Topic 7

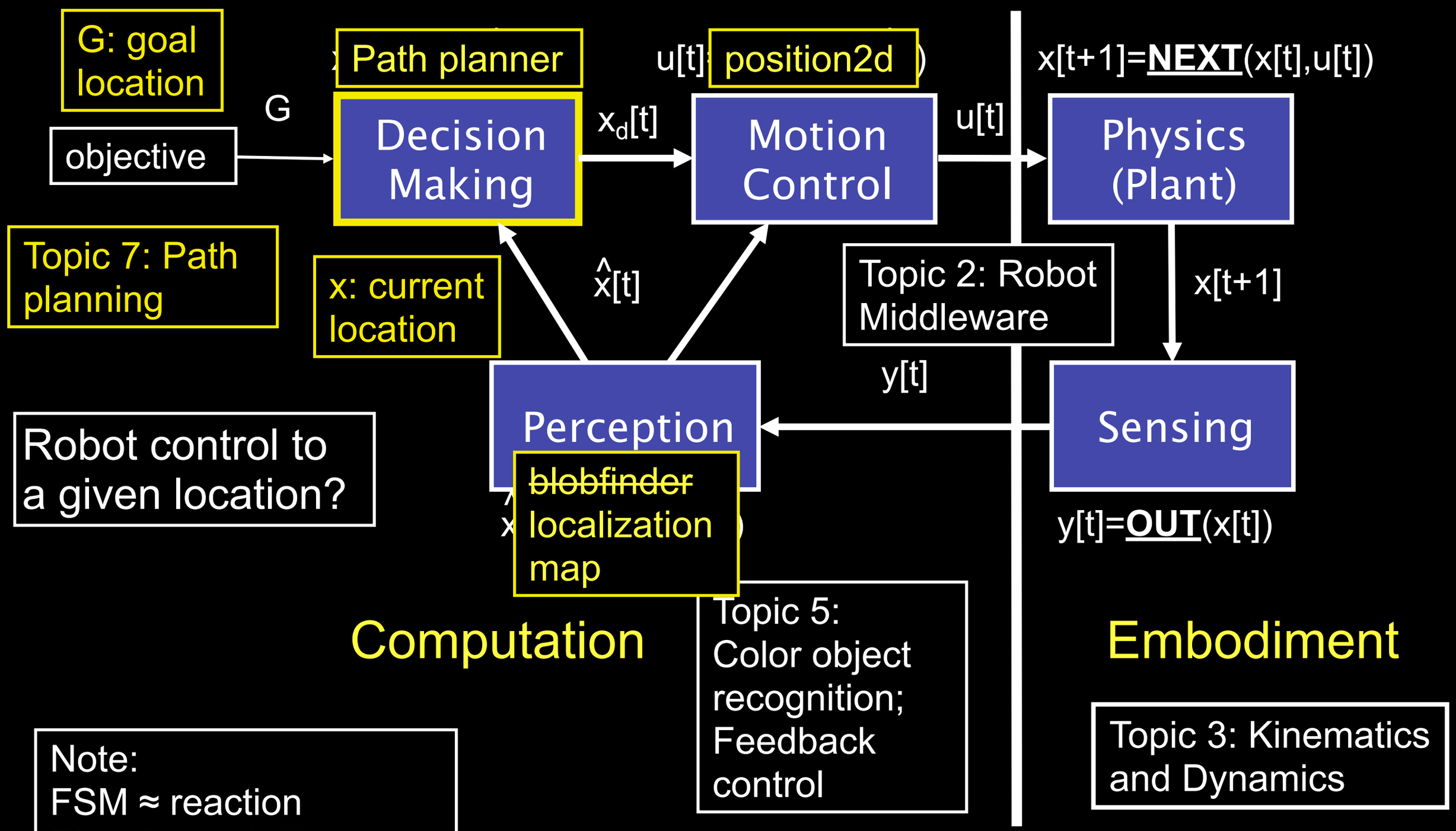
Path planning

the best way to get from A to B

robot control loop

- someone please sketch on the board

The Robot Control Loop



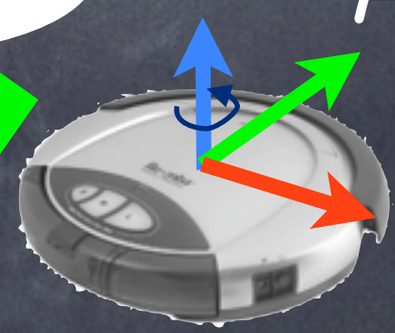
How to get from A to B?

B: Goal

Consider arbitrary
start and goal locations

?????

A: Start



Assume locations are
known (e.g., GPS)

How to get from A to B?

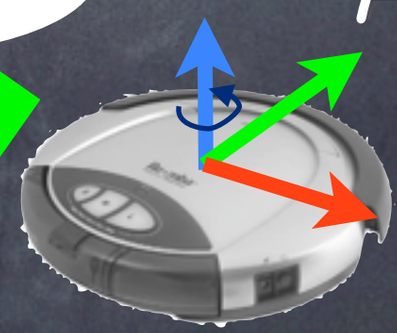
B: Goal

FSM (reactive)

Path planning (deliberative)

?????

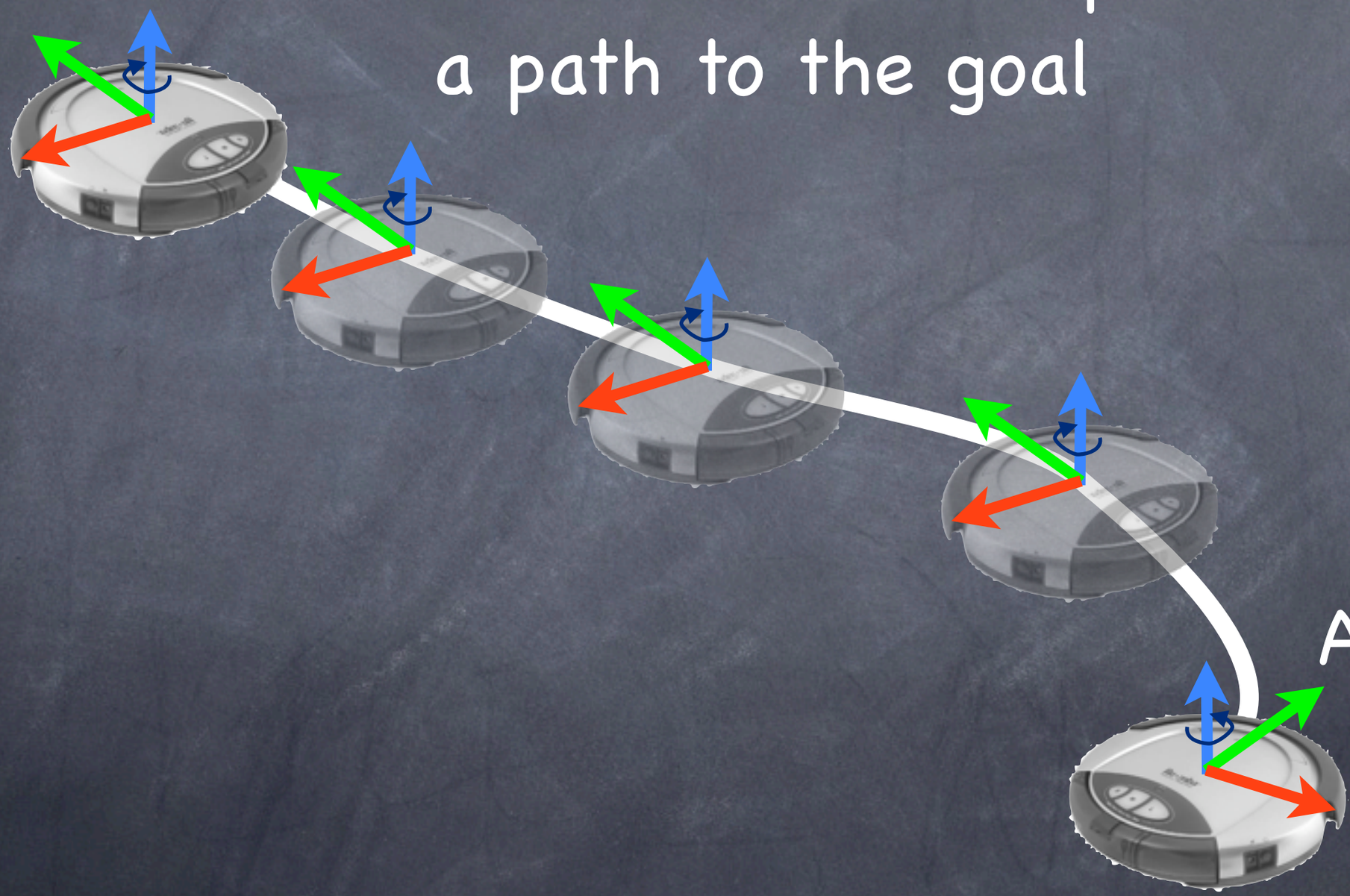
A: Start



Path Planning

B: Goal

Find intermediate poses forming a path to the goal



A: Start

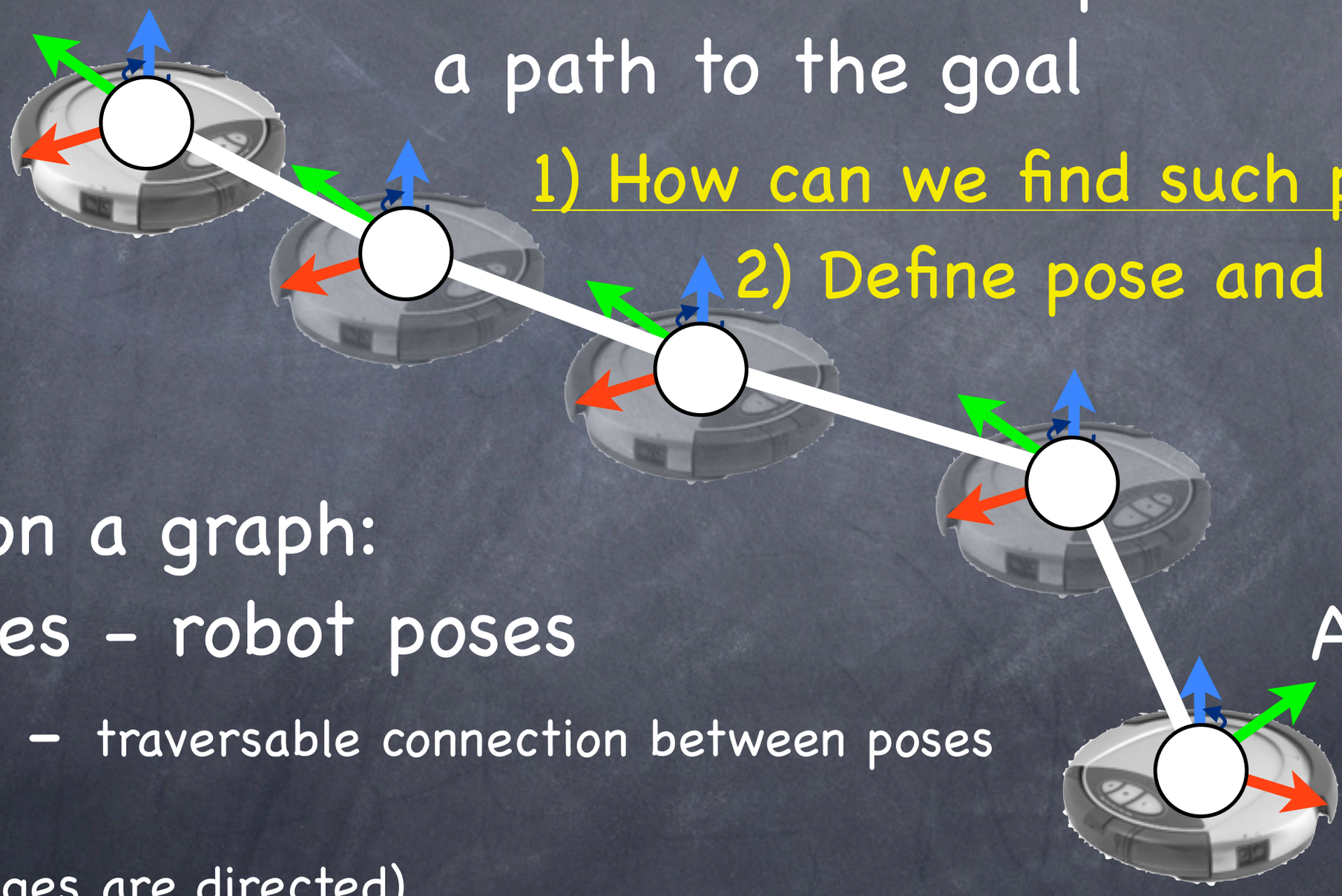
Path Planning

B: Goal

Find intermediate poses forming a path to the goal

1) How can we find such paths?

2) Define pose and controls?



A: Start

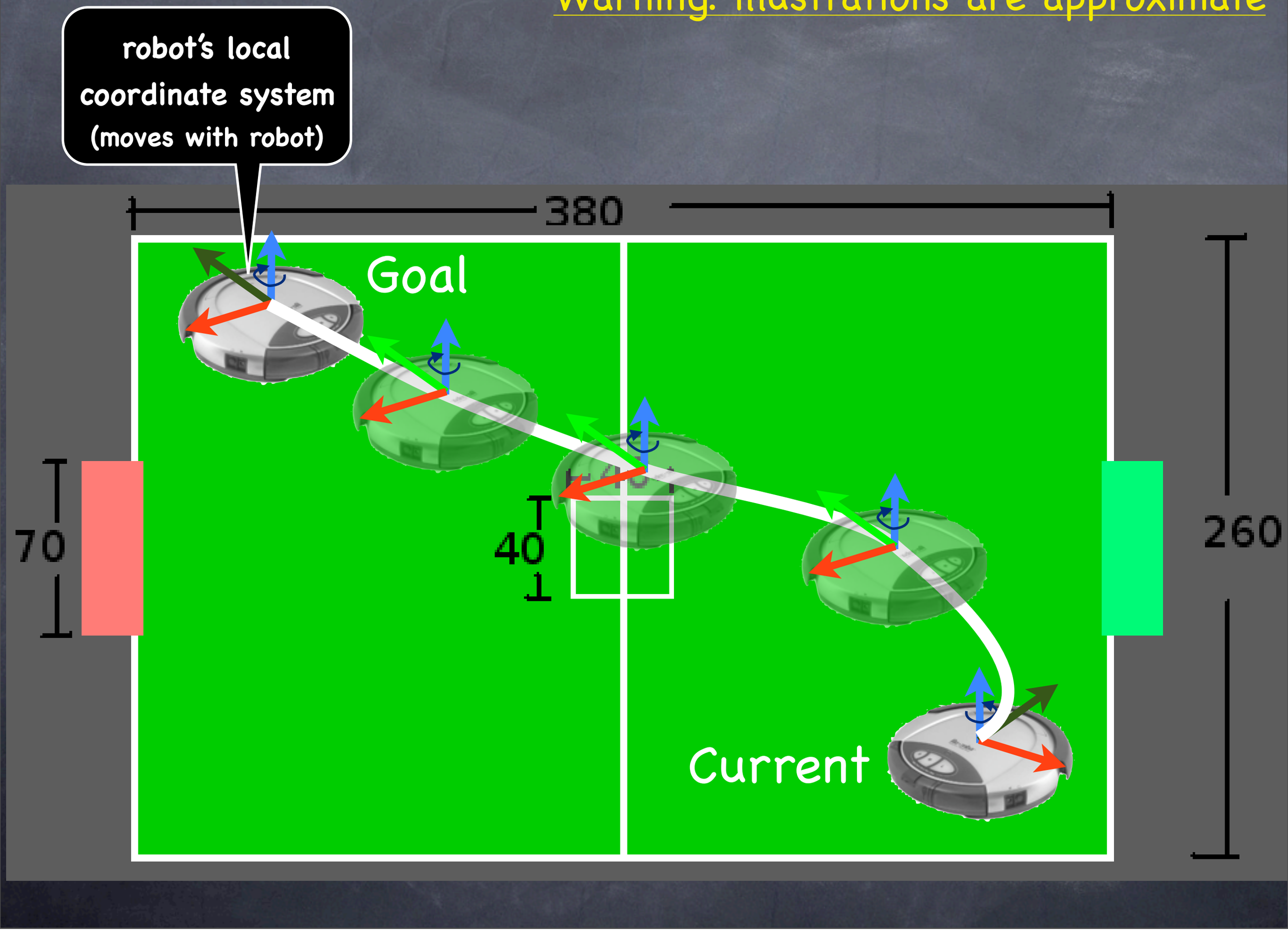
Path on a graph:

vertices - robot poses

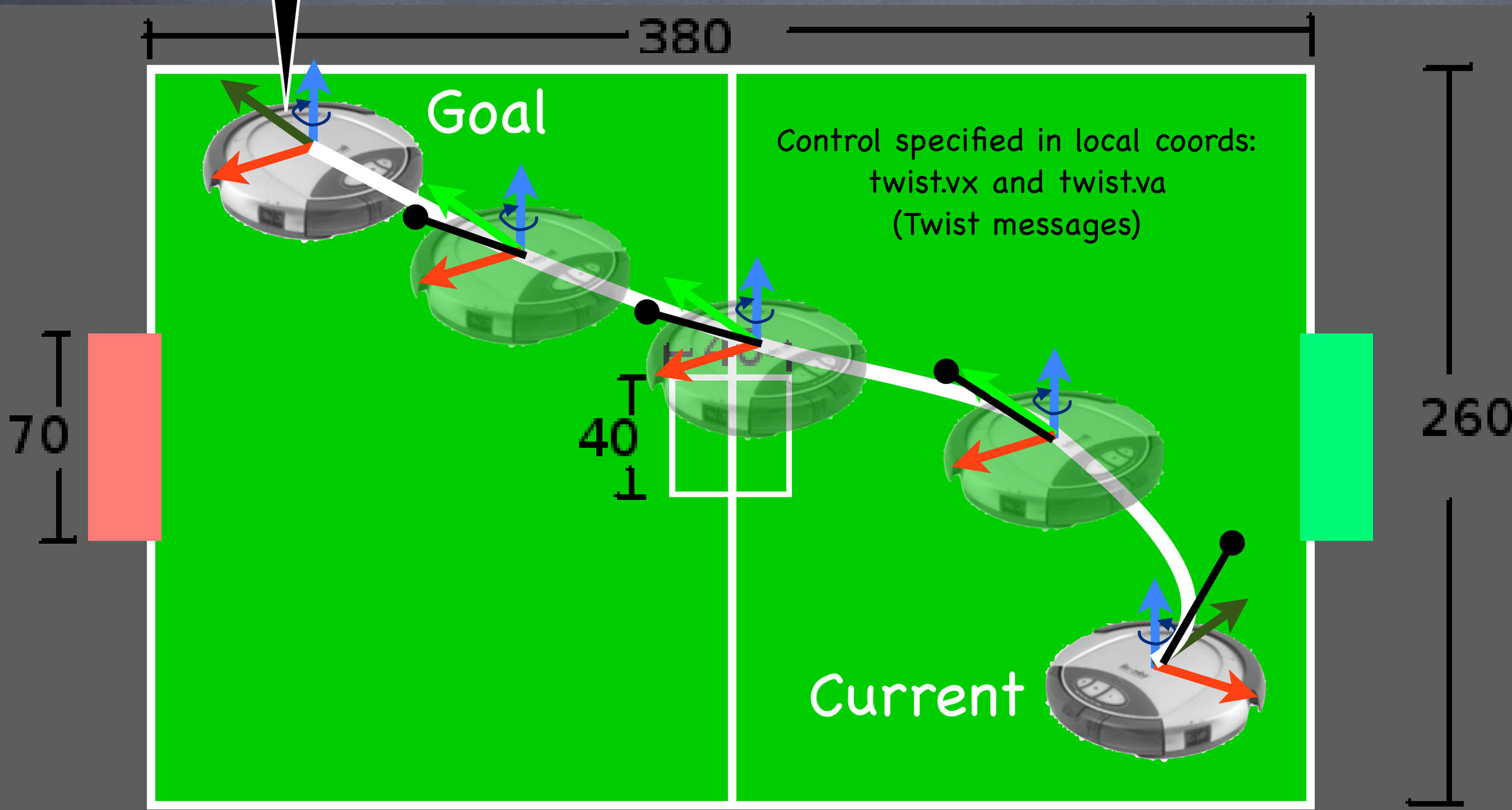
edges - traversable connection between poses

(note: edges are directed)

Warning: illustrations are approximate



robot's local coordinate system (moves with robot)



robot's local coordinate system (moves with robot)

380

Goal

Control specified in local coords:
twist.vx and twist.va
(Twist messages)

70

40

260

Planned path in world coords:
pose defined wrt field

field/global coordinate system (constant)

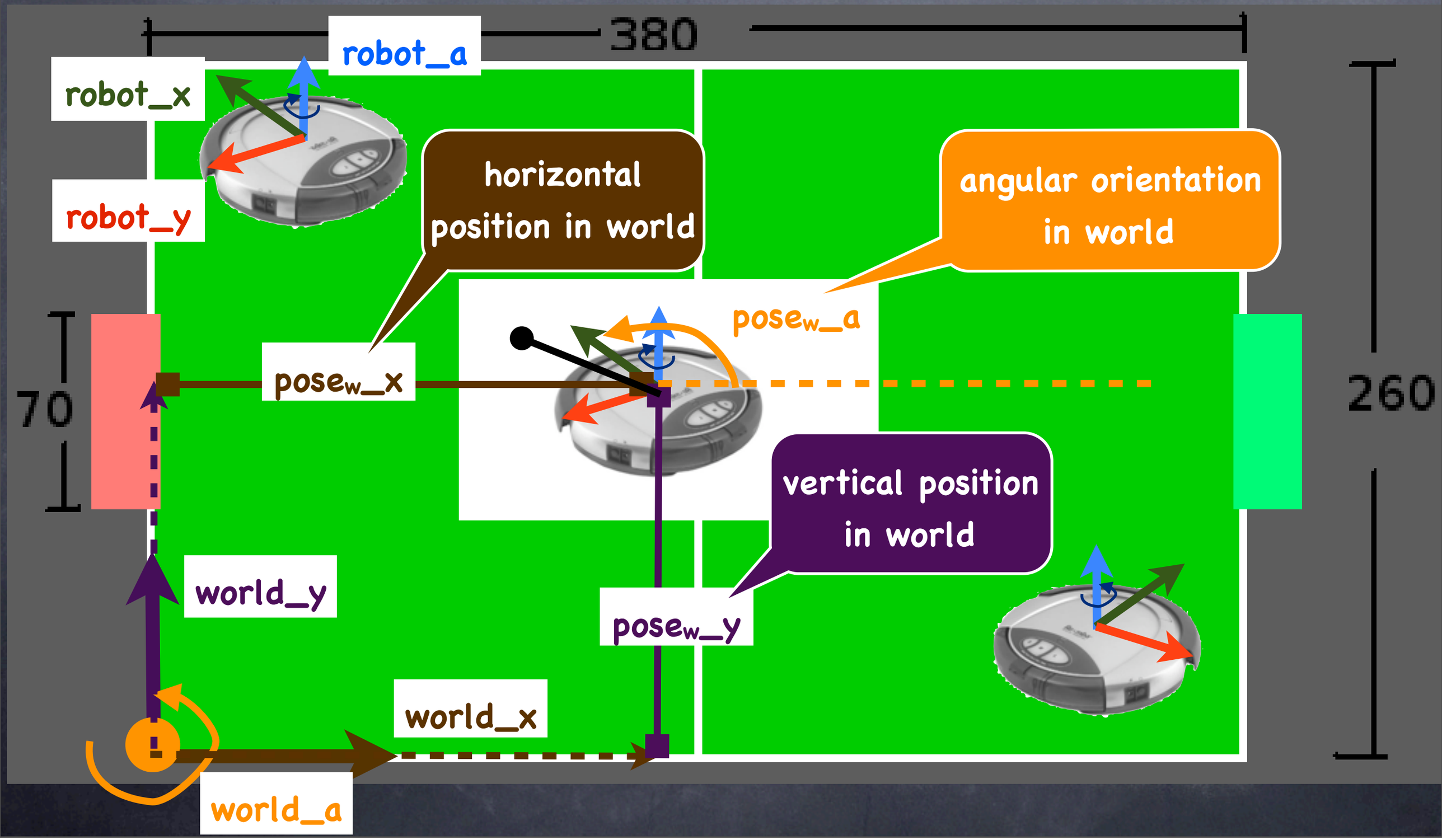
world_y

world_x

world_a

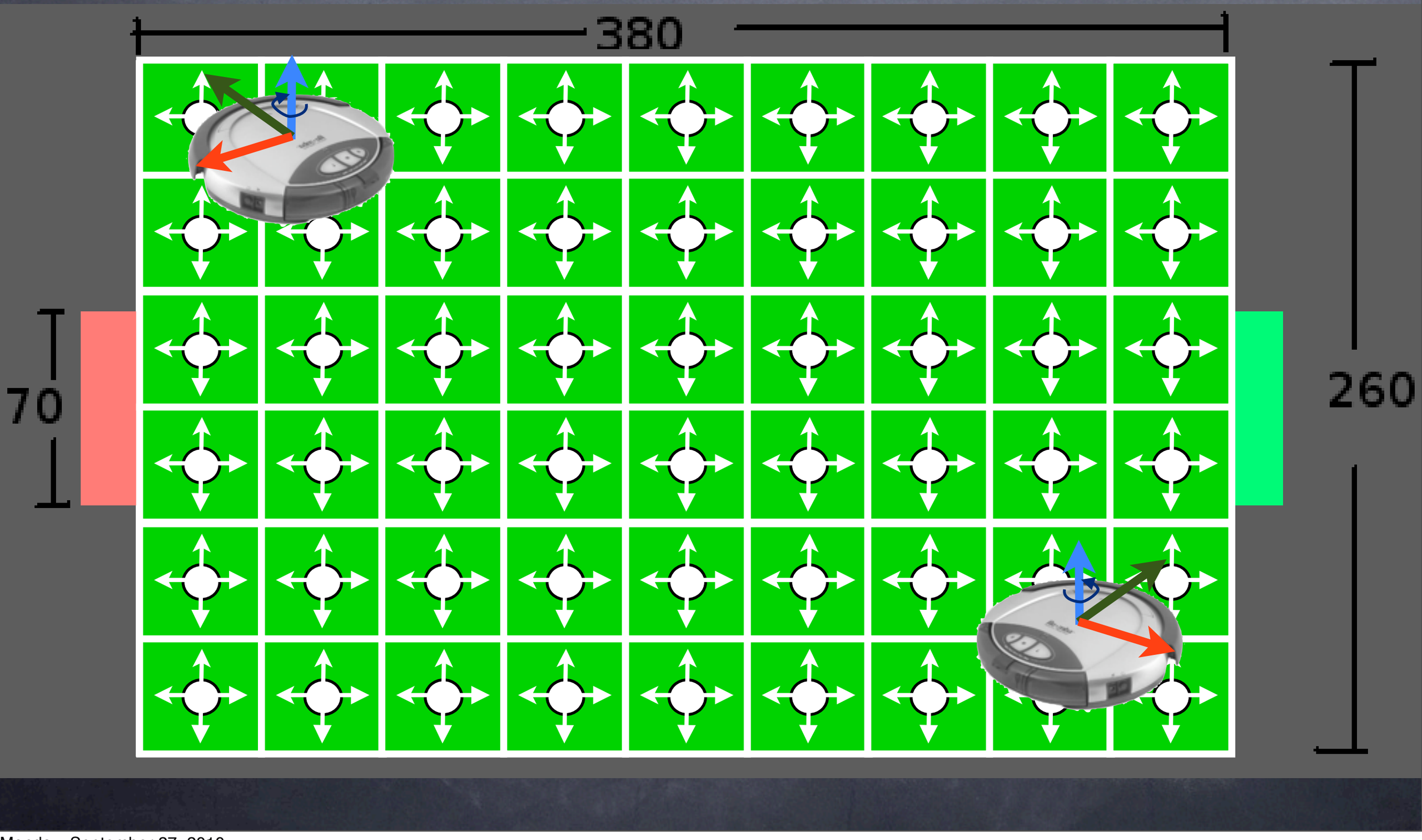
Current

Poses defined w.r.t. the field in 3 dimensions: 2 position DOFs, 1 rotational DOF



Consider all possible poses as graph vertices
Edges connect adjacent (traversable) poses
Edges weighted by distance

How to find a valid path?

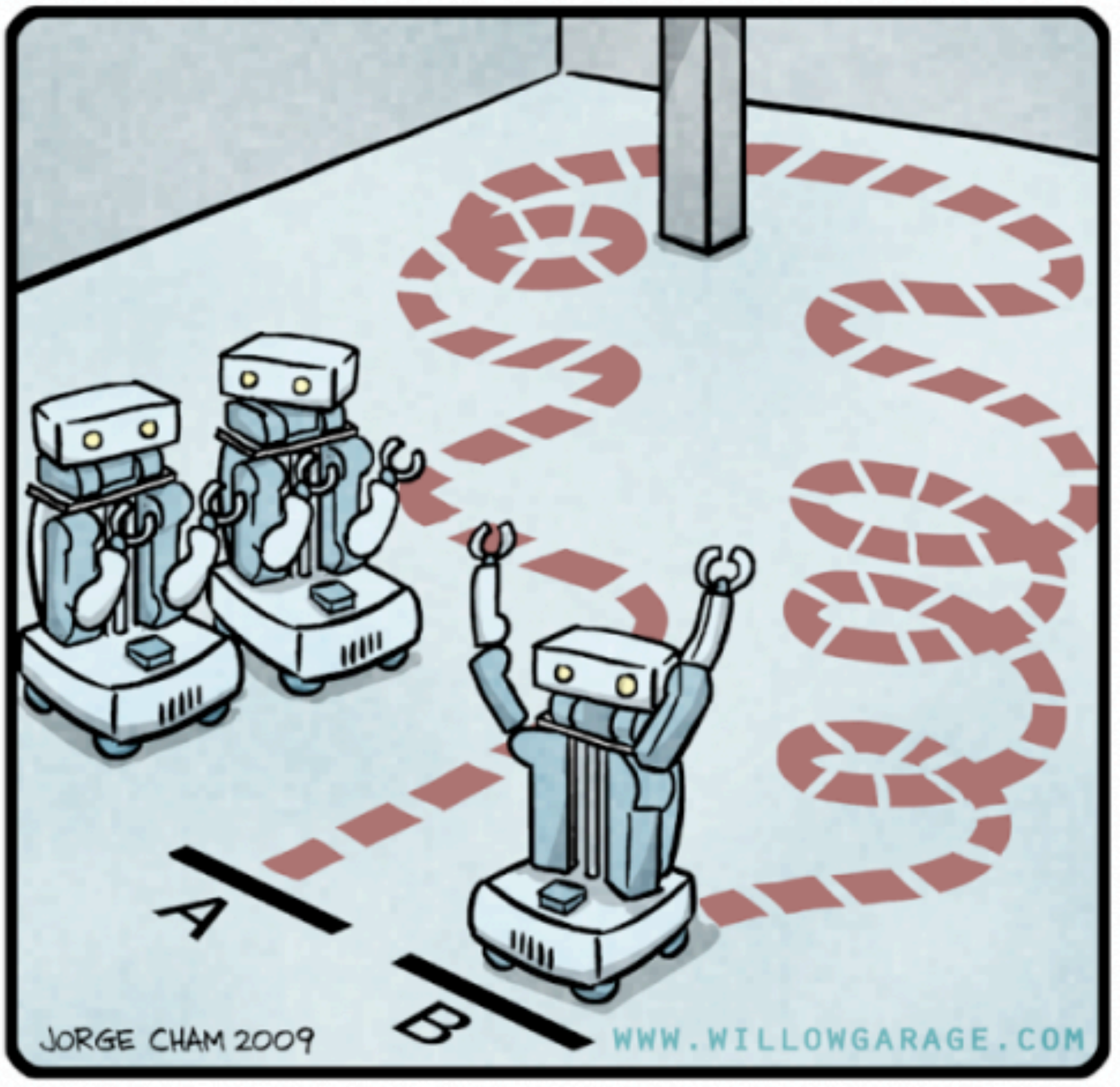


Approaches to path planning

- Search (fixed graph)
  - DFS, BFS, Dijkstra, A*
- Search (build graph):
  - Probabilistic Road Maps
  - Rapidly-exploring Random Trees
- Optimization (local search):
  - Potential fields, gradient descent

What exactly are we searching for?

R.O.B.O.T. Comics

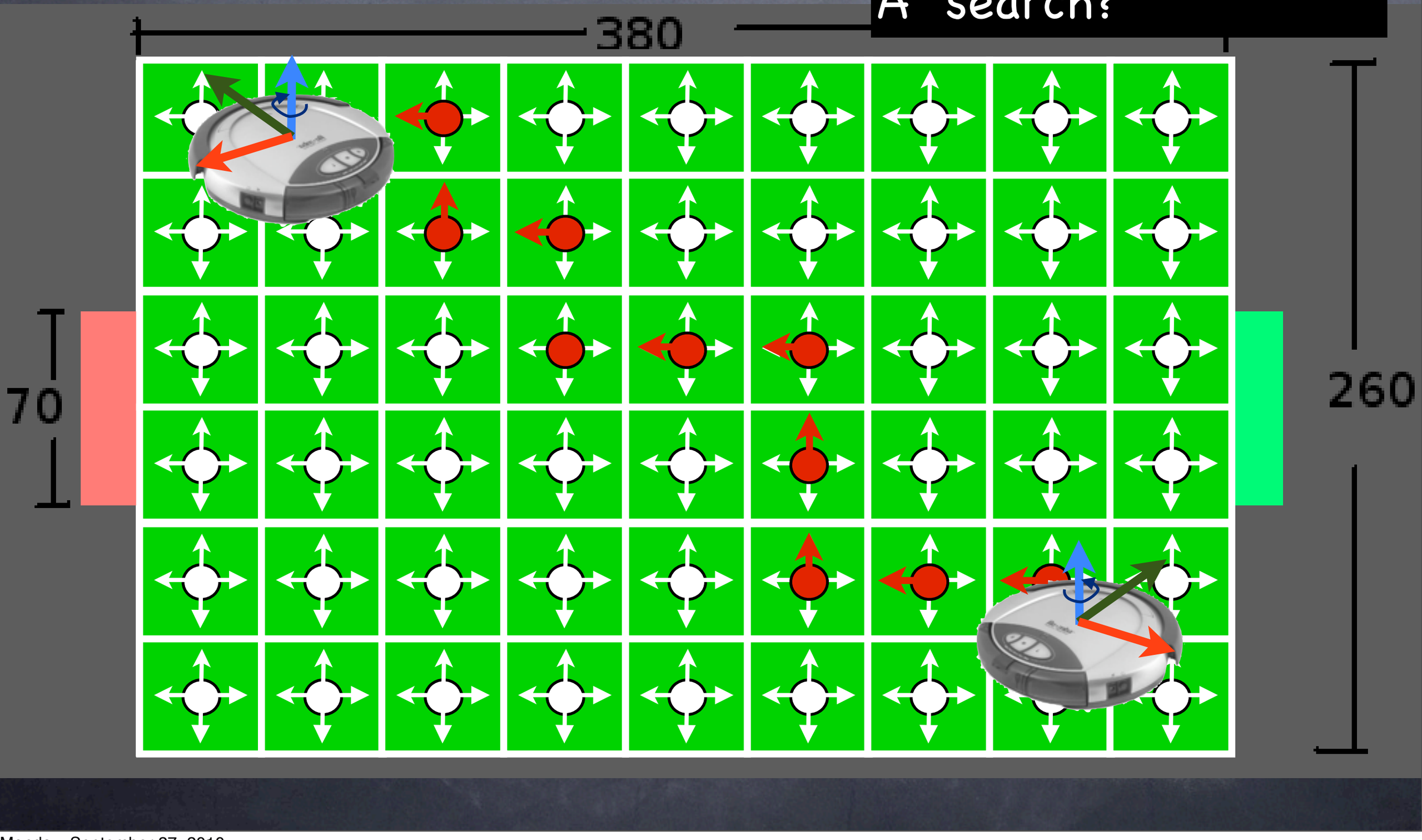


"HIS PATH-PLANNING MAY BE SUB-OPTIMAL, BUT IT'S GOT FLAIR."

Grid Search

Find graph path from start to goal

Depth first search?
Breadth first search?
Dijkstra's algorithm?
A* search?

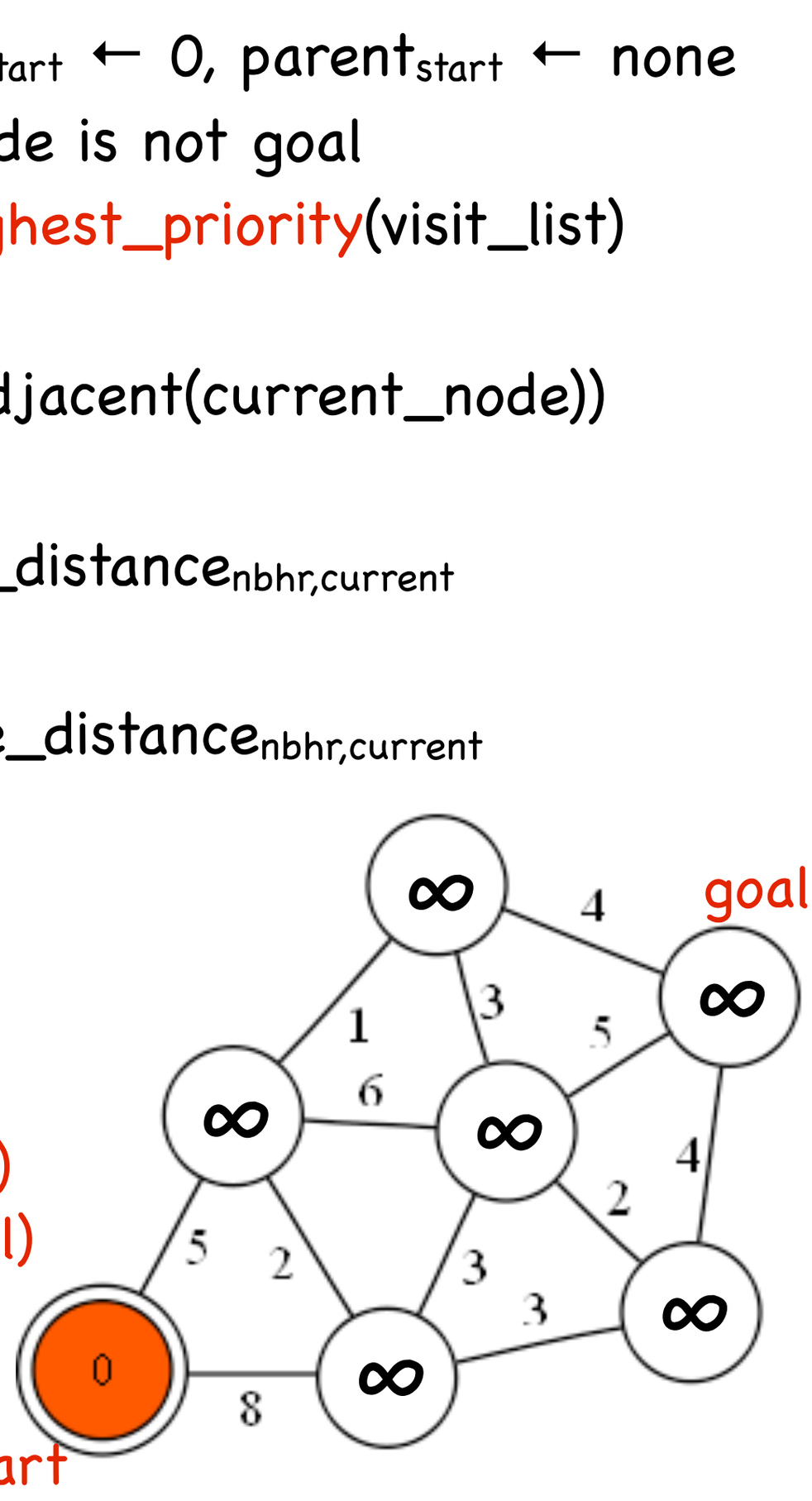


Search algorithm template

```
create visit_list with start node: distance_start ← 0, parent_start ← none
while visit_list not empty and current_node is not goal
  take from visit_list: current_node ← highest_priority(visit_list)
  visited_current ← true
  for each neighbor_node in not_visited(adjacent(current_node))
    add neighbor_node, if not in visit_list
    if distance_neighbor > distance_current + line_distance_nbhr,current
      parent_neighbor ← current_node
      distance_neighbor ← distance_current + line_distance_nbhr,current
  end for loop
end while loop
output ← parent, distance
```

parent: route from any node to start (or goal)
distance: distance along route to start (or goal)

assume $distance_{unlisted}$ = infinity



Depth-first search algorithm (no backtracking)

create visit_stack with start node: $distance_{start} \leftarrow 0$, $parent_{start} \leftarrow$ none

while visit_stack not empty and current_node is not goal

pop from visit_stack: current_node ← most_recent(visit_stack)

$visited_{current} \leftarrow$ true

for each neighbor_node in not_visited(adjacent(current_node))

push neighbor_node, if not in visit_stack

if $distance_{neighbor} > distance_{current} + line\_distance_{nbhr,current}$

$parent_{neighbor} \leftarrow$ current_node

$distance_{neighbor} \leftarrow distance_{current} + line\_distance_{nbhr,current}$

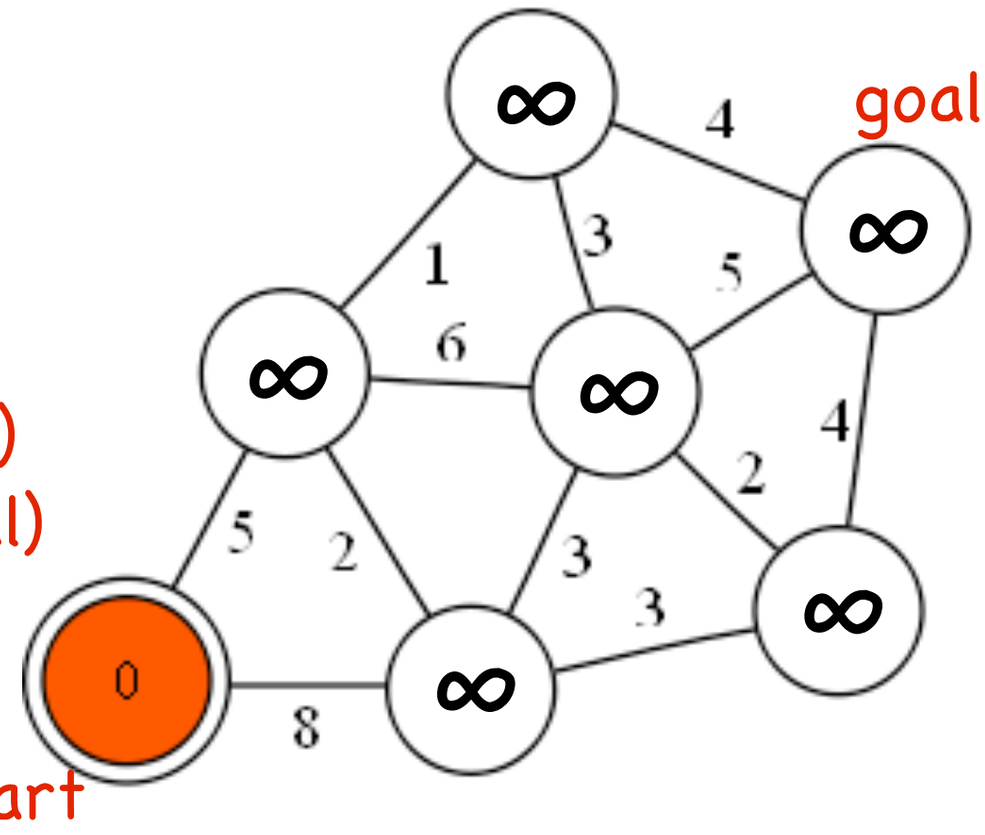
end for loop

end while loop

output ← parent, distance

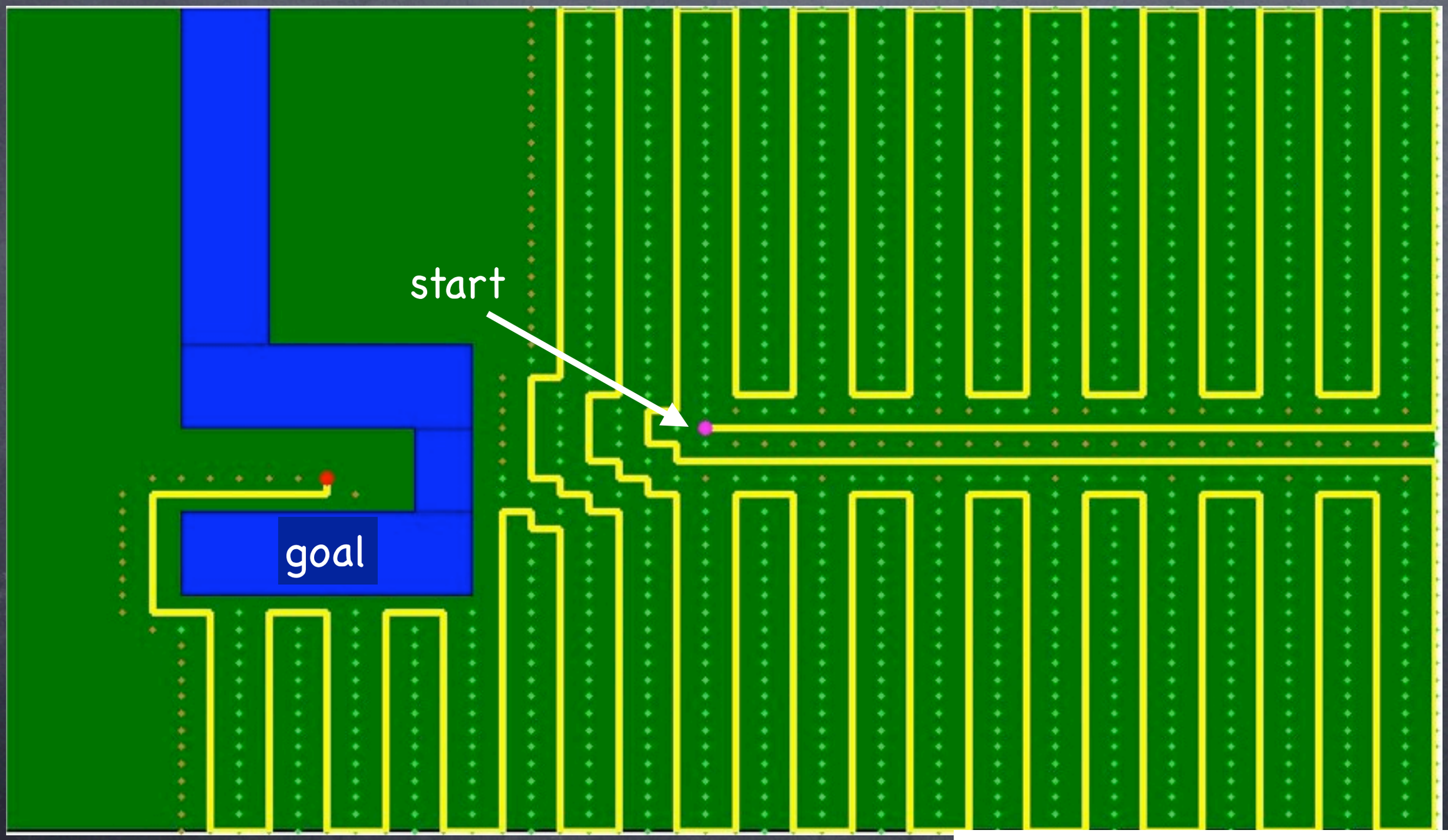
parent: route from any node to start (or goal)

distance: distance along route to start (or goal)



matlab example: DFS

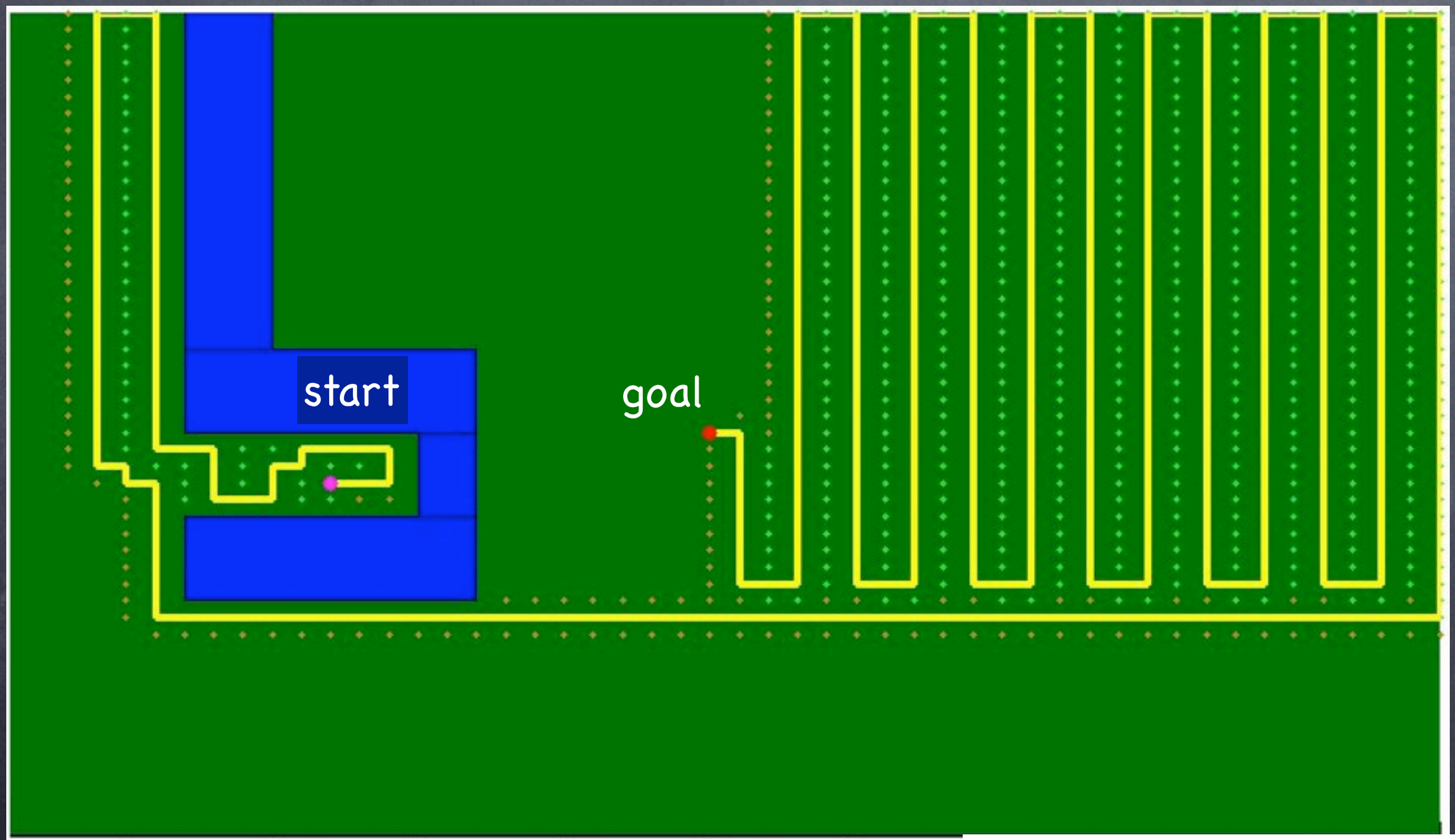
http://www.cs.brown.edu/courses/cs148/pub/pathplan.m



"yellow brick road" graphics by ynm,
visited locations in light green
4-connected grid, right visited first

matlab example: DFS

http://www.cs.brown.edu/courses/cs148/pub/pathplan.m



"yellow brick road" graphics by ynm, visited locations in light green
4-connected grid, right visited first

Breadth-first search algorithm

create visit_queue with start node: $distance_{start} \leftarrow 0$, $parent_{start} \leftarrow$ none

while visit_queue not empty and current_node is not goal

dequeue from visit_queue: current_node ← least_recent(visit_queue)

$visited_{current} \leftarrow$ true

for each neighbor_node in not_visited(adjacent(current_node))

enqueue neighbor_node, if not in visit_queue

if $distance_{neighbor} > distance_{current} + line\_distance_{nbhr,current}$

$parent_{neighbor} \leftarrow$ current_node

$distance_{neighbor} \leftarrow distance_{current} + line\_distance_{nbhr,current}$

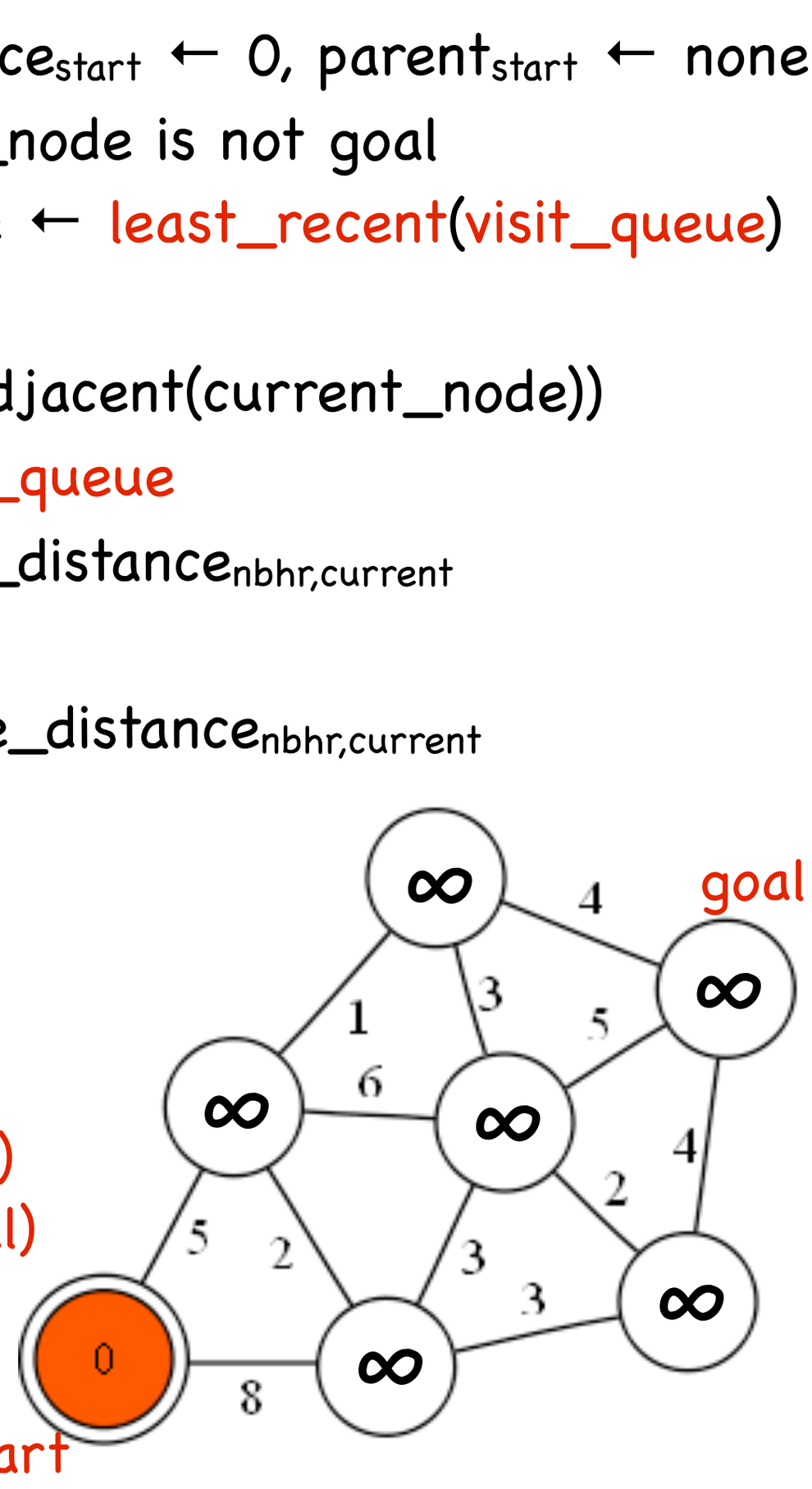
end for loop

end while loop

output ← parent, distance

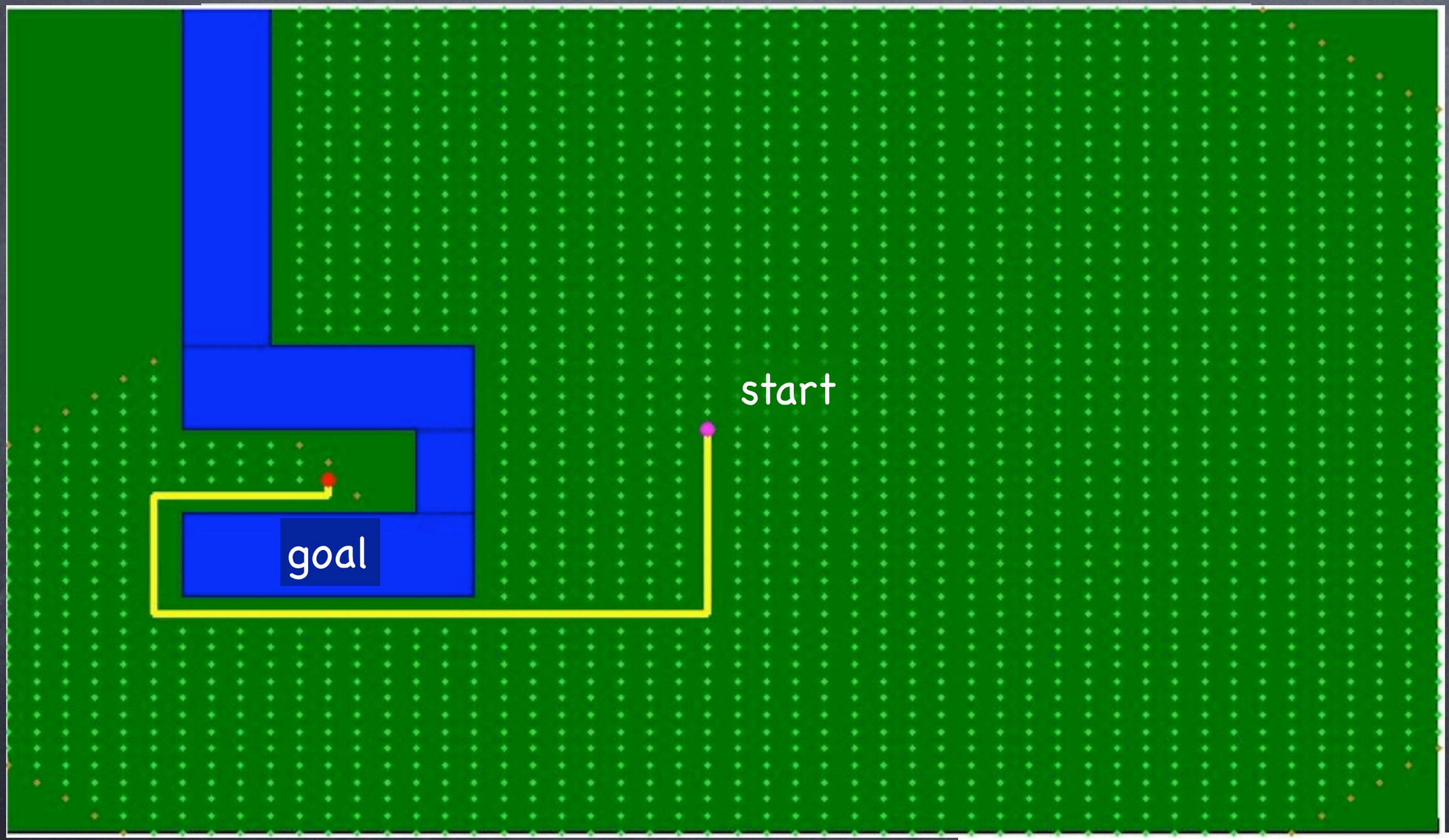
parent: route from any node to start (or goal)

distance: distance along route to start (or goal)



matlab example: BFS

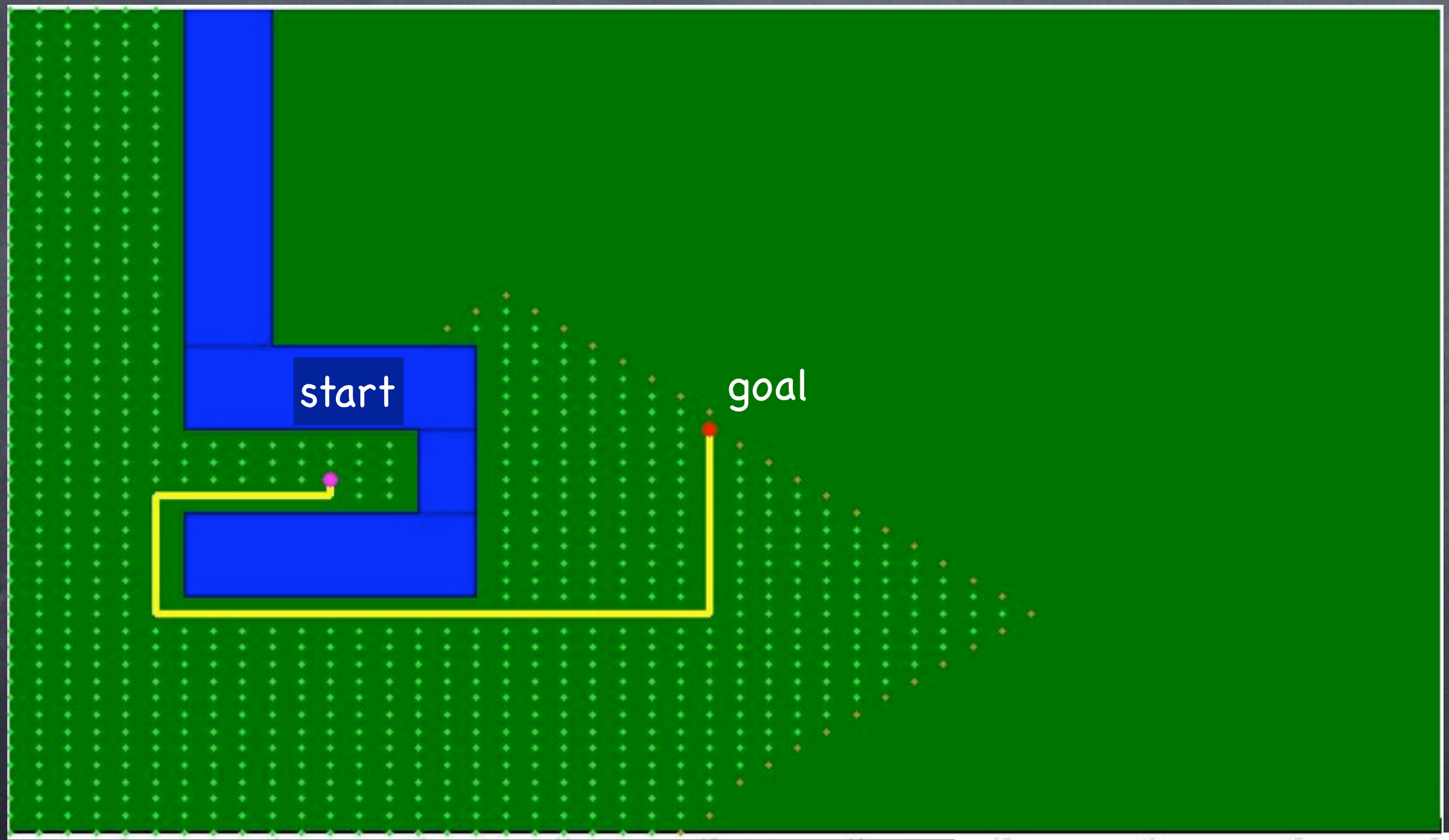
http://www.cs.brown.edu/courses/cs148/pub/pathplan.m



"yellow brick road" graphics by ynm, visited locations in light green
4-connected grid

matlab example: BFS

http://www.cs.brown.edu/courses/cs148/pub/pathplan.m

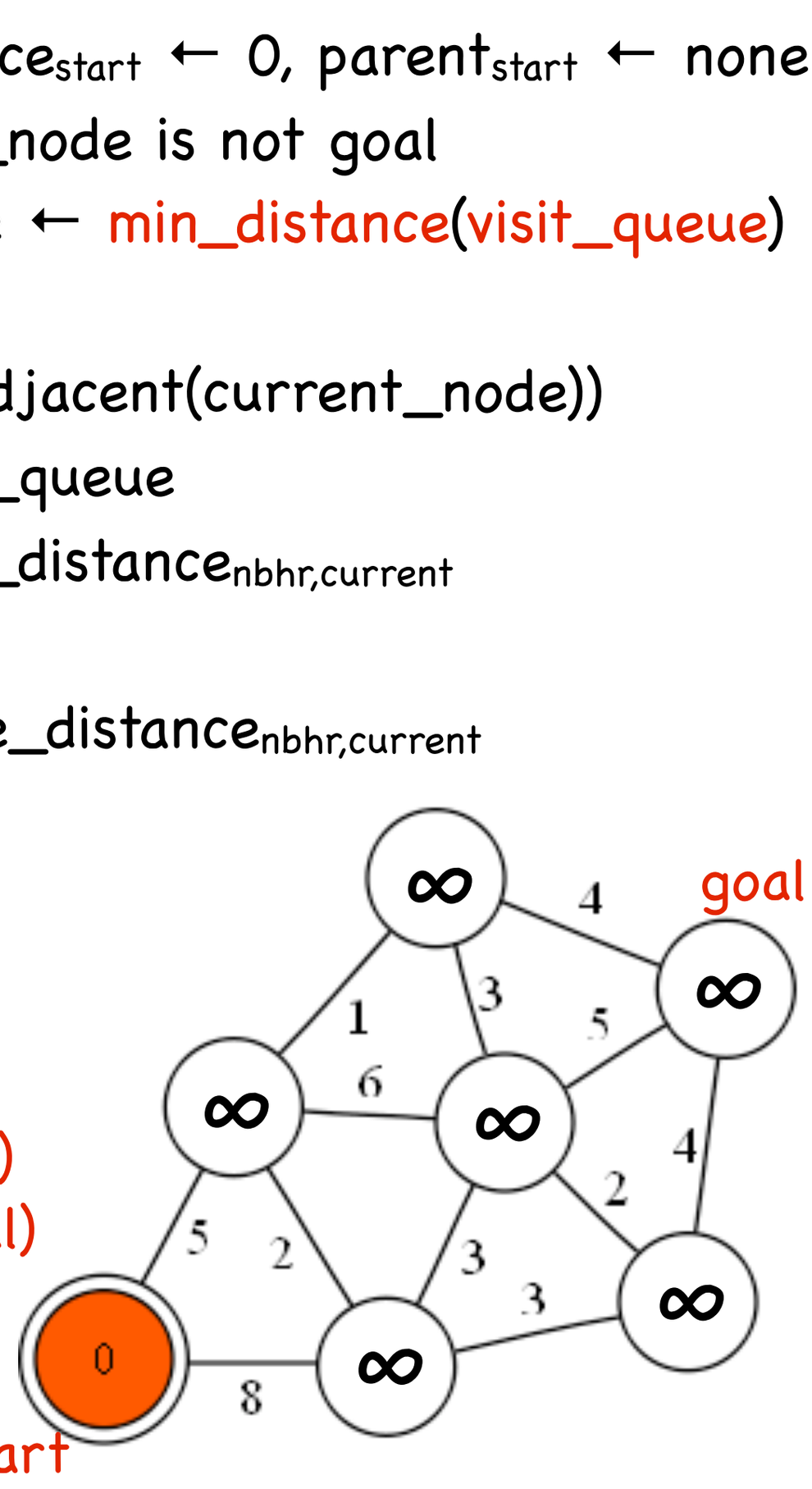


"yellow brick road" graphics by ynm, visited locations in light green
4-connected grid

Dijkstra shortest path algorithm

create visit_queue with start node: $distance_{start} \leftarrow 0$, $parent_{start} \leftarrow$ none
while visit_queue not empty and current_node is not goal
  dequeue from visit_queue: current_node ← min_distance(visit_queue)
  $visited_{current} \leftarrow$ true
  for each neighbor_node in not_visited(adjacent(current_node))
    enqueue neighbor_node, if not in visit_queue
    if $distance_{neighbor} > distance_{current} + line\_distance_{nbhr,current}$
      $parent_{neighbor} \leftarrow$ current_node
      $distance_{neighbor} \leftarrow distance_{current} + line\_distance_{nbhr,current}$
  end for loop
end while loop
output ← parent, distance

parent: route from any node to start (or goal)
distance: distance along route to start (or goal)



Is Dijkstra different than BFS?

Dijkstra shortest path algorithm

create visit_queue with start node: $distance_{start} \leftarrow 0$, $parent_{start} \leftarrow$ none

while visit_queue not empty and current_node is not goal

dequeue from visit_queue: current_node ← min_distance(visit_queue)

$visited_{current} \leftarrow$ true

for each neighbor_node in not_visited(adjacent(current_node))

enqueue neighbor_node, if not in visit_queue

if $distance_{neighbor} > distance_{current} + line\_distance_{nbhr,current}$

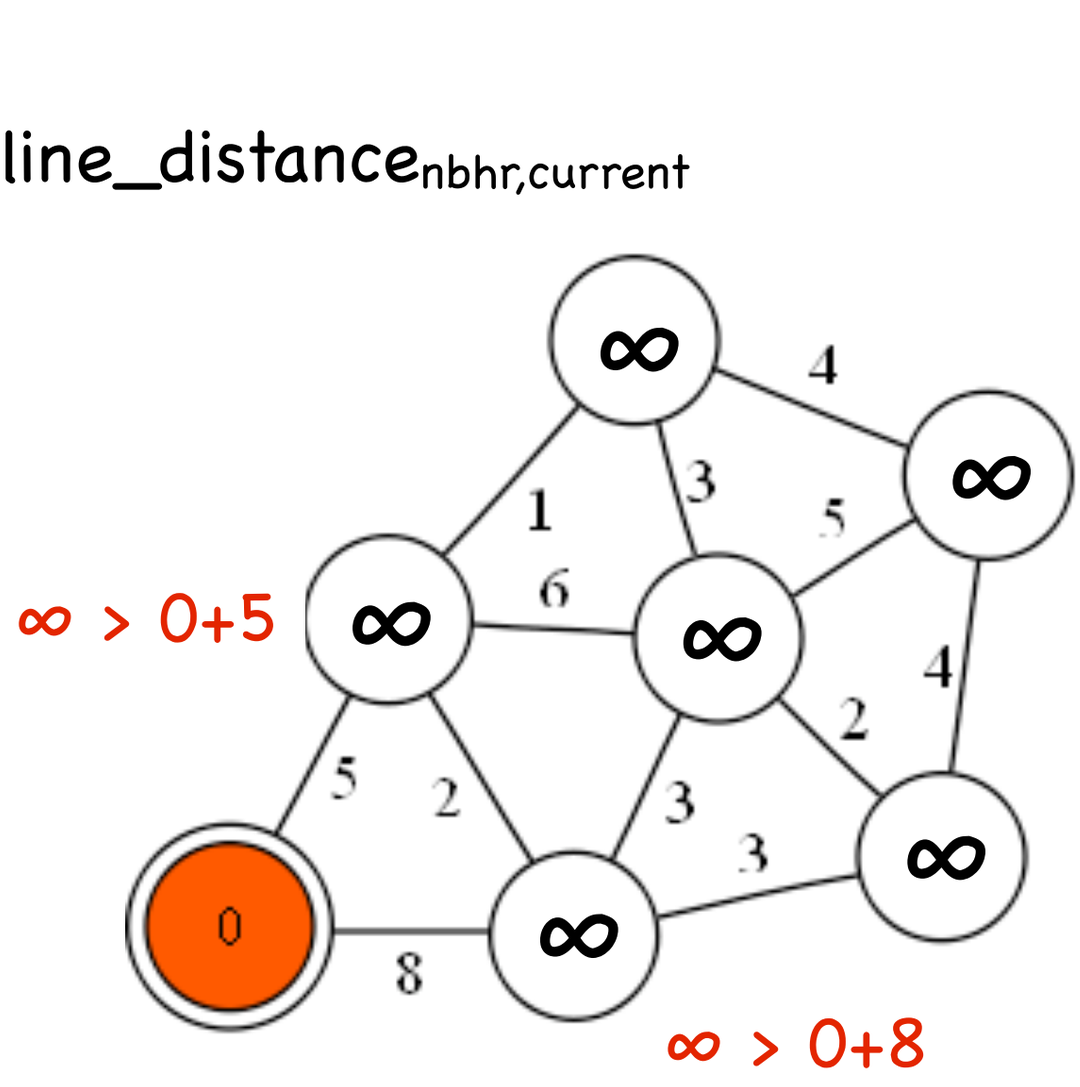
$parent_{neighbor} \leftarrow$ current_node

$distance_{neighbor} \leftarrow distance_{current} + line\_distance_{nbhr,current}$

end for loop

end while loop

output ← parent, distance



Dijkstra shortest path algorithm

create visit_queue with start node: $distance_{start} \leftarrow 0$, $parent_{start} \leftarrow$ none

while visit_queue not empty and current_node is not goal

dequeue from visit_queue: current_node ← min_distance(visit_queue)

$visited_{current} \leftarrow$ true

for each neighbor_node in not_visited(adjacent(current_node))

enqueue neighbor_node, if not in visit_queue

if $distance_{neighbor} > distance_{current} + line\_distance_{nbhr,current}$

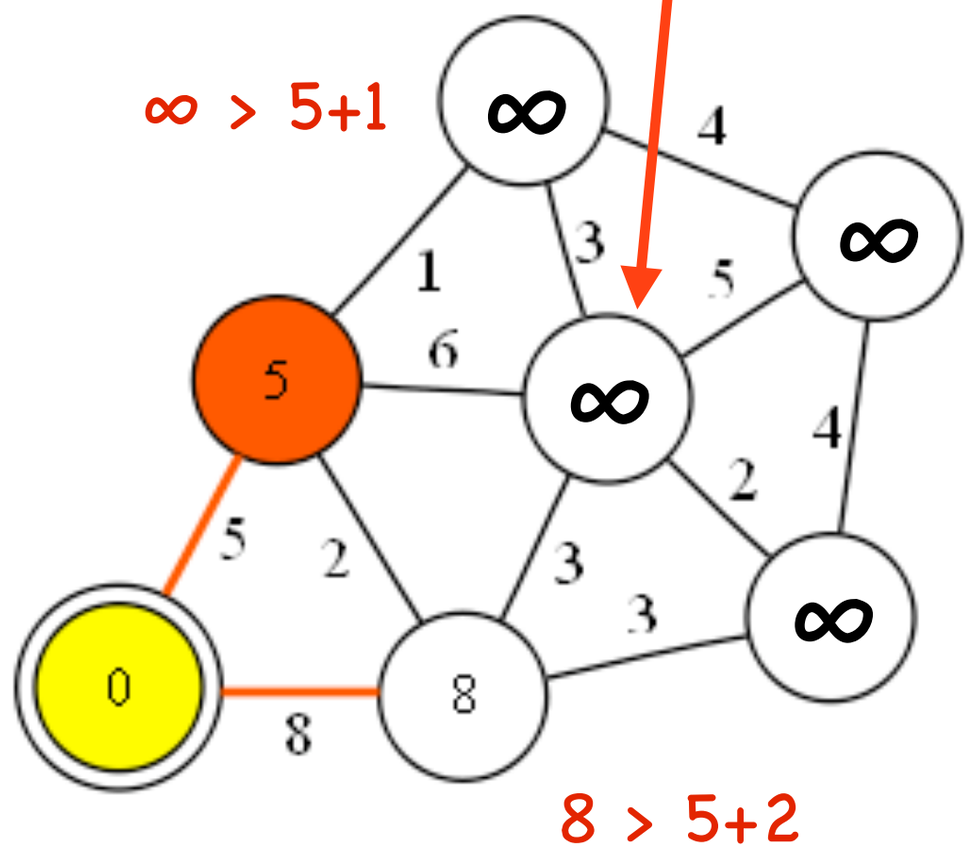
$parent_{neighbor} \leftarrow$ current_node

$distance_{neighbor} \leftarrow distance_{current} + line\_distance_{nbhr,current}$ ∞ > 5+6

end for loop

end while loop

output ← parent, distance



Dijkstra shortest path algorithm

create visit_queue with start node: $distance_{start} \leftarrow 0$, $parent_{start} \leftarrow$ none

while visit_queue not empty and current_node is not goal

dequeue from visit_queue: current_node ← min_distance(visit_queue)

$visited_{current} \leftarrow$ true

for each neighbor_node in not_visited(adjacent(current_node))

enqueue neighbor_node, if not in visit_queue

if $distance_{neighbor} > distance_{current} + line\_distance_{nbhr,current}$

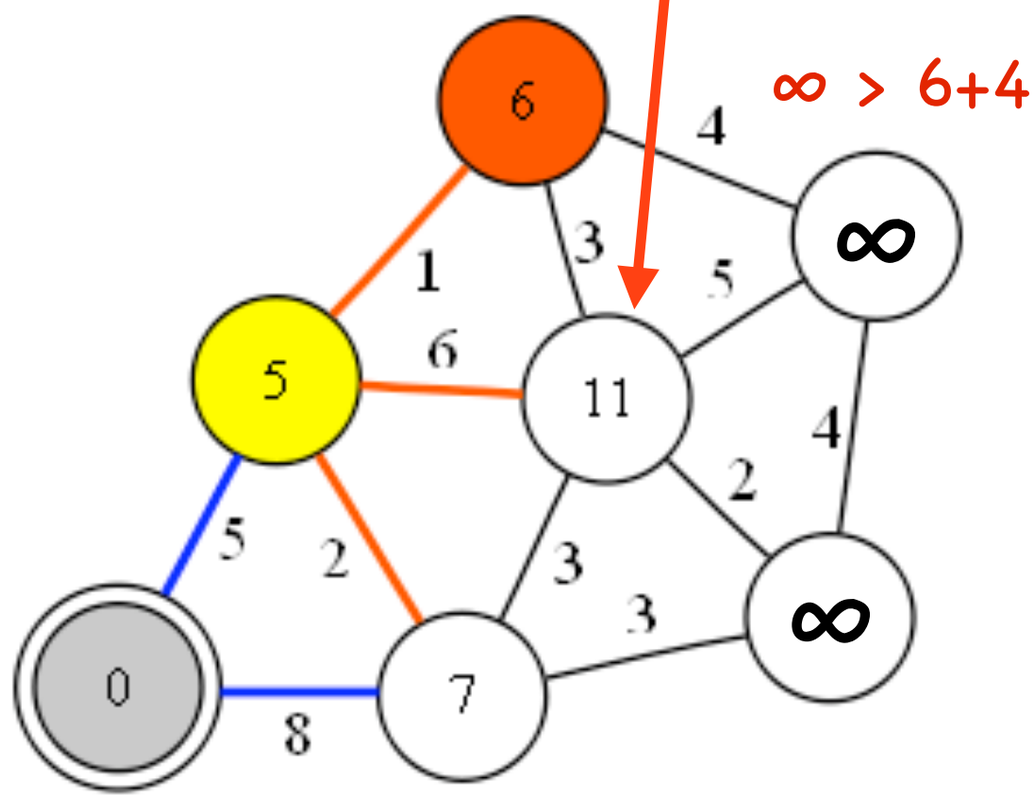
$parent_{neighbor} \leftarrow$ current_node

$distance_{neighbor} \leftarrow distance_{current} + line\_distance_{nbhr,current}$ 11 > 6+3

end for loop

end while loop

output ← parent, distance



Dijkstra shortest path algorithm

create visit_queue with start node: $distance_{start} \leftarrow 0$, $parent_{start} \leftarrow$ none

while visit_queue not empty and current_node is not goal

dequeue from visit_queue: current_node ← min_distance(visit_queue)

$visited_{current} \leftarrow$ true

for each neighbor_node in not_visited(adjacent(current_node))

enqueue neighbor_node, if not in visit_queue

if $distance_{neighbor} > distance_{current} + line\_distance_{nbhr,current}$

$parent_{neighbor} \leftarrow$ current_node

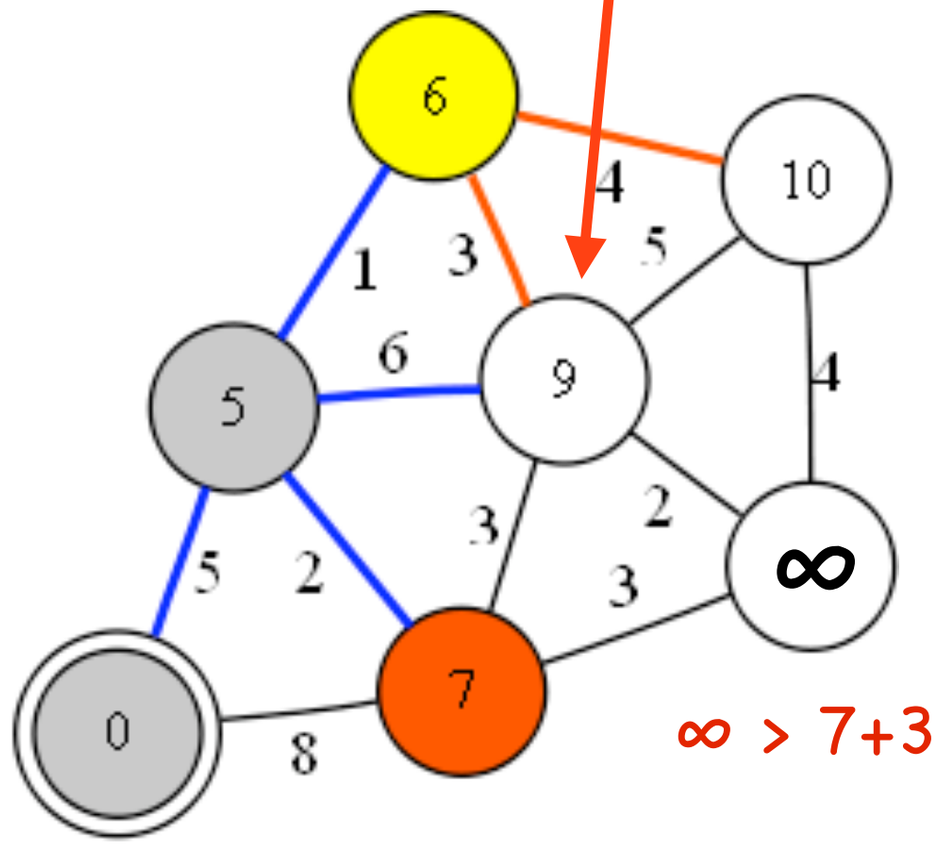
9 !> 7+3

$distance_{neighbor} \leftarrow distance_{current} + line\_distance_{nbhr,current}$

end for loop

end while loop

output ← parent, distance



Dijkstra shortest path algorithm

create visit_queue with start node: $distance_{start} \leftarrow 0$, $parent_{start} \leftarrow none$

while visit_queue not empty and current_node is not goal

dequeue from visit_queue: current_node ← min_distance(visit_queue)

$visited_{current} \leftarrow true$

for each neighbor_node in not_visited(adjacent(current_node))

enqueue neighbor_node, if not in visit_queue

if $distance_{neighbor} > distance_{current} + line\_distance_{nbhr,current}$

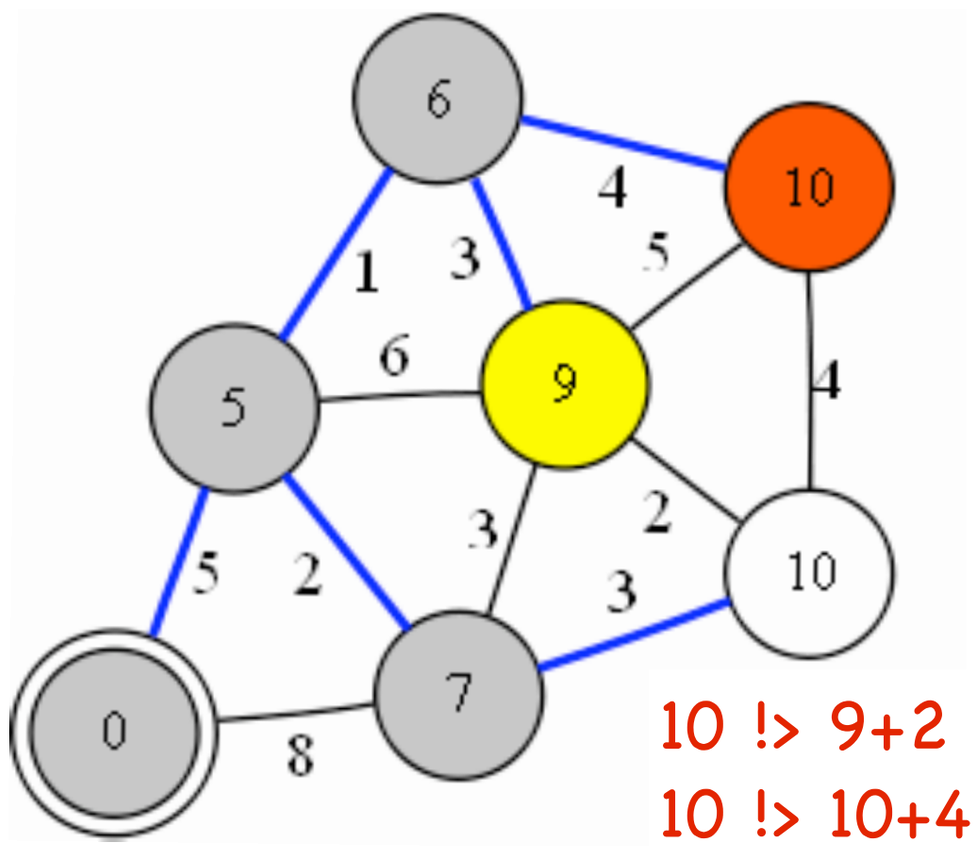
$parent_{neighbor} \leftarrow current\_node$

$distance_{neighbor} \leftarrow distance_{current} + line\_distance_{nbhr,current}$

end for loop

end while loop

output ← parent, distance



Dijkstra shortest path algorithm

create visit_queue with start node: $distance_{start} \leftarrow 0$, $parent_{start} \leftarrow$ none

while visit_queue not empty and current_node is not goal

dequeue from visit_queue: current_node ← min_distance(visit_queue)

$visited_{current} \leftarrow$ true

for each neighbor_node in not_visited(adjacent(current_node))

enqueue neighbor_node, if not in visit_queue

if $distance_{neighbor} > distance_{current} + line\_distance_{nbhr,current}$

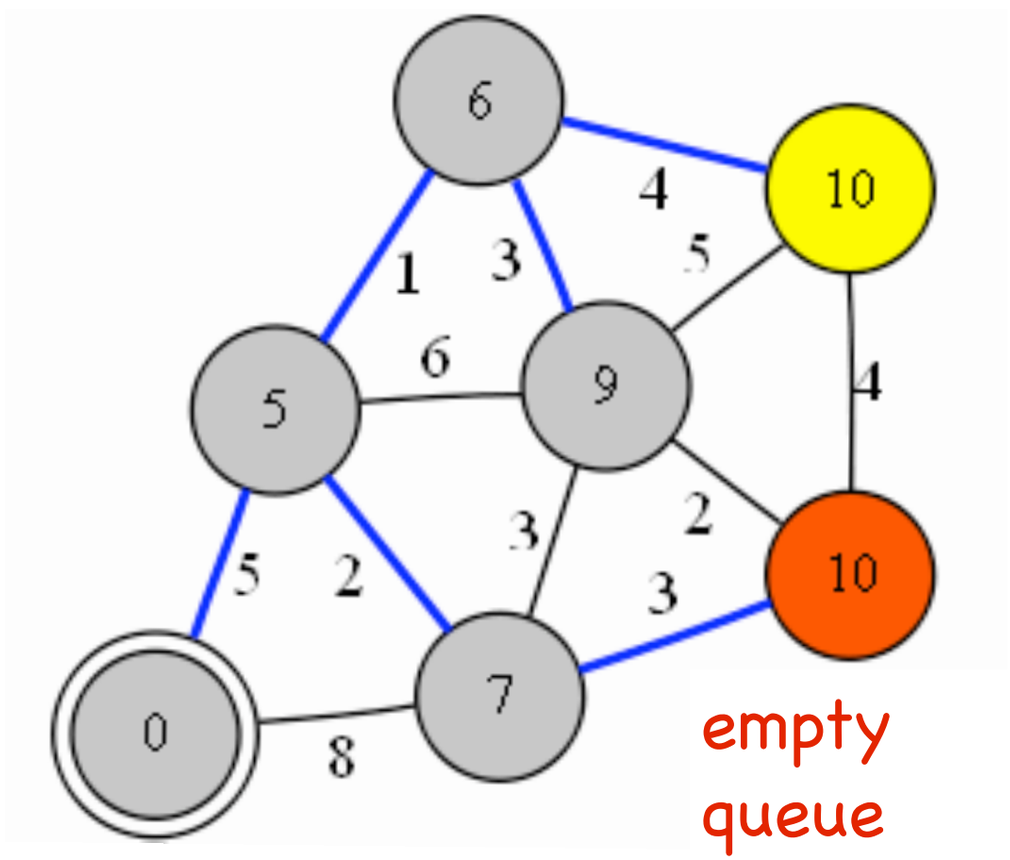
$parent_{neighbor} \leftarrow$ current_node

$distance_{neighbor} \leftarrow distance_{current} + line\_distance_{nbhr,current}$

end for loop

end while loop

output ← parent, distance



Dijkstra shortest path algorithm

create visit_queue with start node: $distance_{start} \leftarrow 0$, $parent_{start} \leftarrow$ none

while visit_queue not empty and current_node is not goal

dequeue from visit_queue: current_node ← min_distance(visit_queue)

$visited_{current} \leftarrow$ true

for each neighbor_node in not_visited(adjacent(current_node))

enqueue neighbor_node, if not in visit_queue

if $distance_{neighbor} > distance_{current} + line\_distance_{nbhr,current}$

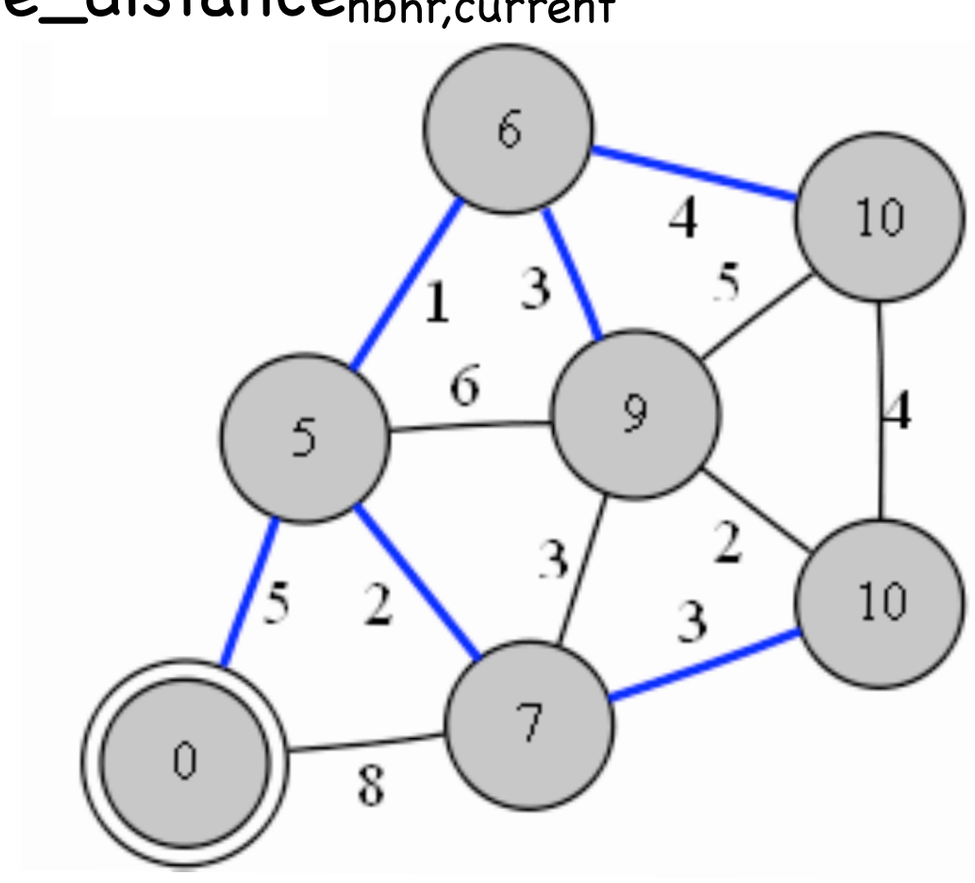
$parent_{neighbor} \leftarrow$ current_node

$distance_{neighbor} \leftarrow distance_{current} + line\_distance_{nbhr,current}$

end for loop

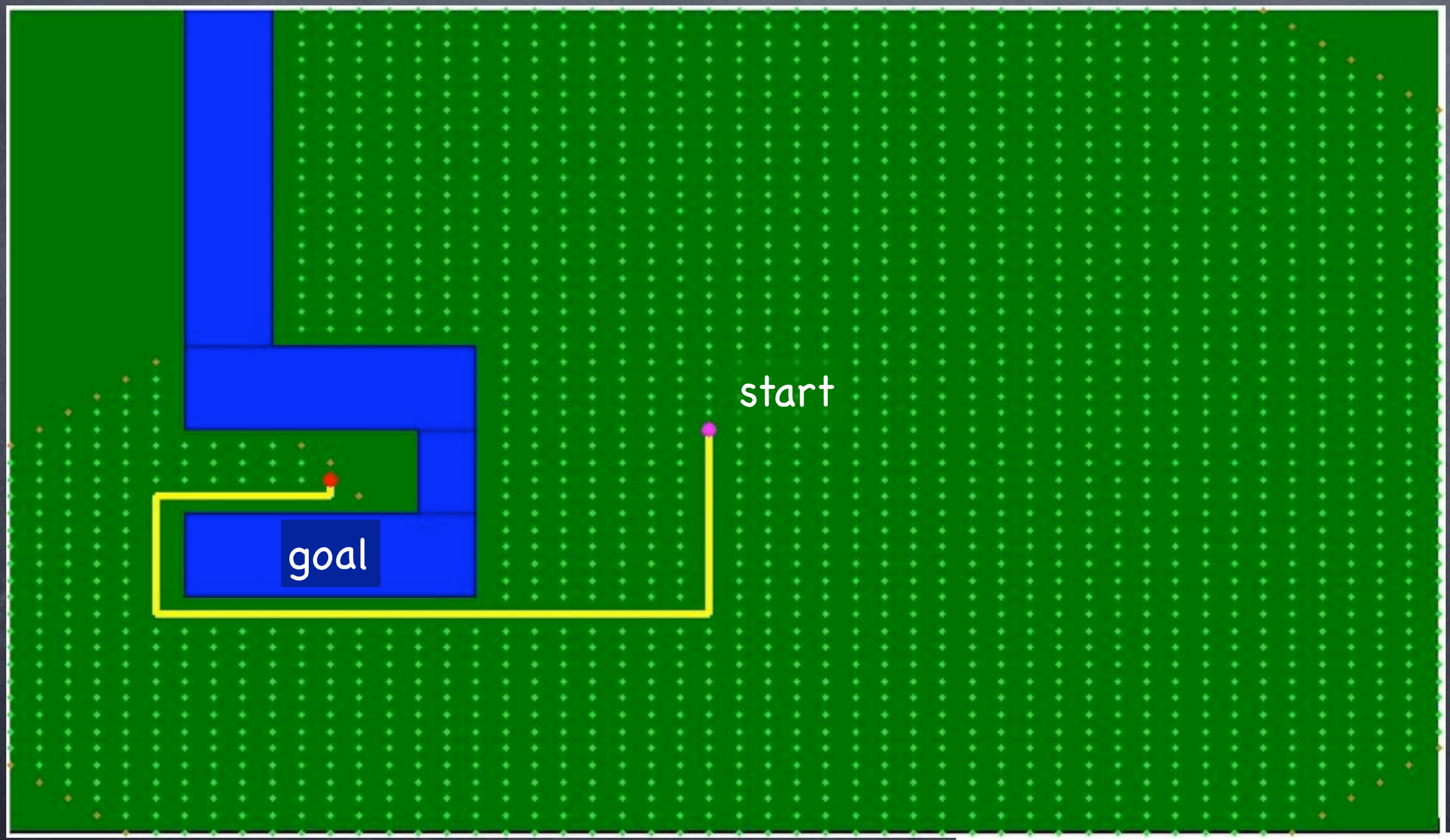
end while loop

output ← parent, distance



matlab example: Dijkstra

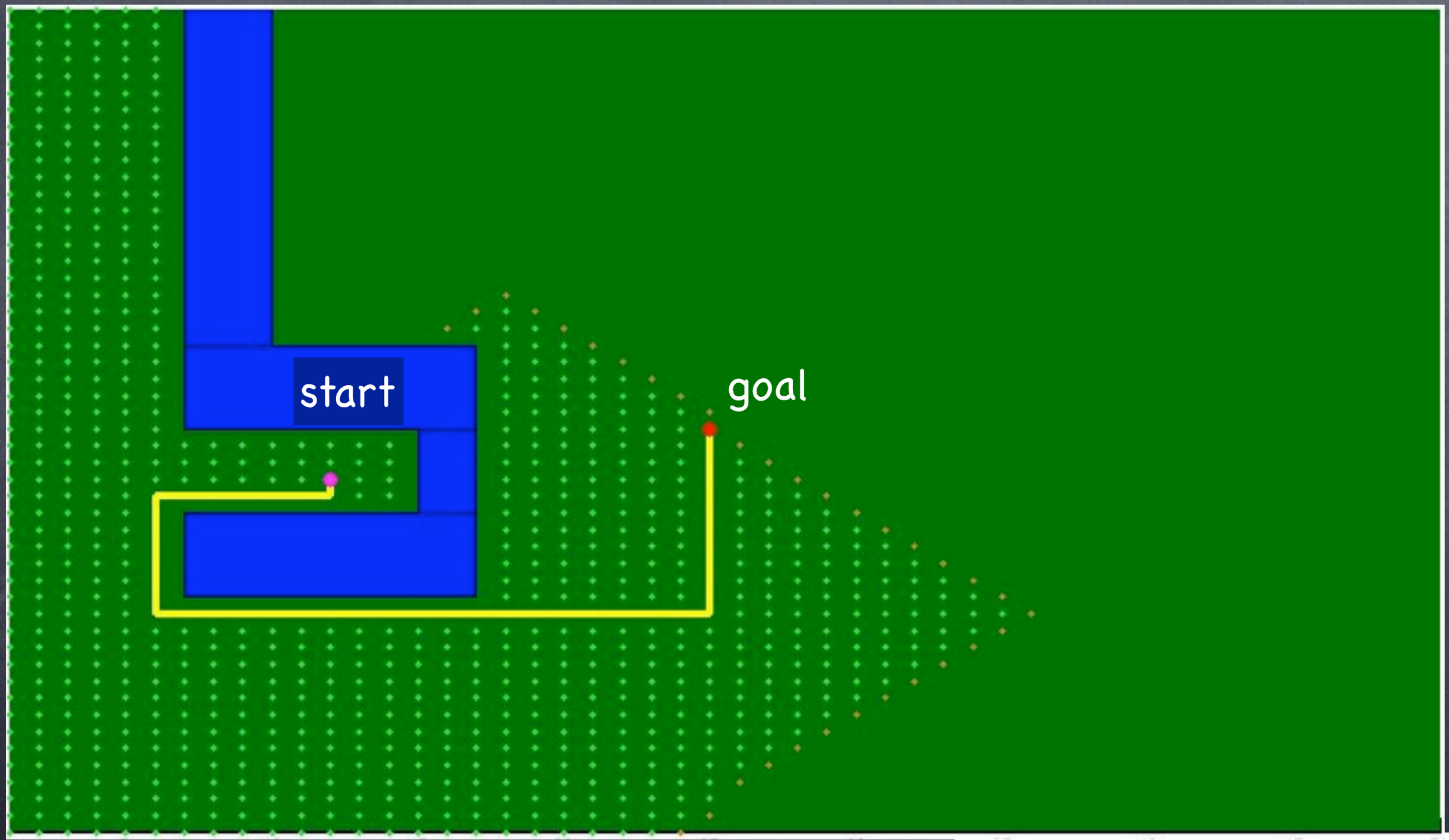
http://www.cs.brown.edu/courses/cs148/pub/pathplan.m



"yellow brick road" graphics by ynm, visited locations in light green
4-connected grid

matlab example: Dijkstra

http://www.cs.brown.edu/courses/cs148/pub/pathplan.m

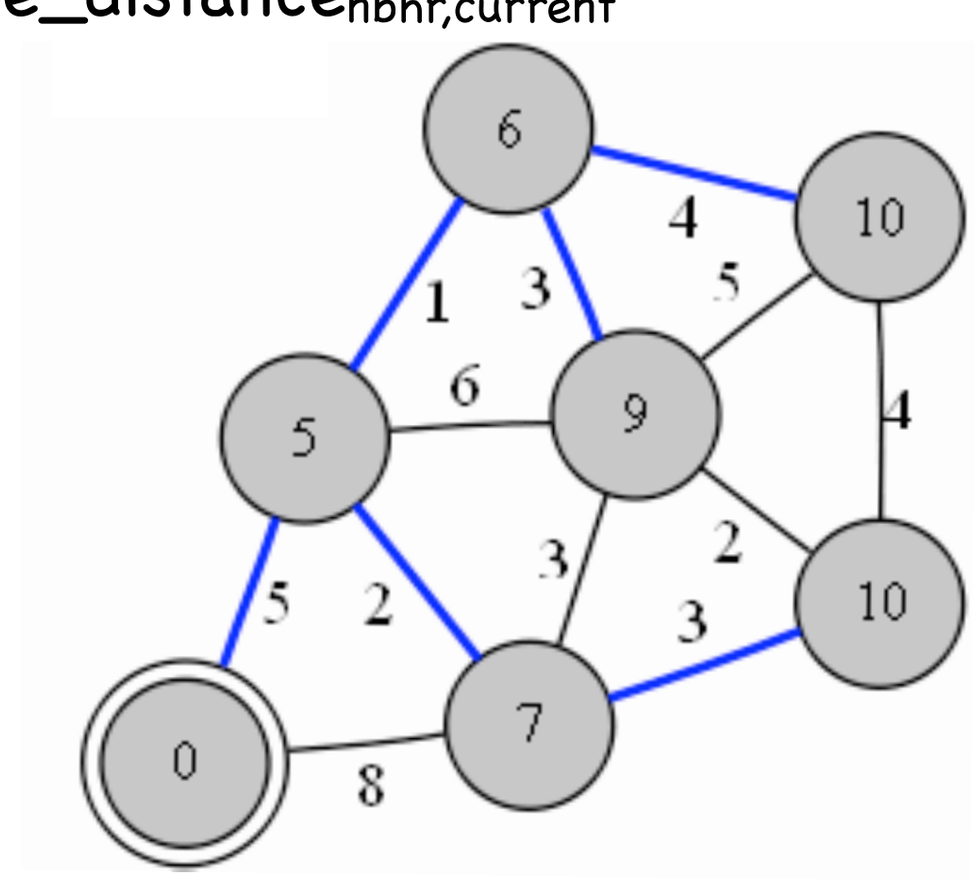


"yellow brick road" graphics by ynm, visited locations in light green
4-connected grid

Dijkstra shortest path algorithm

create visit_queue with start node: $distance_{start} \leftarrow 0$, $parent_{start} \leftarrow$ none
while visit_queue not empty and current_node is not goal
  dequeue from visit_queue: current_node ← min_distance(visit_queue)
  $visited_{current} \leftarrow$ true
  for each neighbor_node in not_visited(adjacent(current_node))
    enqueue neighbor_node, if not in visit_queue
    if $distance_{neighbor} > distance_{current} + line\_distance_{nbhr,current}$
      $parent_{neighbor} \leftarrow$ current_node
      $distance_{neighbor} \leftarrow distance_{current} + line\_distance_{nbhr,current}$
  end for loop
end while loop
output ← parent, distance

Is Dijkstra different than BFS?
Faster search?



A* shortest path algorithm (straight line heuristic)

create visit_queue with start node: $distance_{start} \leftarrow 0$, $parent_{start} \leftarrow$ none

while visit_queue not empty and current_node is not goal

dequeue from visit_queue: current_node ← f_score(visit_queue)

$visited_{current} \leftarrow$ true

for each neighbor_node in not_visited(adjacent(current_node))

enqueue neighbor_node, if not in visit_queue

if $distance_{neighbor} > distance_{current} + line\_distance_{nbhr,current}$

$parent_{neighbor} \leftarrow$ current_node

$distance_{neighbor} \leftarrow distance_{current} + line\_distance_{nbhr,current}$

f_score ← $distance_{nbhr} + line\_distance_{nbhr,goal}$

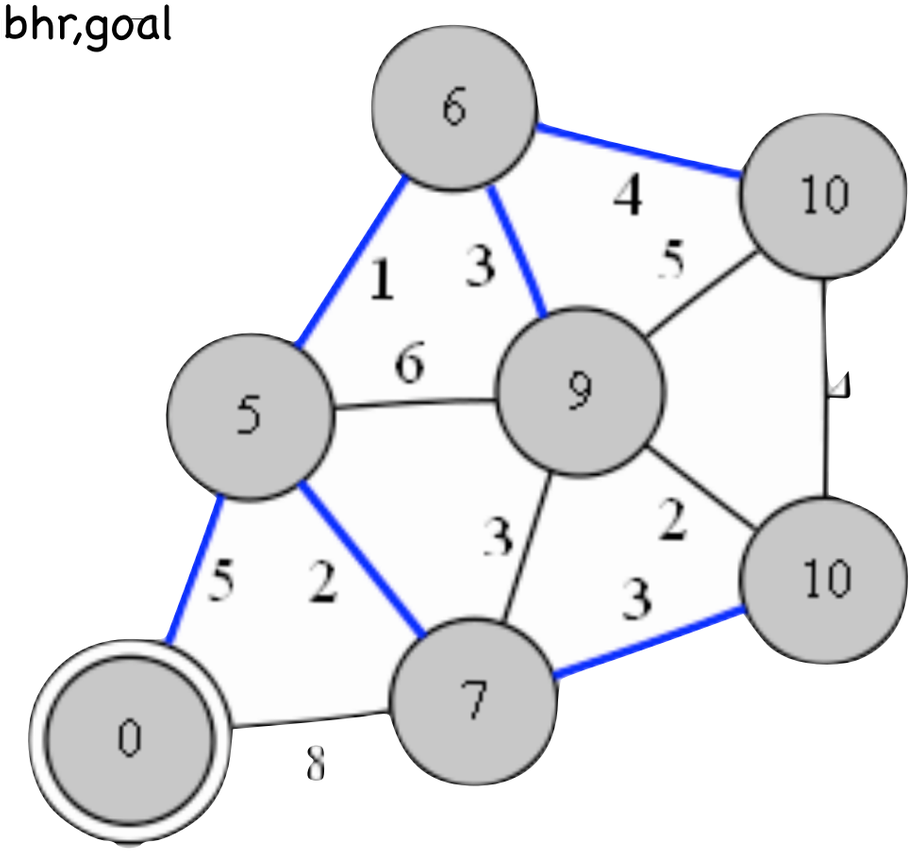
end for loop

end while loop

output ← parent, distance

g_score:
distance along path

h_score:
best distance to goal



A* shortest path algorithm (straight line heuristic)

create visit_queue with start node: $distance_{start} \leftarrow 0$, $parent_{start} \leftarrow$ none

while visit_queue not empty and current_node is not goal

dequeue from visit_queue: current_node ← f_score(visit_queue)

$visited_{current} \leftarrow$ true

for each neighbor_node in not_visited(adjacent(current_node))

enqueue neighbor_node, if not in visit_queue

if $distance_{neighbor} > distance_{current} + line\_distance_{nbhr,current}$

$parent_{neighbor} \leftarrow$ current_node

$distance_{neighbor} \leftarrow distance_{current} + line\_distance_{nbhr,current}$

f_score ← $distance_{nbhr} + line\_distance_{nbhr,goal}$

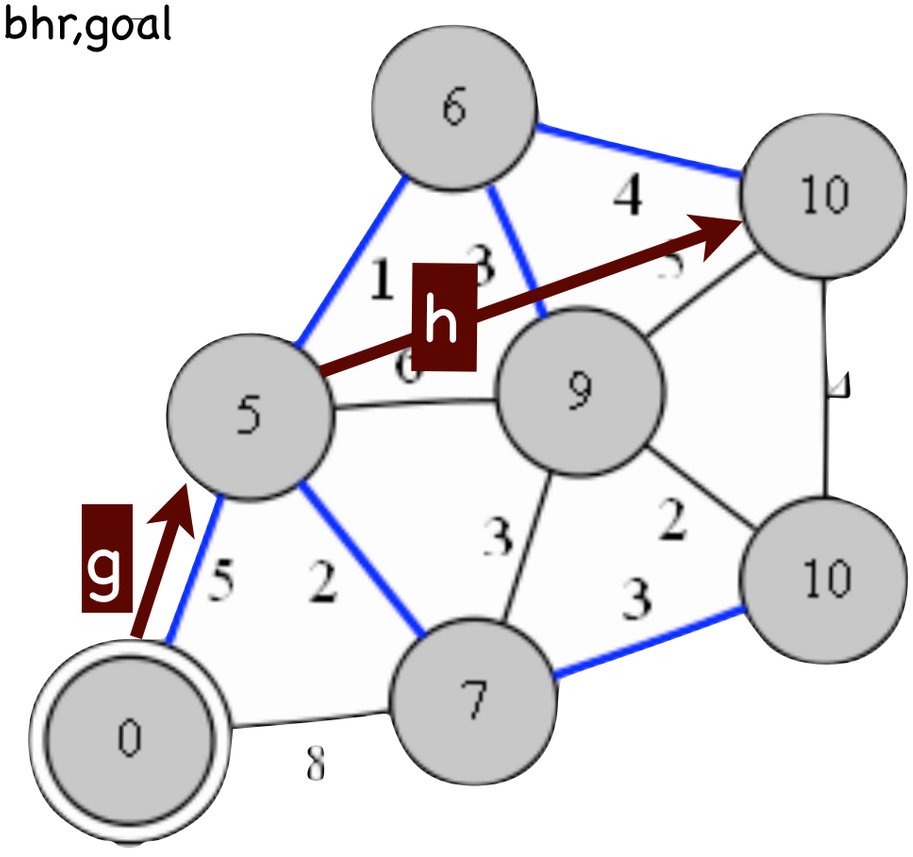
end for loop

end while loop

output ← parent, distance

g_score: distance along path

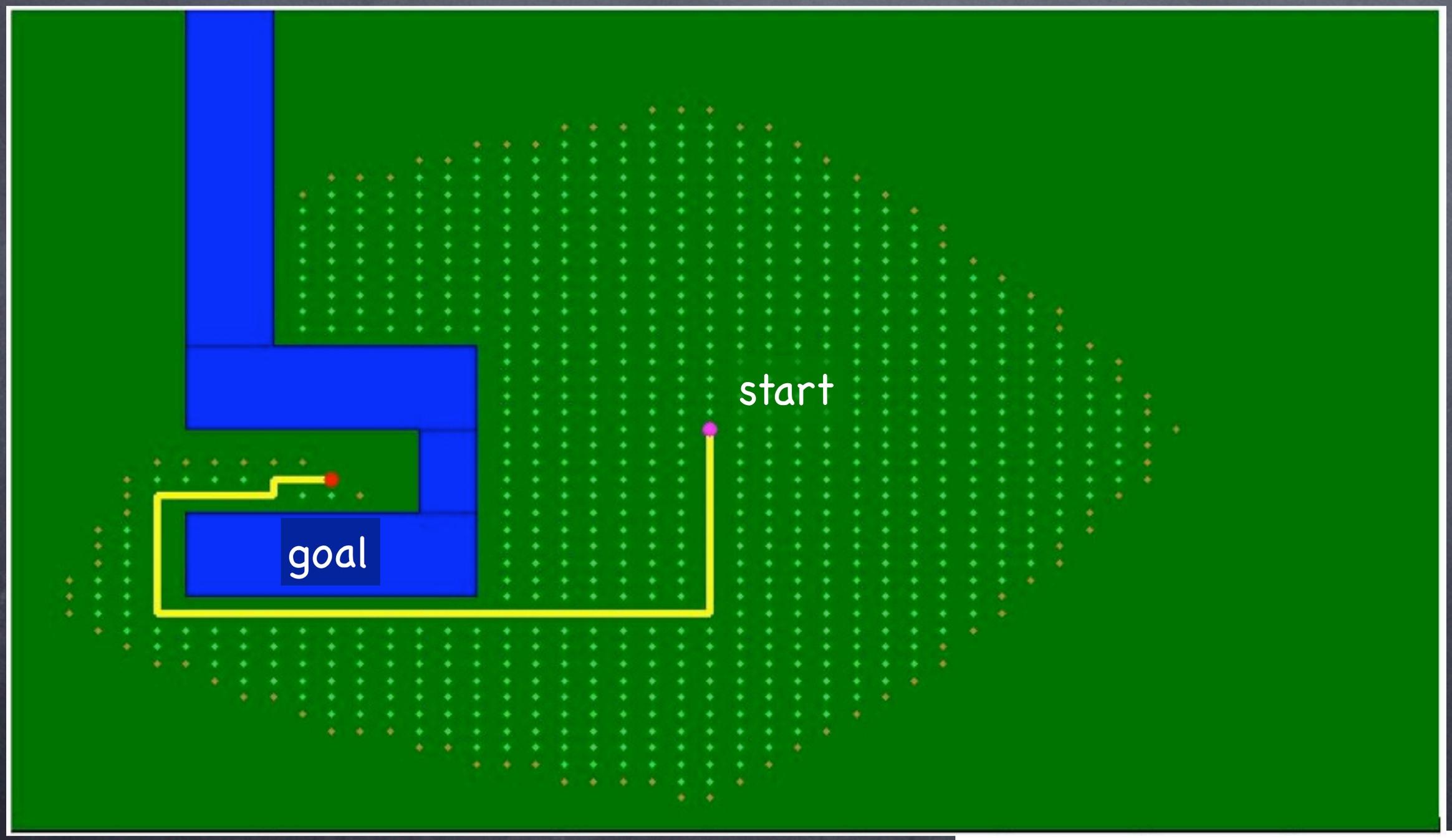
h_score: best distance to goal



How is A* related to Dijkstra?

matlab example: A*

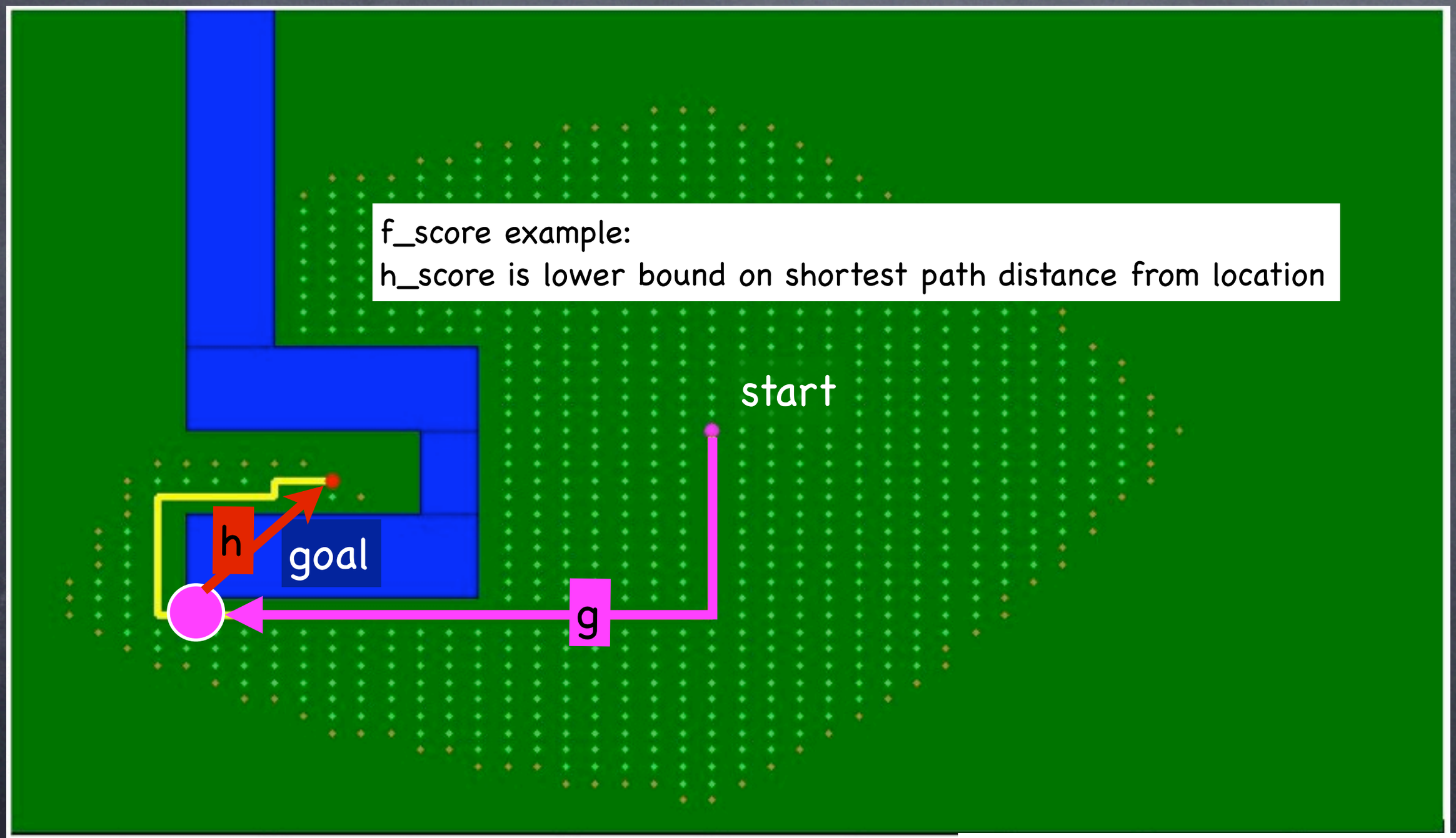
http://www.cs.brown.edu/courses/cs148/pub/pathplan.m



"yellow brick road" graphics by ynm, visited locations in light green
4-connected grid

matlab example: A*

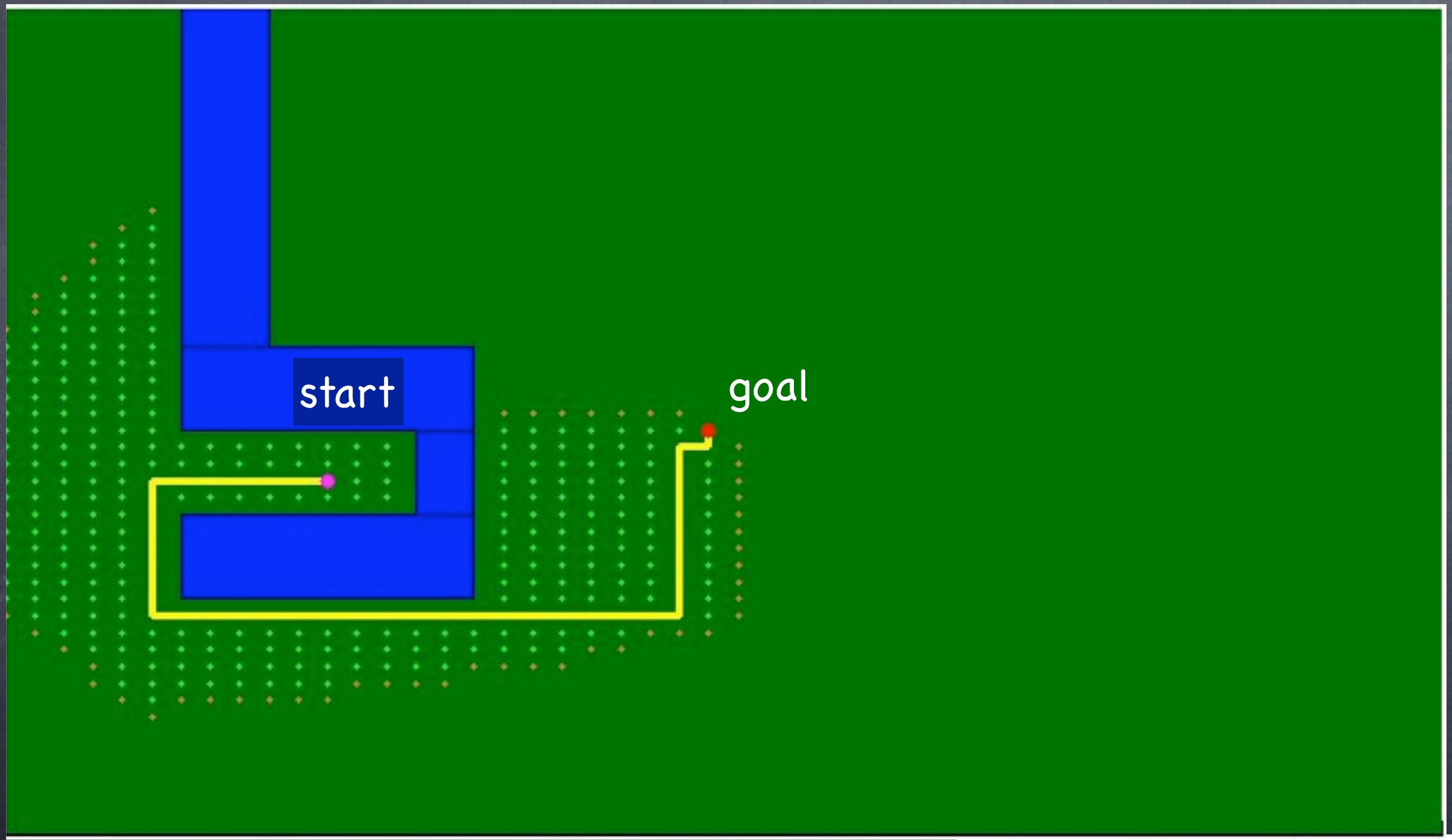
http://www.cs.brown.edu/courses/cs148/pub/pathplan.m



"yellow brick road" graphics by ynm, visited locations in light green
4-connected grid

matlab example: A*

http://www.cs.brown.edu/courses/cs148/pub/pathplan.m



"yellow brick road" graphics by ynm, visited locations in light green
4-connected grid

A* shortest path algorithm (straight line heuristic)

create visit_queue with start node: $distance_{start} \leftarrow 0$, $parent_{start} \leftarrow$ none

while visit_queue not empty and current_node is not goal

dequeue from visit_queue: current_node ← f_score(visit_queue)

$visited_{current} \leftarrow$ true

for each neighbor_node in not_visited(adjacent(current_node))

enqueue neighbor_node, if not in visit_queue

if $distance_{neighbor} > distance_{current} + line\_distance_{nbhr,current}$

$parent_{neighbor} \leftarrow$ current_node

$distance_{neighbor} \leftarrow distance_{current} + line\_distance_{nbhr,current}$

f_score ← $distance_{nbhr} + line\_distance_{nbhr,goal}$

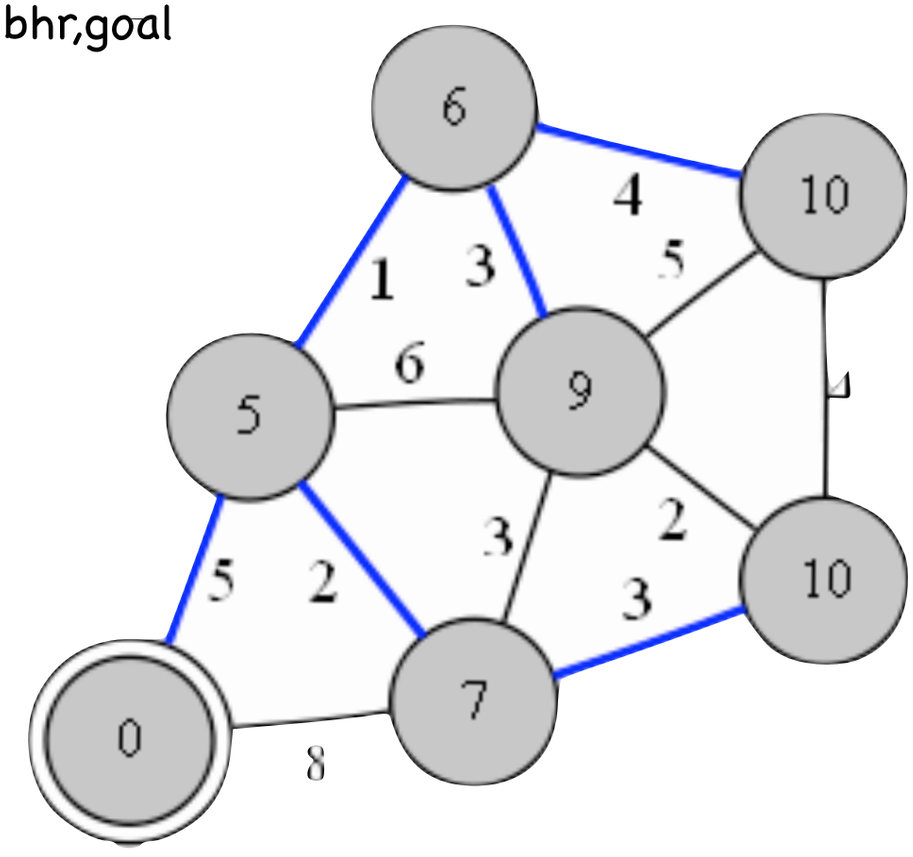
end for loop

end while loop

output ← parent, distance

g_score: distance along path

h_score: best distance to goal



Suppose f_score = h_score

greedy best-first shortest path algorithm

create visit_queue with start node: $distance_{start} \leftarrow 0$, $parent_{start} \leftarrow none$

while visit_queue not empty and current_node is not goal

dequeue from visit_queue: current_node ← f_score(visit_queue)

$visited_{current} \leftarrow true$

for each neighbor_node in not_visited(adjacent(current_node))

enqueue neighbor_node, if not in visit_queue

if $distance_{neighbor} > distance_{current} + line\_distance_{nbhr,current}$

$parent_{neighbor} \leftarrow current\_node$

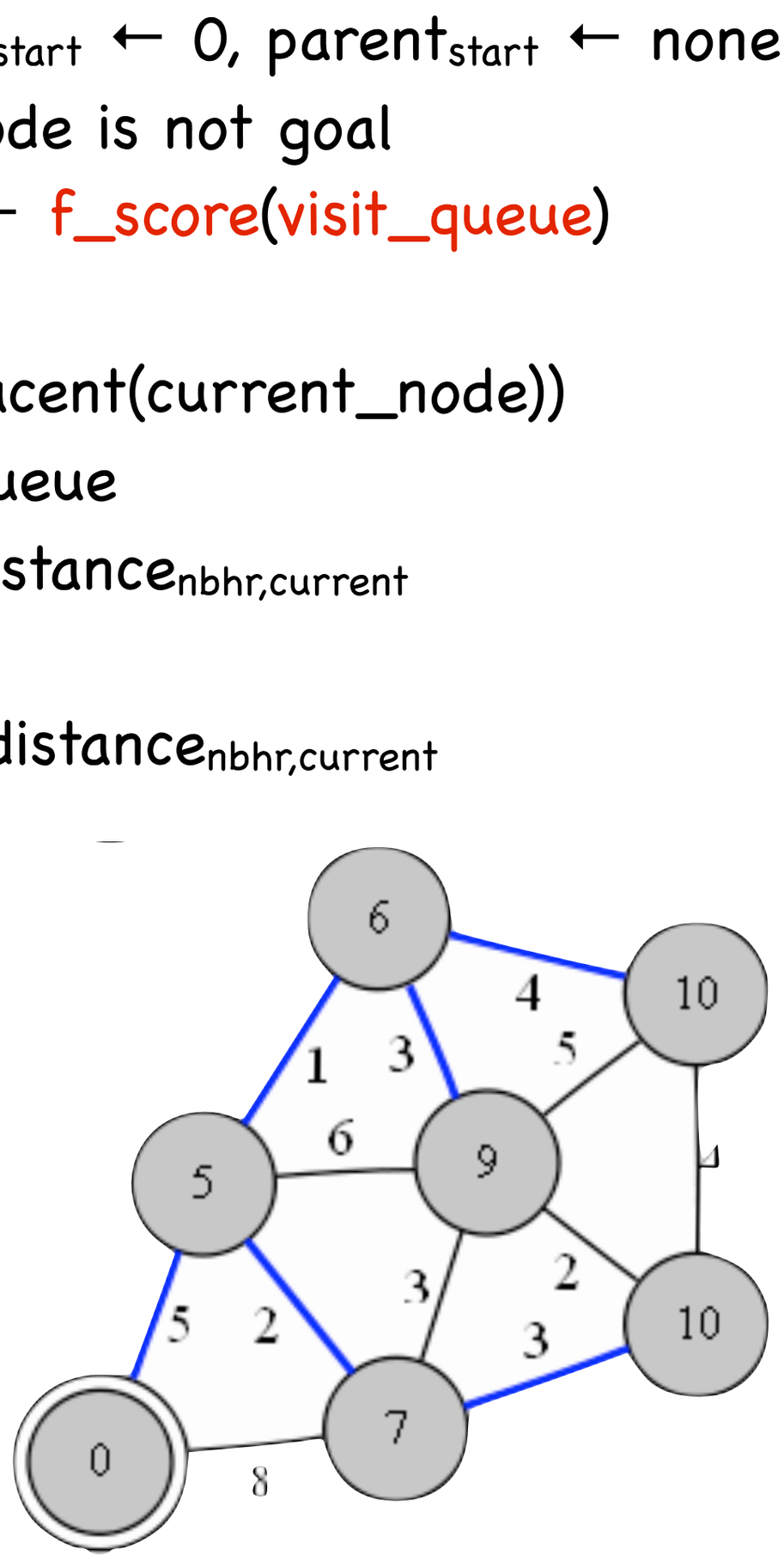
$distance_{neighbor} \leftarrow distance_{current} + line\_distance_{nbhr,current}$

f_score ← $line\_distance_{nbhr,goal}$

end for loop

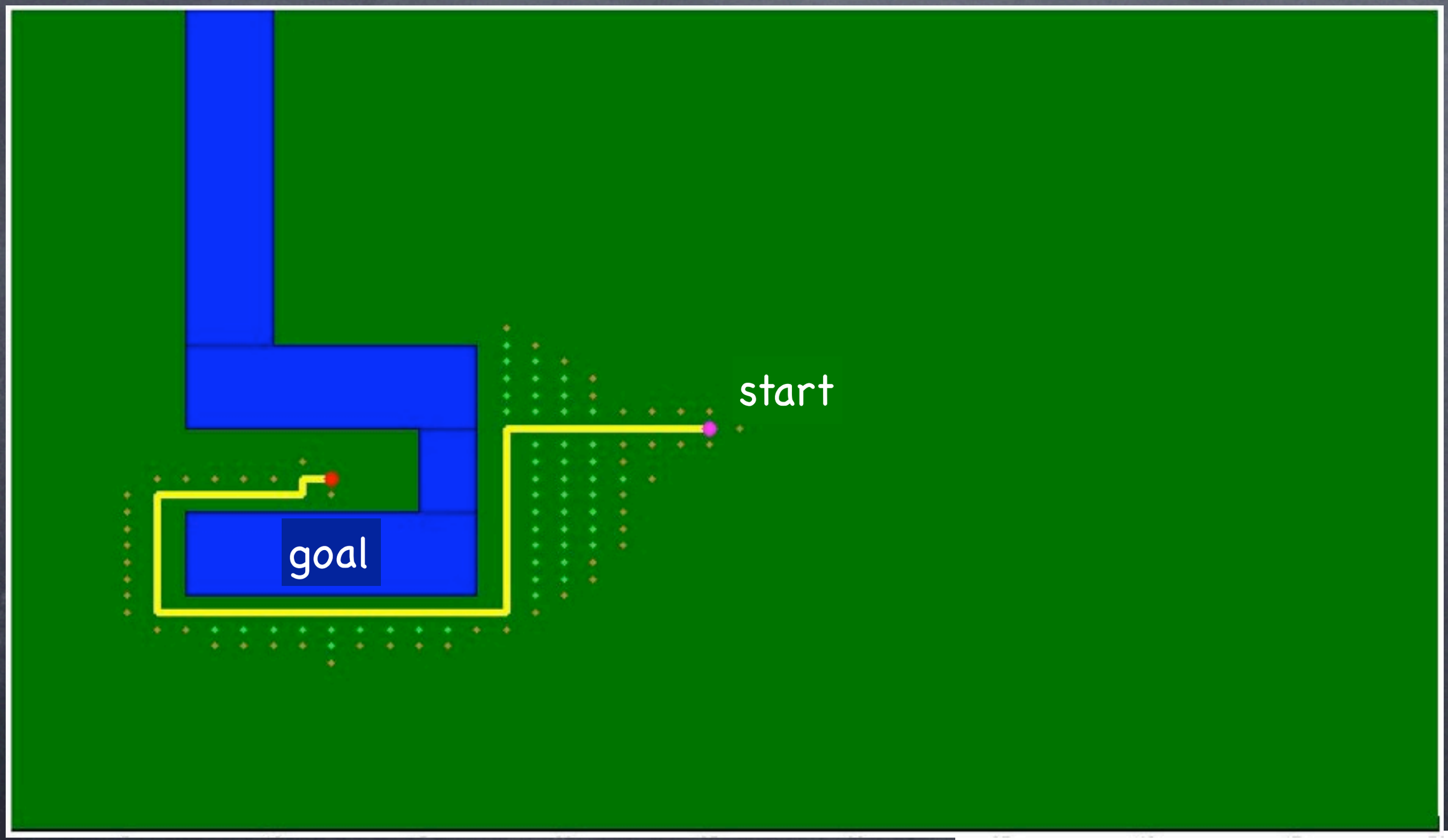
end while loop

output ← parent, distance



matlab example: gr.best-first

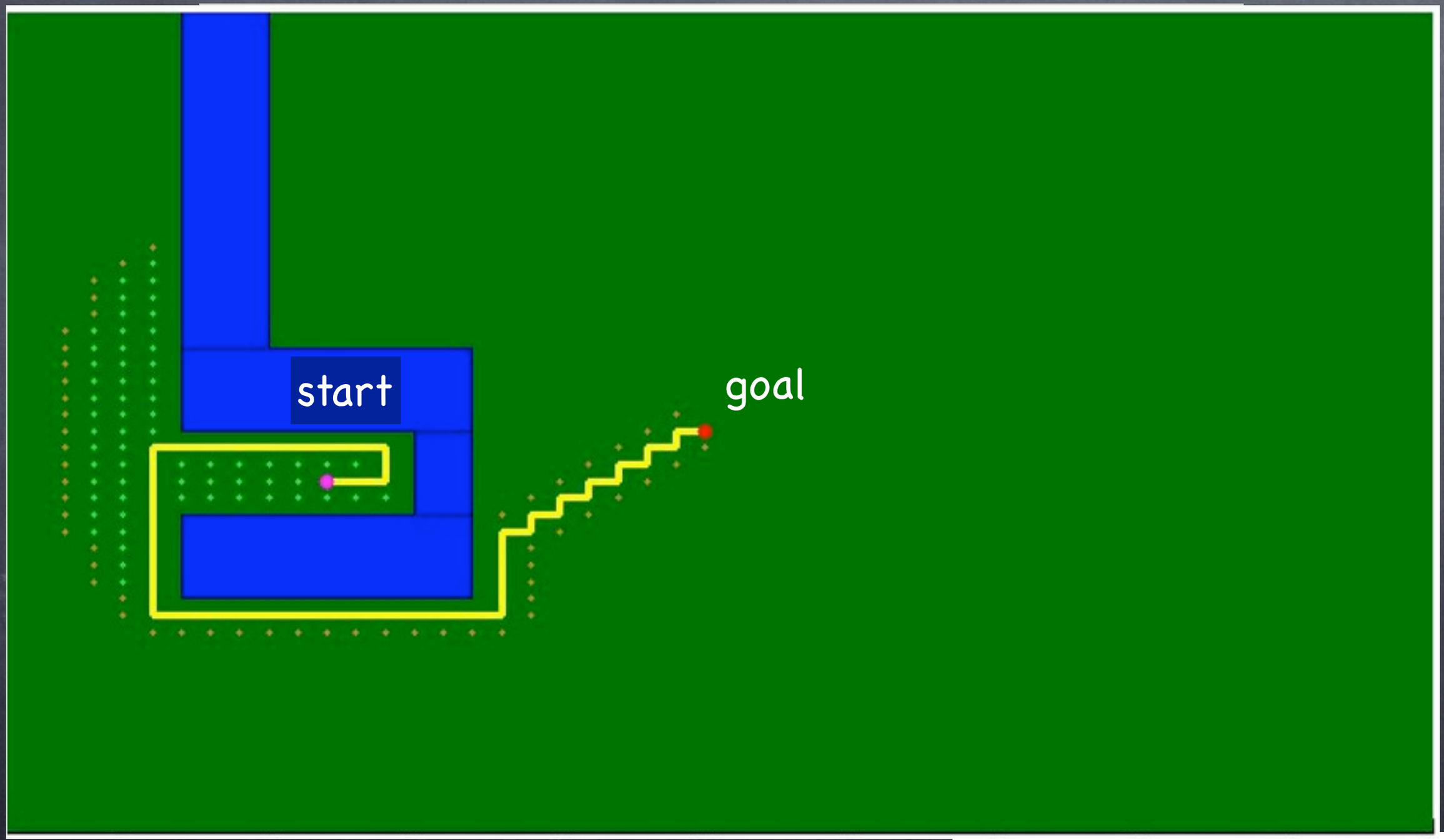
http://www.cs.brown.edu/courses/cs148/pub/pathplan.m



"yellow brick road" graphics by ynm, visited locations in light green
4-connected grid

matlab example: gr.best-first

http://www.cs.brown.edu/courses/cs148/pub/pathplan.m



"yellow brick road" graphics by ynm, visited locations in light green
4-connected grid

Approaches to path planning

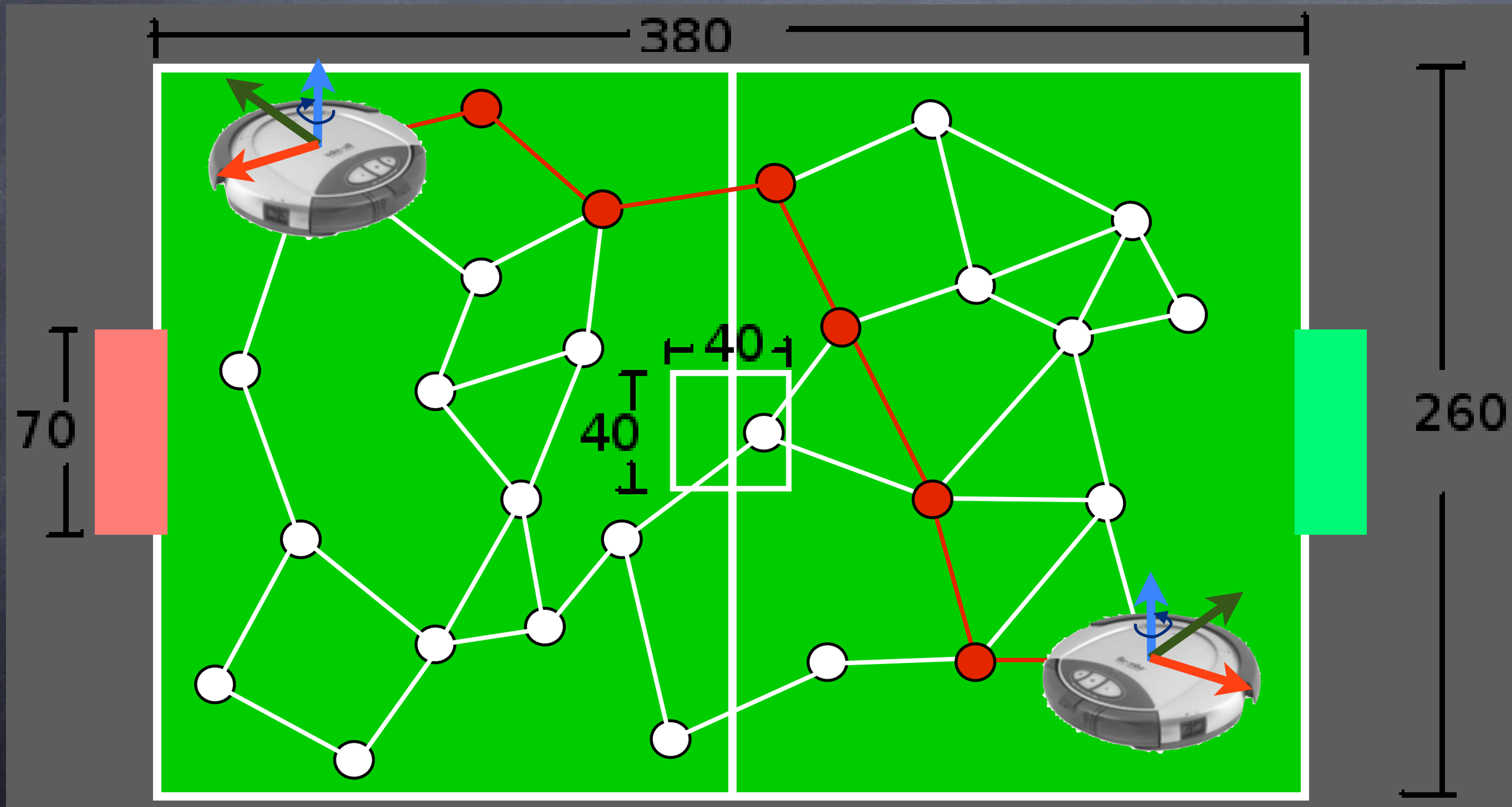
- Search (fixed graph)
  - DFS, BFS, Dijkstra, A*
- Search (build graph):
  - Probabilistic Road Maps
  - Rapidly-exploring Random Trees
- Optimization (local search):
  - Potential fields, gradient descent

Fast forward other planning methods

RRT

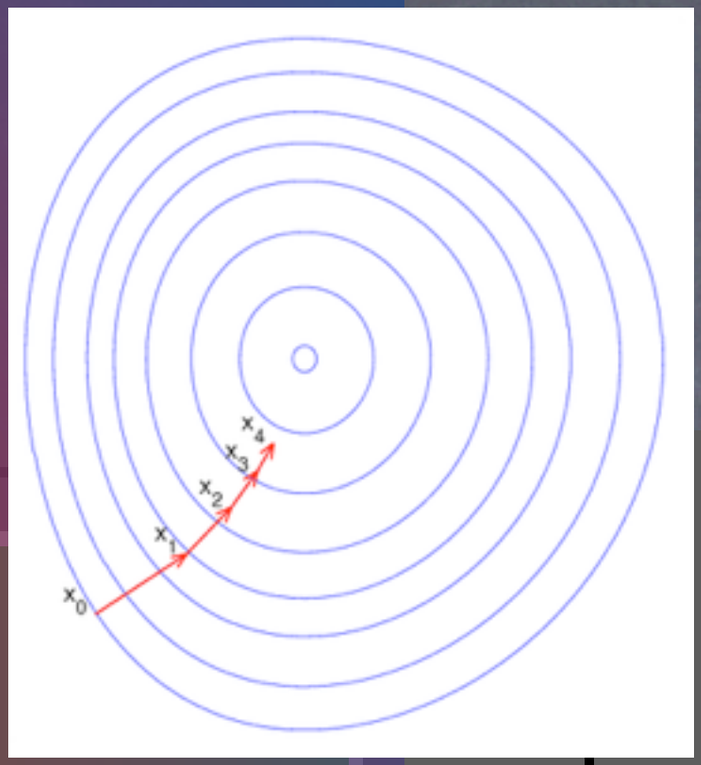
(more detail later)

Explore poses and connectivity;
Find shortest path in built graph



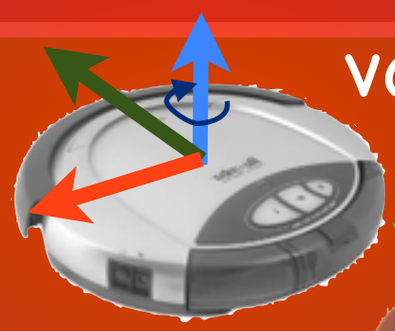
Potential field

(more detail later)

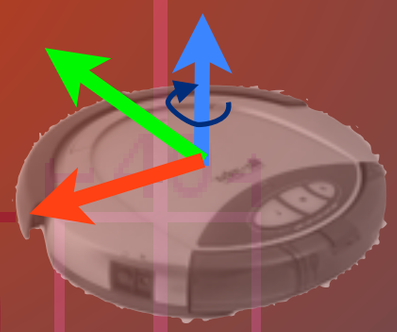


Energy potential converges at goal

volcanic

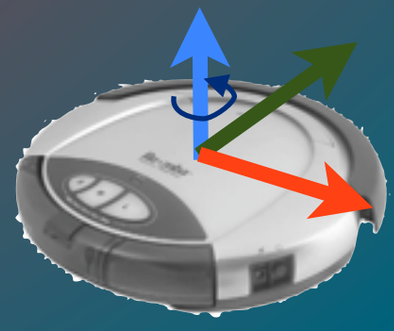


on fire

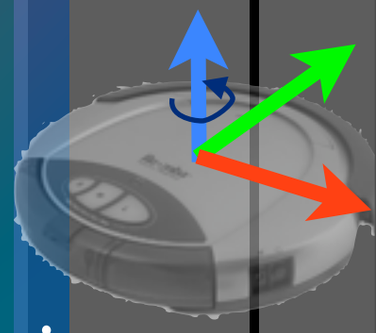


heating up

a little warmer



cold

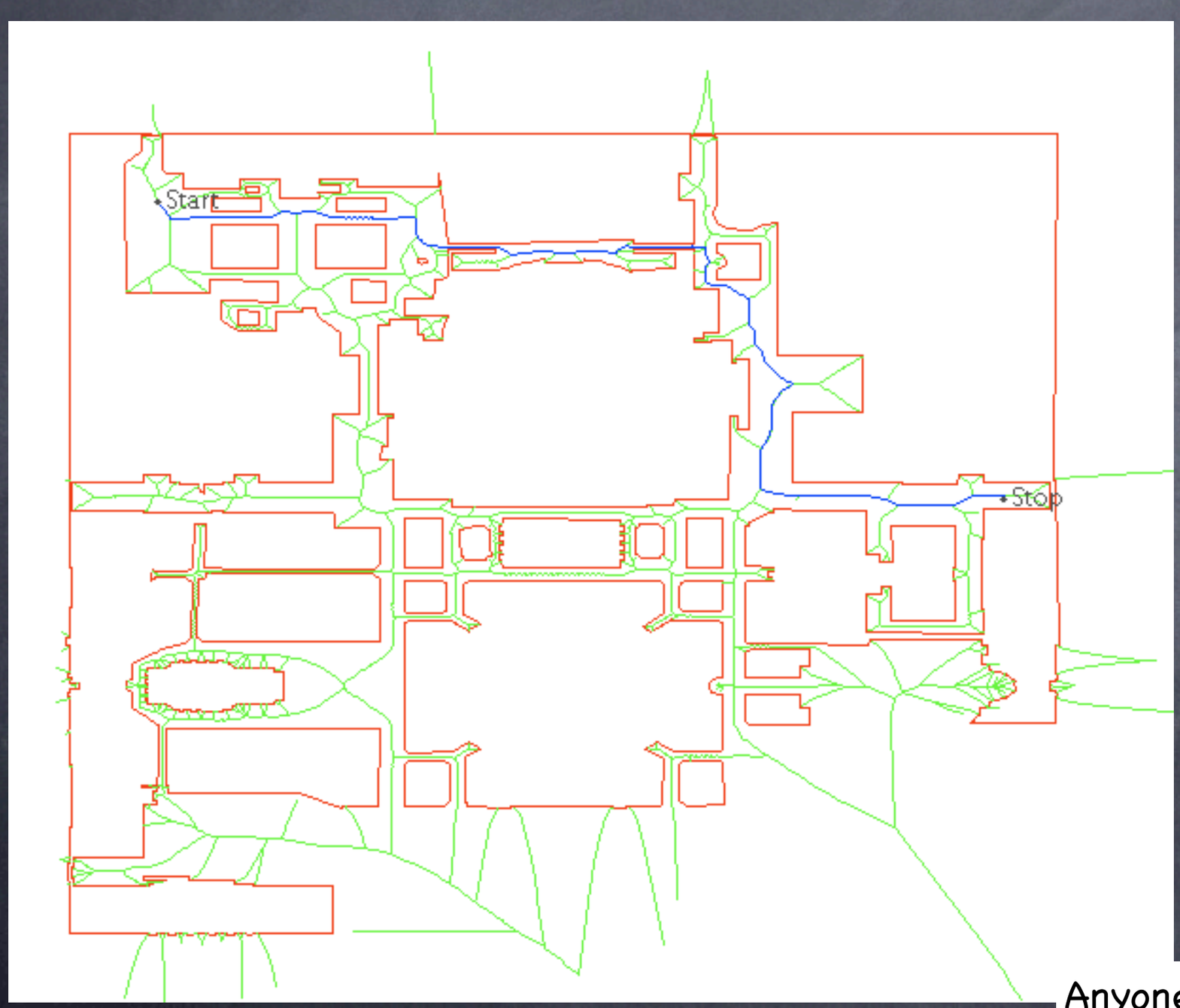


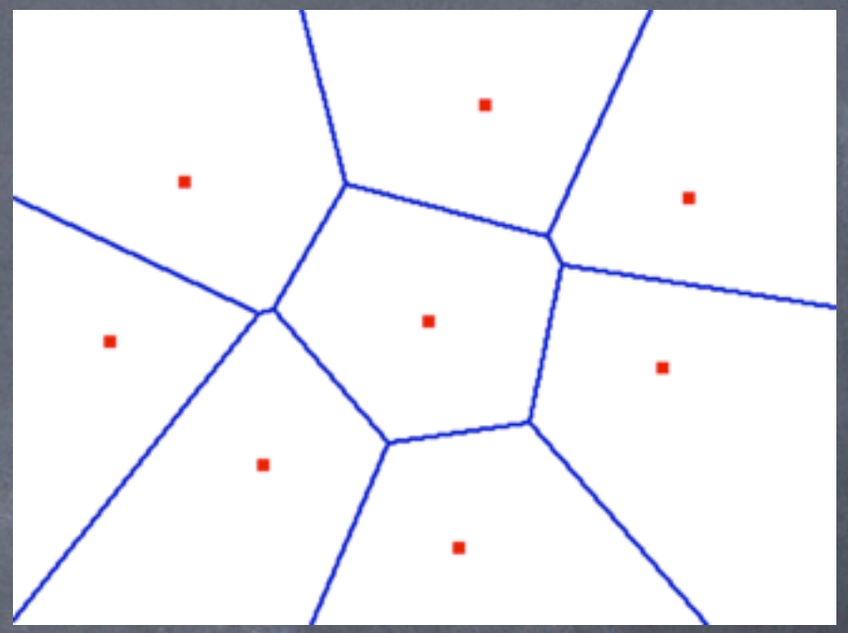
freezing

Similar to "getting warmer, getting colder" game

Voronoi-based Planning

http://www.cs.columbia.edu/~pblaer/projects/path_planner/applet.html

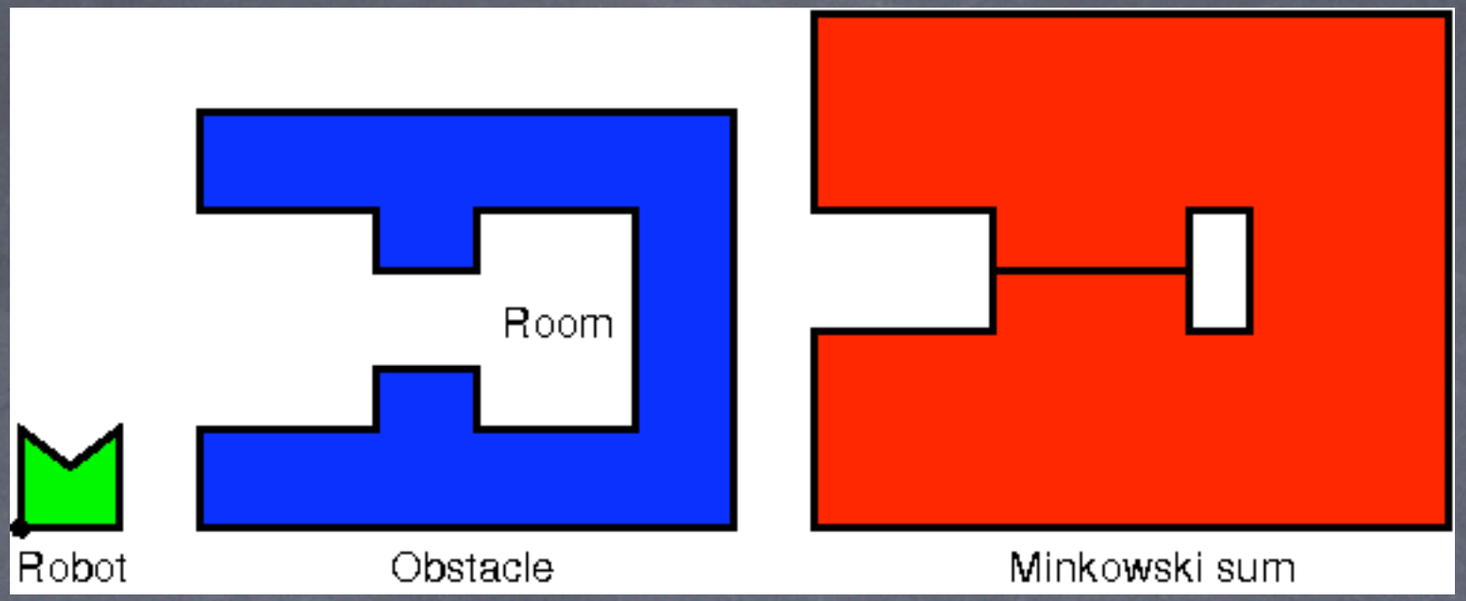


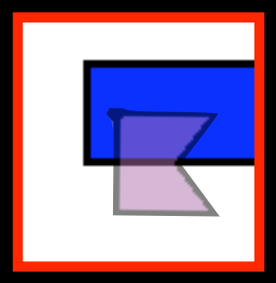


Voronoi diagram: all points equidistant to nearest sites (map boundary)

Anyone heard of a Delaunay triangulation? Medial axis? Skeletonization?

Minkowski Planning





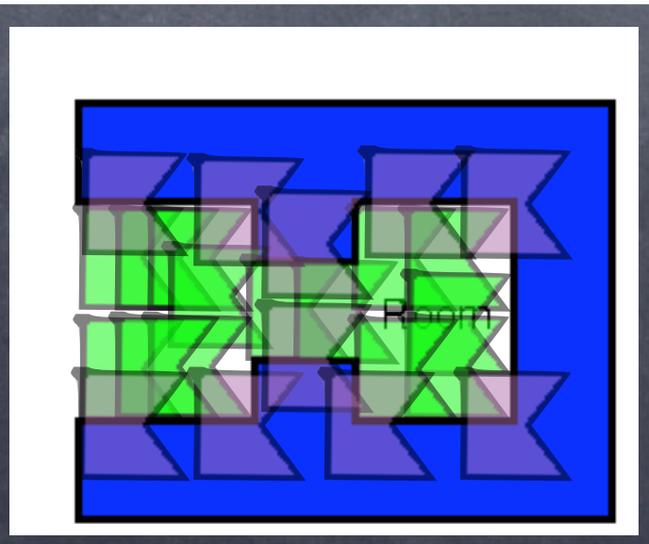
Occlude location if robot intersects obstacle

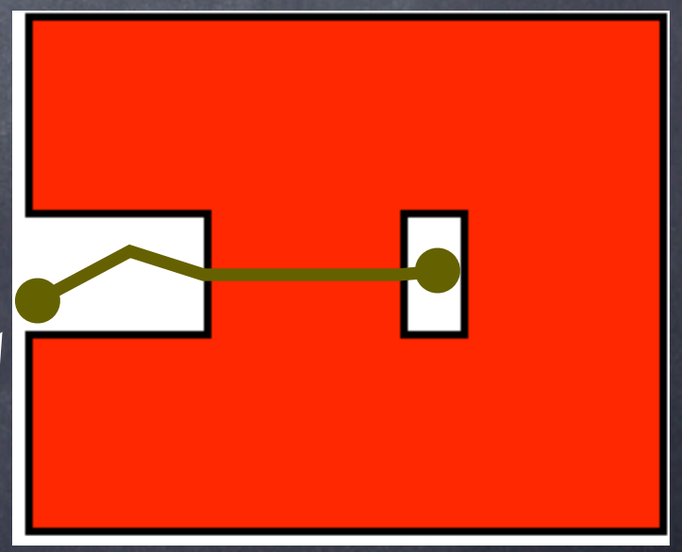
Leave location free if robot non-intersecting with object

Minkowski sum: identify non intersecting robot locations

Space of valid paths defined by Minkowski sum

Robot geometry





A valid path

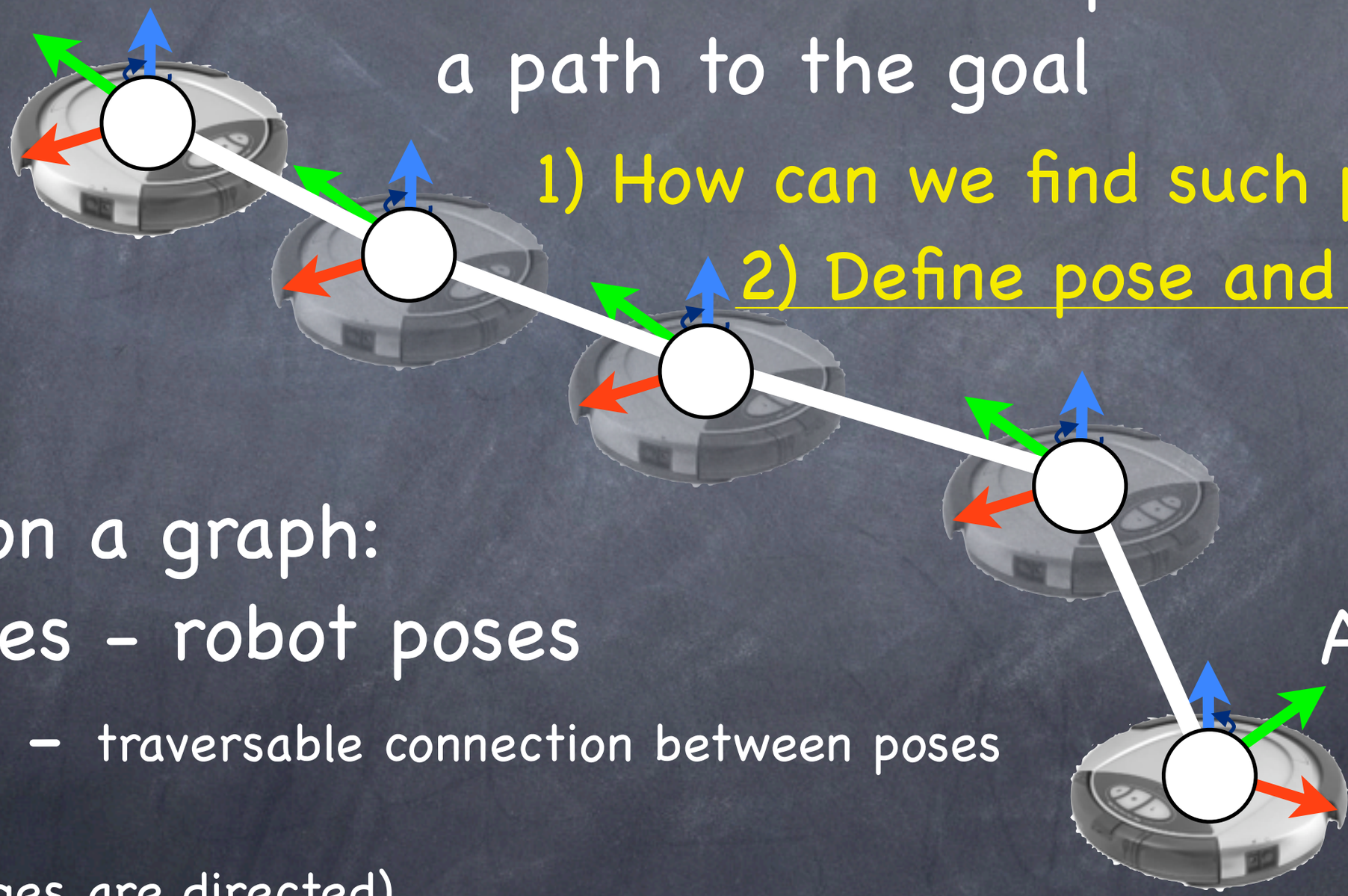
Path Planning

B: Goal

Find intermediate poses forming a path to the goal

1) How can we find such paths?

2) Define pose and controls?



A: Start

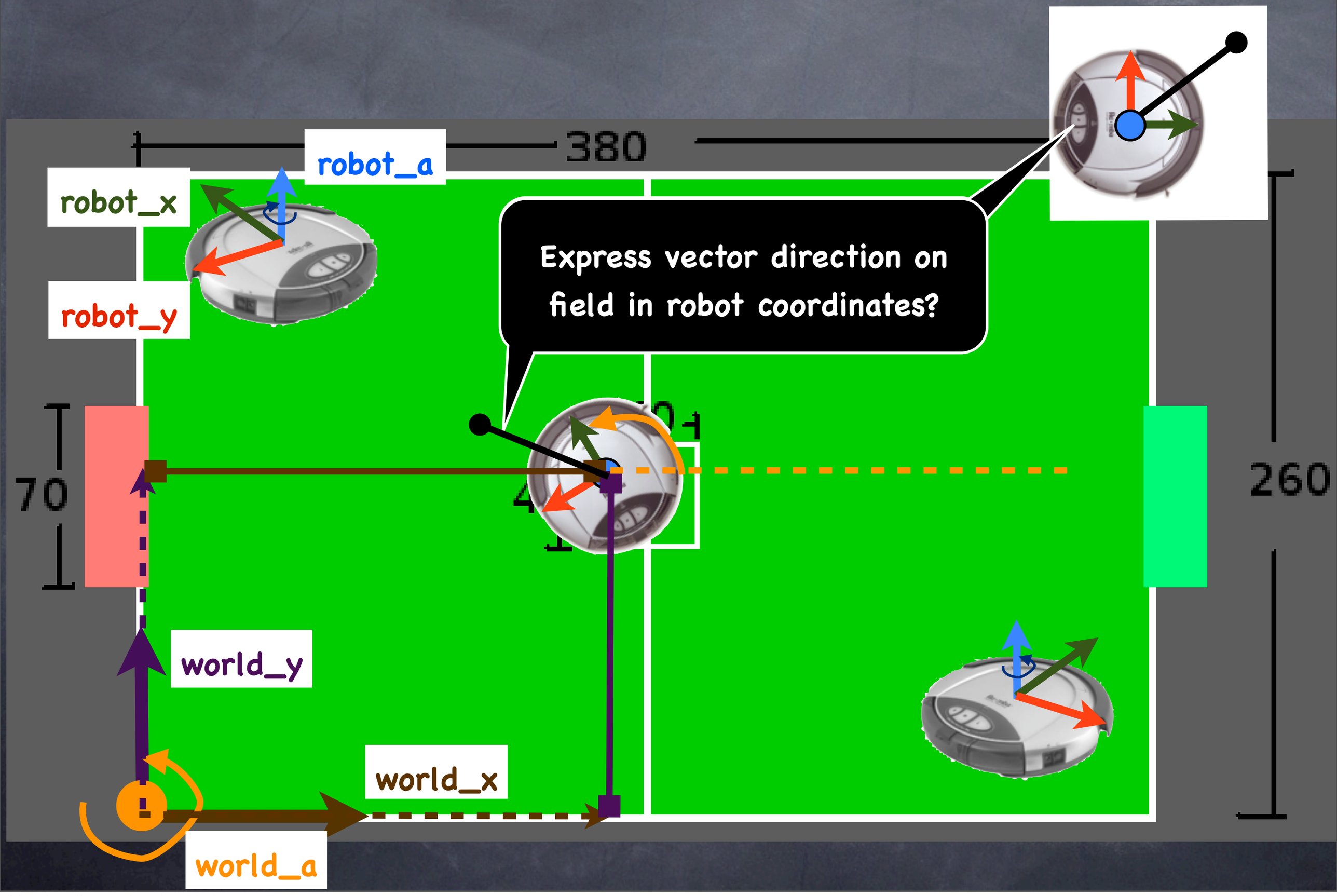
Path on a graph:

vertices - robot poses

edges - traversable connection between poses

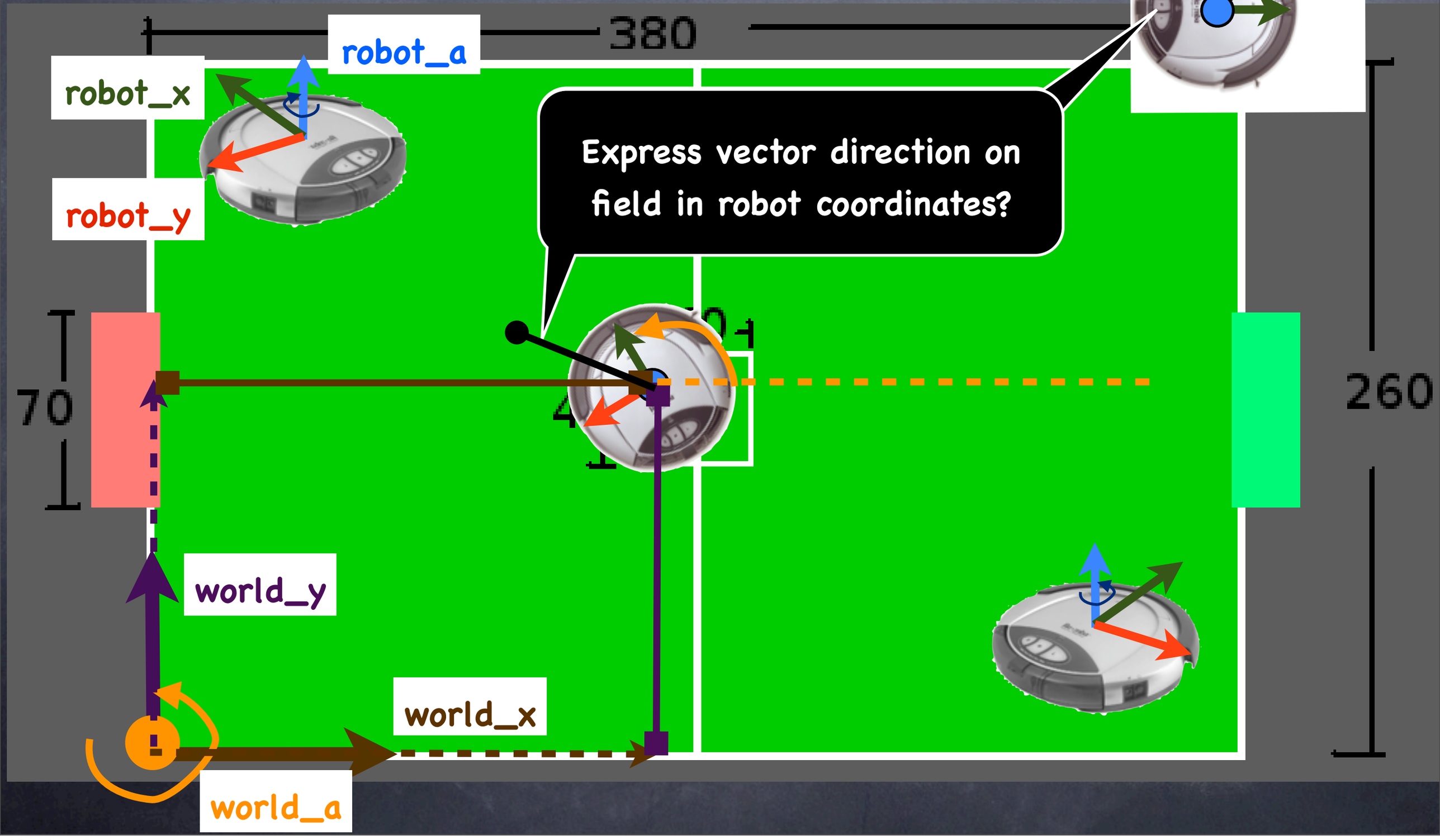
(note edges are directed)

Warning: matrix multiplication ahead



Simple answer rotate vector by $-pose_{w\_a}$

Is there a cleaner way?

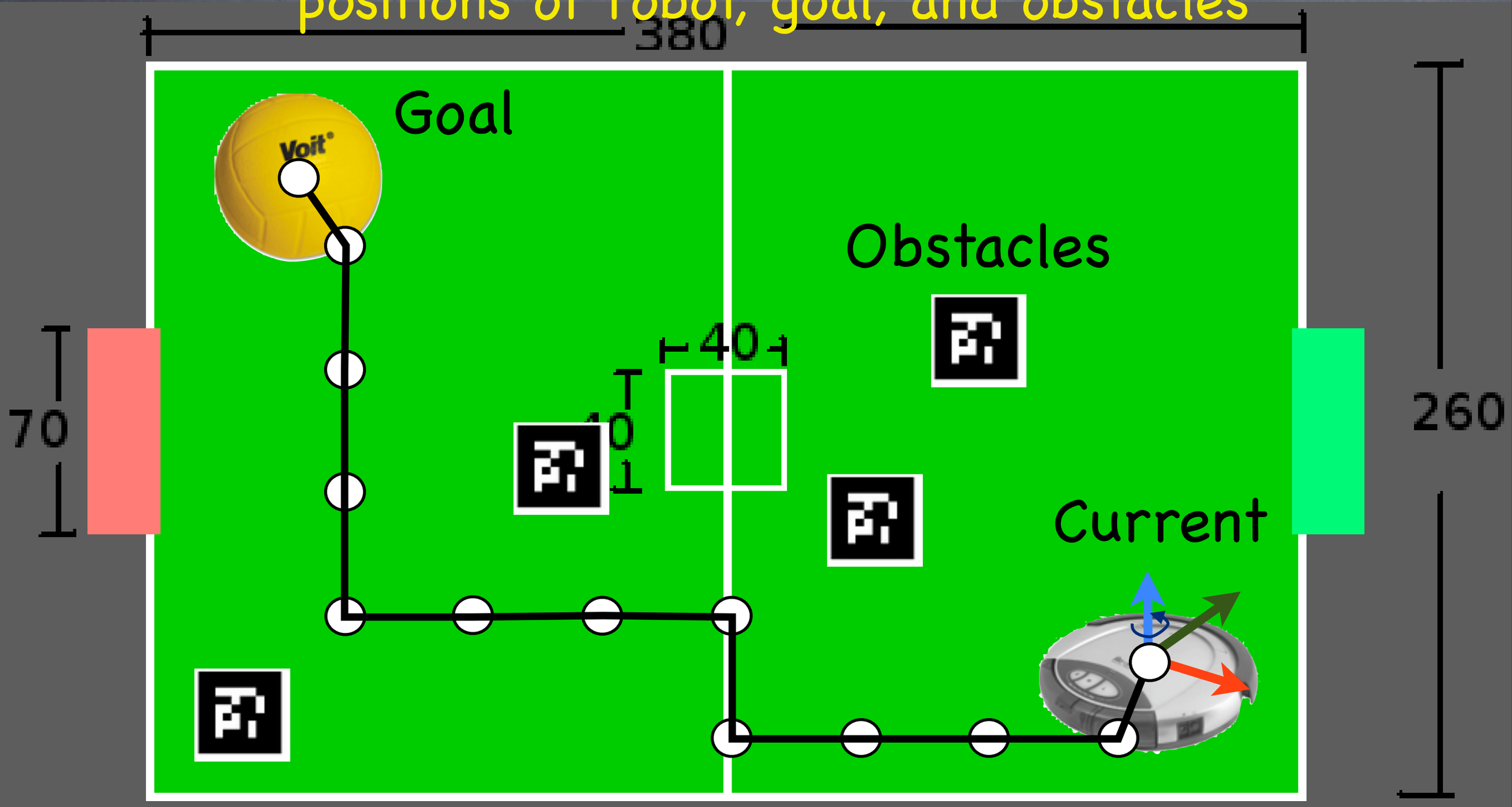


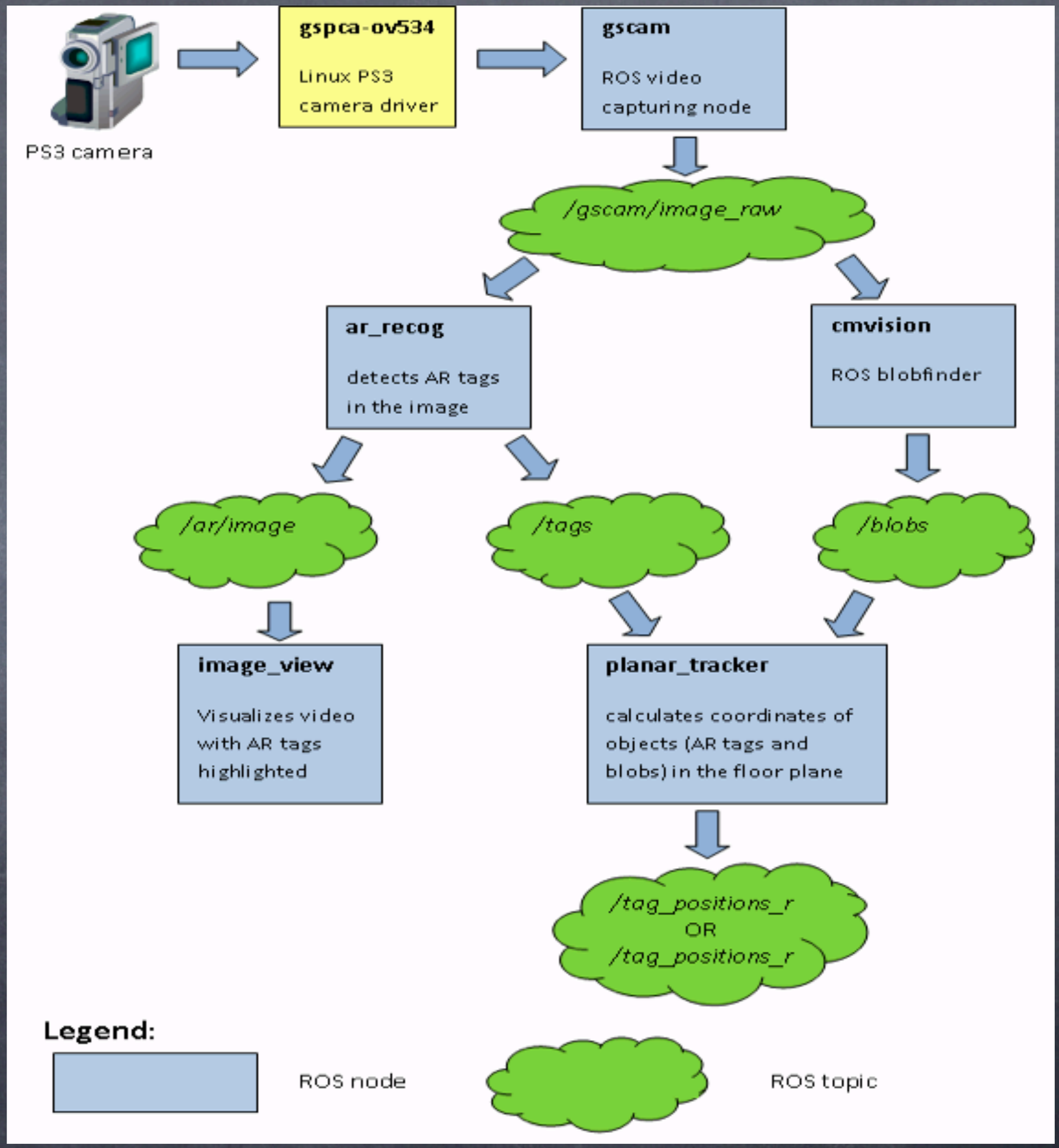
Project 3: milestone - generate path

Generate path to goal given overhead localization

Overhead localization provides state

positions of robot, goal, and obstacles




PS3 camera
gspca-ov534
Linux PS3 camera driver
gscam
ROS video capturing node
/gscam/image_raw
ar_recog
detects AR tags in the image
cmvision
ROS blobfinder
/ar/image
/tags
/blobs
image_view
Visualizes video with AR tags highlighted
planar_tracker
calculates coordinates of objects (AR tags and blobs) in the floor plane
/tag_positions_r
OR
/tag_positions_r
Legend:
ROS node
ROS topic