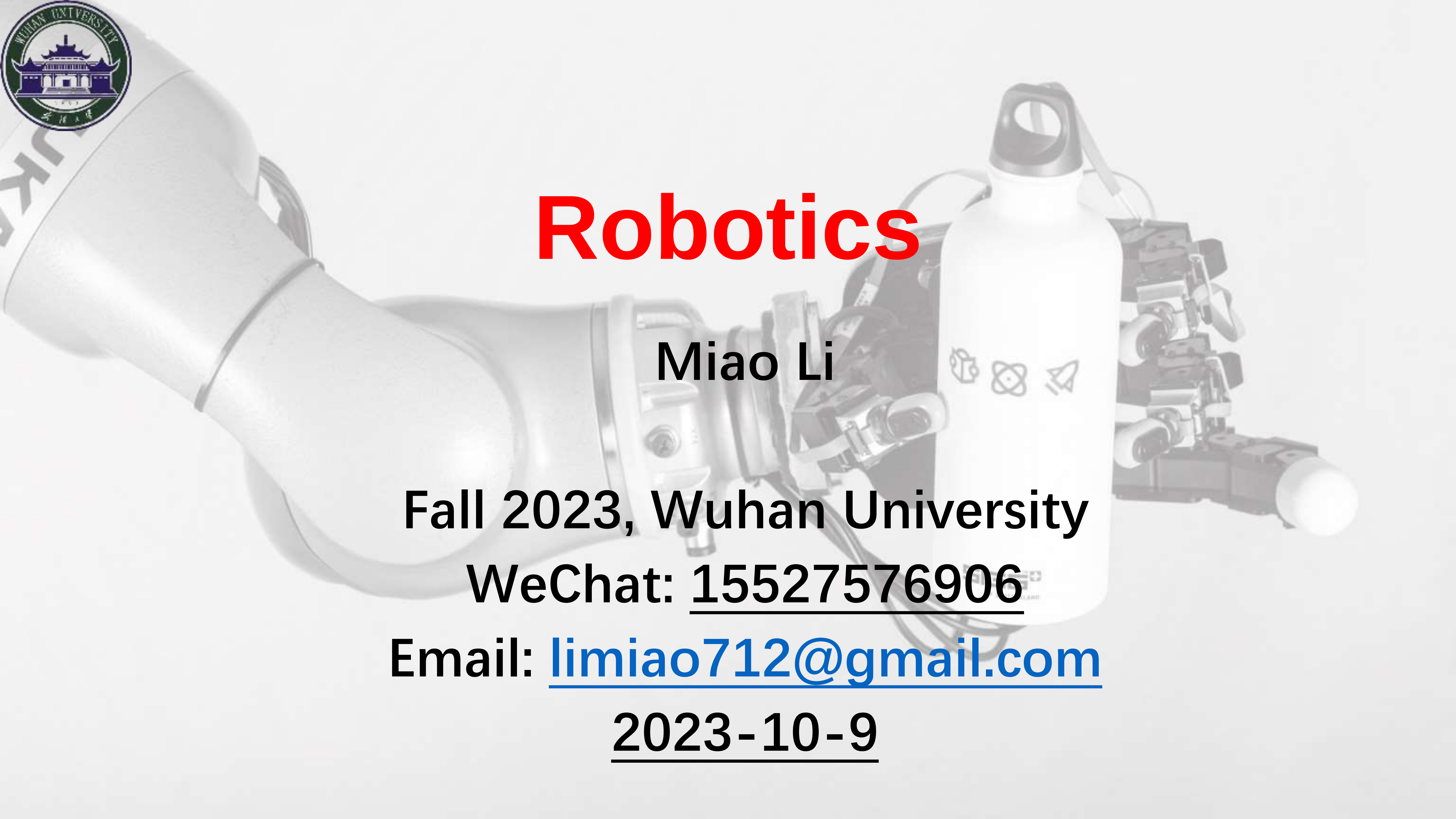


Robotics

Miao Li

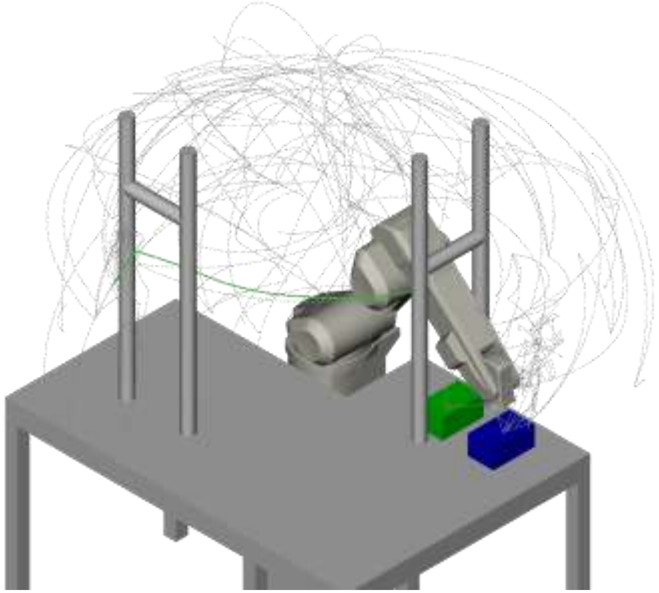
Fall 2023, Wuhan University
WeChat: 15527576906
Email: limiao712@gmail.com
2023-10-9

Goal for this course

- Design： soft hand design x1
- Perception: vision, point cloud, tactile, force/torque x1
- Planning: sampling-based, optimization-based, learning-based x3
- Control: feedback, multi-modal x2
- Learning: imitation learning, RL x2
- Simulation tool (pybullet, matlab, OpenRAVE, Issac Nvidia, Gazebo)
- How to get a robot moving!

Today’s Agenda

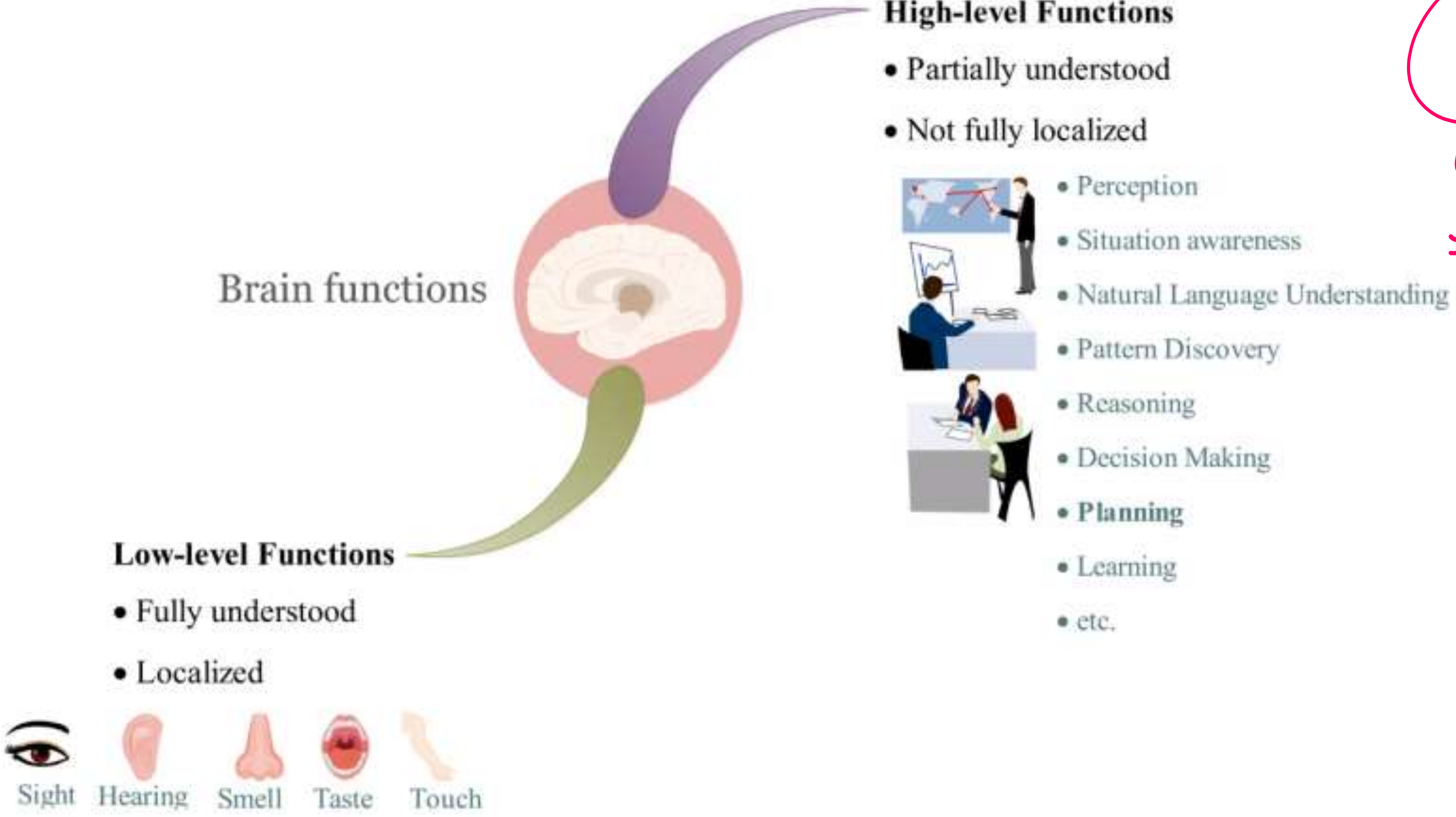
- **What is planning? (~10)**
- **Motion planning in robotic application （~10）**
    - **Self-driving, drone, robot arm, humanoids, medical robots, soft robots …**
- **Formulation of robot motion planning**
- **Planning as searching (~25)**
- **Planning as sampling (~25)**
    - **PRM, RRT, RRT***



What is planning?

Q: How can I get there from here?

A: Planning



Brain functions

High-level Functions

- Partially understood
- Not fully localized
  - Perception
  - Situation awareness
  - Natural Language Understanding
  - Pattern Discovery
  - Reasoning
  - Decision Making
  - **Planning**
  - Learning
  - etc.

Low-level Functions

- Fully understood
- Localized

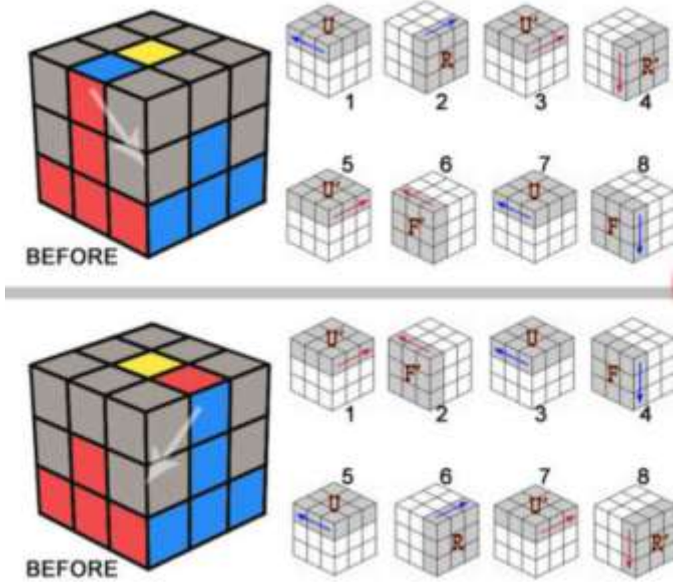
intial state

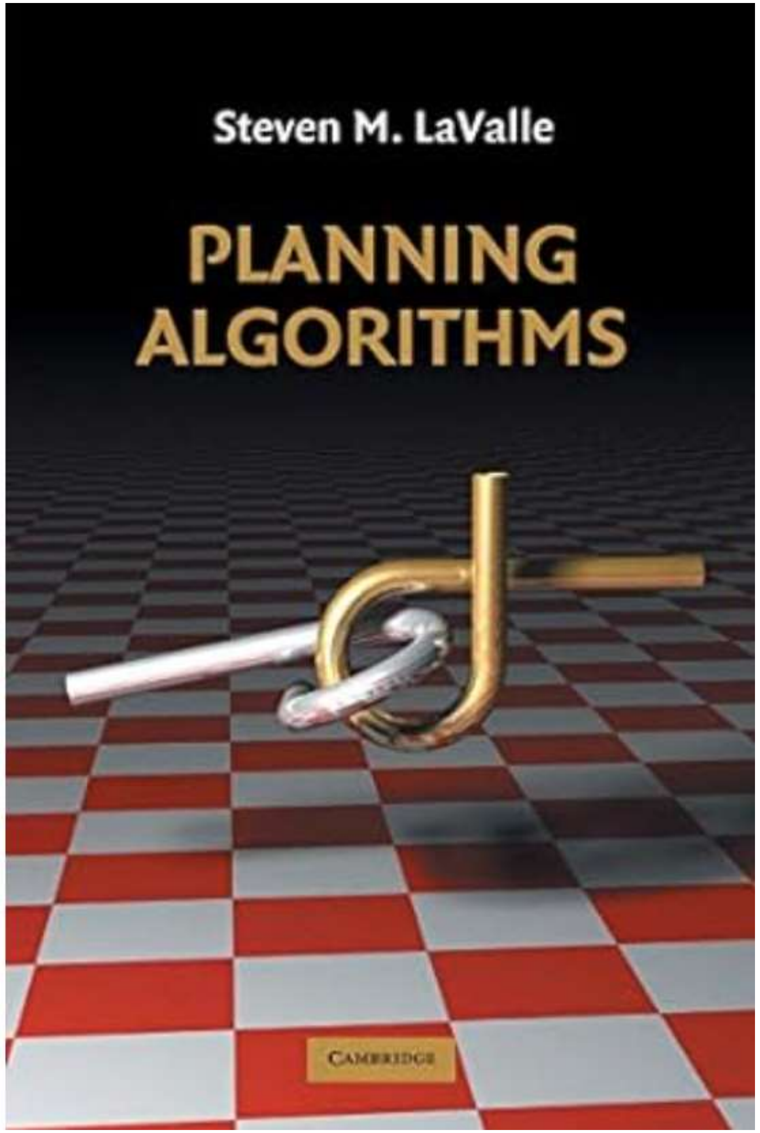
?

↓

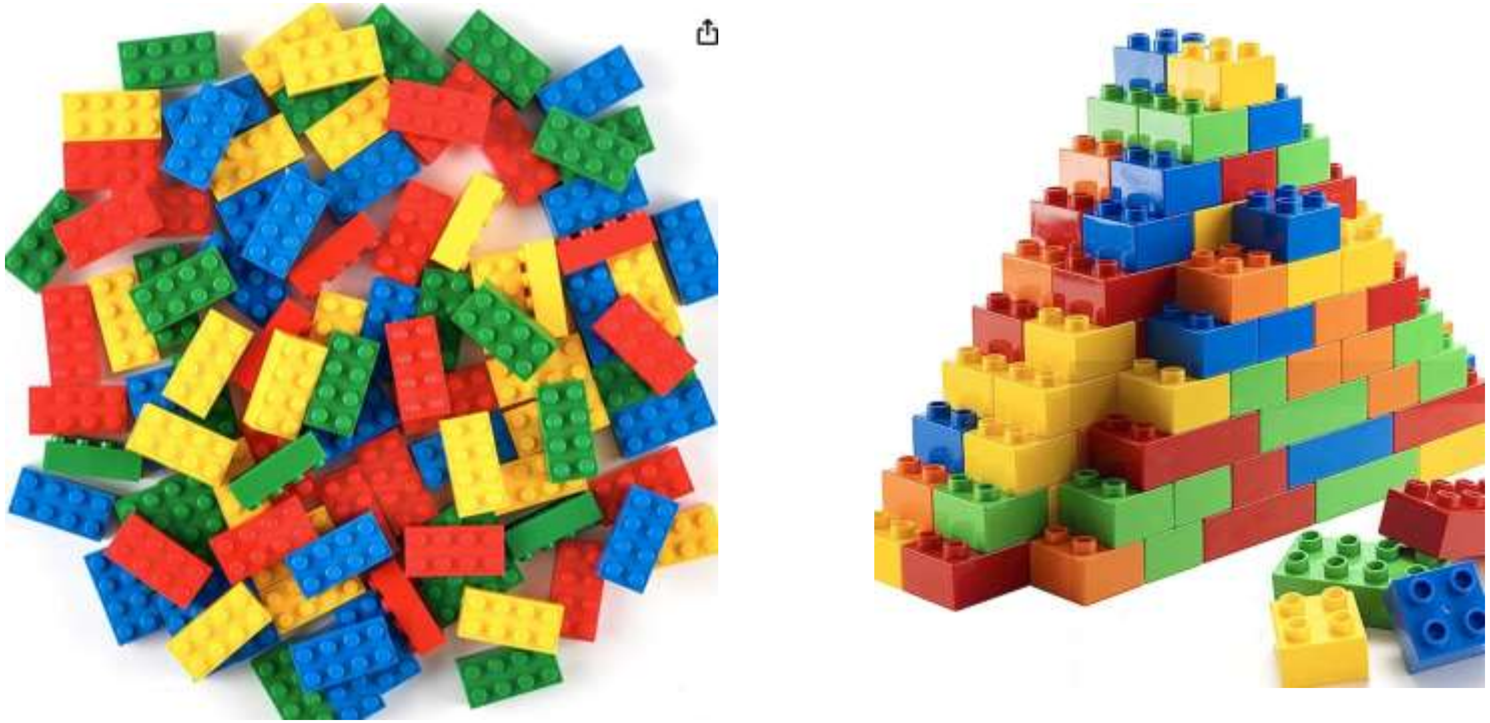
goal state.

What is planning?

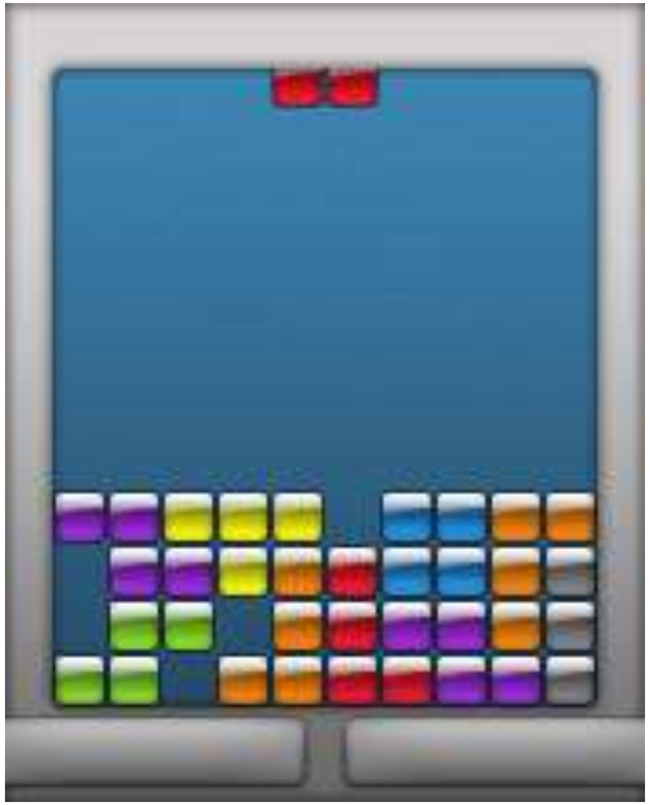




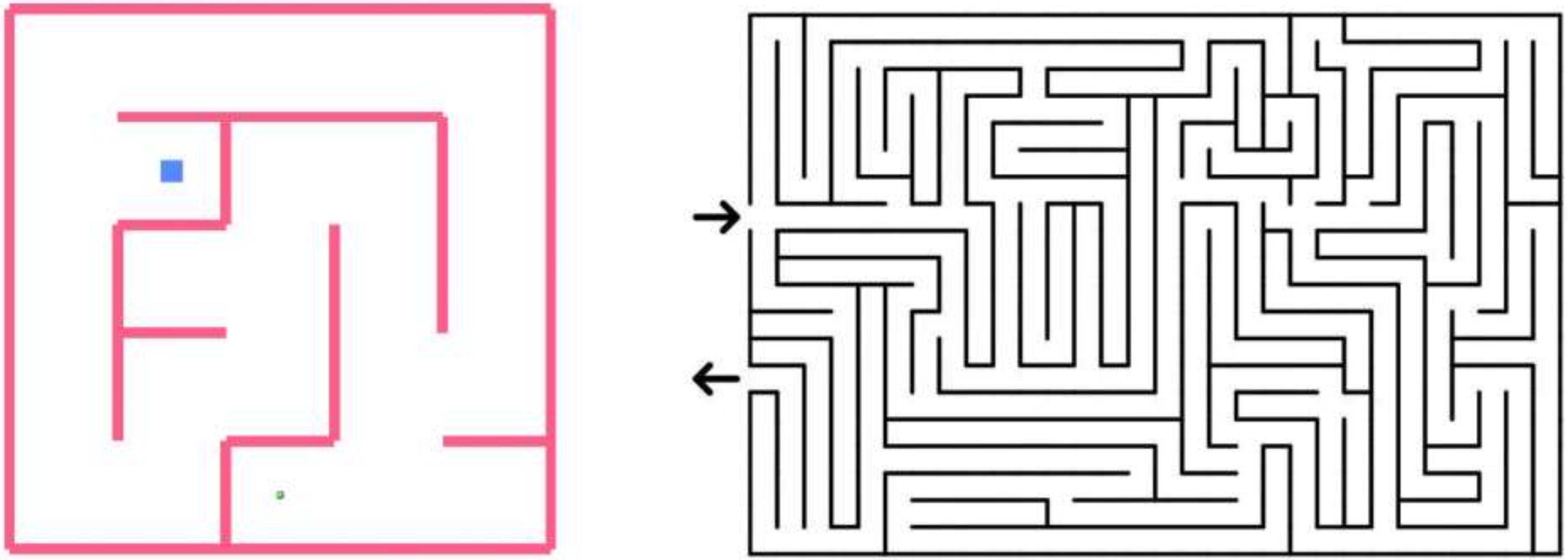
What is planning?



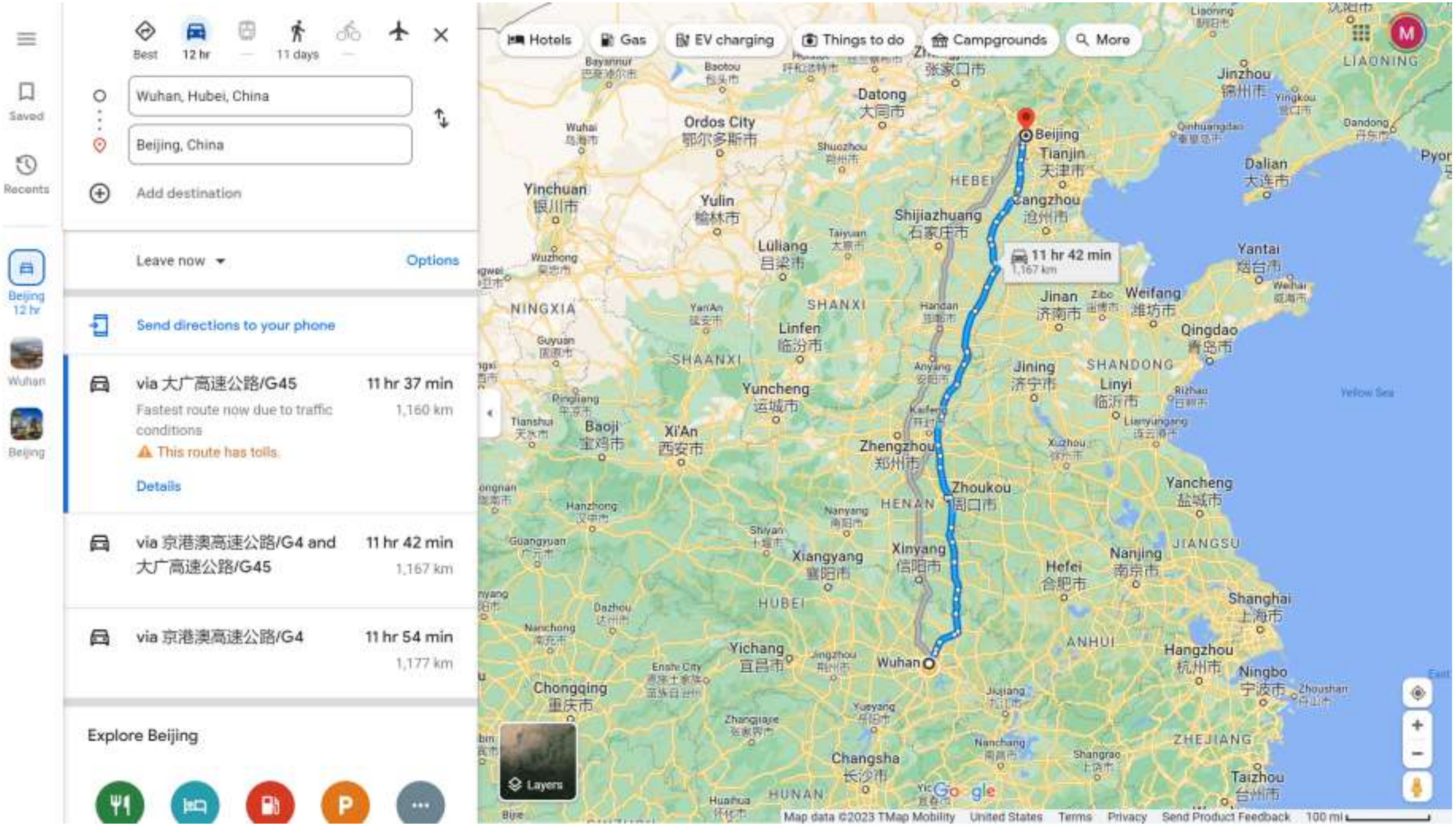
What is planning?



What is planning?



What is planning?



Wuhan, Hubei, China

Beijing, China

Add destination

Leave now

Options

Send directions to your phone

via 大广高速公路/G45 — 11 hr 37 min — 1,160 km

Fastest route now due to traffic conditions

This route has tolls.

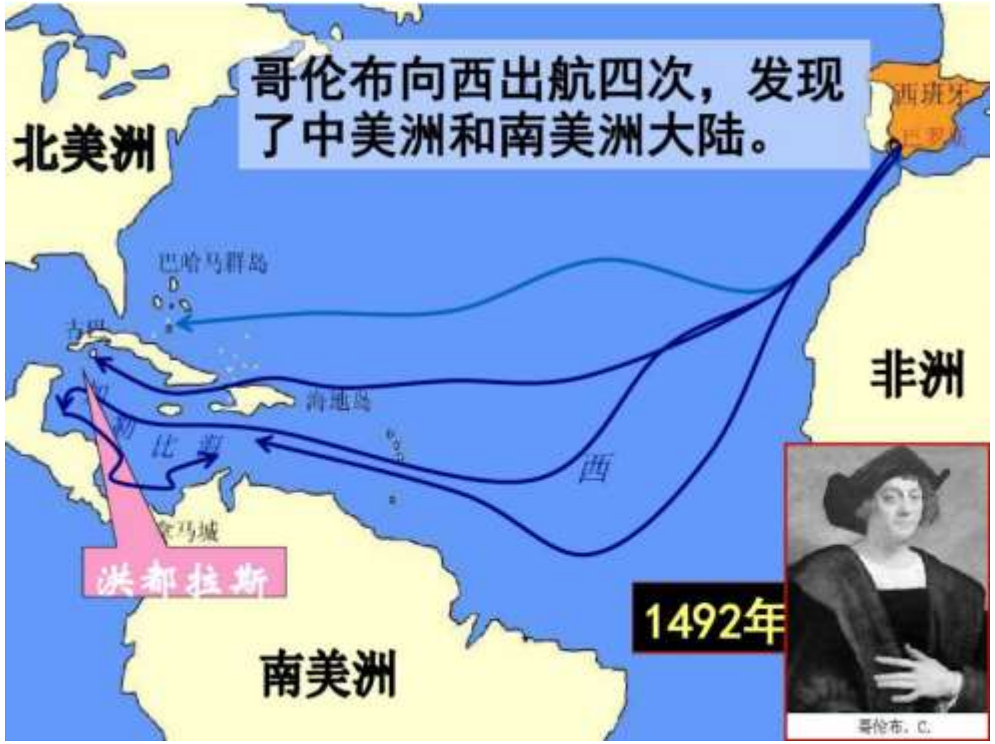
Details

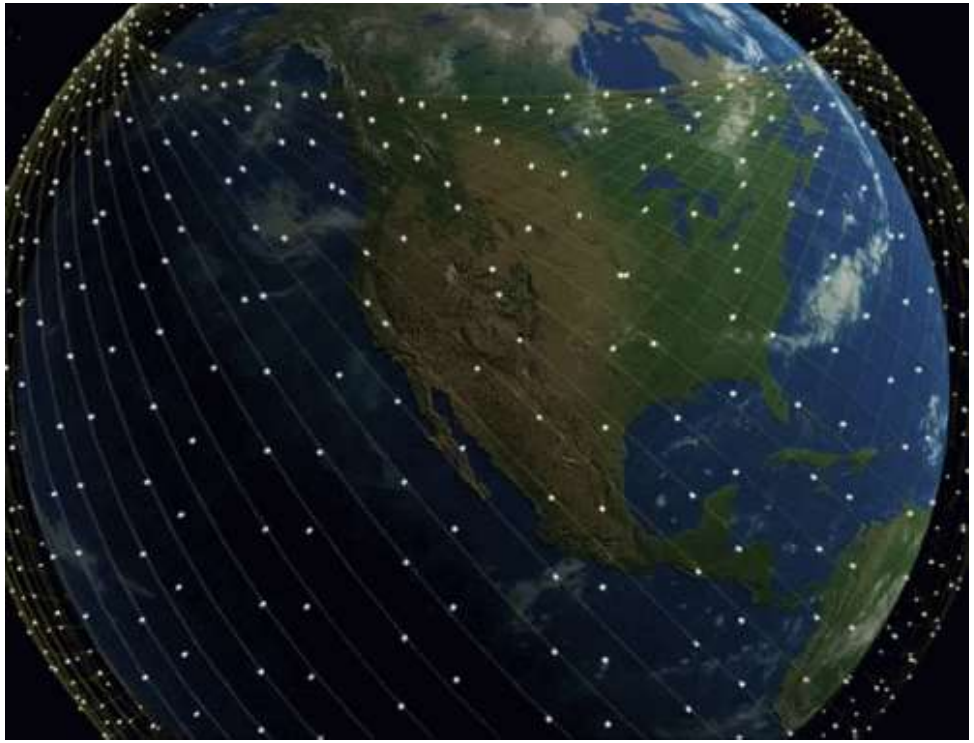
via 京港澳高速公路/G4 and 大广高速公路/G45 — 11 hr 42 min — 1,167 km

via 京港澳高速公路/G4 — 11 hr 54 min — 1,177 km

Explore Beijing

What is planning?





What is planning?

What is planning?

What is planning?

What is planning?

What is planning?

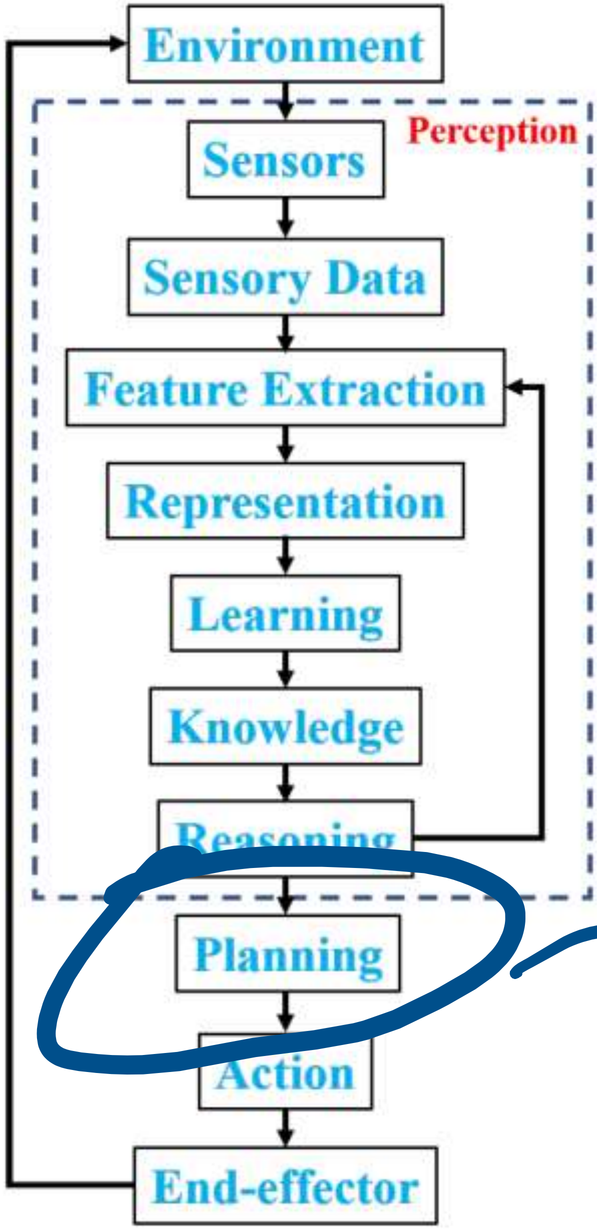
A **plan** is typically any diagram or list of steps with details of timing and resources, used to **achieve an objective to do something**. It is commonly understood as a temporal set of intended actions through which one expects to achieve a goal.

https://en.wikipedia.org/wiki/Plan

What is planning?

Any other interesting examples?

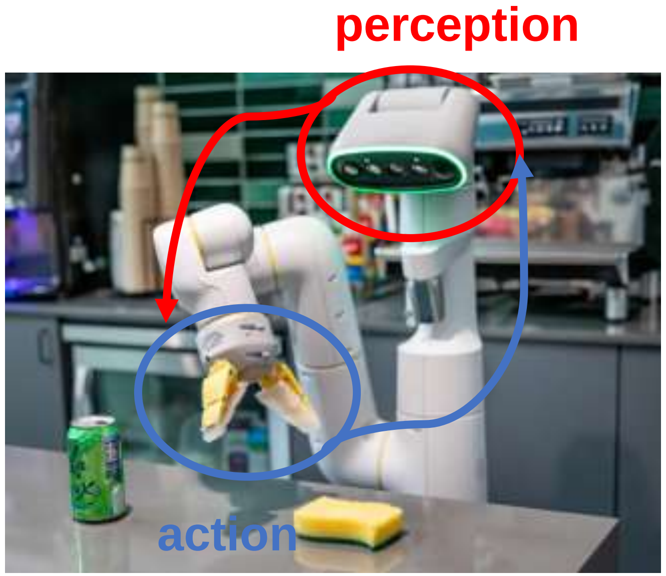
Planning in Robotics

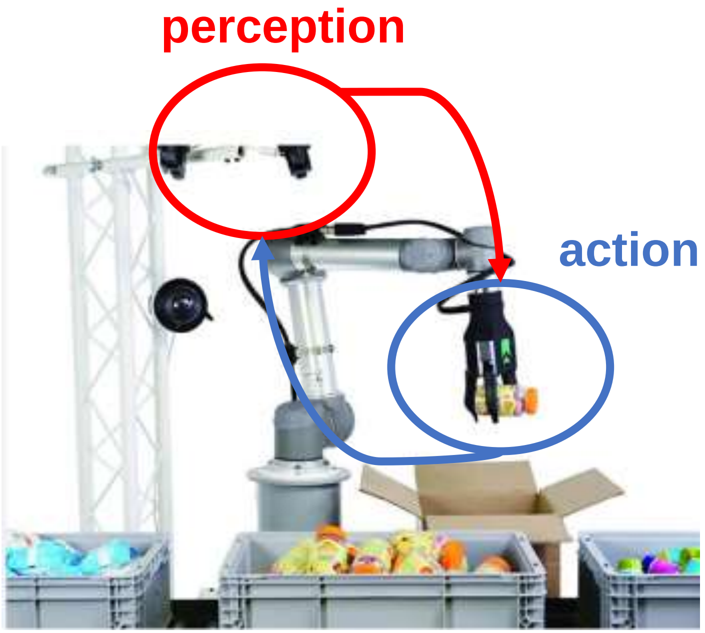

Environment
Perception
Sensors
Sensory Data
Feature Extraction
Representation
Learning
Knowledge
Reasoning
Planning
Action
End-effector

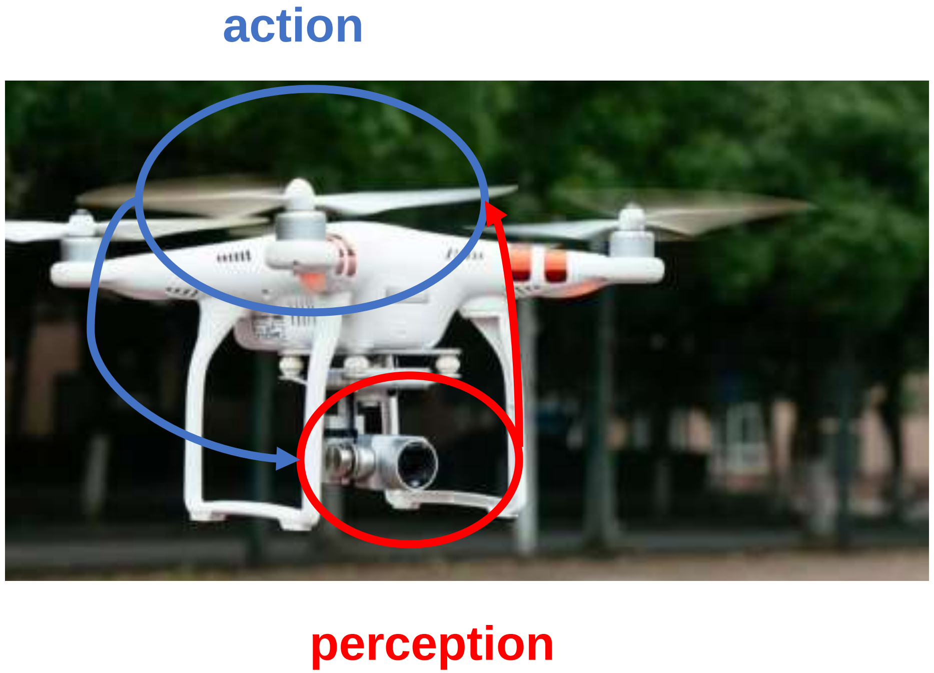
Robotics – Learn the mapping from perception to action

real-time.

Motion Planning in Robotics

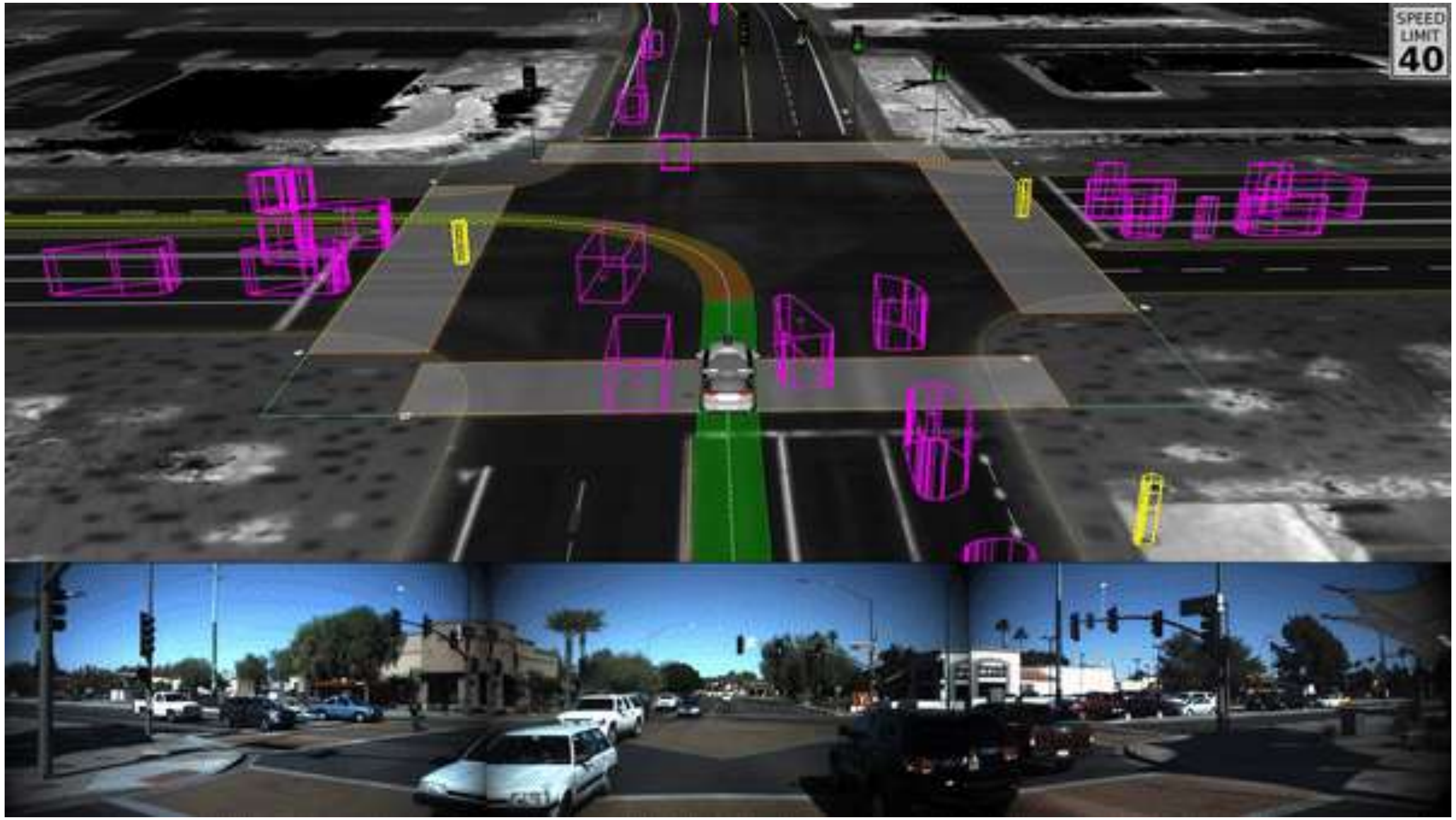






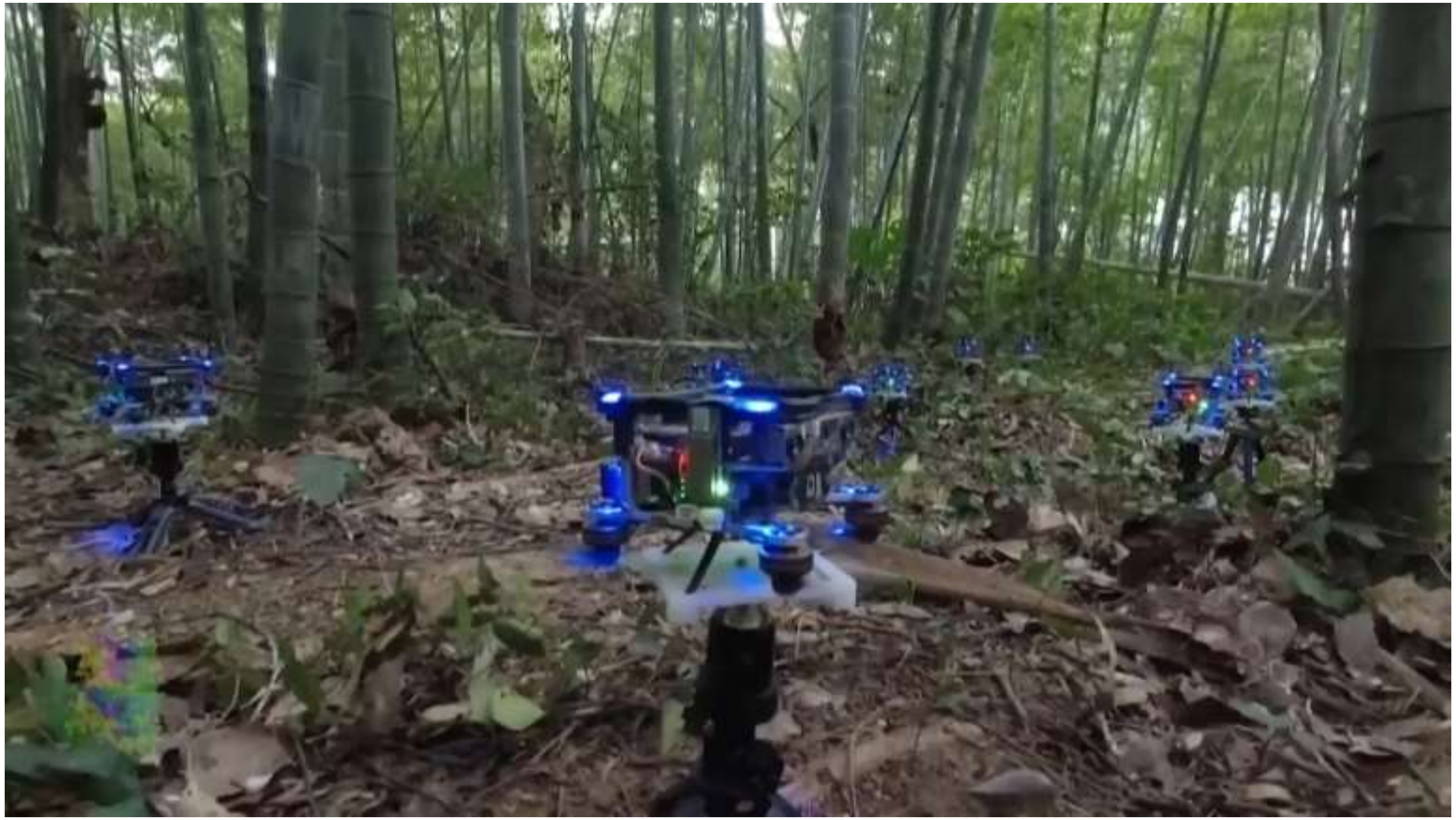
Robotics – Learn the mapping from perception to action

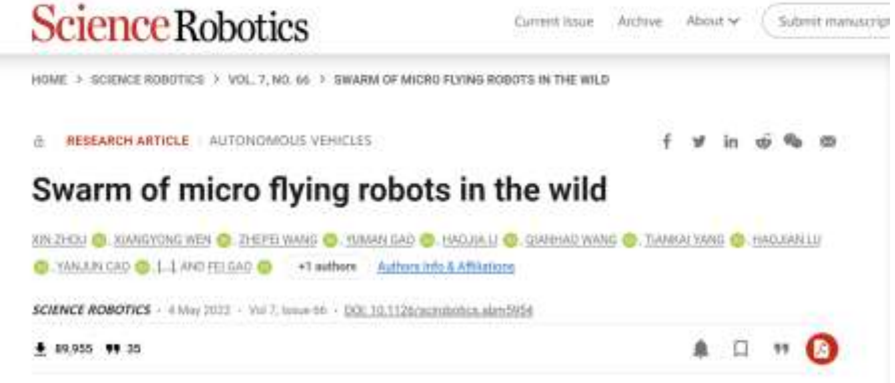
Motion Planning in Robotics



Self-driving

Motion Planning in Robotics





HOME > SCIENCE ROBOTICS > VOL. 7, NO. 66 > SWARM OF MICRO FLYING ROBOTS IN THE WILD

RESEARCH ARTICLE | AUTONOMOUS VEHICLES

Swarm of micro flying robots in the wild

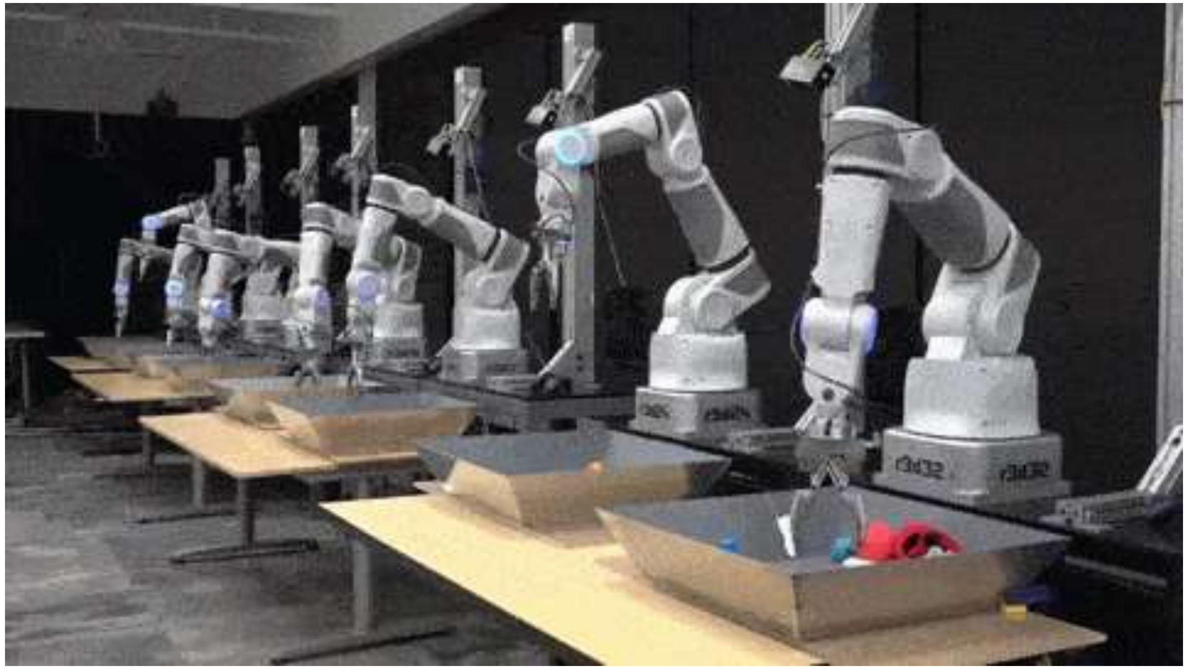
XIN ZHOU, XIANGYONG WEN, ZHEPEI WANG, YUMAN GAO, HAOJIA LI, QIANHAO WANG, TIANKAI YANG, HAOJIAN LU, YANJUN CAO, [...] AND FEI GAO +1 authors Authors Info & Affiliations

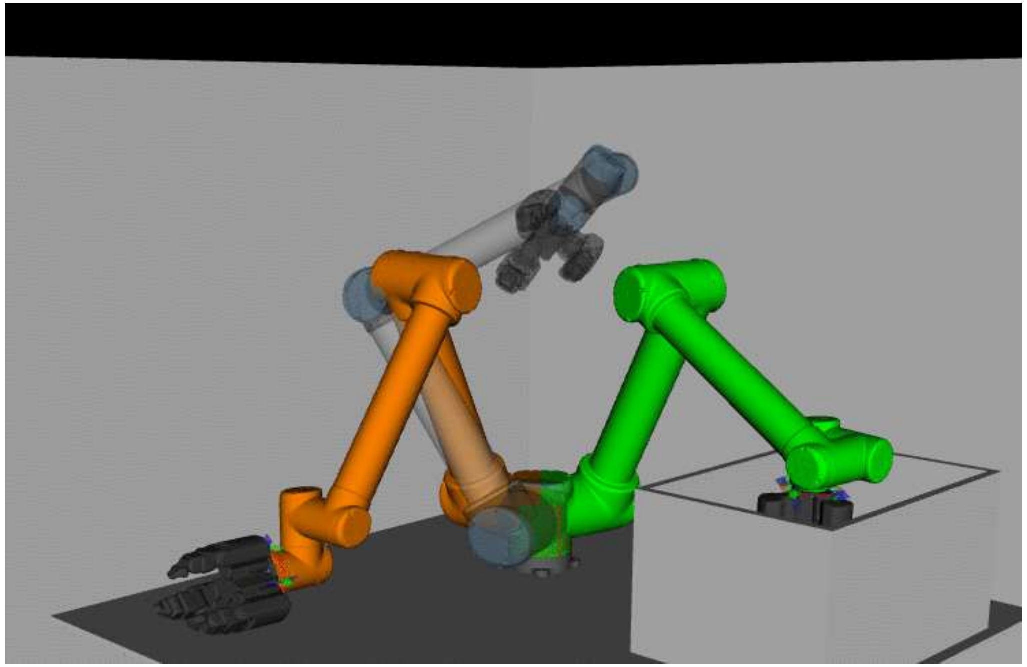
SCIENCE ROBOTICS • 4 May 2022 • Vol 7, Issue 66 • DOI: 10.1126/scirobotics.abm5954

89,955 35

Drones

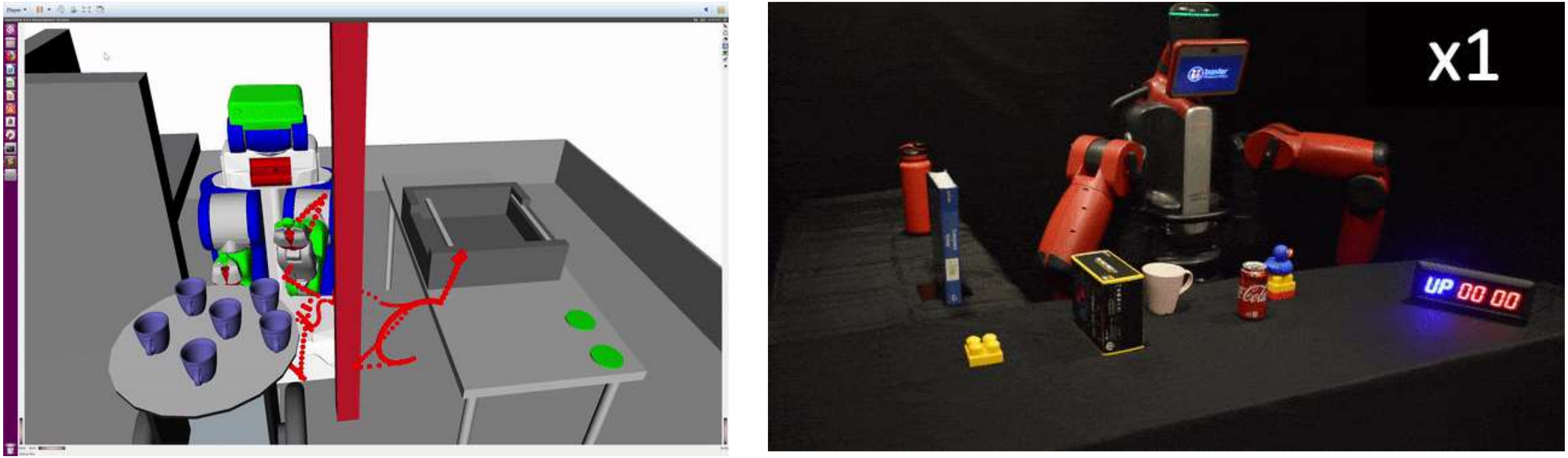
Motion Planning in Robotics





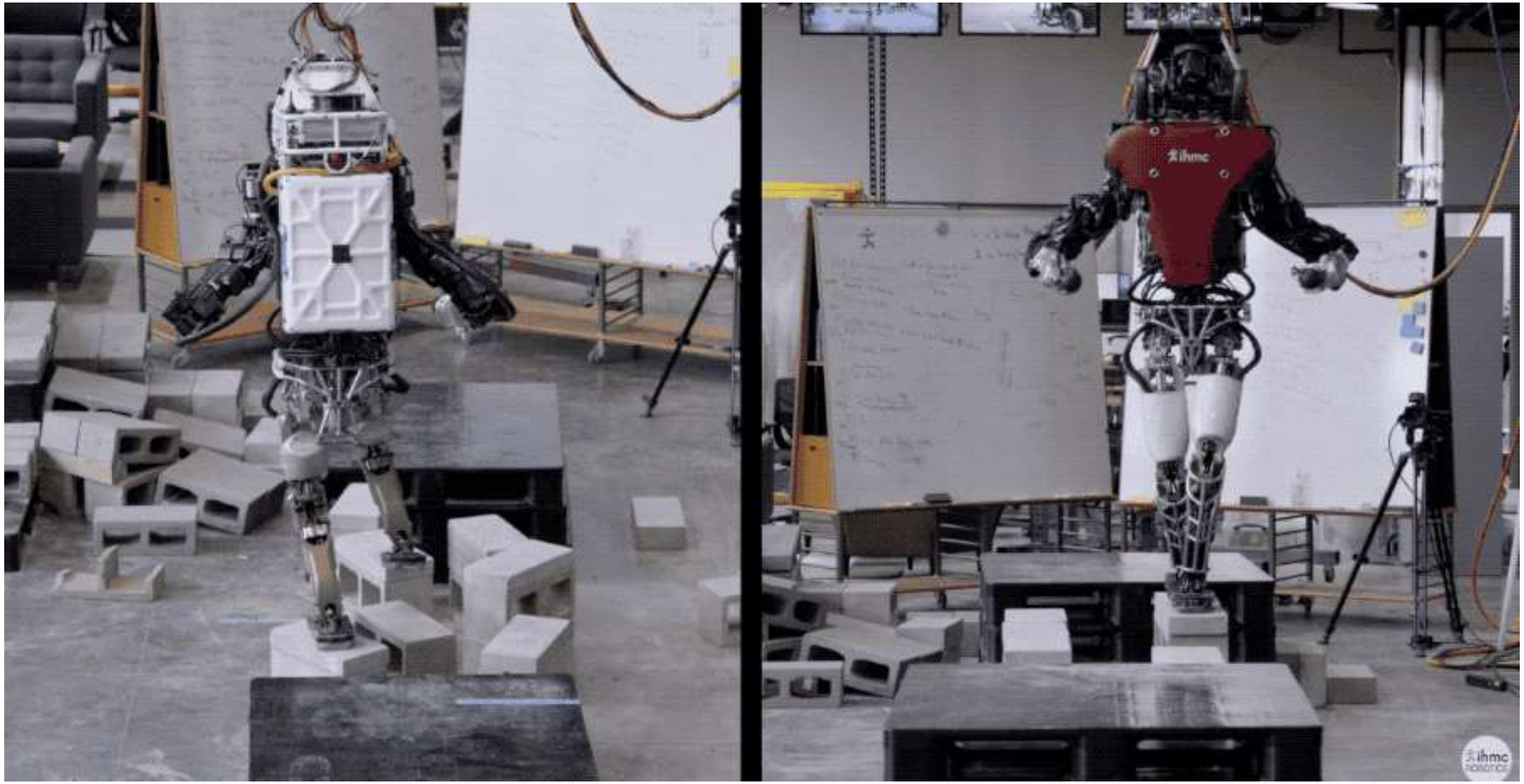
Robot arms

Motion Planning in Robotics



Bimanual manipulation

Motion Planning in Robotics



Humanoids

Motion Planning in Robotics

Humanoids

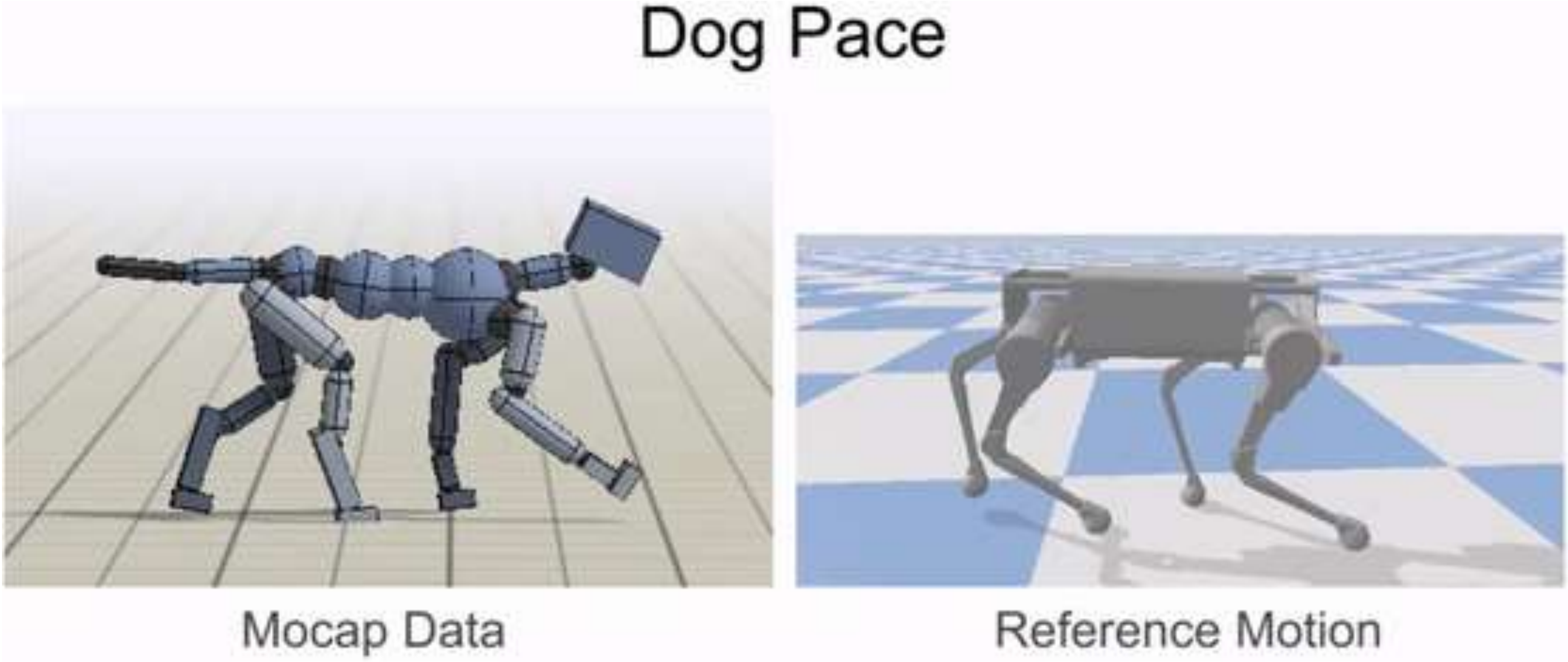
Motion Planning in Robotics

Optimus is now capable of self-calibrating its arms and legs

Tesla's Optimus Robot Sort Objects Autonomously
https://www.youtube.com/watch?v=oL5YNtDUQXU&ab_channel=CNETHighlights

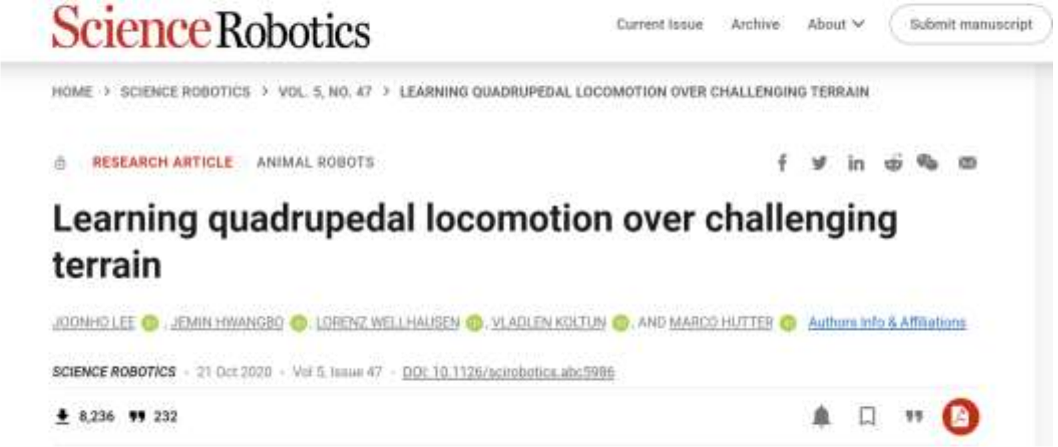
Humanoids

Motion Planning in Robotics



Quadruped robot

Motion Planning in Robotics



Science Robotics

Current Issue Archive About Submit manuscript

HOME › SCIENCE ROBOTICS › VOL. 5, NO. 47 › LEARNING QUADRUPEDAL LOCOMOTION OVER CHALLENGING TERRAIN

RESEARCH ARTICLE ANIMAL ROBOTS

Learning quadrupedal locomotion over challenging terrain

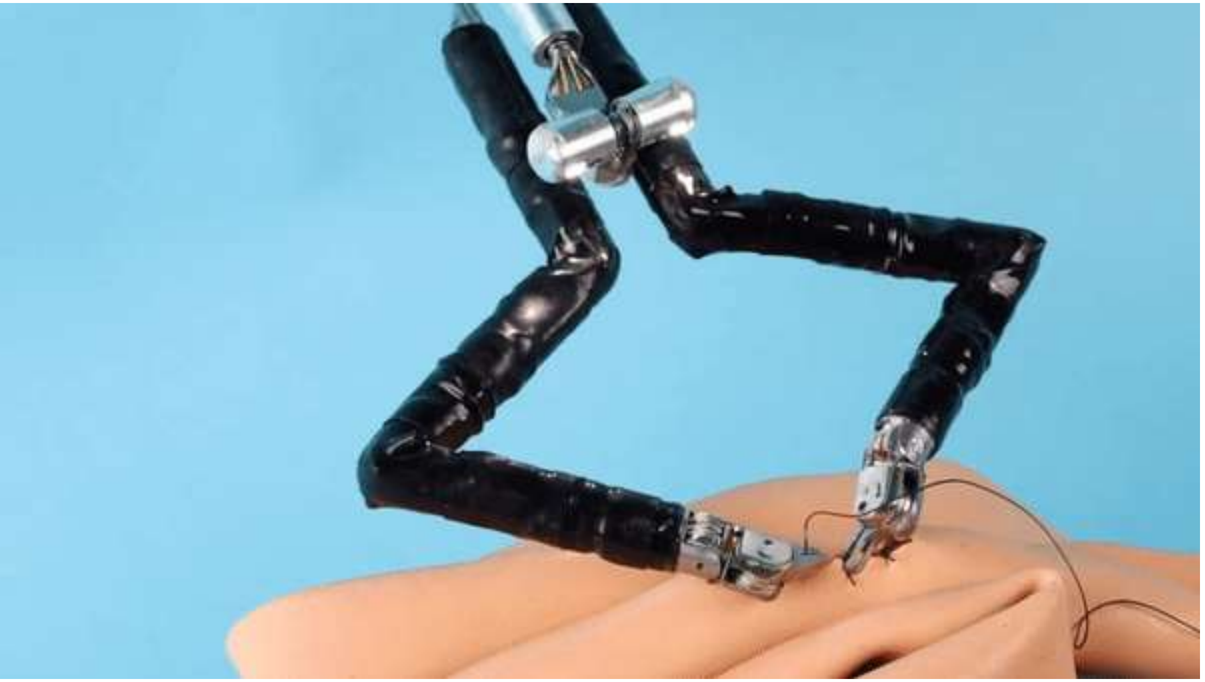
JOONHO LEE, JEMIN HWANGBO, LORENZ WELLHAUSEN, VLADLEN KOLTUN, AND MARCO HUTTER Authors Info & Affiliations

SCIENCE ROBOTICS • 21 Oct 2020 • Vol 5, Issue 47 • DOI: 10.1126/scirobotics.abc5986

8,236 232

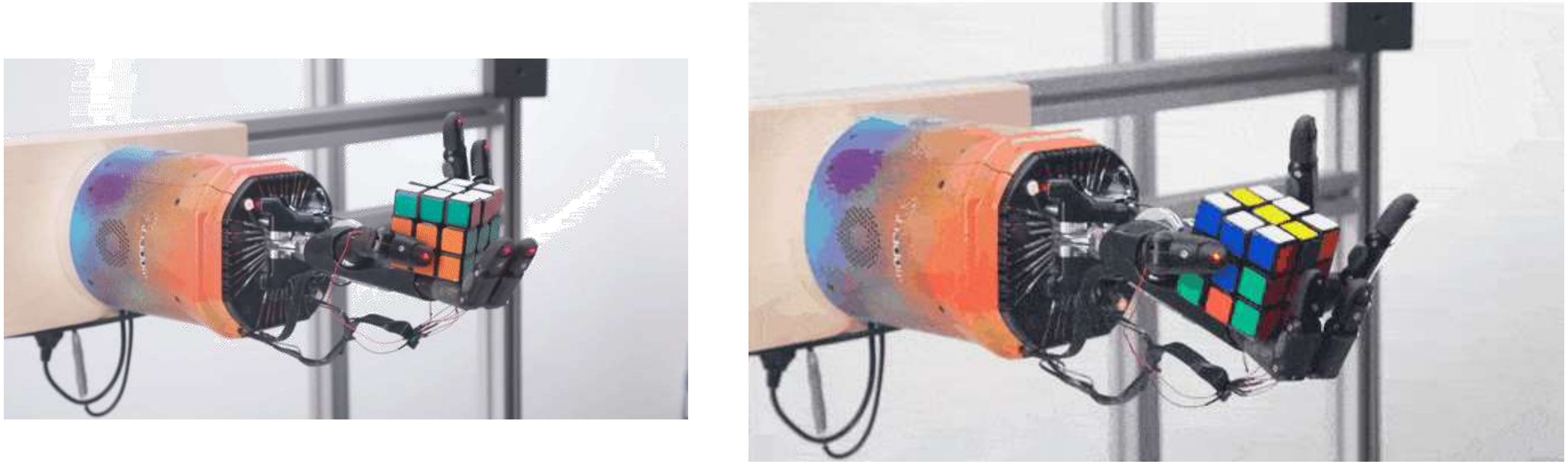
Quadruped robot

Motion Planning in Robotics



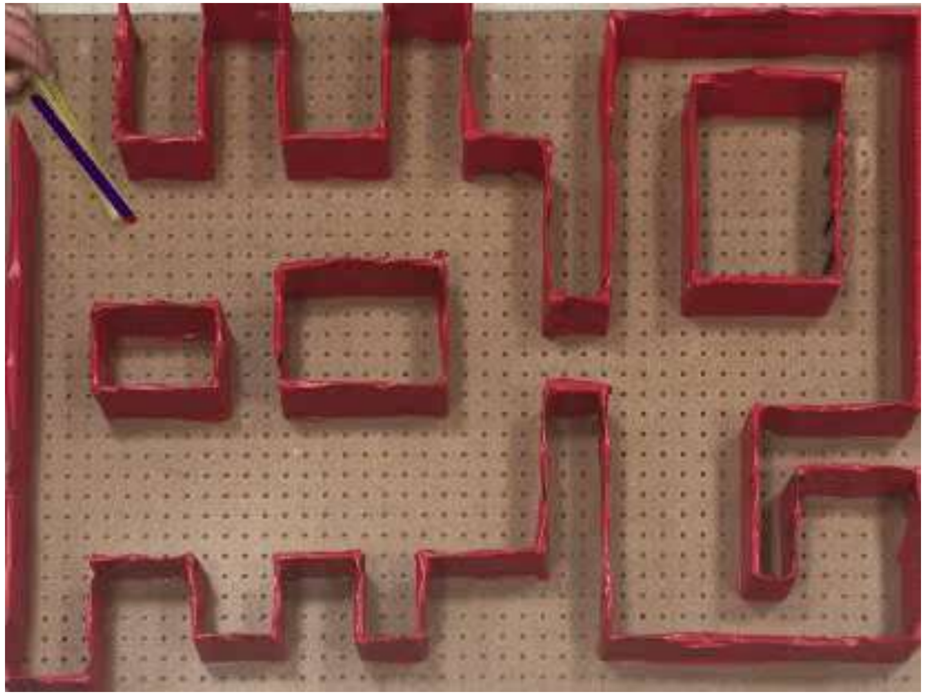
Medical robot

Motion Planning in Robotics



Dexterous manipulation

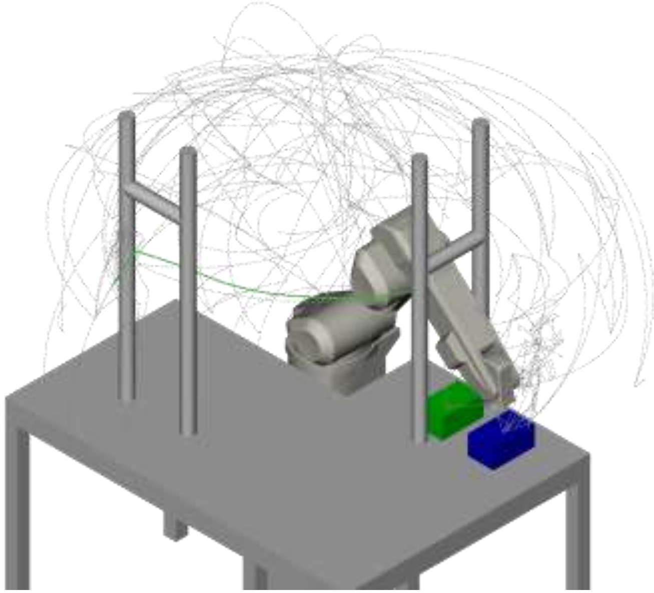
Motion Planning in Robotics



Soft robots

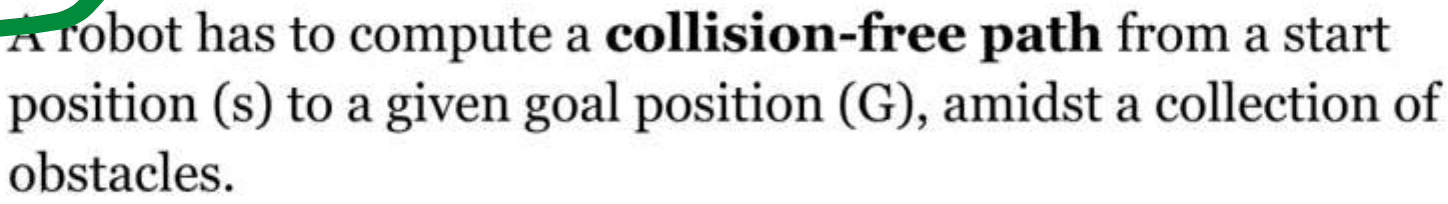
Today’s Agenda

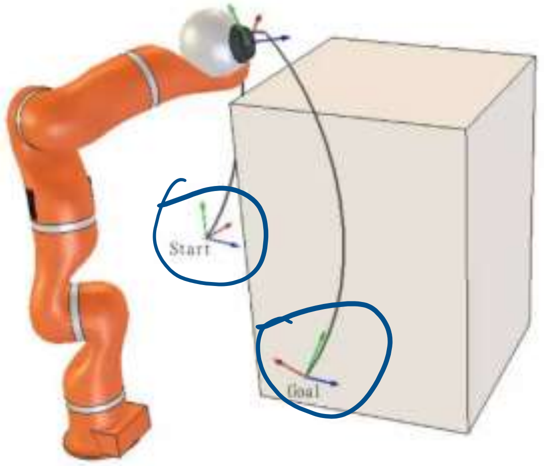
- **What is planning? (~10)**
- **Motion planning in robotic application （~10）**
  - **Self-driving, drone, robot arm, humanoids, medical robots, soft robots …**
- **Formulation of robot motion planning**
- **Planning as searching (~15)**
- **Planning as sampling (~25)**
  - **PRM, RRT, RRT***



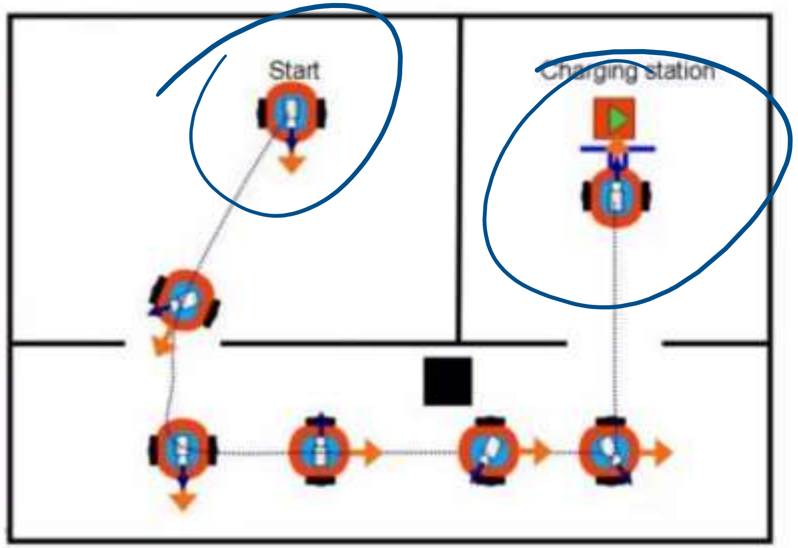
Robot Motion Planning

- ➢ **Motion planning is a term used in robotics for the process of breaking down a desired movement task into discrete motions that satisfy movement constraints and possibly optimize some aspect of the movement.**


A robot has to compute a **collision-free path** from a start position (s) to a given goal position (G), amidst a collection of obstacles.



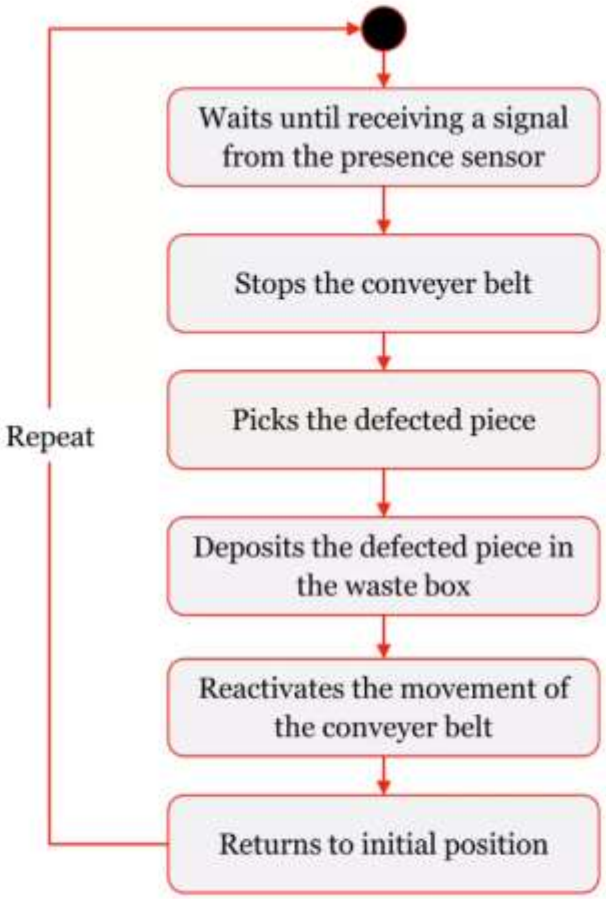
Articulated Robot



Rigid Robot

Robot Motion Planning

Structured environment



Waits until receiving a signal from the presence sensor

Stops the conveyer belt

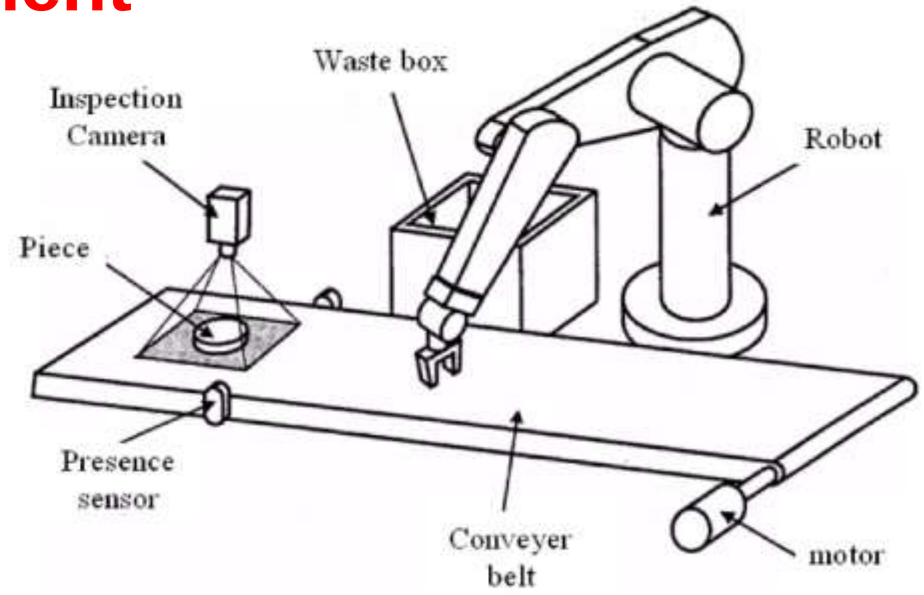
Picks the defected piece

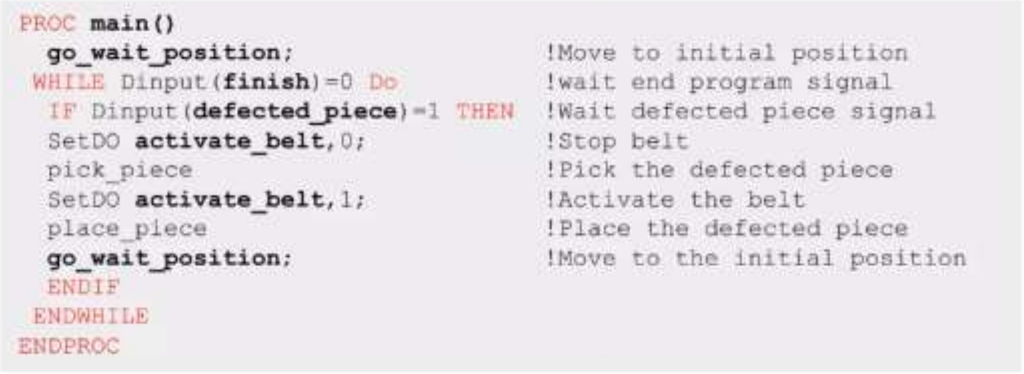
Deposits the defected piece in the waste box

Reactivates the movement of the conveyer belt

Returns to initial position

Repeat



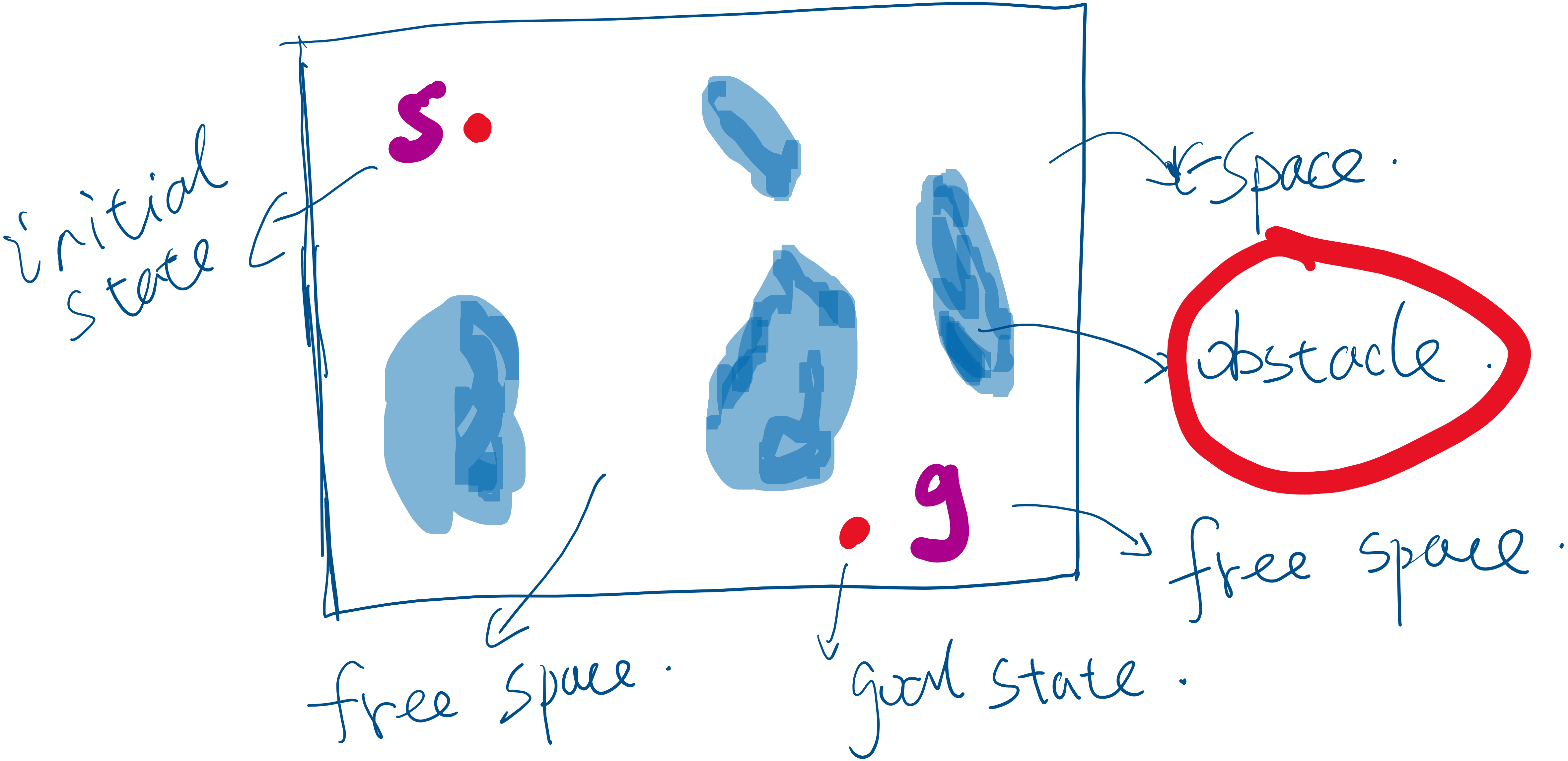


```
PROC main()
  go_wait_position;                                 !Move to initial position
 WHILE Dinput(finish)=0 Do                          !wait end program signal
  IF Dinput(defected_piece)=1 THEN                  !Wait defected piece signal
  SetDO activate_belt,0;                            !Stop belt
  pick_piece                                        !Pick the defected piece
  SetDO activate_belt,1;                            !Activate the belt
  place_piece                                       !Place the defected piece
  go_wait_position;                                 !Move to the initial position
  ENDIF
 ENDWHILE
ENDPROC
```

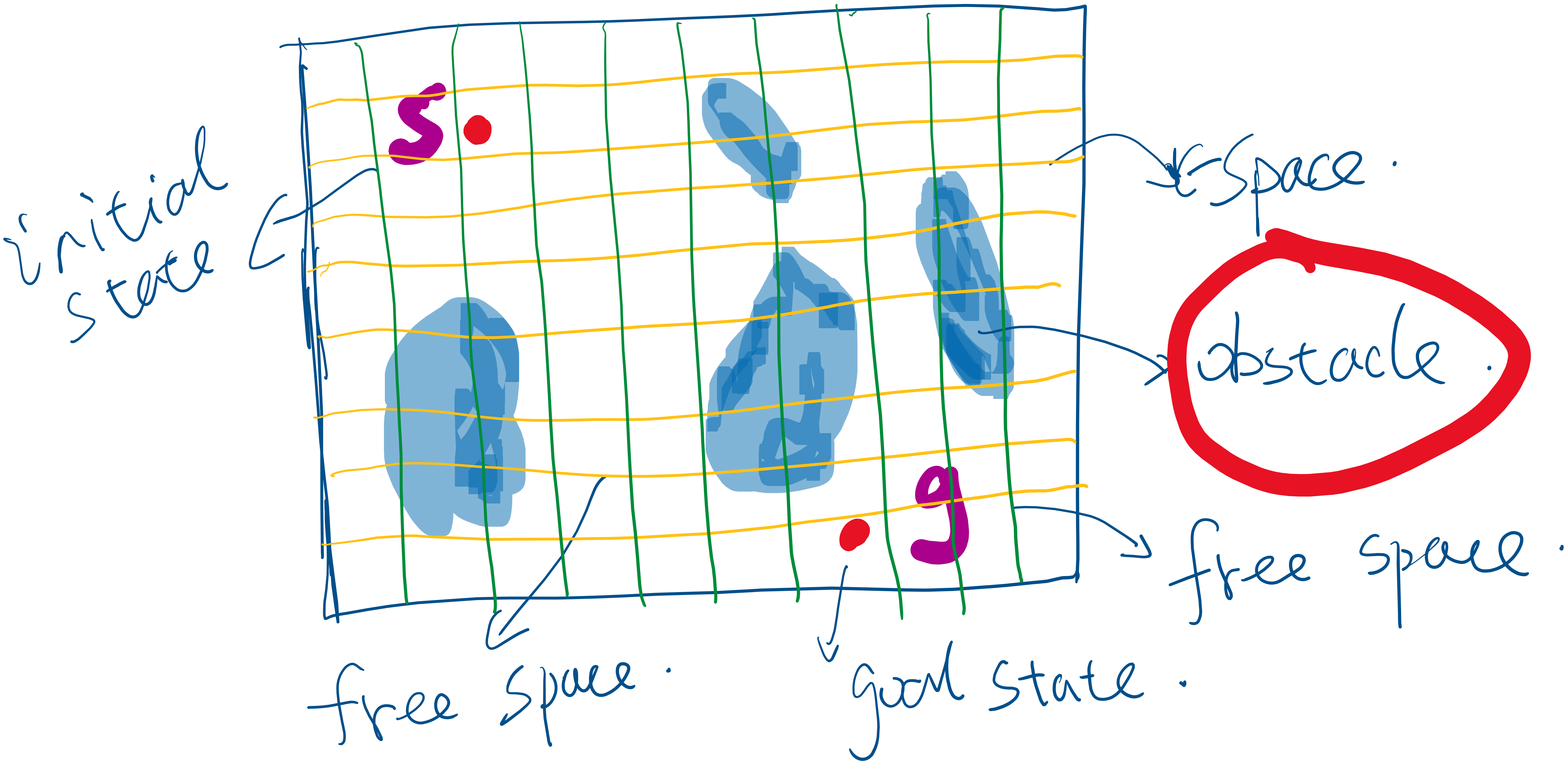
Plan: Activity Diagram

ABB RAPID Program

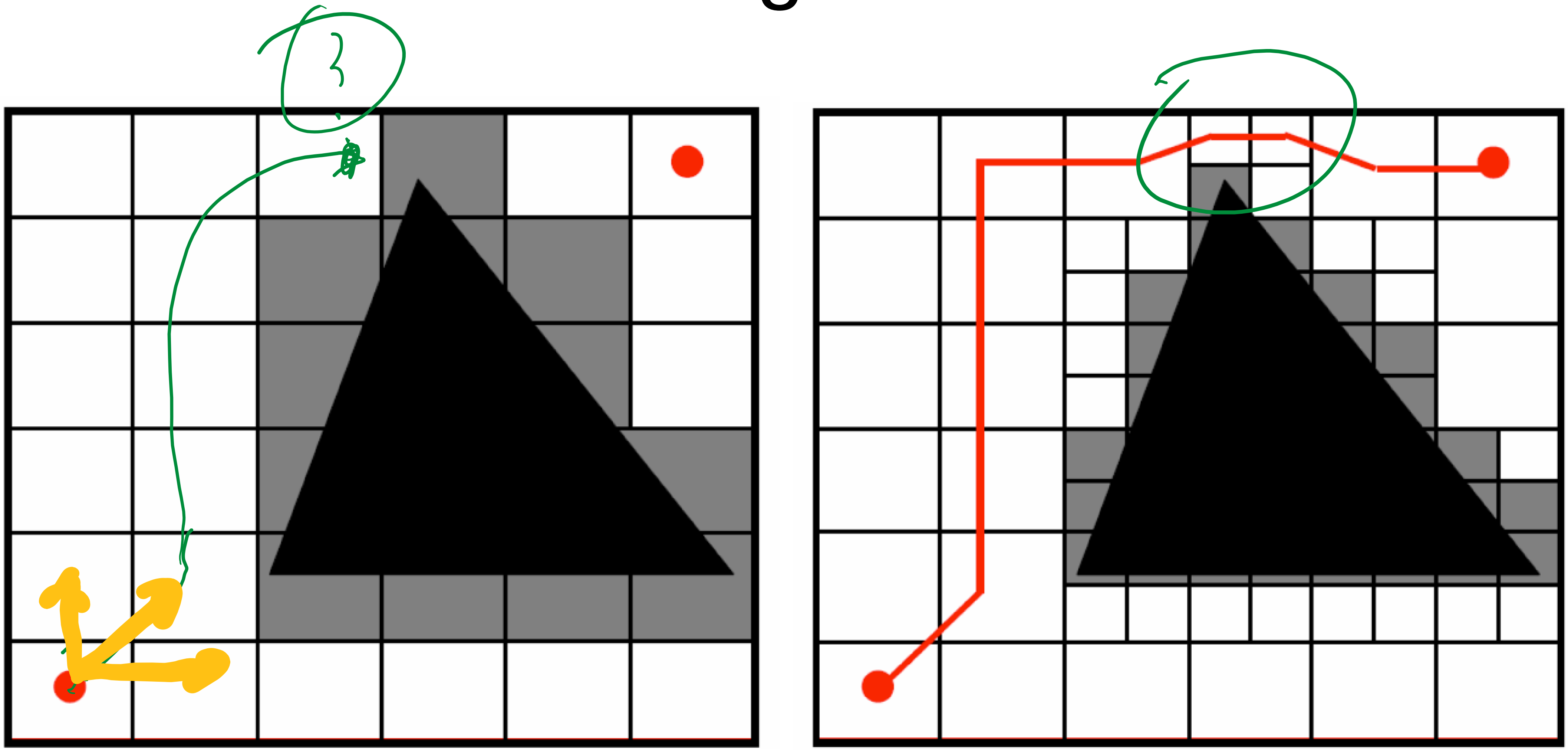
Motion Planning in 2D



Motion Planning in Grid World



Motion Planning in Grid World



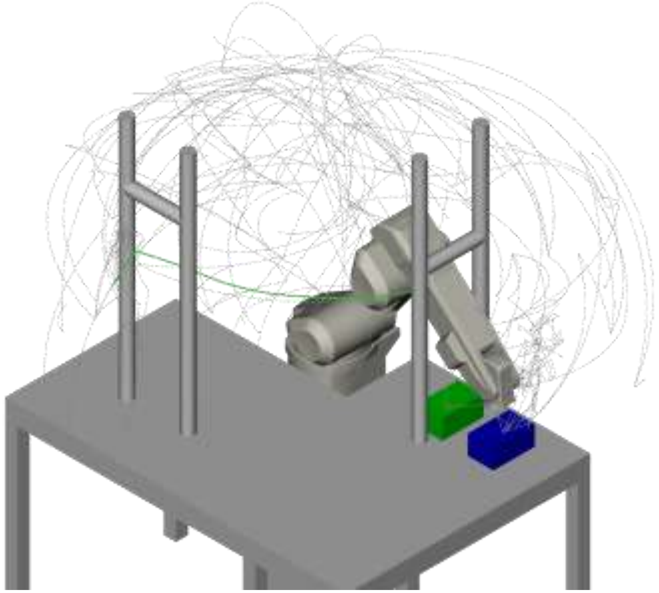
Try to find a path using the current set of cells

If no path found:

- Subdivide the *MIXED* cells and try again with the new set of cells

Today's Agenda

- **What is planning? (~10)**
- **Motion planning in robotic application （~10）**
  - **Self-driving, drone, robot arm, humanoids, medical robots, soft robots …**
- **Formulation of robot motion planning**
- **Planning as searching (~25)**
- **Planning as sampling (~25)**
  - **PRM, RRT, RRT***



Discrete Planning

Blind.

- **Breadth-First Search**
- **Depth-First Search**
- **Brute-Force Search**

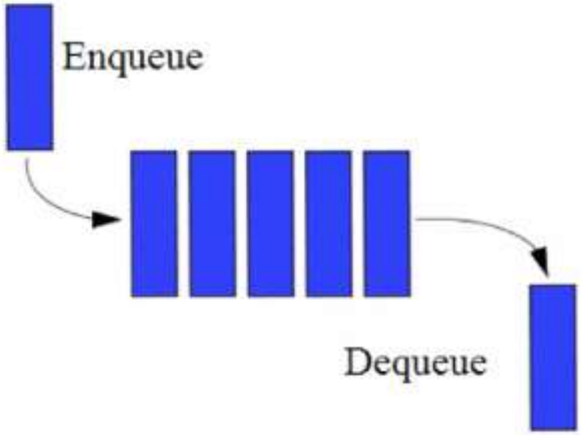
Informed

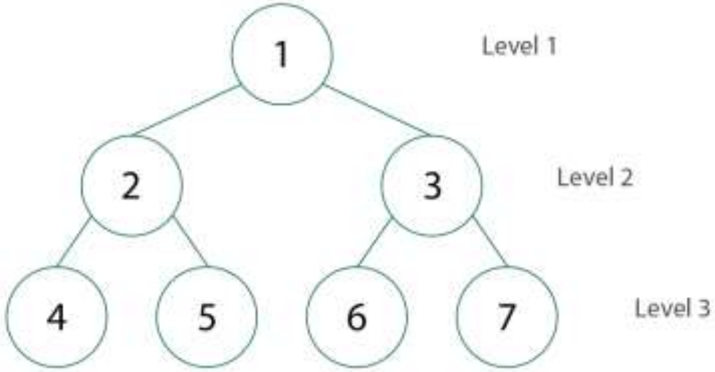
- **Best-First**
- **A***

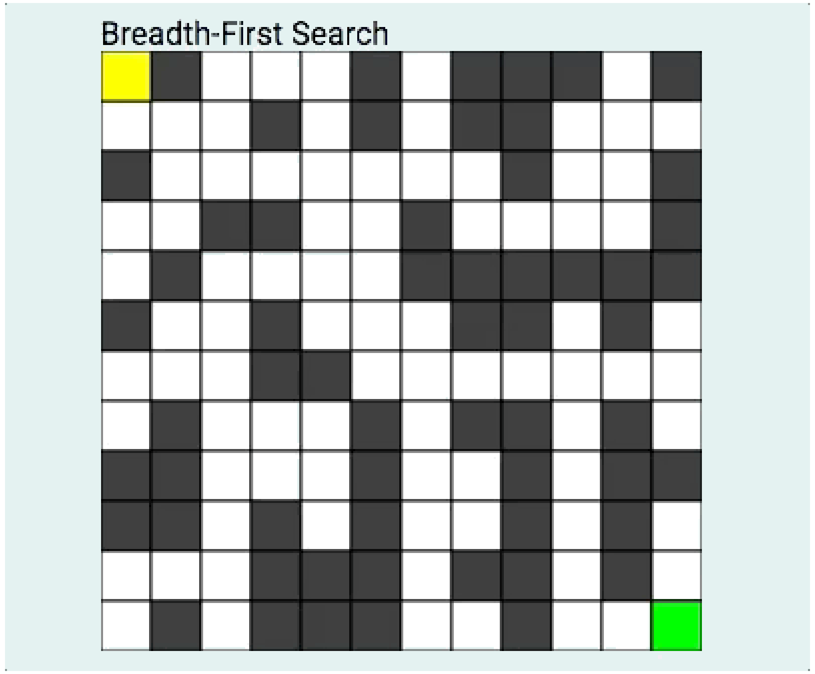
https://medium.com/omarelgabrys-blog/path-finding-algorithms-f65a8902eb40

Demo: http://qiao.github.io/PathFinding.js/visual/

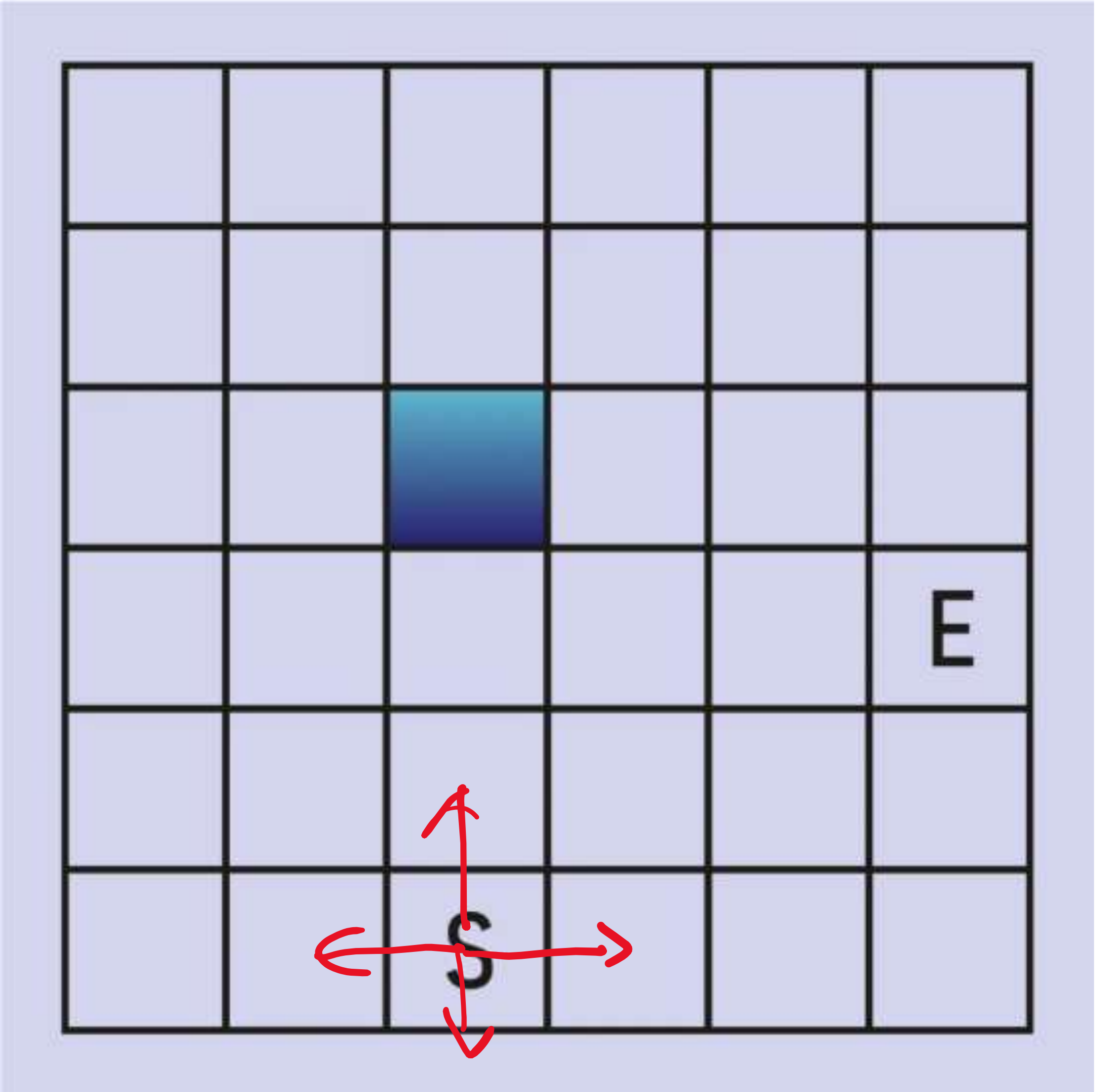
Breadth-First Search

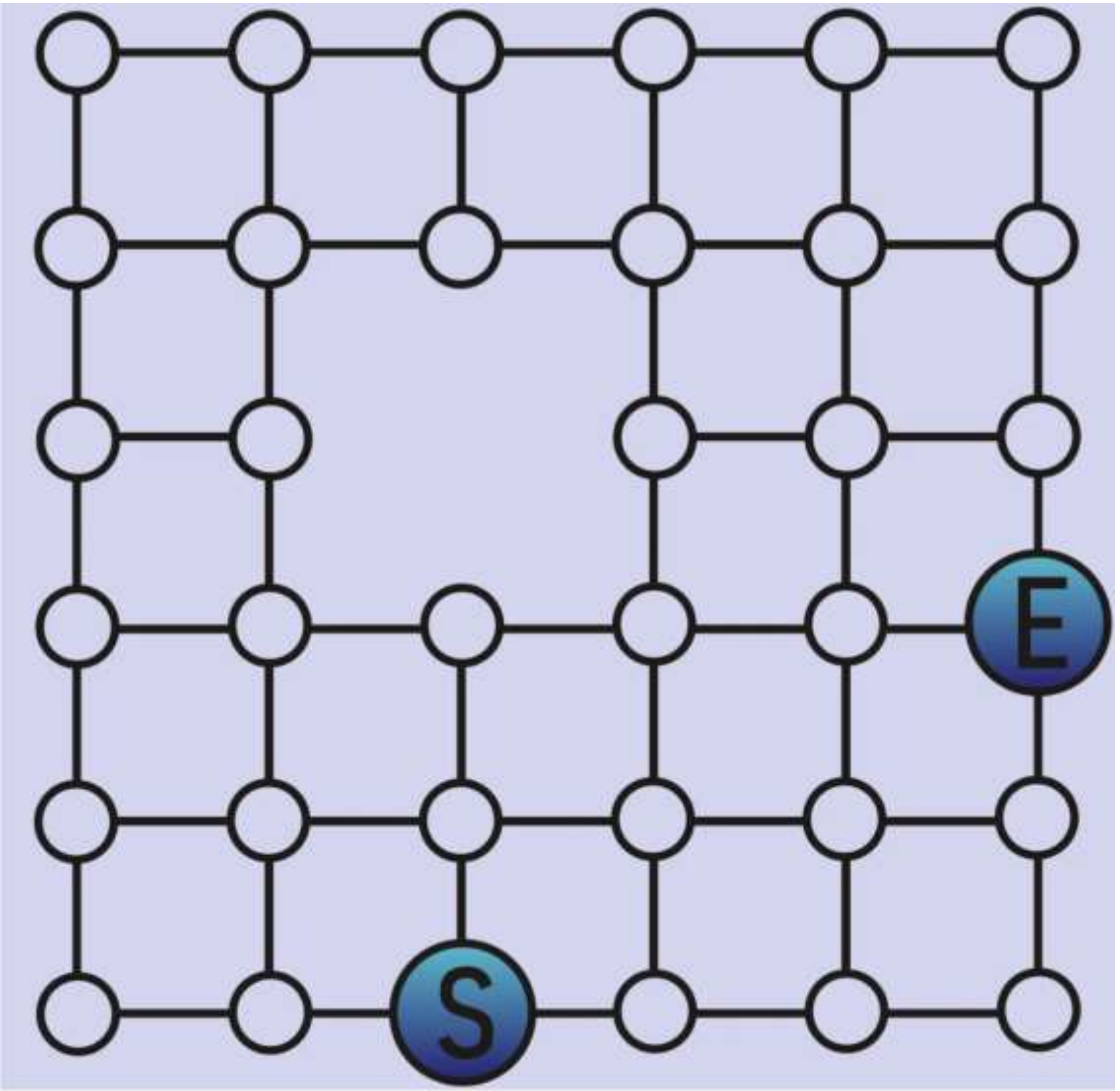




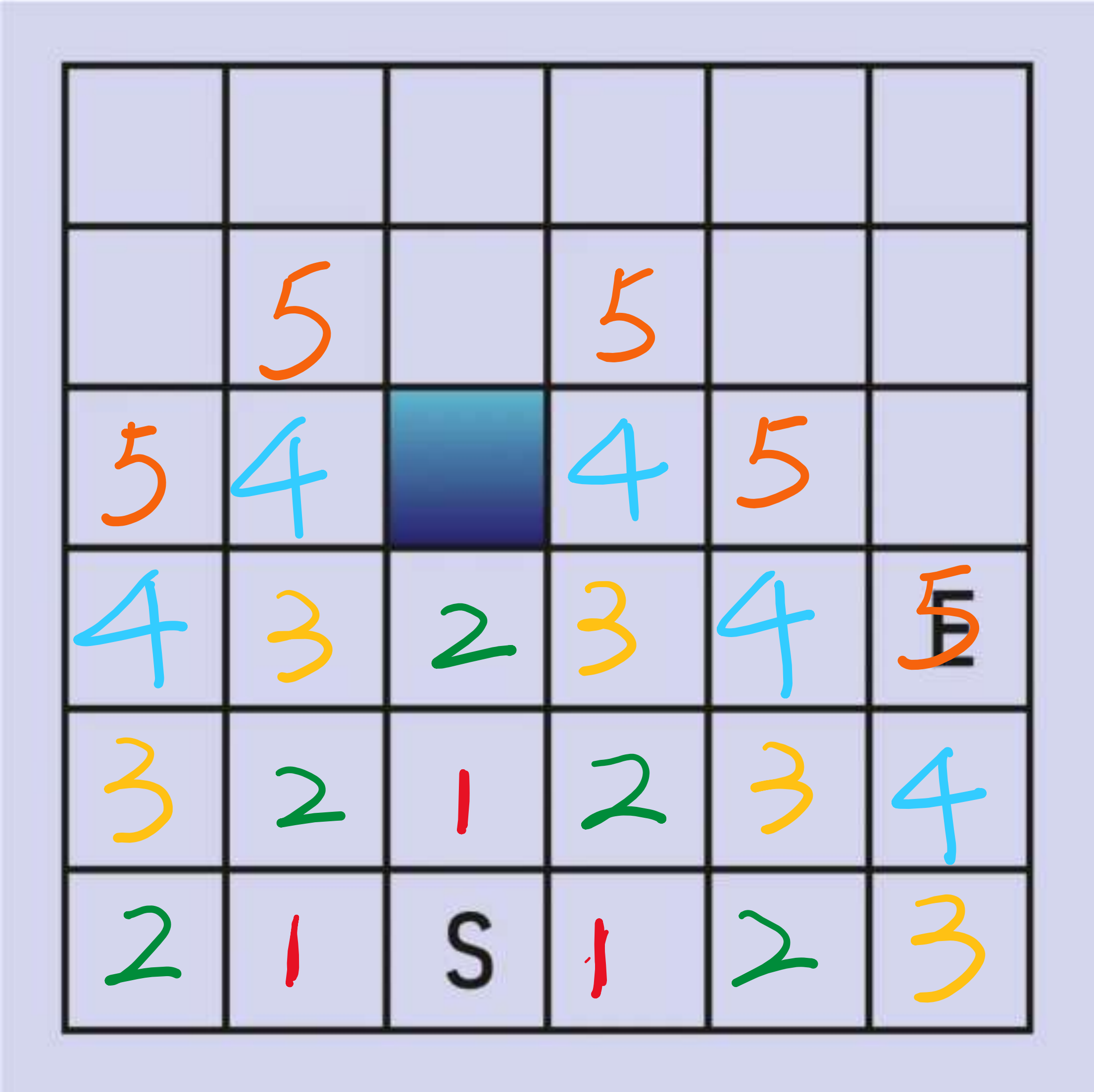


Breadth-First Search

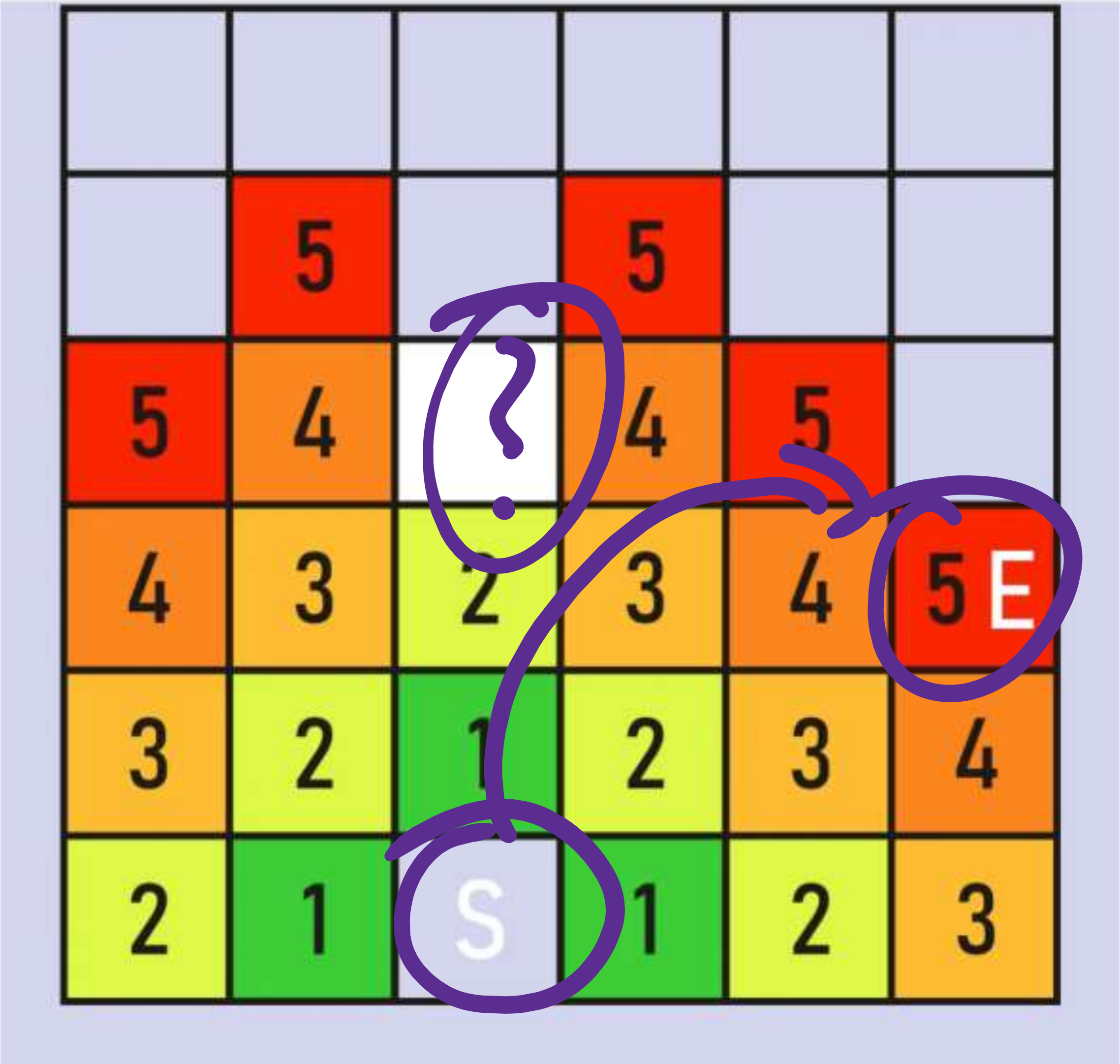




Breadth-First Search

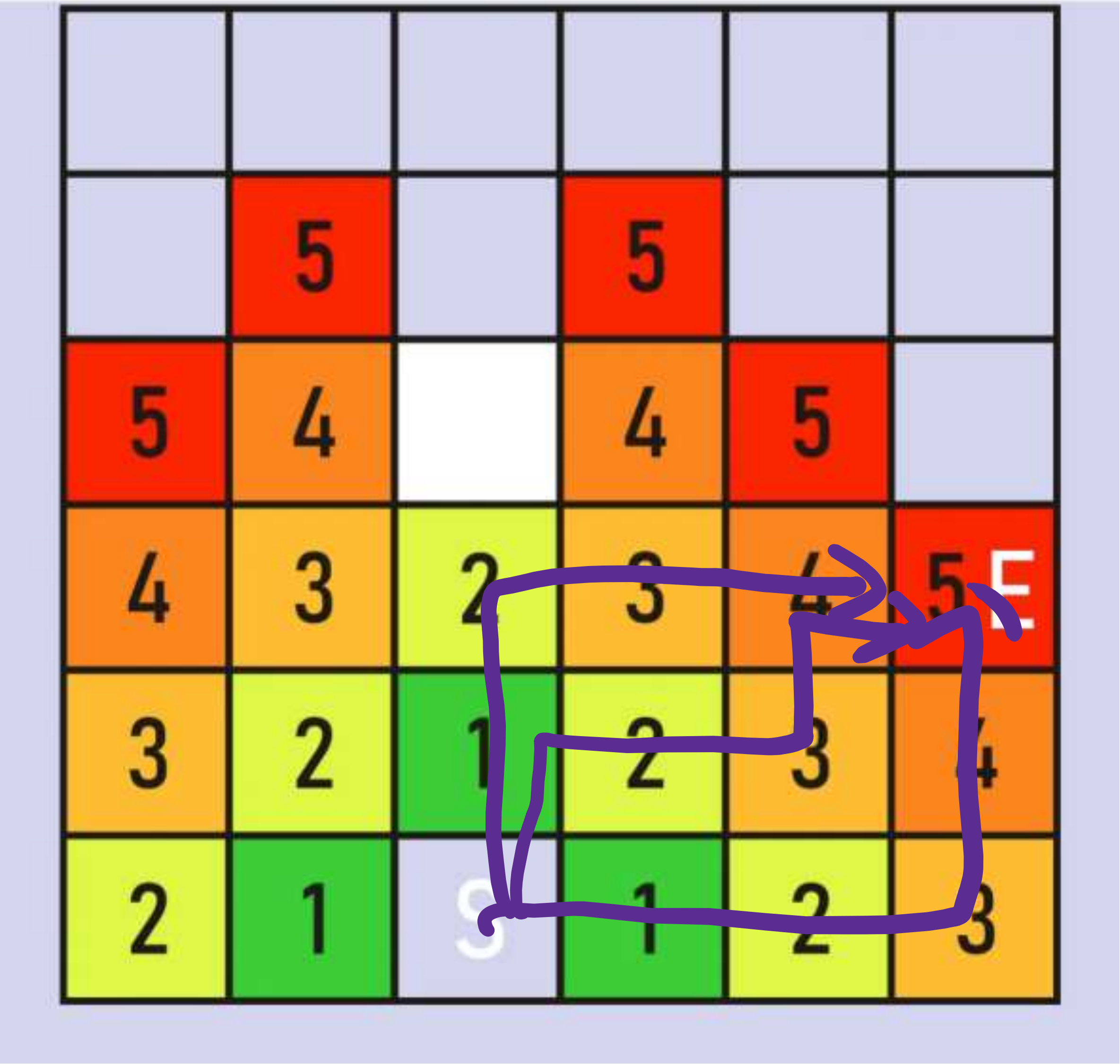


	5		5		
5	4	■	4	5	
4	3	2	3	4	E 5
3	2	1	2	3	4
2	1	S	1	2	3



	5		5		
5	4	?	4	5	
4	3	2	3	4	5 E
3	2	1	2	3	4
2	1	S	1	2	3

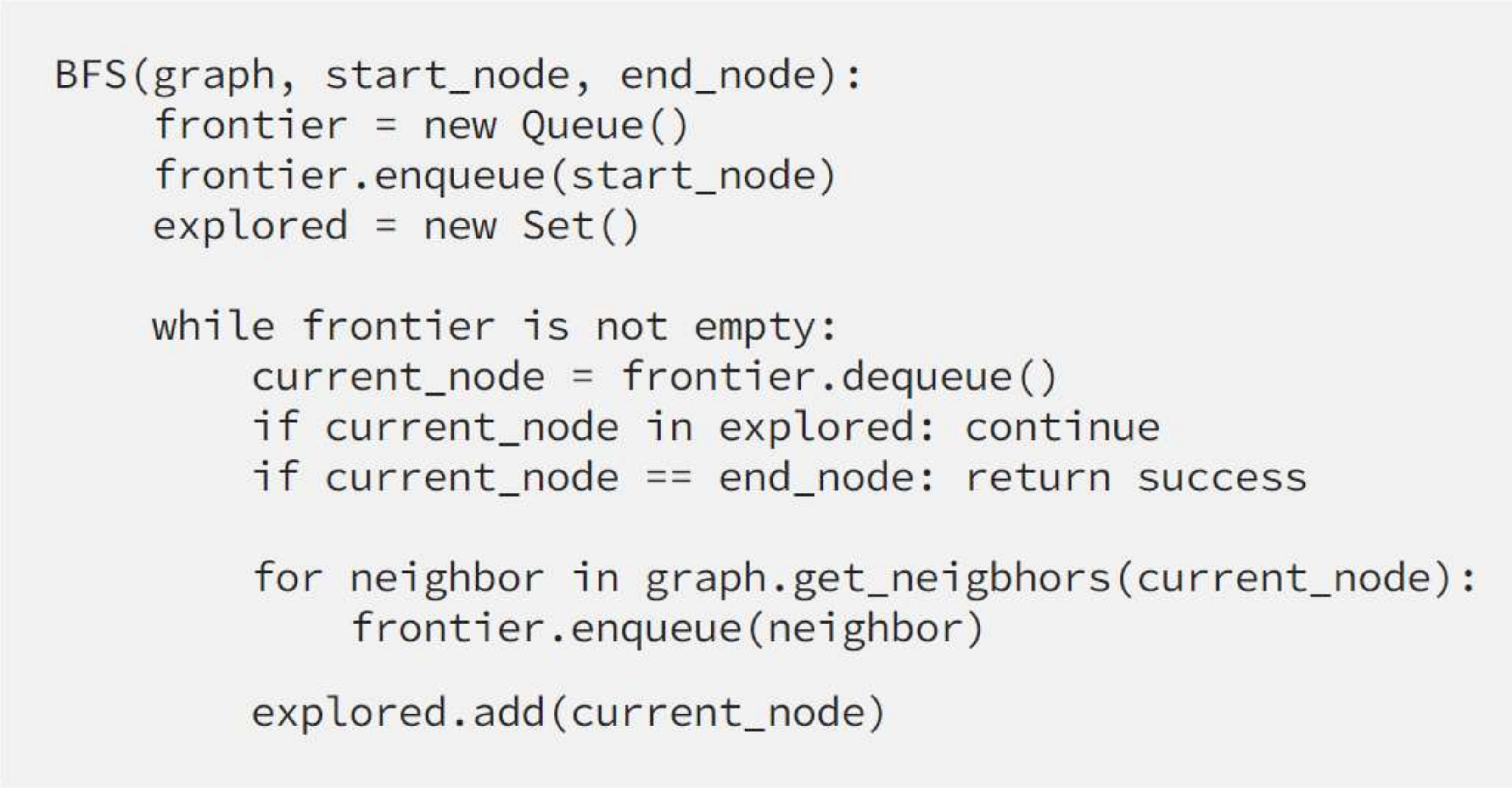
Breadth-First Search



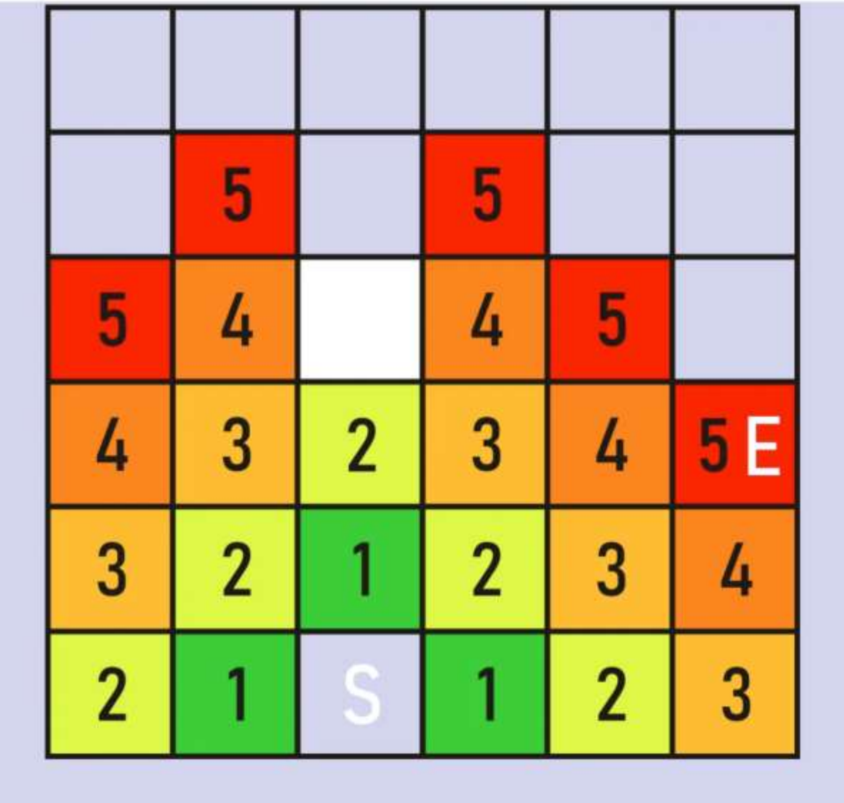
	5		5		
5	4		4	5	
4	3	2	3	4	5 E
3	2	1	2	3	4
2	1	S	1	2	3

⑦

Breadth-First Search



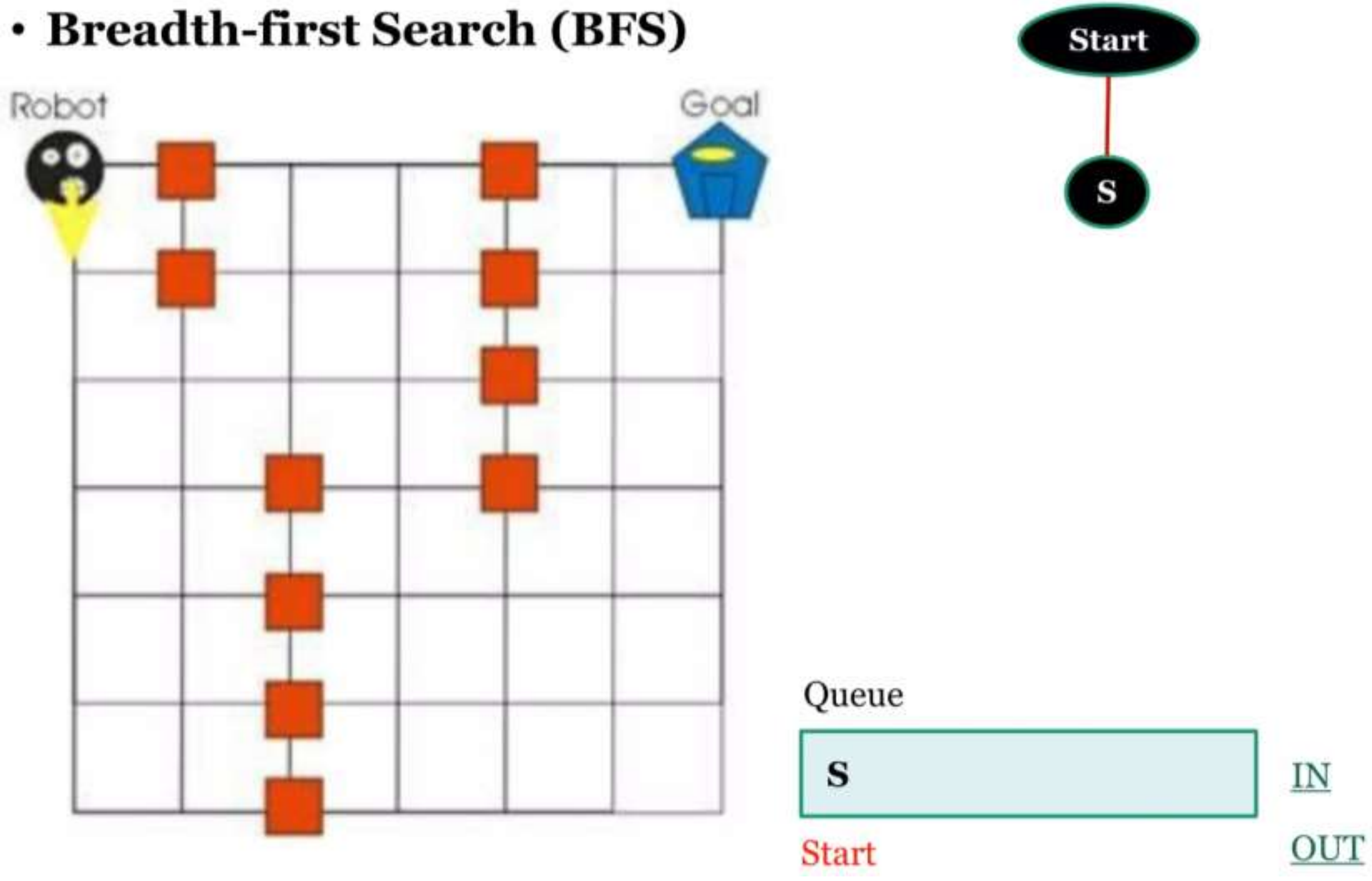
```
BFS(graph, start_node, end_node):
    frontier = new Queue()
    frontier.enqueue(start_node)
    explored = new Set()

    while frontier is not empty:
        current_node = frontier.dequeue()
        if current_node in explored: continue
        if current_node == end_node: return success

        for neighbor in graph.get_neigbhors(current_node):
            frontier.enqueue(neighbor)

        explored.add(current_node)
```



	5		5		
5	4		4	5	
4	3	2	3	4	5 E
3	2	1	2	3	4
2	1	S	1	2	3

https://medium.com/tebs-lab/breadth-first-search-and-depth-first-search-4310f3bf8416

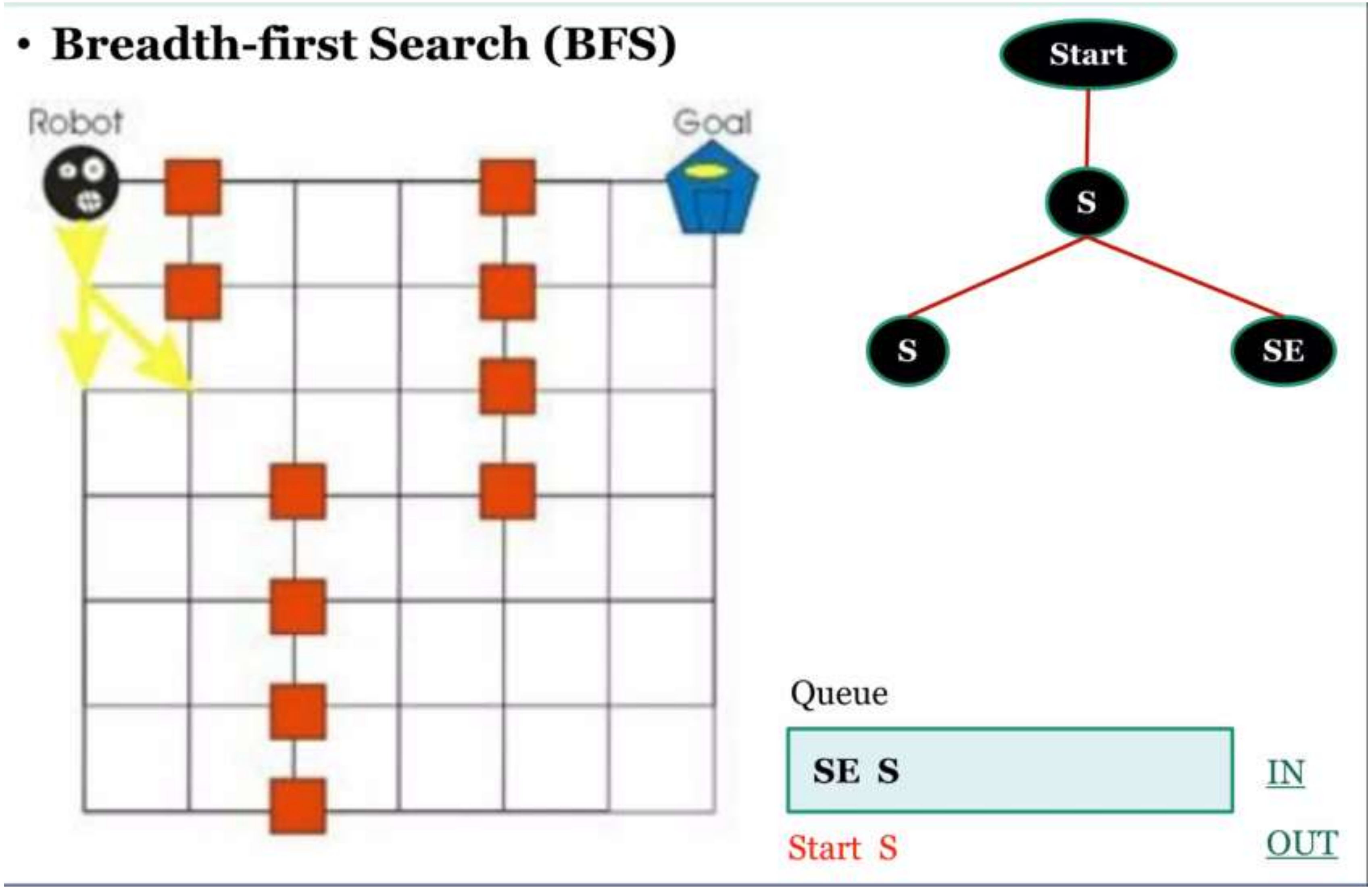
Breadth-First Search



- **Breadth-first Search (BFS)**

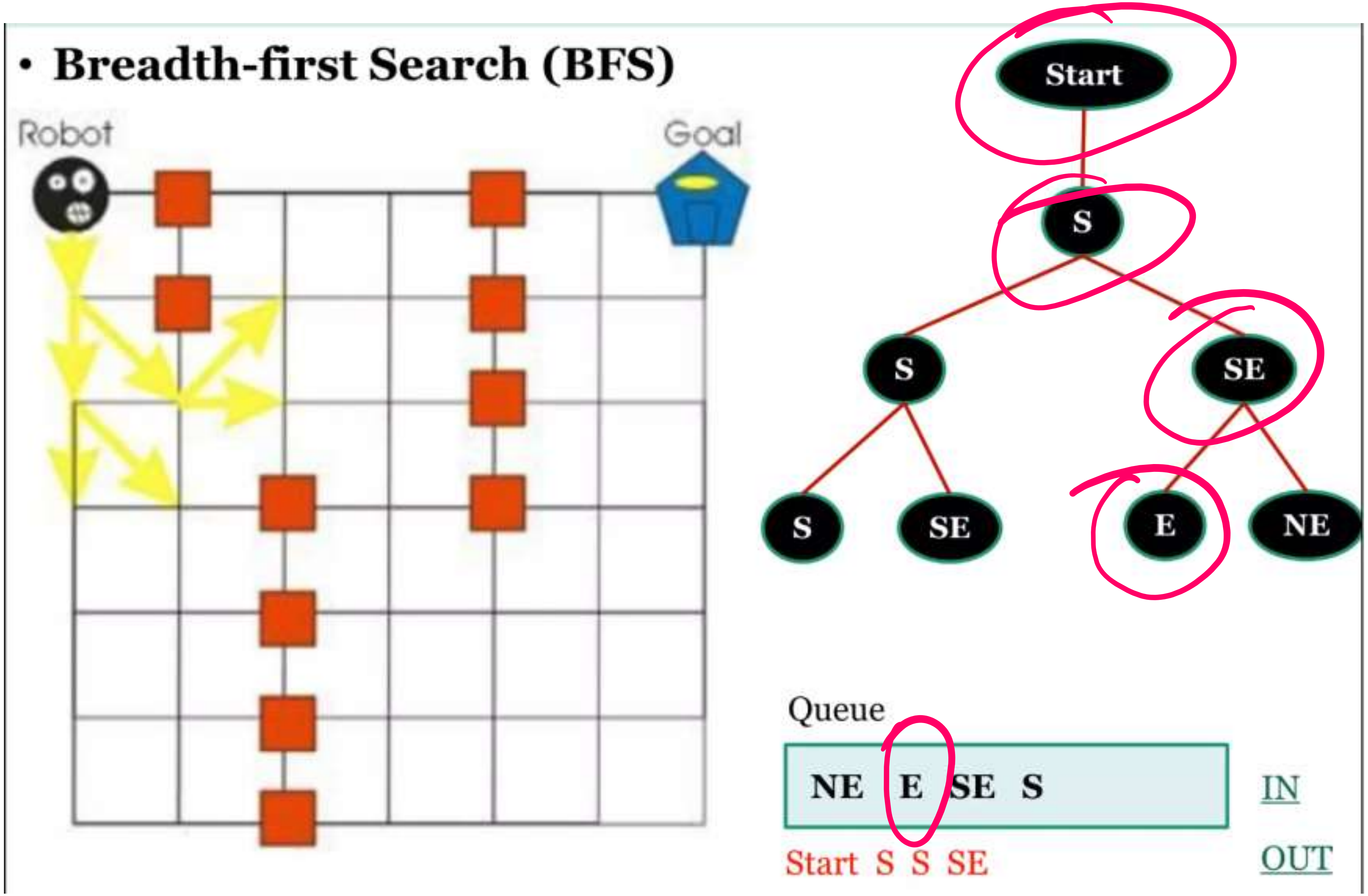
Example from the slides: https://www.slideshare.net/AlaaKhamis/motion-planning

Breadth-First Search

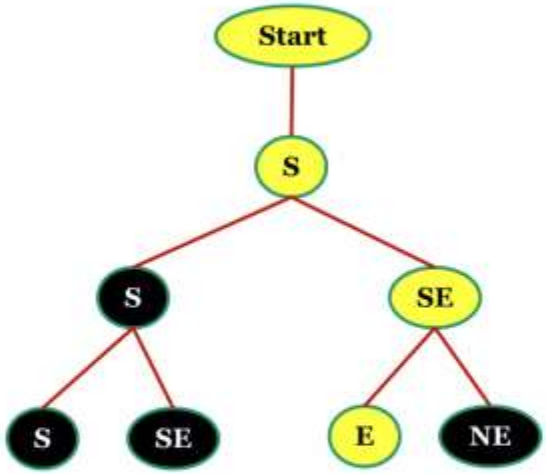


- **Breadth-first Search (BFS)**

Breadth-First Search



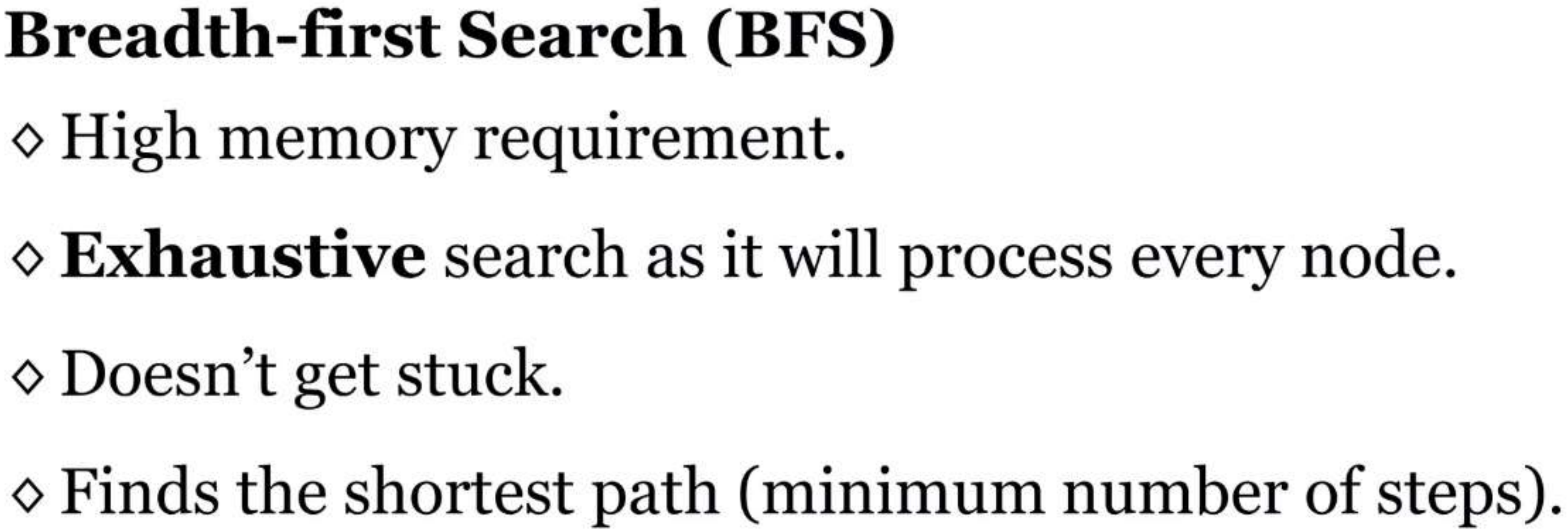
- **Breadth-first Search (BFS)**



Assume that **E** is the goal,

Path is: **Start→ S→SE→E**

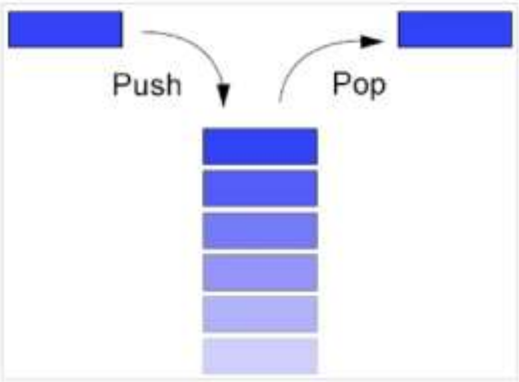
Breadth-First Search



Breadth-first Search (BFS)

- High memory requirement.
- **Exhaustive** search as it will process every node.
- Doesn’t get stuck.
- Finds the shortest path (minimum number of steps).

Depth-First Search

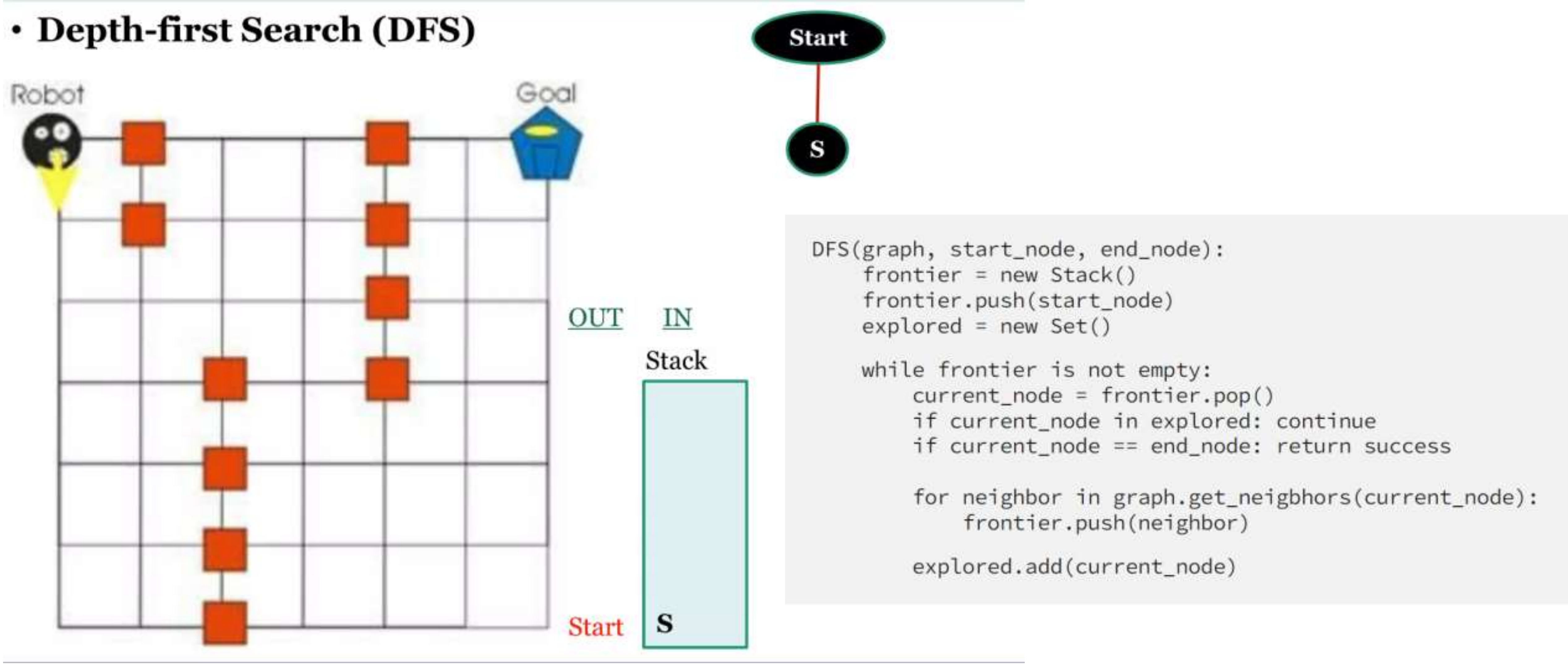


Stack (LIFO)

http://wiki.ugcnet4cs.in/t/Stack

Depth first search is dead simple. First, go to the specified start node. Now, arbitrarily pick one of that node’s neighbors and go there. If that node has neighbors, arbitrarily pick one of those and go there *unless we’ve already seen that node*. And we just repeat this process until one of two things happens. If reach the specified end node we terminate the algorithm and report success. If we reach a node with only neighbors we’ve already seen, or no neighbors at all, we go back one step and try one of the neighbors we *didn’t* try last time.

Depth-First Search

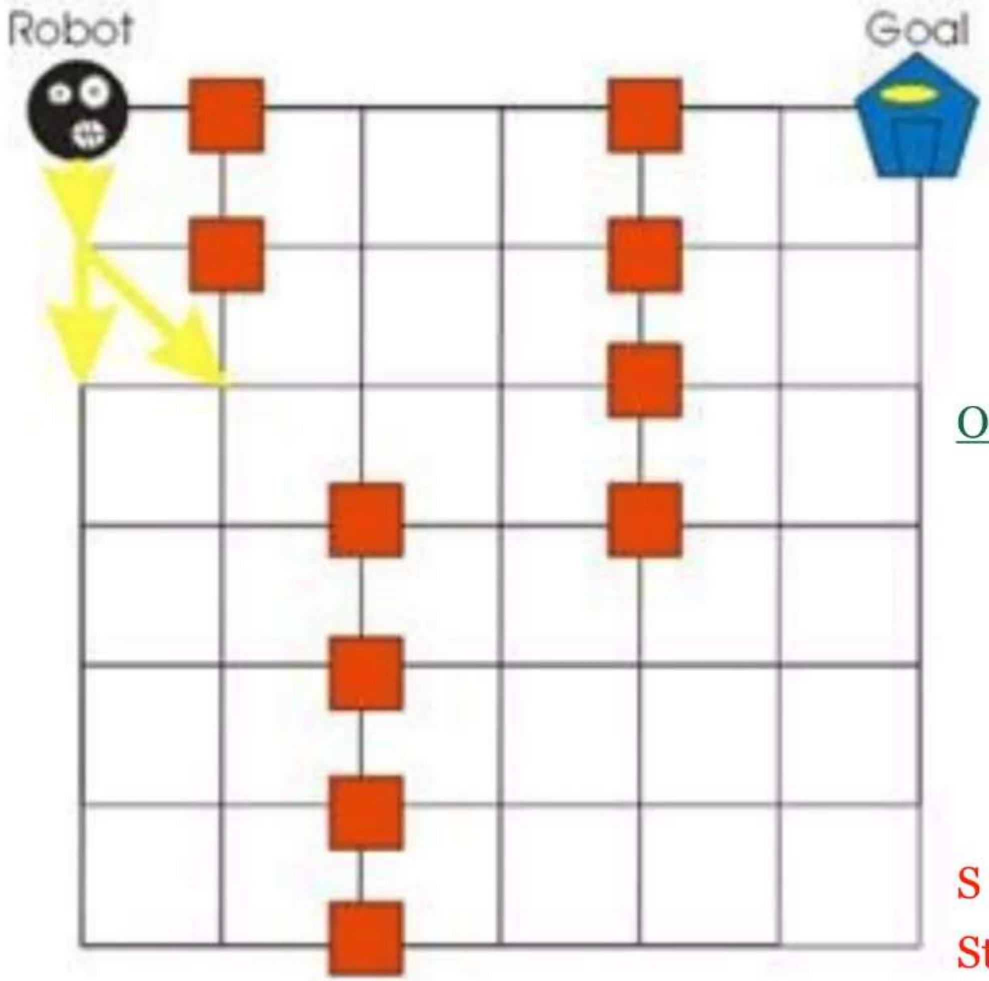


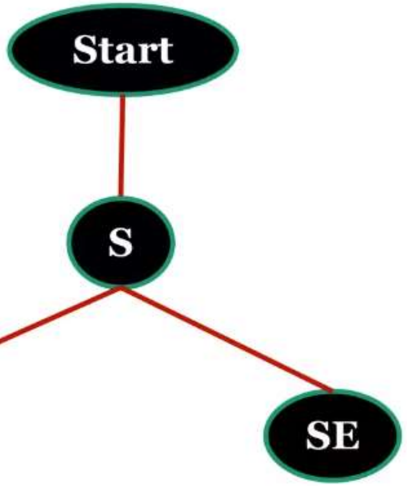
- **Depth-first Search (DFS)**

```
DFS(graph, start_node, end_node):
    frontier = new Stack()
    frontier.push(start_node)
    explored = new Set()

    while frontier is not empty:
        current_node = frontier.pop()
        if current_node in explored: continue
        if current_node == end_node: return success

        for neighbor in graph.get_neigbhors(current_node):
            frontier.push(neighbor)

        explored.add(current_node)
```

Depth-First Search

- **Depth-first Search (DFS)**



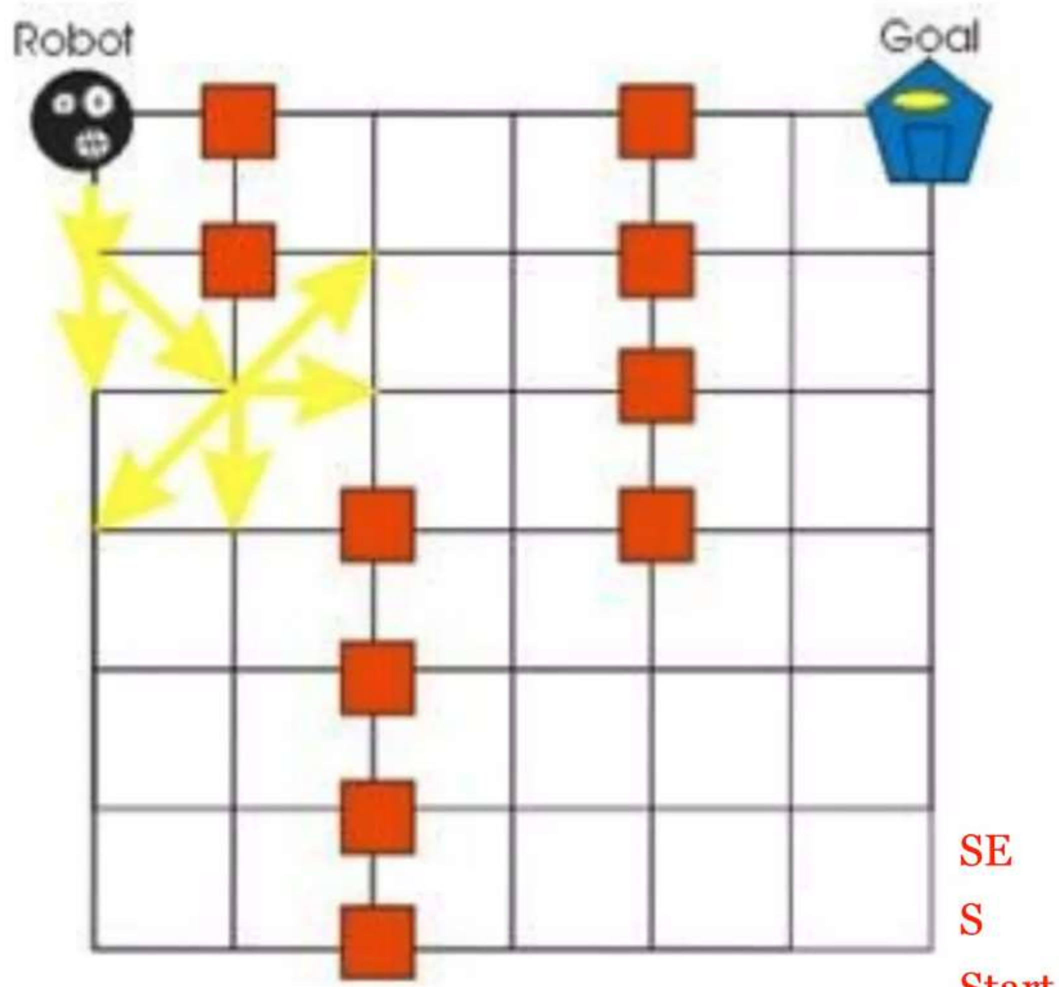


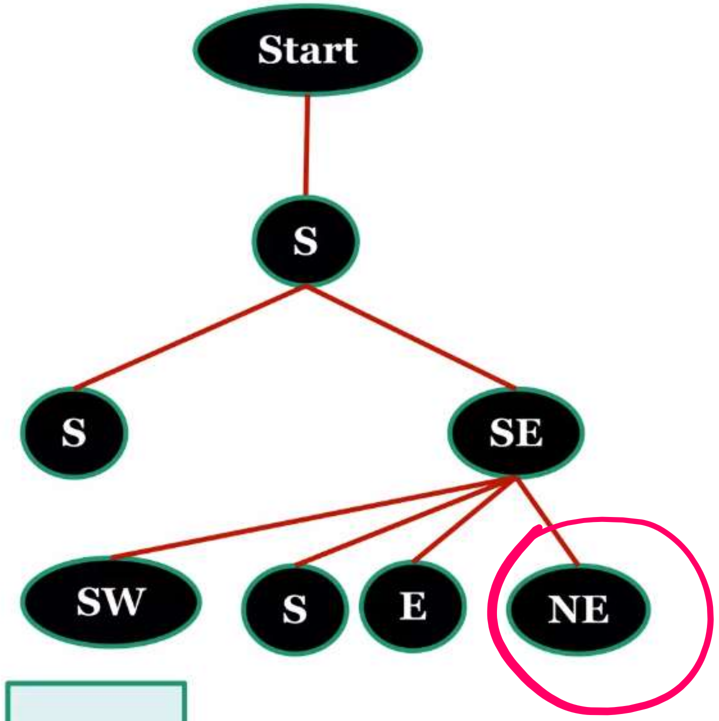
Stack

OUT	IN
S	**SE**
Start	**S**

Depth-First Search

- **Depth-first Search (DFS)**

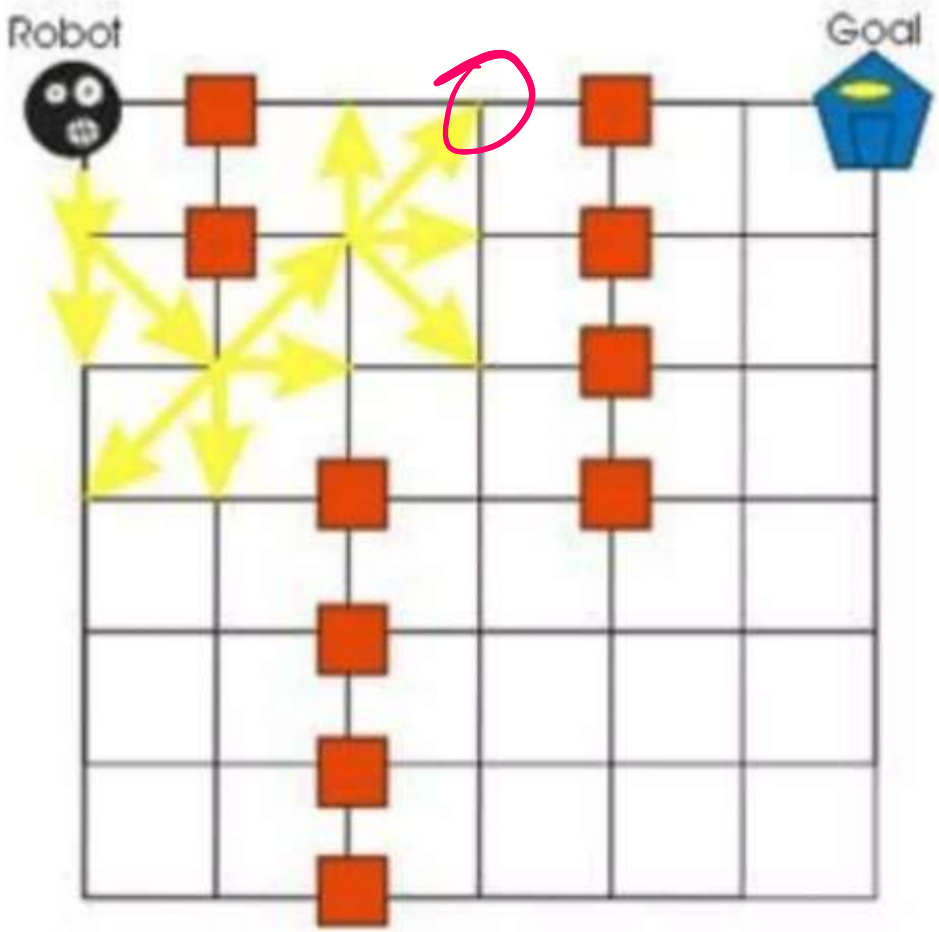




	NE
	E
SE	**S**
S	**SW**
Start	**S**

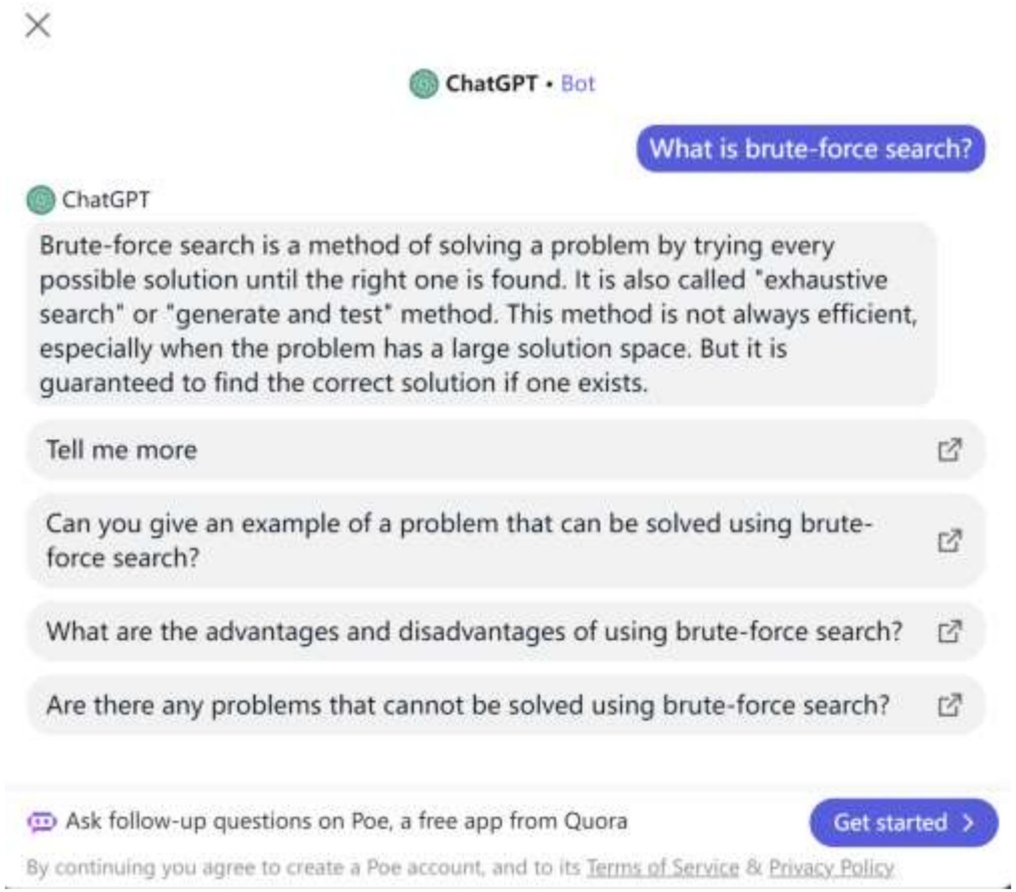
Depth-First Search

- **Depth-first Search (DFS)**



- ◇ The next node to be expanded would be **NE** and its successors would be added to the stack and this loop continues until the goal is found.
- ◇ Once the goal is found, you can then **trace back** through the tree to obtain the path for the robot to follow.

Brute-Force Search



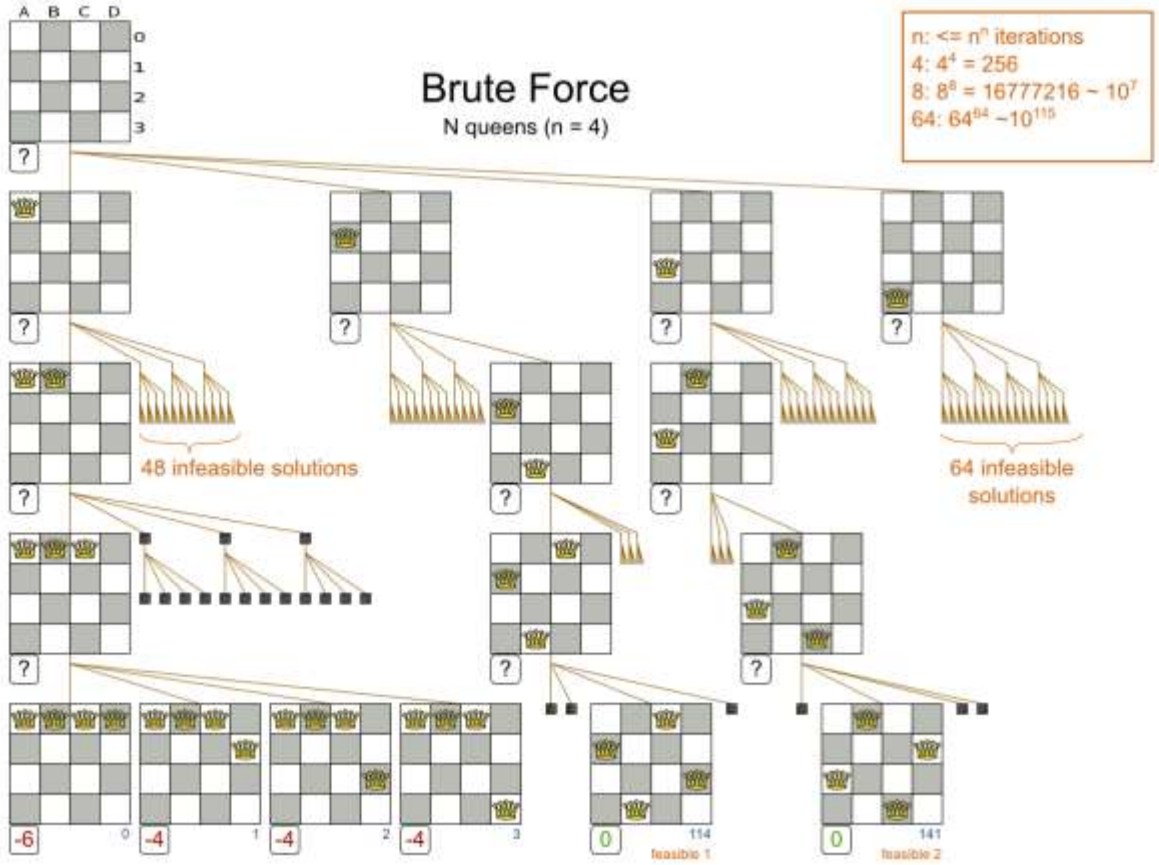
What is brute-force search?

ChatGPT

Brute-force search is a method of solving a problem by trying every possible solution until the right one is found. It is also called "exhaustive search" or "generate and test" method. This method is not always efficient, especially when the problem has a large solution space. But it is guaranteed to find the correct solution if one exists.

- Tell me more
- Can you give an example of a problem that can be solved using brute-force search?
- What are the advantages and disadvantages of using brute-force search?
- Are there any problems that cannot be solved using brute-force search?

By continuing you agree to create a Poe account, and to its Terms of Service & Privacy Policy



Discrete Planning

Blind.

- **Breadth-First Search**
- **Depth-First Search**
- **Brute-Force Search**

Only use the starting nodes

Informed

- **Best-First**
- **A***

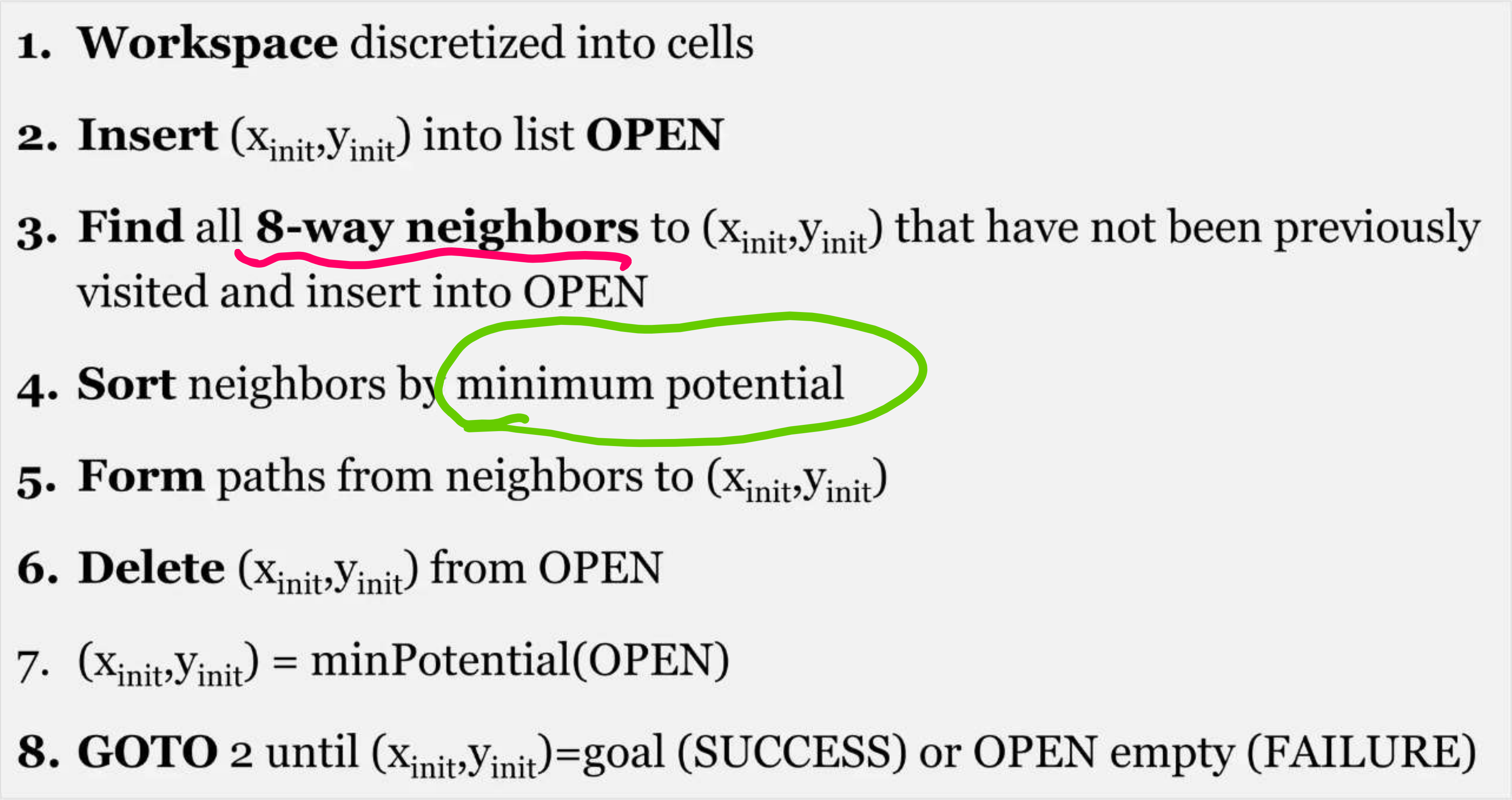
goal?

https://medium.com/omarelgabrys-blog/path-finding-algorithms-f65a8902eb40

http://qiao.github.io/PathFinding.js/visual/

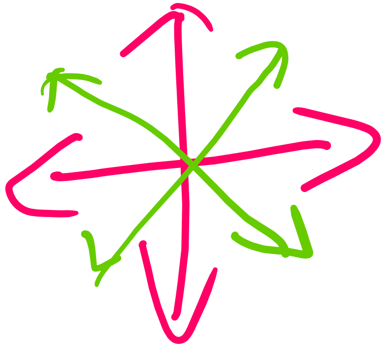
Best-First Search

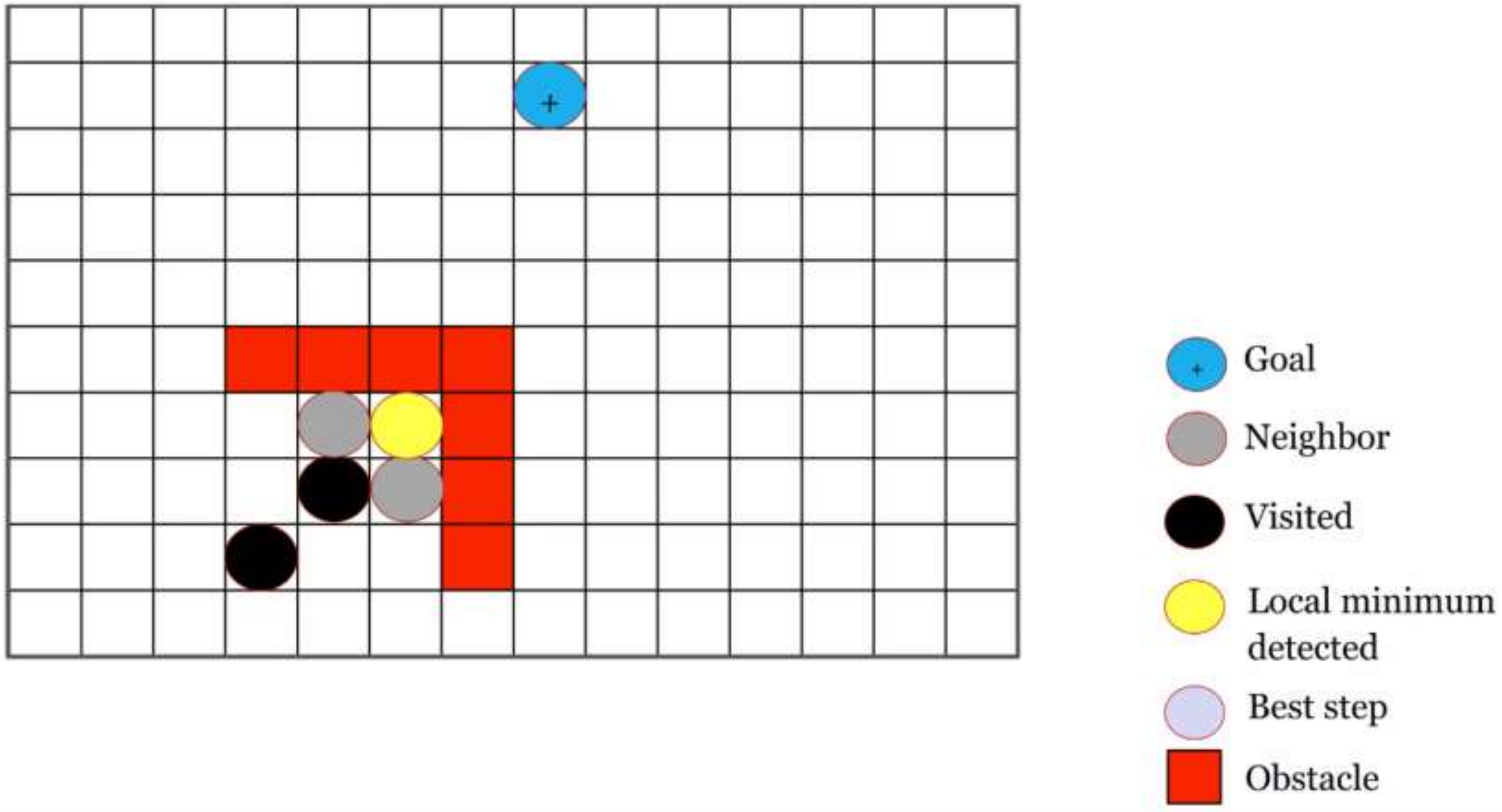
- **Best-first**



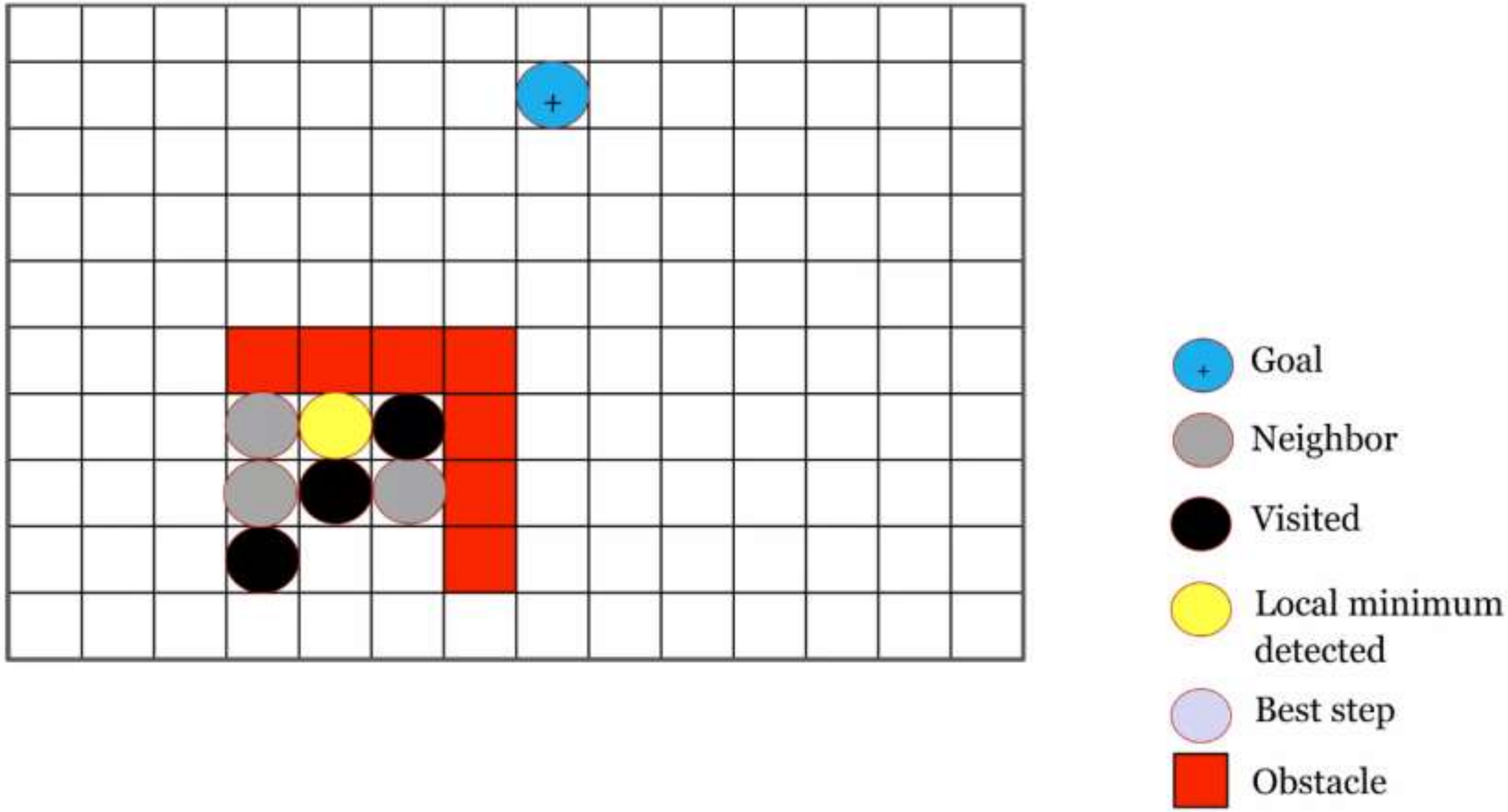
1. **Workspace** discretized into cells
2. **Insert** (x_{init}, y_{init}) into list **OPEN**
3. **Find** all **8-way neighbors** to (x_{init}, y_{init}) that have not been previously visited and insert into OPEN
4. **Sort** neighbors by minimum potential
5. **Form** paths from neighbors to (x_{init}, y_{init})
6. **Delete** (x_{init}, y_{init}) from OPEN
7. (x_{init}, y_{init}) = minPotential(OPEN)
8. **GOTO** 2 until (x_{init}, y_{init})=goal (SUCCESS) or OPEN empty (FAILURE)

Best-First Search

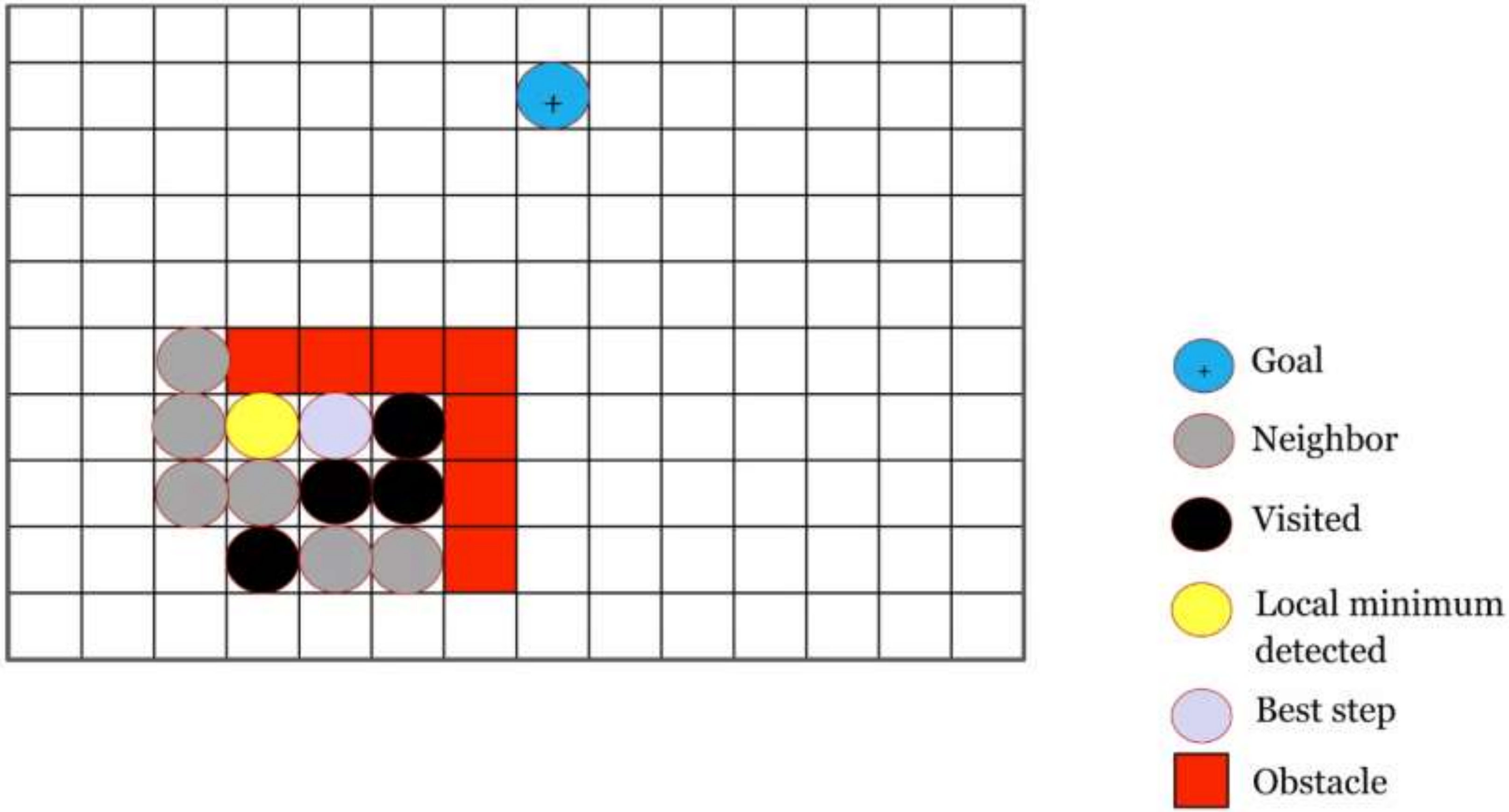




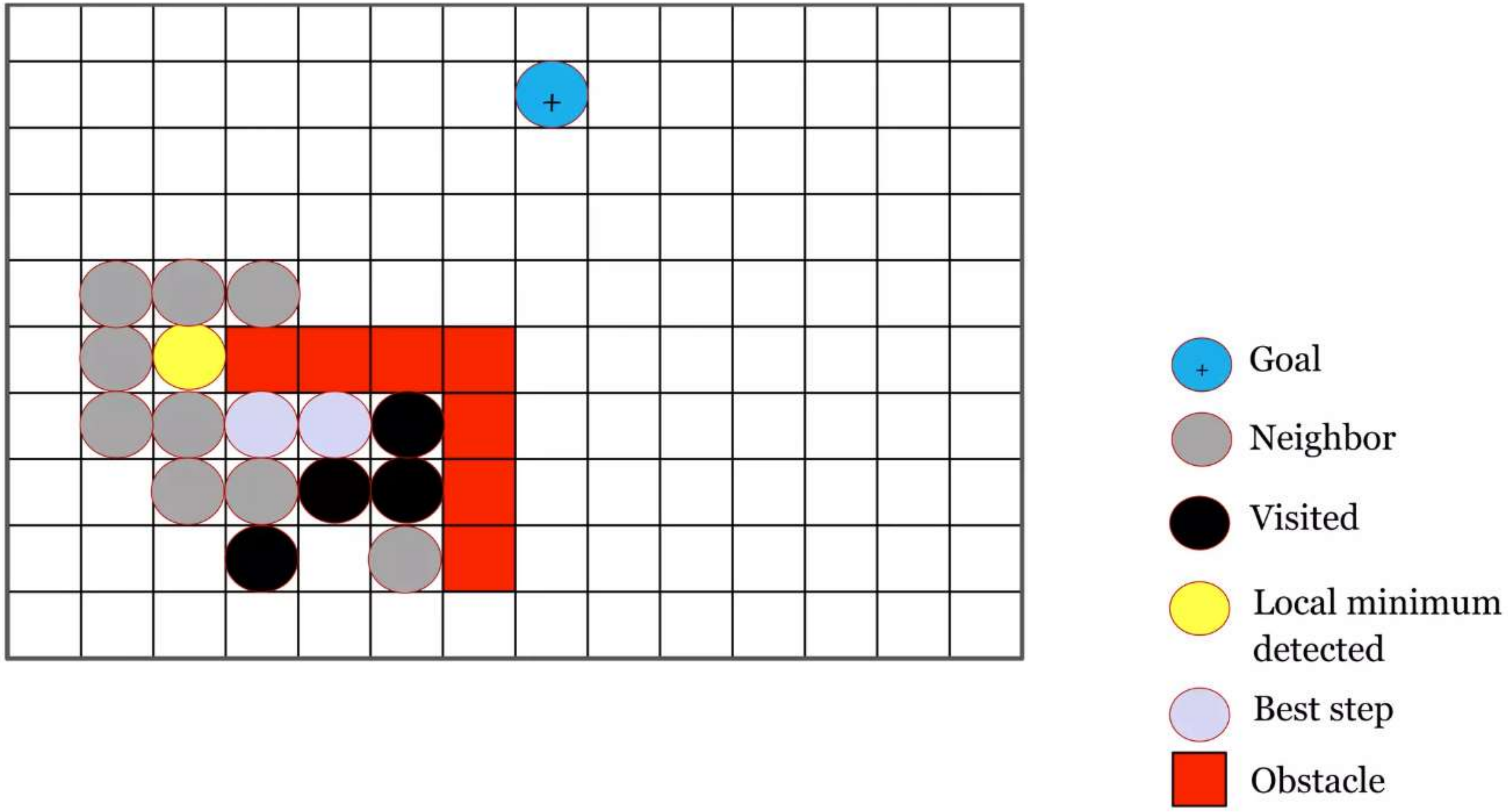
Best-First Search



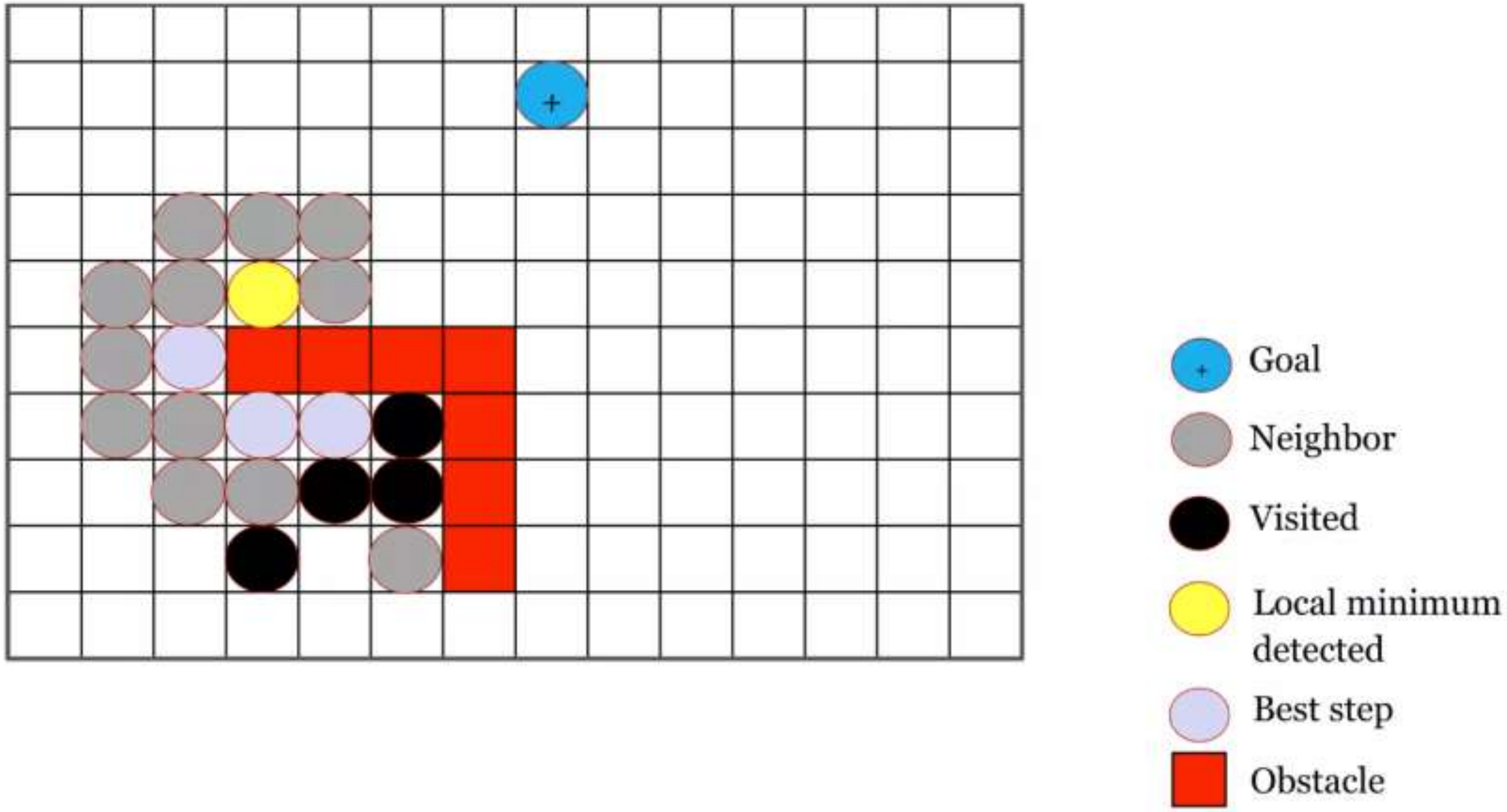
Best-First Search



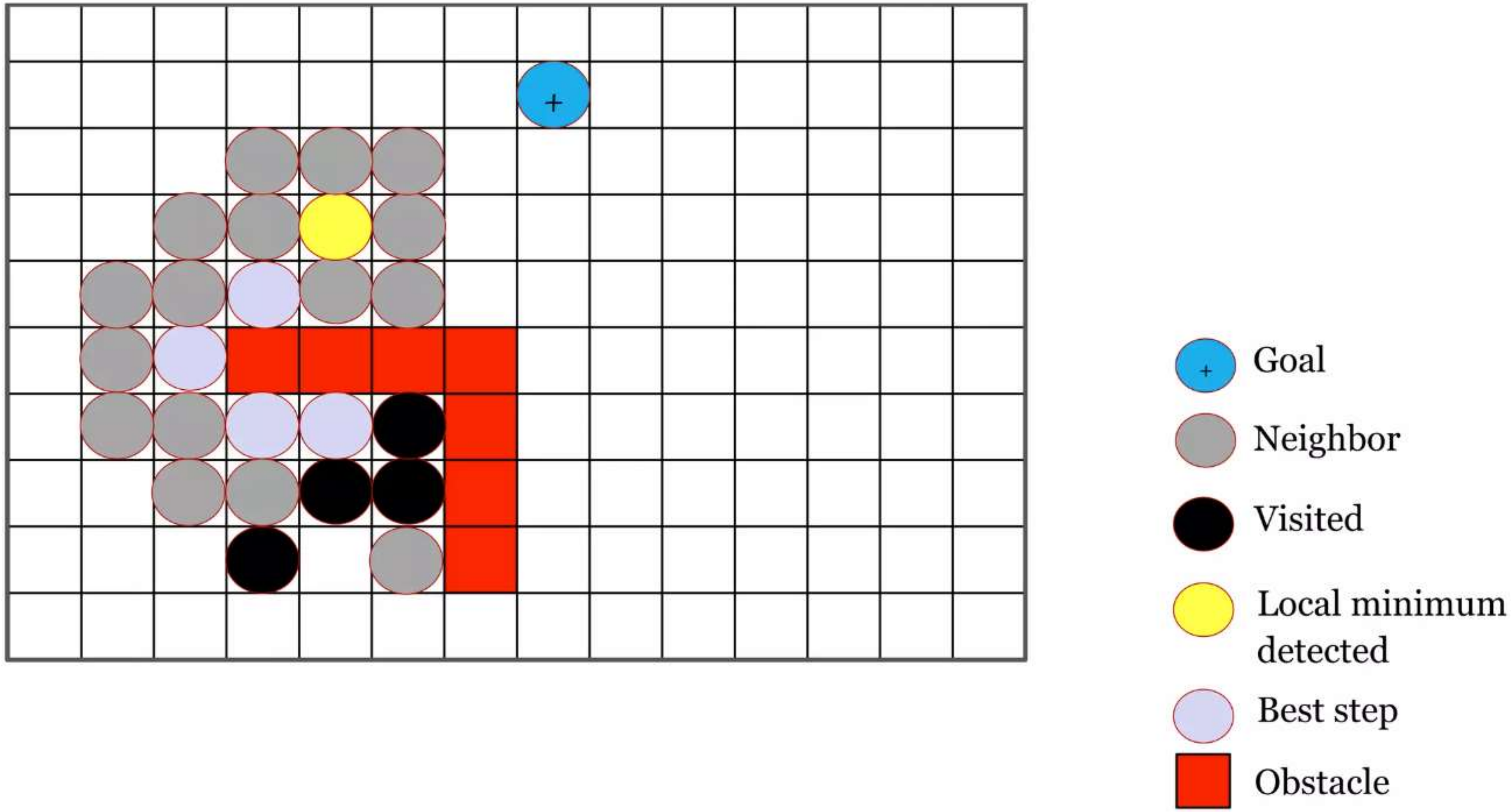
Best-First Search



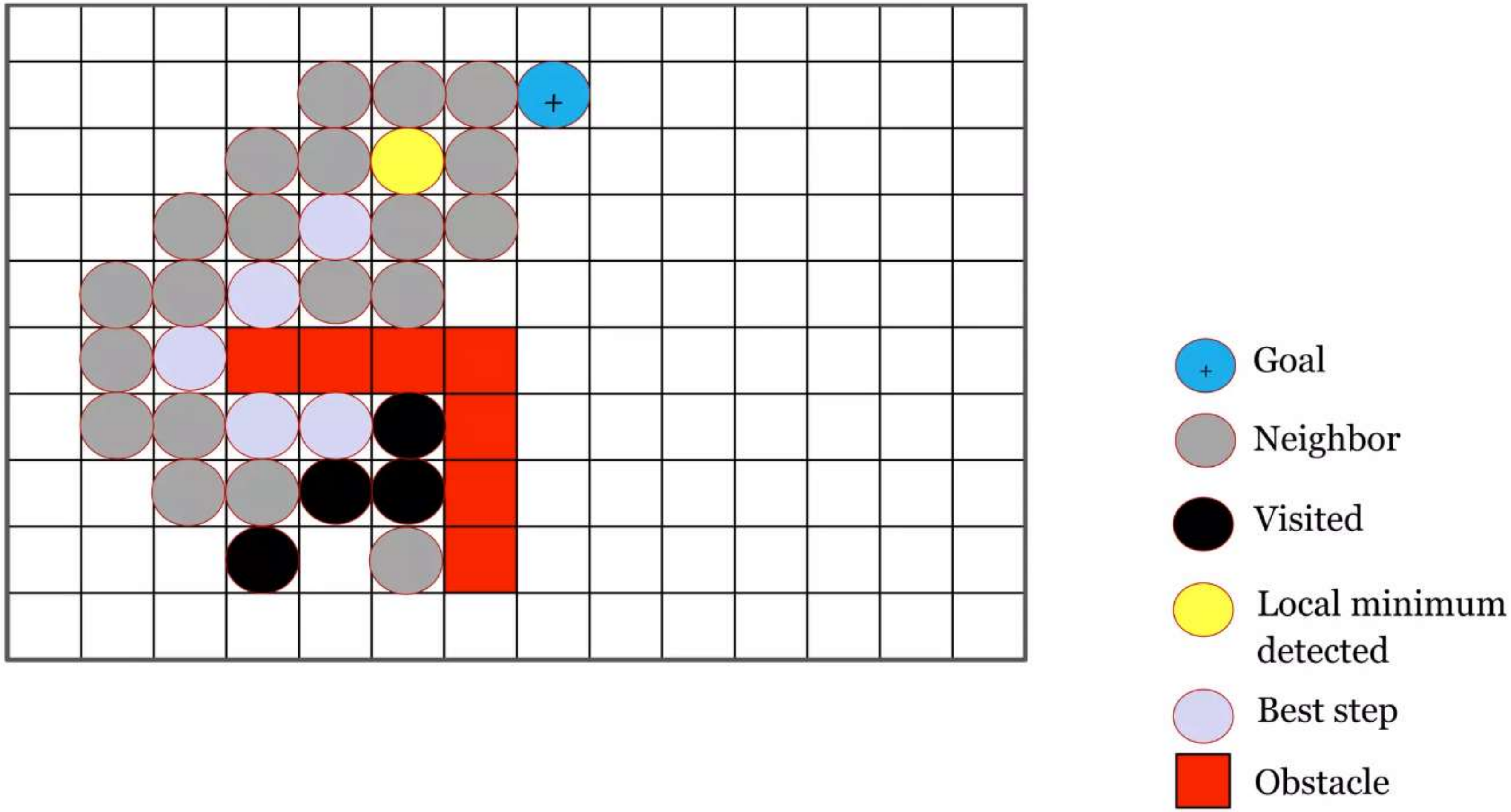
Best-First Search



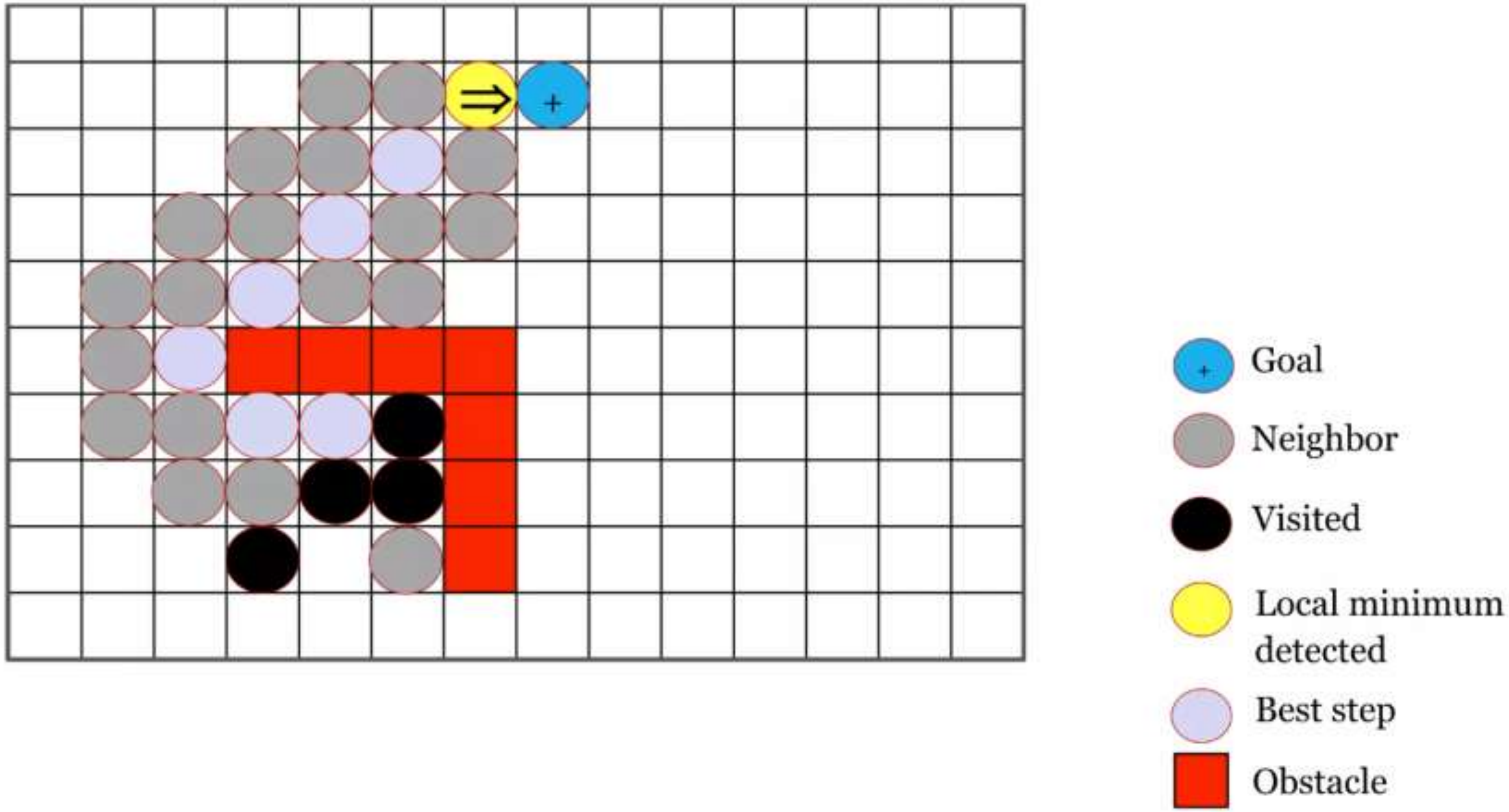
Best-First Search



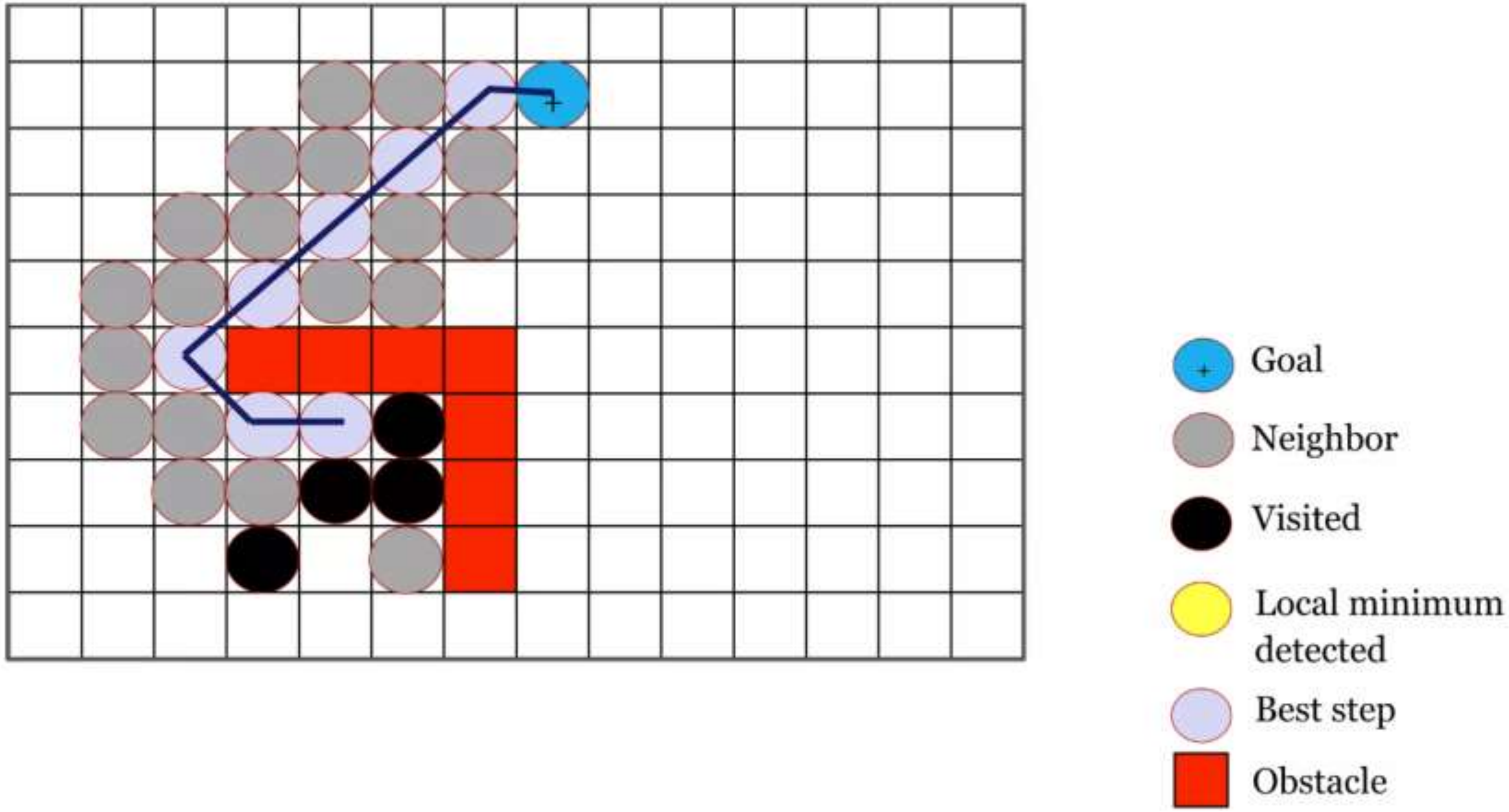
Best-First Search



Best-First Search



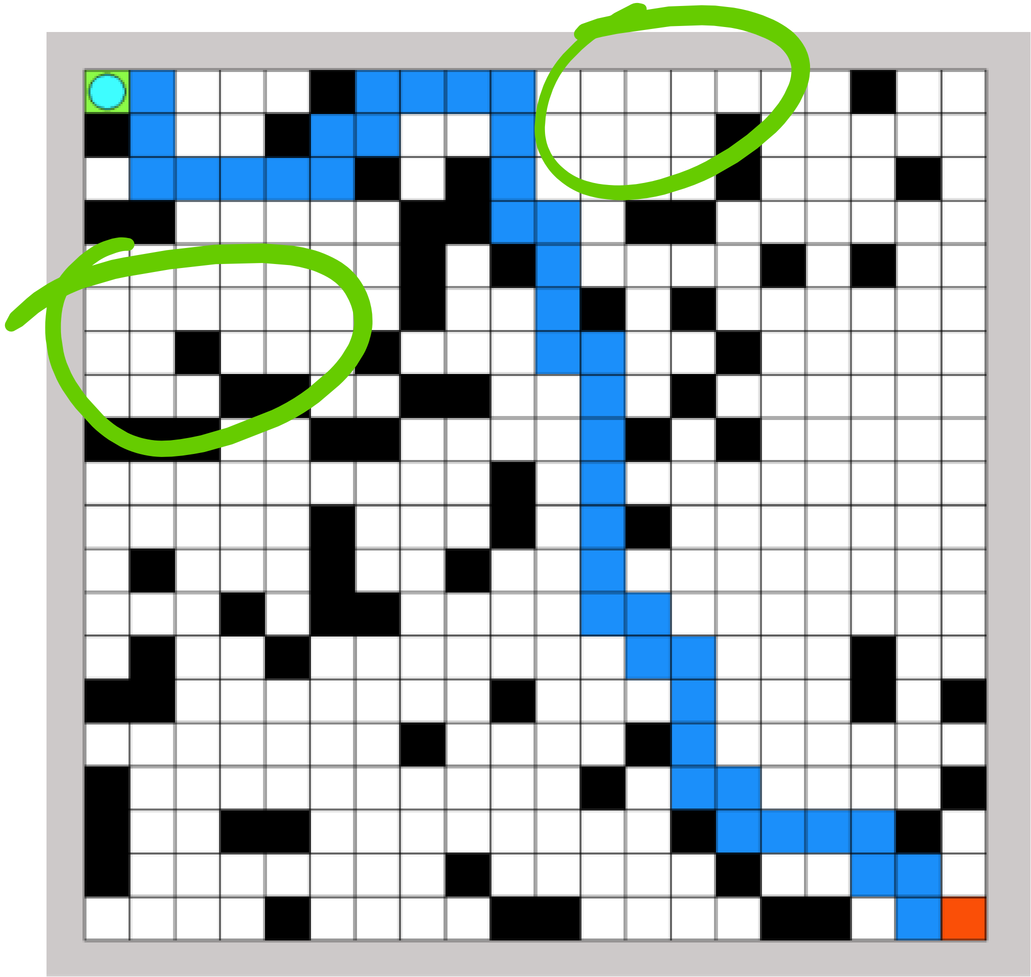
Best-First Search



Best-First Search

- **Best-first**
  - ◇ It is a kind or **mixed depth and breadth first** search.
  - ◇ Adds the successors of a node to the expand list.
  - ◇ All nodes on the list are sorted according to the **heuristic values**.
  - ◇ Expand **most desirable unexpanded** node.
  - ◇ Special Case: **A***.

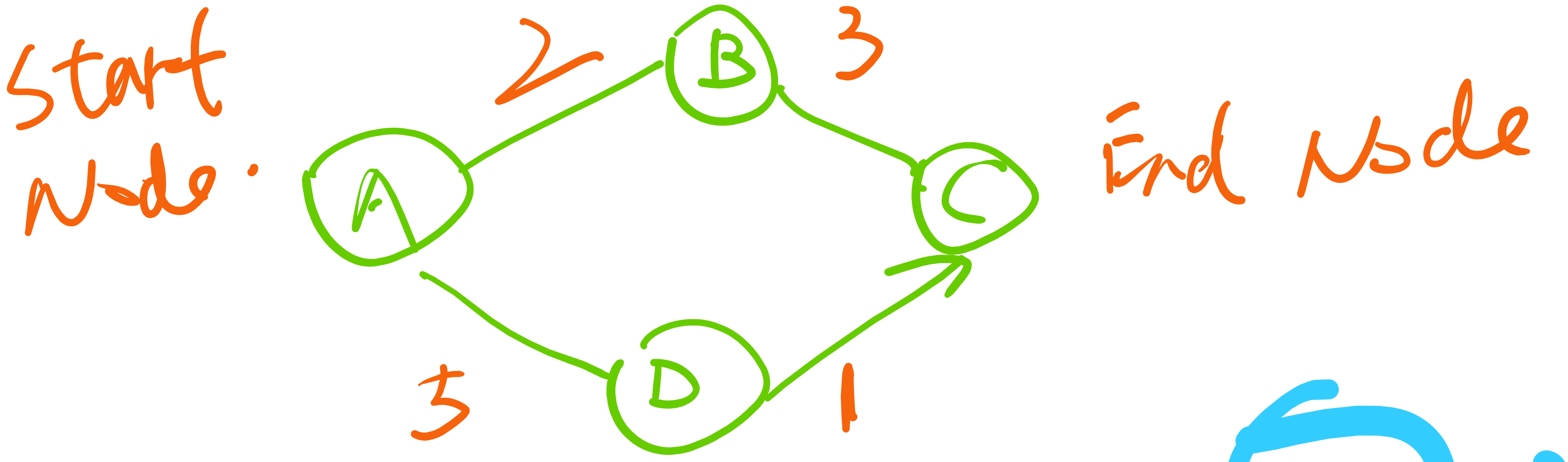
A*



https://en.wikipedia.org/wiki/A*_search_algorithm

https://www.redblobgames.com/pathfinding/a-star/introduction.html

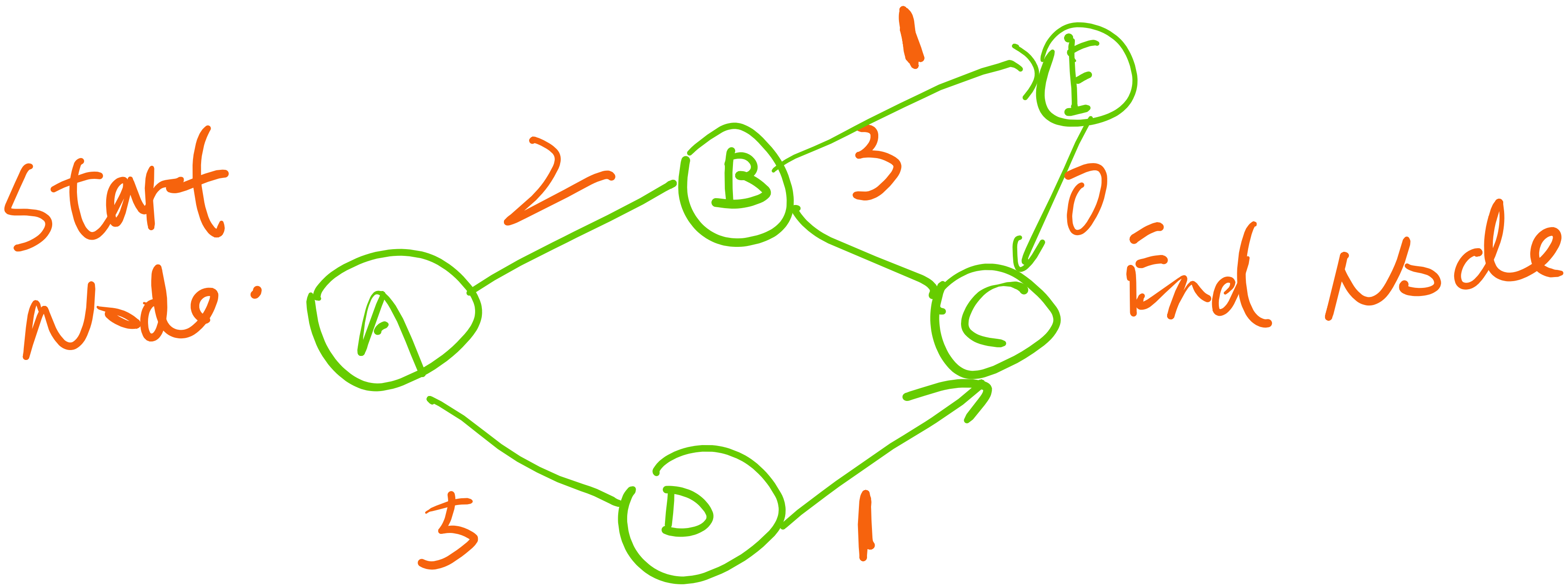
A*



A-B-C = 2+3=$\boxed{5}$ ✓

A-D-C = 5+1=$\underline{6}$

A*



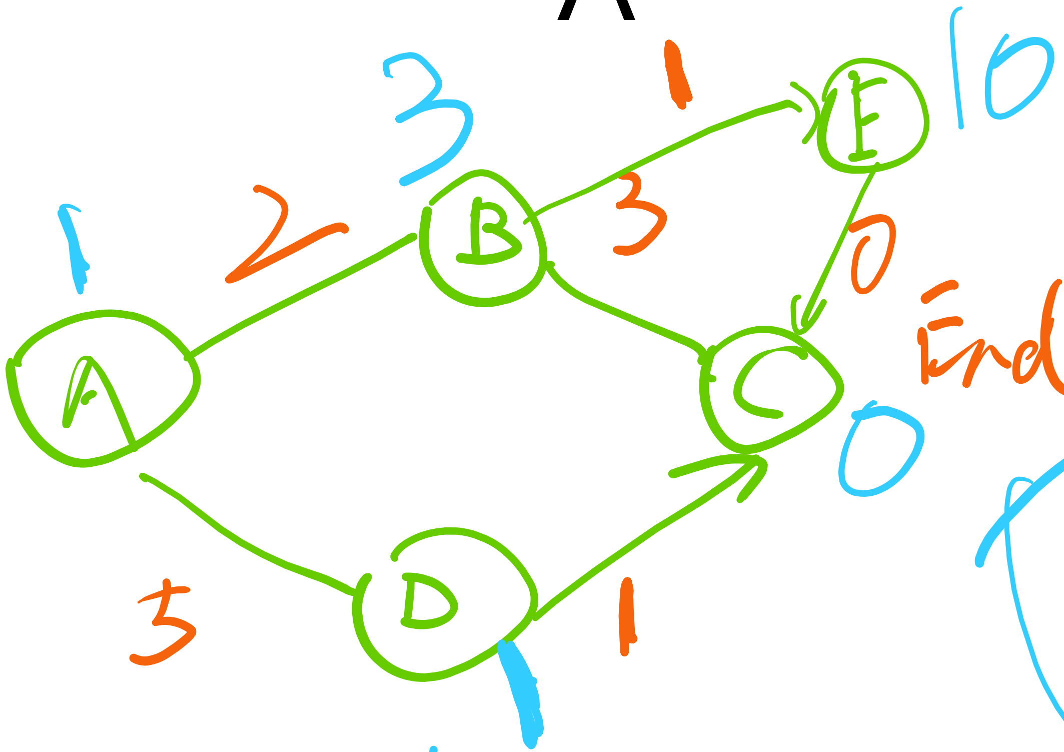
A-B-C = 2+3=5

A-D-C = 5+1=6

A-B-E-C = 2+1=3. ✓

A*

Start Node.



End Node

Heuristic value

$h(n)$

path weight $g(n)$

☆ cost function $f(n) = g(n) + h(n)$

A*

$$f(n) = g(n) + h(n)$$

$f(n)$ ↓ total cost

$g(n)$ ↓ dist(start)

$h(n)$ ↓ dist(goal)

↓

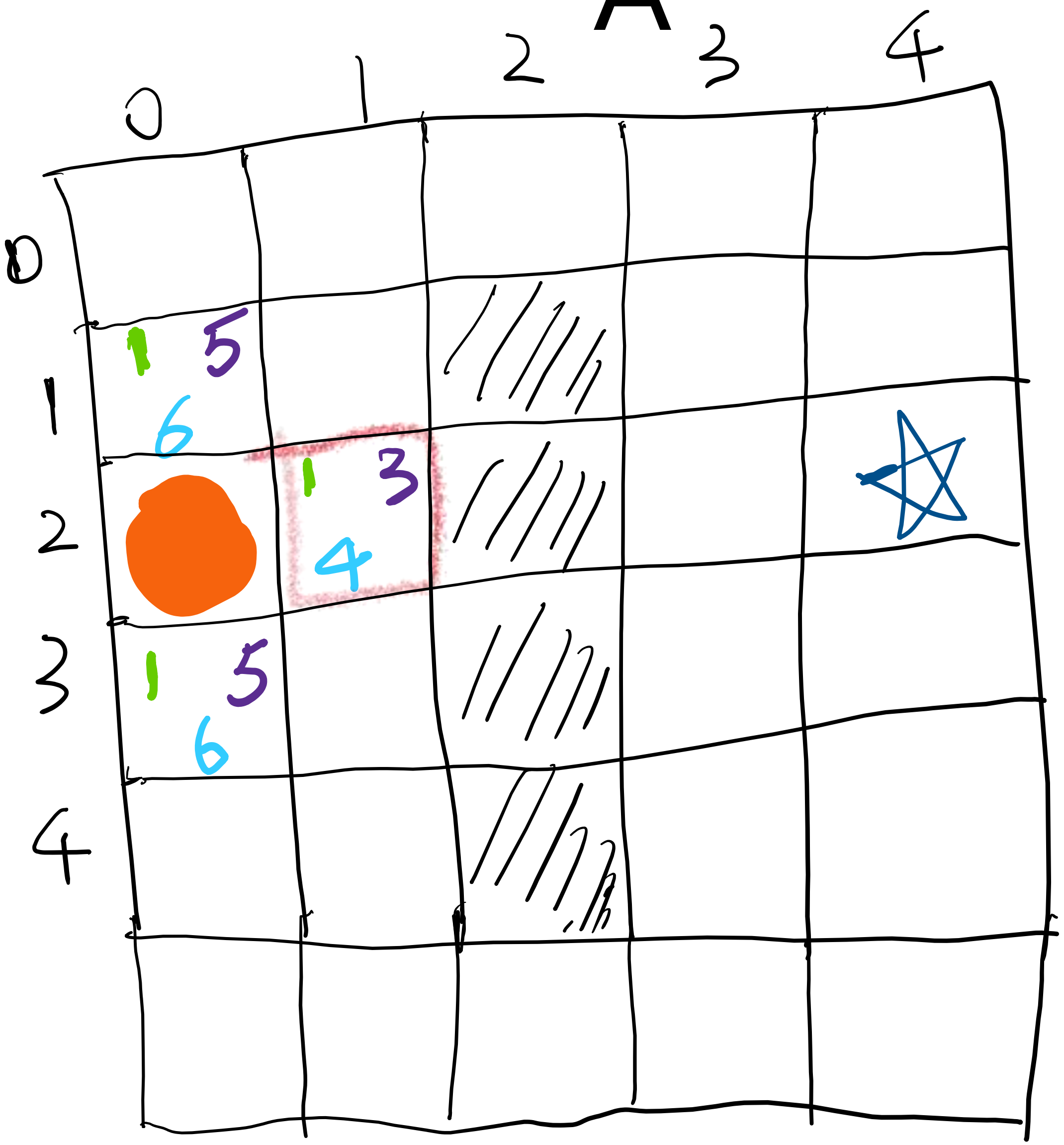
estimated distance from the current node to the end node

A*

h(n)

Manhattan Distance

$$d_{12} = |x_1 - x_2| + |y_1 - y_2|$$



$f(n) = g(n) + h(n)$

open-list (to visit)

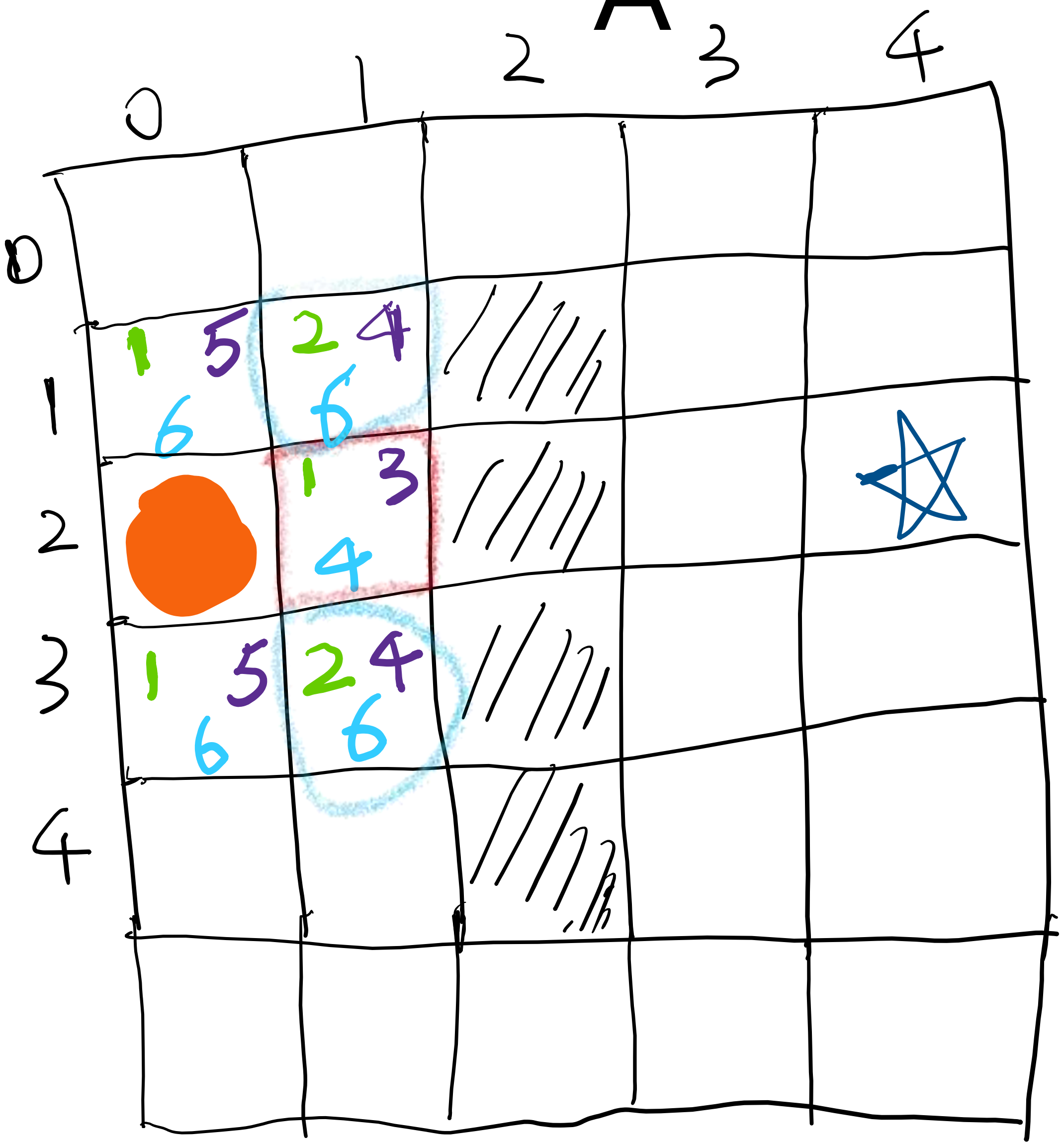
(1,2), p: start

(0,1), p: start

(0,3), p: start

closed-list:

A*



$f(n) = g(n) + h(n)$

open-list · (to visit)

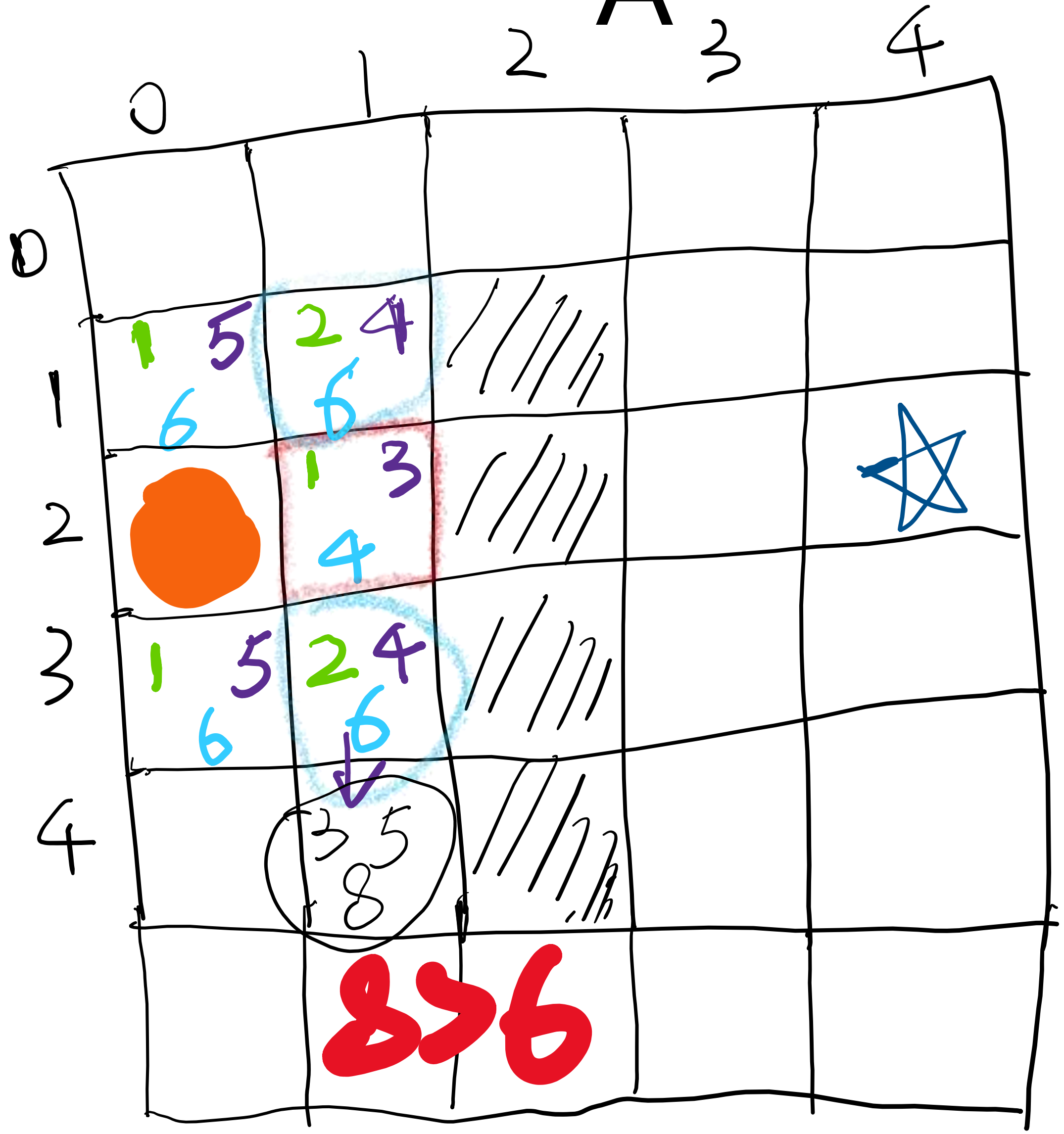
(1, 1) p: (1, 2) 6

(1, 3) p: (1,2), 6

closed-list:

(1, 2), p: start.

A*



$f(n)=g(n)+h(n)$

open-list (to visit)

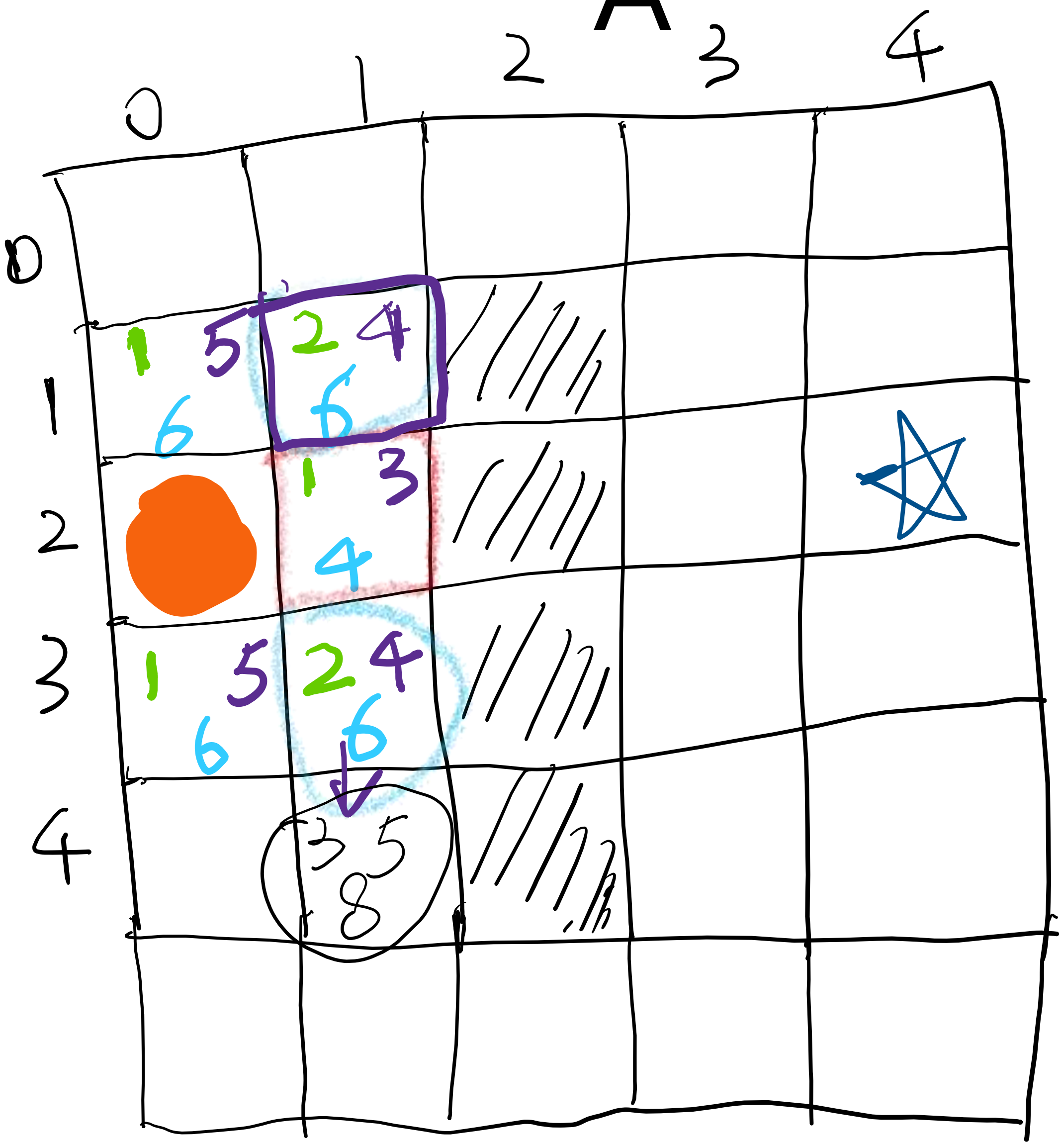
(1,1) p: (1,2) 6

(1,4) p: (1,~~2~~3), 8 ?

closed-list:

(1,2), p: start.

A*



$f(n) = g(n) + h(n)$

open-list · (to visit)

(1, 1) P: (1, 2) 6

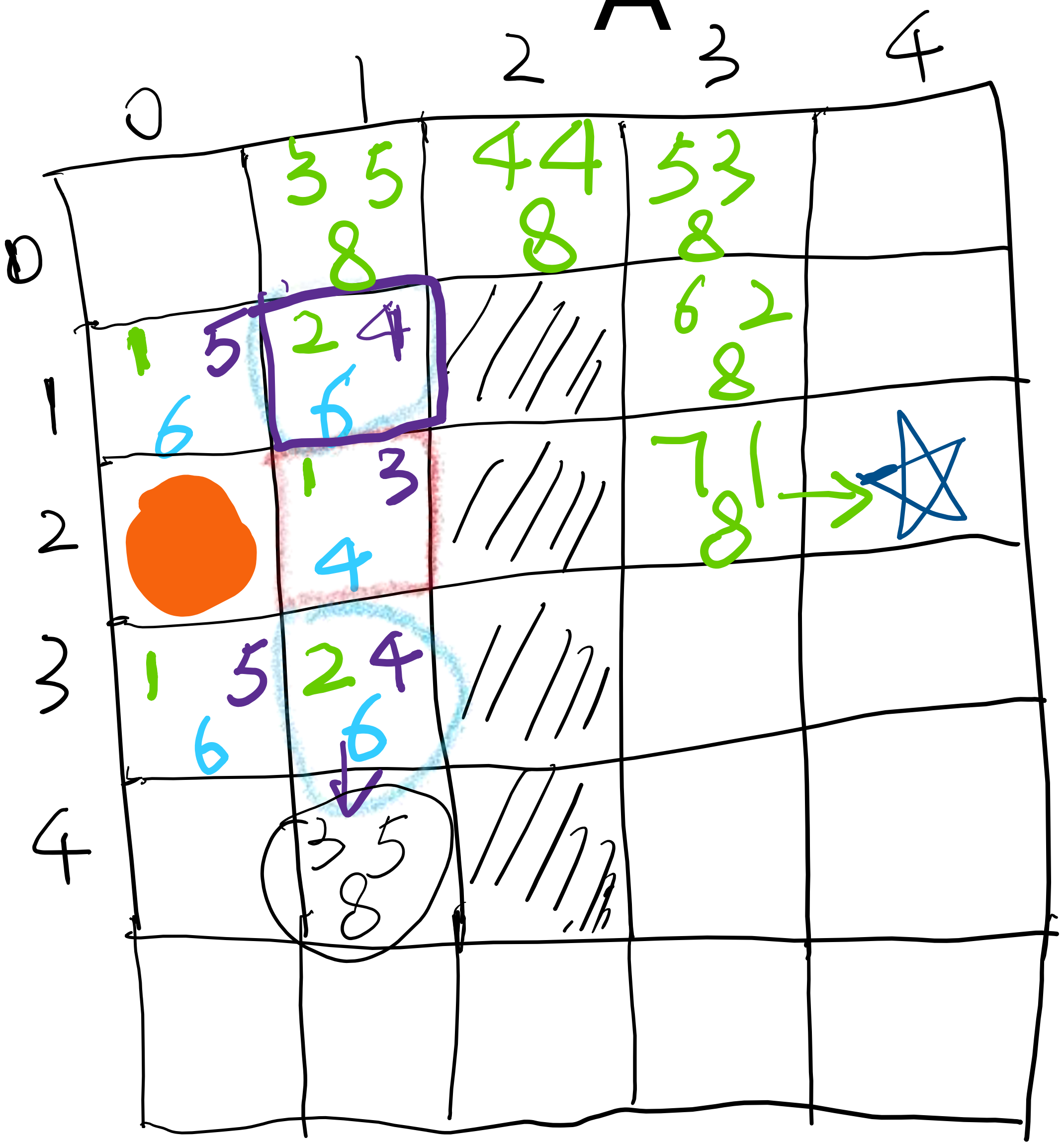
(1, 4) P: (1, ~~2~~3), 8 (?)

closed-list:

(1, 2), P: Start.

(1, 3), P: (1, 2), 6

A*



$f(n) = g(n) + h(n)$

open-list · (to visit)

(1, 1) P: (1, 2) 6

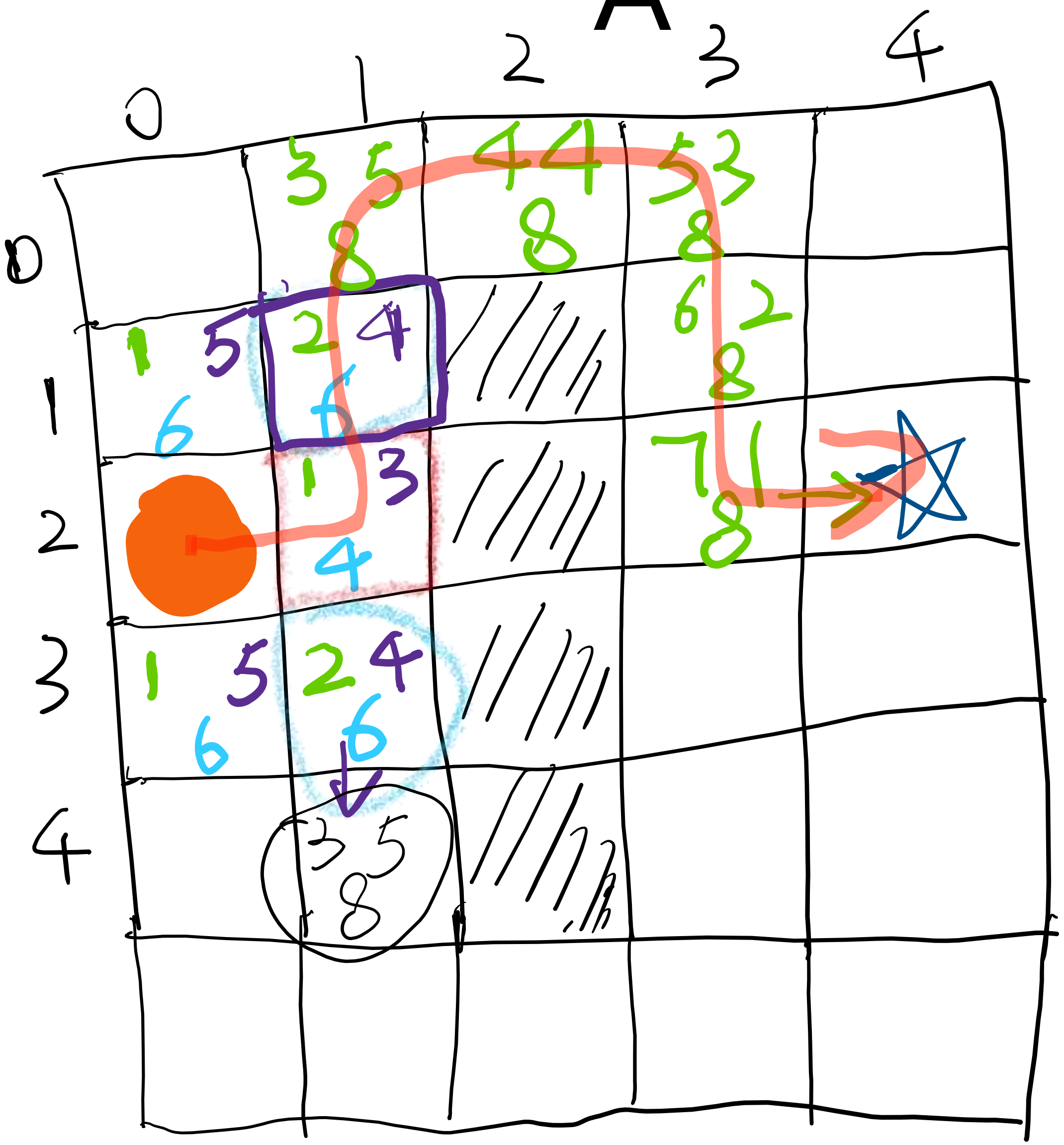
(1, 4) P: (1, ~~2~~3), 8 ?

closed-list:

(1, 2), P: Start.

(1, 3), P: (1, 2). 6

A*



f(n) = g(n) + h(n)

open-list (to visit)

(1, 1) p: (1, 2) 6

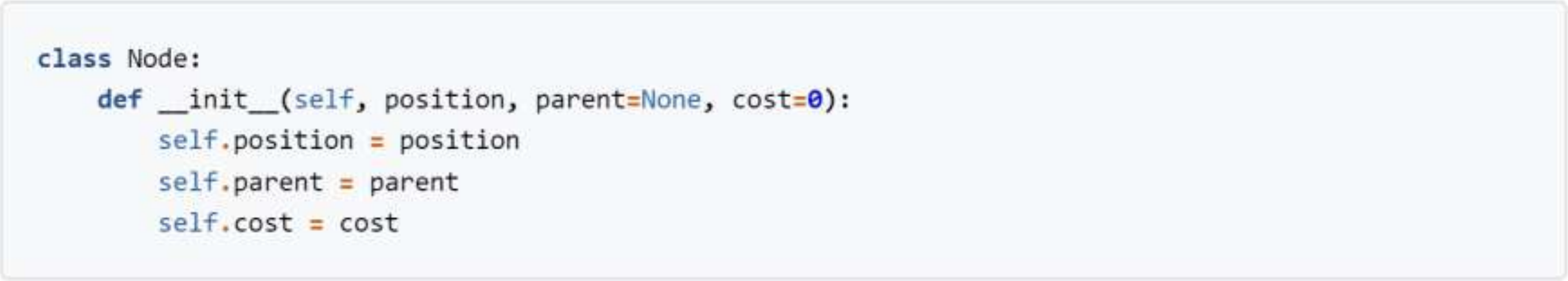
(1, 4) p: (1, ~~2~~3), 8 ?

closed-list:

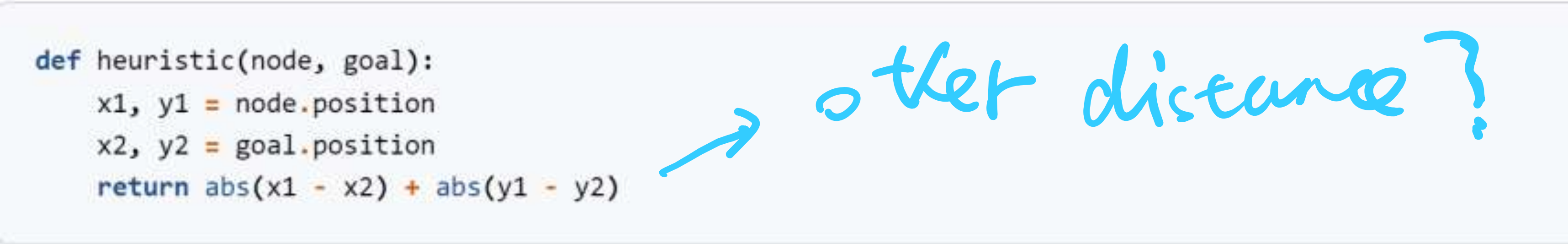
(1, 2), p: start.

(1, 3), p: (1, 2). 6

A* implementation

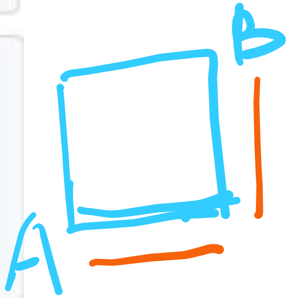


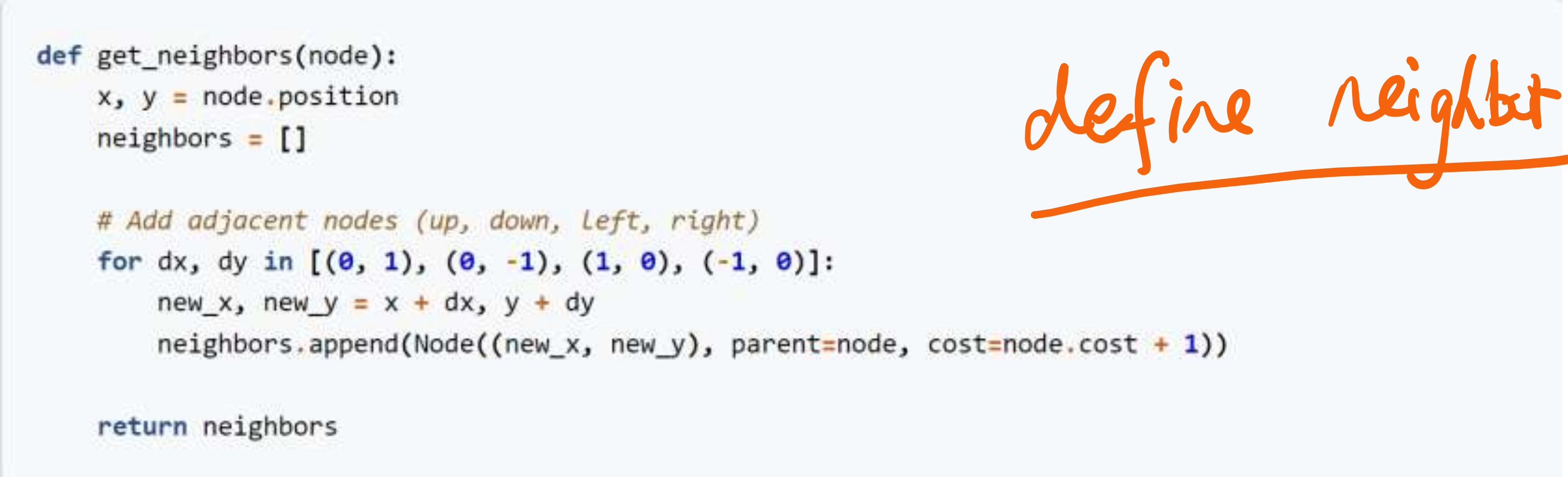
```python
class Node:
    def __init__(self, position, parent=None, cost=0):
        self.position = position
        self.parent = parent
        self.cost = cost
```



```python
def heuristic(node, goal):
    x1, y1 = node.position
    x2, y2 = goal.position
    return abs(x1 - x2) + abs(y1 - y2)
```

→ other distance?



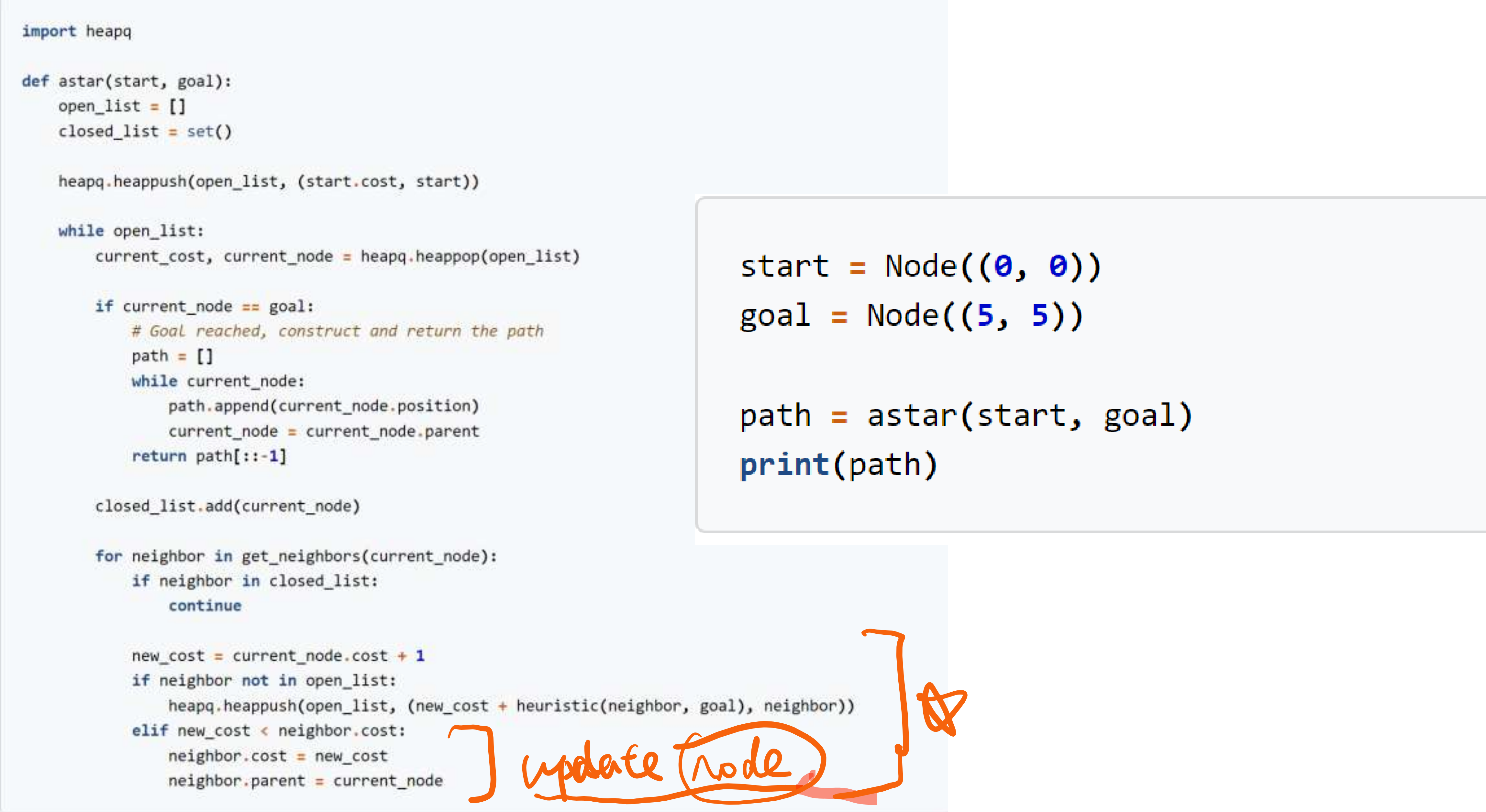


```python
def get_neighbors(node):
    x, y = node.position
    neighbors = []

    # Add adjacent nodes (up, down, left, right)
    for dx, dy in [(0, 1), (0, -1), (1, 0), (-1, 0)]:
        new_x, new_y = x + dx, y + dy
        neighbors.append(Node((new_x, new_y), parent=node, cost=node.cost + 1))

    return neighbors
```

define neighbor nodes.

https://saturncloud.io/blog/implementing-the-a-algorithm-in-python-a-stepbystep-guide/

A* implementation



```
import heapq

def astar(start, goal):
    open_list = []
    closed_list = set()

    heapq.heappush(open_list, (start.cost, start))

    while open_list:
        current_cost, current_node = heapq.heappop(open_list)

        if current_node == goal:
            # Goal reached, construct and return the path
            path = []
            while current_node:
                path.append(current_node.position)
                current_node = current_node.parent
            return path[::-1]

        closed_list.add(current_node)

        for neighbor in get_neighbors(current_node):
            if neighbor in closed_list:
                continue

            new_cost = current_node.cost + 1
            if neighbor not in open_list:
                heapq.heappush(open_list, (new_cost + heuristic(neighbor, goal), neighbor))
            elif new_cost < neighbor.cost:
                neighbor.cost = new_cost
                neighbor.parent = current_node
```

```
start = Node((0, 0))
goal = Node((5, 5))

path = astar(start, goal)
print(path)
```

update node

A*

$$f(n)=g(n)+\underline{h(n)}$$

Advantages:

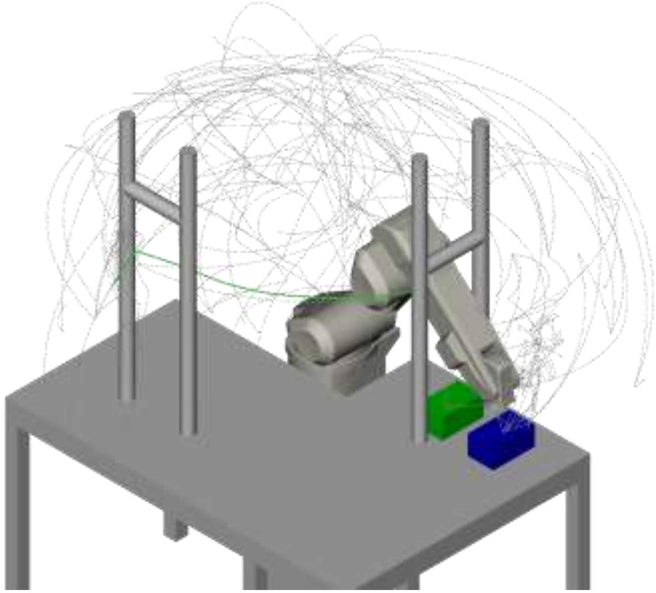
- It is optimal search algorithm in terms of heuristics.
- It is one of the best heuristic search techniques.
- It is used to solve complex search problems.
- There is no other optimal algorithm guaranteed to expand fewer nodes than A*.

Disadvantages:

- This algorithm is complete if the branching factor is finite and every action has fixed cost.
- The performance of A* search is dependant on accuracy of heuristic algorithm used to compute the function h(n).

Today's Agenda

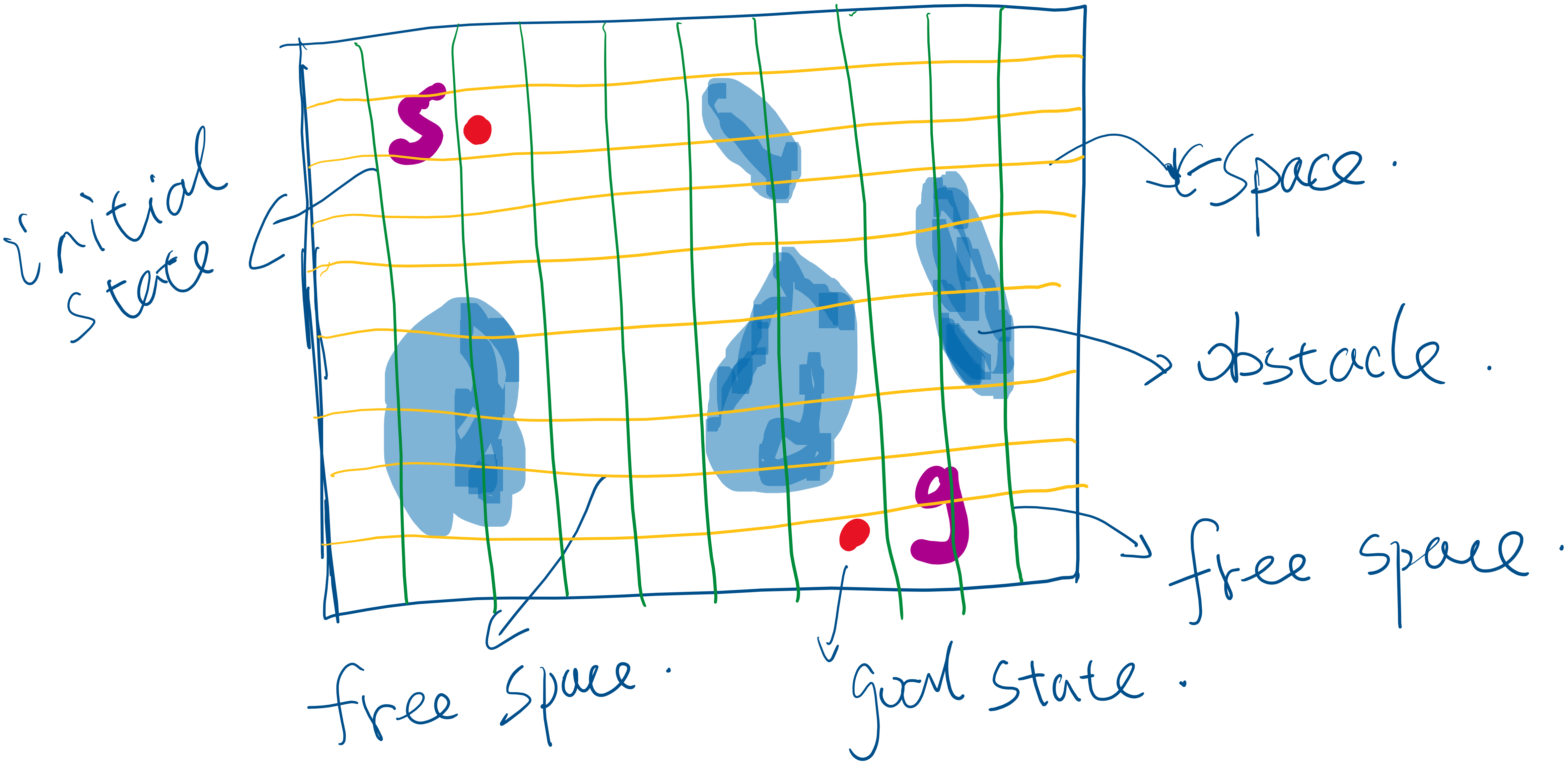
- What is planning? (~10)
- Motion planning in robotic application （~10）
    - Self-driving, drone, robot arm, humanoids, medical robots, soft robots …
- Formulation of robot motion planning
- Planning as searching (~25)
- Planning as sampling (~25)
    - PRM, RRT, RRT*



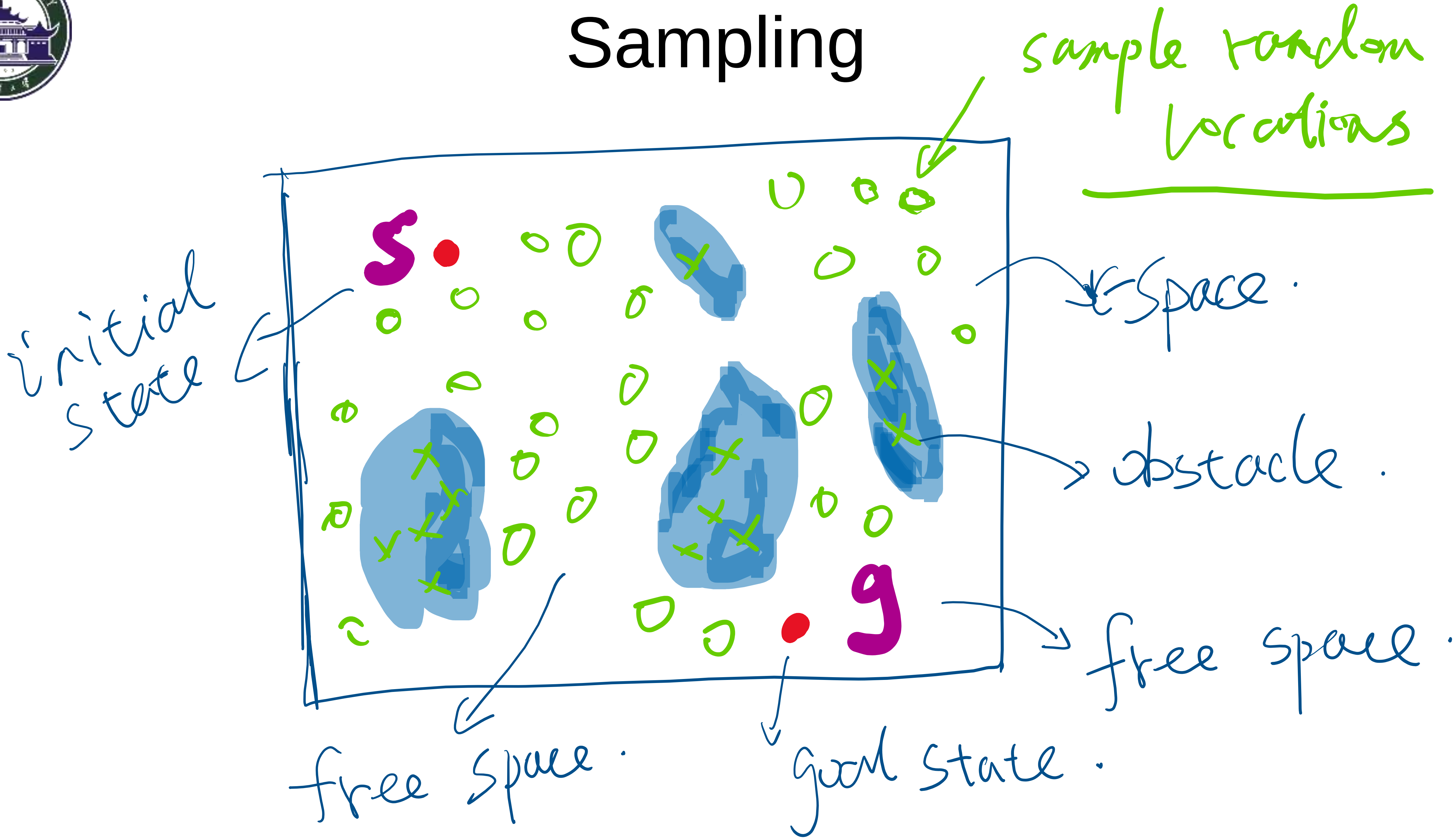
Sampling

- **Completely describing and optimally exploring is too hard in high dimension space**
- **It is not necessary**
- **Limit ourselves to finding a “good” sampling**

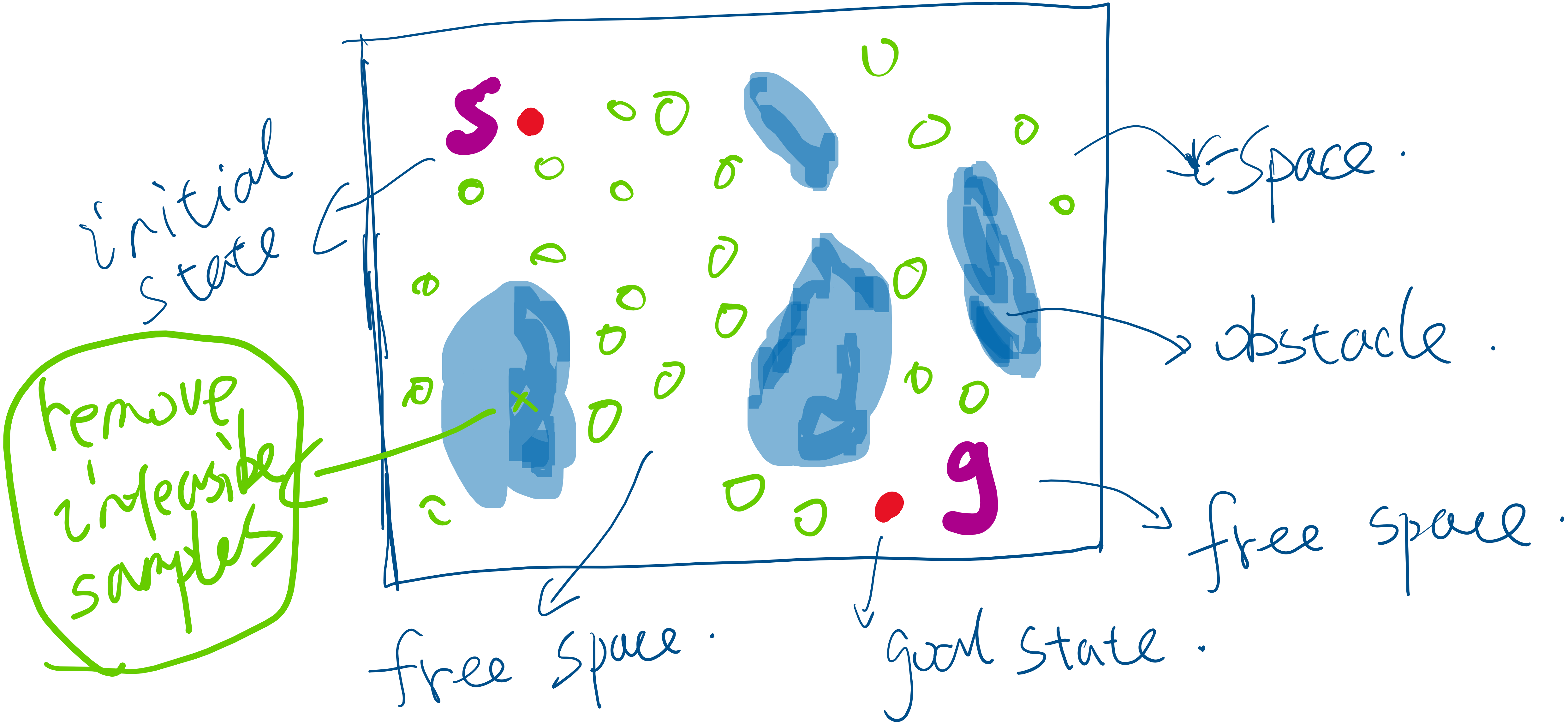
Sampling


s
g
initial state
X-space.
obstacle.
free space.
free space.
goal state.

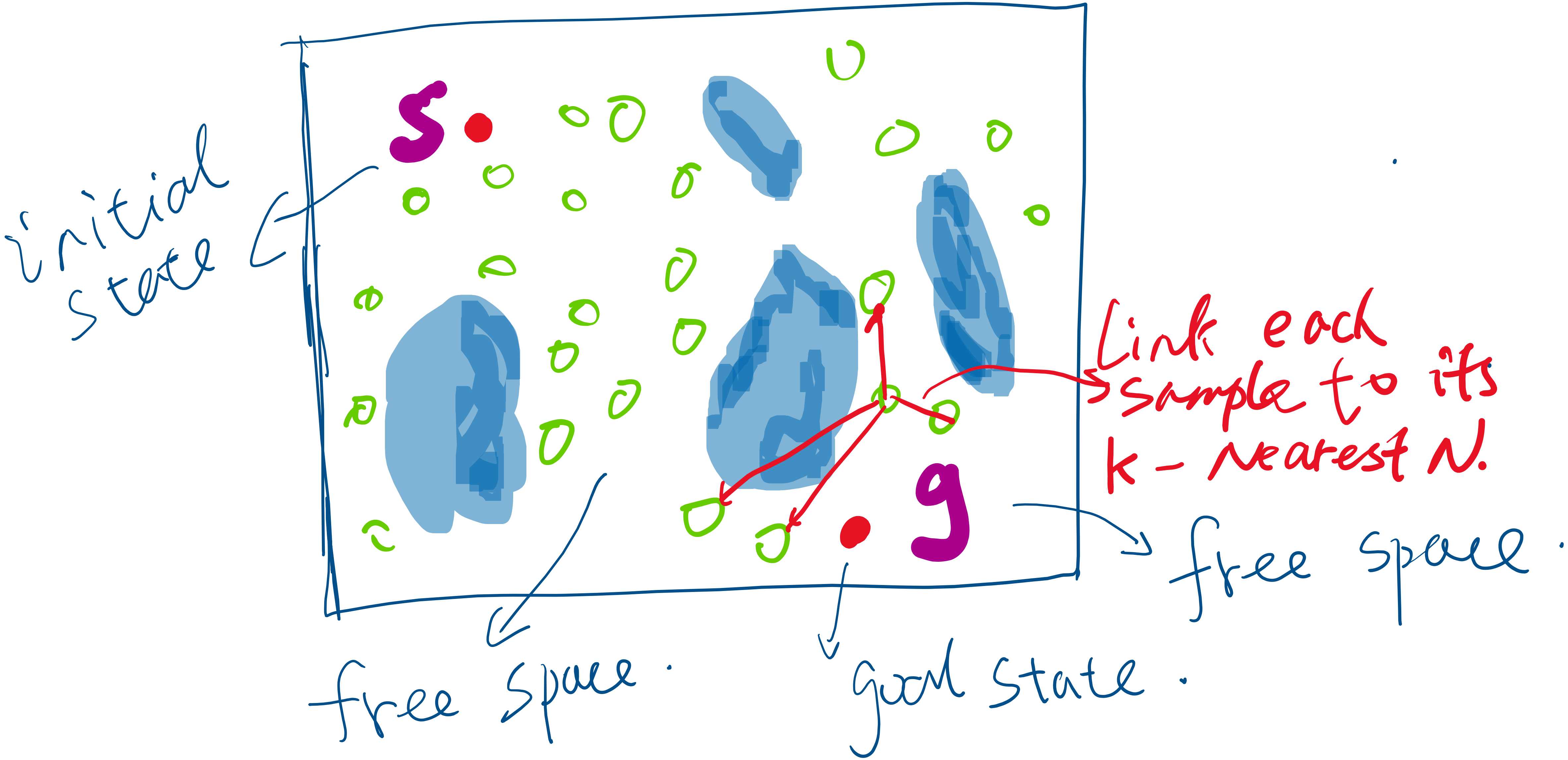
Sampling


sample random locations
initial state
s
space
obstacle
g
free space
free space
goal state

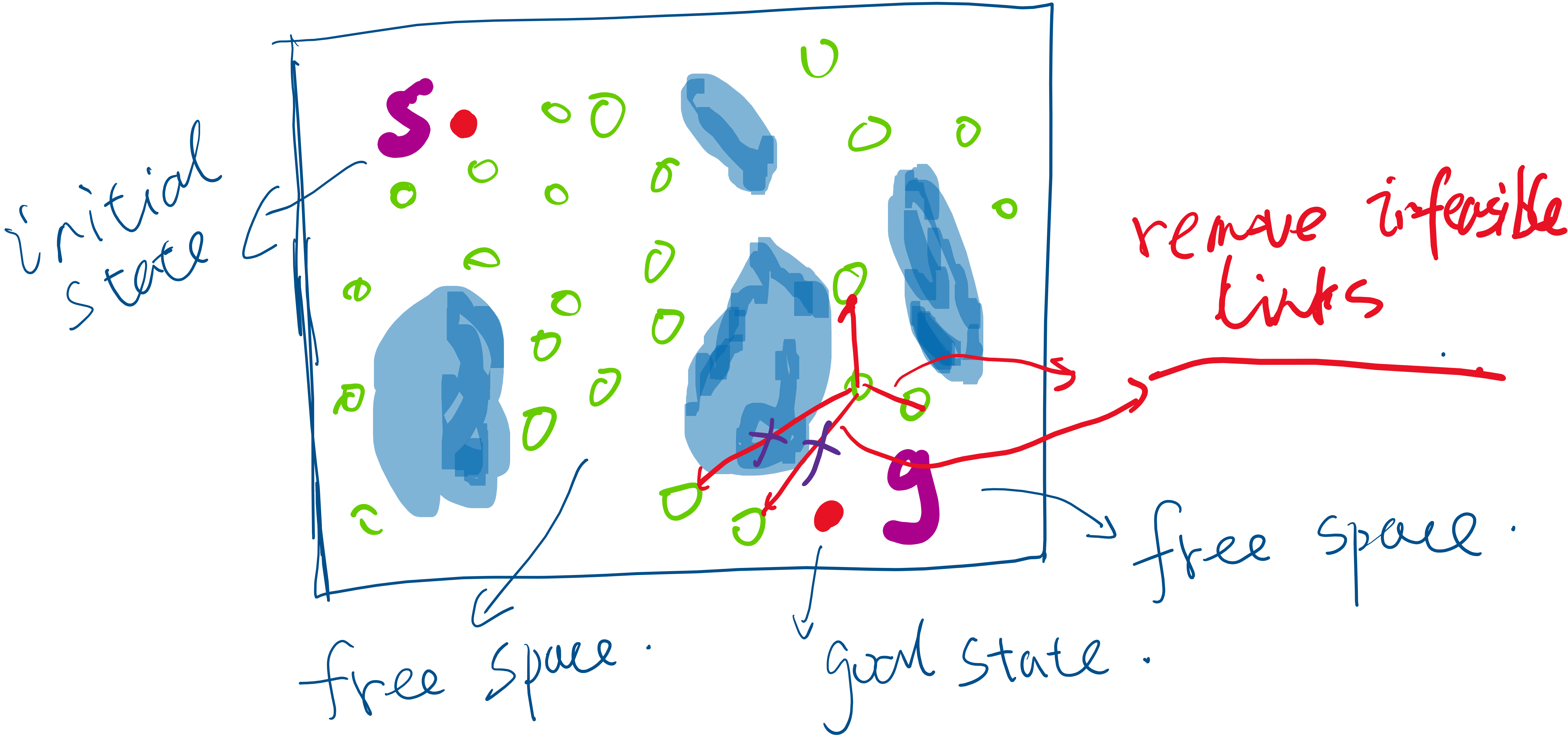
Sampling


initial state
s
g
remove infeasible samples
C-space.
obstacle.
free space.
free space.
goal state.

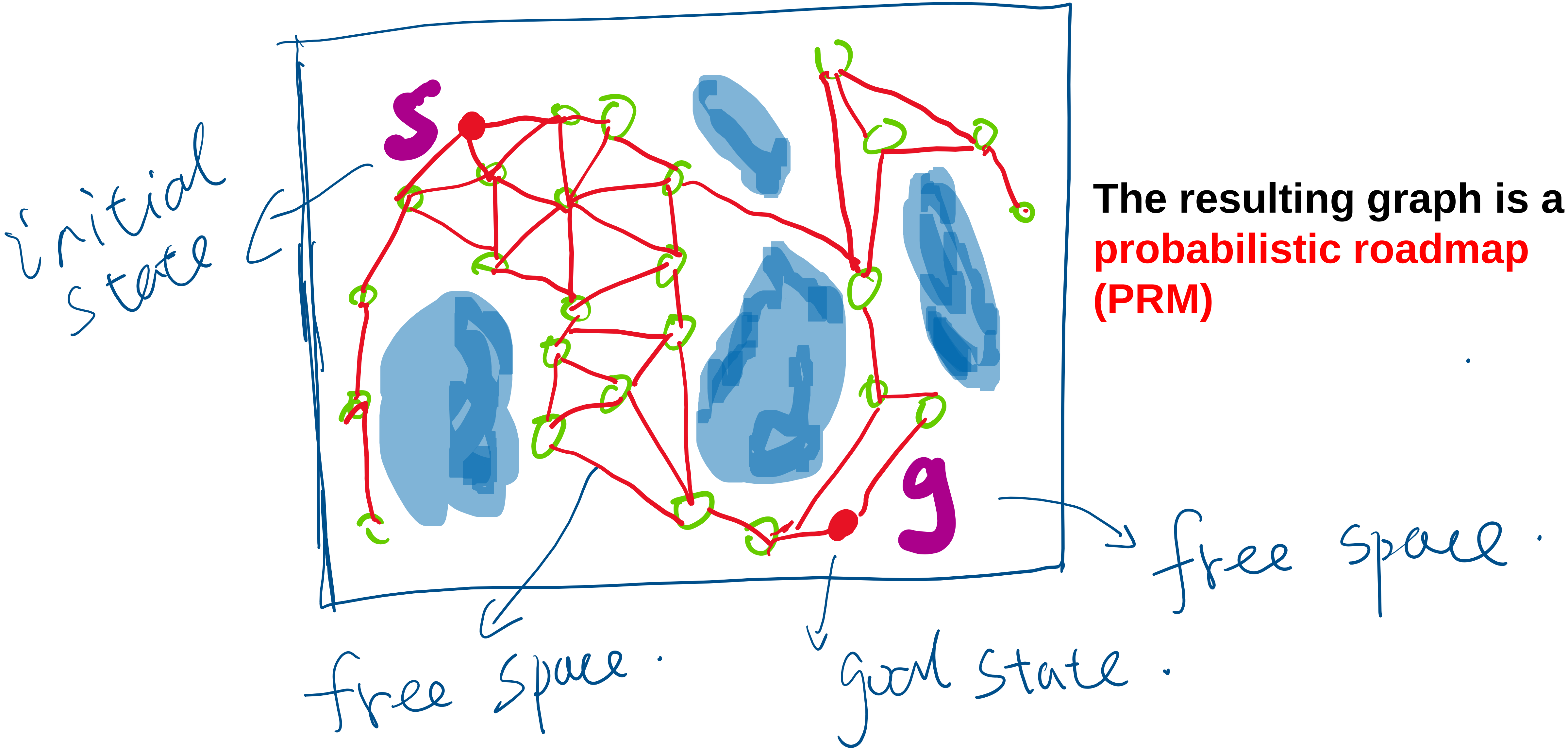
Sampling


s
g
initial state
free space
goal state
free space
Link each sample to its k-Nearest N.

Sampling

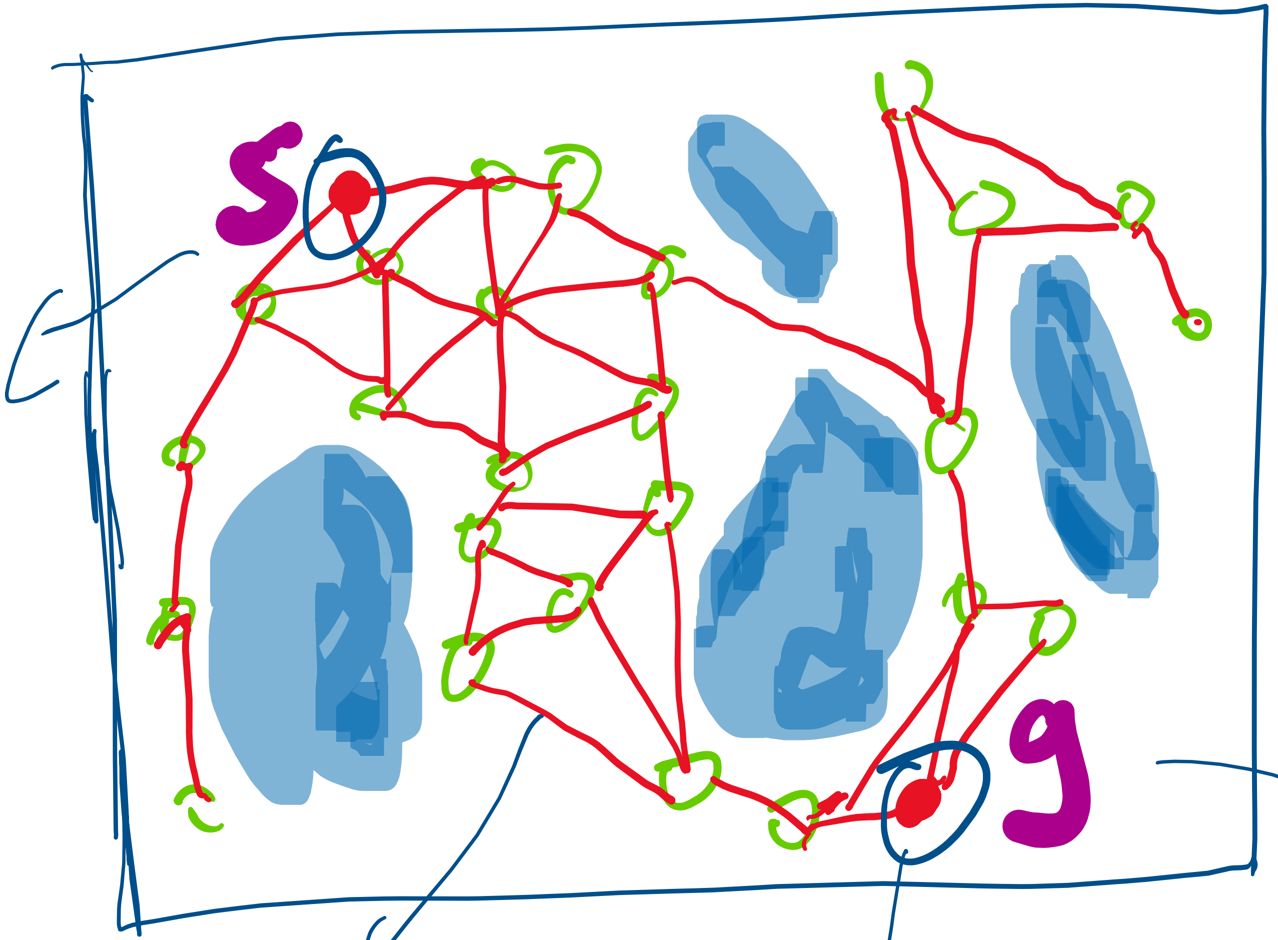

initial state
S
remove infeasible links
free space.
goal state.
g
free space.

PRM
(probabilistic roadmap)



The resulting graph is a probabilistic roadmap (PRM)

PRM
(probabilistic roadmap)



initial state

search using A* on PRM

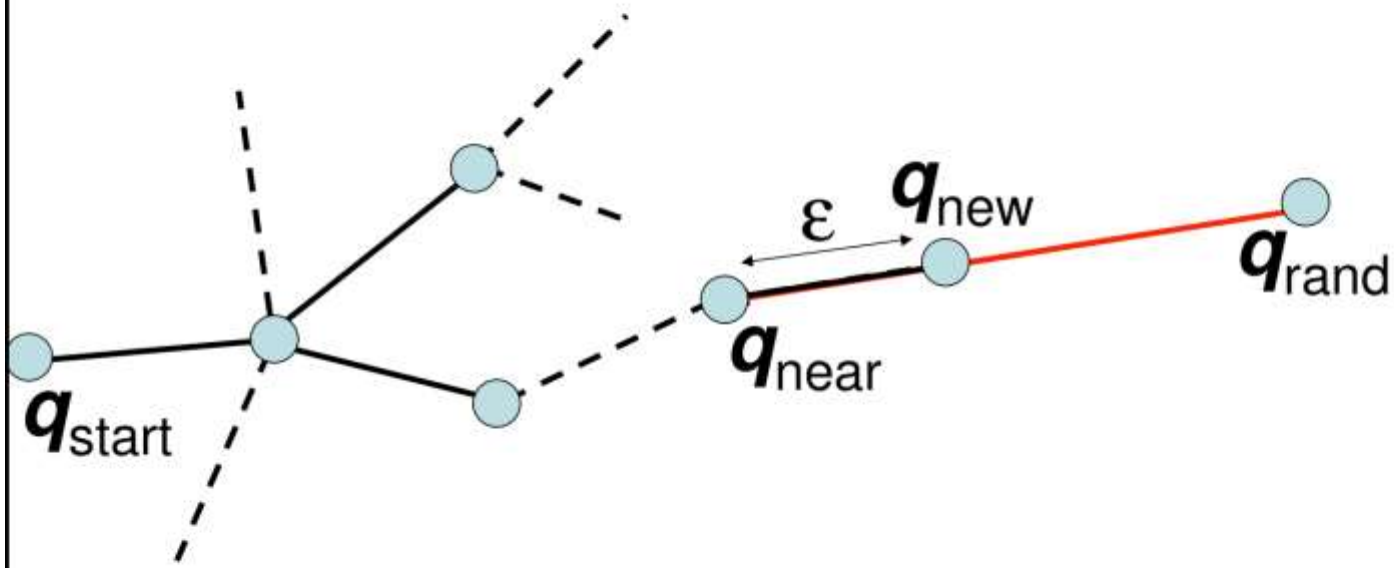
free space.

free space.

goal state.

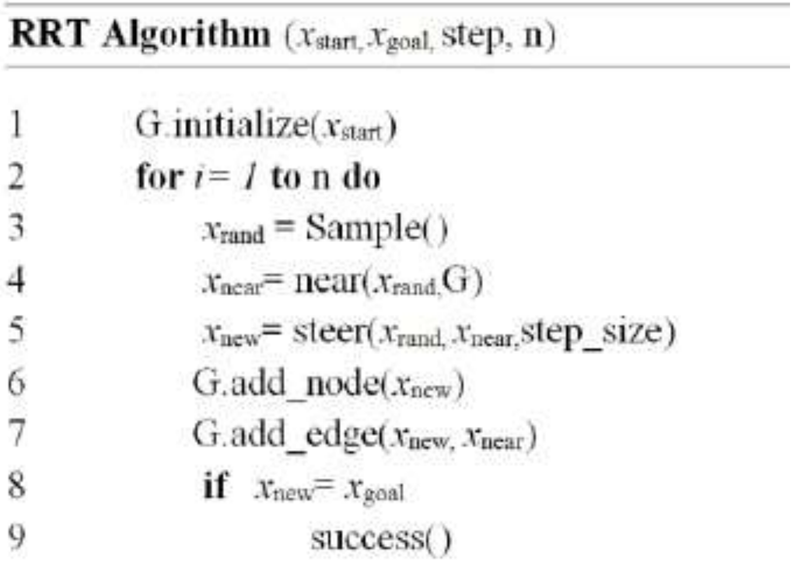
RRT

Rapidly Exploring Random Trees



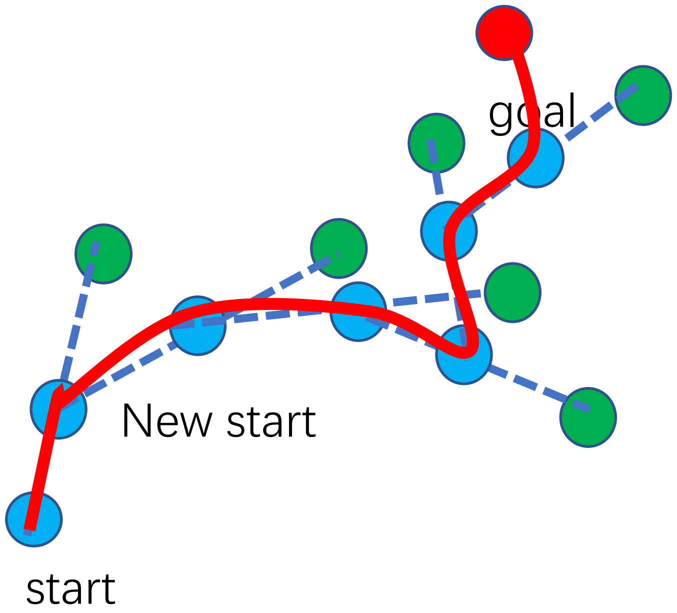
Remarkably, we can find a solution by using *relatively few randomly* sampled points.

RRT

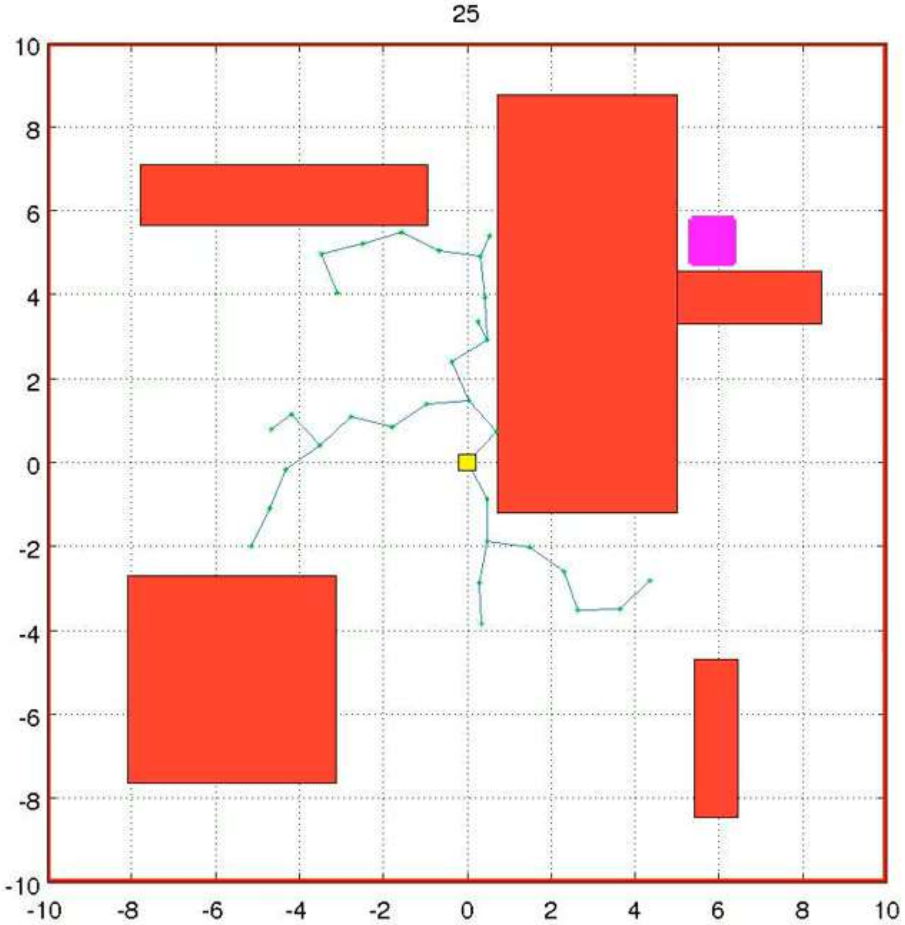


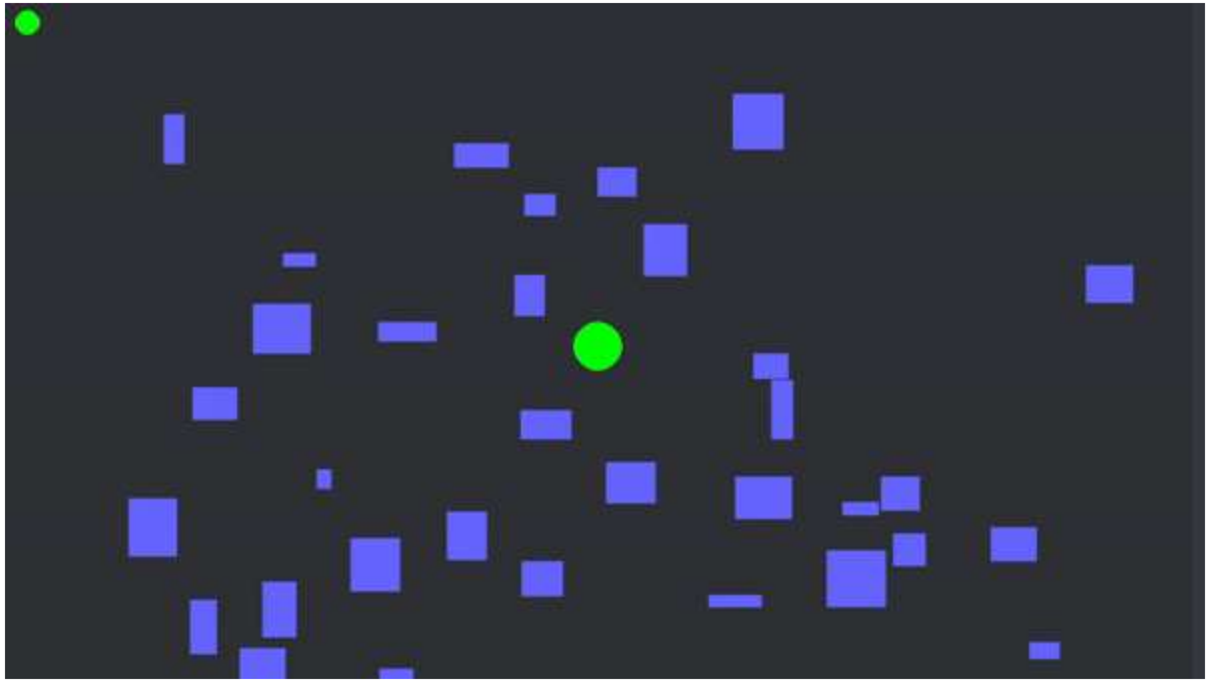
```
RRT Algorithm ($x_{start}$, $x_{goal}$, step, n)
    G.initialize($x_{start}$)
    for $i = 1$ to n do
        $x_{rand}$ = Sample()
        $x_{near}$ = near($x_{rand}$, G)
        $x_{new}$ = steer($x_{rand}$, $x_{near}$, step_size)
        G.add_node($x_{new}$)
        G.add_edge($x_{new}$, $x_{near}$)
        if $x_{new} = x_{goal}$
            success()
```

- J-C. Latombe. Robot Motion Planning. Kluwer. 1991.
- S. Lavalle. Planning Algorithms. 2006. http://msl.cs.uiuc.edu/planning/
- H. Choset et al., Principles of Robot Motion: Theory, Algorithms, and Implementations. 2006.



RRT



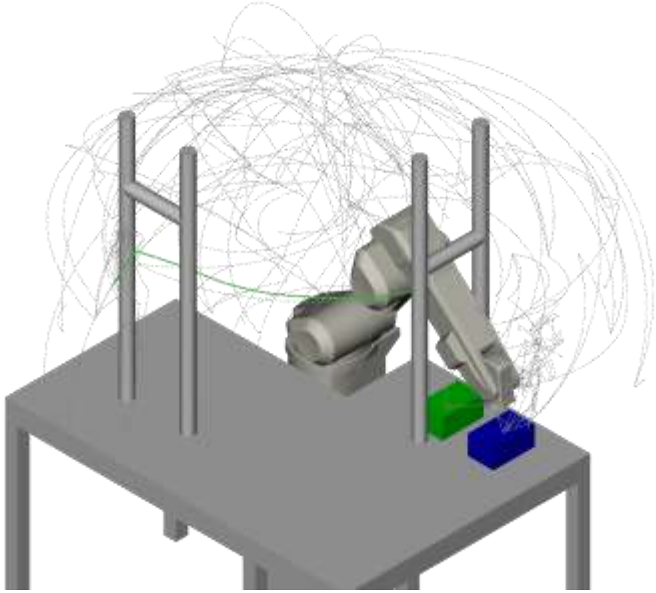


RRT

https://www.youtube.com/watch?v=gP6MRe_IHFo&ab_channel=JacksonBernatchez

Today’s Agenda

- **What is planning? (~10)** ✓
- **Motion planning in robotic application (~10)** ✓
  - **Self-driving, drone, robot arm, humanoids, medical robots, soft robots …** ✓
- **Formulation of robot motion planning** ✓
- **Planning as searching (~25)** ✓
- **Planning as sampling (~25)** ✓
  - **PRM, RRT, RRT***



Summary

- **Planning is *searching*! (Find a feasible or an optimal solution)**
- **All the robots are represented as “dots”**
- **All the obstacles representation are not explicitly discussed.**
- **The constraints from the robot itself are not considered right now**
- **The dimensionality (DOF, Task space)**
- **Robot kinematics and dynamics, hardware (GPU)**
- **Heuristics design, learning, optimization**
- **Time complexity, real-time applications**

h(n)

M-P
E-D

Goal for this course

- **Design：soft hand design x1**
- **Perception: vision, point cloud, tactile, force/torque x1**
- **Planning: sampling-based, optimization-based, learning-based x3**
- **Control: feedback, multi-modal x2**
- **Learning: imitation learning, RL x2**
- **Simulation tool (pybullet, matlab, OpenRAVE, Issac Nvidia, Gazebo)**
- **How to get a robot moving!**