the gamedesigninitiative at cornell university

Lecture 21

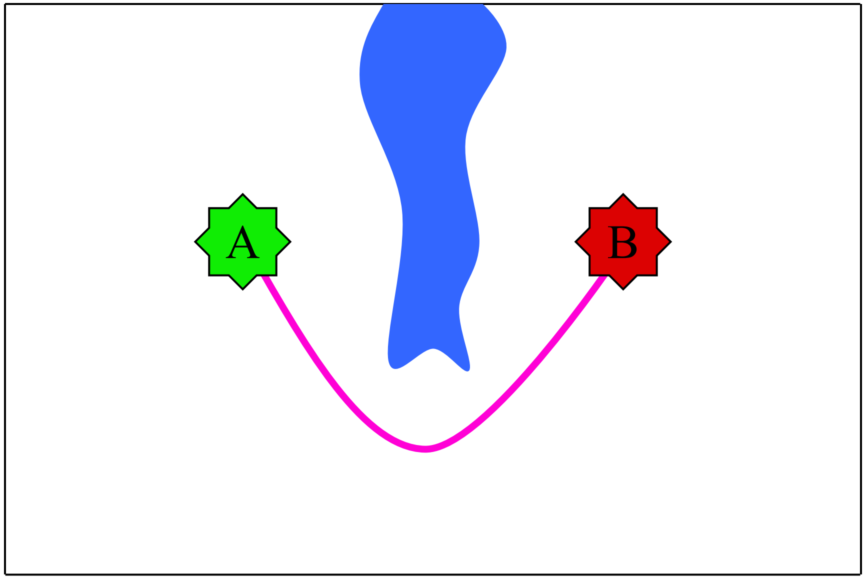
Pathfinding

Take Away for this Lecture

- What are the primary goals for pathfinding?
- Identify advantages/disadvantages of A*
  - In what situations does A* fail (or look bad)?
  - What can we do to fix these problems?
- Why combine steering and A*?
  - Is this combination always appropriate?
- What do commercial games use?

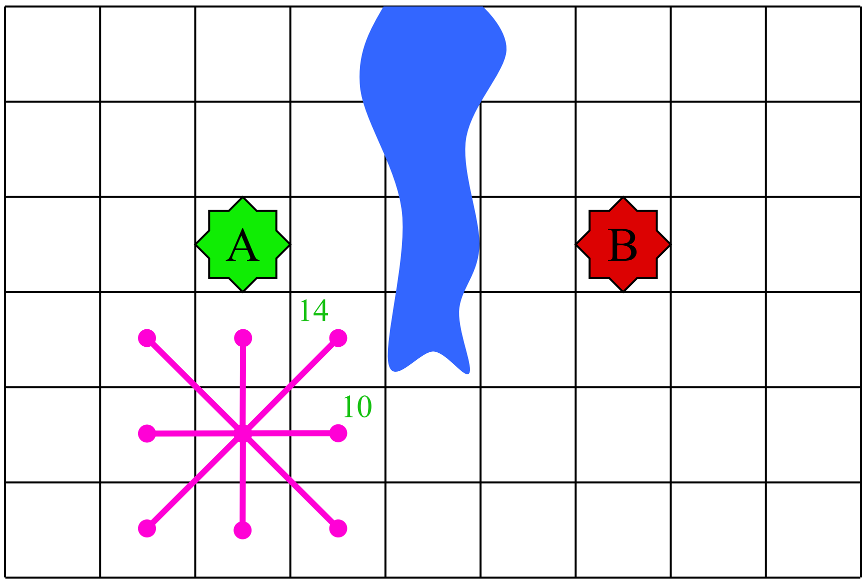
Pathfinding

- You are given
  - Starting location *A*
  - Goal location *B*
- Want **valid** path *A* to *B*
  - Avoid "impassible" terrain
  - Eschew hidden knowledge
- Want **natural** path *A* to *B*
  - Reasonably short path
  - Avoid unnecessary turns
  - Avoid threats in the way



Abstraction: Grid & Graph

- Break world into grid
  - Roughly size of NPCs
  - Terrain is all-or-nothing
  - Majority terrain of square
  - Terrain covering "center"
- Gives us a weighted graph
  - Nodes are grid centers
  - Each node has 8 neighbors
  - Weight = distance/terrain
- **Search for shortest path**



- Real distance not required
  - 14:10 ratio for diagonals
  - Allows us to use integers

Breadth-First Search (Lab 2)

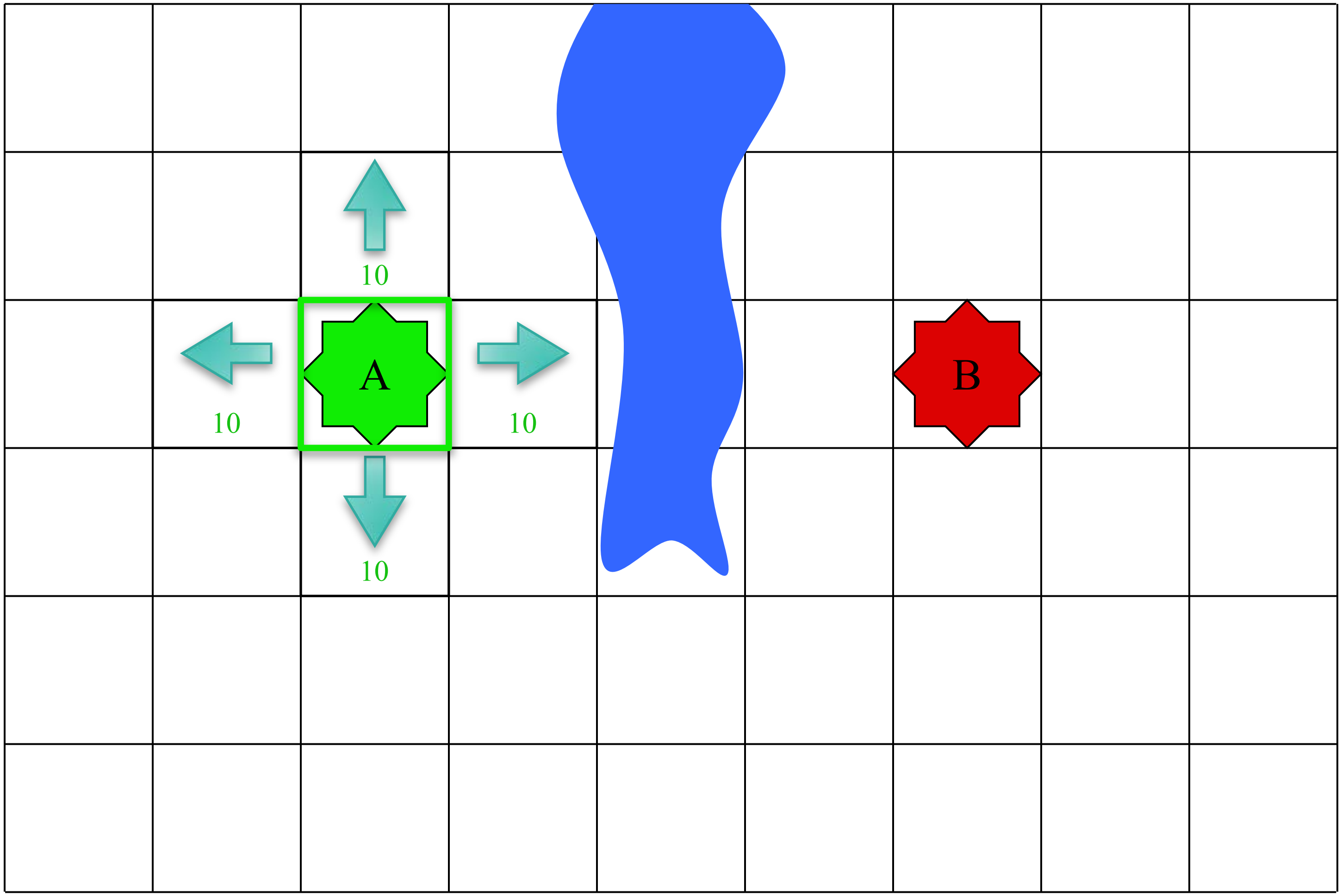
Intuition

- **Search maintains**
  - Current node, initially **start**
  - List of nodes to visit
- **Basic Steps**
  - Have we reached the **goal**?
  - Add neighbors to *end* of list
  - Work from *first* node in list
  - Process "first-in first-out"

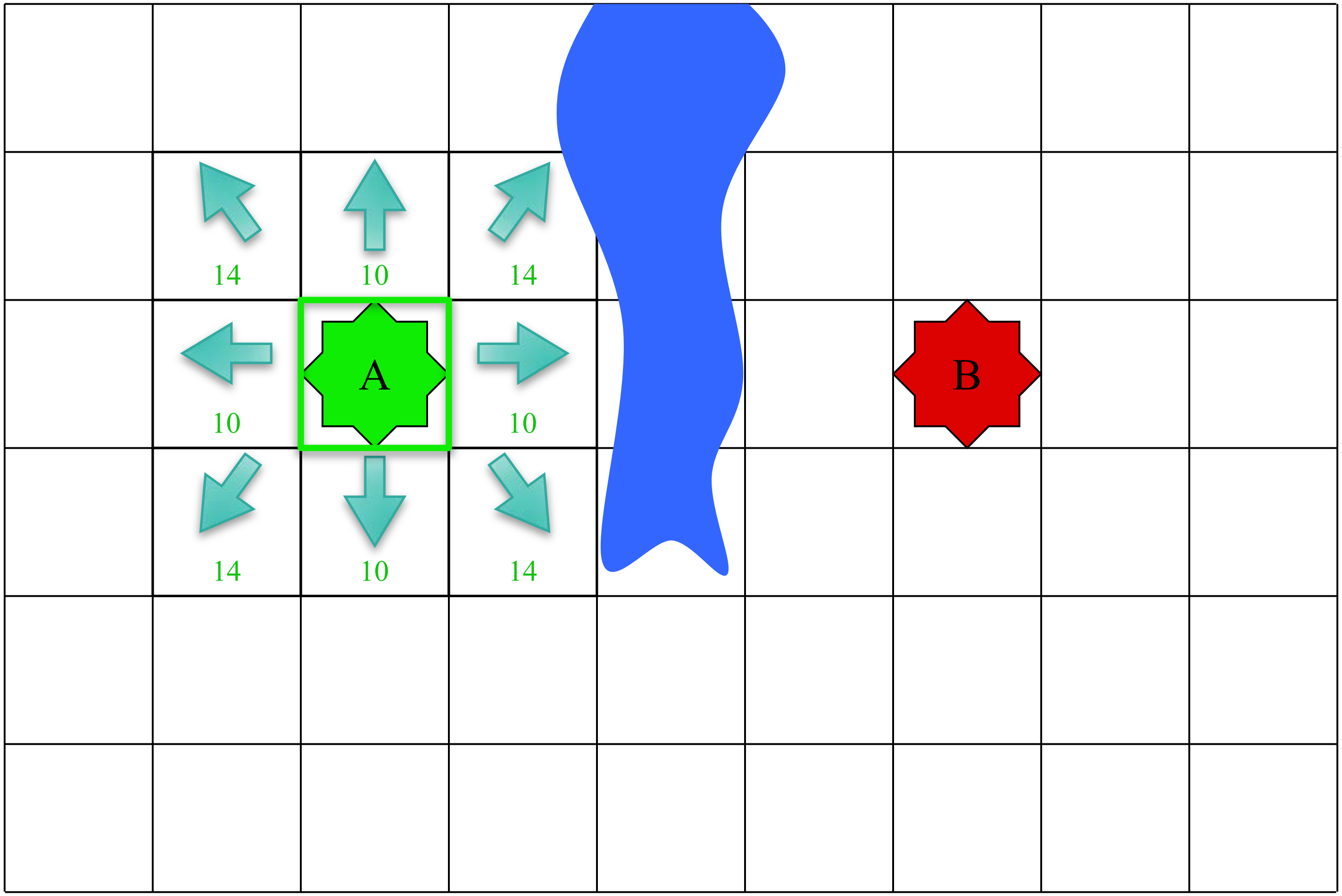
Algorithm

```
n = start; L = { };
while (n not goal) {
  add n to visited;
  N(n) = unvisited neighbors
  foreach (m ∈ N(n)) {
    add m to end of L;
  }
  n = removeFirst(L);
}
return path to goal;
```

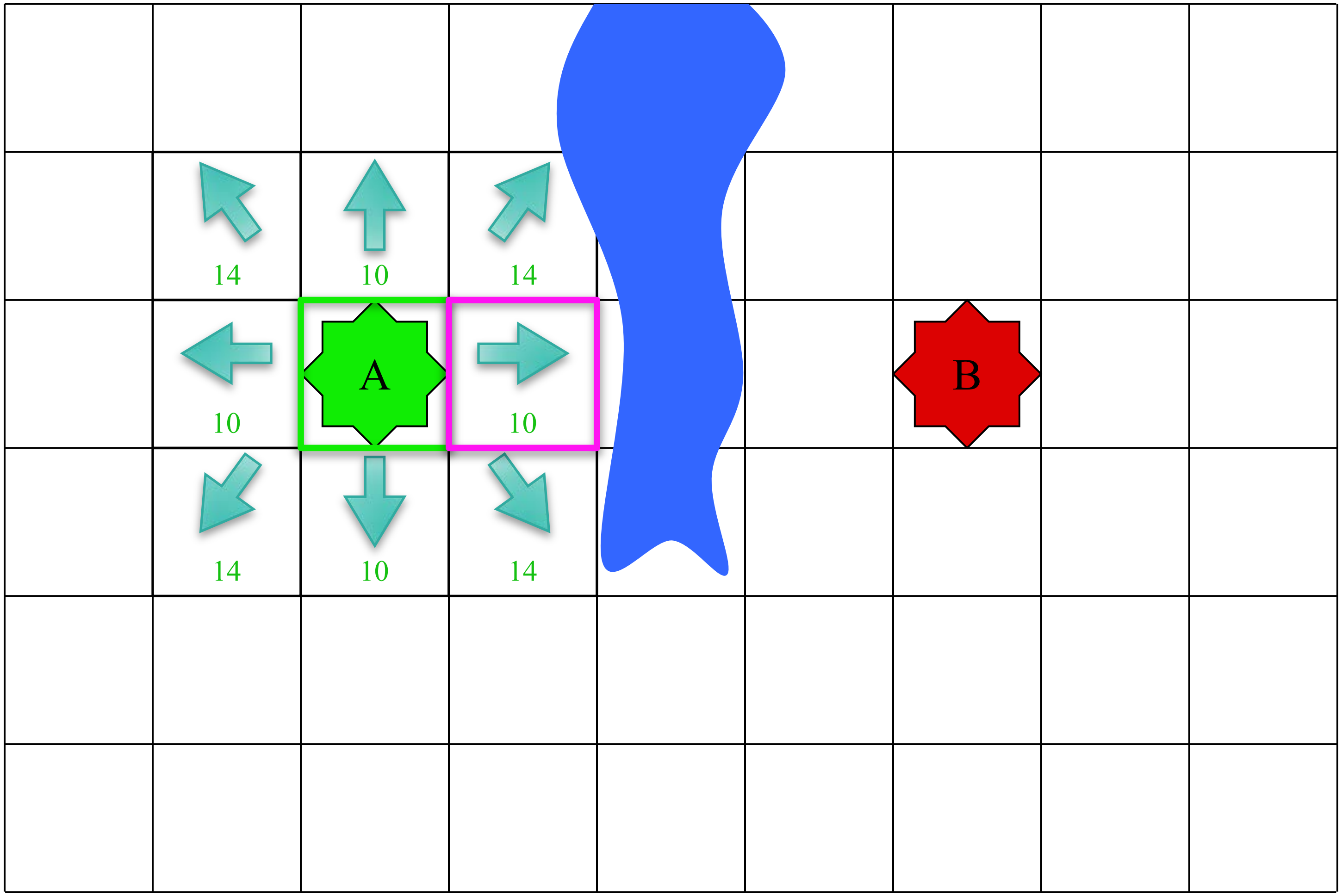
Pathfinding: Breadth-First



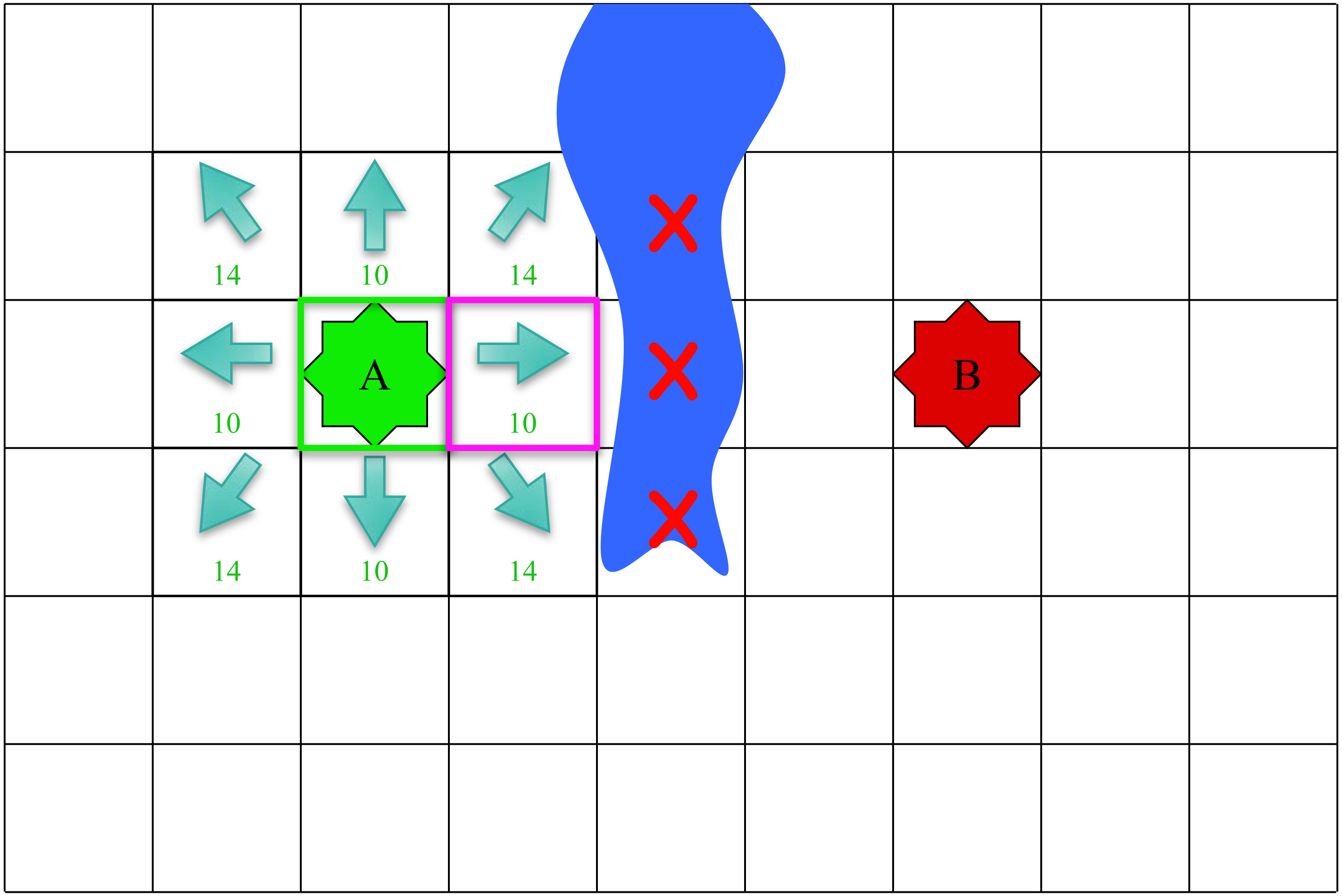
Pathfinding: Breadth-First



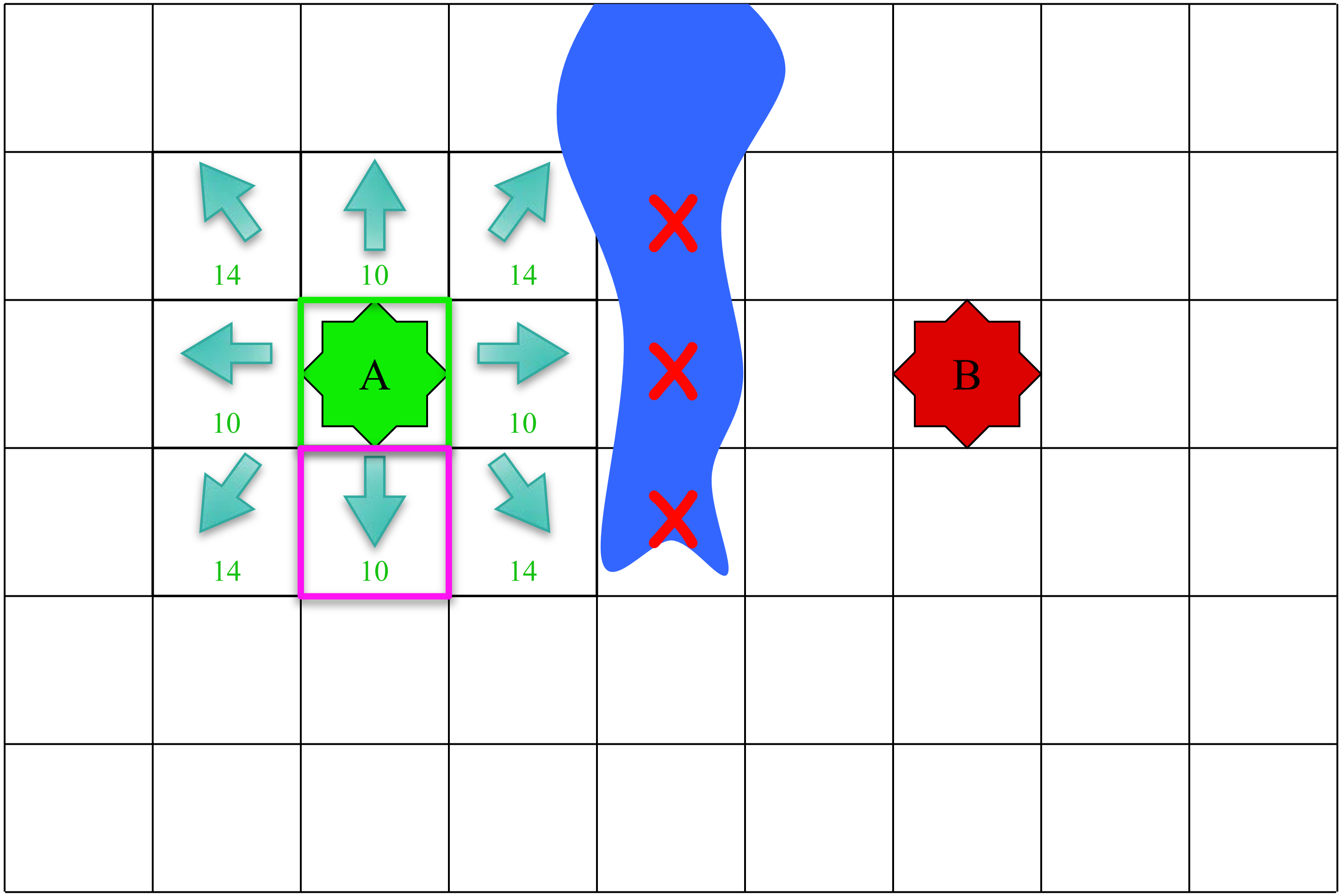
Pathfinding: Breadth-First



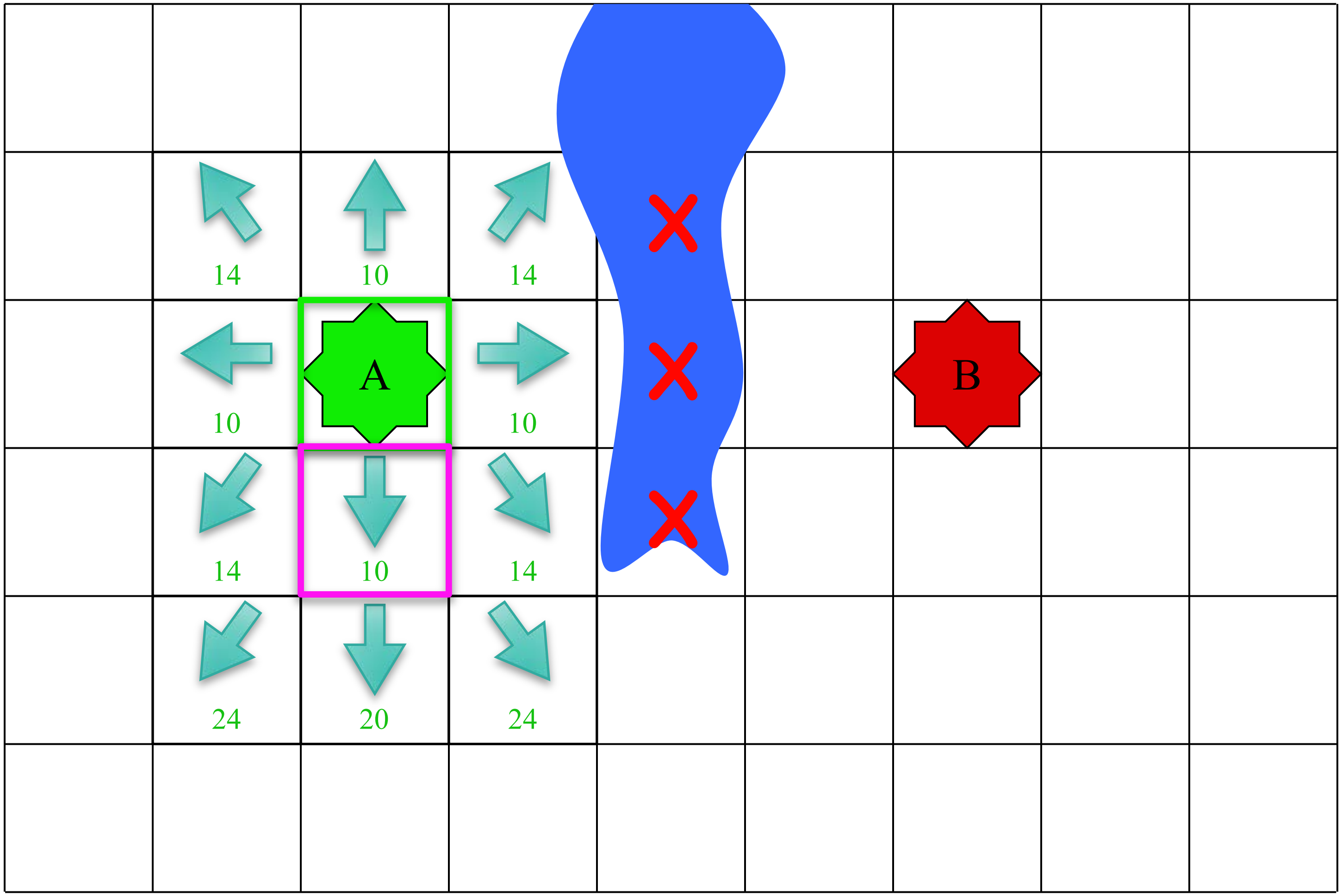
Pathfinding: Breadth-First



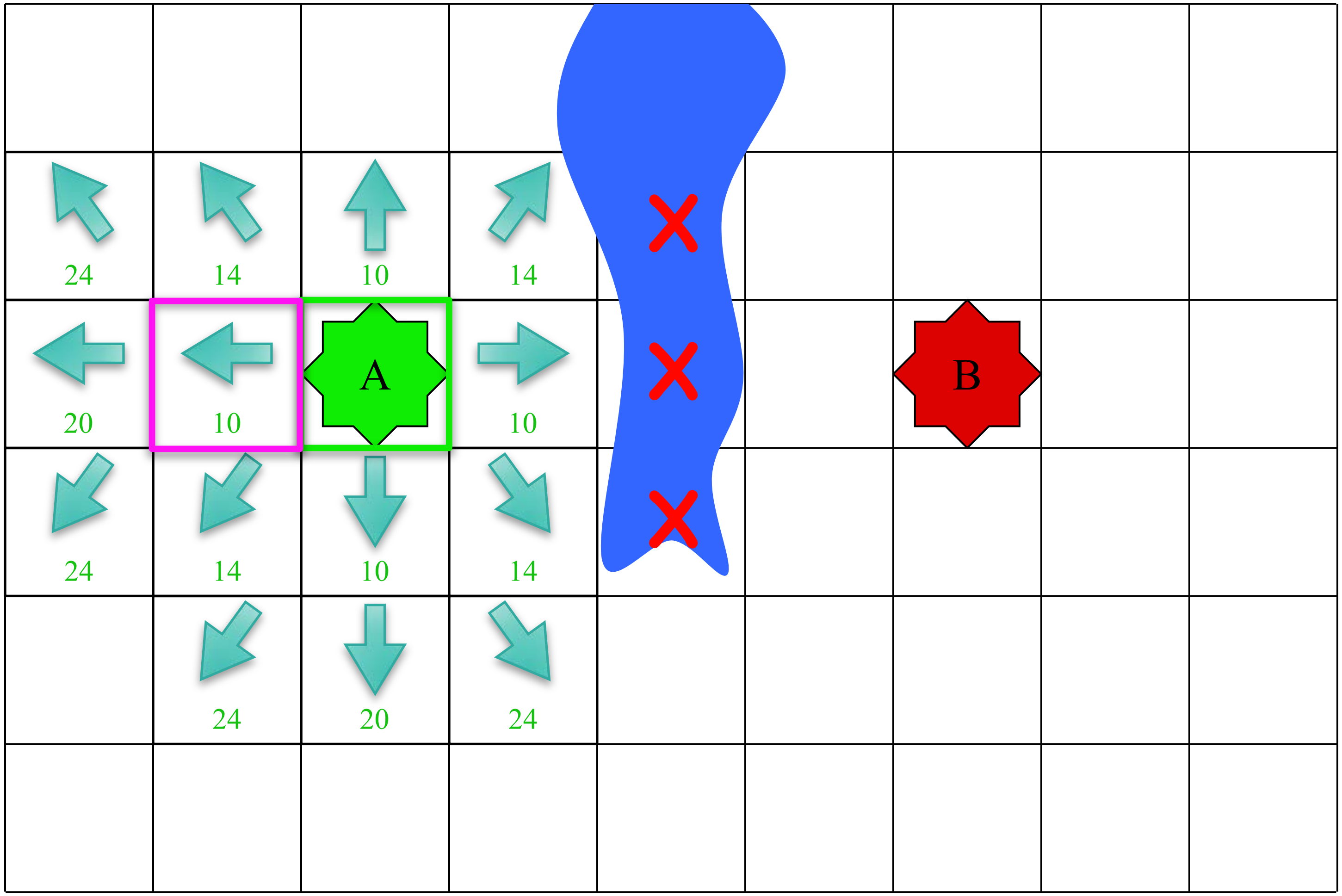
Pathfinding: Breadth-First



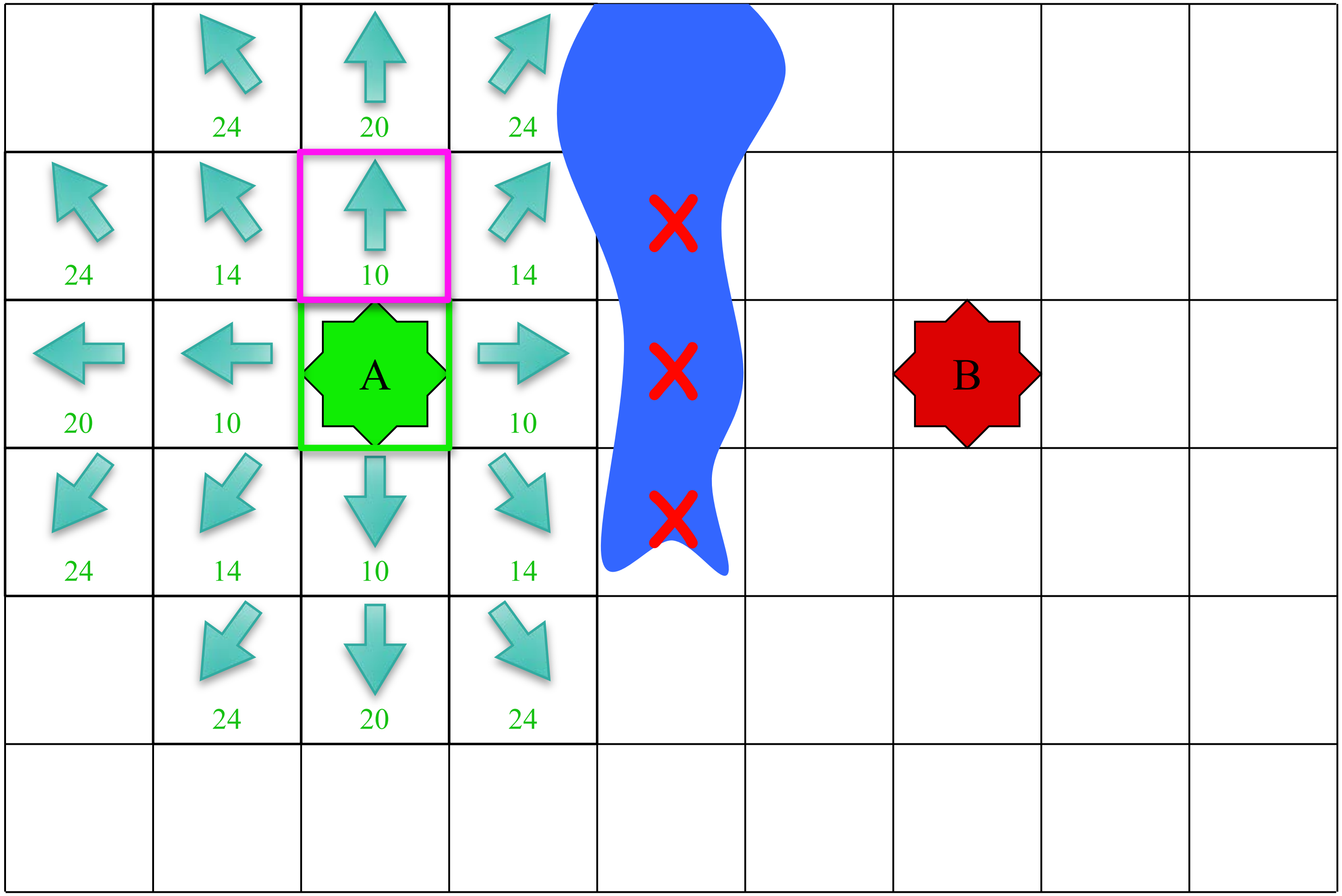
Pathfinding: Breadth-First



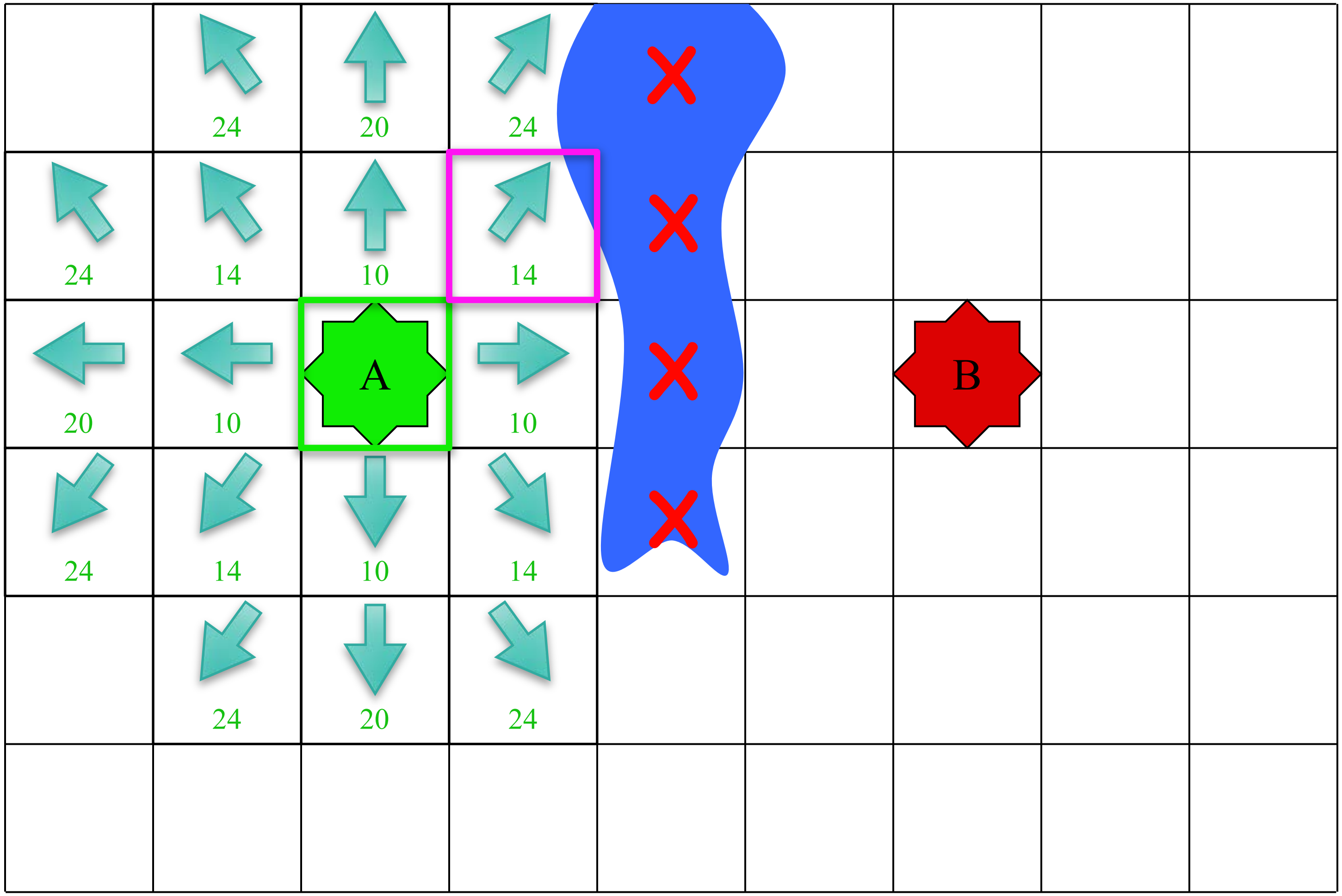
Pathfinding: Breadth-First



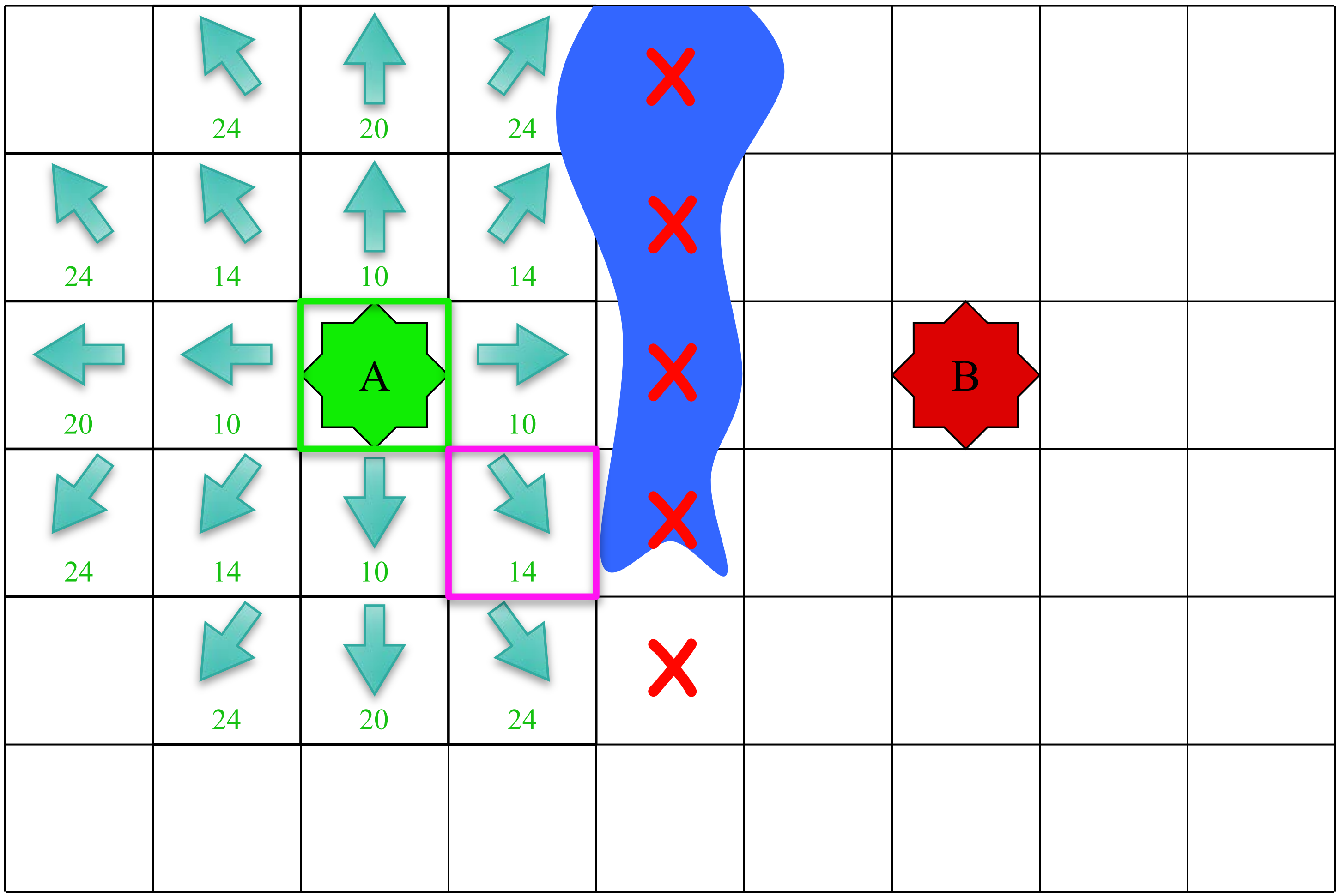
Pathfinding: Breadth-First



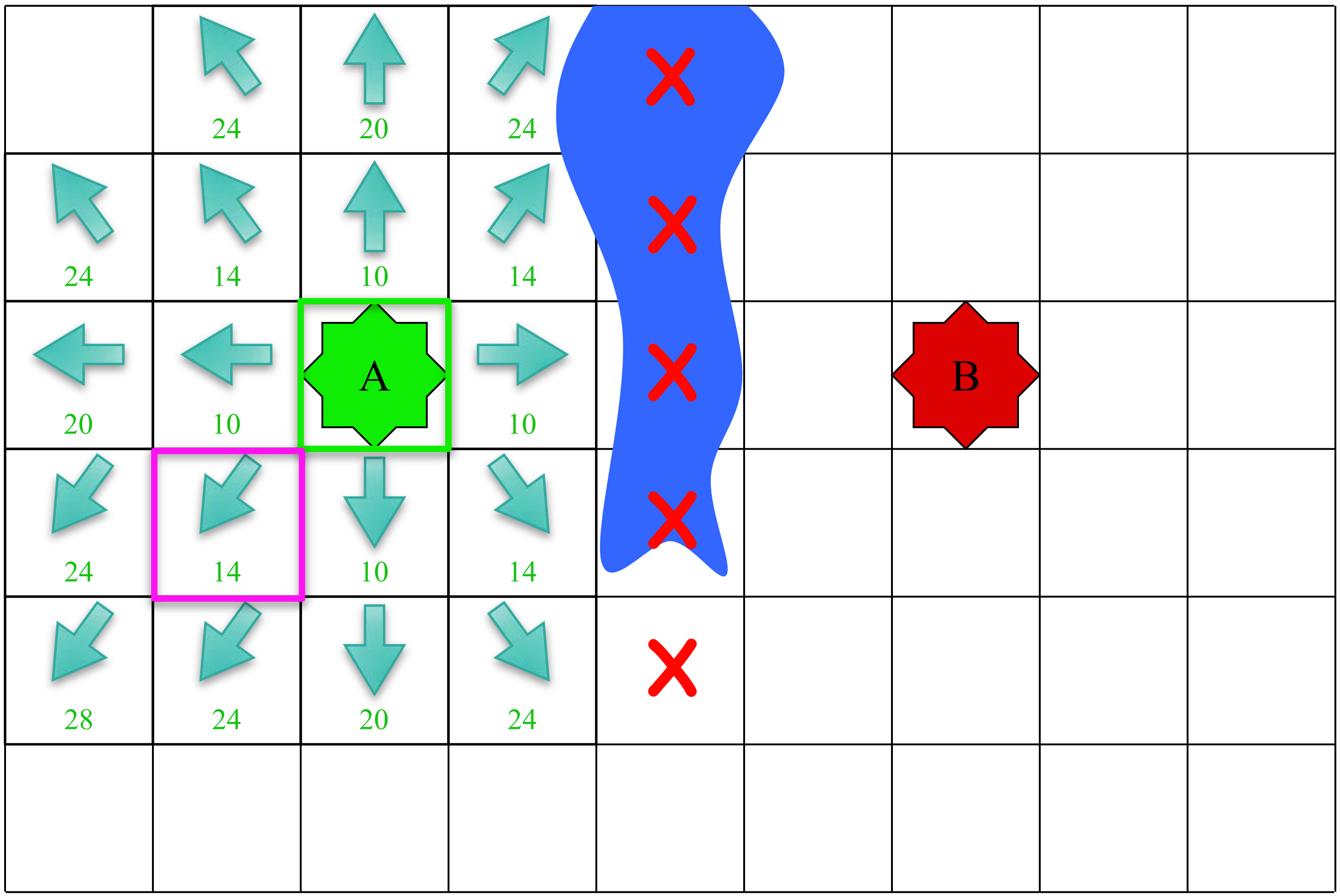
Pathfinding: Breadth-First



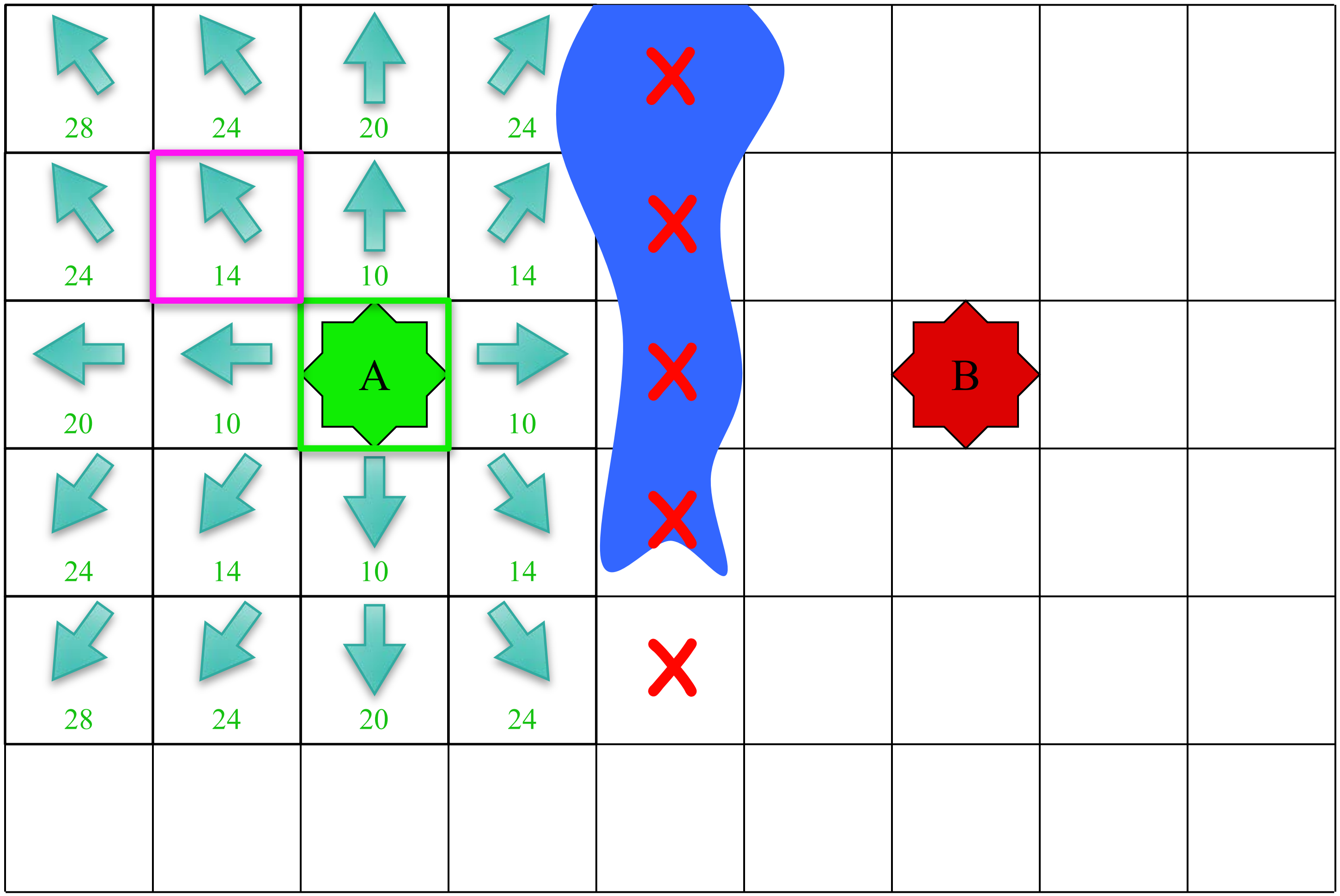
Pathfinding: Breadth-First



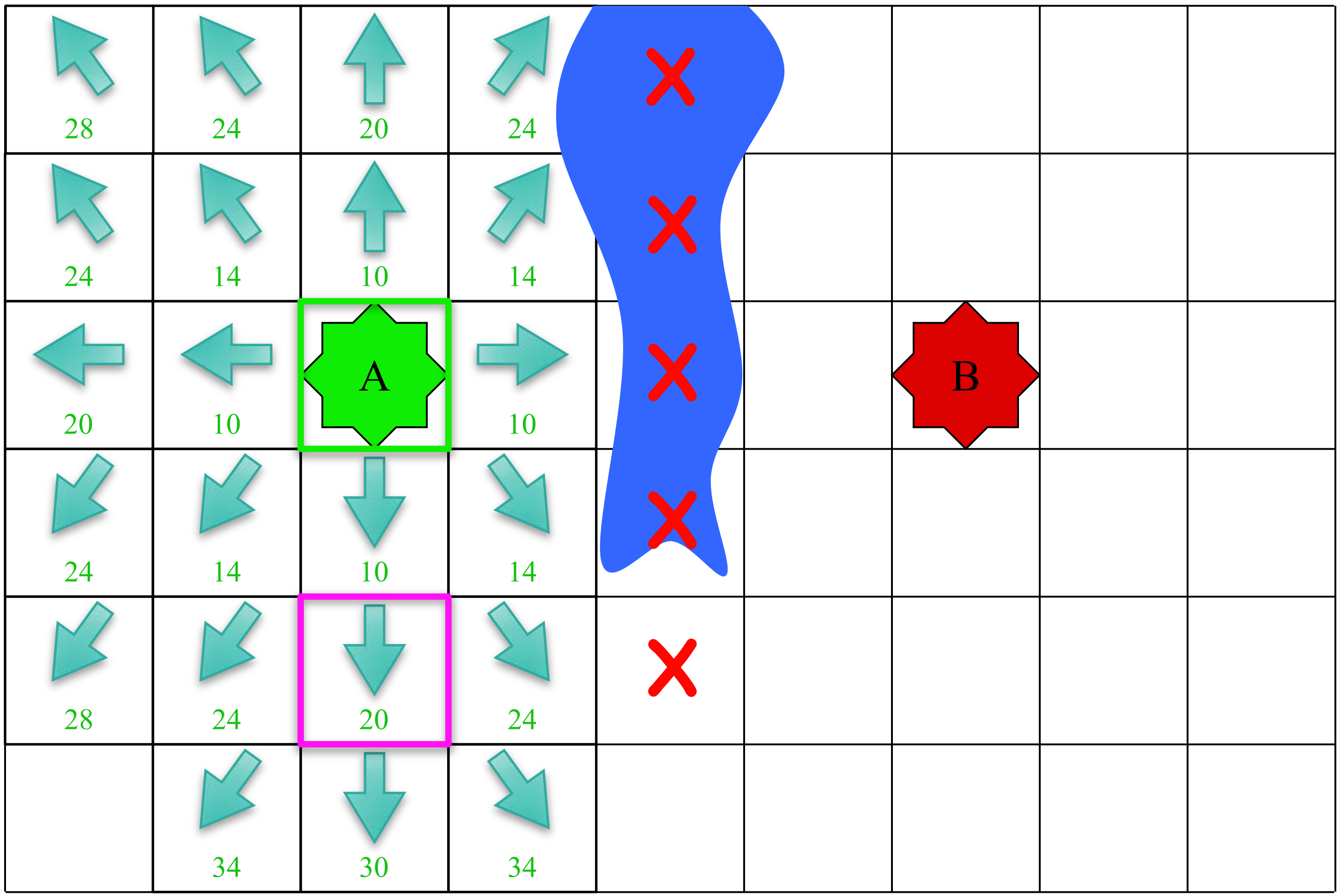
Pathfinding: Breadth-First



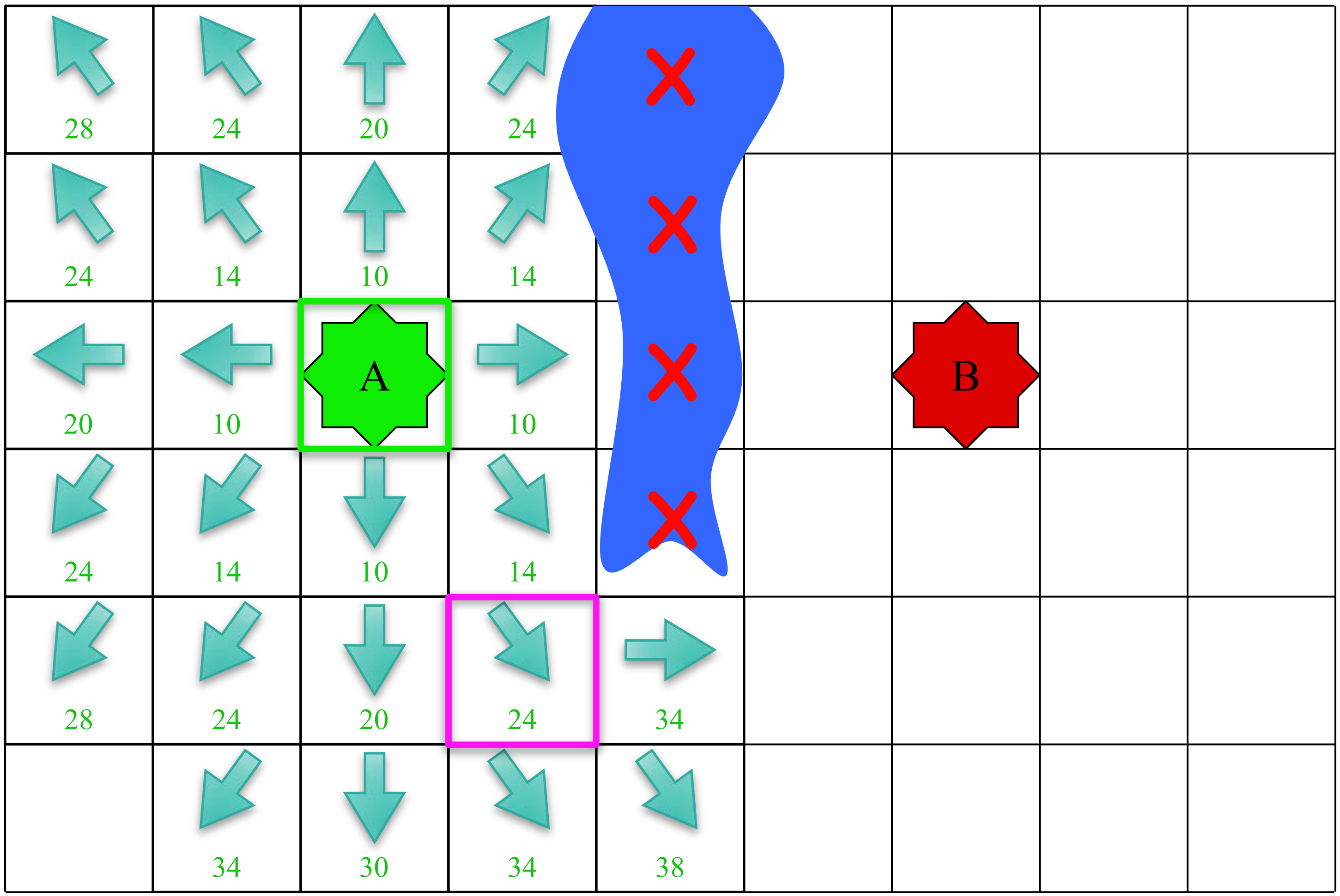
Pathfinding: Breadth-First



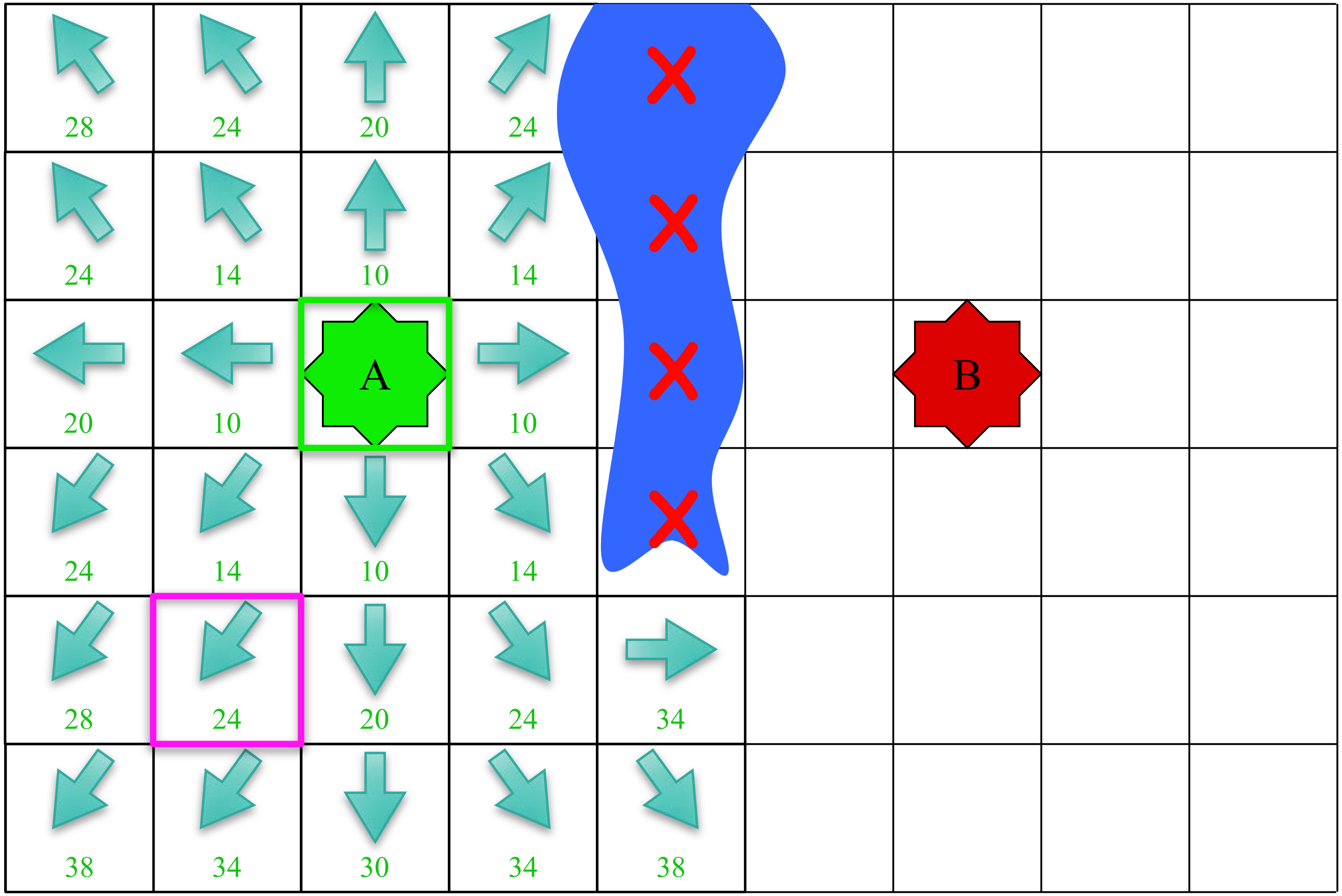
Pathfinding: Breadth-First



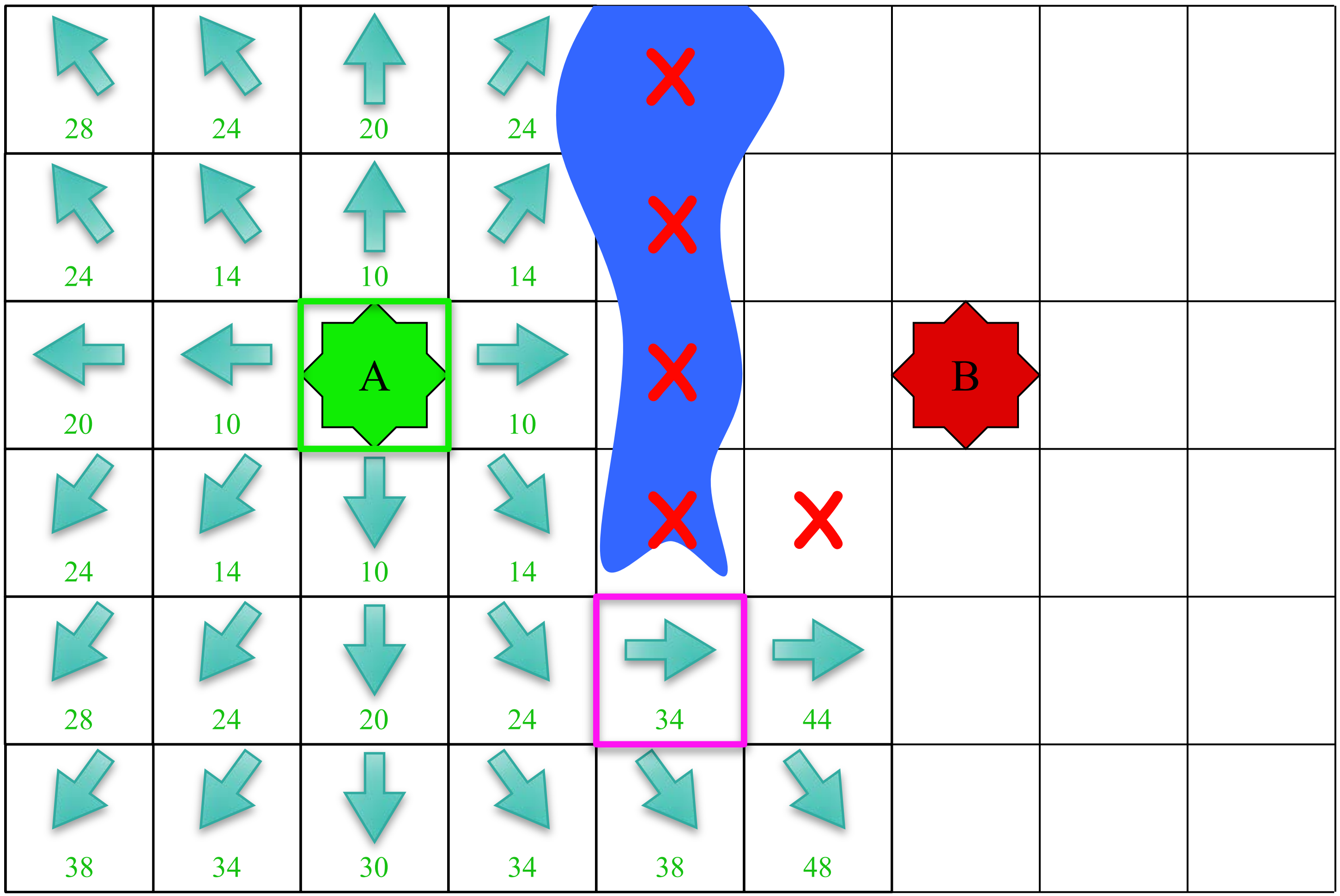
Pathfinding: Breadth-First



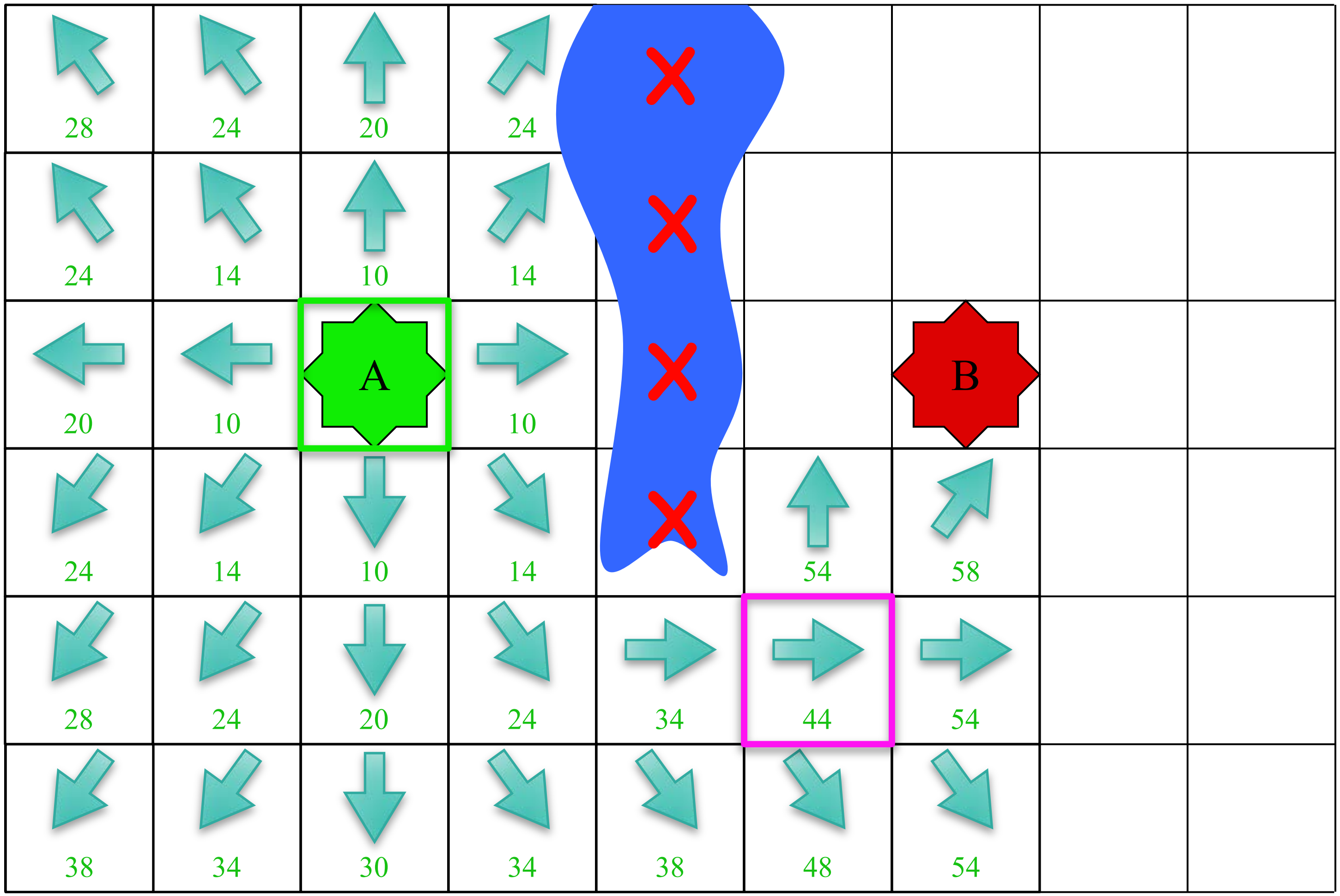
Pathfinding: Breadth-First



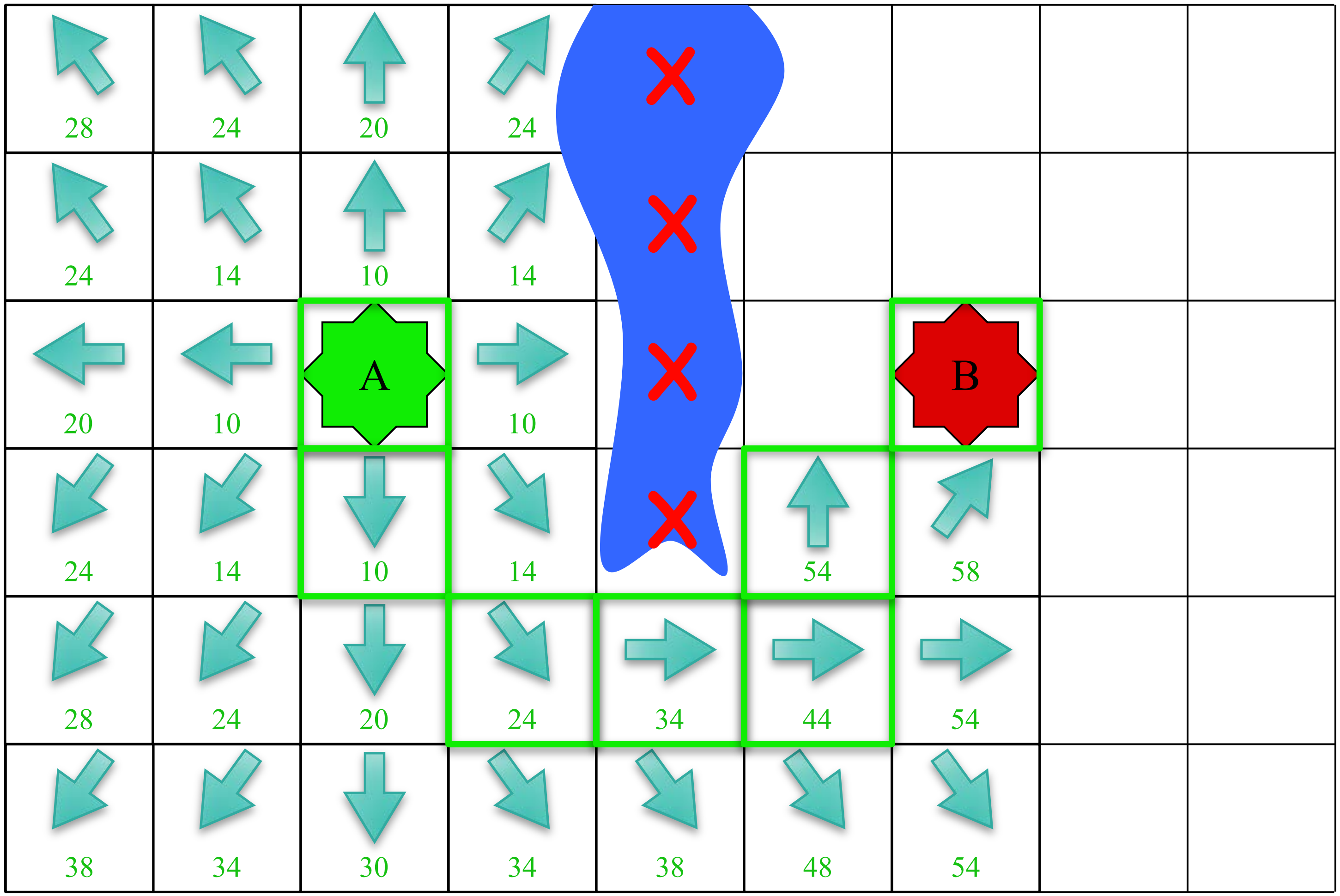
Pathfinding: Breadth-First



Pathfinding: Breadth-First

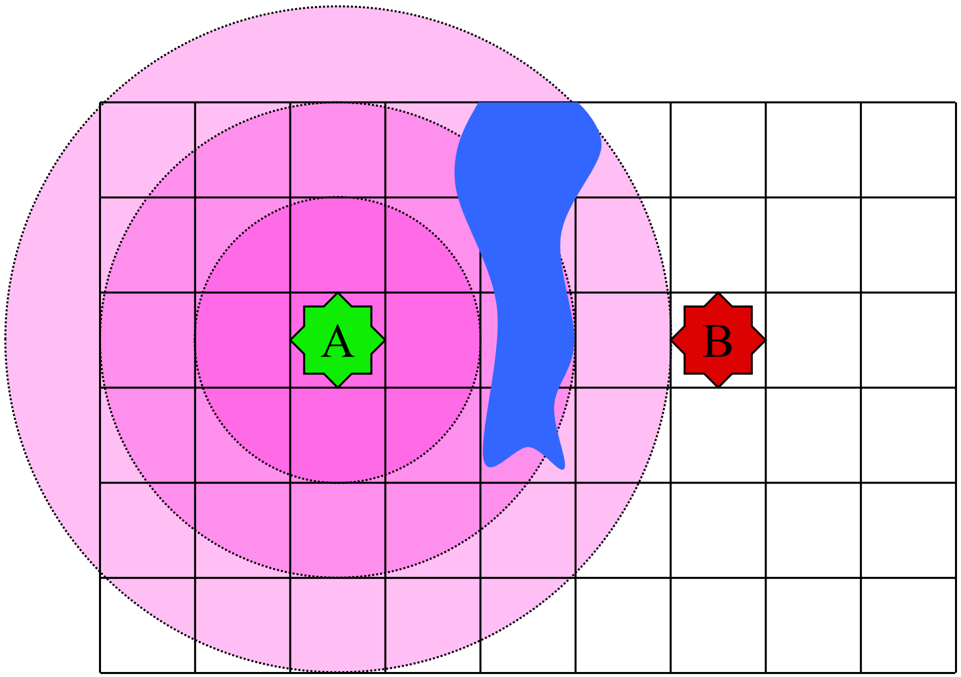


Pathfinding: Breadth-First



Breadth-First is Slow!

- Searches too many grids
  - Grids far away from goal
  - Works "radially outward"
- What is the problem?
  - Using **graph** algorithms
  - No spatial knowledge
- **Idea**: Spatial+Graph
  - Measure distance normally
  - Pick neighbor close to goal



Heuristic Search

Intuition

- Modified version of BFS
  - Have a list of candidates
  - Always pick *best* candidate
- Need *f*, **heuristic** function
  - Used to pick next step
  - Avoids stupid choices
- Regularly **update** *f*
  - Recompute on all neighbors
  - Reassign value if smaller

Algorithm

```
n = start; L = { };
while (n not goal) {
  add n to visited;
  N(n) = unvisited neighbors
  foreach (m ∈ N(n)) {
    add m to L;
    update f(m);
  }
  pick n ∈ L with f least;
}
return path to goal;
```

Heuristic Search

Intuition

- Modified version of BFS
  - Have a list of
  - Always pick
- Need *f*, **heurist**
  - Used to pick
  - Avoids stupic
- Regularly **update** *f*
  - Recompute on all neighbors
  - Reassign value if smaller

Algorithm

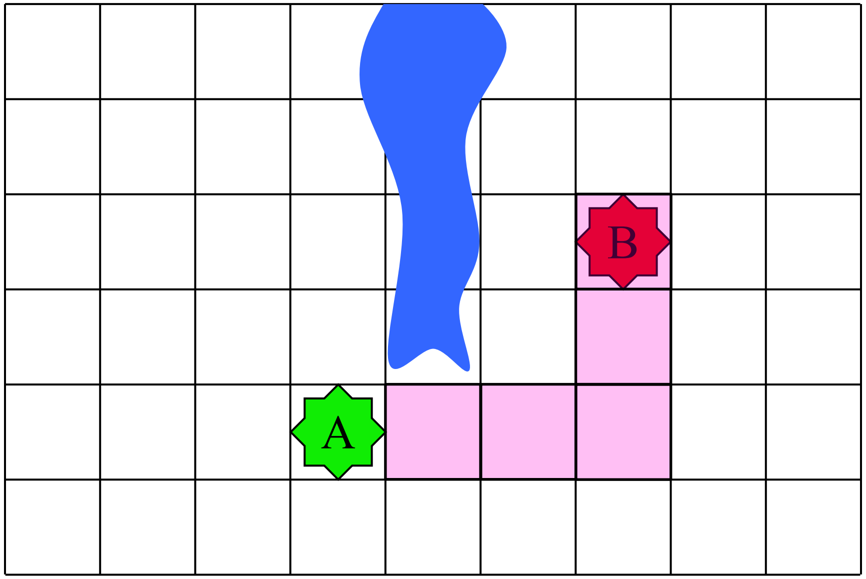
```
n = start; L = { };
           t goal) {
           ited;
           sited neighbors
           ∈ N(n)) {
           ;
           n);

  pick n ∈ L with f least;
}
return path to goal;
```

Examples:

- *Dijkstra's Algorithm*
  f = dist. from source
- *Greedy Algorithm*
  f = estimated dist. to goal

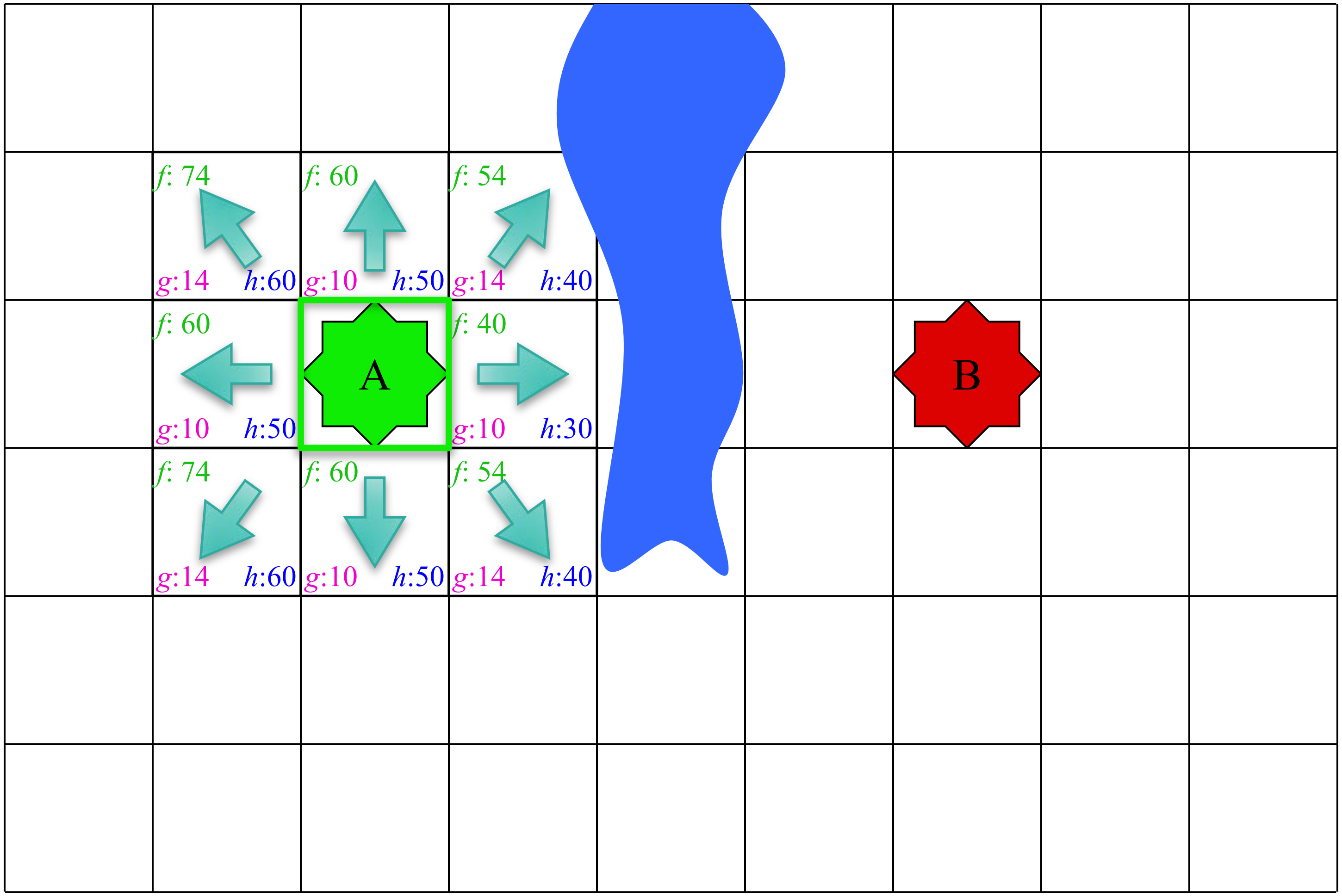
A* Algorithm

- **Idea**: Dijkstra + Greedy
  - *g*: distance on **current path**
    - An "exact calculation"
    - Distance along graph
  - *h*: estimated dist. to **goal**
    - *Spatial* distance
    - Ignores all obstacles
  - Final heuristic $f = g+h$
- Many variations for *h*
  - Regular distance
  - "Manhattan Metric"

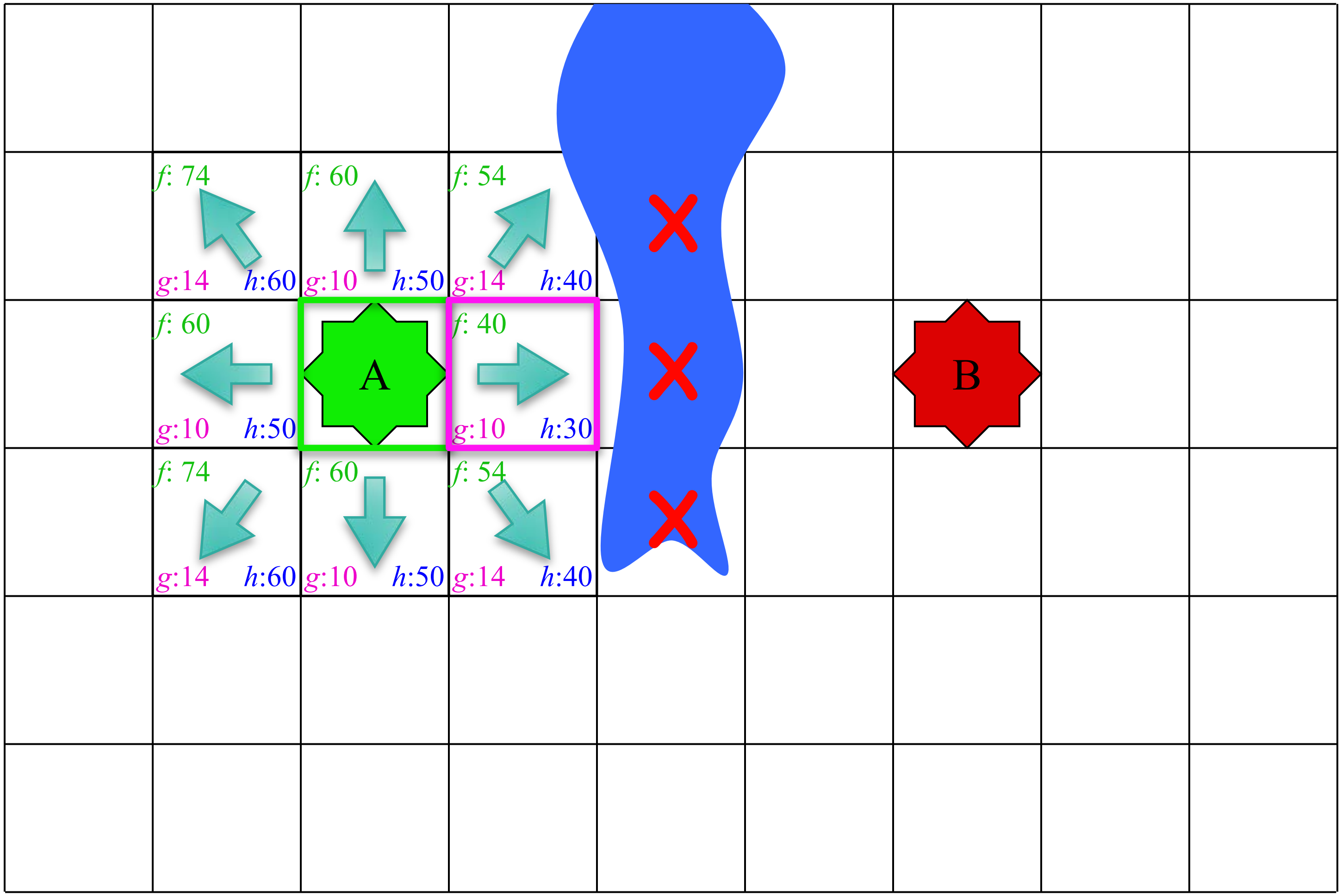


Manhattan distance = 30+20 = 50

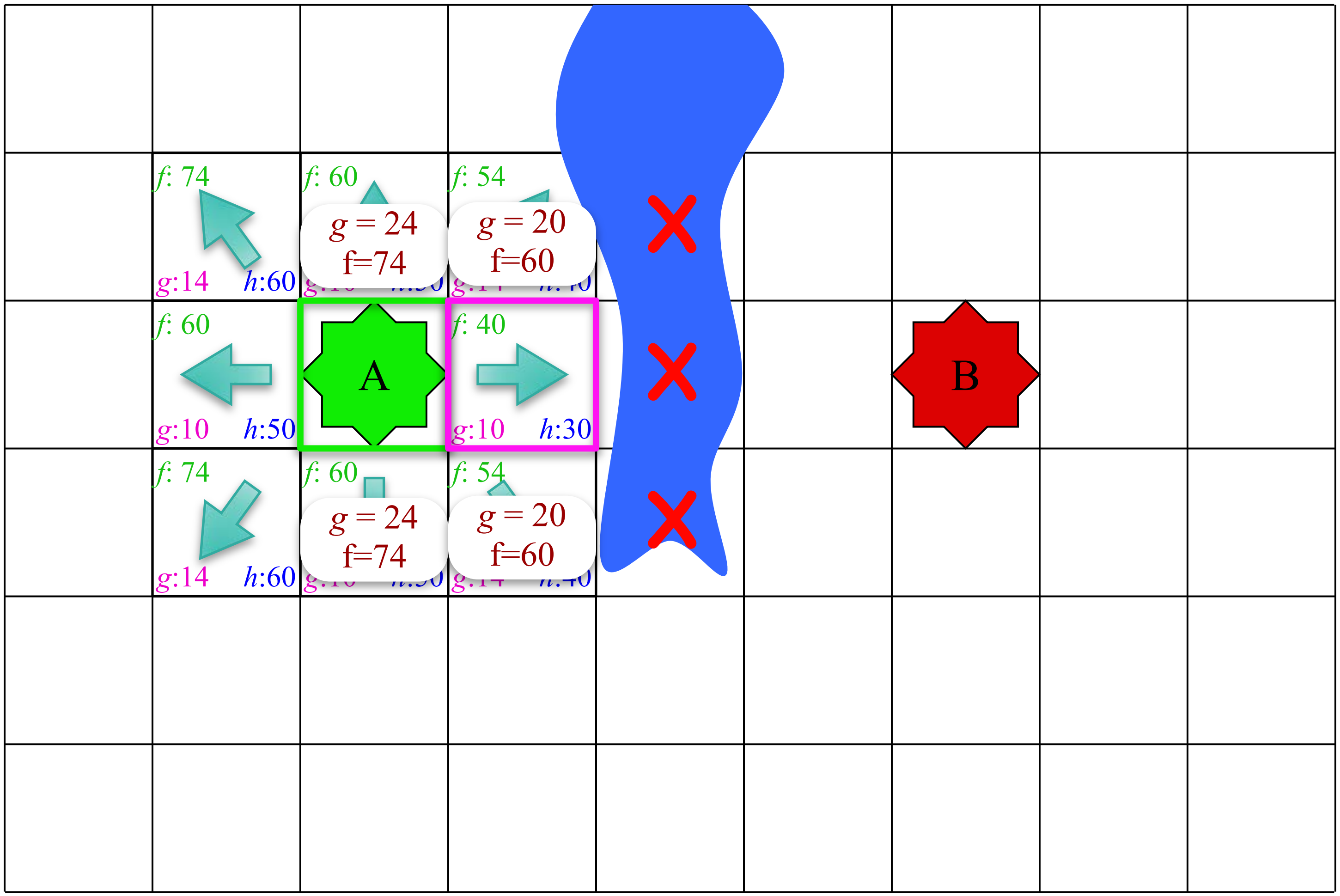
Pathfinding: A* Algorithm



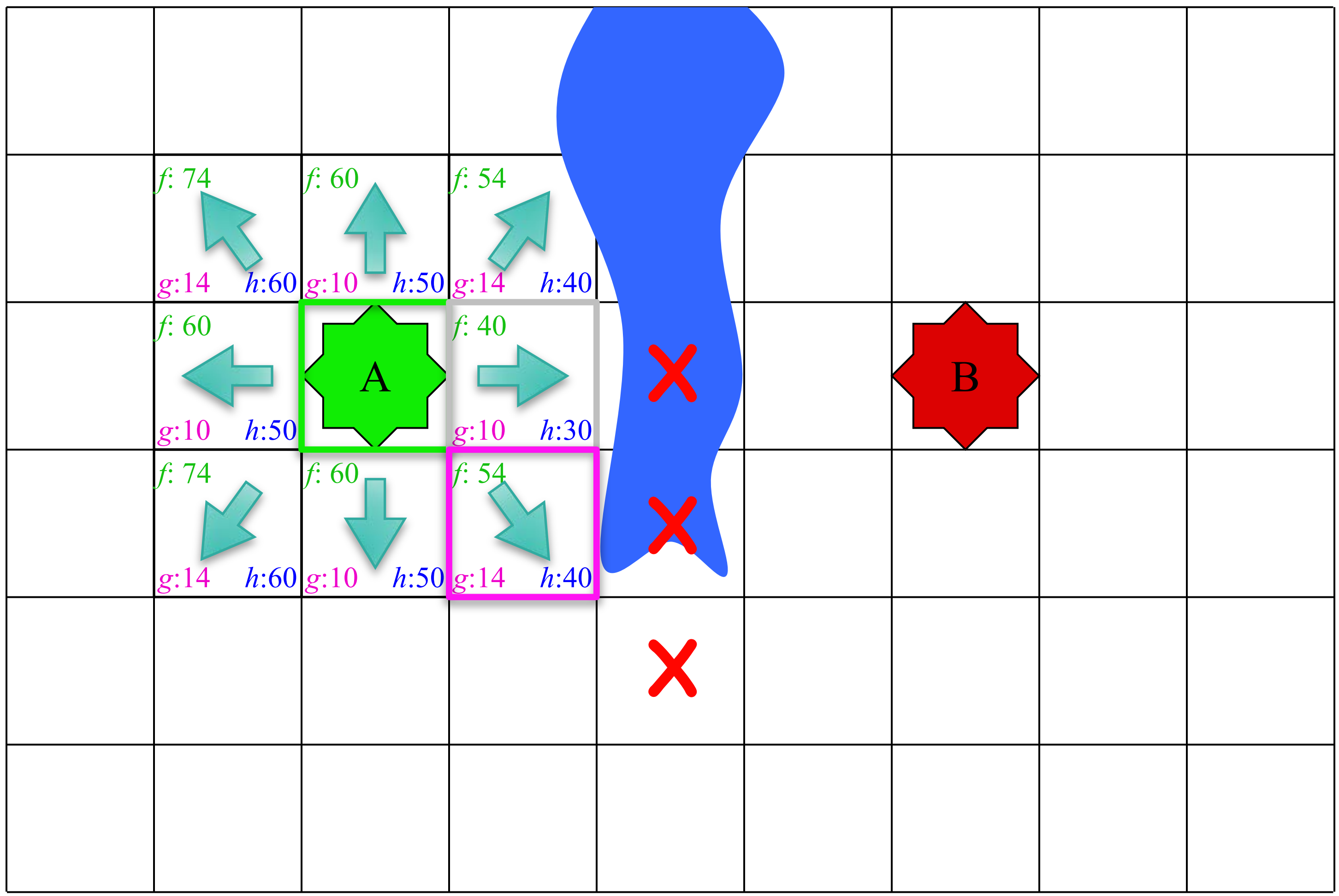
Pathfinding: A* Algorithm



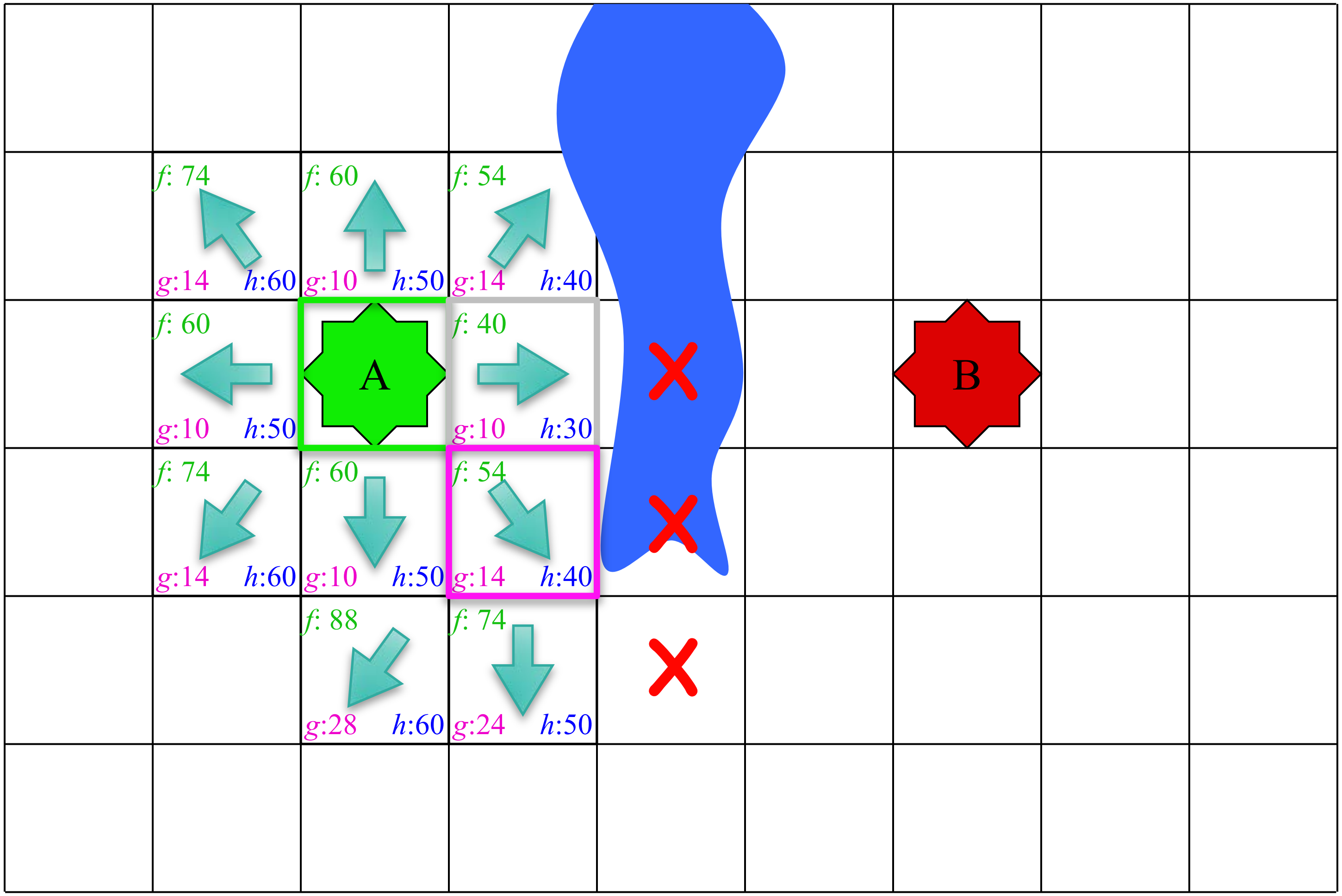
Pathfinding: A* Algorithm



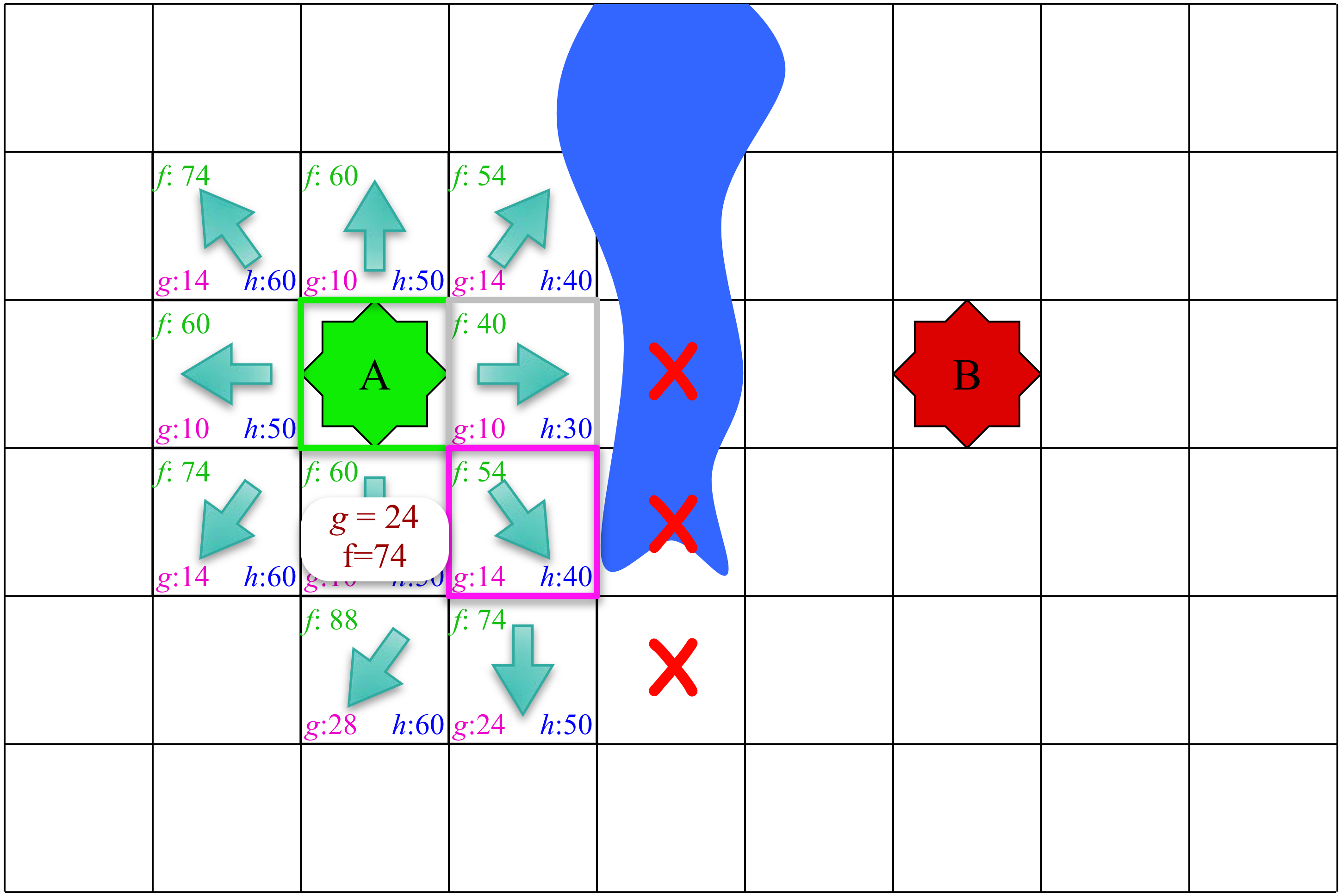
Pathfinding: A* Algorithm



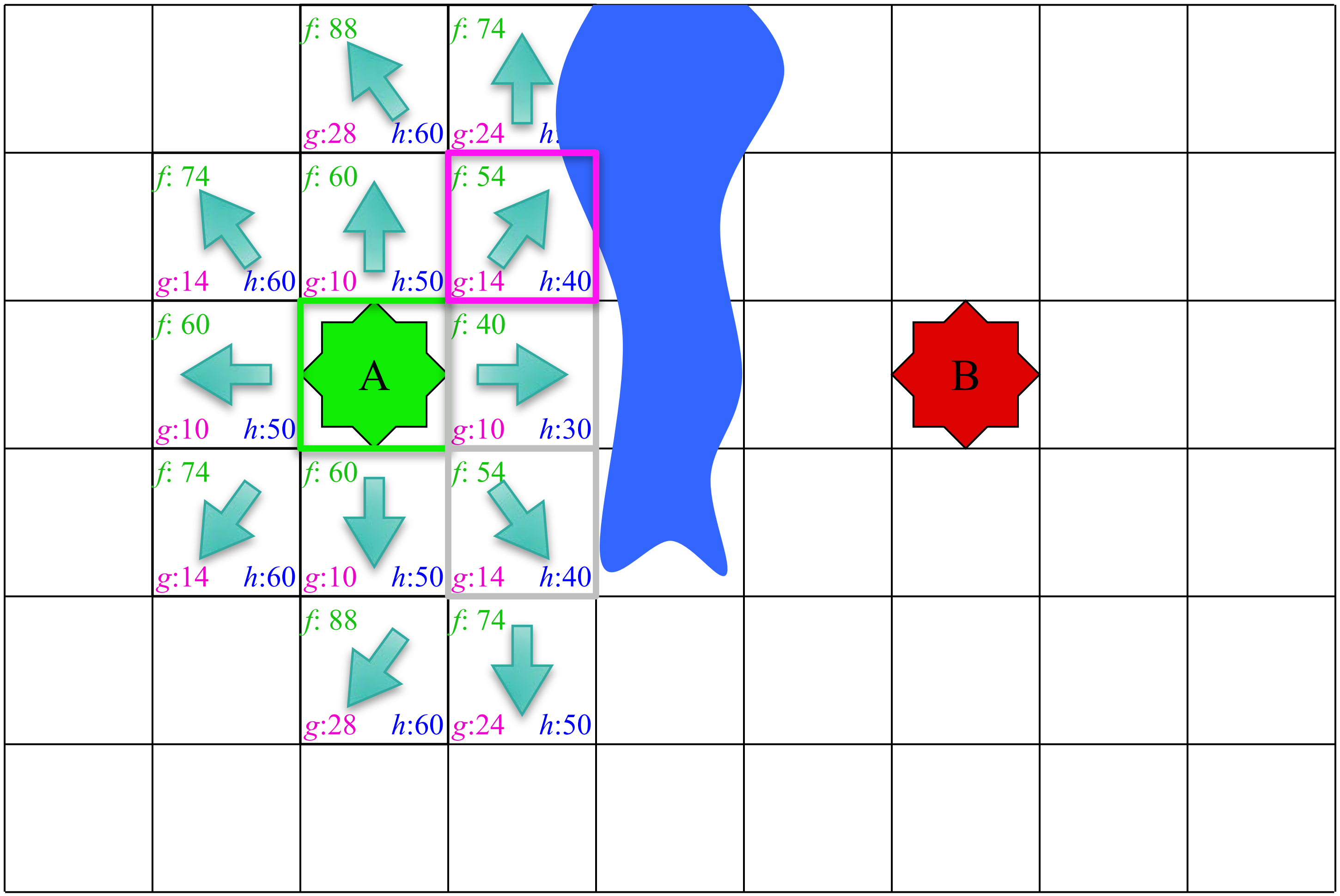
Pathfinding: A* Algorithm



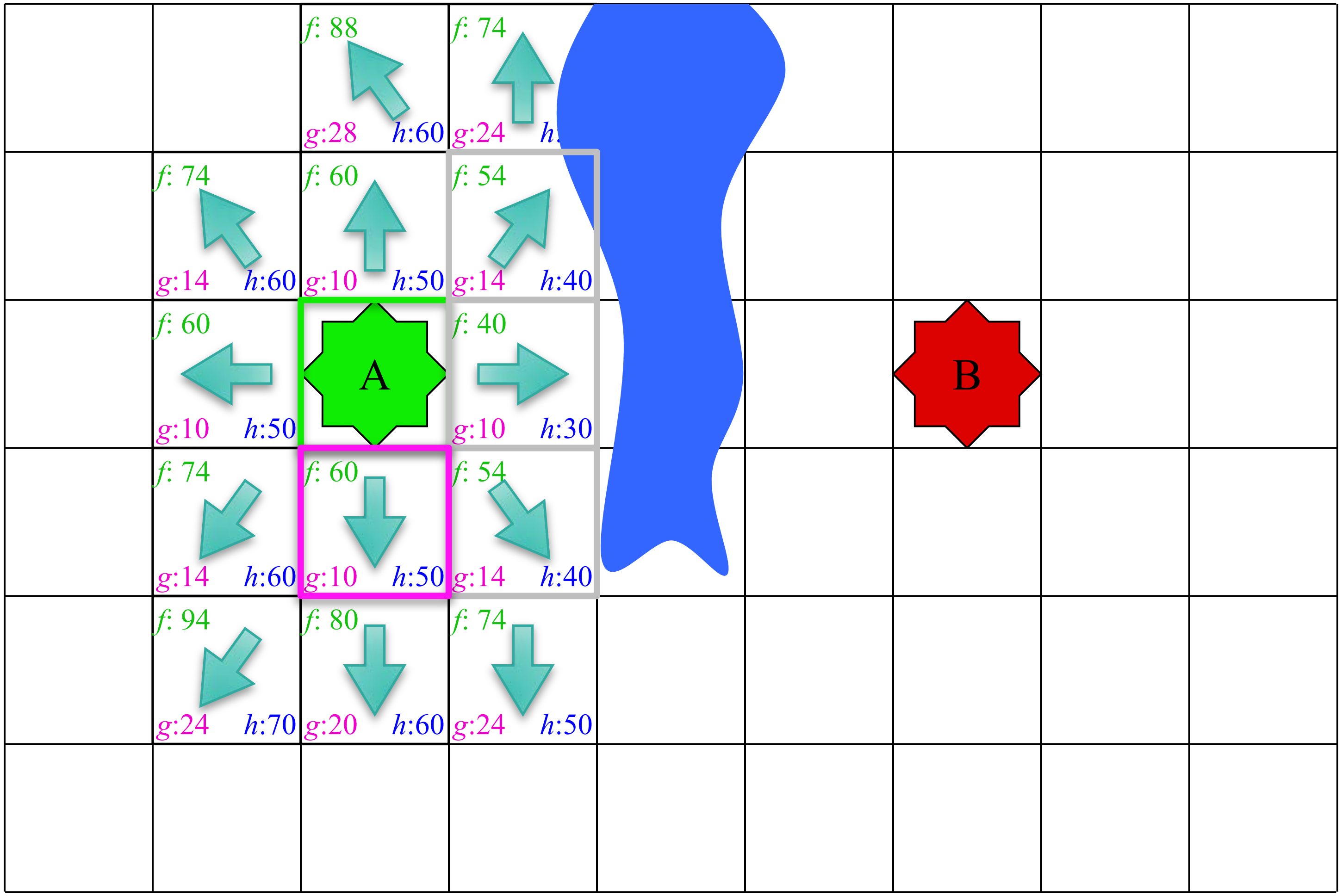
Pathfinding: A* Algorithm



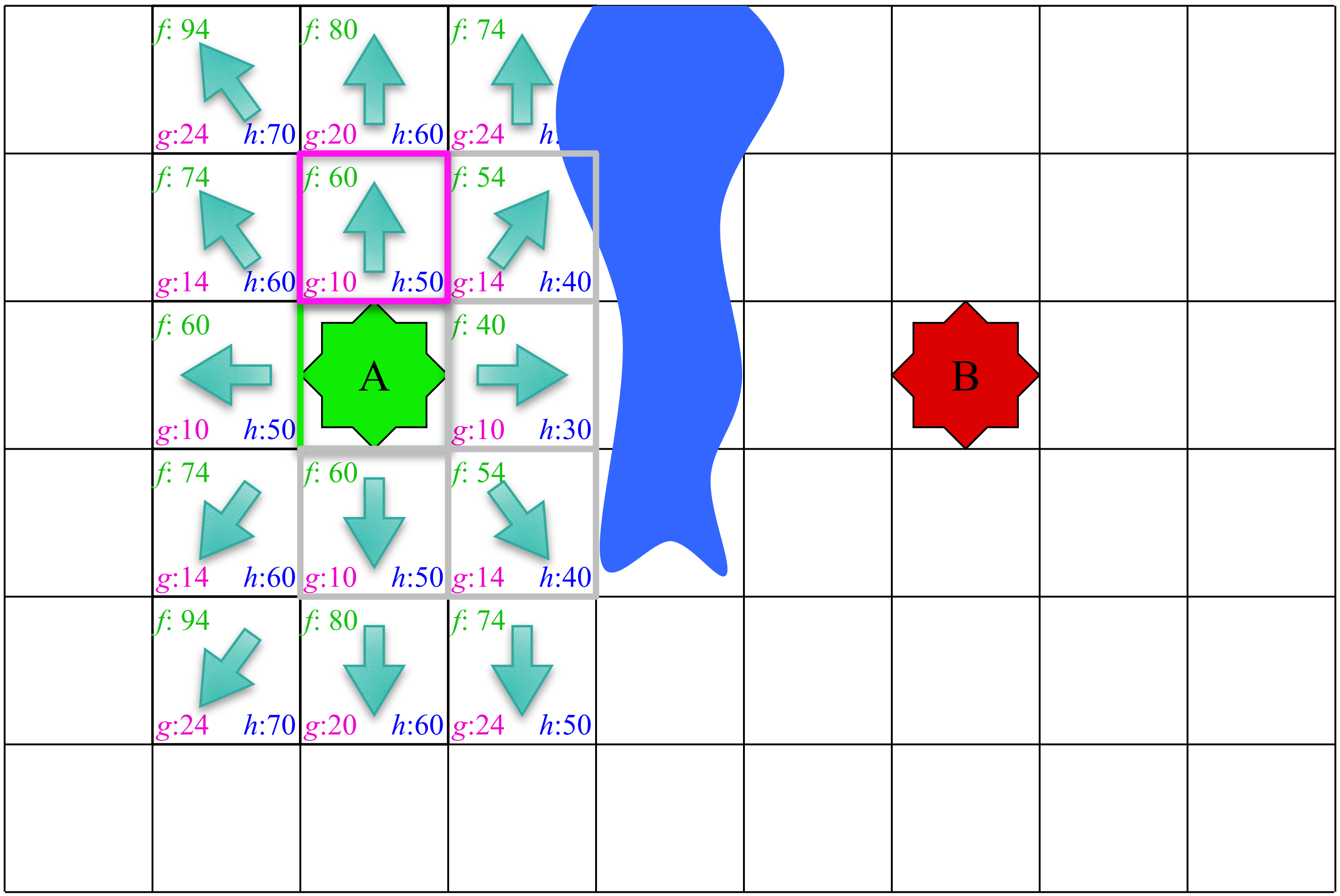
Pathfinding: A* Algorithm



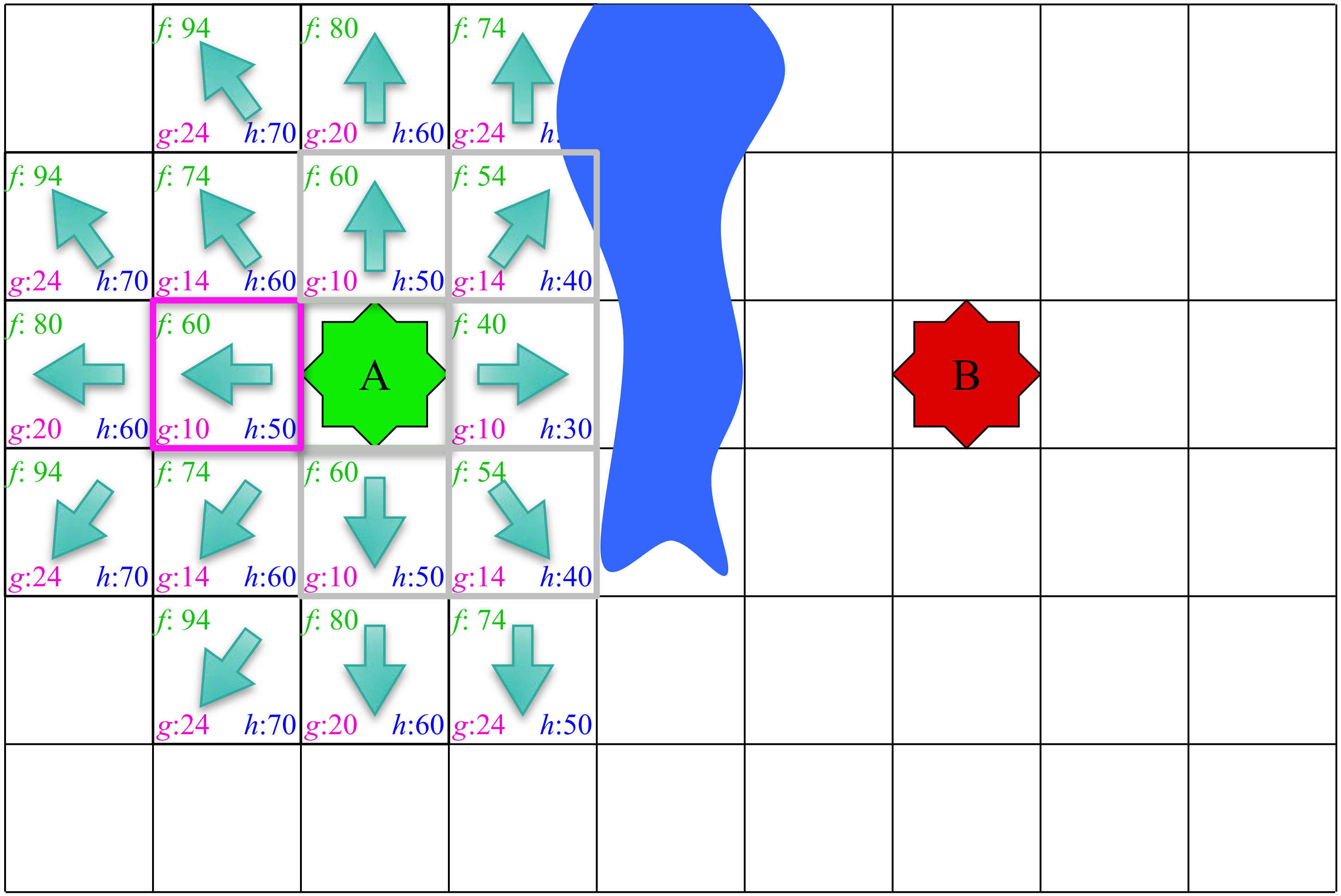
Pathfinding: A* Algorithm



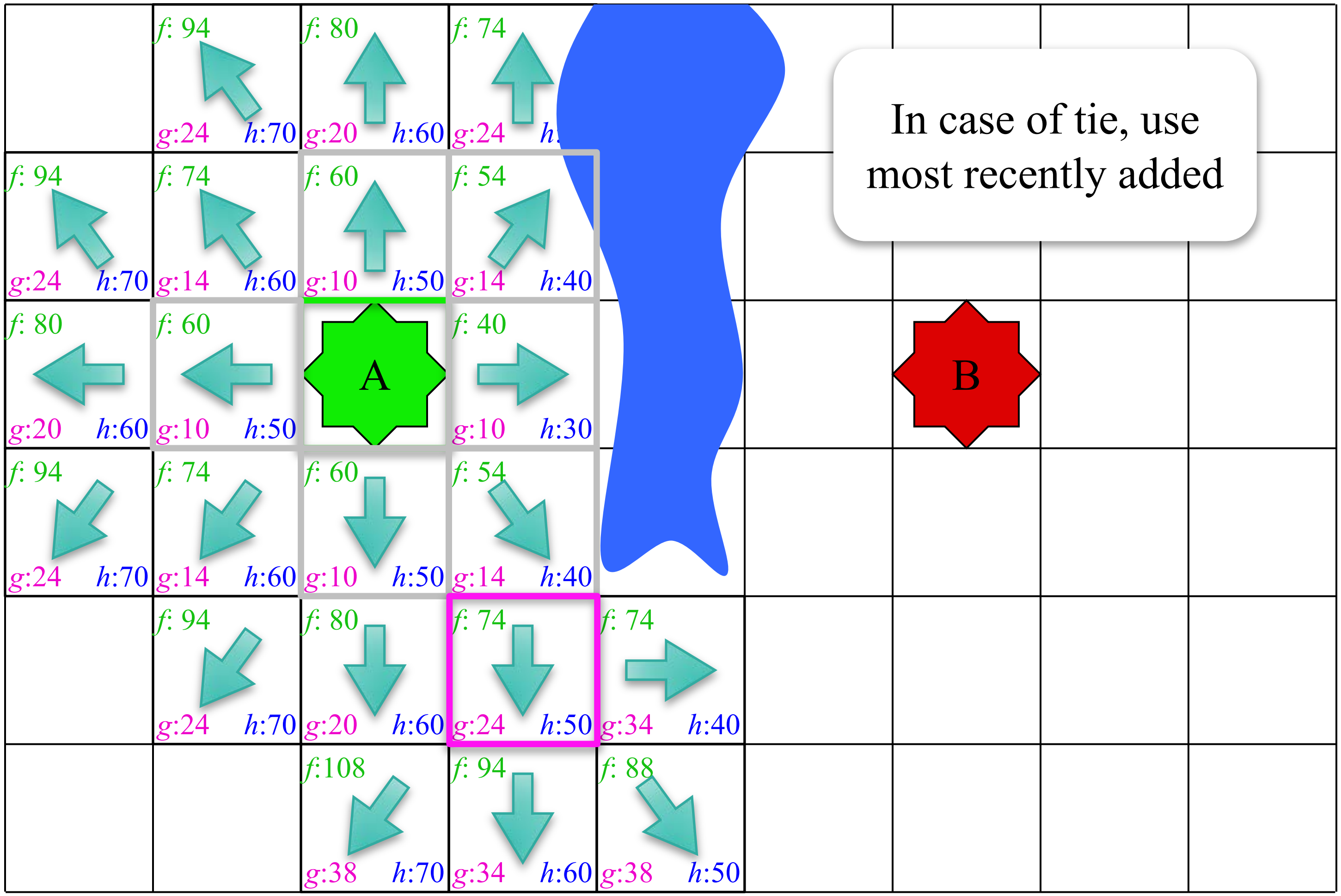
Pathfinding: A* Algorithm



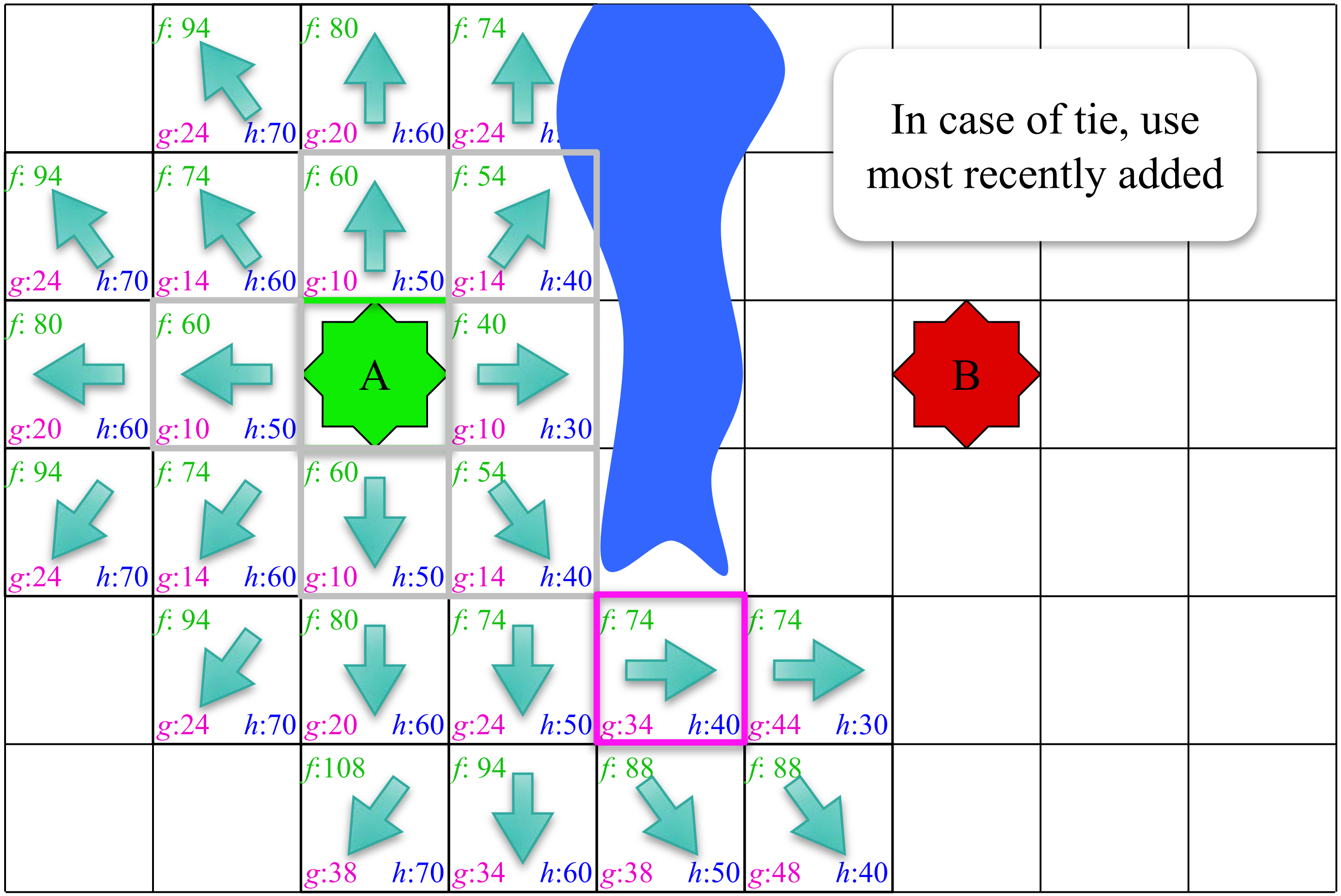
Pathfinding: A* Algorithm



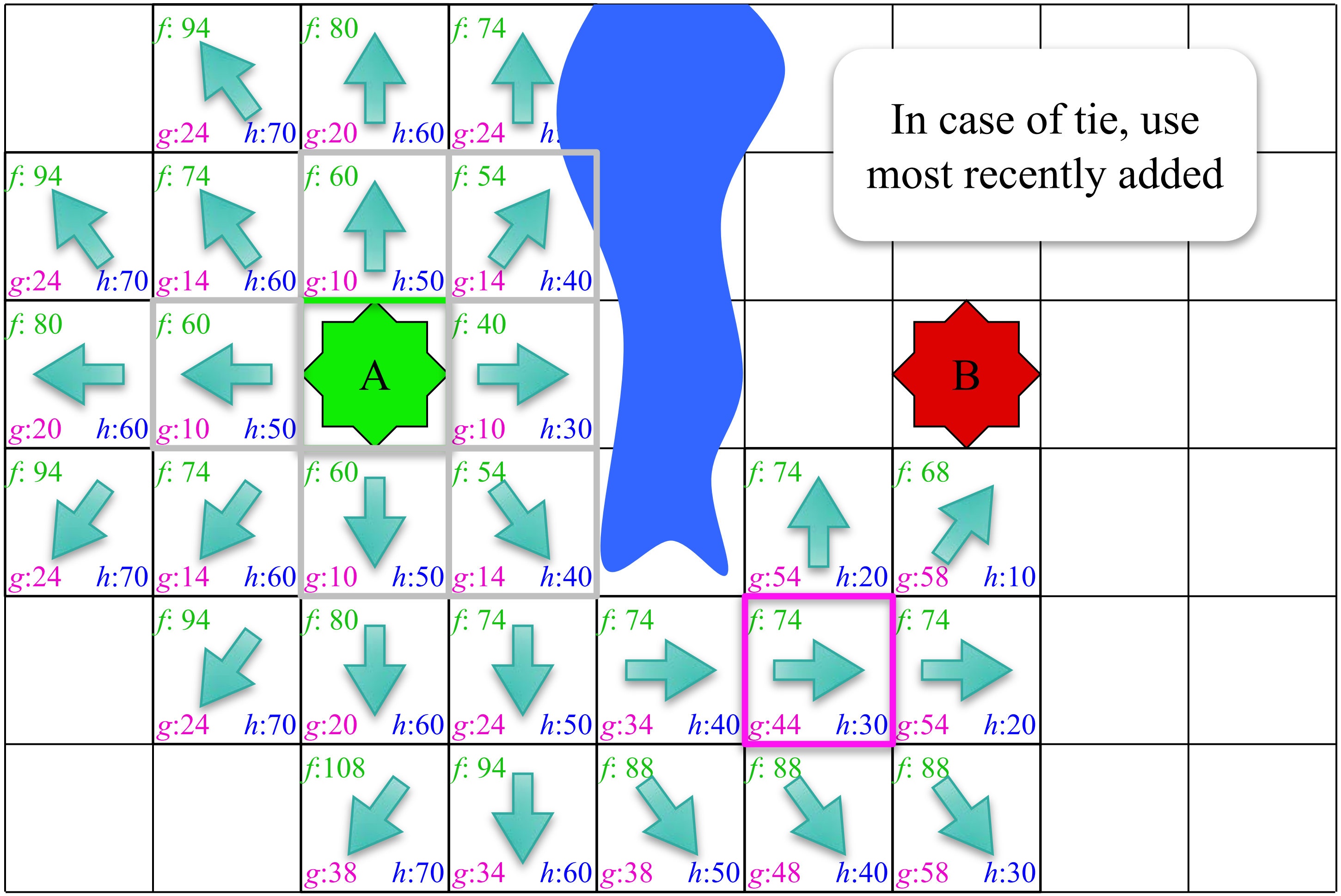
Pathfinding: A* Algorithm



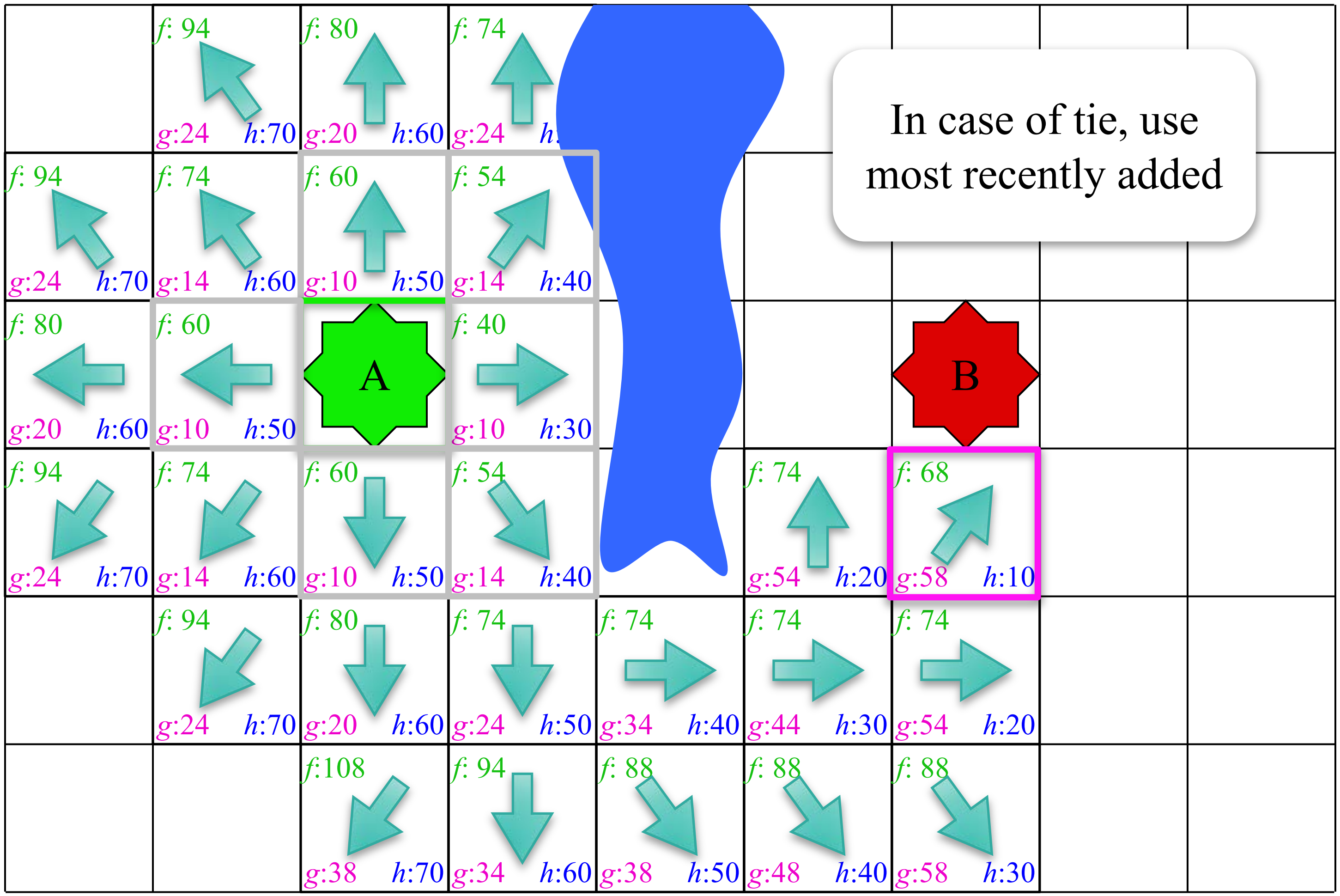
Pathfinding: A* Algorithm



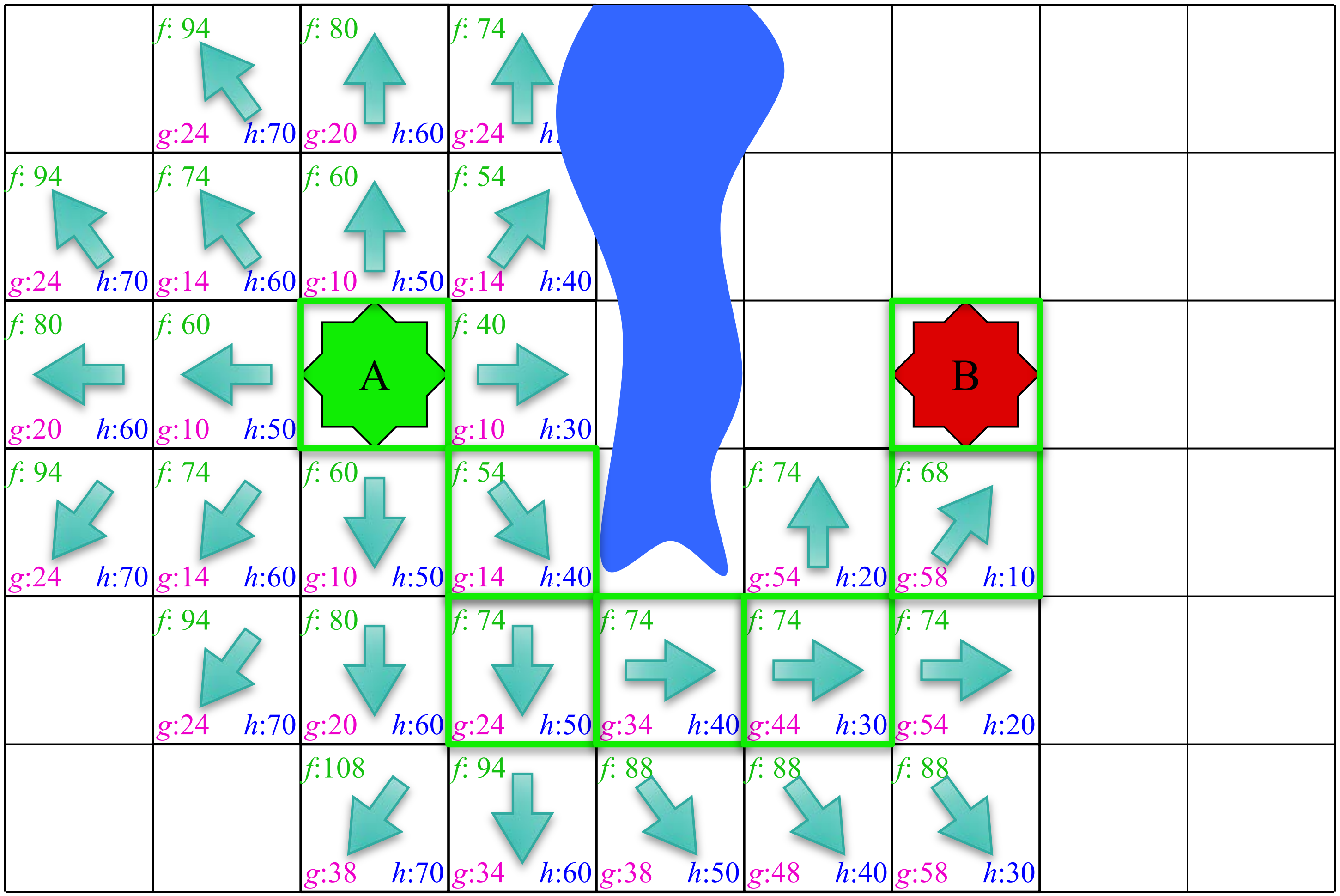
Pathfinding: A* Algorithm



Pathfinding: A* Algorithm



Pathfinding: A* Algorithm



LibGDX Support

IndexedGraph

- Array of `IndexedNode` objs
  - Can implement as an array
  - Hard part is IndexedNode
- Each `IndexNode` must store
  - Index into the graph array
  - Array of `Connection` objs
- Each `Connection` must have
  - The start and end node
  - The cost to traverse edge

IndexedAStarPathFinder

- Construct with a graph
  - Must use with `IndexedGraph`
  - Graph reference immutable
- To search for path, give
  - The start and end nodes
  - `Heuristic` implementation
  - `GraphPath` for the answer
- Can give search a *timeout*
  - Abort if it takes too long

LibGDX Support

IndexedGraph

- Array of IndexedNode objs
  - Can implement as an array
  - Hard part is Ind
- Each IndexNode r
  - Index into the g
  - Array of Connection objs
- Each Connection must have
  - The start and end node
  - The cost to traverse edge

IndexedAStarPathFinder

- Construct with a graph
  - Must use with IndexedGraph
  - h reference immutable
- ch for path, give
  - start and end nodes
  - Heuristic implementation
  - GraphPath for the answer
- Can give search a *timeout*
  - Abort if it takes too long

Everything in blue is an interface

LibGDX Support

IndexedGraph

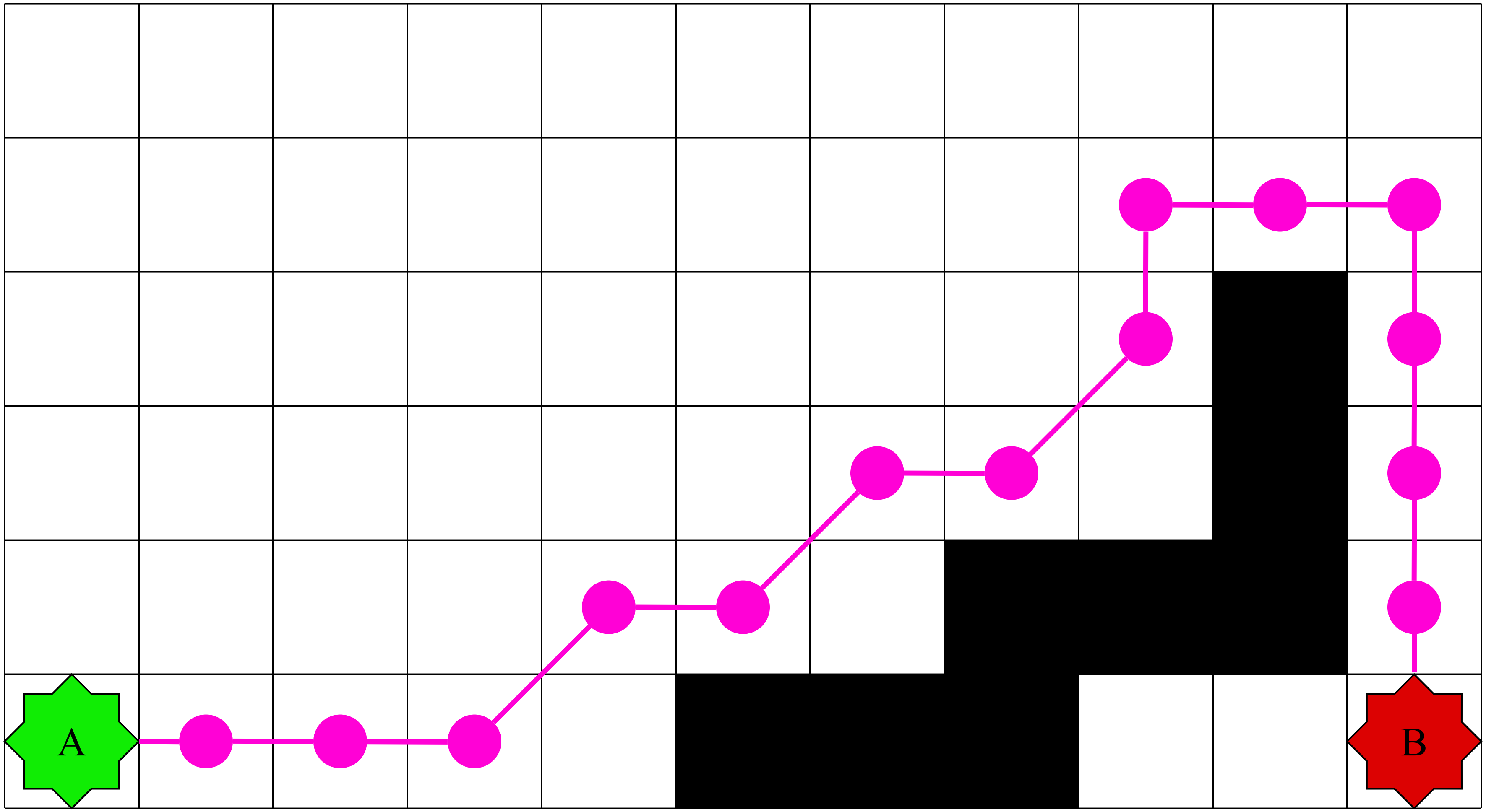
- Array of IndexedNode objs
  - Can implement as an array
  - Hard part is
- Each IndexNo
  - Index into t
  - Array of Connection objs
- Each Connection must have
  - The start and end node
  - The cost to traverse edge

IndexedAStarPathFinder

- Construct with a graph
  - Must use with IndexedGraph
  - h reference immutable
- ch for path, give
  - start and end nodes
  - Heuristic implementation
  - GraphPath for the answer
- Can give search a *timeout*
  - Abort if it takes too long

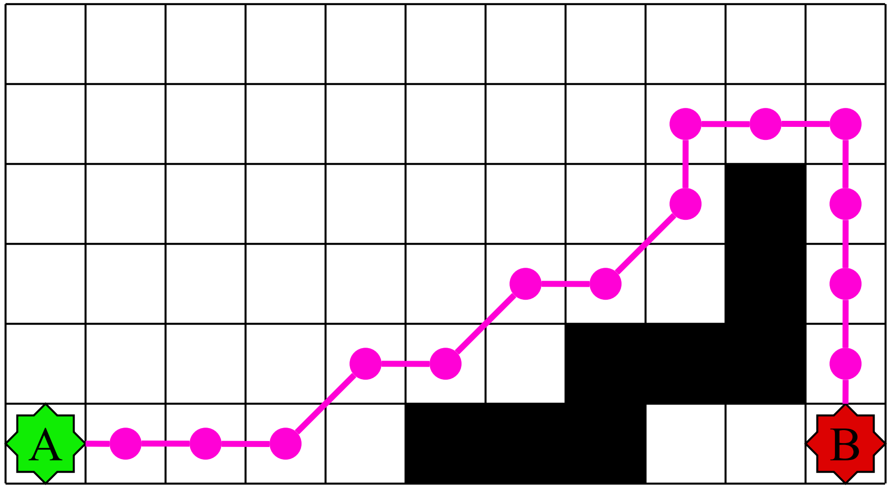
Only these have implementations

Issues with A*: Stair Stepping



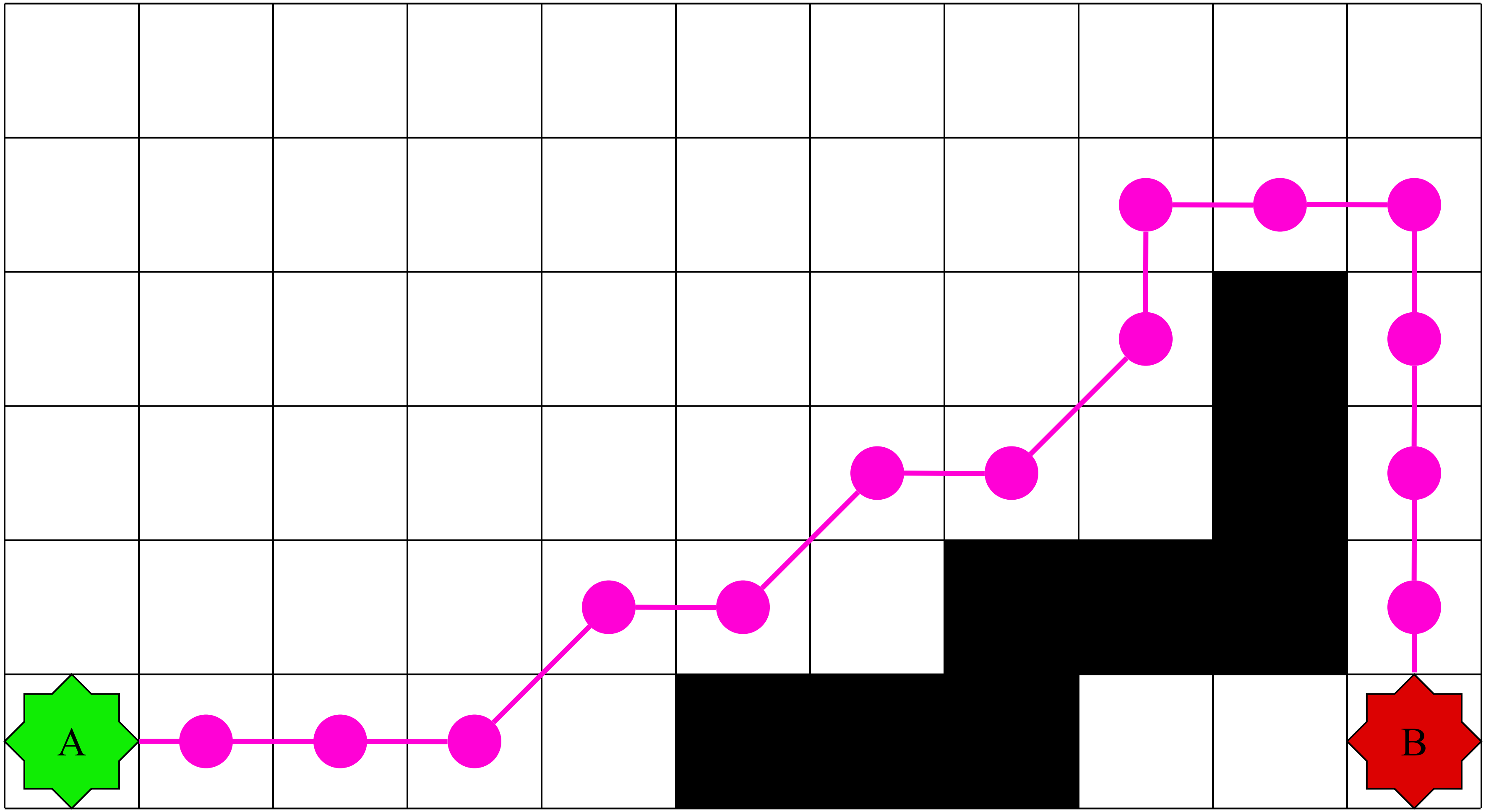
Stair Stepping

- What is the problem?
  - Move one square at a time
  - All turns are at 45°
- **Idea**: Path smoothing
  - Path is a series of waypoints
  - Straight line between points
  - Remove unnecessary points
- Can combine with A*
  - Get *degenerative* solution
  - Remove to get waypoints

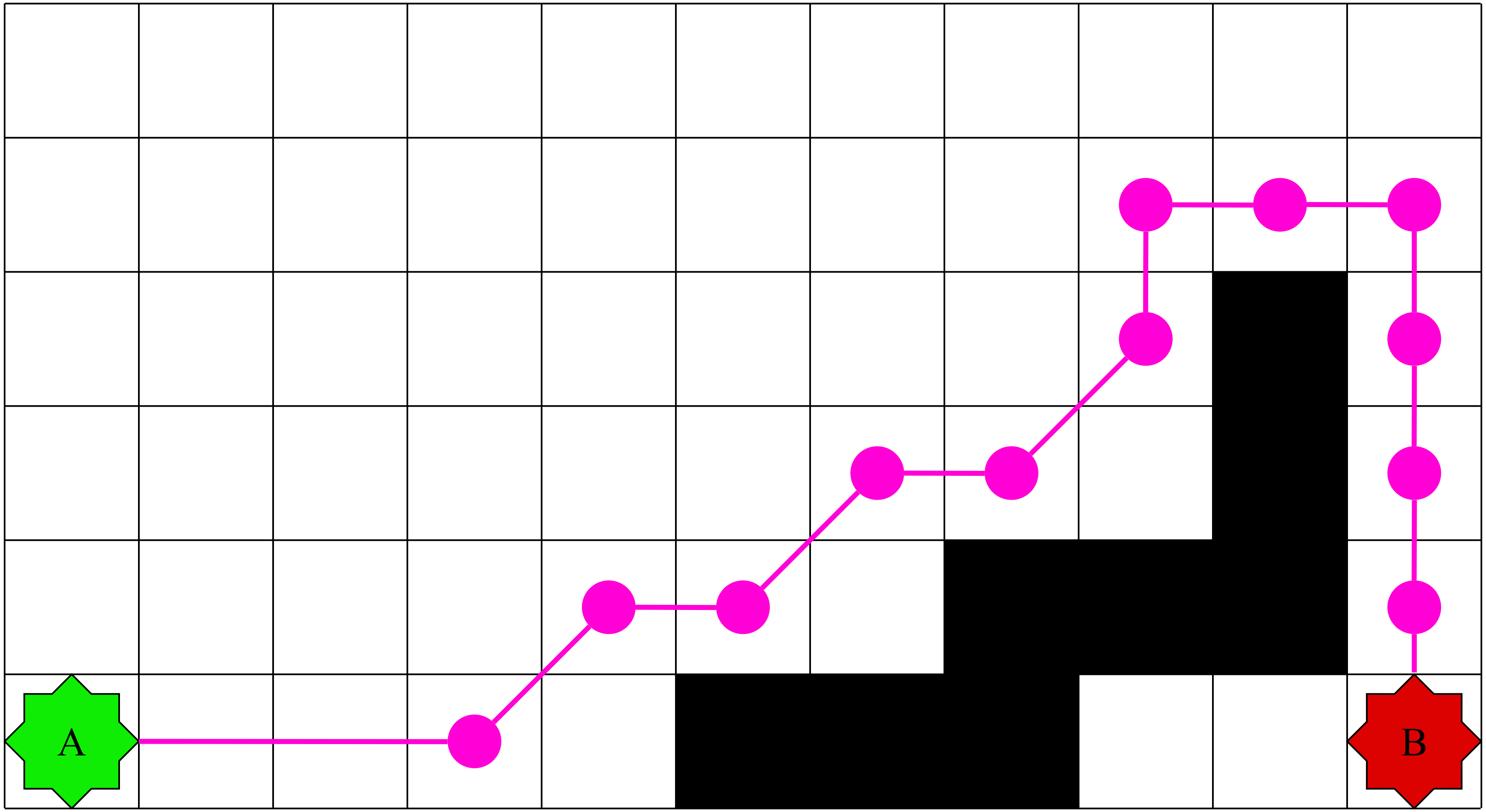


- Choose first **q** after **p** where
  - Line **pq** is valid
  - Point **q** has successor **s**
  - Line **ps** is not valid

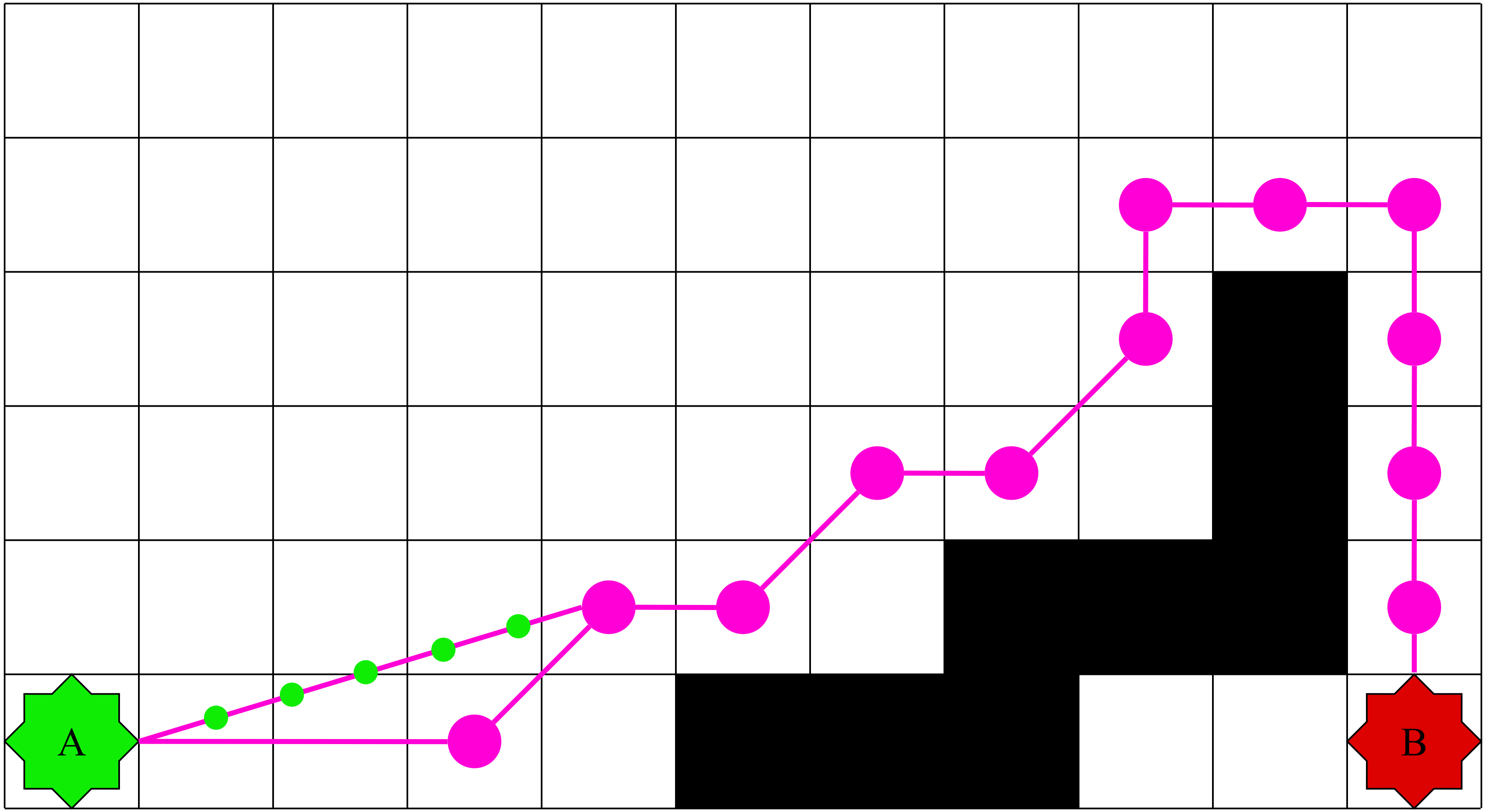
Path Smoothing



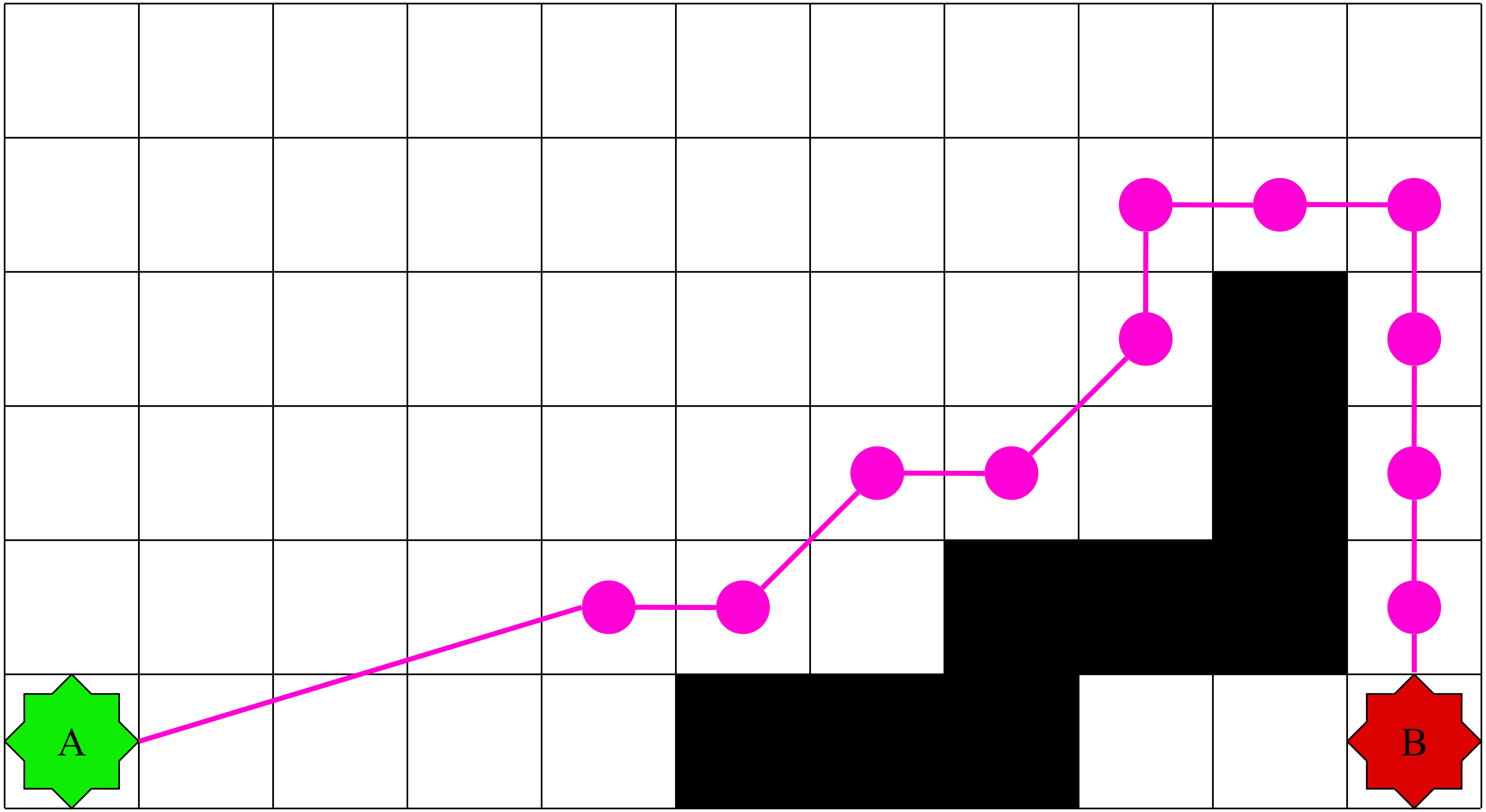
Path Smoothing



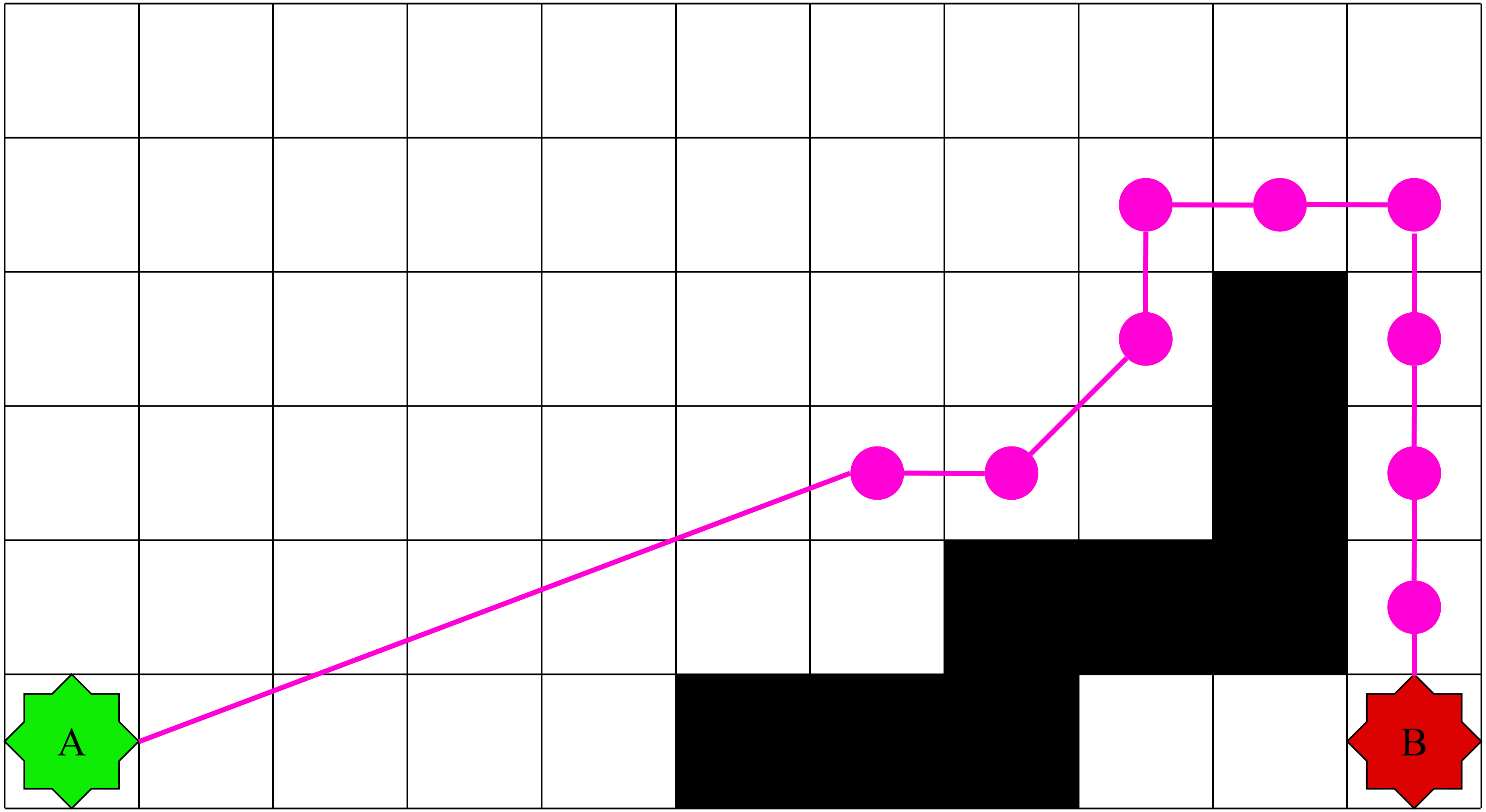
Path Smoothing



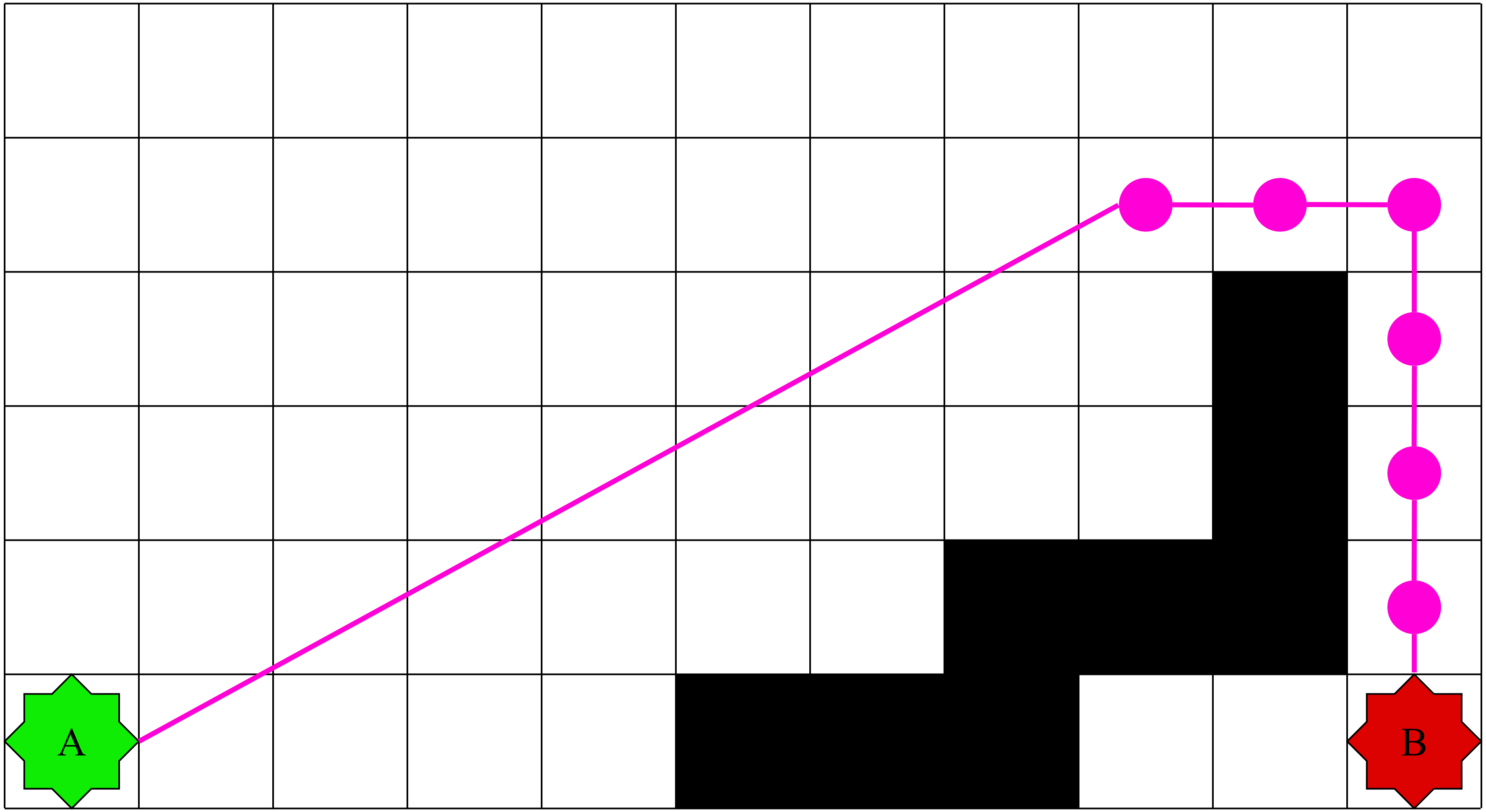
Path Smoothing



Path Smoothing

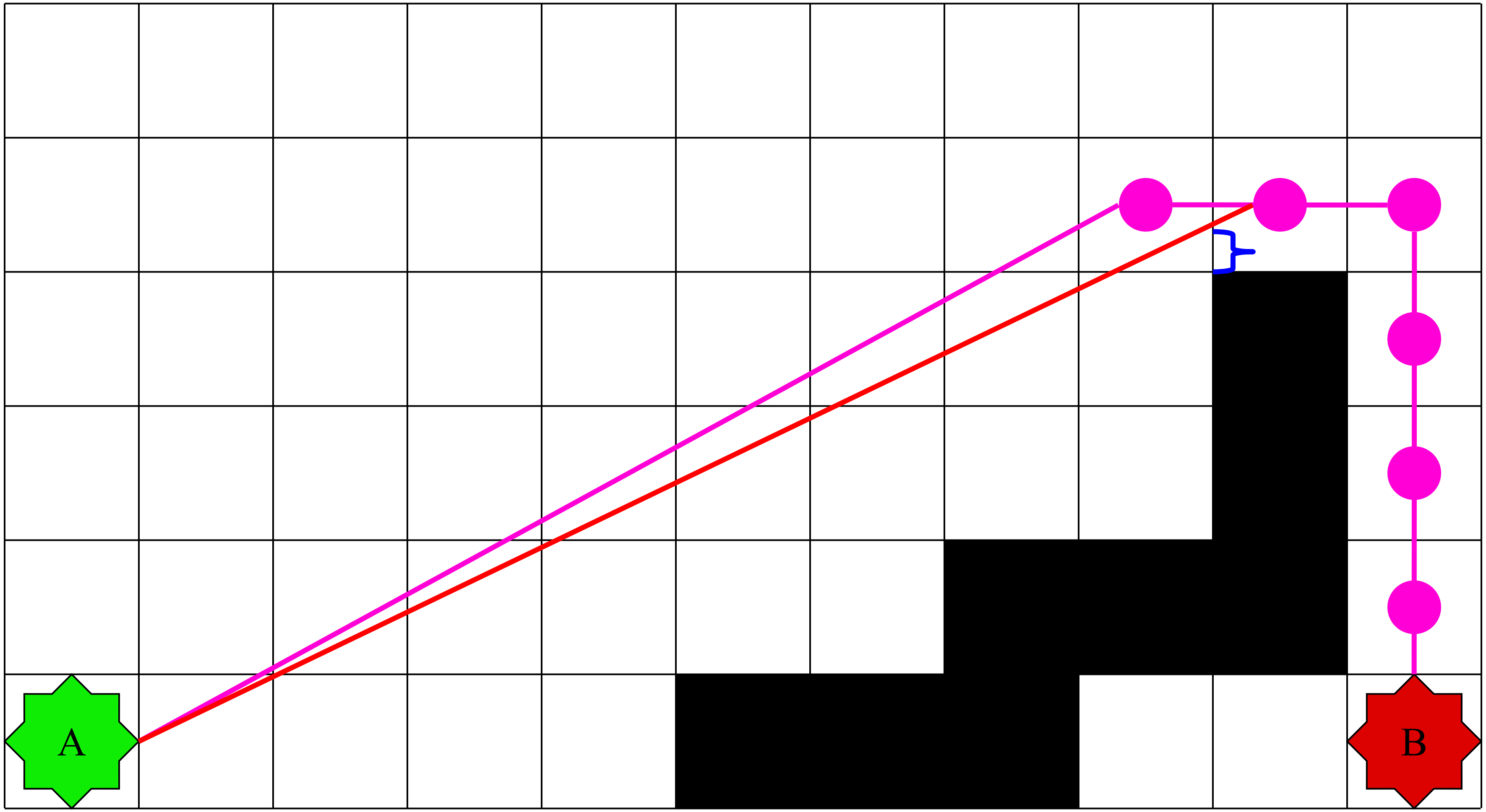


Path Smoothing

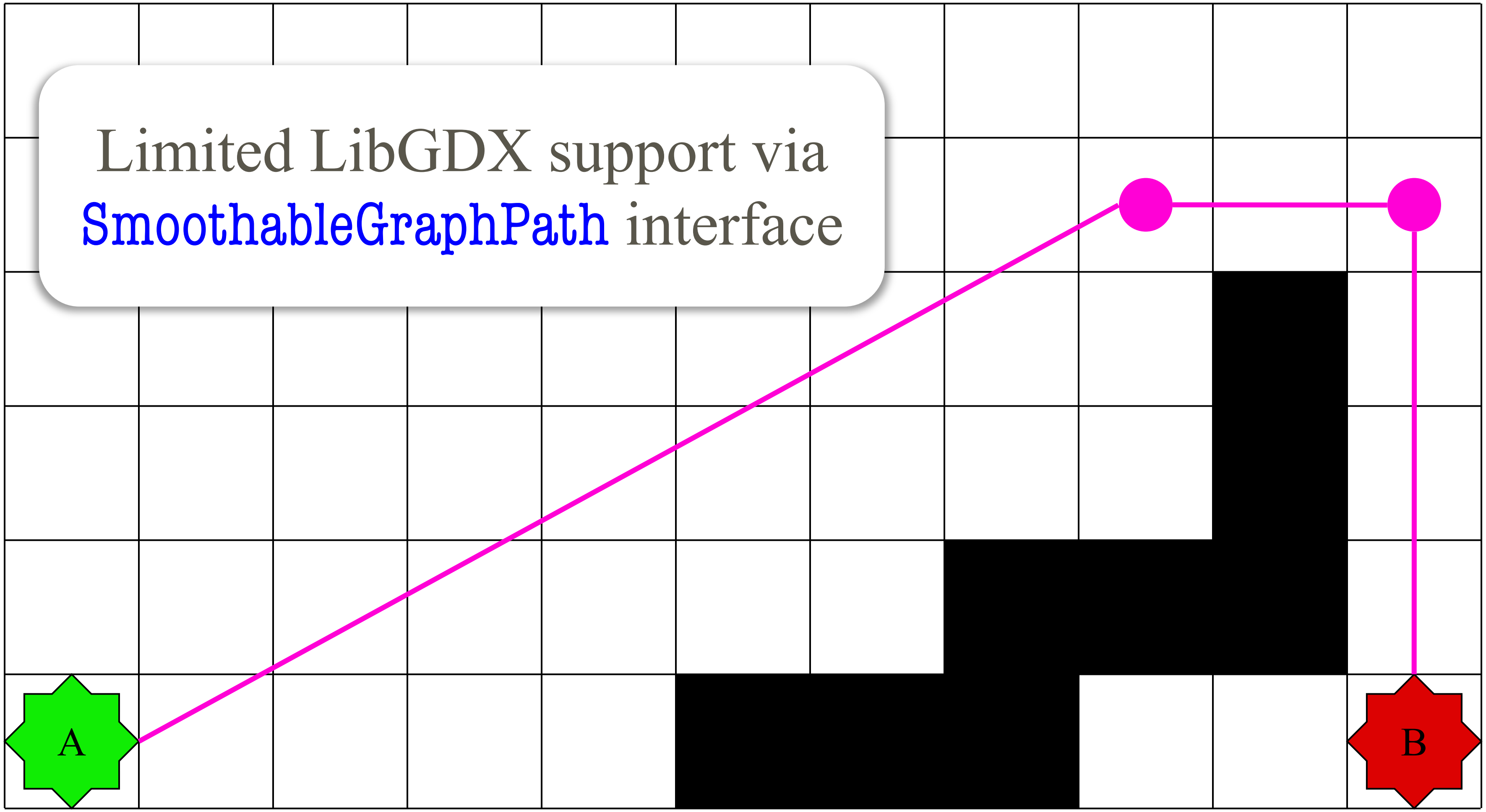


the gamedesigninitiative at cornell university

Path Smoothing

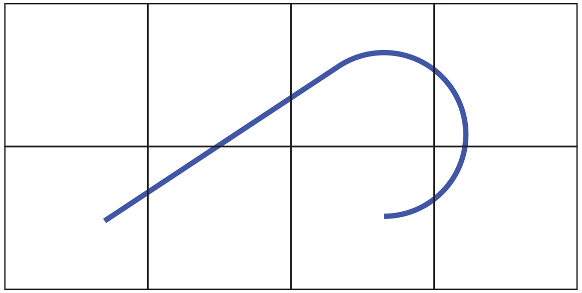


Path Smoothing

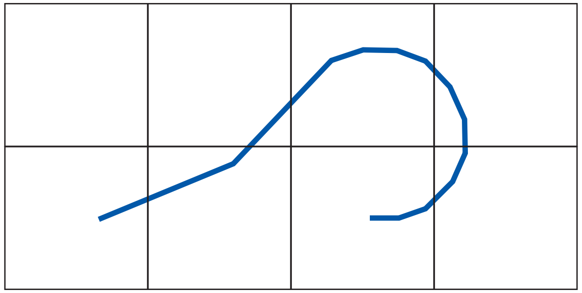


Turning

- **Realistic** turns
  - Smooth paths into line segments
  - Round corners for realistic movement



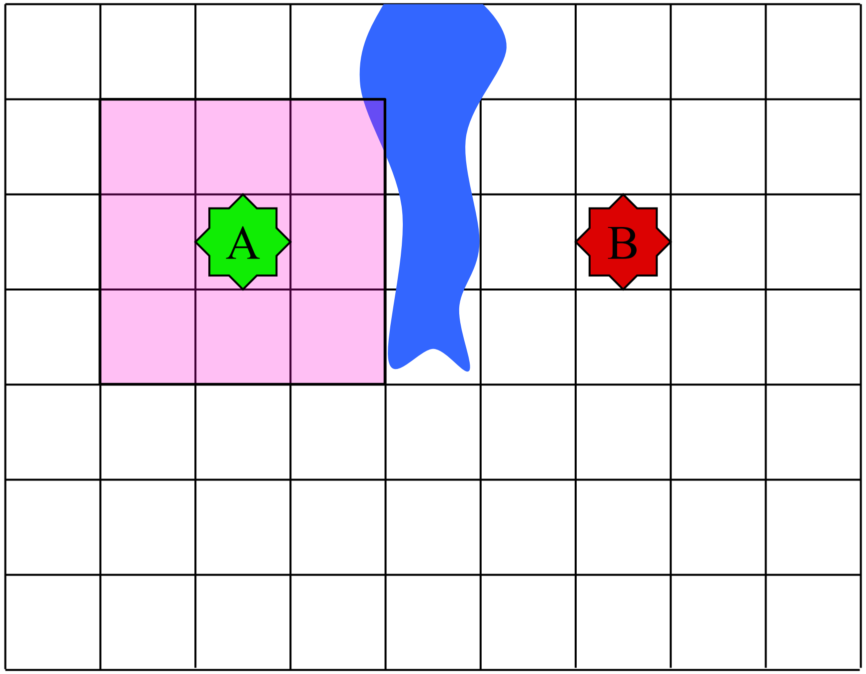
- **Restricted** turns
  - Limit turns to angles drawn by artist
  - 16 angles standard for 2D top-down



- See online reading for today
  - Pinter, "Toward More Realistic Pathfinding"
  - Techniques from the sprite days of RTSs

Multiple NPC Sizes

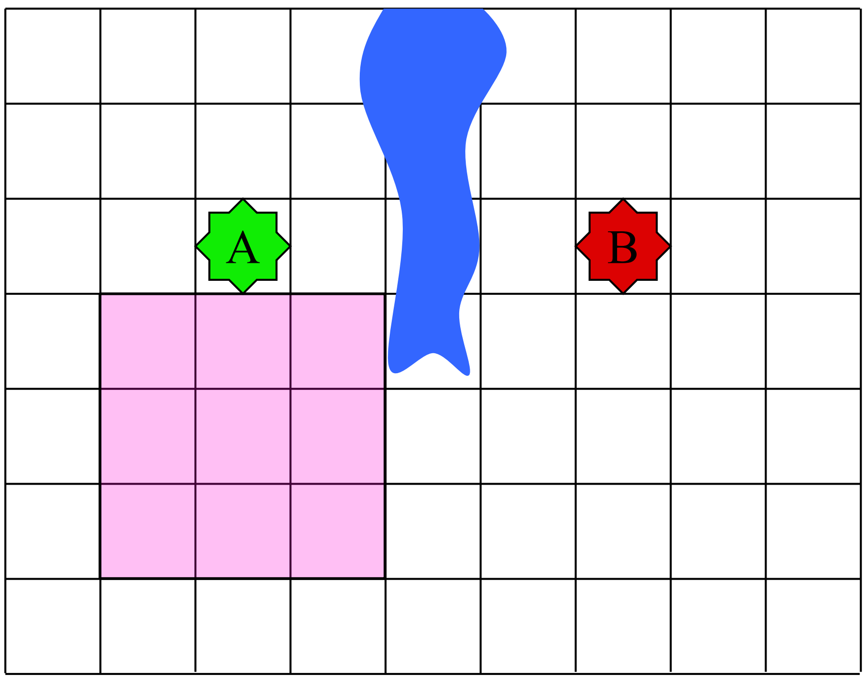
- Grid to largest NPC?
  - Bad for small units
  - Unnecessary blocking
- Grid to smallest NPC!
  - Multiple squares for larger
  - Center fits on grid square
- Pathfinding larger NPCs
  - A* for center-to-center
  - Size to check blocking
  - May alter the path



the gamedesigninitiative at cornell university

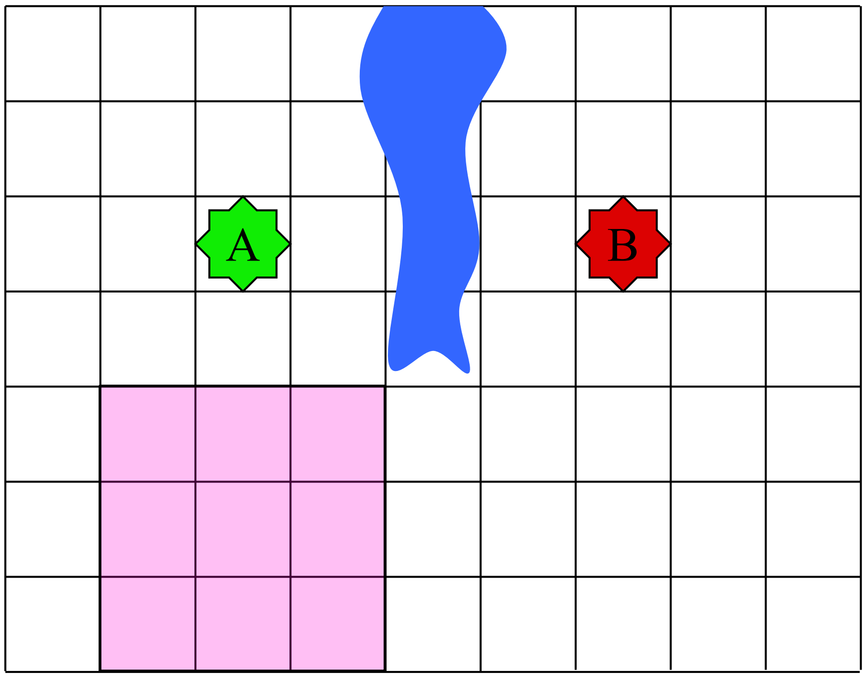
Multiple NPC Sizes

- Grid to largest NPC?
  - Bad for small units
  - Unnecessary blocking
- Grid to smallest NPC!
  - Multiple squares for larger
  - Center fits on grid square
- Pathfinding larger NPCs
  - A* for center-to-center
  - Size to check blocking
  - May alter the path



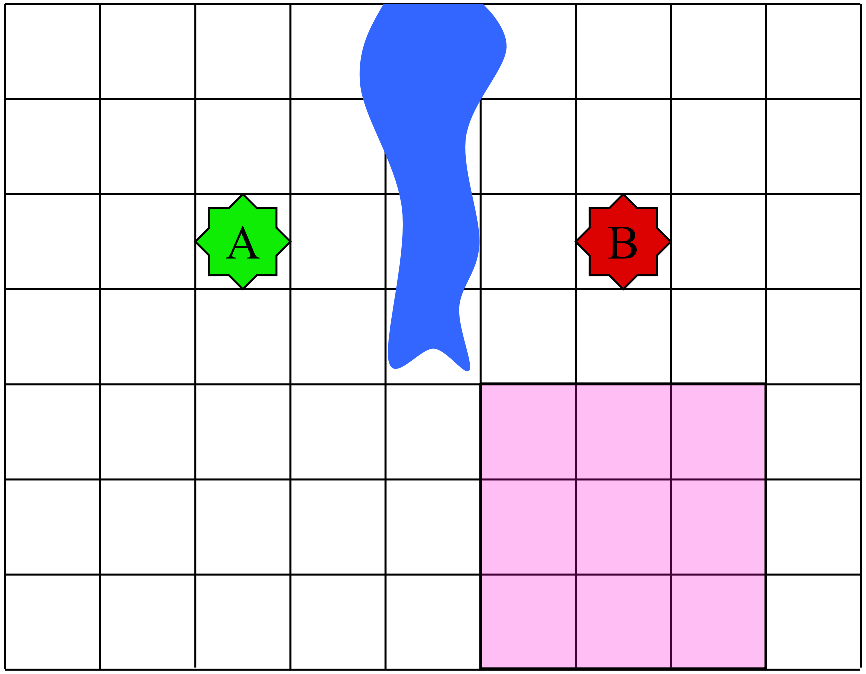
Multiple NPC Sizes

- Grid to largest NPC?
  - Bad for small units
  - Unnecessary blocking
- Grid to smallest NPC!
  - Multiple squares for larger
  - Center fits on grid square
- Pathfinding larger NPCs
  - A* for center-to-center
  - Size to check blocking
  - May alter the path



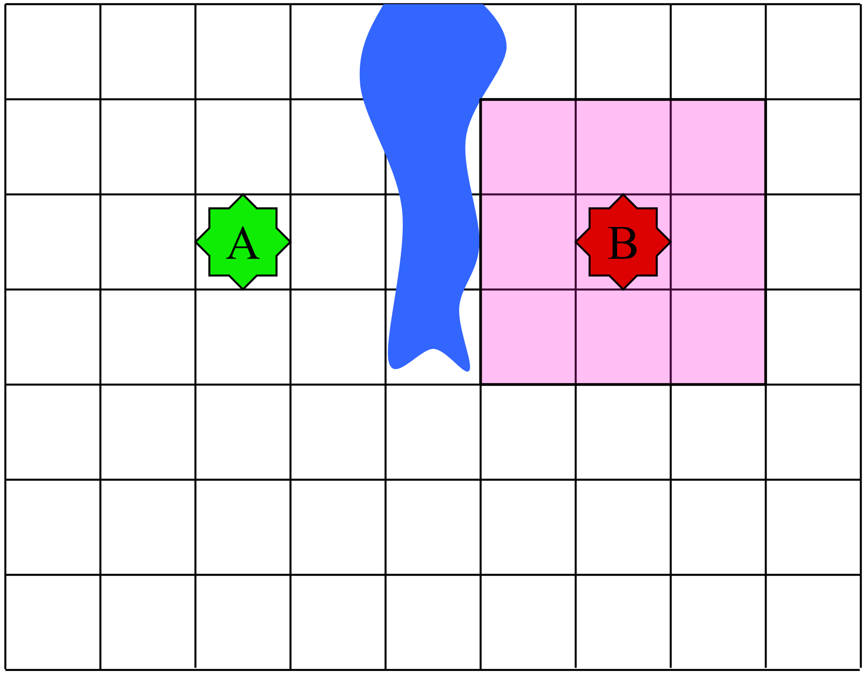
Multiple NPC Sizes

- Grid to largest NPC?
  - Bad for small units
  - Unnecessary blocking
- Grid to smallest NPC!
  - Multiple squares for larger
  - Center fits on grid square
- Pathfinding larger NPCs
  - A* for center-to-center
  - Size to check blocking
  - May alter the path



Multiple NPC Sizes

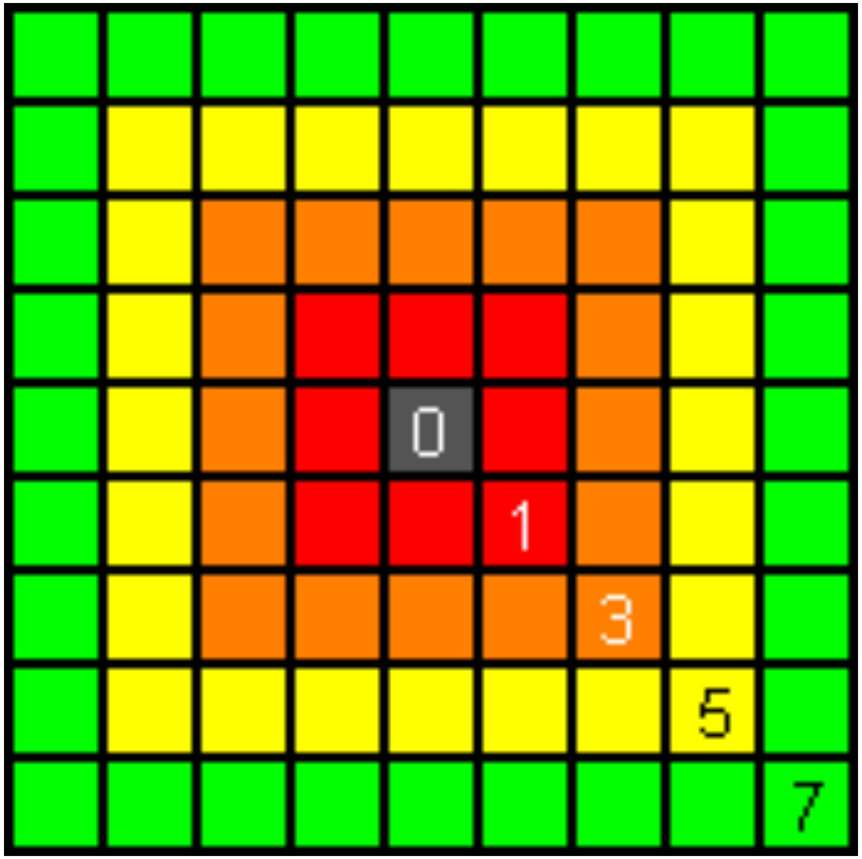
- Grid to largest NPC?
  - Bad for small units
  - Unnecessary blocking
- Grid to smallest NPC!
  - Multiple squares for larger
  - Center fits on grid square
- Pathfinding larger NPCs
  - A* for center-to-center
  - Size to check blocking
  - May alter the path



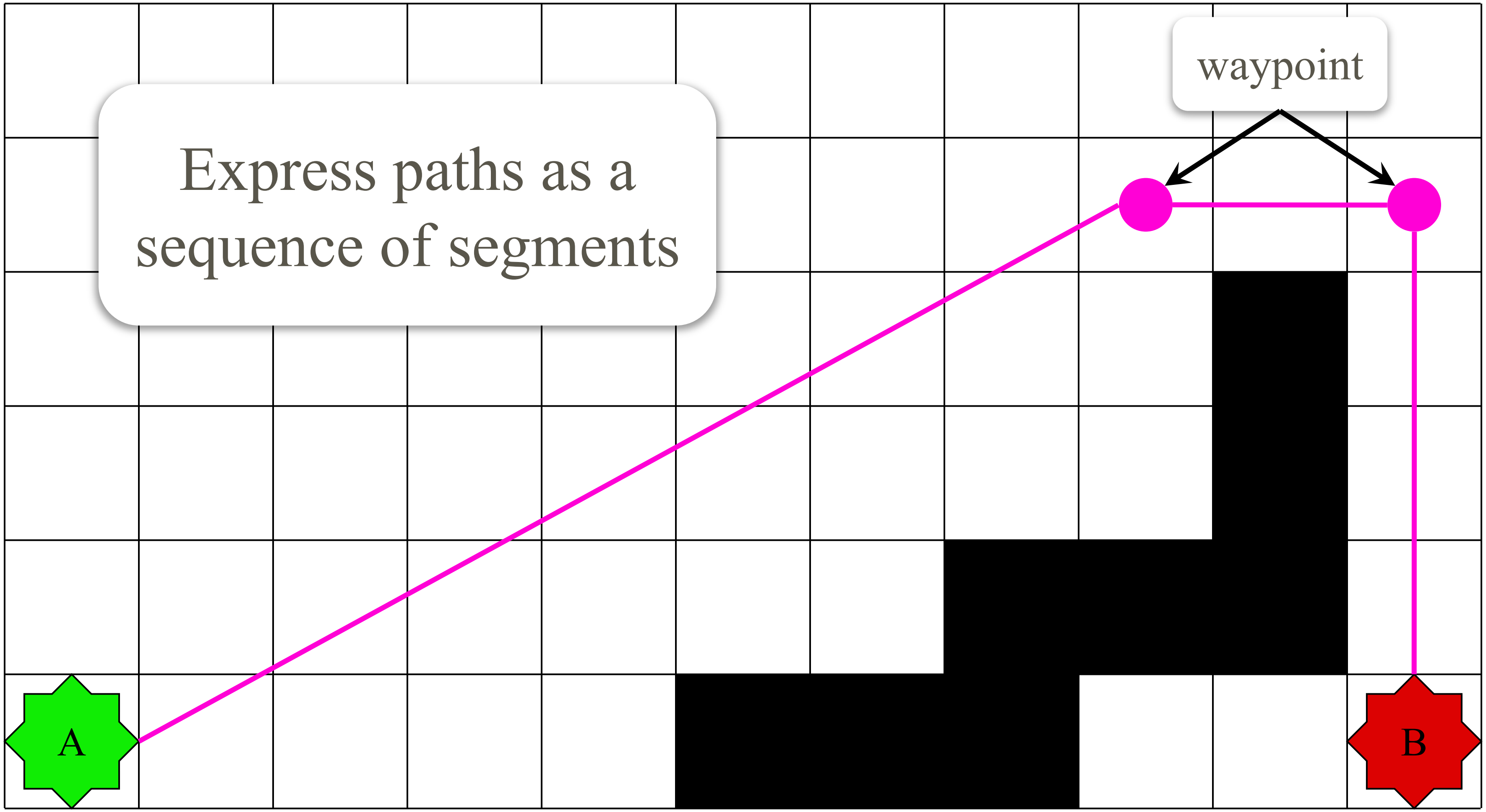
Multiple NPC Sizes

Fitting NPCs on a Grid

- Assume NPC is square
  - Represents "reach"
  - Simplifies turning
- Requires "odd" sizes
  - Center must be a grid
  - Radius in full grid squares
  - What about even sizes?
- "**Tabletop**" solution
  - Round down when moving
  - Round up when in place

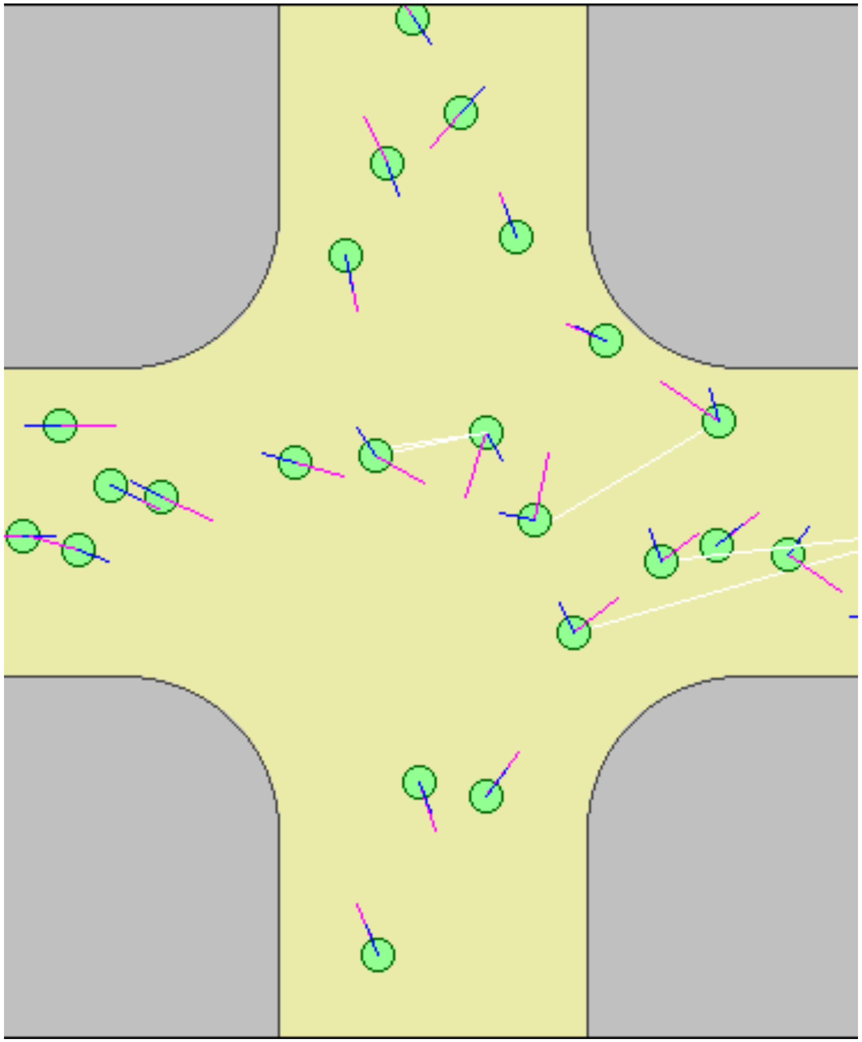


Waypoints



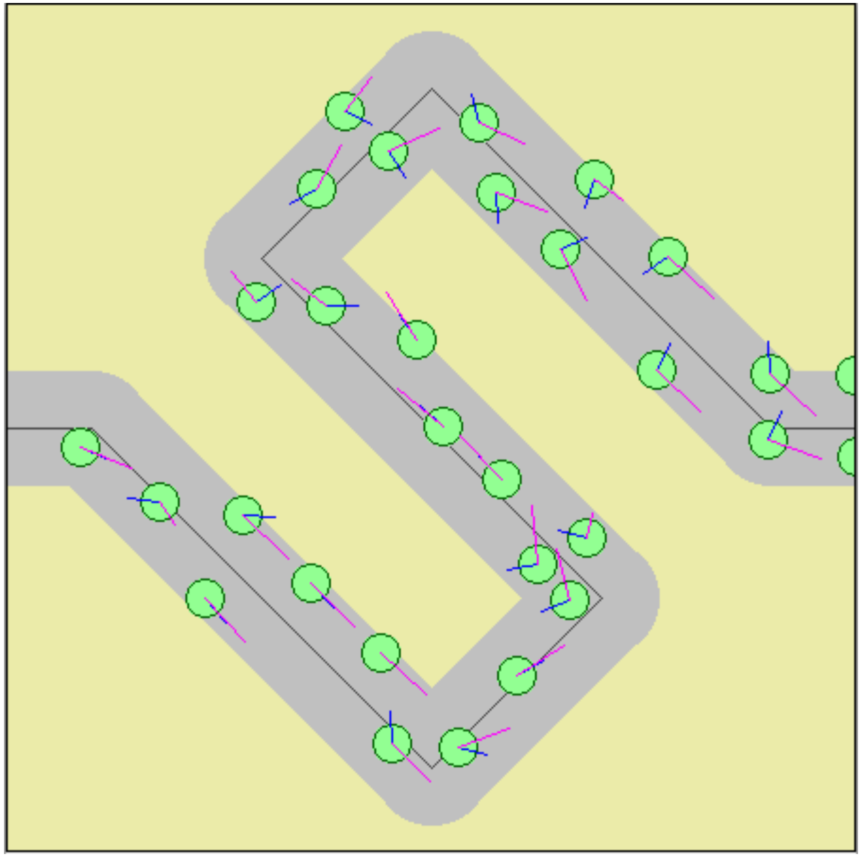
Steering

- Alternative to pathfinding
  - Uses forces to move NPCs
  - Great for **small** paths
- **Examples**
  - Artificial potential fields
  - Vortex fields
  - Custom steering behaviors
- See Craig Reynold's page
  - See "Physics & Motion"
  - com.badlogic.gdx.ai.steer



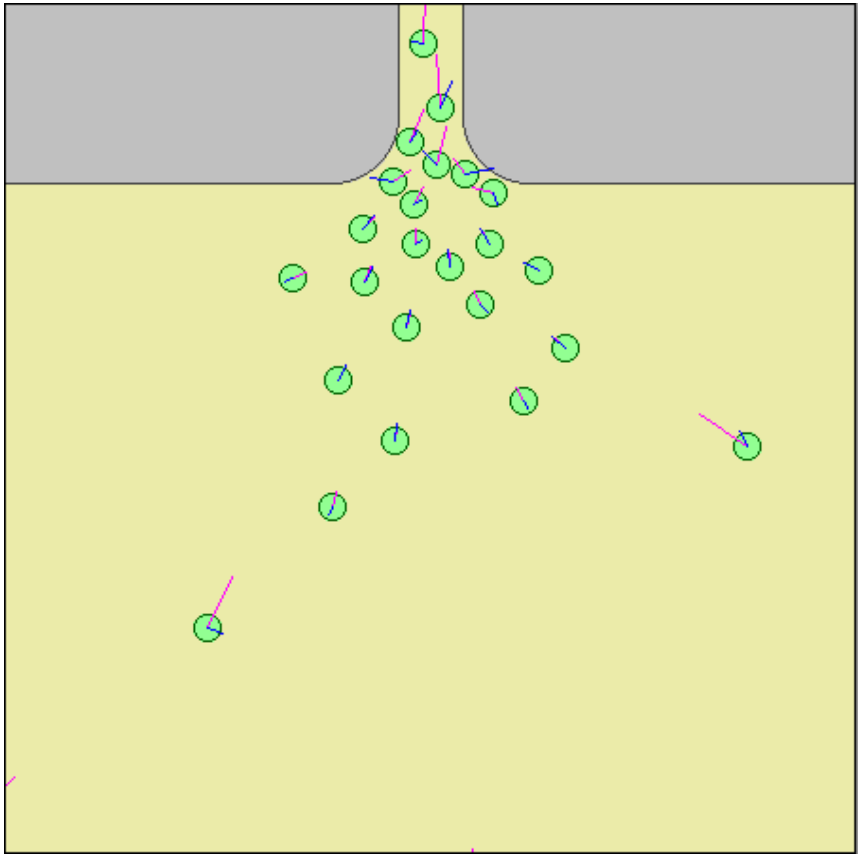
Steering and Pathfinding

- Use waypoint as "goal"
  - Attract NPC to waypoint
  - When close, next waypoint
- Great for multiple NPCs
  - Pathfind for largest NPC
  - Steering to move along path
  - Repulsion keeps NPCs apart
- **Drawbacks**:
  - Military formations are hard
  - Get stuck at bottlenecks



Dynamic Obstructions

- Others can get in way
  - Enemies guarding locale
  - Friends waiting in queue
- Correct response?
  - Compute a new path?
  - Wait to be unblocked?
- What would you do?
  - See what is blocking
  - Making an educated guess
  - Character AI solution



Why Obstructions Matter

Steering Interfaces in LibGDX

Steerable

- Access to **physics data**
  - `getLinearVelocity()`
  - `getAngularVelocity()`
  - `getBoundingRadius()`
- Also has **limiter** info
  - `get/setMaxLinearSpeed()`
  - `get/setMaxAngularSpeed()`
  - `get/setMaxLinearAccel()`
  - `get/setMaxAngularAccel()`

SteeringBehavior

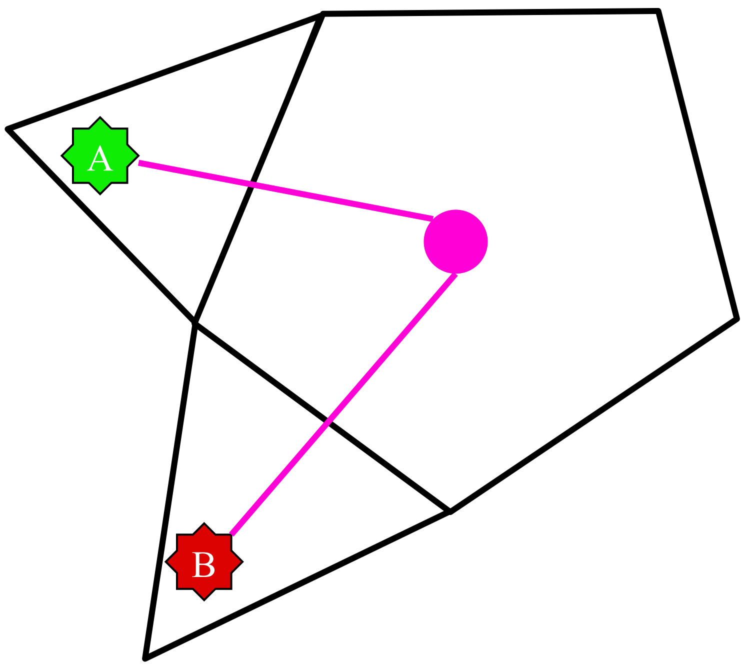
- Has a Steerable **owner**
  - Object being steered
- Other potential attributes
  - **Target** (goal location)
  - **Path** (path following)
- Calcs `SteeringAcceleration`
  - Physics *recommendation*
  - DOES NOT set physics

Pathfinding in Practice

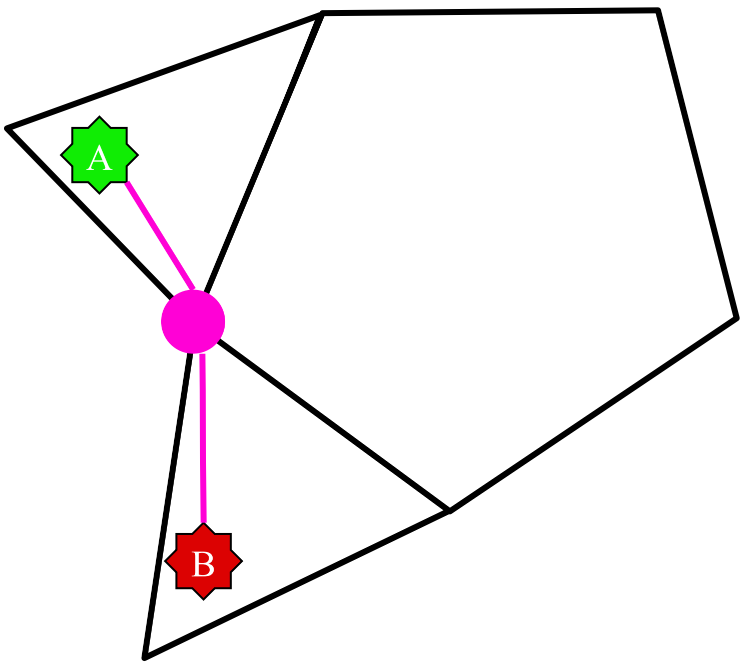
- Navigation Meshes
  - Indicates walkable areas
  - 2D geometric representation
  - Connected convex shapes
  - A* graph: center-to-center
- Making Nav Meshes
  - Often done by level editor
  - Can be modified by hand
  - Annotate special movement
  - **Example**: jump points

Easy Pathfinding on Meshes

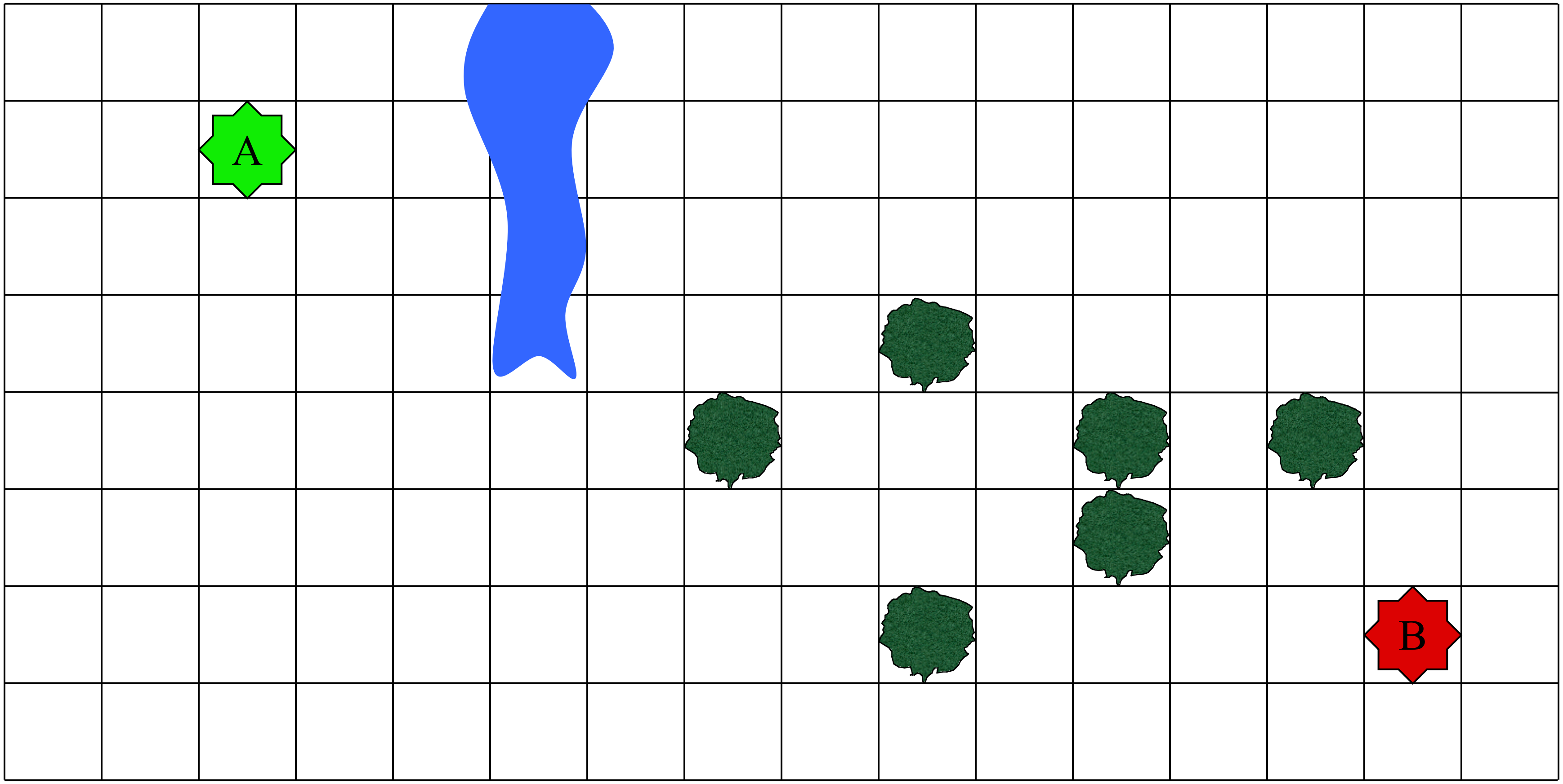
Center of each Region



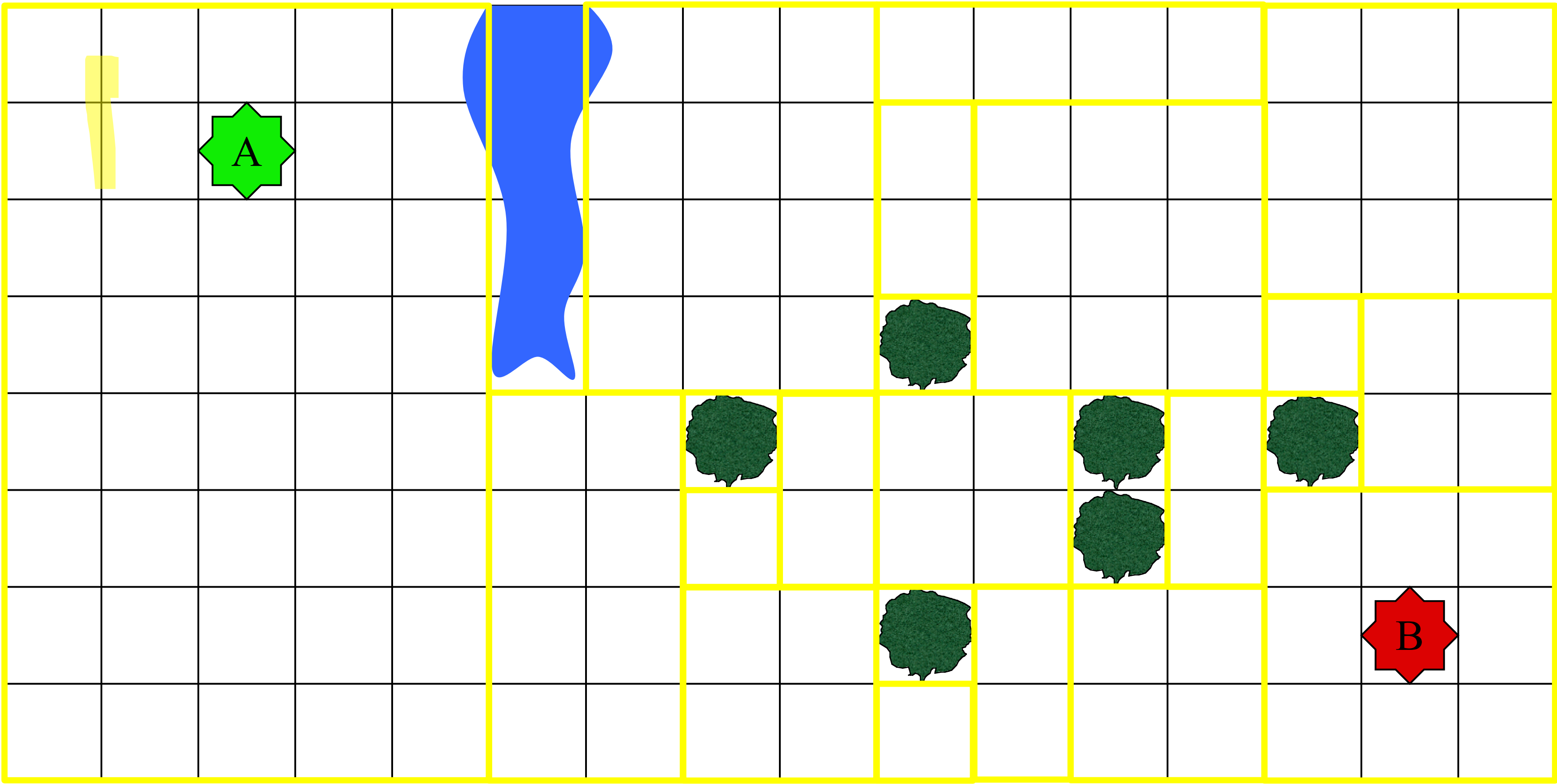
Corners of the Mesh



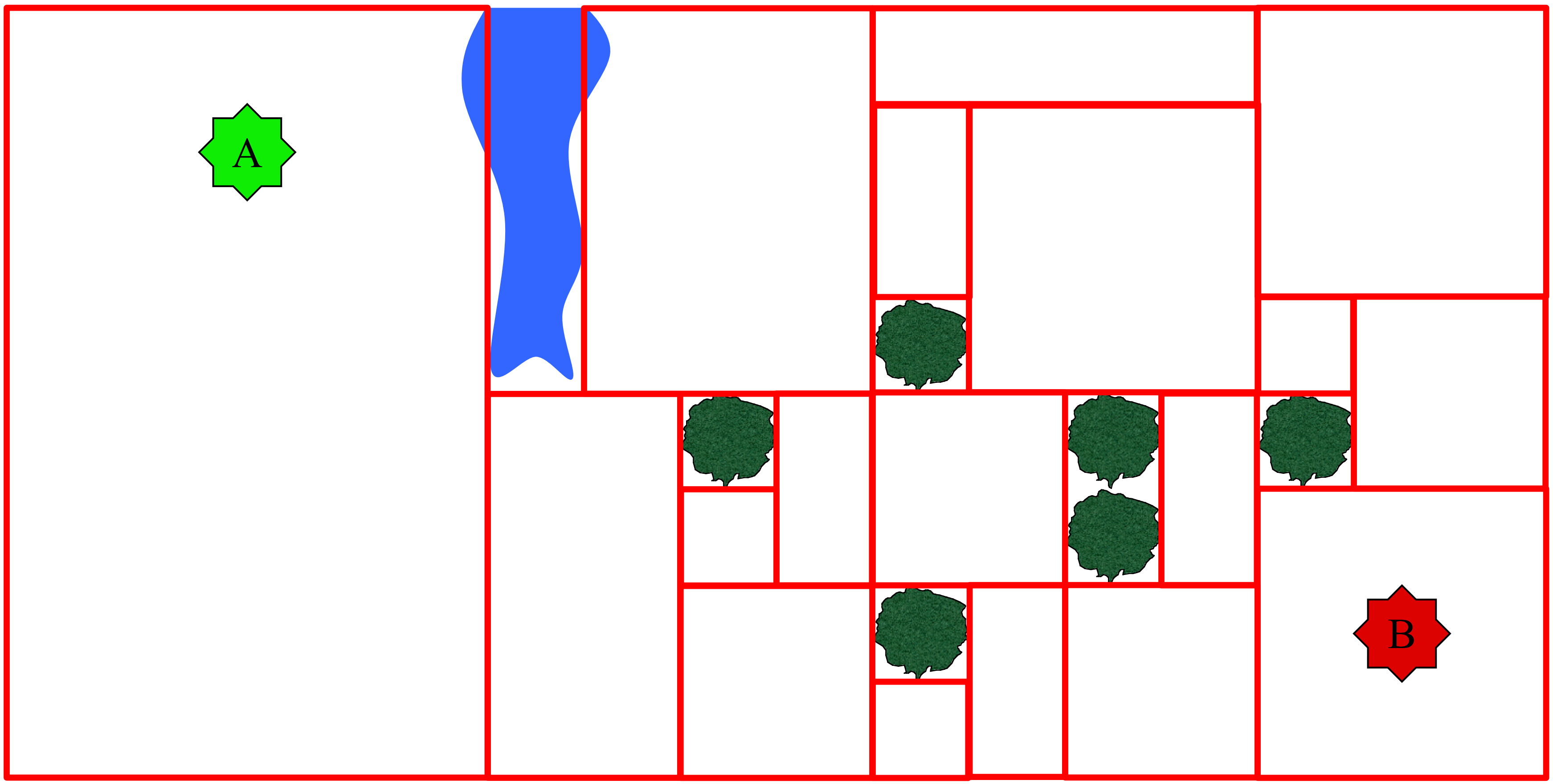
Optimization: Hierarchical Pathfinding



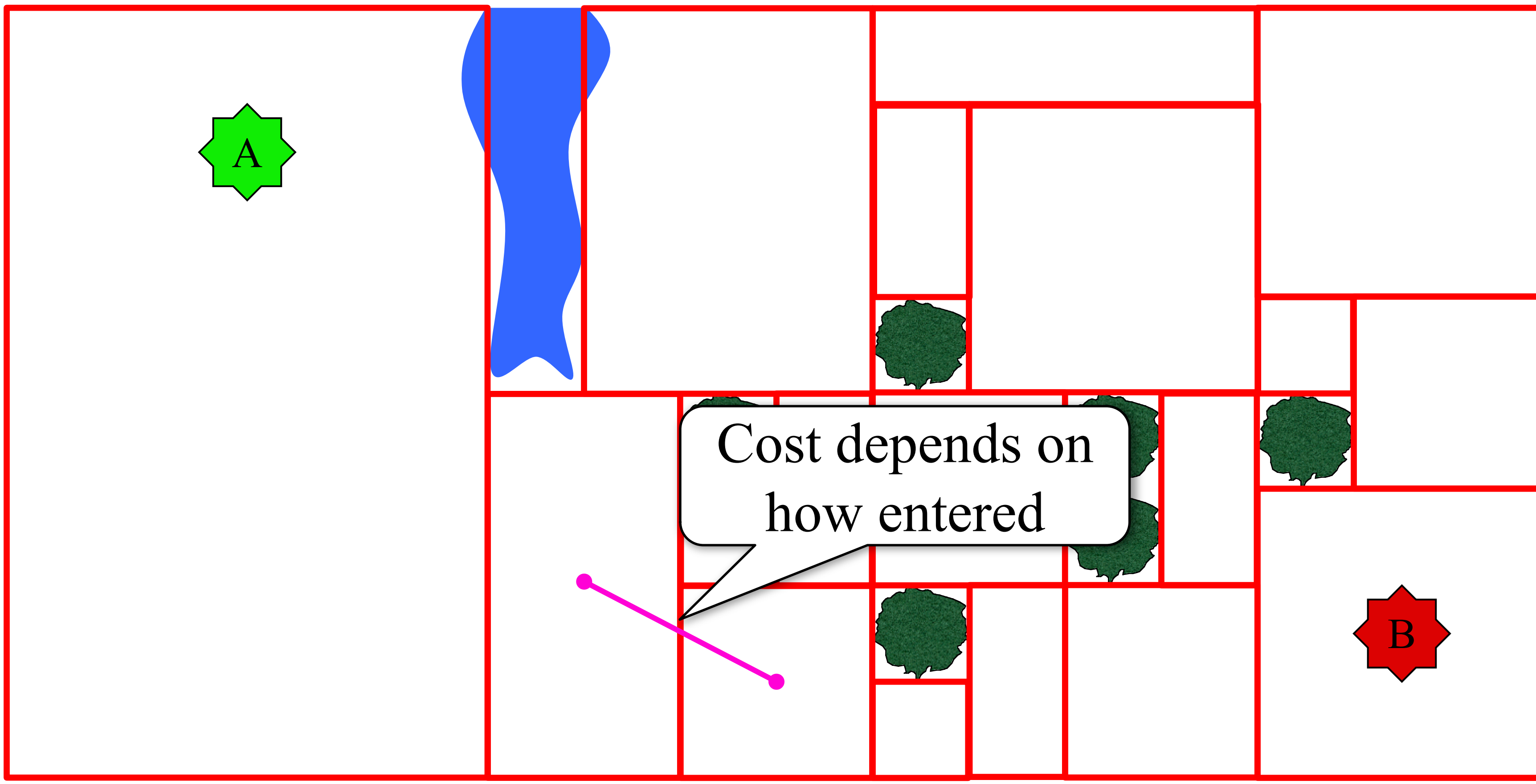
Optimization: Hierarchical Pathfinding



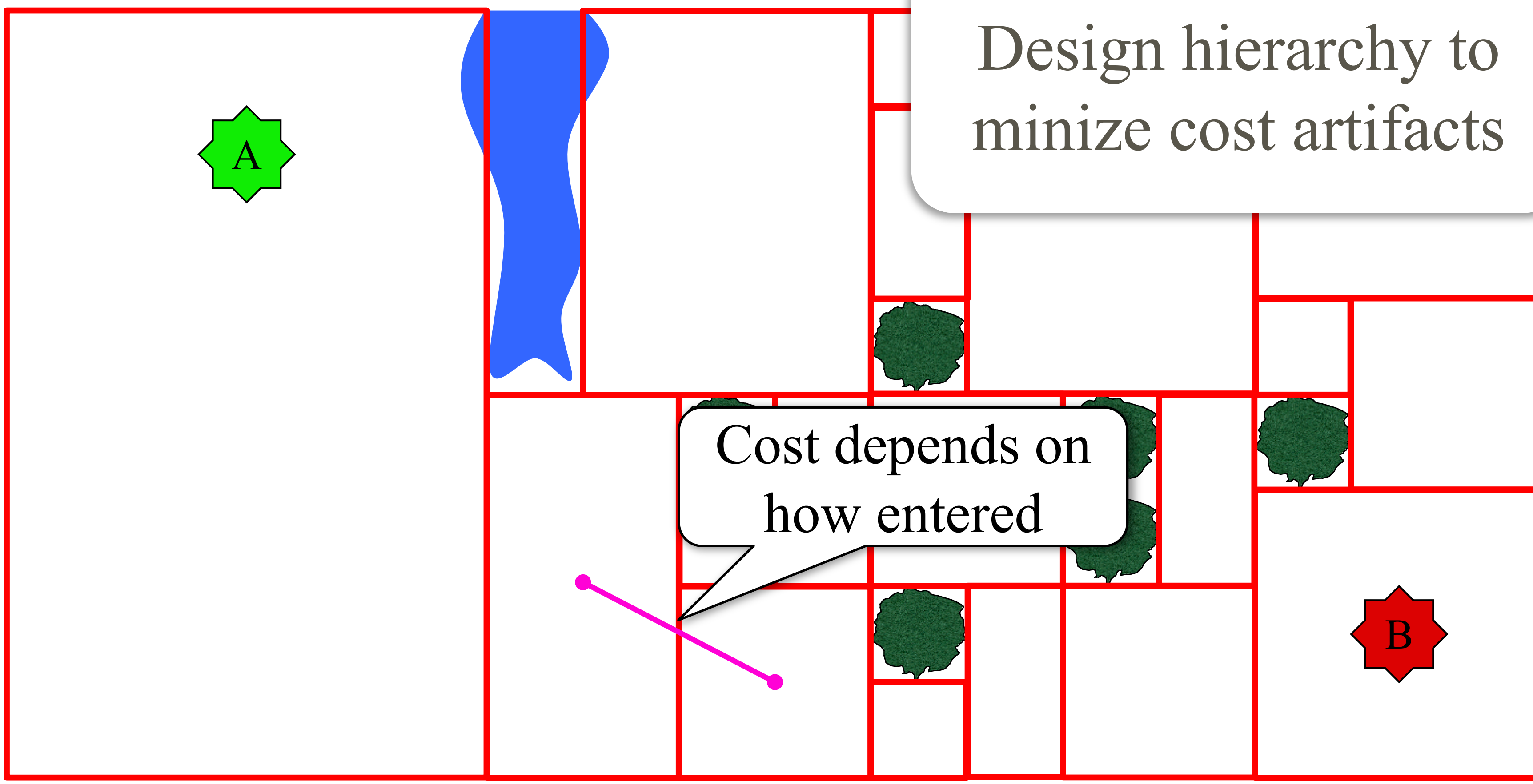
Optimization: Hierarchical Pathfinding



Optimization: Hierarchical Pathfinding



Optimization: Hierarchical Pathfinding



Design hierarchy to minize cost artifacts

LibGDX Support

HierachicalGraph

- Graph with multiple levels
  - Has a current active level
  - Graph API matches level
  - Can switch this level on fly
- Also can convert levels
  - node + level => node
  - Rules to group nodes
  - Rules to split nodes

HierachicalPathFinder

- Specify a pathfinder to use
  - Could be A* or otherwise
  - Will use it on each level
- The implementation
  - Finds path at highest level
  - Expands nodes to next level
  - Refines path to expansion
  - Repeats until level 0

LibGDX Support

HierachicalGraph

- Graph with multiple levels
  - Has a current active level
  - Graph API r… vel
  - … on fly
- Als… can convert levels
  - node + level => node
  - Rules to group nodes
  - Rules to split nodes

Interface

HierachicalPathFinder

- Specify a pathfinder to use
  - Could be A* or otherwise
  - Will use it on … el
- Th…
  - F… path at highest level
  - Expands nodes to next level
  - Refines path to expansion
  - Repeats until level 0

Class

Summary

- **A* algorithm** is primary pathfinding tool
  - Make world into a grid/navigation mesh
  - Search for a path on associated graph
  - Adjust heuristics for terrain, threats
- But there are a lot of "special tricks"
  - Tricks to make movement realistic
  - Tricks to handle coordinated movement
  - Talk to Instructor (or TAs) if need more tricks