CS354R

DR SARAH ABRAHAM

PATH PLANNING

PATH FINDING

- Problem Statement: Given a start point A and a goal point B, find a path from A to B that is clear
  - Generally want to minimize a cost: distance, travel time
  - Travel time depends on terrain
  - May be complicated by dynamic changes: paths being blocked or removed
- Very common problem in games:
  - In FPS: How does the AI get from room to room?
  - In RTS: User clicks on units, tells them to go somewhere. How do they get there? How do they avoid each other?
  - Chase games, sports games, etc

SEARCH OR OPTIMIZATION?

- Path planning (also called route-finding) can be phrased as a search problem:
  - Find a path to the goal B that minimizes Cost(path)
- Path planning is also a kind of optimization problem:
  - Minimize Cost(path) subject to the constraint that path joins A and B
  - State space is paths joining A and B
- The difference is mostly terminology of different communities (AI vs. Optimization)
- Search is normally through a discrete state space

BRIEF OVERVIEW OF TECHNIQUES

- Discrete algorithms: BFS, Greedy search, A*
- Potential fields:
  - Put a "force field" around obstacles, and follow the "potential valleys"
- Pre-compute plans with dynamic re-planning
  - Plan as search, but pre-compute answer and modify as required
- Special algorithms for special cases:
  - e.g. Given a fixed start point, fast ways to find paths around polygonal obstacles

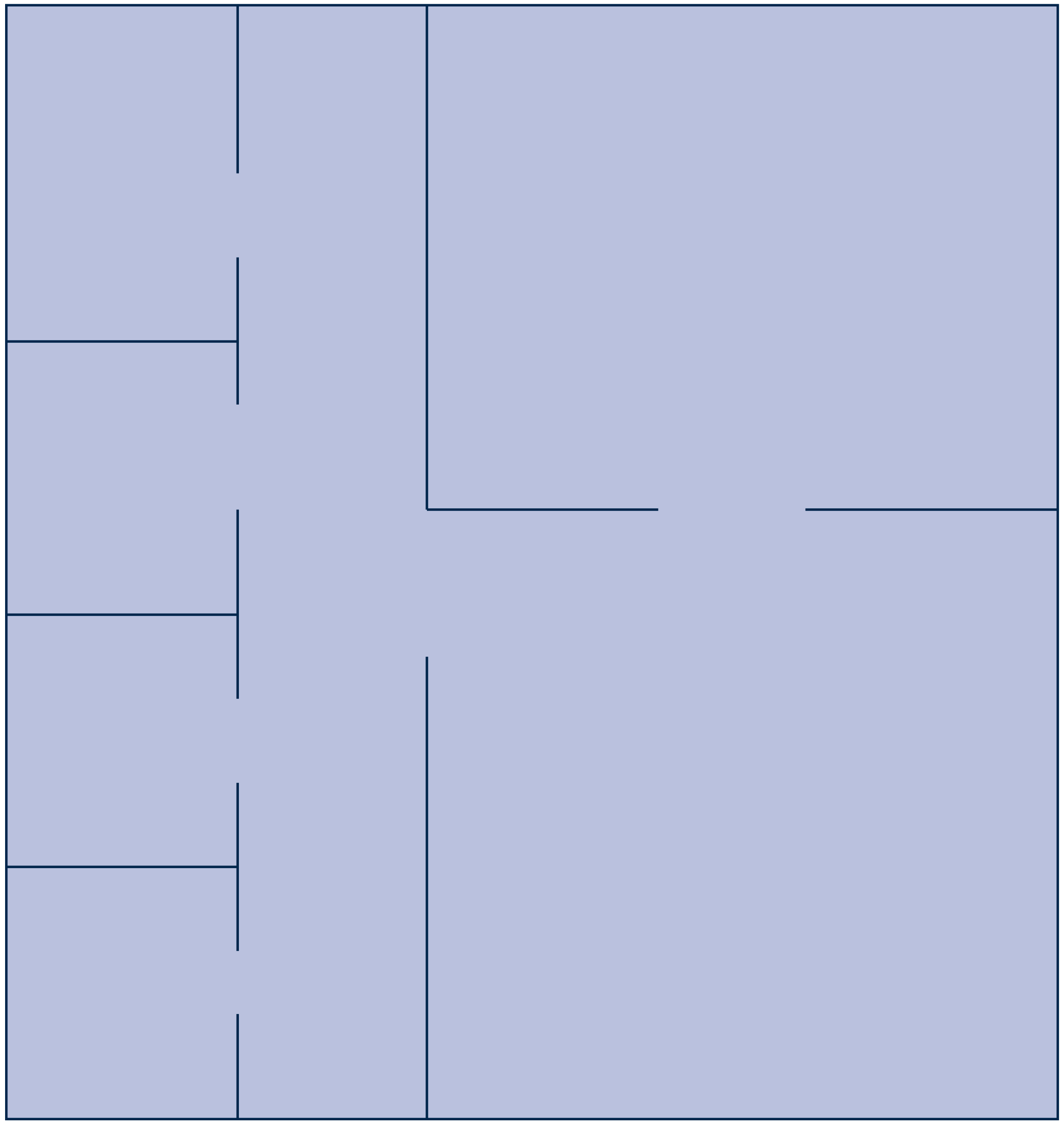
GRAPH-BASED ALGORITHMS

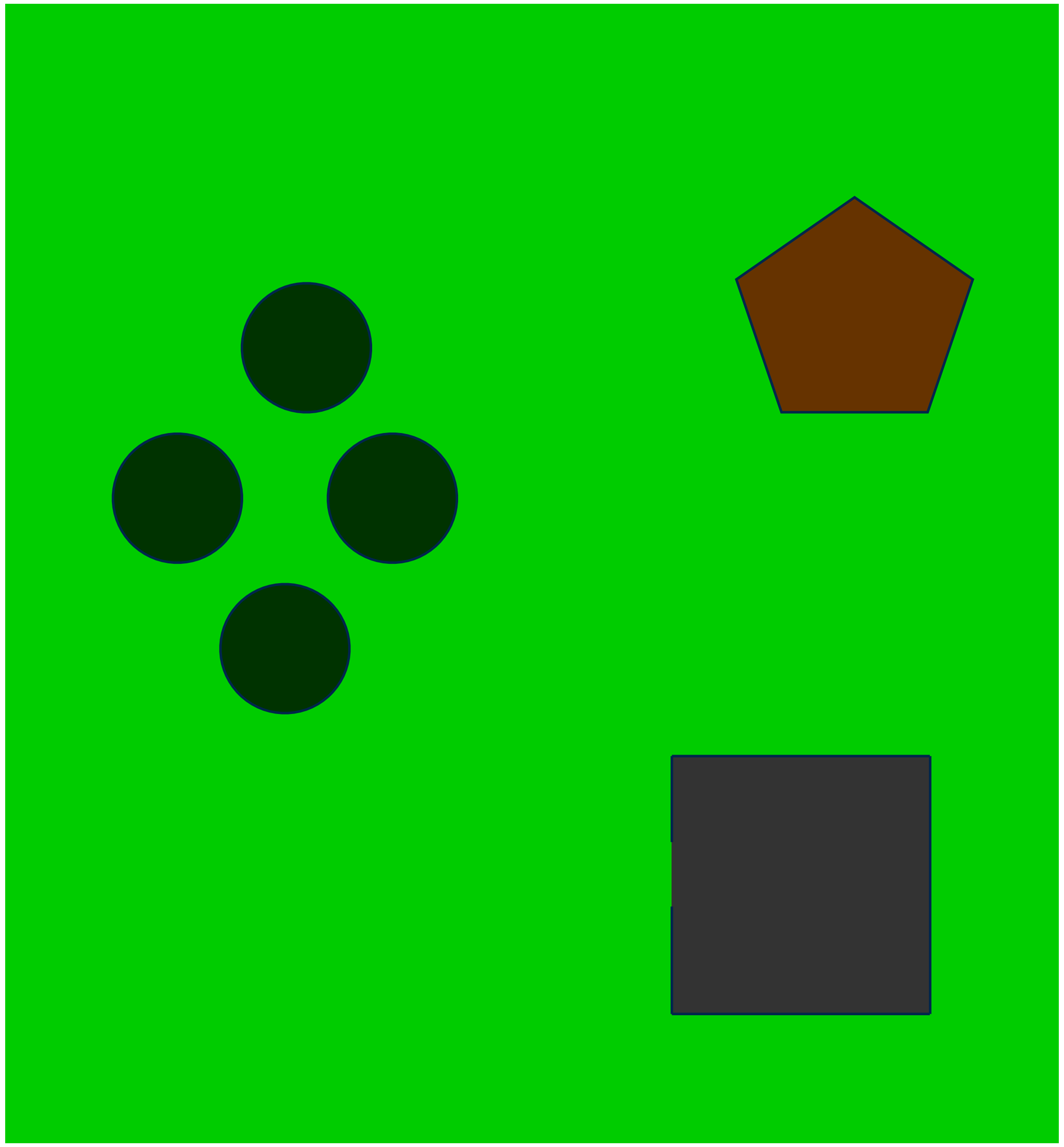
- Path planning is "point to point" where places in world are connected through an unoccupied point
- Such a search space is complex (space of arbitrary curves)
- Necessary to discretize the search space
  - Restrict the start and goal points to a finite set
  - Restrict paths to be along lines (or simple curves) joining points
- Discretized search space forms a graph
  - Nodes are points
  - Edges join nodes reachable along a single curve segment

WAYPOINTS

- The discrete set of points along a path are called waypoints
- How to choose waypoint locations?
- How to determine if there’s a simple path between them?
  - Almost always assume straight lines
- Selection depends on game genre and intended experience

WHERE WOULD YOU PUT WAYPOINTS?





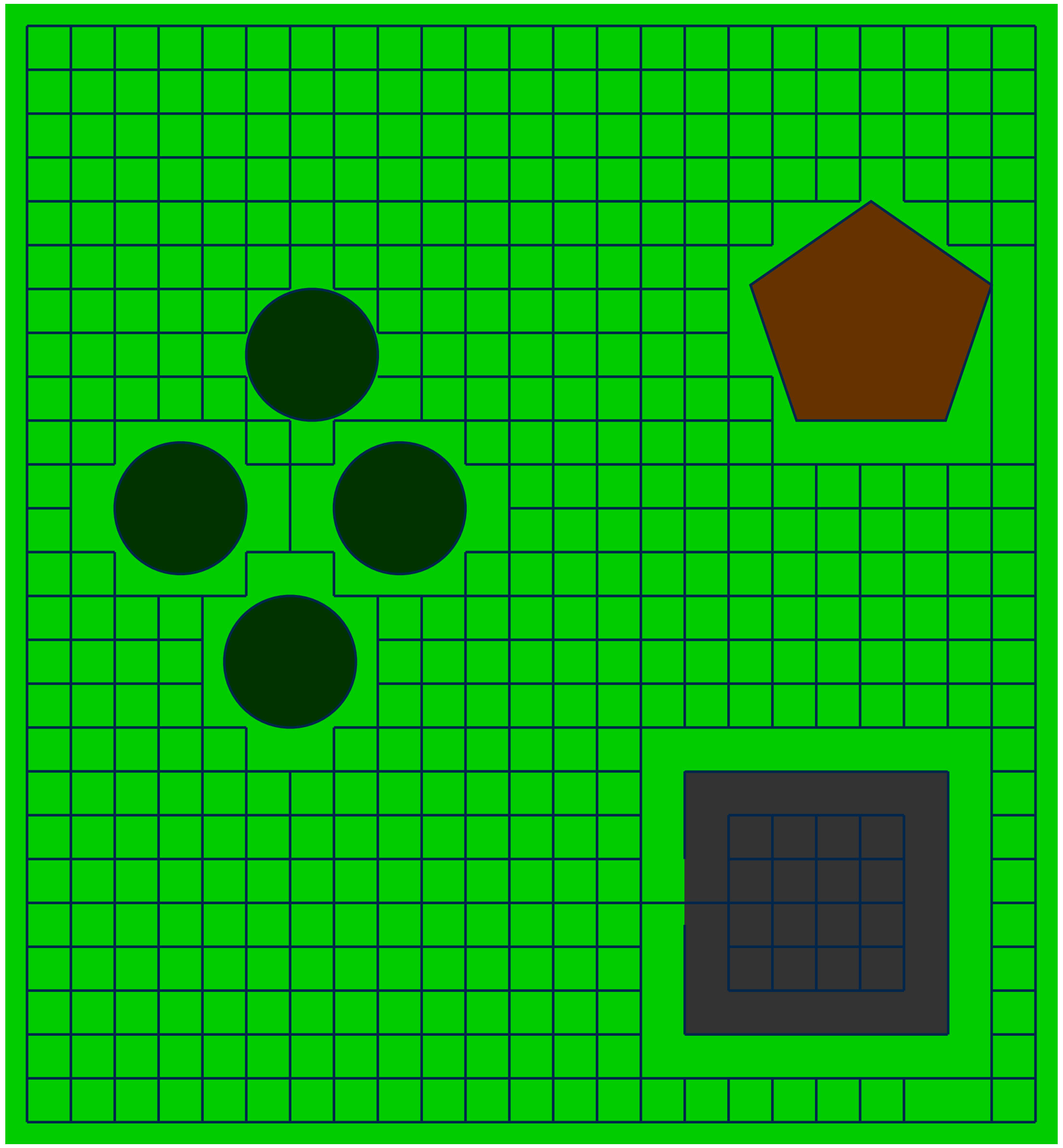
WAYPOINTS BY HAND

- Can place waypoints as part of level design
  - Fine-grain designer control
  - Time-consuming
  - Good choice of waypoints can make the AI seem smarter
- Many heuristics for good places:
  - In doorways
  - Along walls
  - At other discontinuities in the environments
  - At corners
- What are the advantages/disadvantages of these?

WAYPOINTS BY GRID

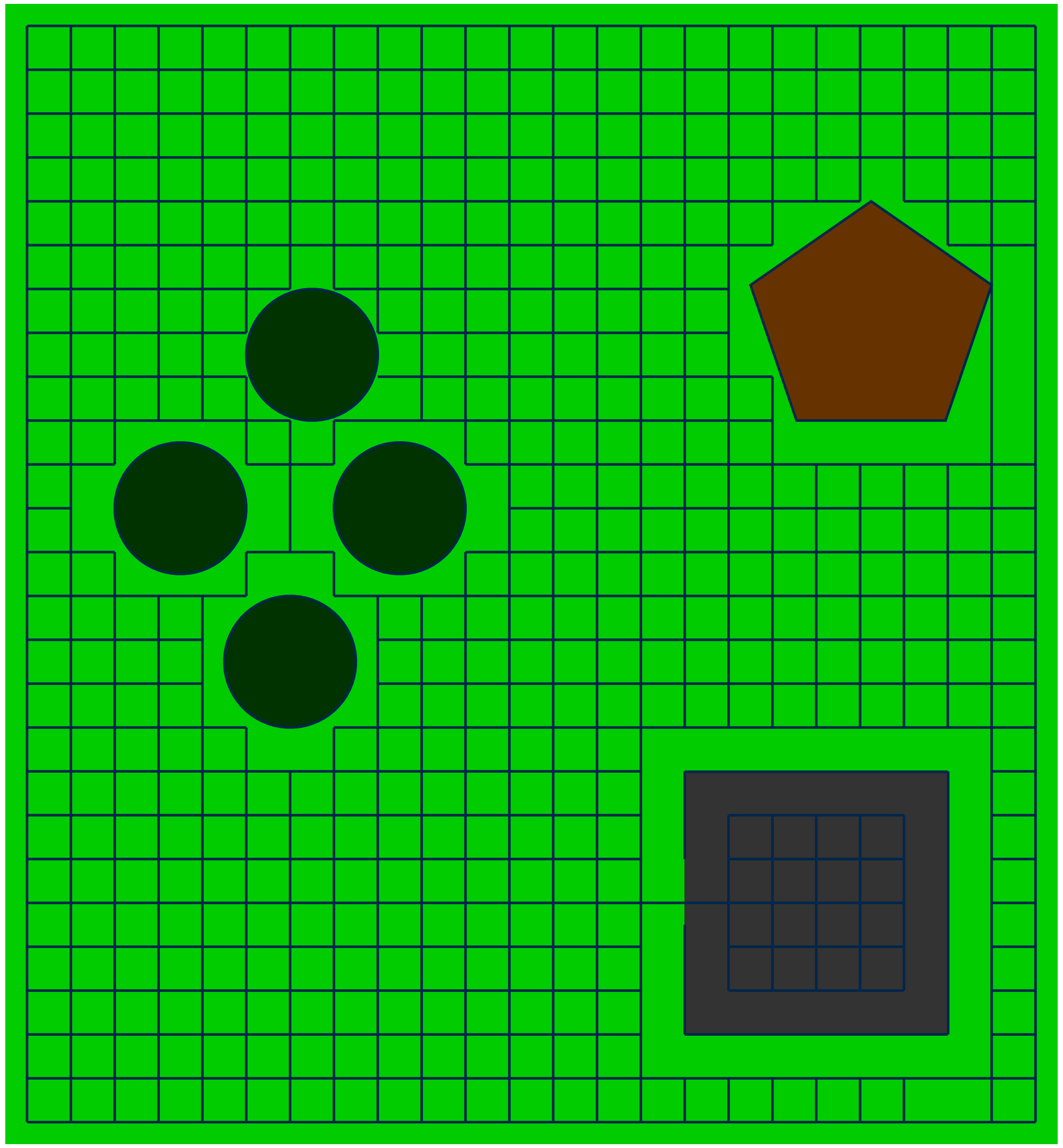
- Place a grid over the world, and put a waypoint at every open grid-point
  - Automated method
  - Potentially implicit to the environment
- Perform an edge/world intersection test to decide which waypoints should be joined
  - Allows movement between immediate (or maybe corner) neighbors

GRID EXAMPLE



- What sorts of environments will this work for?
- What are its advantages?
- What are its problems?

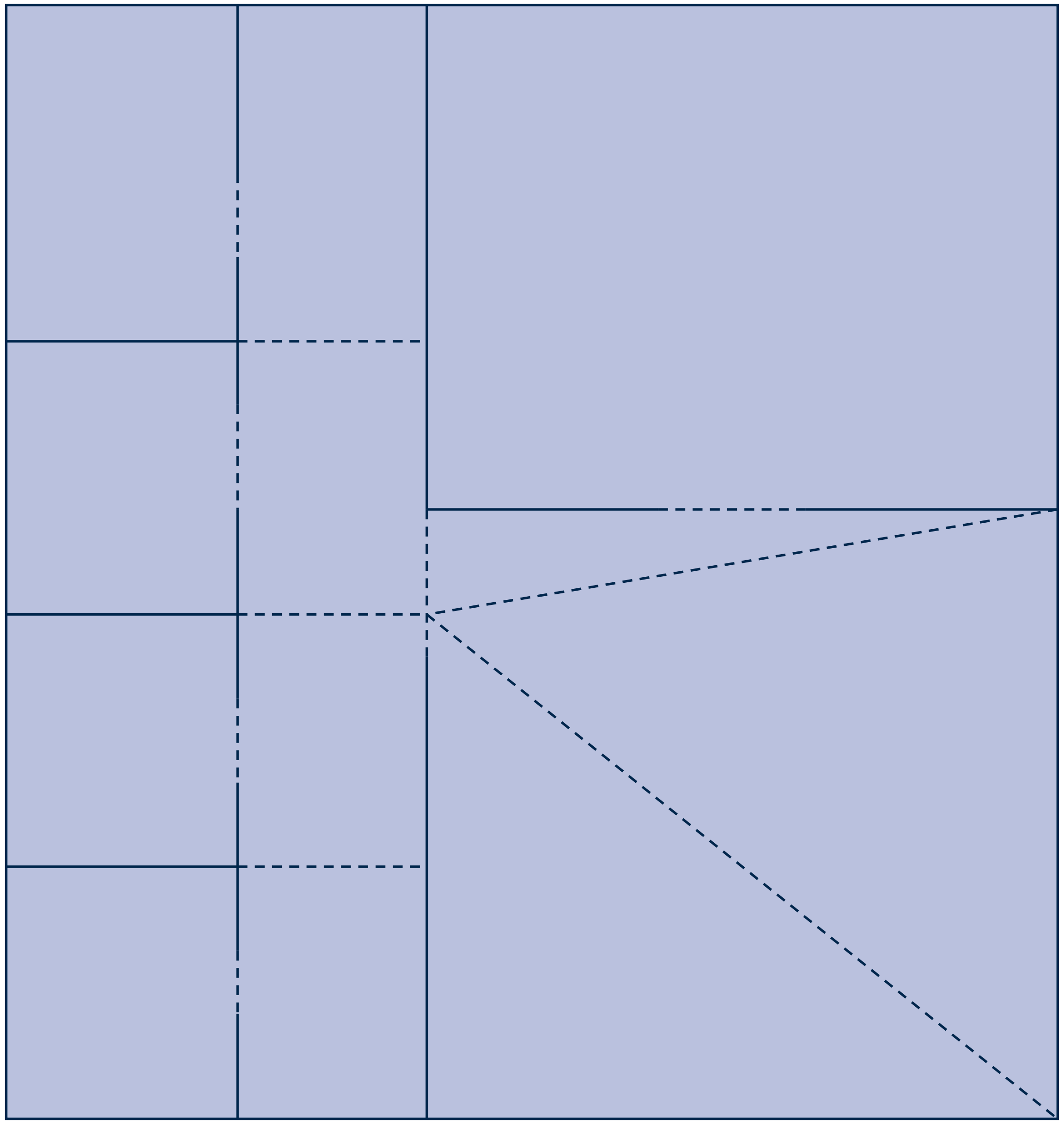
GRID EXAMPLE



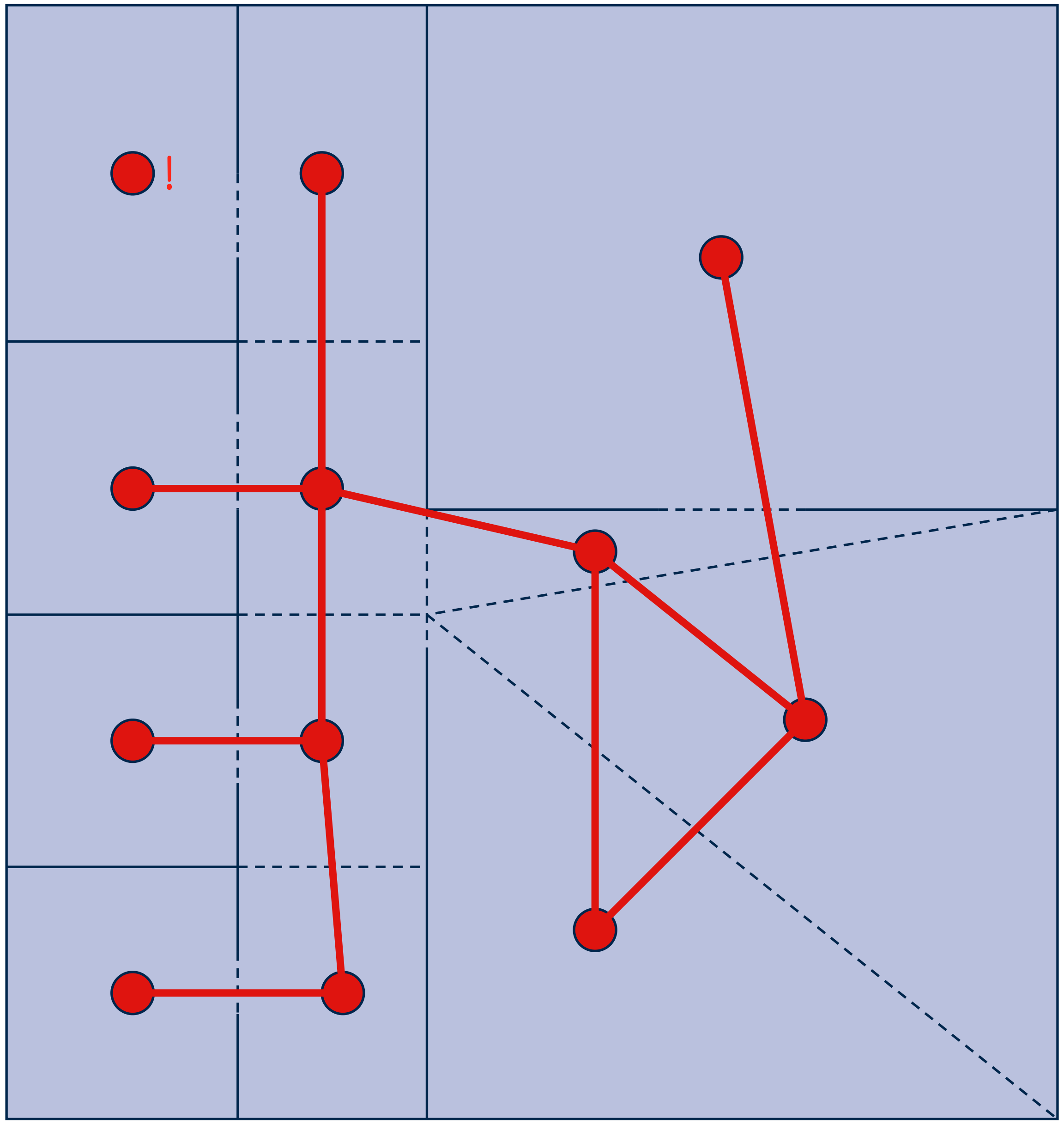
- Potential fixes:
  - Perturb grid to move edges closer to obstacles
  - Adjust grid resolution
  - Joins between outside and inside waypoints

WAYPOINTS FROM POLYGONS

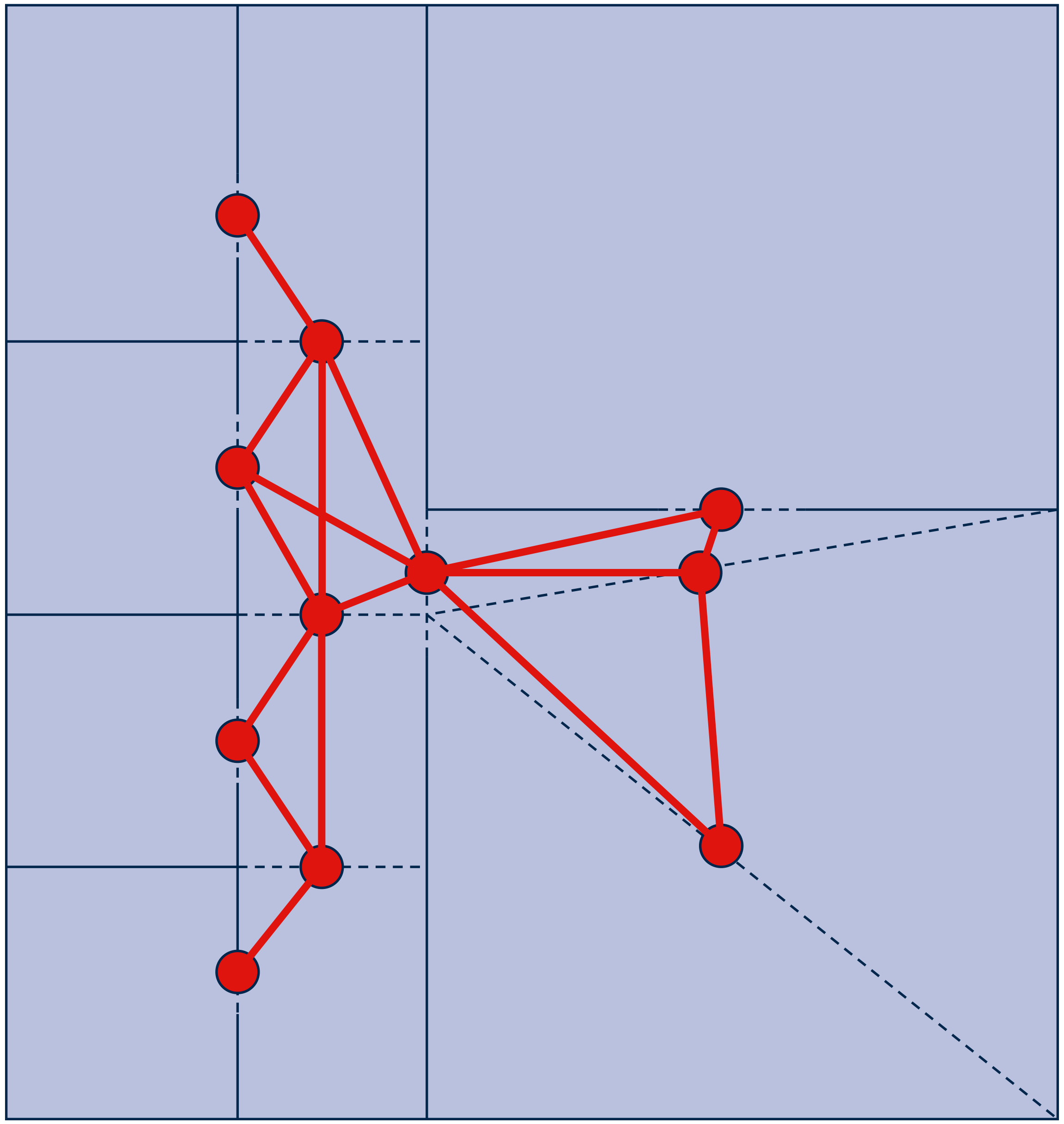
- Choose waypoints based on the floor polygons in your world
- Or use specific polygons that generate waypoints
- How do we go from polygons to waypoints?



WAYPOINTS FROM POLYGONS



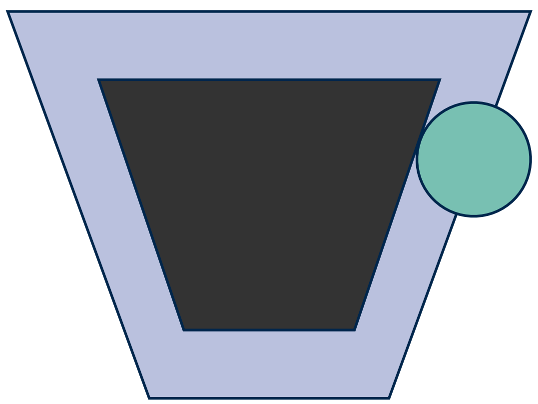
Waypoints at the center of polygons



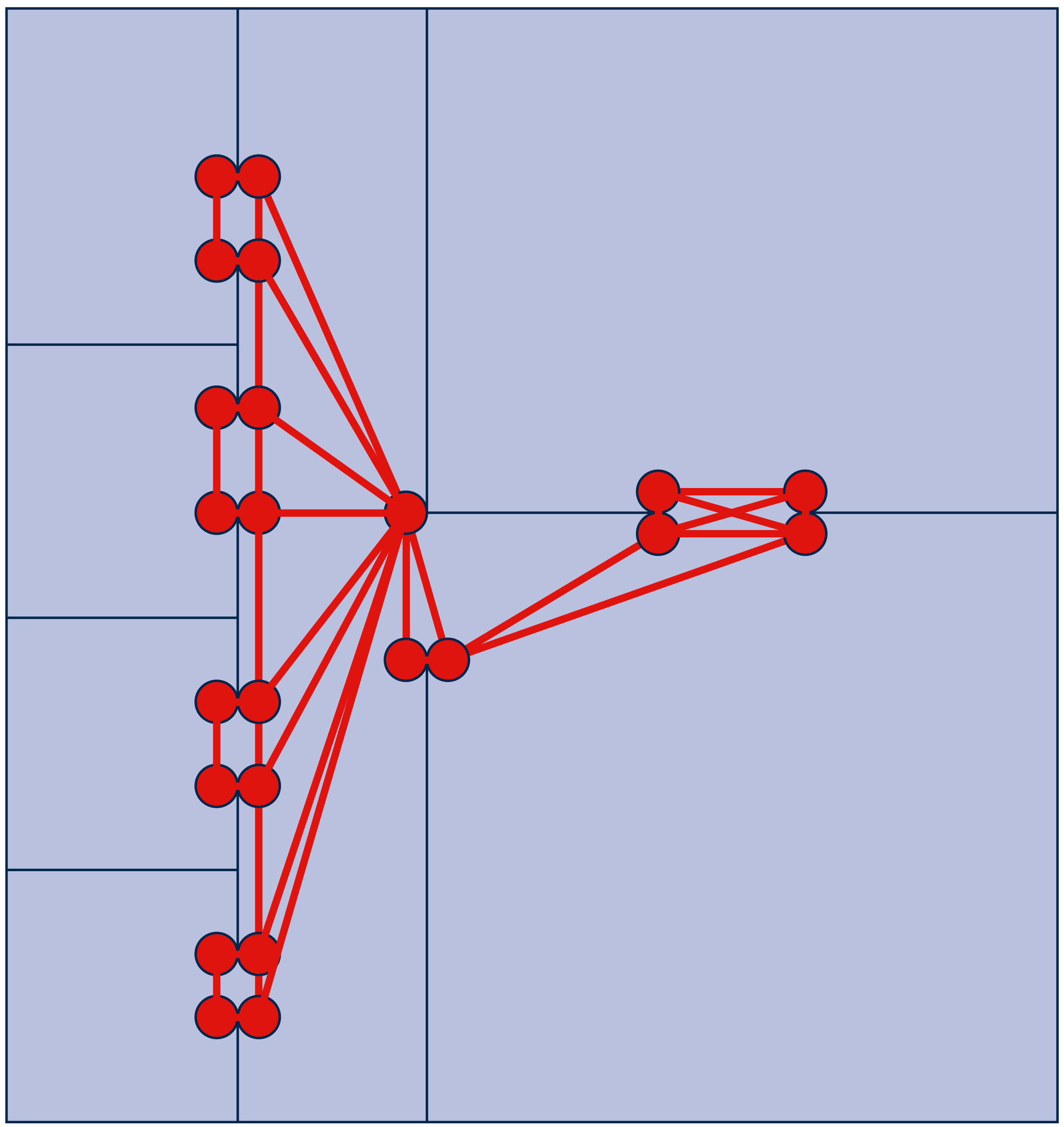
Add waypoints along polygon walls

WAYPOINTS FROM CORNERS

- Place waypoints at every convex corner of the obstacles
  - Take into account width of moving objects
  - Or, compute corners of offset polygons
- Connects all corners that see each other
- Results in the shortest path
- Some unnatural paths may result
  - Characters will stick to walls



WAYPOINTS FROM CORNERS

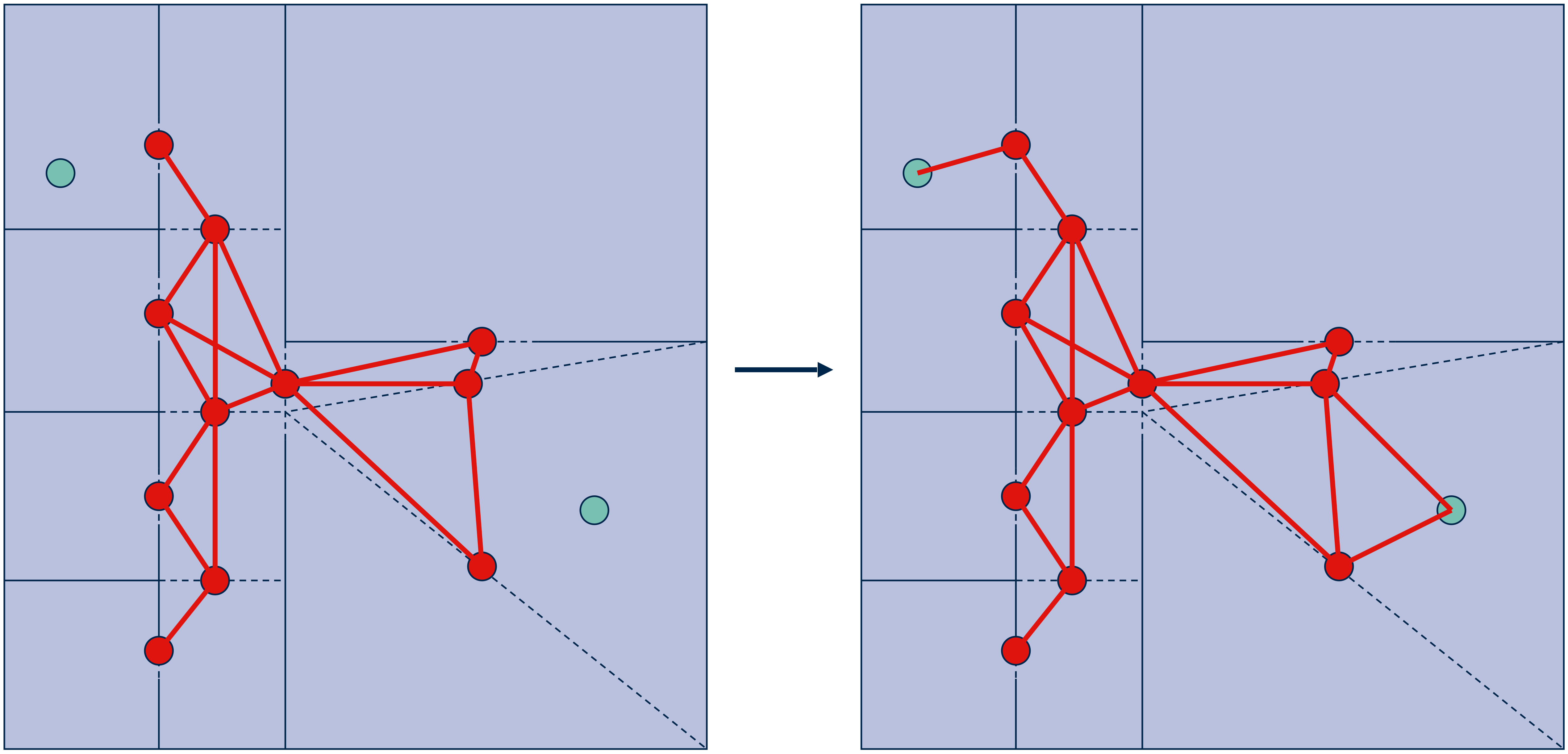


- Note that not every edge is drawn
- Produces very dense graphs

ENTERING AND EXITING THE PATHWAY

- Don't restrict the character to waypoints or graph edges
  - Not necessarily a problem with grid methods
- To enter, find the closest waypoint and move toward that
  - Or, find a waypoint in the direction of the goal
  - Or, try all potential starting waypoints and see which gives the shortest path
- To exit, jump off at closest waypoint to goal
  - Ideally agent can go straight to the goal from waypoint
- Best option: Add a temporary waypoint at the precise start/finish point, and join it to nearby waypoints

ENTERING AND EXITING THE PATHWAY



WE HAVE A PATH…NOW WHAT?

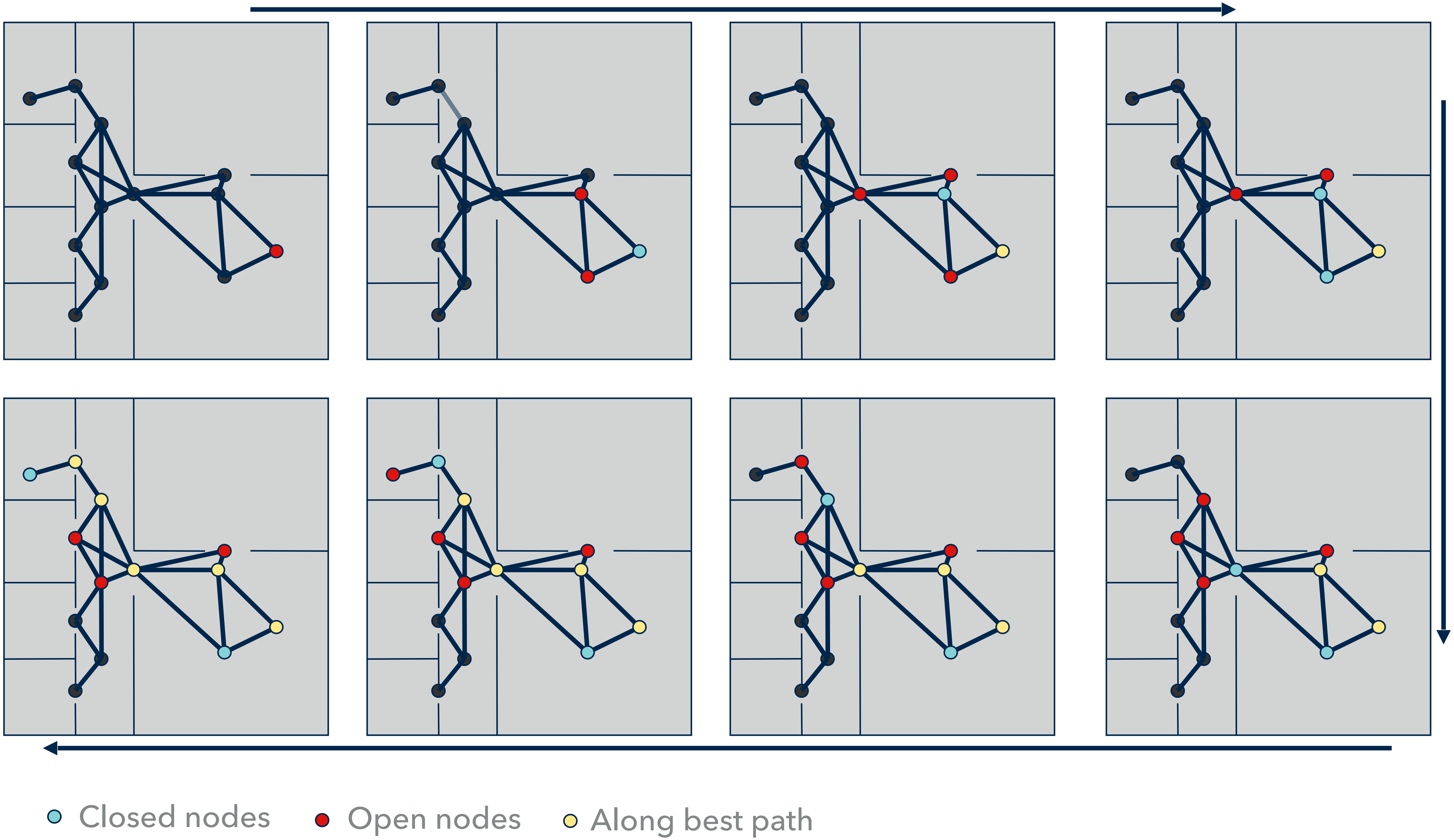
BEST-FIRST-SEARCH

- Search out from start node
- Maintain two sets of nodes:
  - **Open nodes** - reached nodes that may or may not be on best path
  - **Closed nodes** - best path to these nodes are known
- Open nodes sorted by cost

BFS IN ACTION

- Expand "best" open node
  - If it's the goal, we're done
  - If not, move the "best" open node to the closed set
  - Add any nodes reachable from the "best" node to the open set (unless already there or closed)
  - Update the cost for nodes reachable from the "best" node
  - New cost is *min*(old-cost, cost-through-best)
- Repeat

EXPANDING FRONTIER



BEST-FIRST-SEARCH PROPERTIES

- Precise properties depend on how “best” is defined
- But in general:
  - Will always find any reachable goal
- To store the best path:
  - Keep a pointer in each node *n* to the previous node along the best path to *n*
  - Update these as nodes are added to the open set and as nodes are expanded (i.e. whenever the cost changes)
  - To find path to goal, trace pointers back from goal nodes

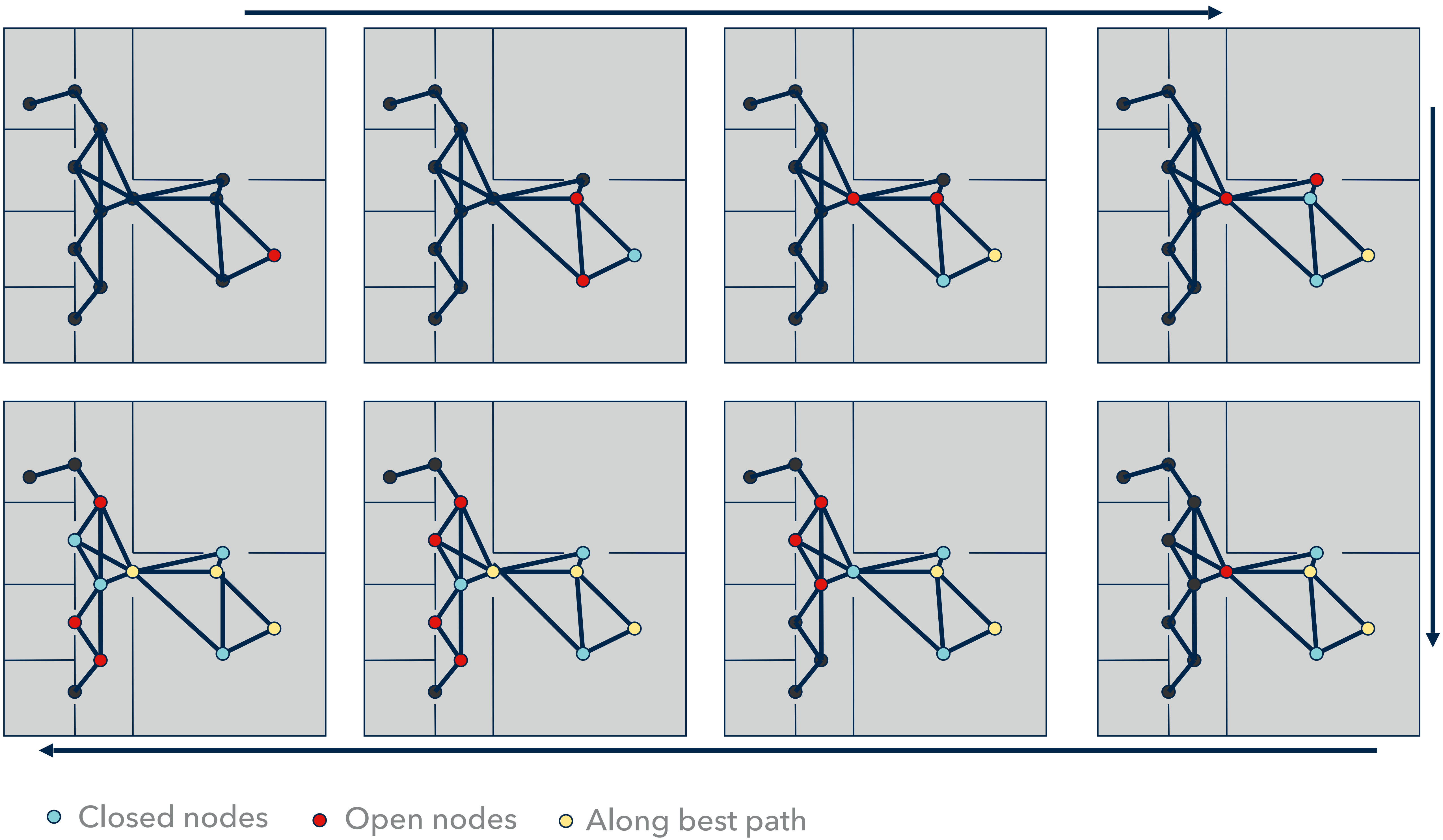
DEFINING BEST

- *g(n)*: The current known best cost for getting **to** a node from the start point
    - Can be computed based on the cost of traversing each edge along the current shortest path to *n*
- *h(n)*: The current estimate for how much more it will cost to get **from** a node to the goal
    - A **heuristic**: The exact value is unknown but this is your best guess
    - Some algorithms place conditions on this estimate

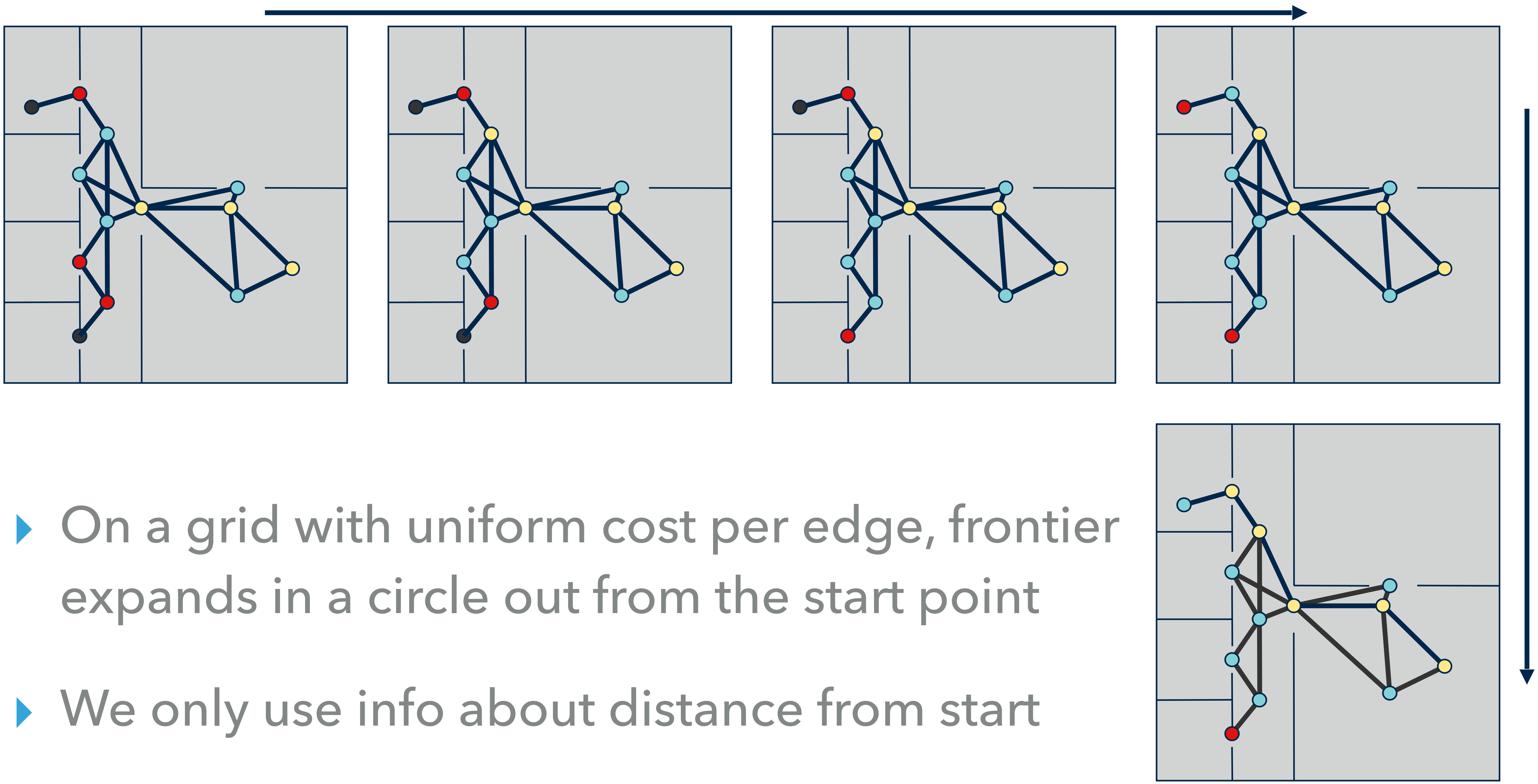
USING G(N) ONLY (BREADTH FIRST SEARCH)

- Define “best” according to $f(n) = g(n)$ (shortest known path from the start to the node)

BREADTH FIRST SEARCH



BREADTH FIRST SEARCH



- On a grid with uniform cost per edge, frontier expands in a circle out from the start point
- We only use info about distance from start

Closed nodes Open nodes Along best path

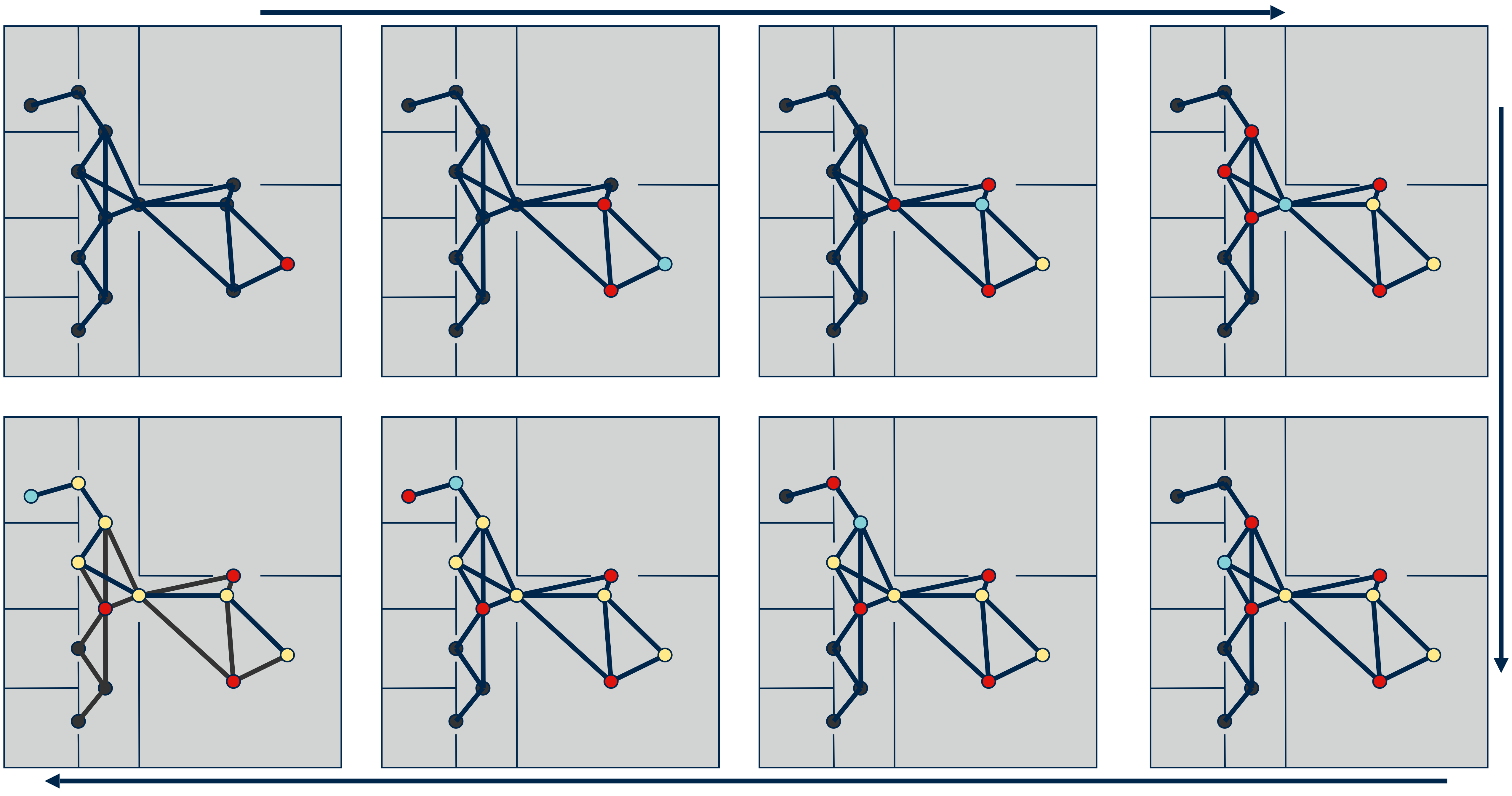
BREADTH FIRST SEARCH

- Is it optimal?
  - Is the goal node along the shortest path?
- Is it efficient?
  - How many nodes does it explore?

USING H(N) ONLY (GREEDY SEARCH)

- Define “best” according to $f(n) = h(n)$ (our best guess)
  - Behavior depends on choice of heuristic
  - Straight line distance is a good choice
- Set the cost for a node with no exit to be infinite
  - If we expand such a node, our guess of the cost was wrong

GREEDY SEARCH (STRAIGHT-LINE-DISTANCE HEURISTIC)



Closed nodes Open nodes Along best path

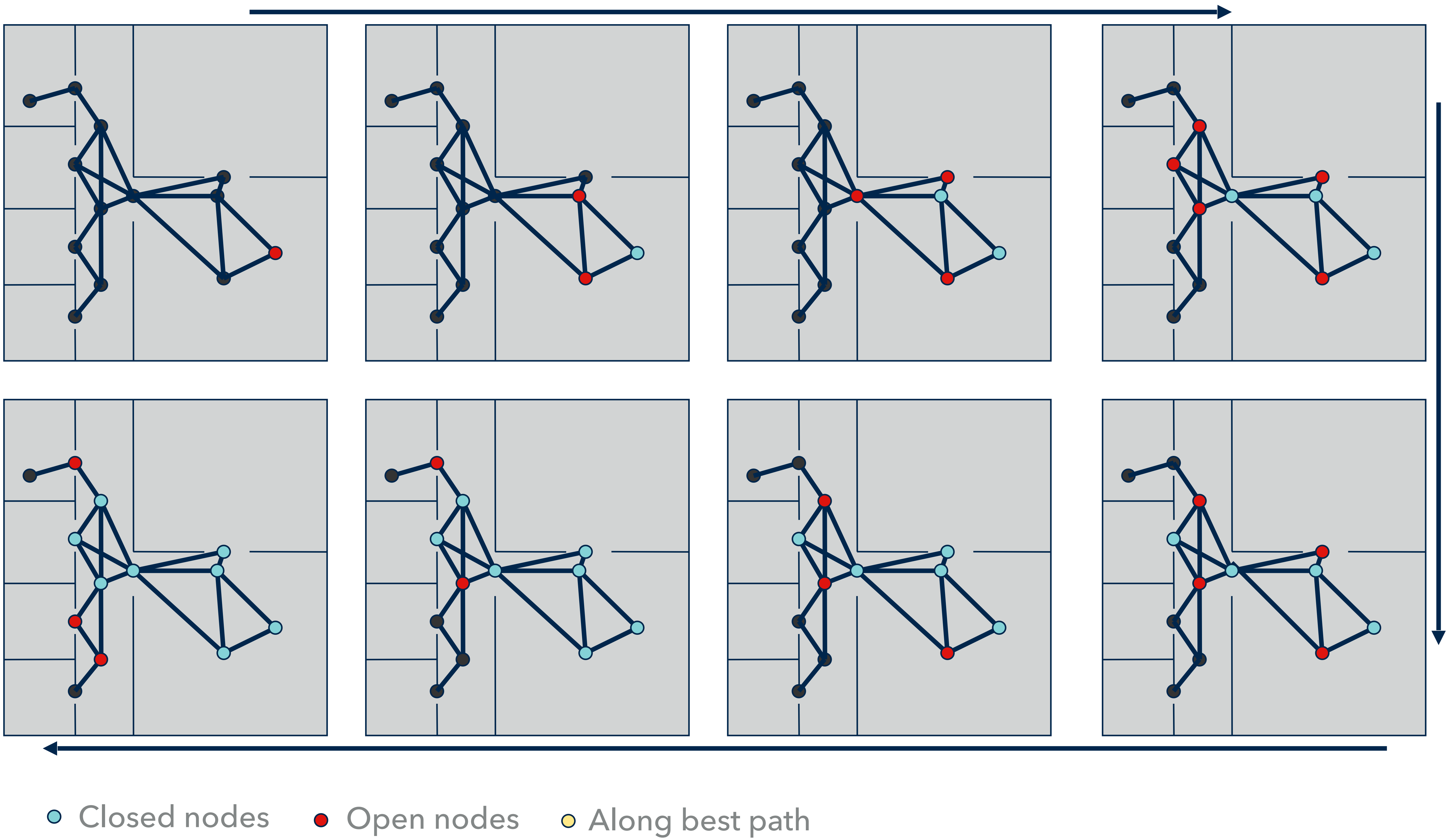
GREEDY-SEARCH

- Is it optimal?
  - When the goal node is expanded, is it along the shortest path?
- Is it efficient?
  - How many nodes does it explore?

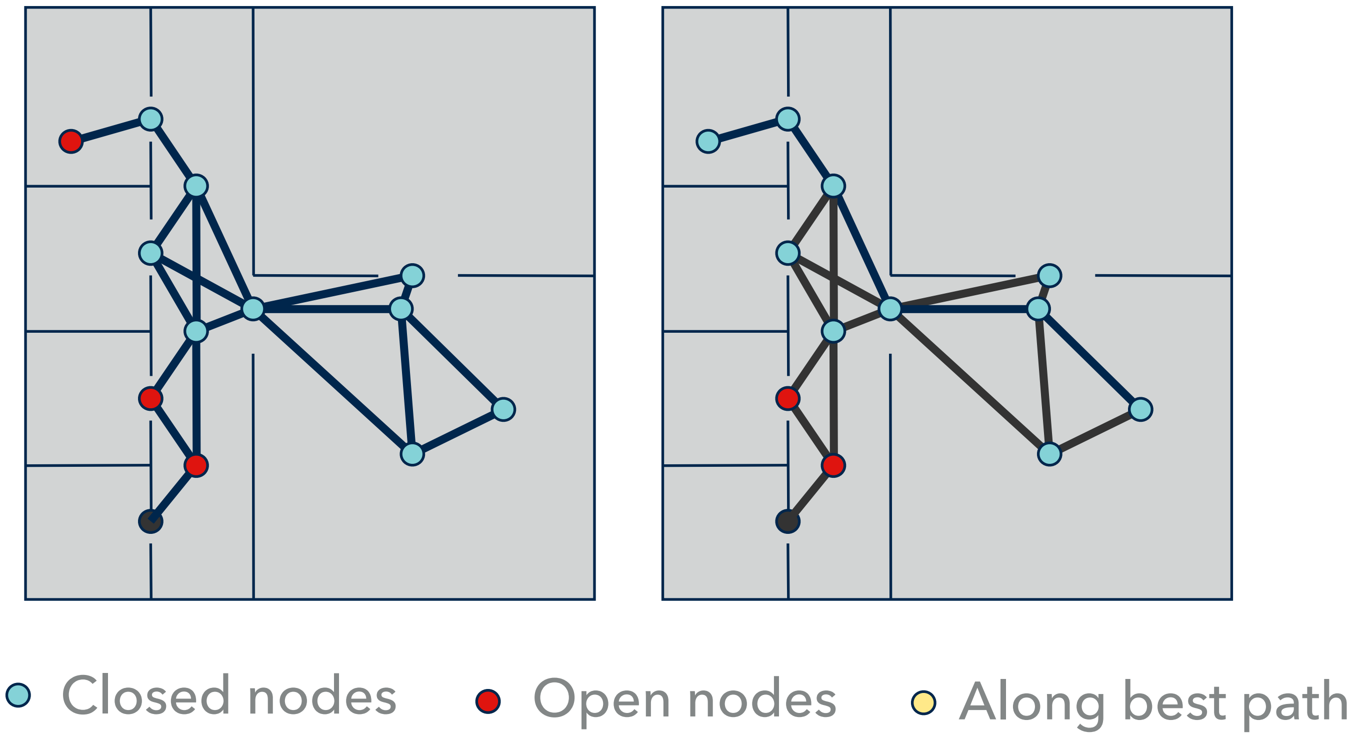
A* SEARCH

- *f(n)*: The current best estimate for the best path through a node: *f(n)=g(n)+h(n)*
- This expands nodes according to best estimated total path cost
- Is it optimal?
  - Depends on *h(n)*
- Is it efficient?
  - Most efficient of any optimal algorithm that uses the same *h(n)*
- A* is the ubiquitous algorithm for path planning in games
  - Much effort goes into making it fast, and making it produce pretty looking paths
  - More articles on it than you can poke a stick at

A* SEARCH (STRAIGHT-LINE-DISTANCE HEURISTIC)



A* SEARCH



- Note that A* expands fewer nodes than breadth-first, but more than greedy
- It's the price you pay for optimality
- Keys are:
    - Data structure for a node
    - Priority queue for sorting open nodes
    - Nodes track their predecessor to reconstruct path

A * PATHFINDING EXAMPLE

- https://www.youtube.com/watch?v=Ju7IxDNbt-4