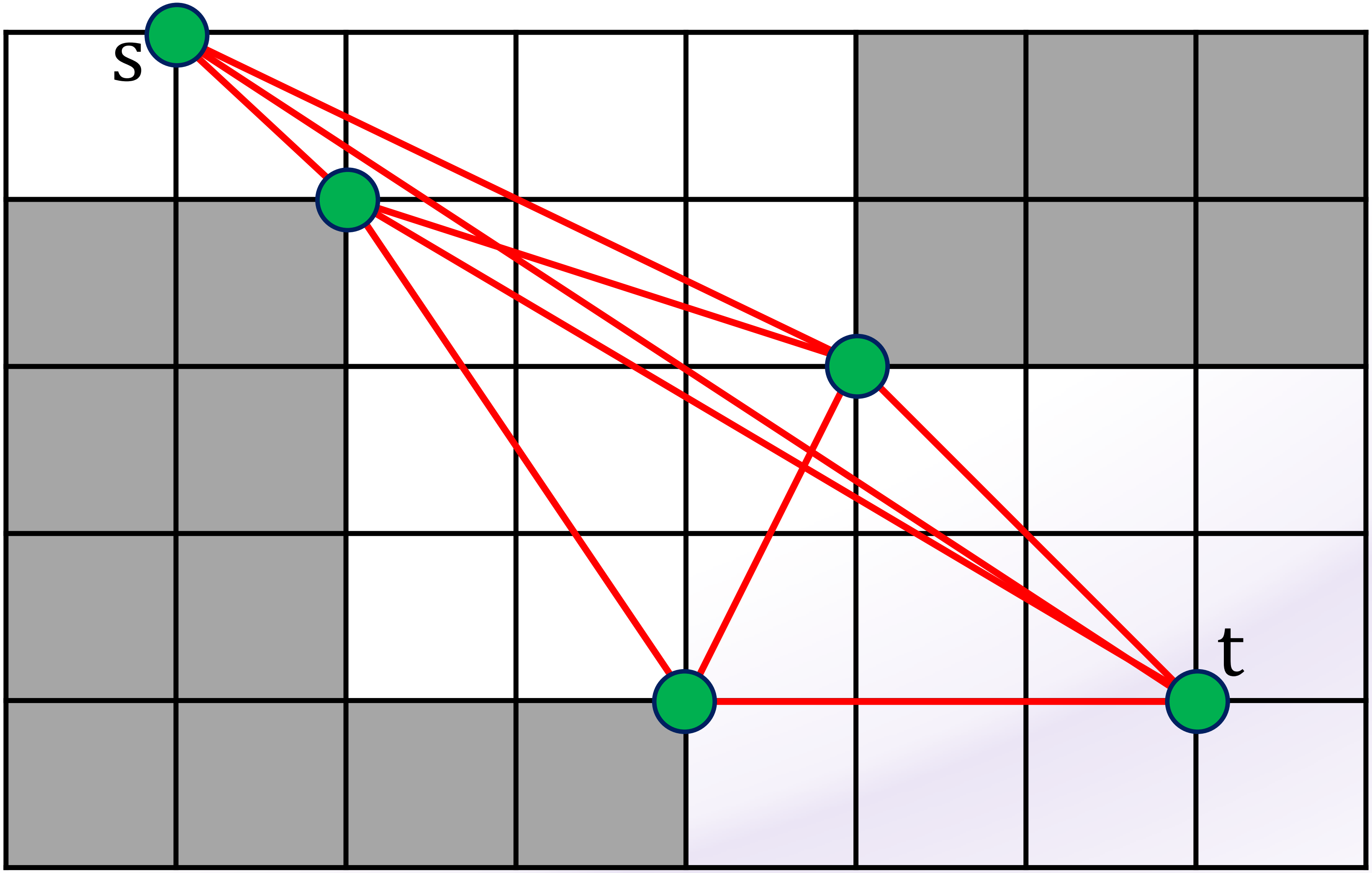
# Inner Corners of Walls: NO



# Connect all pairs of nodes with Line-of-Sight to each other.



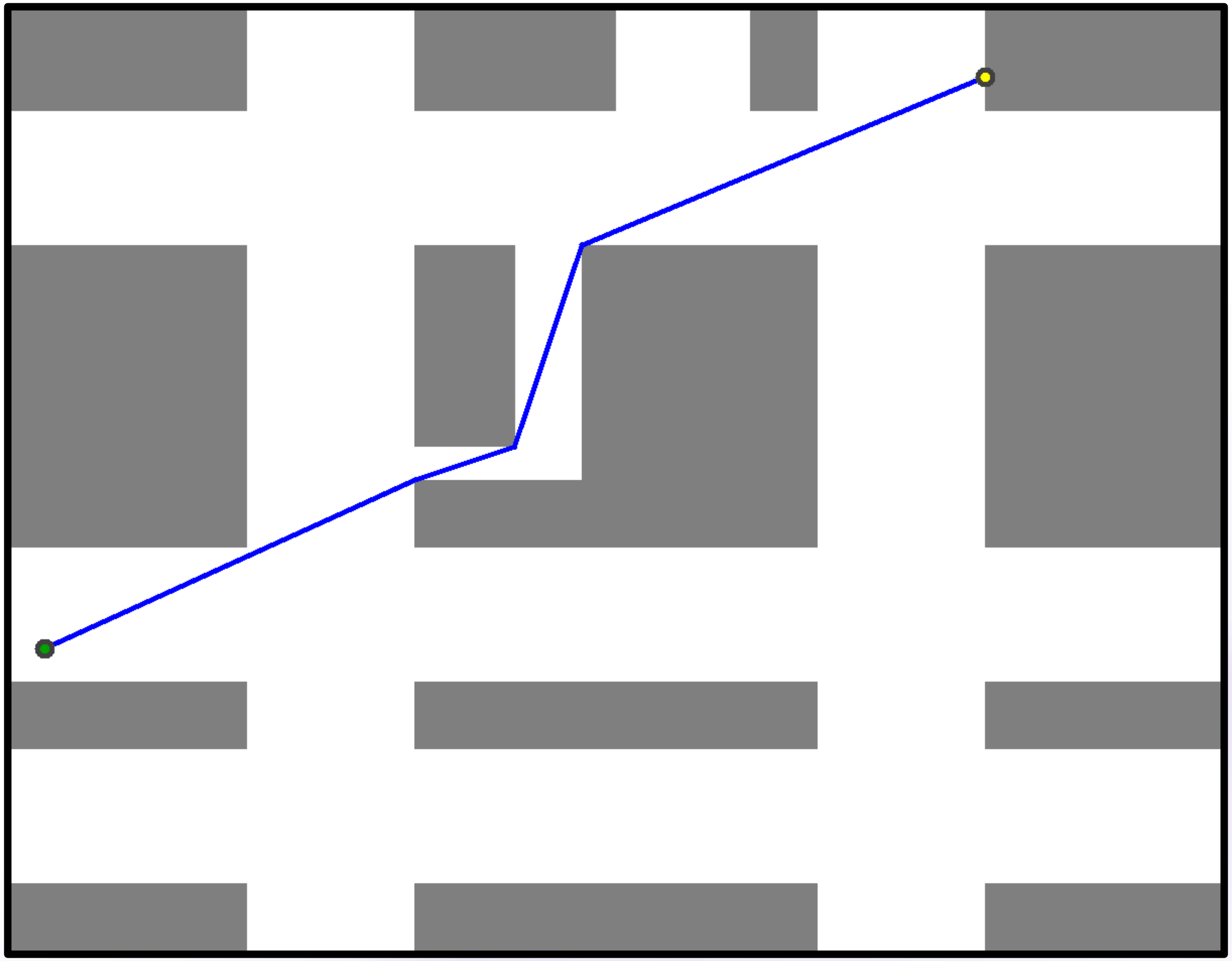
# The length of each line is its weight.



# Visibility Graph Algorithm Demo

VGraph_Hard
VGraph_LT

# Visibility Graphs are Optimal



# Further Reading

## Post-Smoothing:

**Not a pathfinding algorithm, but an extra post-processing step to "smoothen out" jagged paths.**

## Lazy Theta*:

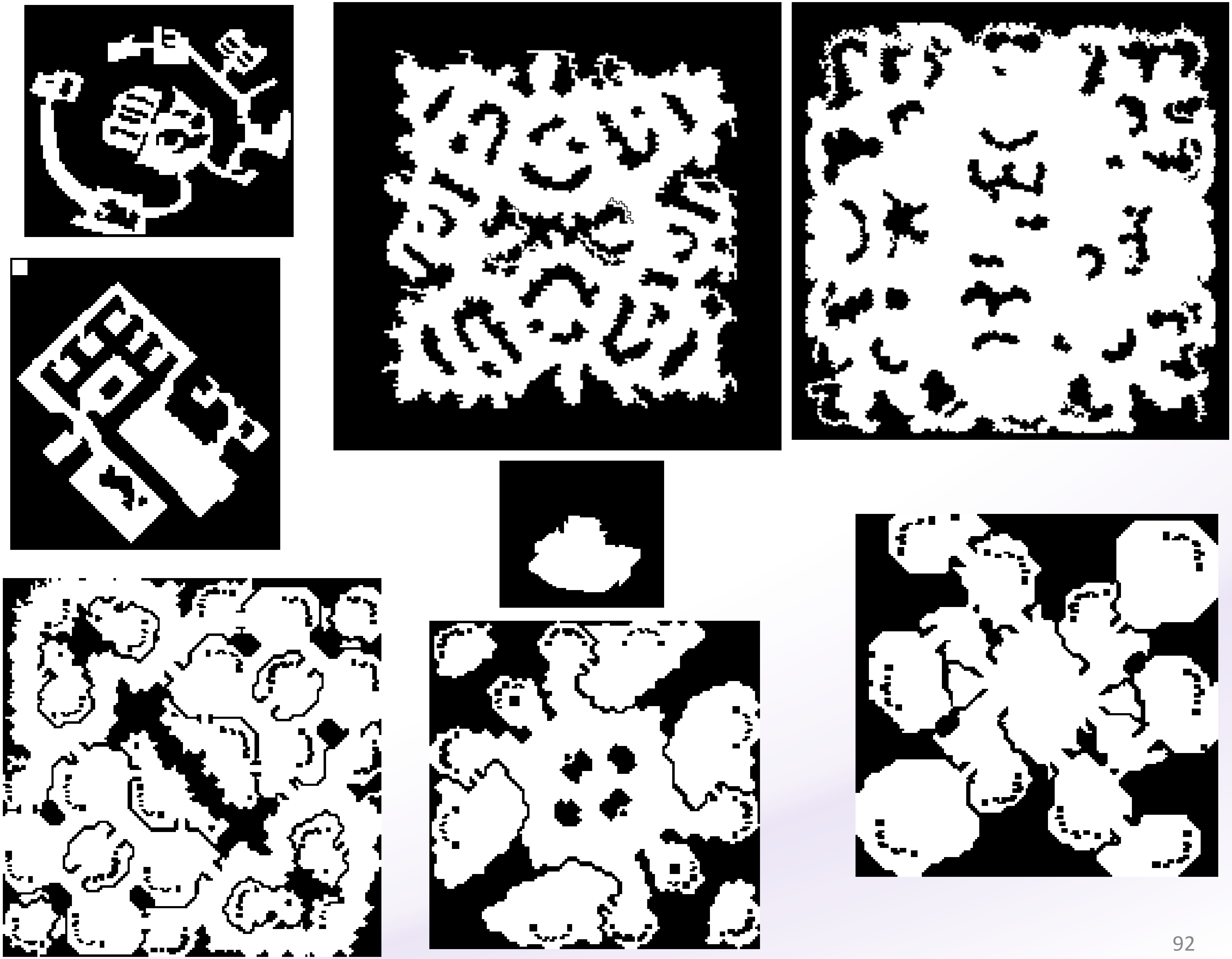
**If you find Theta* too slow due to the many Line-of-Sight checks,**
**Lazy Theta* runs faster, but gives slightly longer path lengths.**

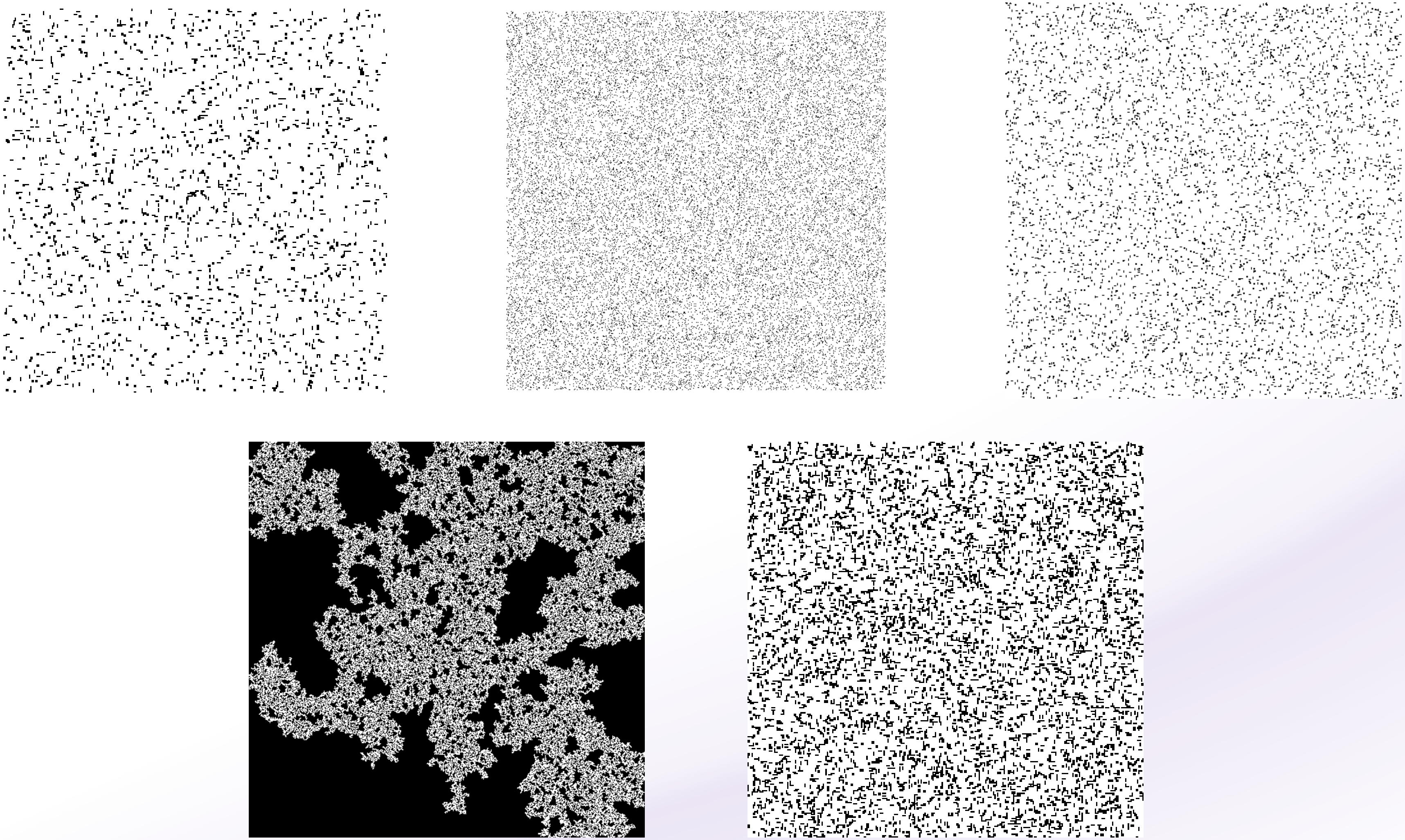
# Further Reading

## Jump Point Search:

**A very fast variation of A* for uniform grids. If speed is top priority, use this with post-smoothing.**

# Observations





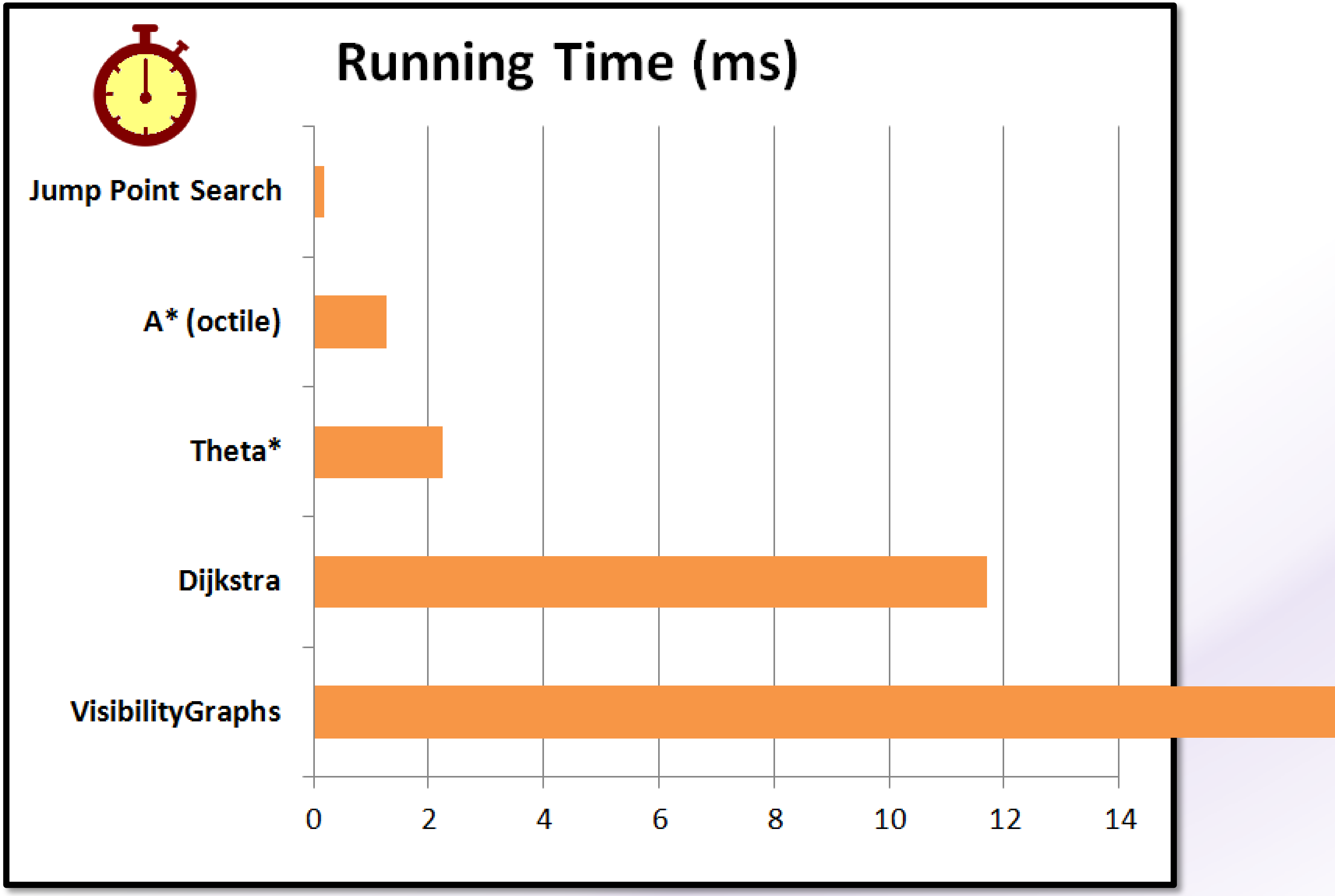
# Running Time (ms)

| Name | JPS | JPS PS | A*(OCT) PS | A*(OCT) | Lazy Theta* | A*(SLD) | Theta* | VisibilityGraphs Reuse | Dijkstra | VisibilityGraphs |
|---|---|---|---|---|---|---|---|---|---|---|
| sc2_steppesofwar | 0.067 | 0.057 | 0.788 | 0.736 | 1.178 | 1.032 | 1.829 | 0.723 | 4.166 | 65.560 |
| sc2_losttemple | 0.032 | 0.044 | 0.616 | 0.726 | 0.917 | 0.691 | 1.512 | 0.593 | 2.398 | 58.255 |
| sc2_extinction | 0.100 | 0.206 | 1.624 | 1.541 | 2.221 | 1.917 | 3.254 | 1.243 | 5.745 | 199.491 |
| baldursgate_AR0070SR | 0.016 | 0.031 | 0.510 | 0.499 | 0.653 | 0.513 | 1.172 | 0.440 | 0.893 | 18.666 |
| baldursgate_AR0705SR | 0.048 | 0.078 | 0.343 | 0.393 | 0.512 | 0.531 | 0.656 | 0.170 | 0.793 | 5.166 |
| baldursgate_AR0418SR | 0.000 | 0.032 | 0.032 | 0.030 | 0.000 | 0.016 | 0.031 | 0.046 | 0.311 | 0.031 |
| wc3_icecrown | 0.315 | 0.465 | 5.845 | 5.460 | 7.210 | 7.837 | 14.818 | 2.750 | 55.873 | 464.290 |
| wc3_dragonfire | 0.200 | 0.080 | 3.552 | 3.663 | 5.353 | 5.637 | 9.563 | 1.275 | 38.400 | 221.108 |
| Low_50x50_6% | 0.013 | 0.015 | 0.057 | 0.045 | 0.046 | 0.076 | 0.061 | 0.077 | 0.495 | 1.592 |
| Med_50x50_20% | 0.019 | 0.032 | 0.059 | 0.061 | 0.070 | 0.085 | 0.111 | 0.117 | 0.513 | 5.042 |
| High_50x50_40% | 0.035 | 0.035 | 0.151 | 0.118 | 0.189 | 0.142 | 0.245 | 0.105 | 0.376 | 3.698 |
| Low_300x300_6% | 0.818 | 0.720 | 2.428 | 2.262 | 1.499 | 4.053 | 1.967 | 9.803 | 27.706 | 2624.590 |
| Med_300x300_20% | 0.436 | 0.564 | 1.464 | 1.529 | 1.973 | 3.047 | 2.503 | 9.381 | 23.058 | 5573.942 |
| High_300x300_40% | 1.027 | 1.092 | 3.979 | 4.133 | 5.508 | 4.678 | 5.881 | 5.279 | 17.238 | 4524.737 |
| Low_500x500_6% | 1.593 | 1.640 | 4.938 | 5.340 | 2.427 | 8.247 | 3.709 | 27.724 | 79.467 | 19887.504 |
| Med_500x500_20% | 1.351 | 1.144 | 3.913 | 4.100 | 6.067 | 9.273 | 6.911 | 24.322 | 74.567 | DNF |
| High_500x500_40% | 3.884 | 3.813 | 14.936 | 14.978 | 20.080 | 18.489 | 22.118 | 17.569 | 69.876 | DNF |
| obst10_random512 | 3.650 | 3.733 | 5.728 | 6.033 | 2.606 | 9.256 | 2.933 | 118.889 | 76.483 | DNF |
| obst40_random512 | 1.178 | 1.251 | 4.100 | 3.996 | 5.987 | 9.116 | 7.013 | 23.938 | 74.327 | DNF |
| Mean | 0.778 | 0.791 | 2.898 | 2.929 | 3.394 | 4.455 | 4.541 | 12.865 | 29.089 | DNF |

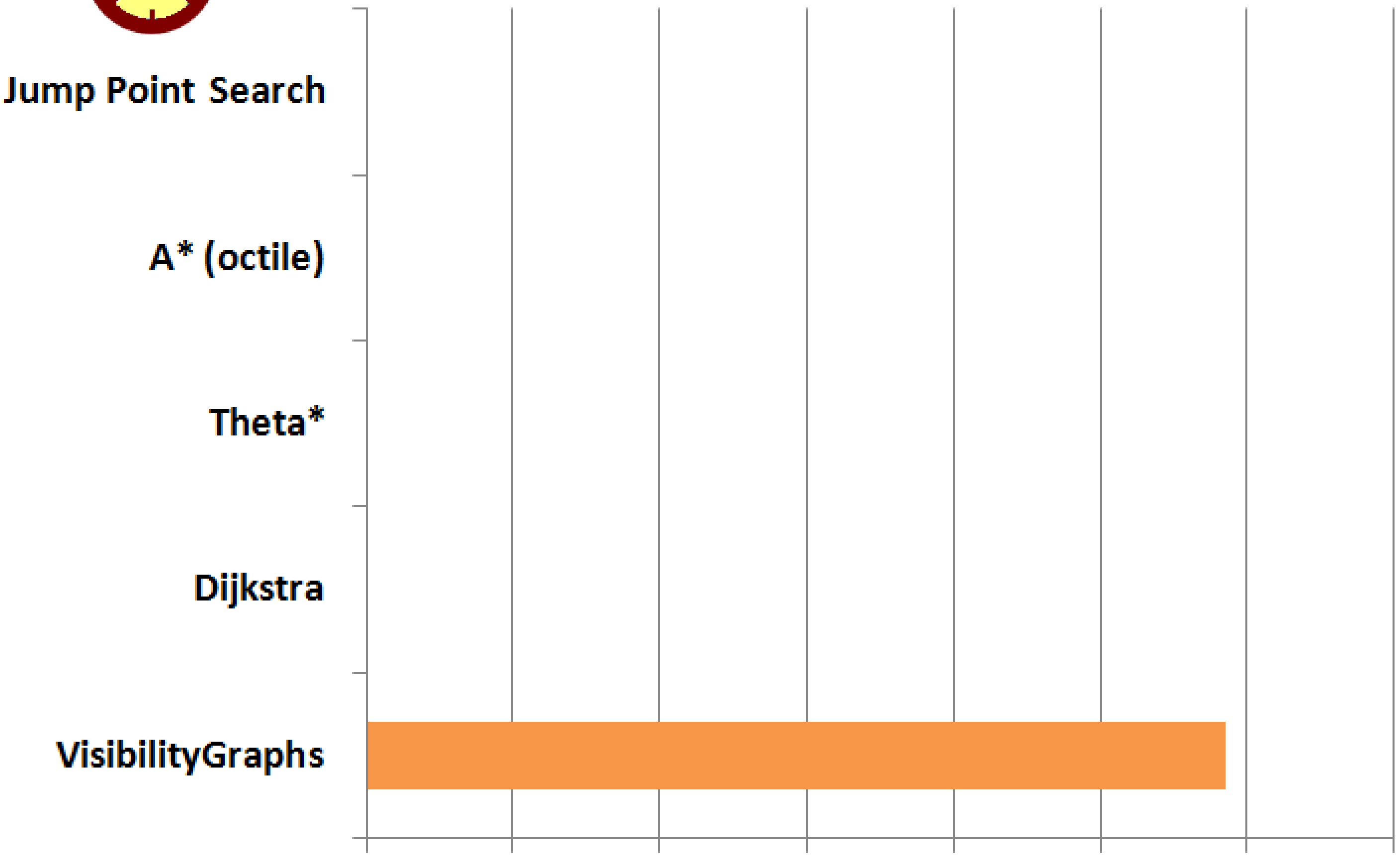
# Path Lengths (ratio to optimal)

| Name | VisibilityGraphs Reuse | VisibilityGraphs | Theta* | Lazy Theta* | A*(OCT) PS | JPS PS | JPS | Dijkstra | A*(SLD) | A*(OCT) |
|---|---|---|---|---|---|---|---|---|---|---|
| sc2_steppesofwar | 1 | 1 | 1.0003 | 1.0007 | 1.0034 | 1.0114 | 1.0507 | 1.0507 | 1.0507 | 1.0507 |
| sc2_losttemple | 1 | 1 | 1.0002 | 1.0020 | 1.0040 | 1.0145 | 1.0464 | 1.0464 | 1.0464 | 1.0464 |
| sc2_extinction | 1 | 1 | 1.0005 | 1.0016 | 1.0033 | 1.0176 | 1.0512 | 1.0512 | 1.0512 | 1.0512 |
| baldursgate_AR0070SR | 1 | 1 | 1.0004 | 1.0013 | 1.0075 | 1.0132 | 1.0460 | 1.0460 | 1.0460 | 1.0460 |
| baldursgate_AR0705SR | 1 | 1 | 1.0004 | 1.0024 | 1.0030 | 1.0189 | 1.0467 | 1.0467 | 1.0467 | 1.0467 |
| baldursgate_AR0418SR | 1 | 1 | 1 | 1 | 1 | 1 | 1.0493 | 1.0493 | 1.0493 | 1.0493 |
| wc3_icecrown | 1 | 1 | 1.0003 | 1.0007 | 1.0039 | 1.0154 | 1.0539 | 1.0539 | 1.0539 | 1.0539 |
| wc3_dragonfire | 1 | 1 | 1.0010 | 1.0021 | 1.0042 | 1.0184 | 1.0509 | 1.0509 | 1.0509 | 1.0509 |
| Low_50x50_6% | 1 | 1 | 1.0004 | 1.0004 | 1.0053 | 1.0097 | 1.0479 | 1.0479 | 1.0479 | 1.0479 |
| Med_50x50_20% | 1 | 1 | 1.0007 | 1.0015 | 1.0094 | 1.0117 | 1.0471 | 1.0471 | 1.0471 | 1.0471 |
| High_50x50_40% | 1 | 1 | 1.0008 | 1.0028 | 1.0052 | 1.0096 | 1.0364 | 1.0364 | 1.0364 | 1.0364 |
| Low_300x300_6% | 1 | 1 | 1.0009 | 1.0010 | 1.0243 | 1.0280 | 1.0569 | 1.0569 | 1.0569 | 1.0569 |
| Med_300x300_20% | 1 | 1 | 1.0019 | 1.0029 | 1.0158 | 1.0219 | 1.0500 | 1.0500 | 1.0500 | 1.0500 |
| High_300x300_40% | 1 | 1 | 1.0013 | 1.0046 | 1.0083 | 1.0117 | 1.0450 | 1.0450 | 1.0450 | 1.0450 |
| Low_500x500_6% | 1 | 1 | 1.0009 | 1.0009 | 1.0262 | 1.0320 | 1.0552 | 1.0552 | 1.0552 | 1.0552 |
| Med_500x500_20% | 1 | DNF | 1.0021 | 1.0037 | 1.0180 | 1.0208 | 1.0475 | 1.0475 | 1.0475 | 1.0475 |
| High_500x500_40% | 1 | DNF | 1.0016 | 1.0052 | 1.0099 | 1.0143 | 1.0449 | 1.0449 | 1.0449 | 1.0449 |
| obst10_random512 | 1 | DNF | 1.0017 | 1.0015 | 1.0399 | 1.0417 | 1.0573 | 1.0573 | 1.0573 | 1.0573 |
| obst40_random512 | 1 | DNF | 1.0021 | 1.0037 | 1.0180 | 1.0208 | 1.0475 | 1.0475 | 1.0475 | 1.0475 |
| Mean | 1.000 | 1.000 | 1.001 | 1.002 | 1.011 | 1.017 | 1.049 | 1.049 | 1.049 | 1.049 |

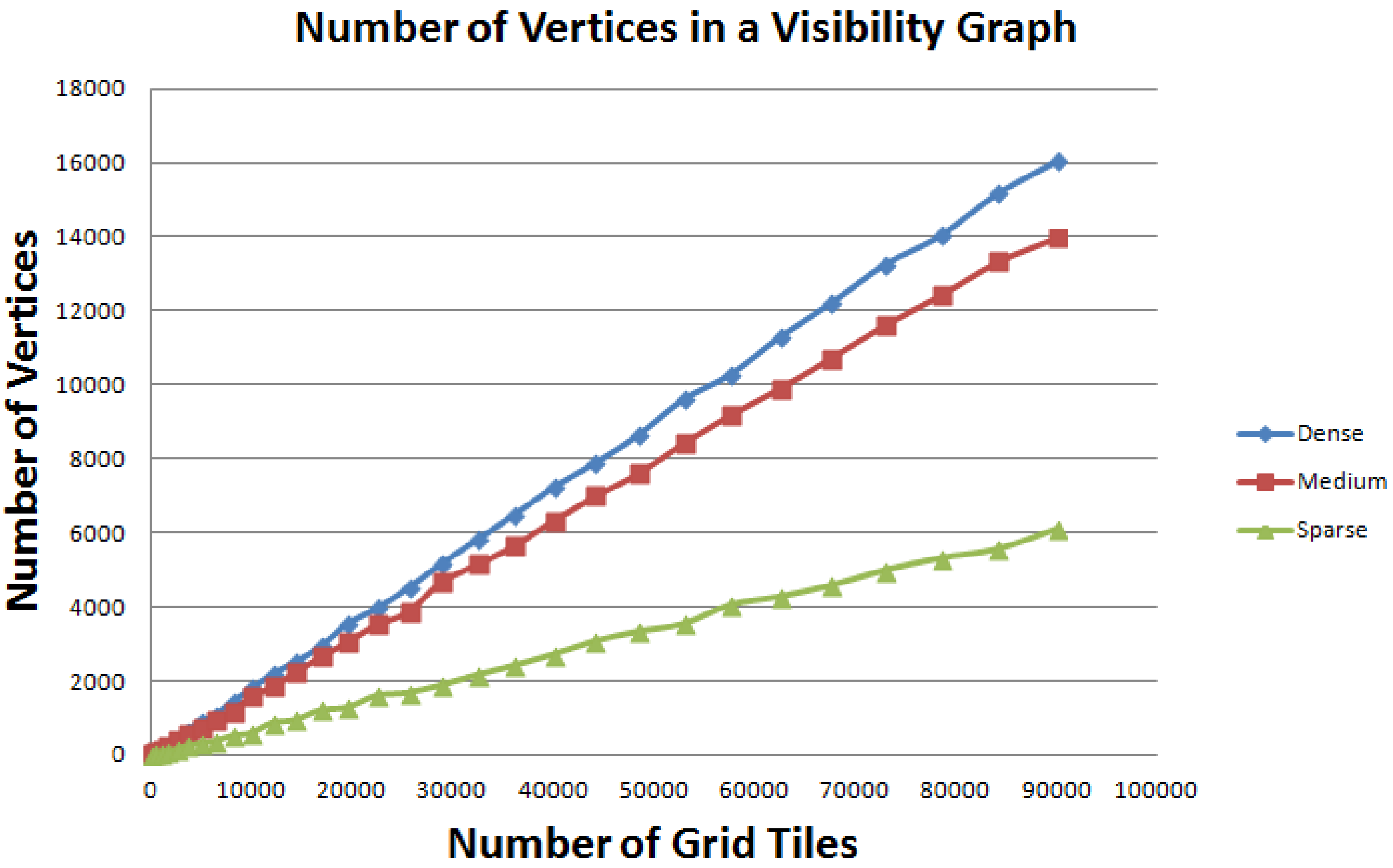
# Running Time Comparison


Running Time (ms)
Jump Point Search
A* (octile)
Theta*
Dijkstra
VisibilityGraphs
0
2
4
6
8
10
12
14

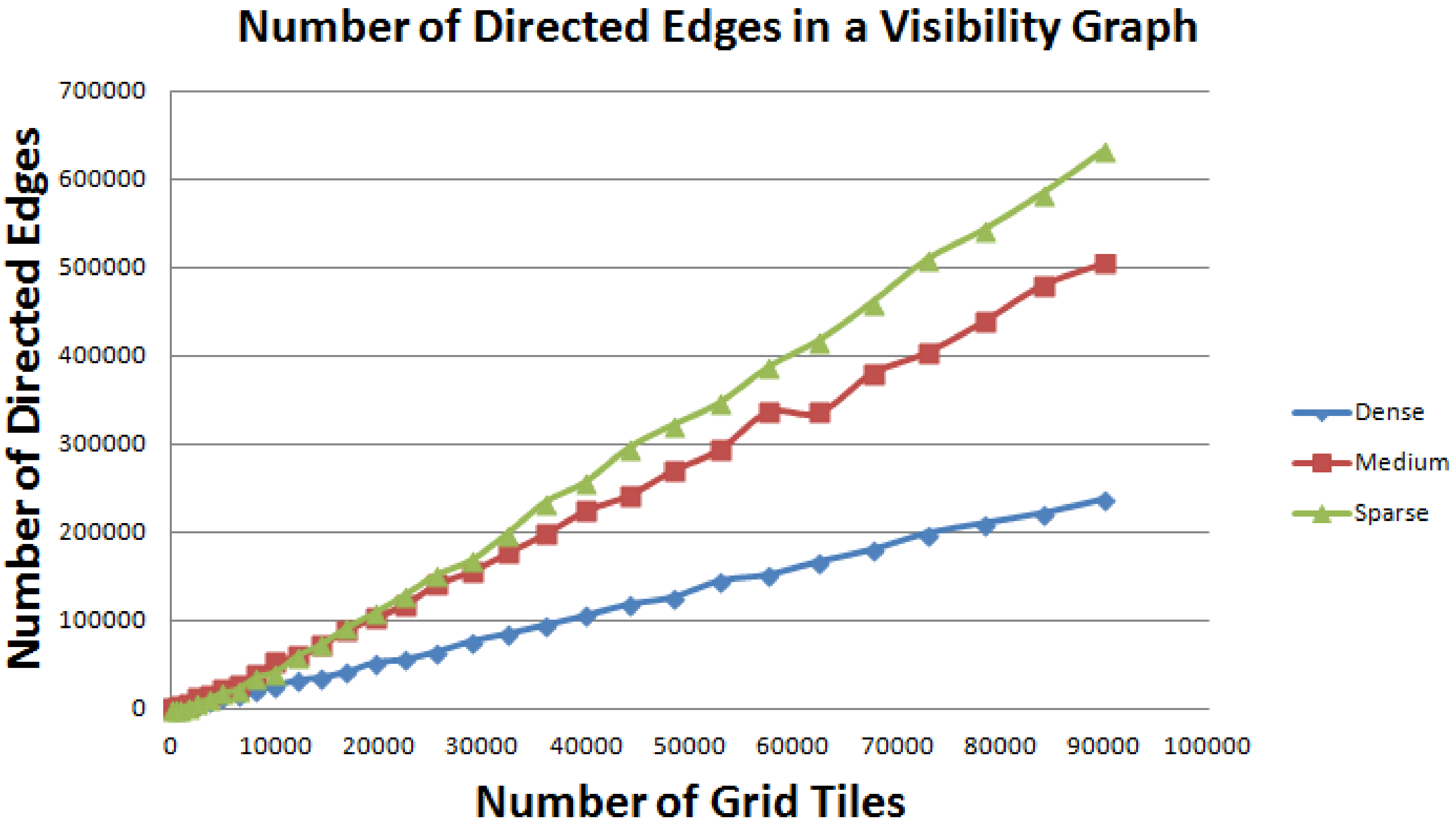
# Running Time



# Observation: Visibility Graphs are Optimal, But Extremely Slow

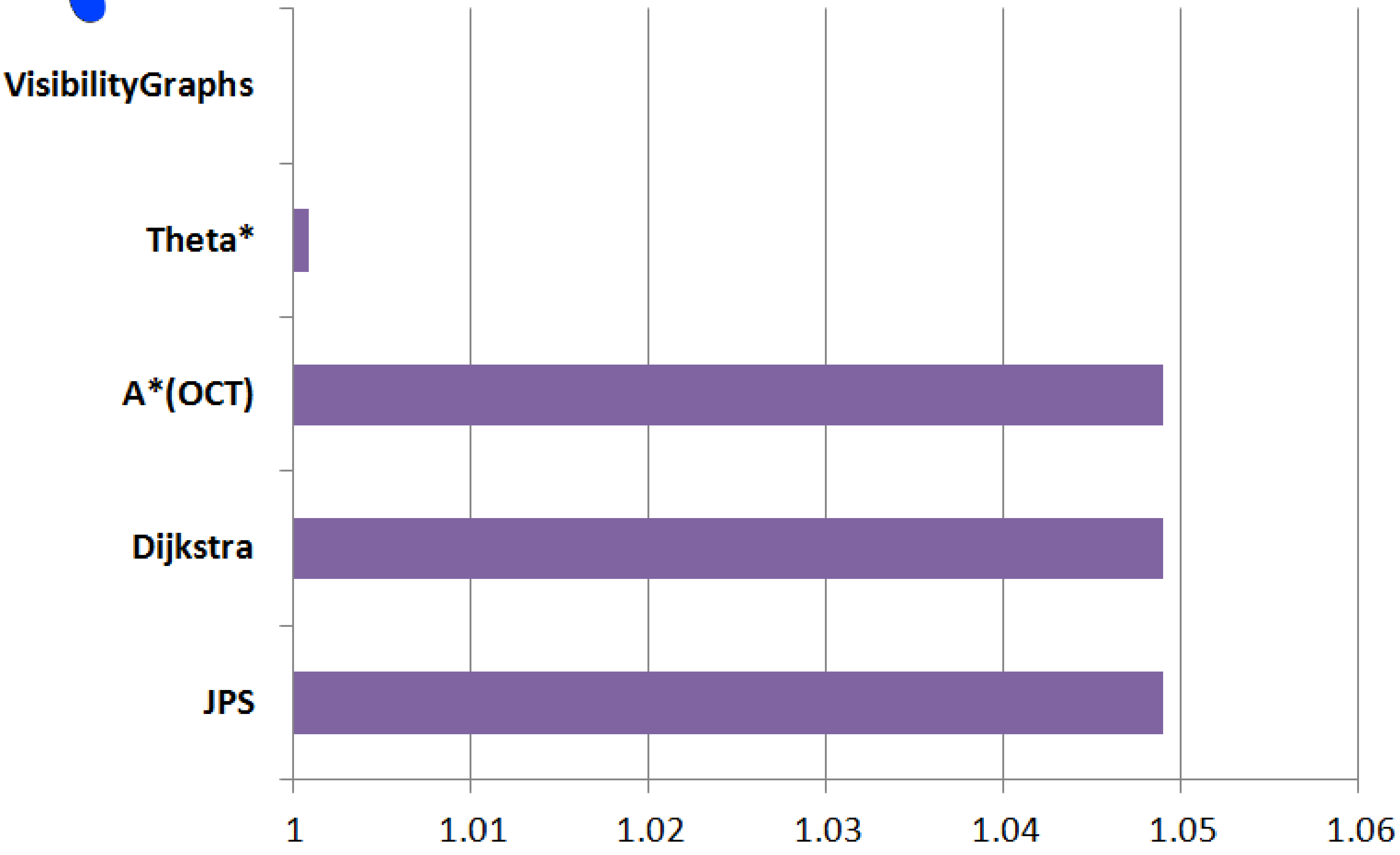


# Observation: Visibility Graphs are Optimal, But Extremely Slow

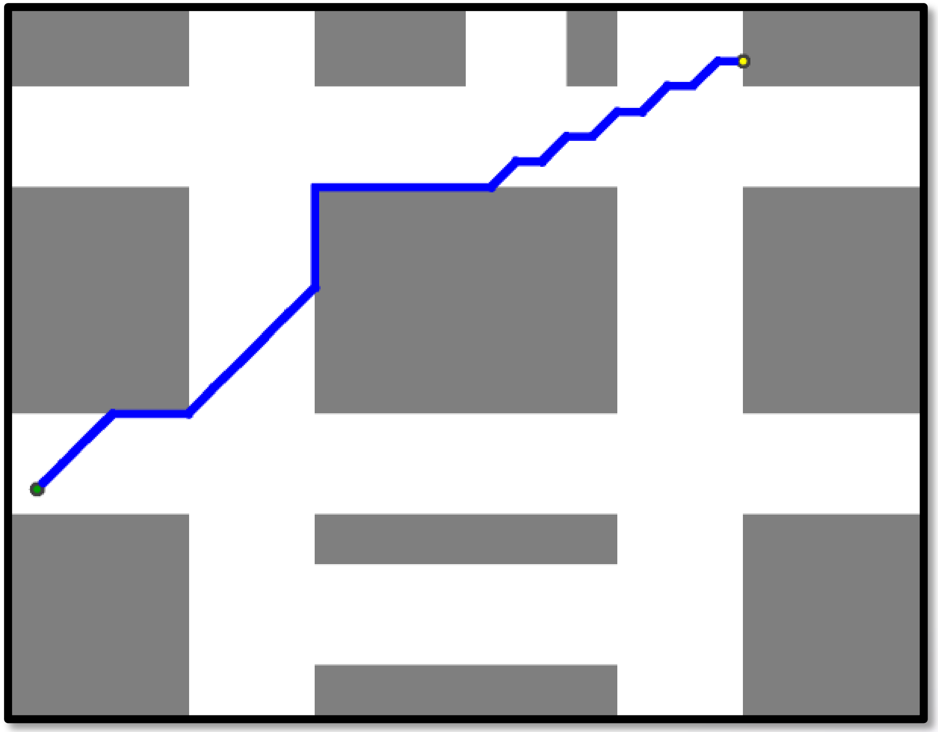


# Path Length Comparison

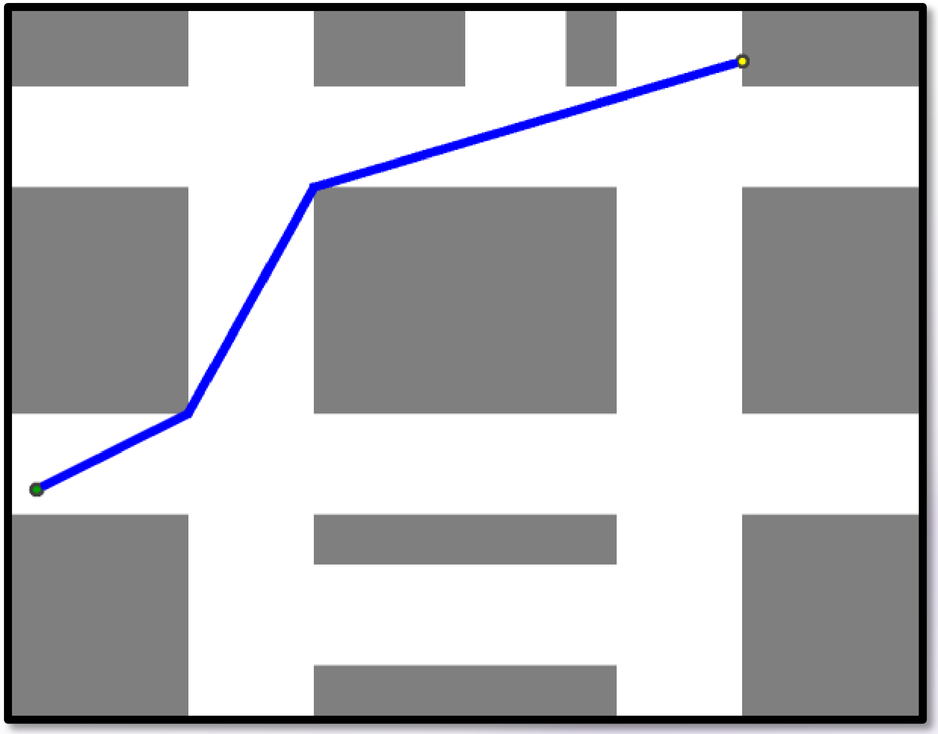
# Path Length (ratio to optimal)

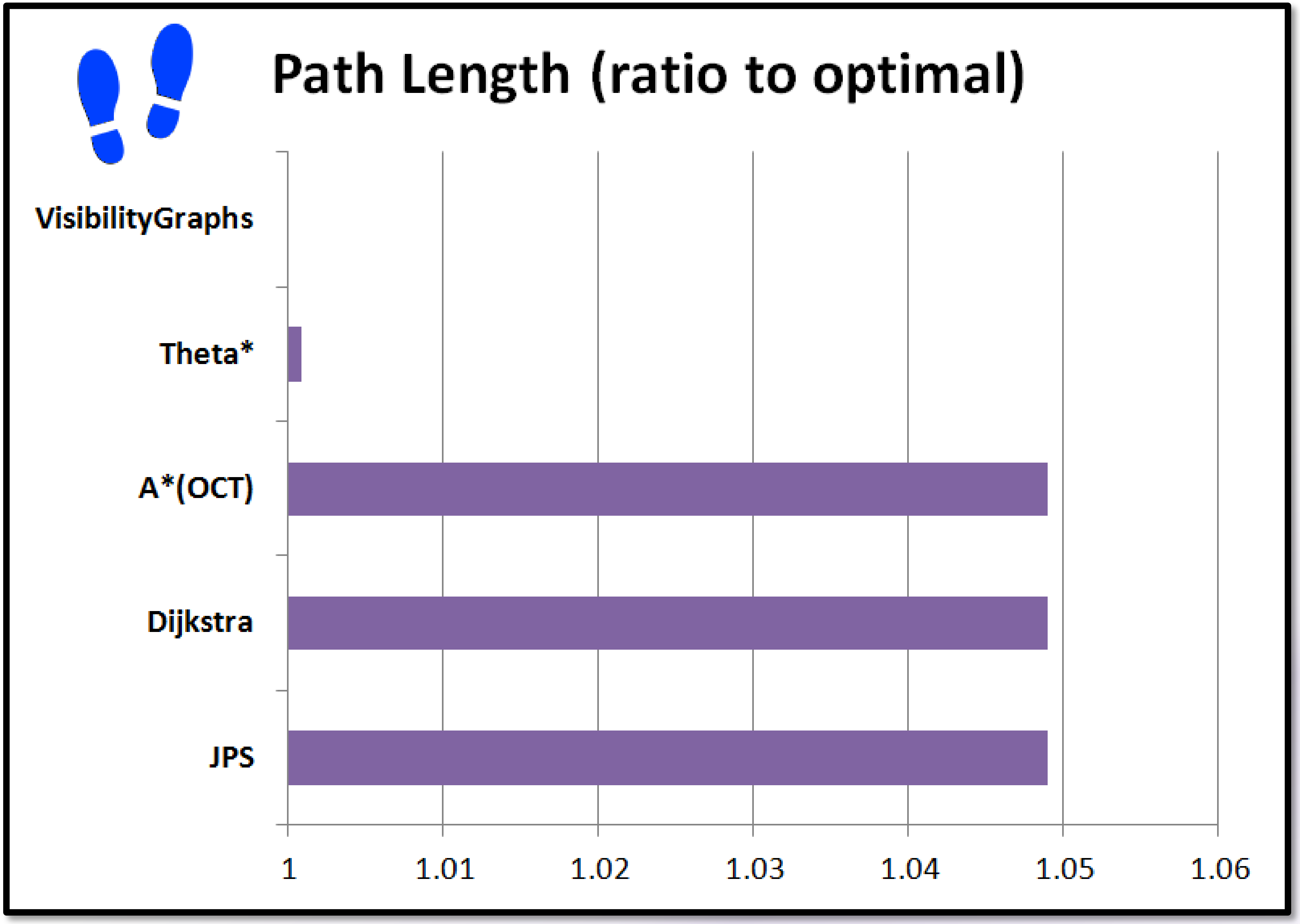


## A*,
## Dijkstra,
## Jump Point Search

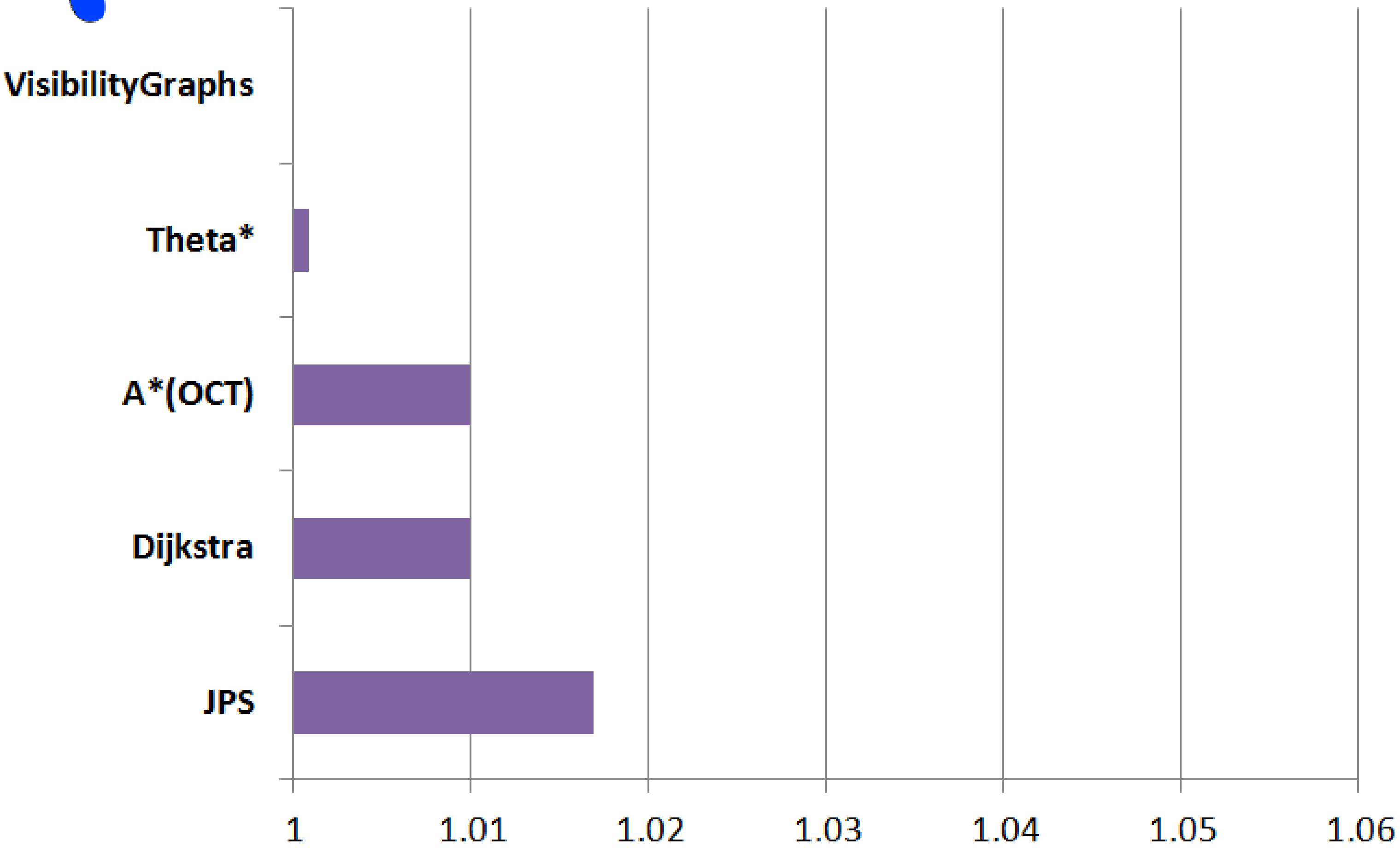


## Theta*,
## VisibilityGraphs



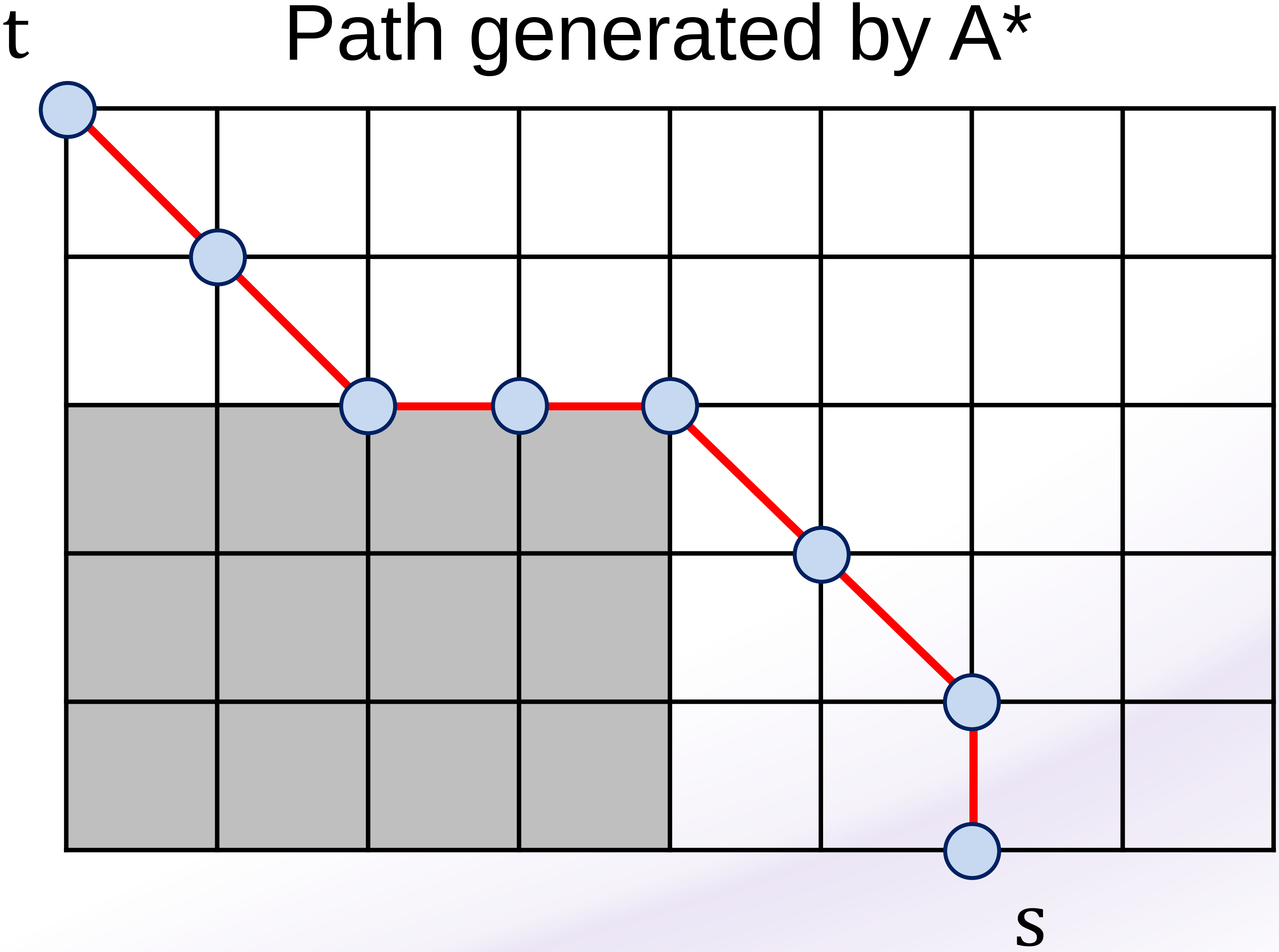

Path Length (ratio to optimal)
VisibilityGraphs
Theta*
A*(OCT)
Dijkstra
JPS
1
1.01
1.02
1.03
1.04
1.05
1.06

# Path Length - after smoothing

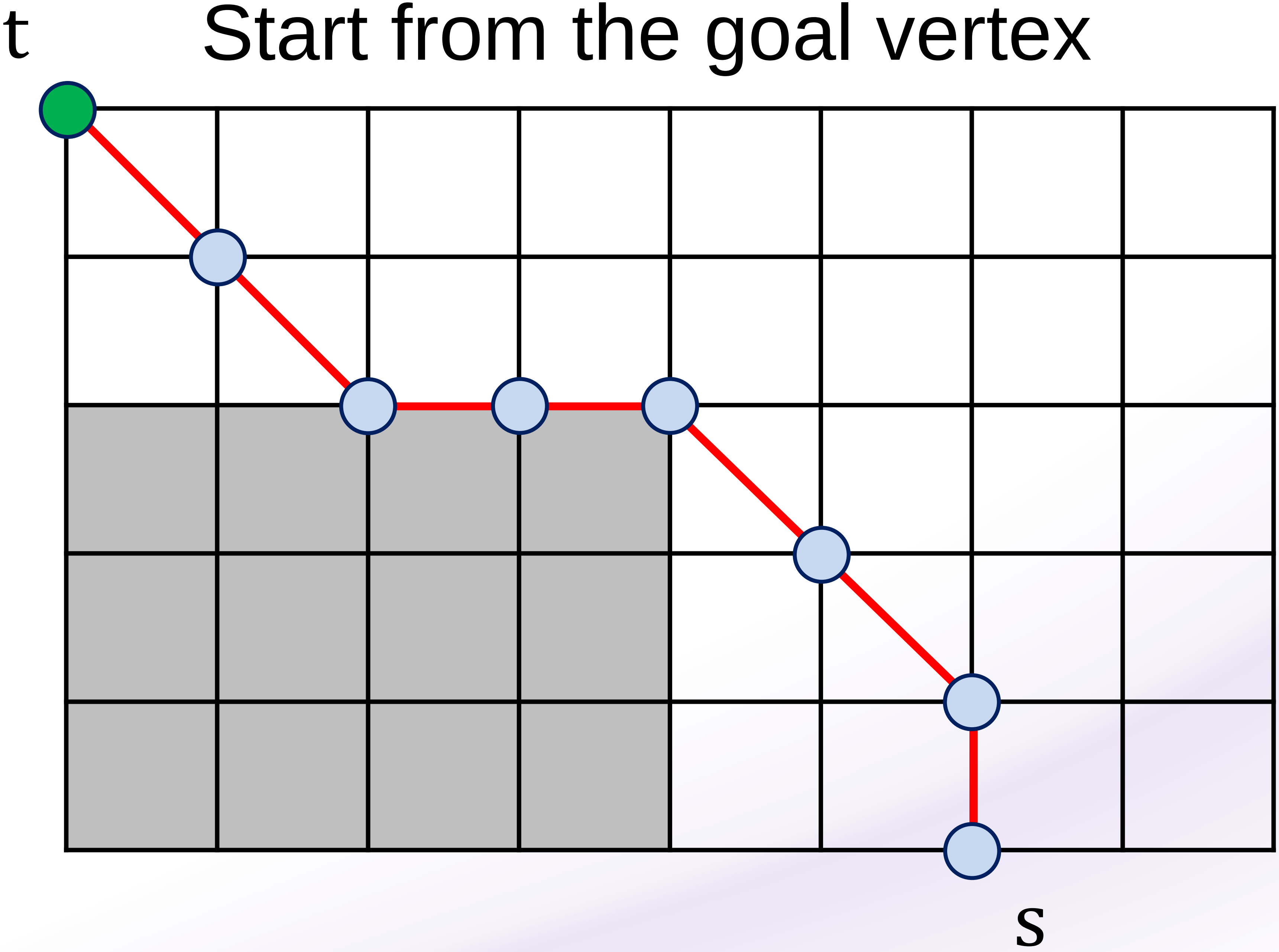


# A* PS
# Post-Smoothing Step

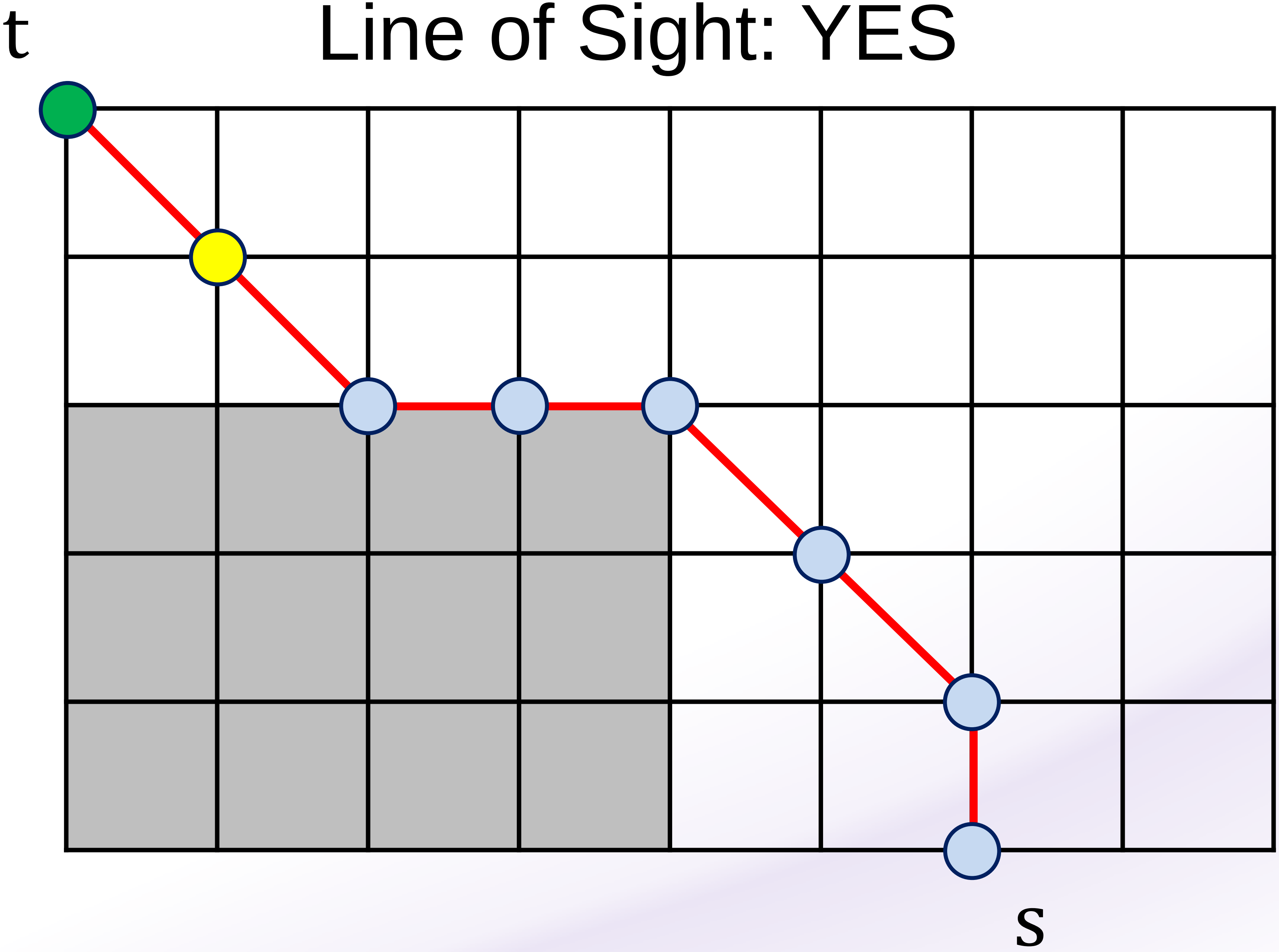
# Path generated by A*



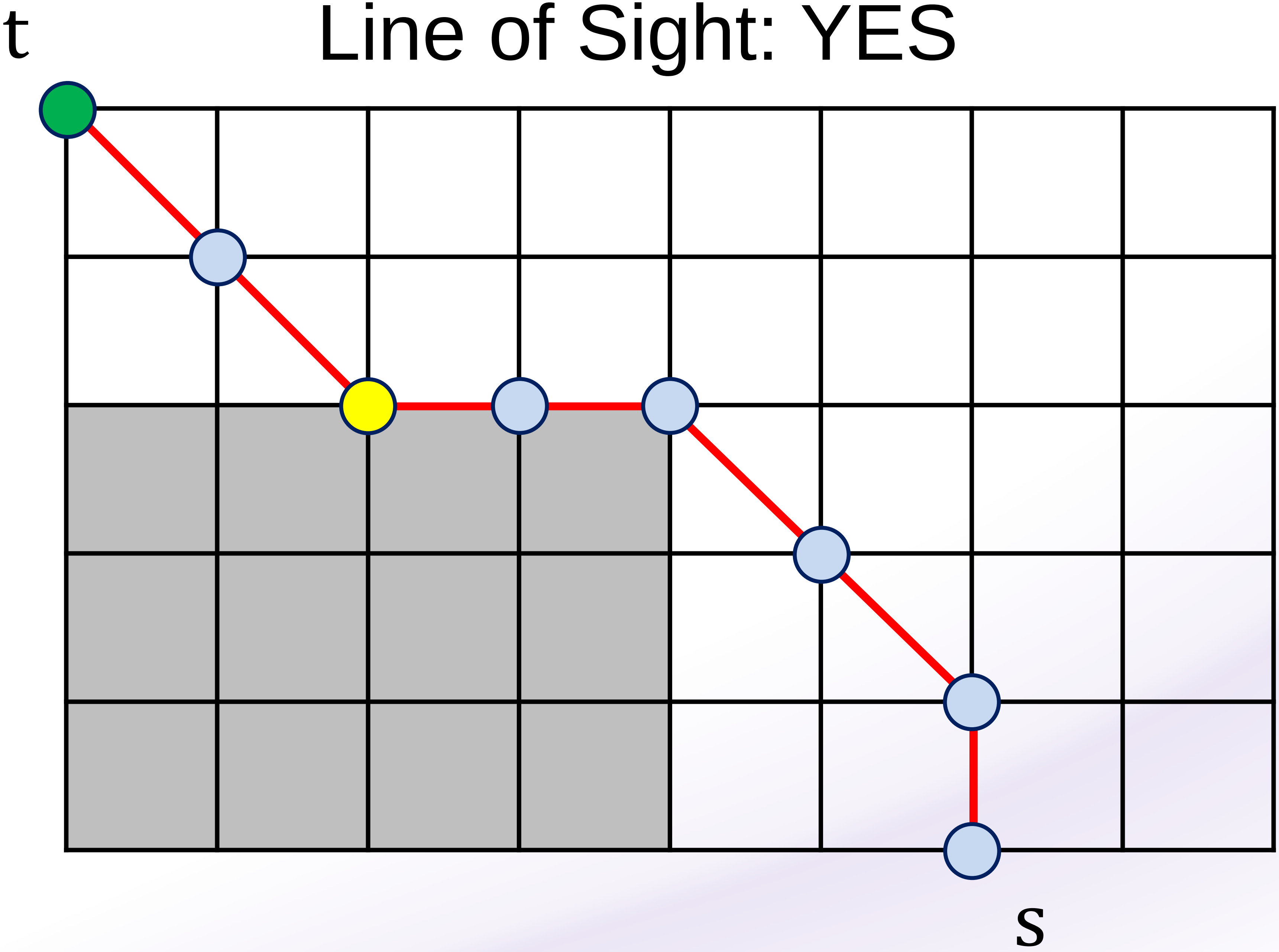
# Start from the goal vertex



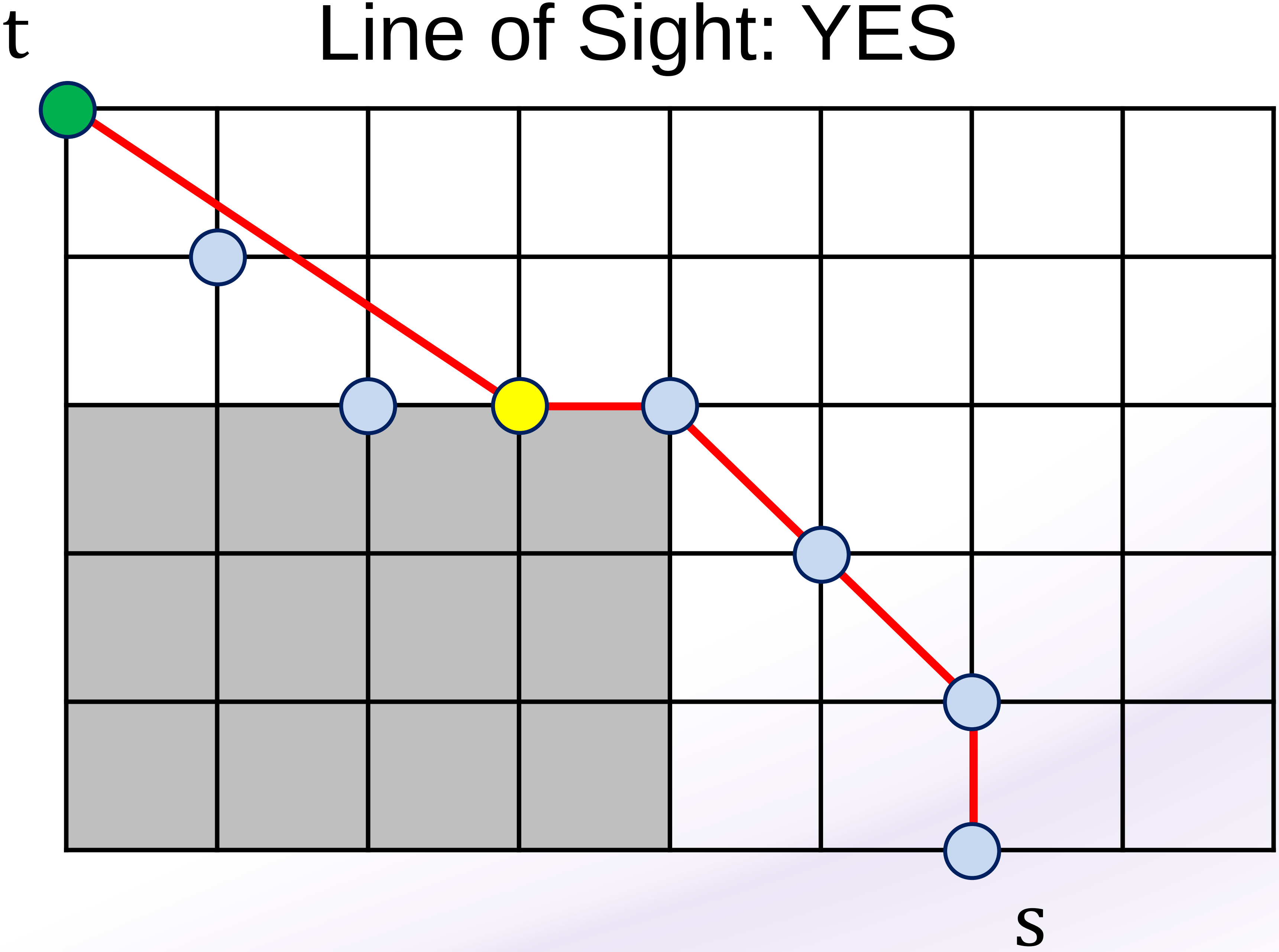
# Line of Sight: YES



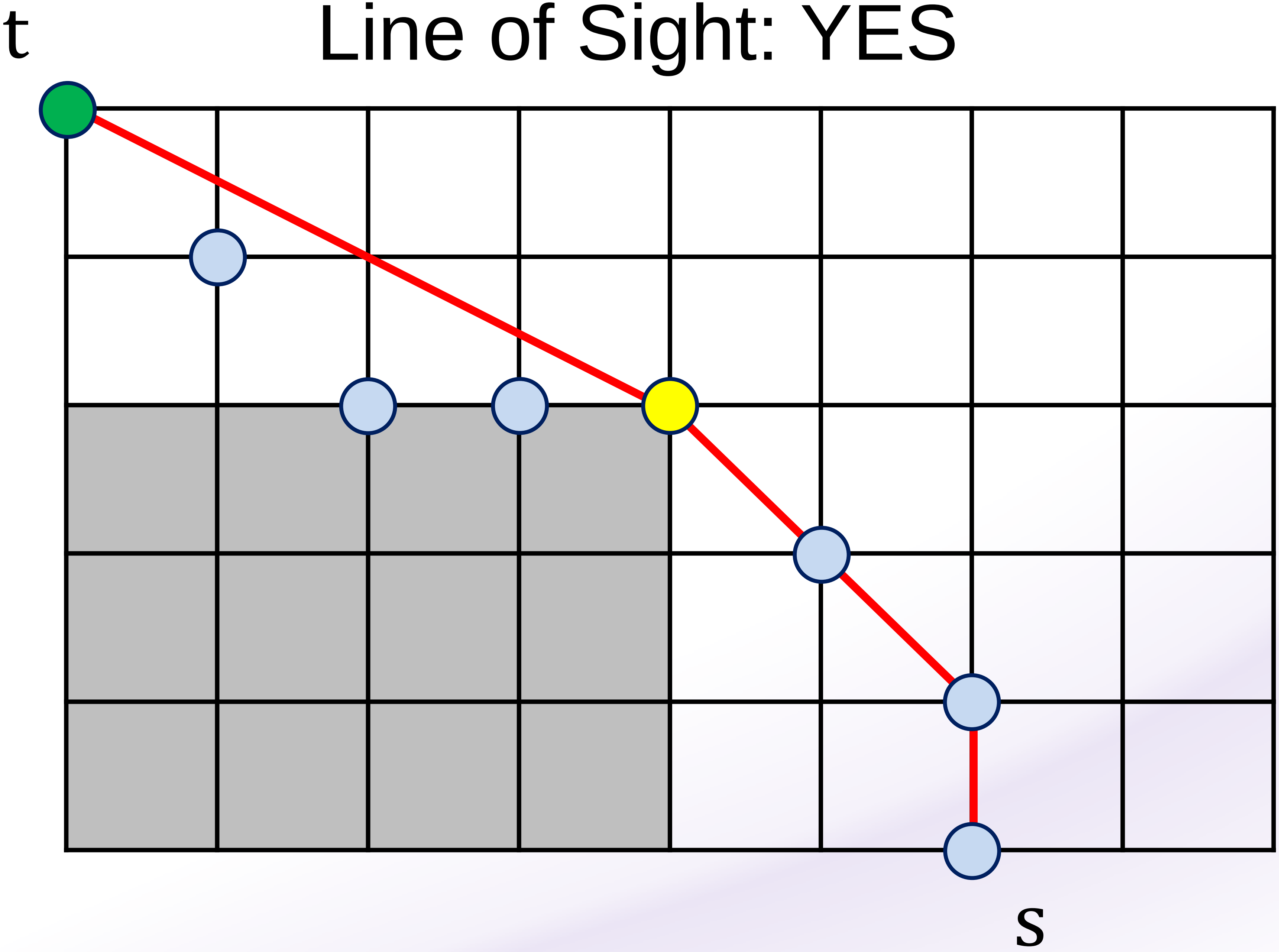
# Line of Sight: YES



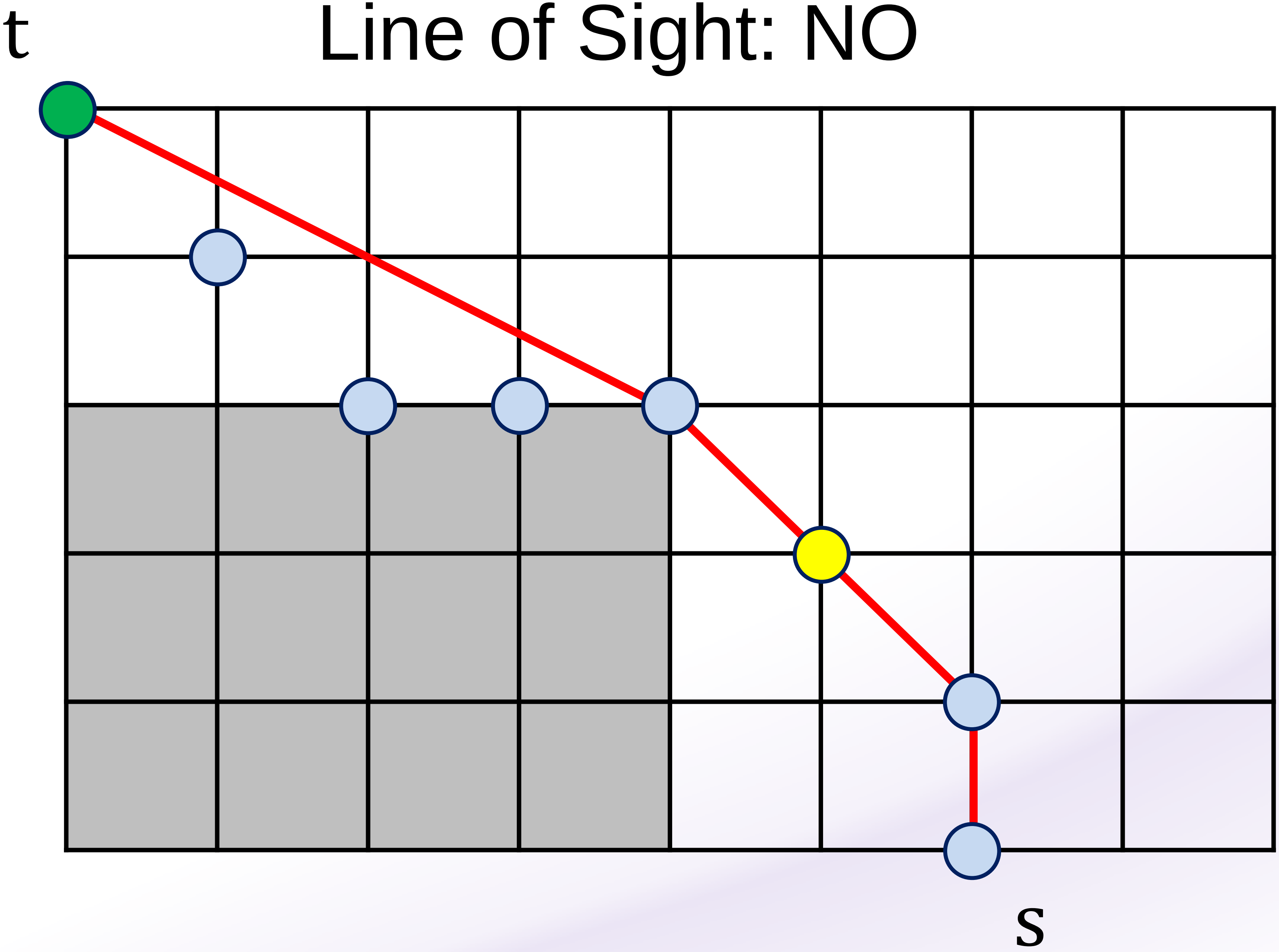
# Line of Sight: YES



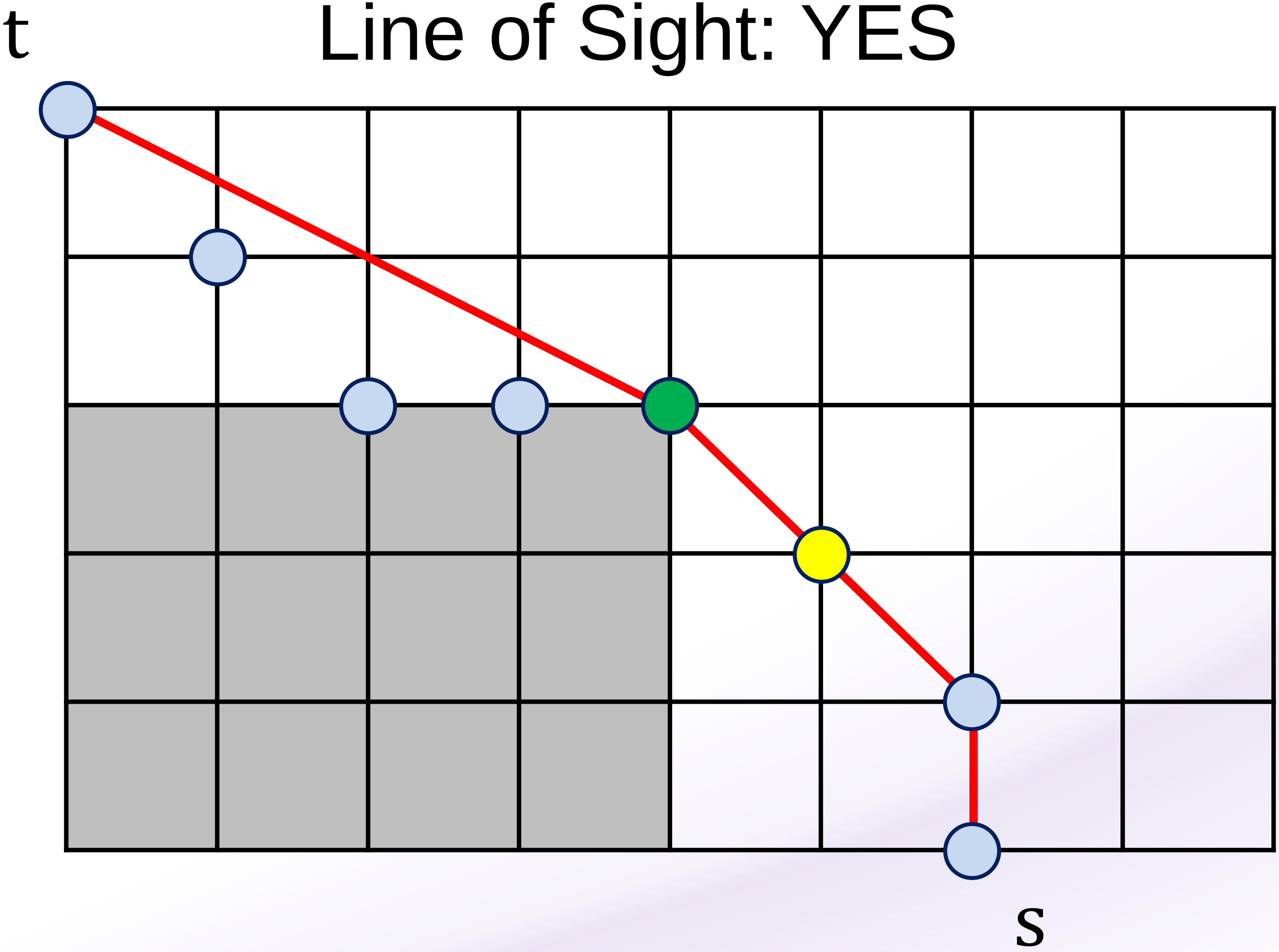
# Line of Sight: YES



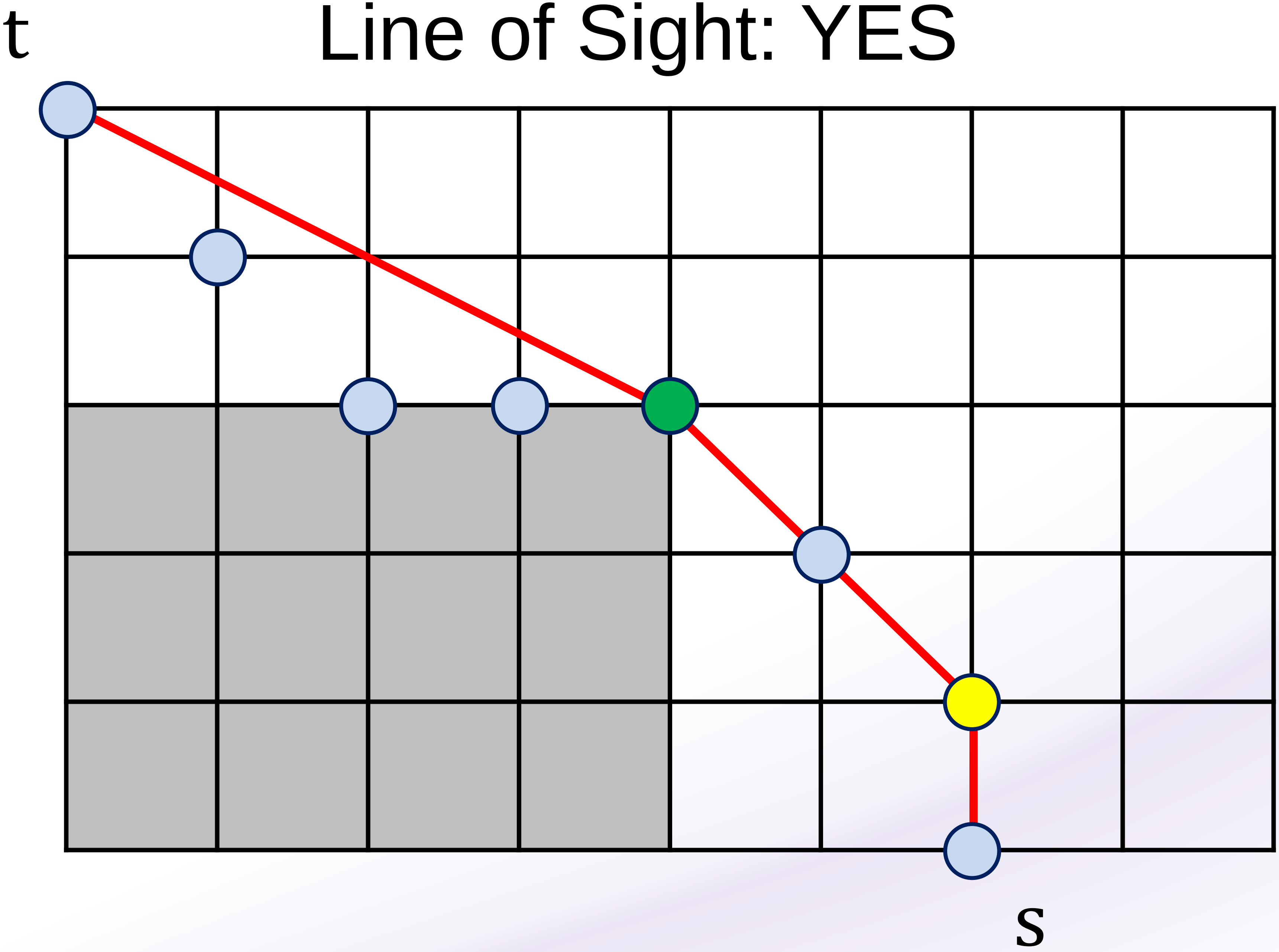
# Line of Sight: NO



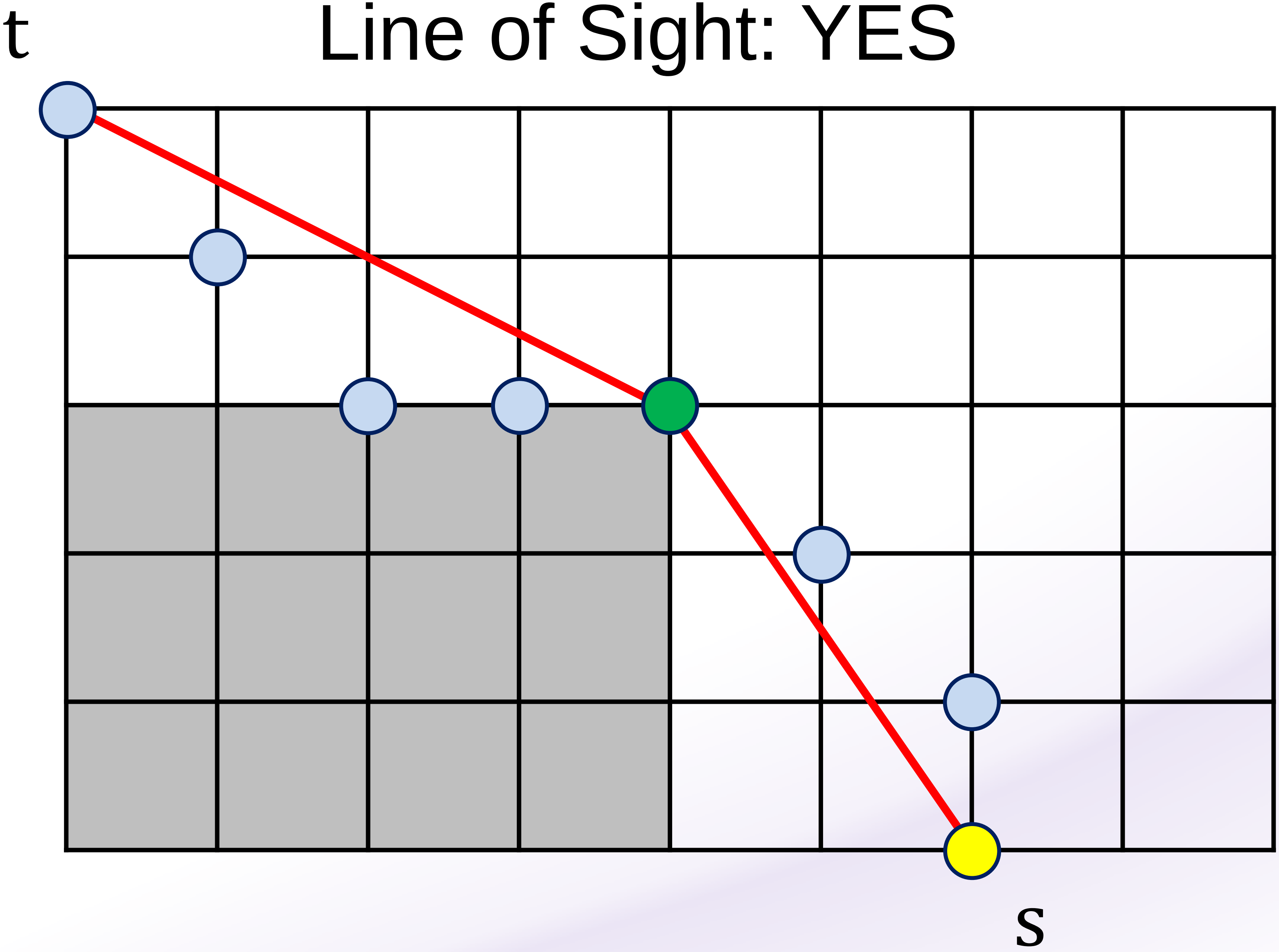
# Line of Sight: YES



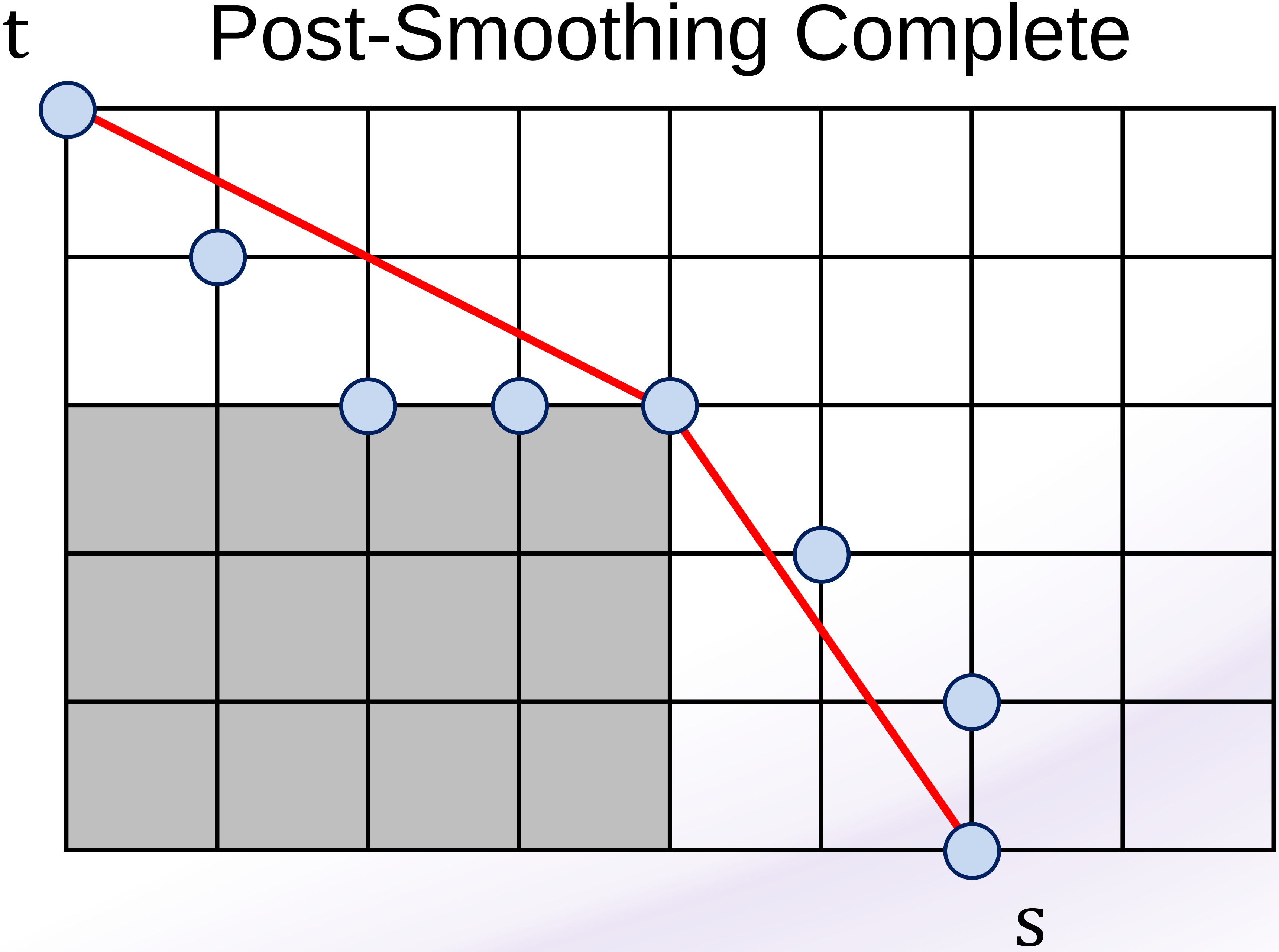
# Line of Sight: YES



# Line of Sight: YES



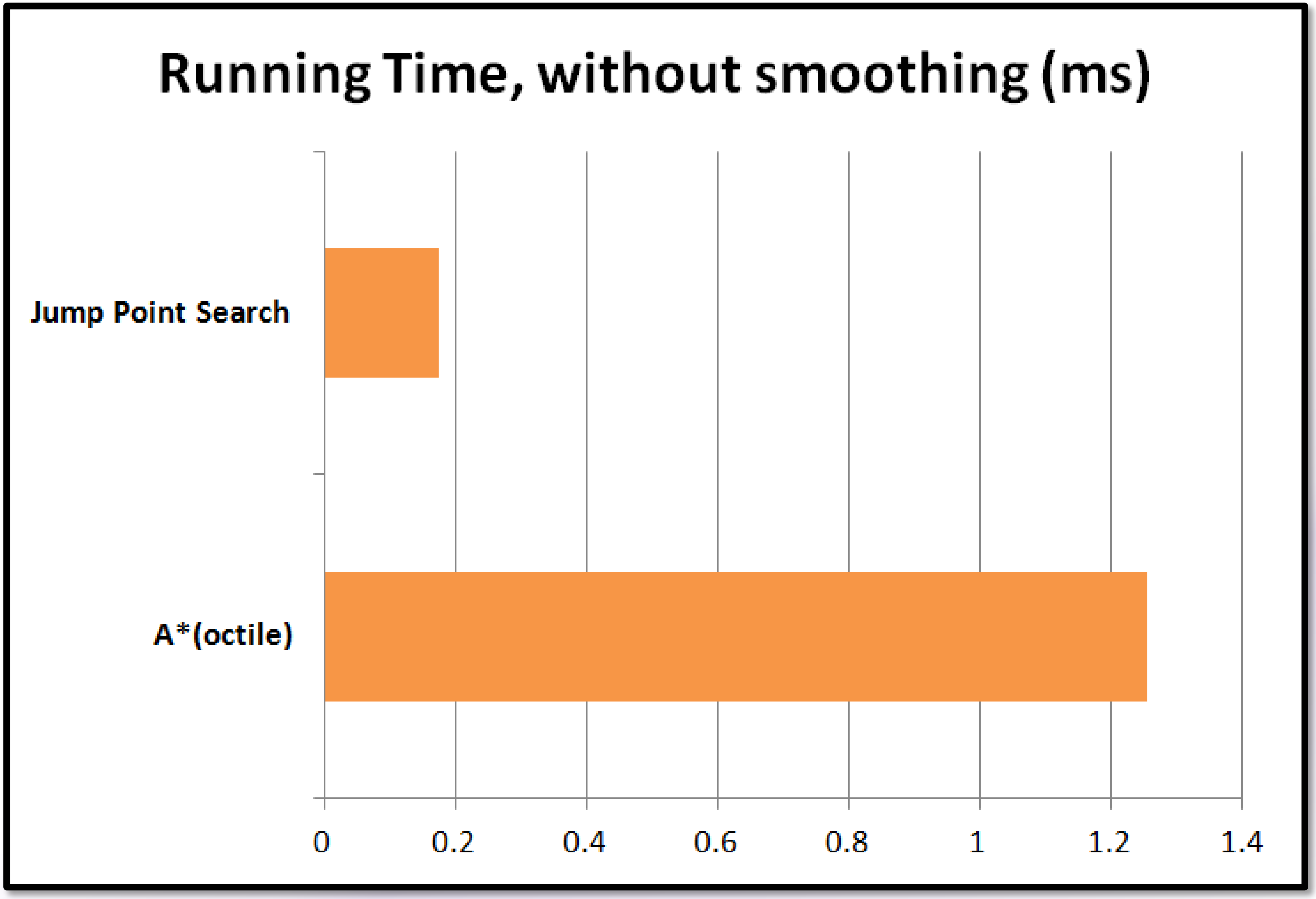
# Post-Smoothing Complete



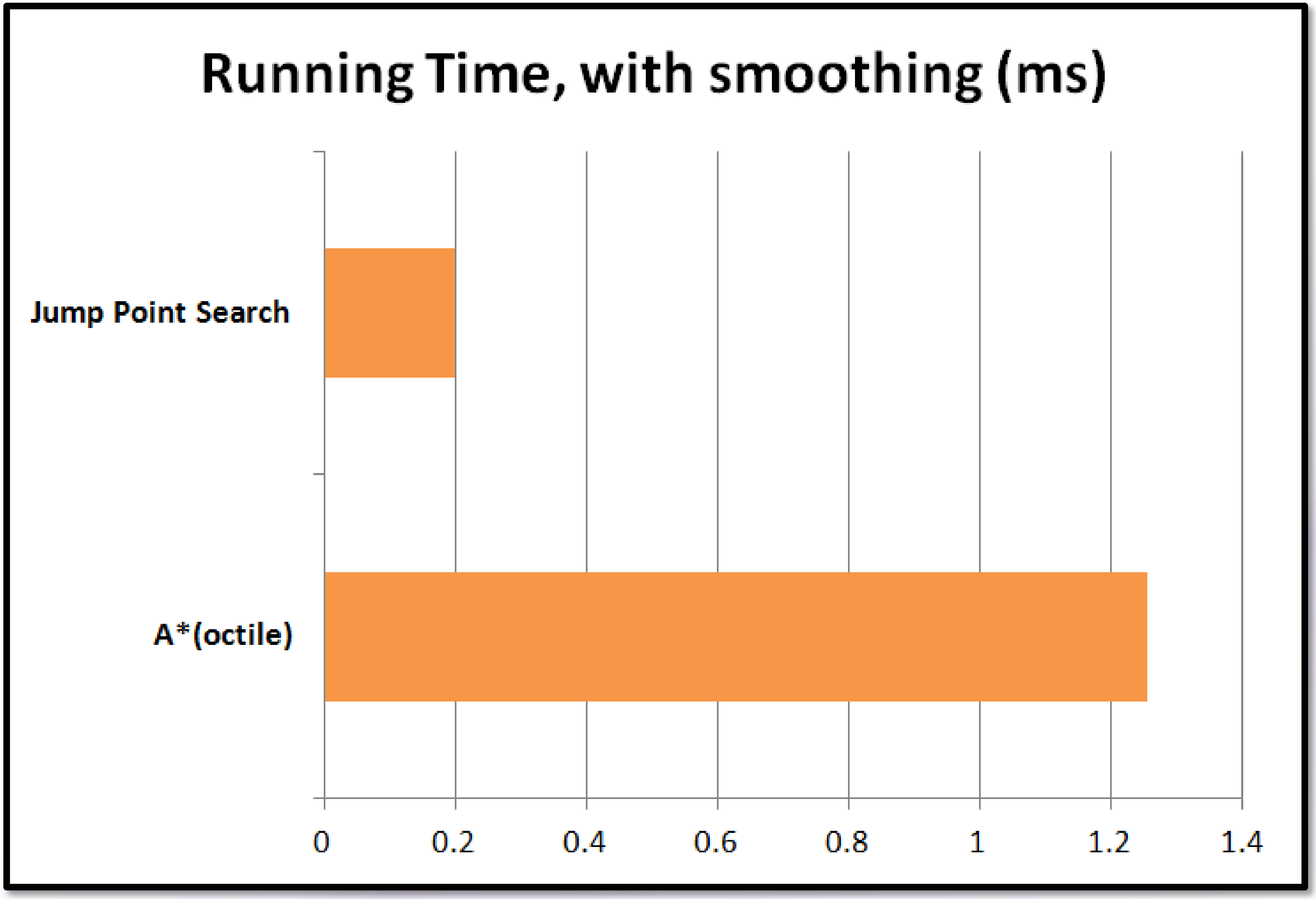
# Observations

## Post-smoothing takes negligible time

# Post-smoothing takes negligible time



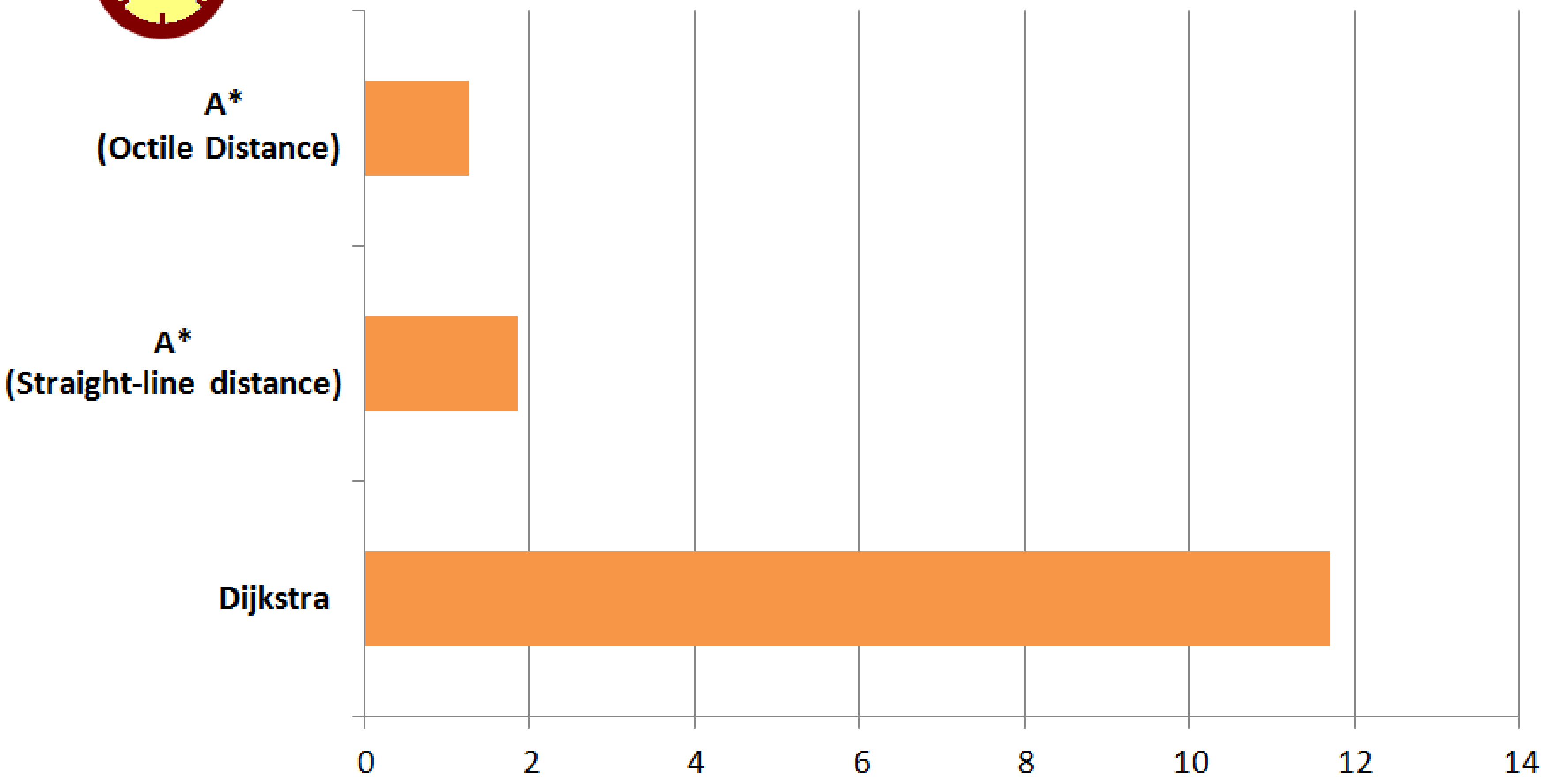
# Post-smoothing takes negligible time



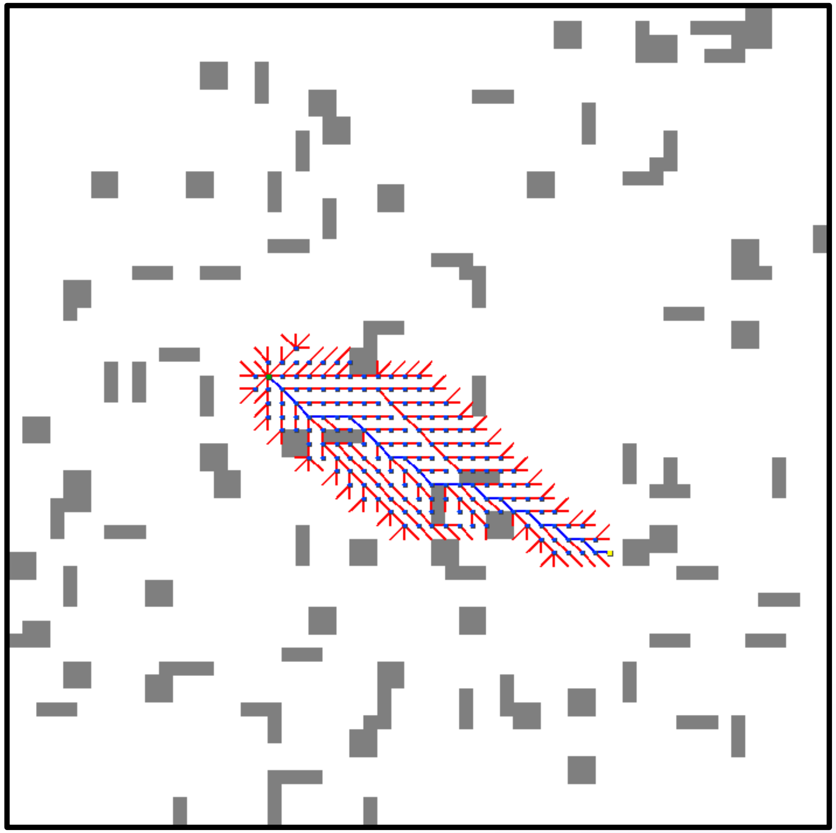
# Observations

**Don’t do pathfinding without a heuristic!**
**It makes a lot of difference in running time!**

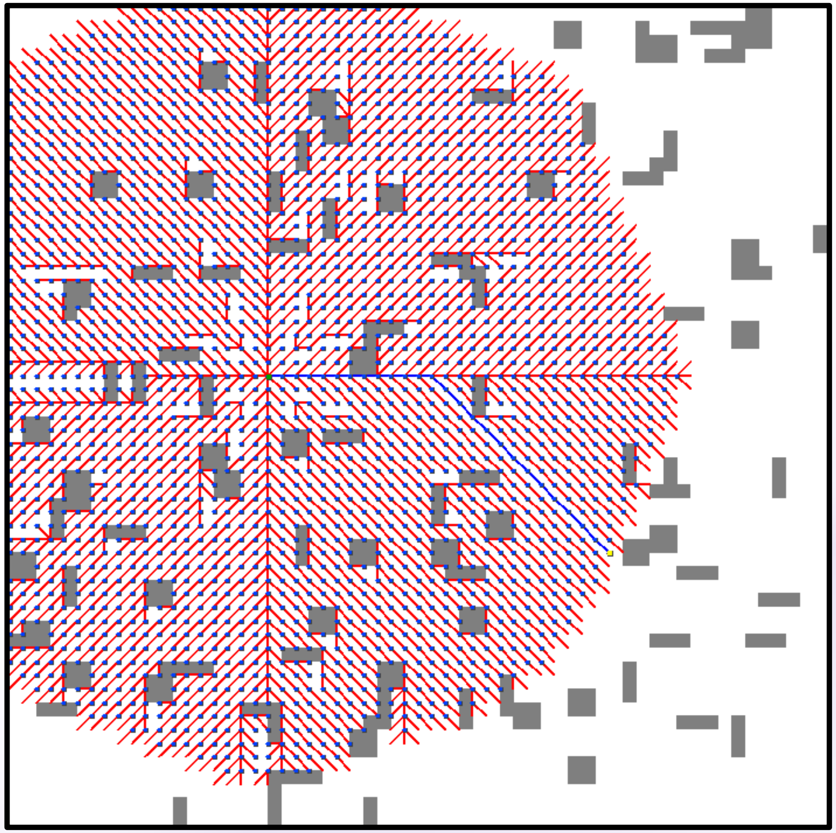
# Running Time (ms)



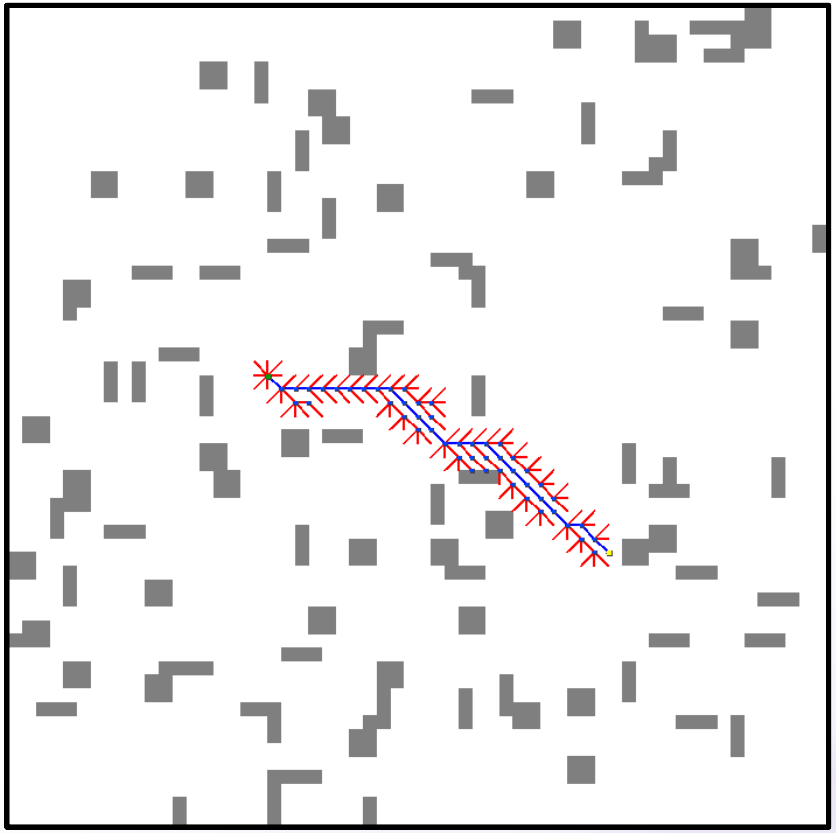
# A*
# Search Tree



# Dijkstra
# Search Tree



# A* with Octile Heuristic



# Conclusion

**Theta*:**

Runs fast, not optimal
But very close to optimal

**Visibility Graphs:**

Optimal, but runs very
slowly on large maps

**A* (8-directional):**

With the right heuristic, can run very fast.
But paths are low-quality, even after smoothing

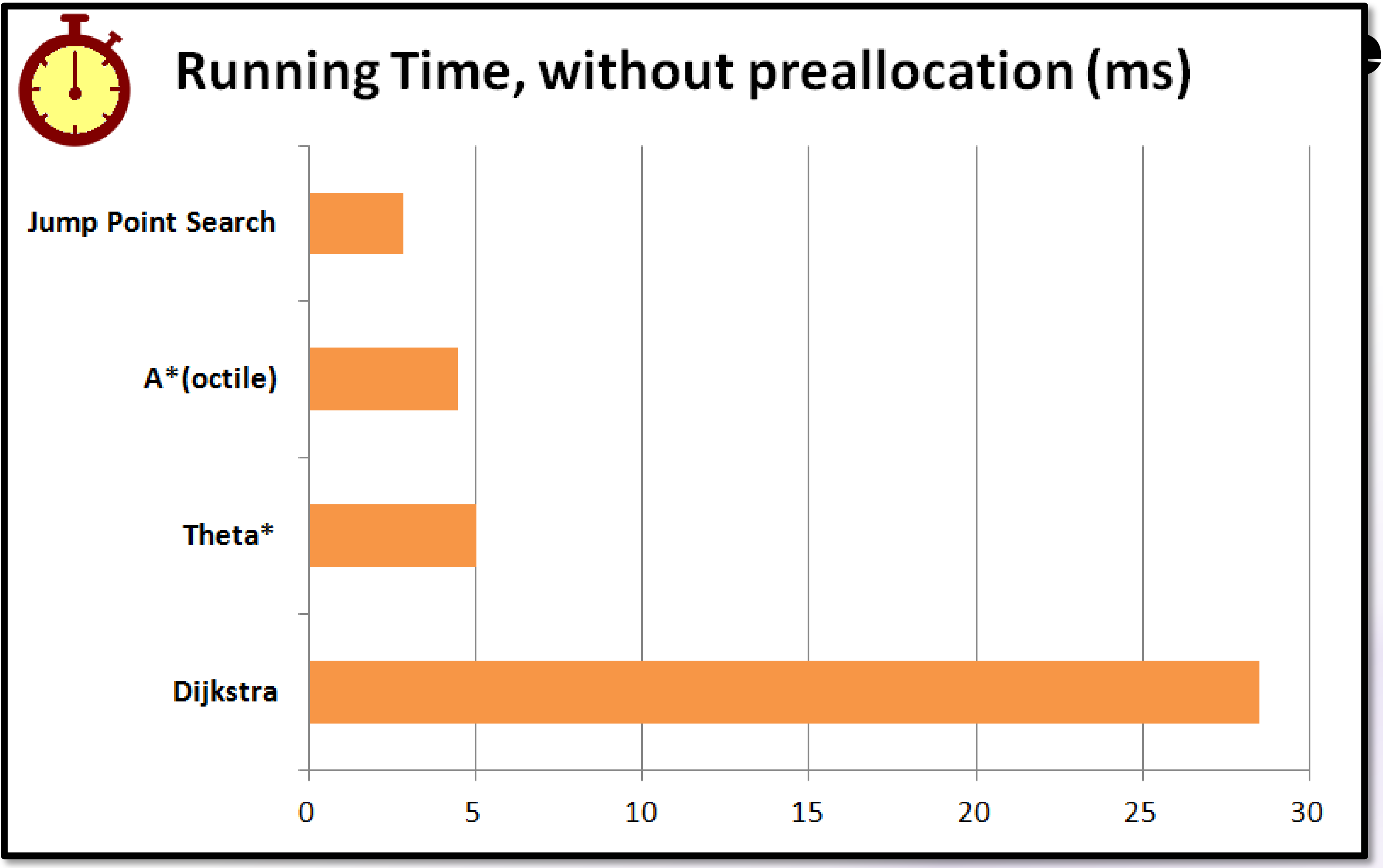
# General tips for Pathfinding in Games

1) The algorithm is often some variant of A*, specific to the game. First find out what is important to your game.
2) Preallocate all memory.
3) Have a debugging view to observe how your pathfinding algorithm works.

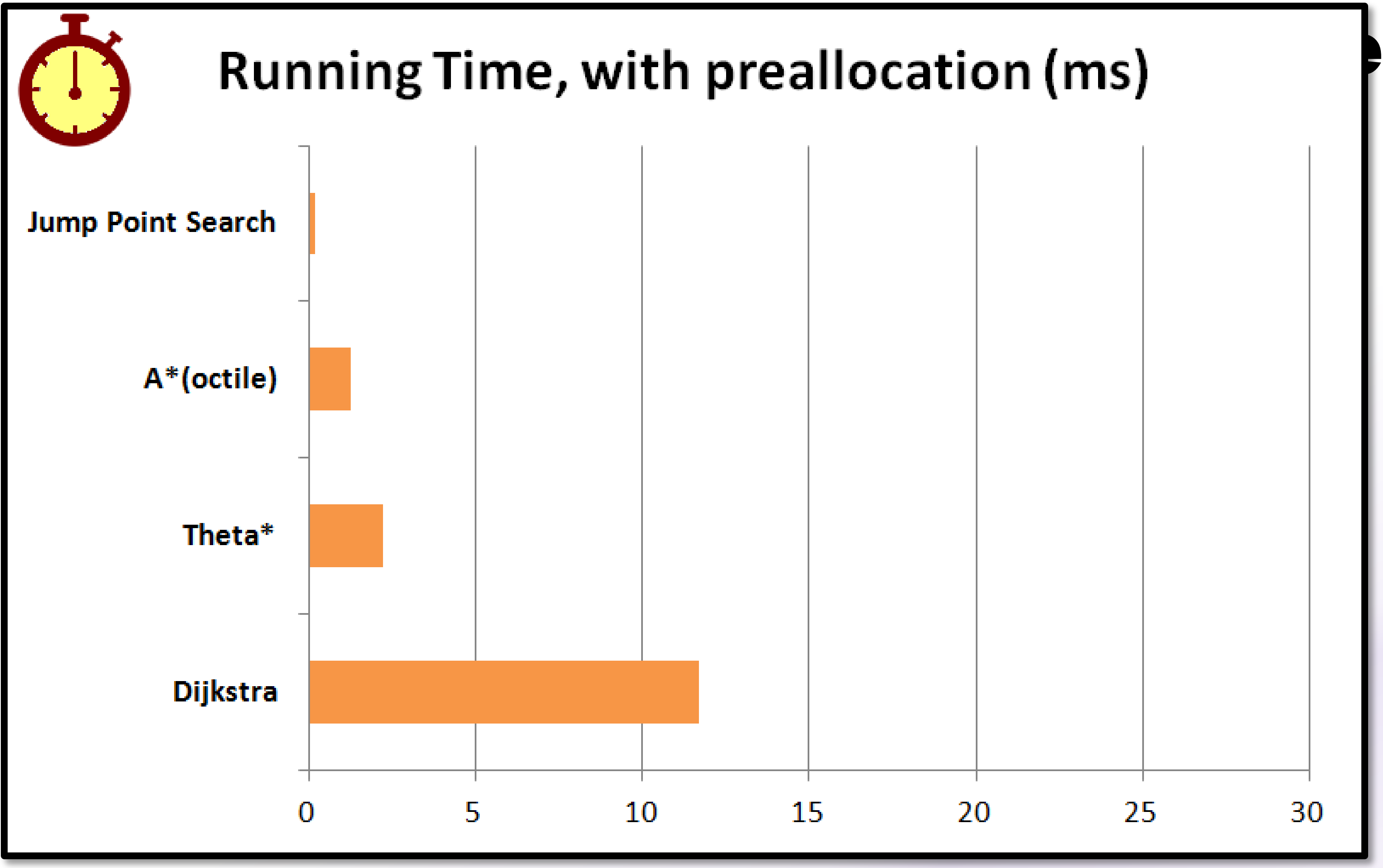
# General tips for Pathfinding in Games

1) The algorithm is often some variant of A*, specific to the game. First find out what is important to your game.
2) Preallocate all memory.
3) Have a debugging view to observe how your pathfinding algorithm works.

# G
es


Running Time, without preallocation (ms)

# G es



Running Time, with preallocation (ms)

# General tips for Pathfinding in Games

1) The algorithm is often some variant of A*, specific to the game. First find out what is important to your game.
2) Preallocate all memory.
3) Have a debugging view to observe how your pathfinding algorithm works.

# General tips for Pathfinding in Games

1) The algorithm is often some variant of A*, specific to the game. First find out what is important to your game.
2) Preallocate all memory.
3) Have a debugging view to observe how your pathfinding algorithm works.

# General tips for Pathfinding in Games

1) The algorithm is often some variant of A*, specific to the game. First find out what is important to your game.
2) Preallocate all memory.
3) Have a debugging view to observe how your pathfinding algorithm works.

# What I do:

Making a detailed comparison of the various algorithms regarding their utility when making games

Developing a variant of Theta* which finds much better paths than the original

Making animated visualisations for all the algorithms

And making games ☺

**Implementations on Github:**
github.com/Ohohcakester/Any-Angle-Pathfinding

# Advancements in Any-Angle path planning

**2005: Field D*** (Ferguson, Stentz)

**2007: Theta*** (Daniel, Nash, Koenig)

**2009: Accelerated A*** (Sislak, Volf, Pechoucek)

**2011: Block A*** (Yap, Burch, Holte, Schaeffer)

**2013: Anya** (Daniel, Alban)

# Acknowledgements

A significant number of the benchmark maps used for testing are taken from *movingai.com*.

Daniel, K.; Nash, A.; Koenig, S.; and Felner, A. (2010). Theta*: Any-angle path planning on grids. JAIR 39:533–579.

James, A. (2011). AI Navigation: It's Not a Solved Problem... Yet!. In GDC AI Summit 2011.

Daniel, H.; Alban, G. (2011). Online Graph Pruning for Pathfinding on Grid Maps. In AAAI.

Alex, J. (2013, July 16). Lazy Theta*: Faster Any-Angle Path Planning. Retrieved July 2, 2015, from http://aigamedev.com/open/tutorial/lazy-theta-star/

Sturtevant, N. (n.d.). A* Tie Breaking. Retrieved July 2, 2015, from http://movingai.com/astar.html

# Let’s end with a little demo of Theta* in action

# Questions?

# Questions?

Preallocating Memory?
Turning time?
Character size?
Weighted maps?
Avoiding other moving characters?
What algorithms do games use?
Navmeshes or Grids?
What are the factors I should consider?

# Questions?

**Other Demos:**

Jump Point Search
Post-Smoothing
Lazy Theta*