Greedy Search and Path Networks

Matthew Guzdial

guzdial@ualberta.ca

Announcements

- HW1 released Wednesday, due next Friday
- Now streaming lecture via Zoom for recording purposes
- Practice Quiz today! (None of the “make your own” questions in this one)
- Clarification on expected programming experience (Syllabus)

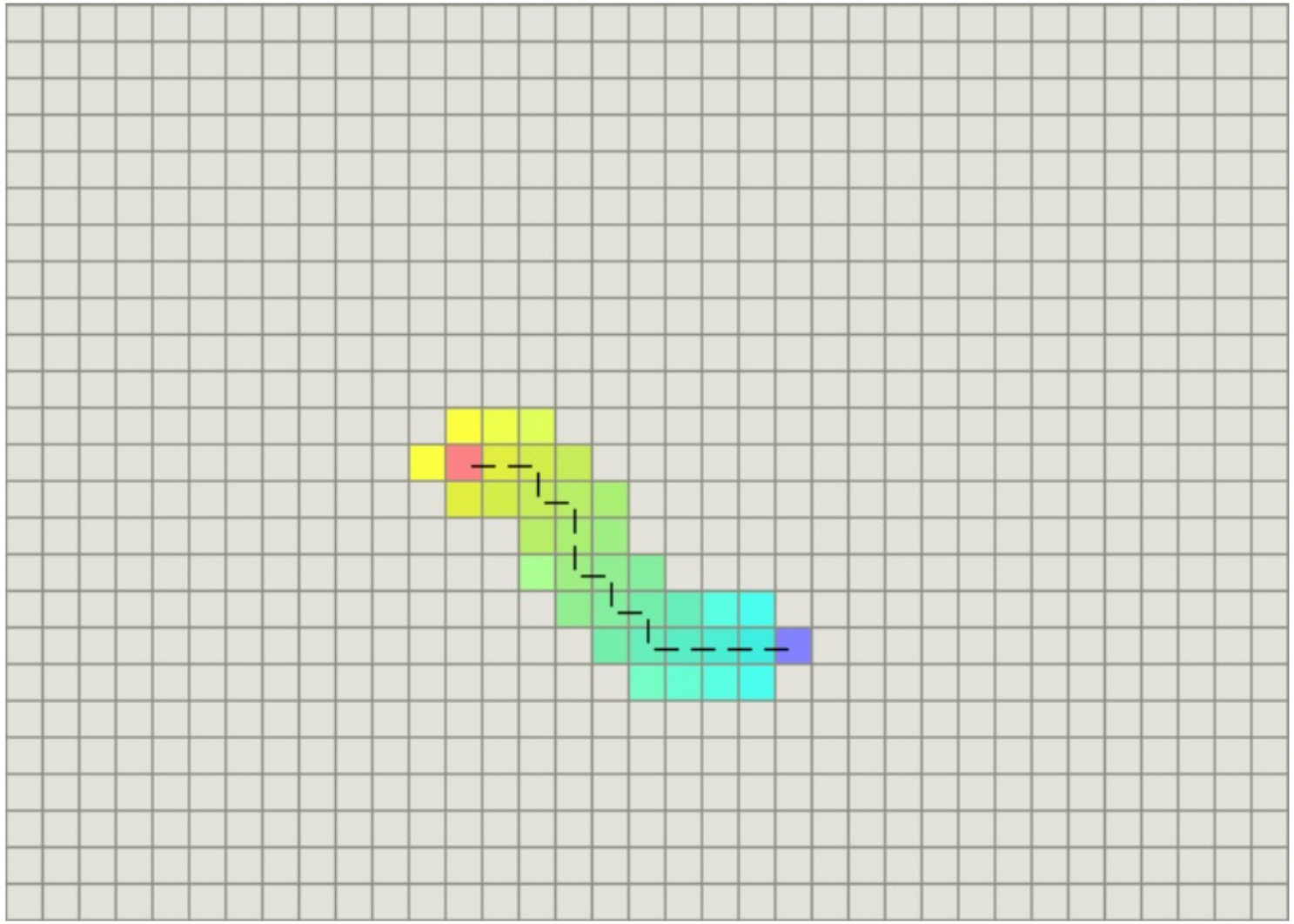
  “All assignments will take place in Unity with C# as the primary language. We will cover Unity basics in class. However, students will be required to be comfortable making use of the available Unity documentation. Students who do not have the required prerequisites at the time of taking this course should not expect supplementary professorial tutoring from the instructor.”

Last Class

- Introduction to grids, their pros and cons.

- Introduction to assignment 1.
  - Having trouble? Attend an office hour (they'll start next week, expect an email tonight)
  - Expect a helper video Monday + the lab Thursday
  - PLEASE DO NOT CHEAT

Greedy Path Search



Greedy Search: Path Planning

1. If current grid cell is the goal, stop the search and construct a path by retracing cell's parents.
2. Check each unchecked neighbor grid cell of the current grid cell.
3. Choose the closest grid cell to the goal according to some heuristic H(x,y) as the next current grid cell.
   - Set the previous current grid cell to this current grid cell's parent.
4. Go back to 1.

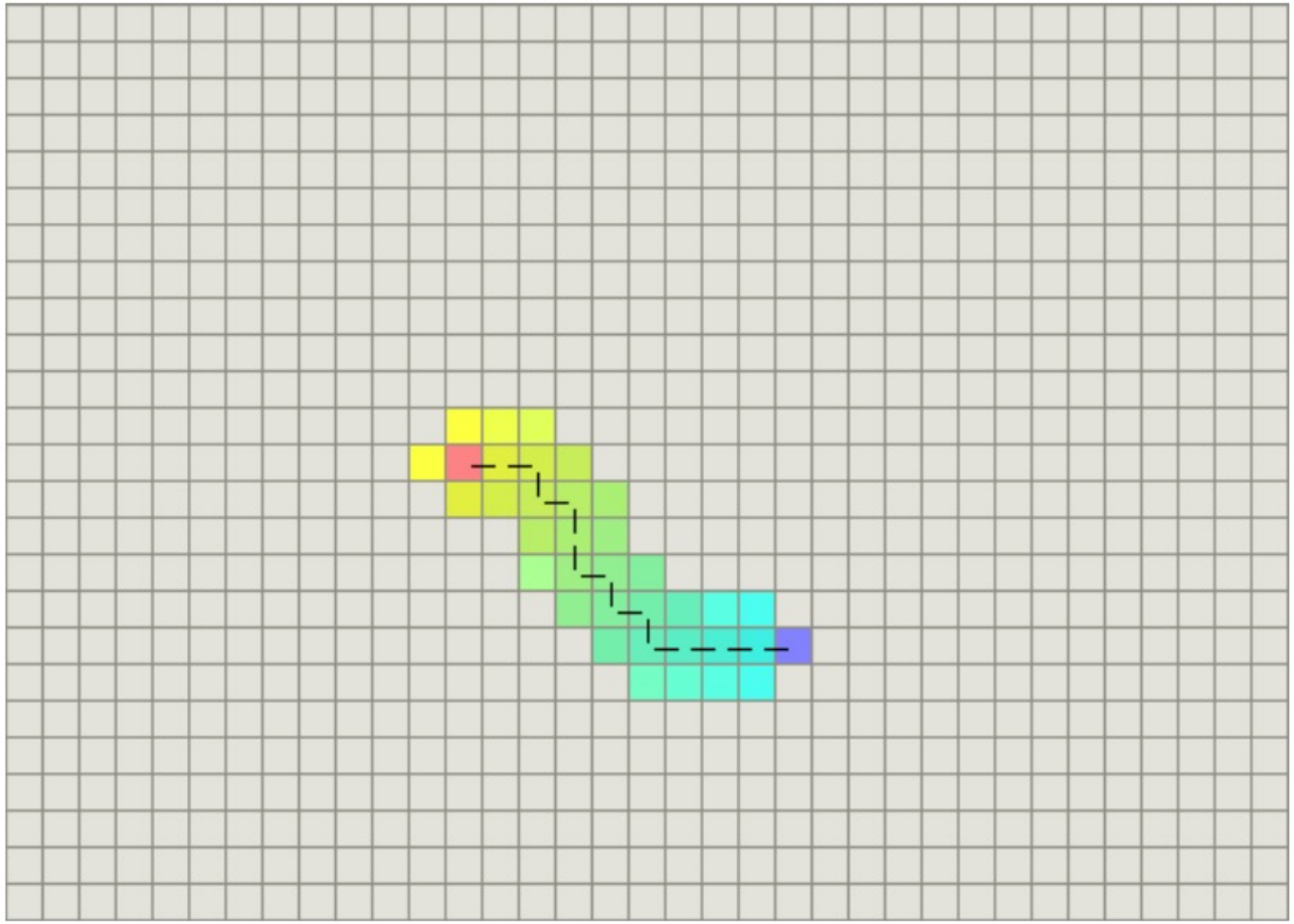
Greedy Search: Pseudocode

```
currNode = startingNode
while currNode != goalNode:
    neighbors = currNode.getNeighbors()
    bestNeighbor = None
    minDist = infinity
    for n in neighbors:
        n.parent = currNode
        if Heuristic(n)<minDist:
            bestNeighbor = n
            minDist = Heuristic(n)
    currNode = bestNeighbor
return ReconstructPath(currNode)
```

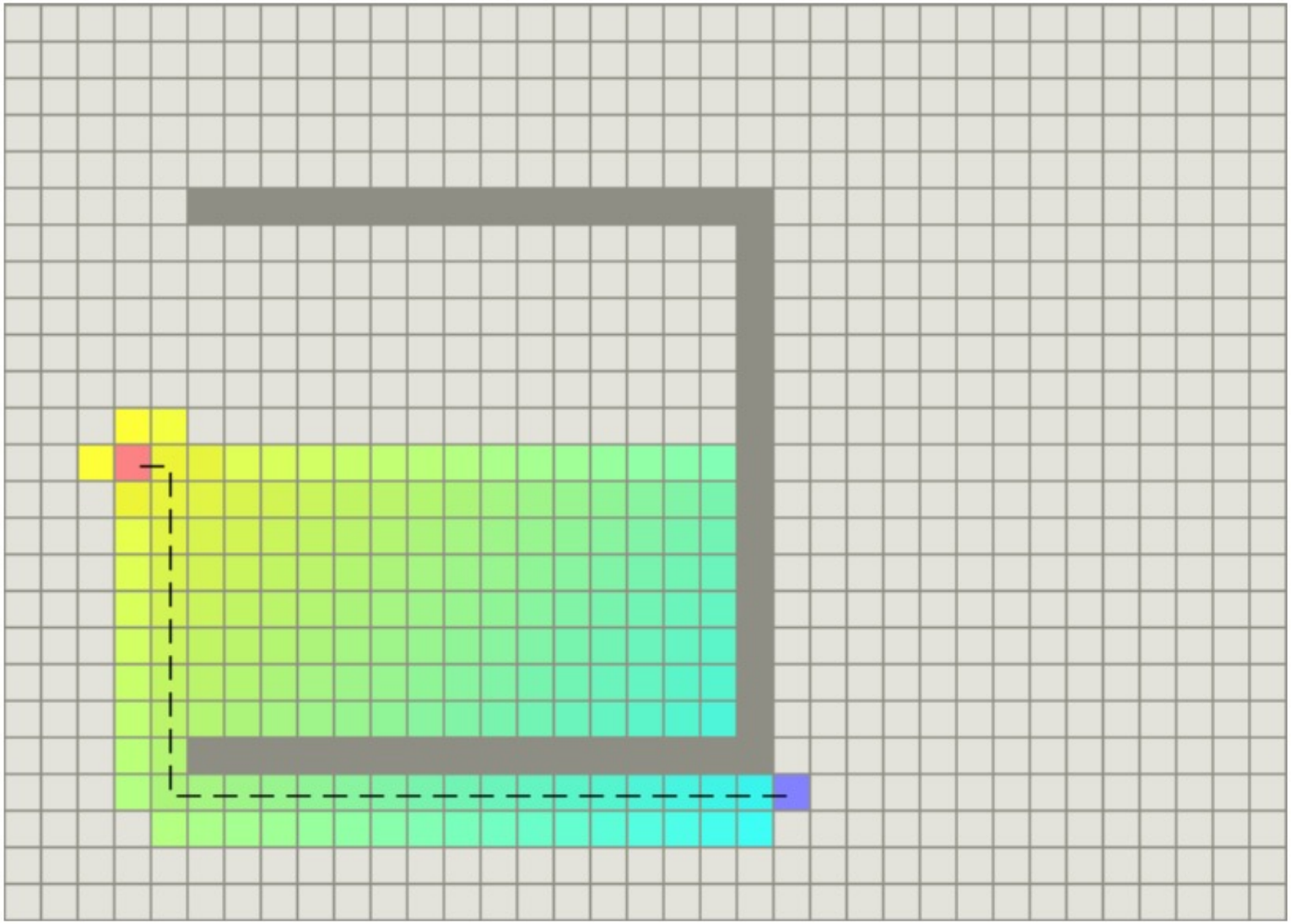
ReconstructPath Pseudocode

```
Path = []
Path.append(currNode)
while currNode.parent != None:
    Path.append(currNode.parent)
    currNode = currNode.parent
Path.reverse()
return Path
```

Greedy Path Search



Greedy Path Search



Heuristics in Game AI

- Typical heuristics: straight line distance, Manhattan distance, etc.

- In non-game AI, a Heuristic must *under-estimate* the remaining distance to a goal to be admissible.

- In Game AI we care less about admissibility, and more about how the heuristic appears to the user

Question 1: Greedy Search

https://forms.gle/xuFkbkBtRcyz7wiNA

https://tinyurl.com/guz-pq3

List the current nodes/grid cells in order according to the given greedy search algorithm and Manhattan distance ($abs(x_2-x_1)+abs(y_2-y_1)$).

Start: A, Goal: F

A	B	C	D	E		F	G
H	I	J	K	L		M	N
O	P	Q	R	S	T	U	V

Is there another heuristic that could have found the goal faster?

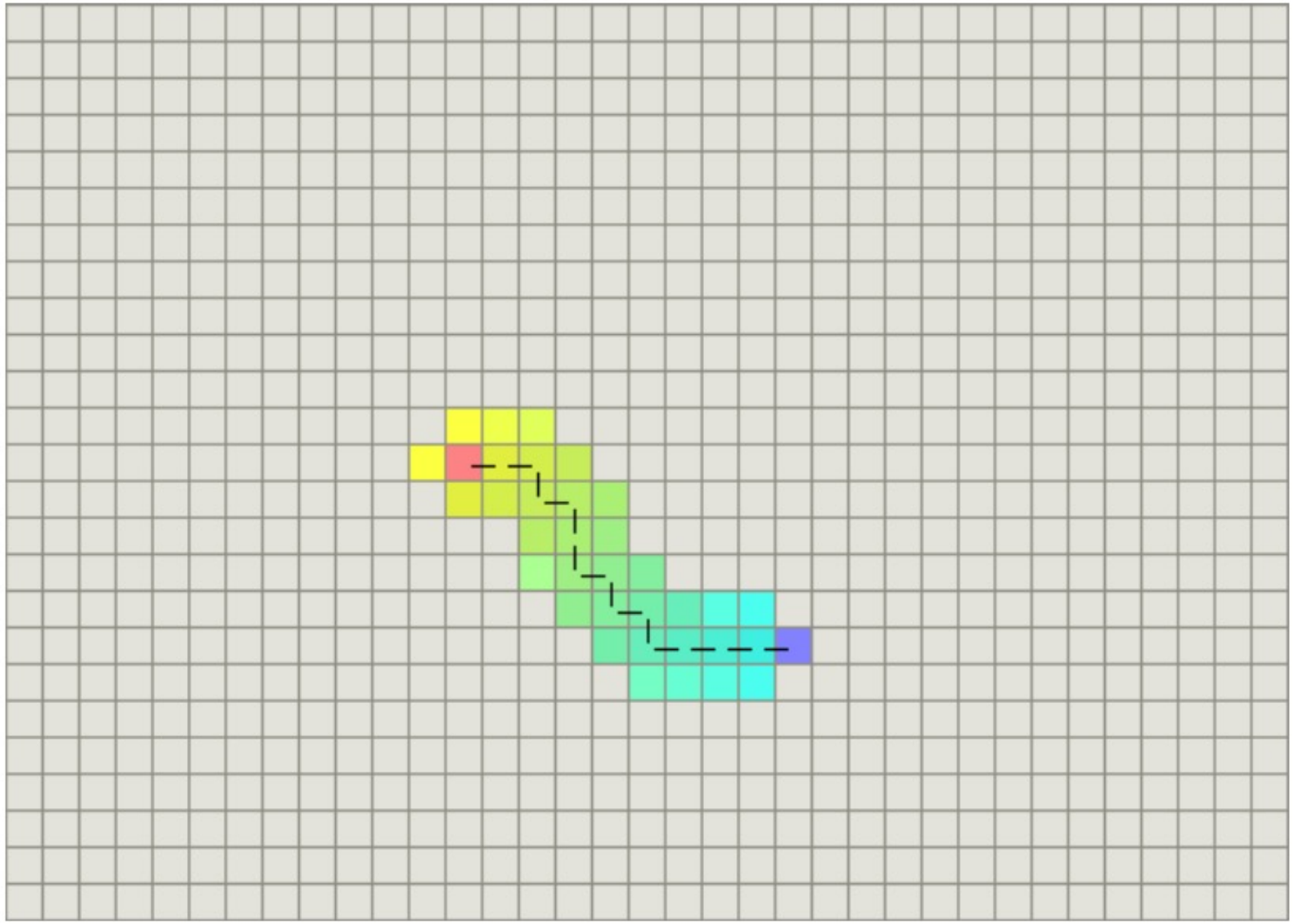
Answer: Greedy Search

A, B, C, D, E, L, S, T, U, M F

A	B	C	D	E		F	G
H	I	J	K	L		M	N
O	P	Q	R	S	T	U	V

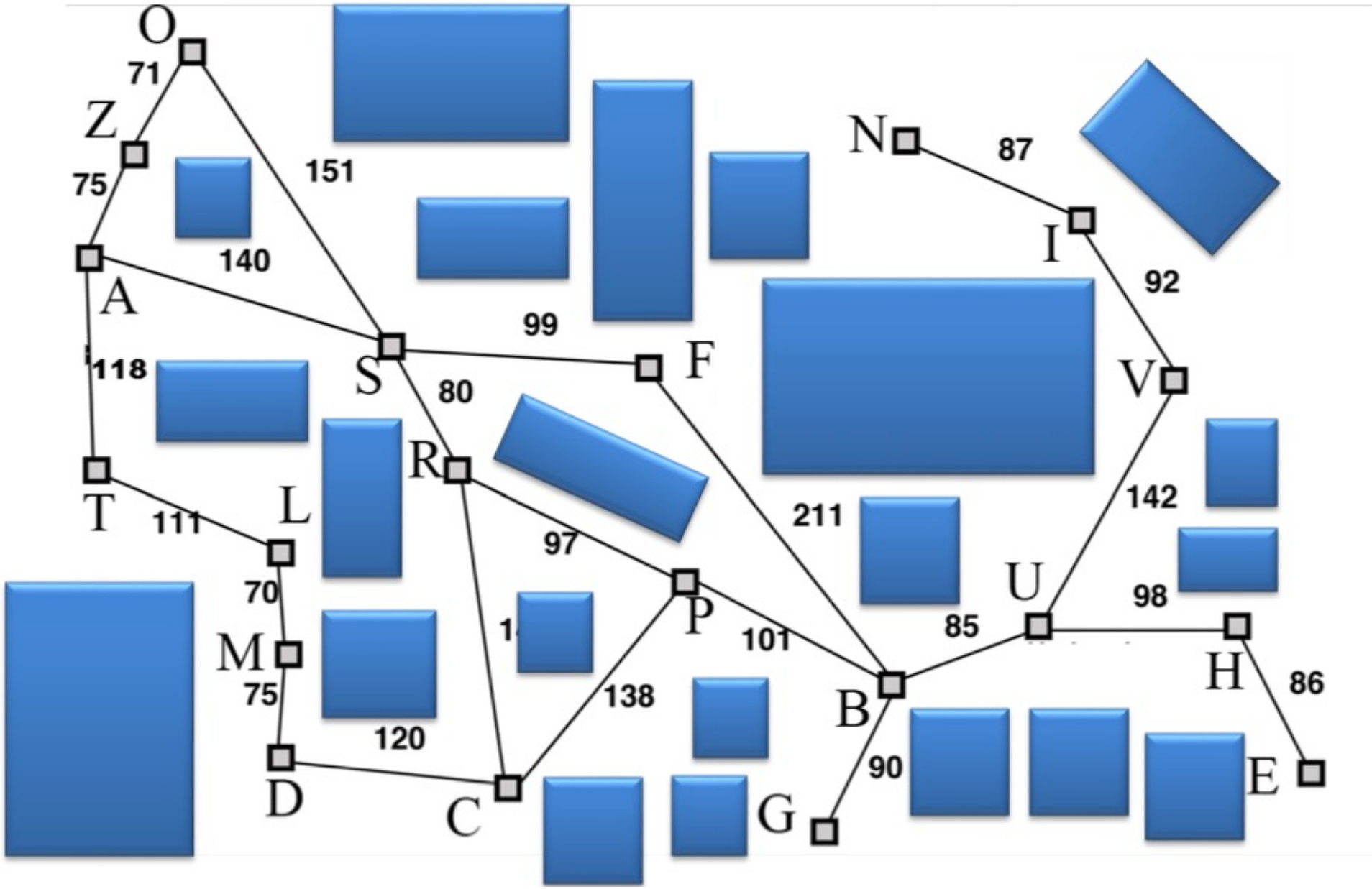
Example answer: All heuristics would lead to the same amount of steps, but a heuristic that discounted any cell with an obstacle between it and the goal might look more natural.

How can we change our **space representation** to avoid clunky movement?



Path Networks

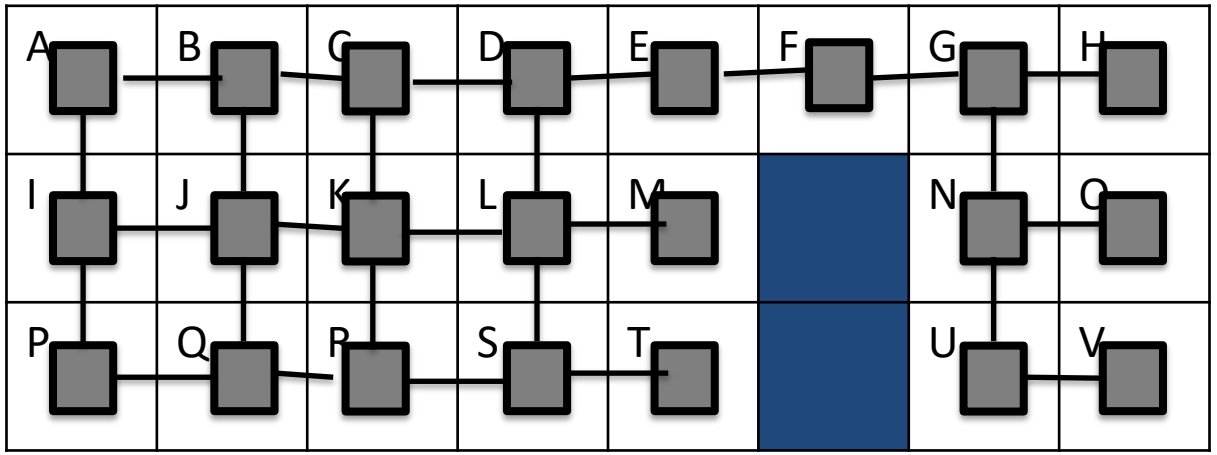
- Discretization of space into sparse network of nodes with paths between them as edges
  - Weight of edge, cost of travelling that path
- Connects points *visible to each other* in all important areas of map
- Usually hand-tailored (can use flood-fill)


O
71
Z
75
151
140
A
99
S
80
F
N
87
I
92
V
118
T
111
L
R
97
211
142
70
M
75
P
101
85
U
98
H
86
120
138
B
90
D
C
G
E

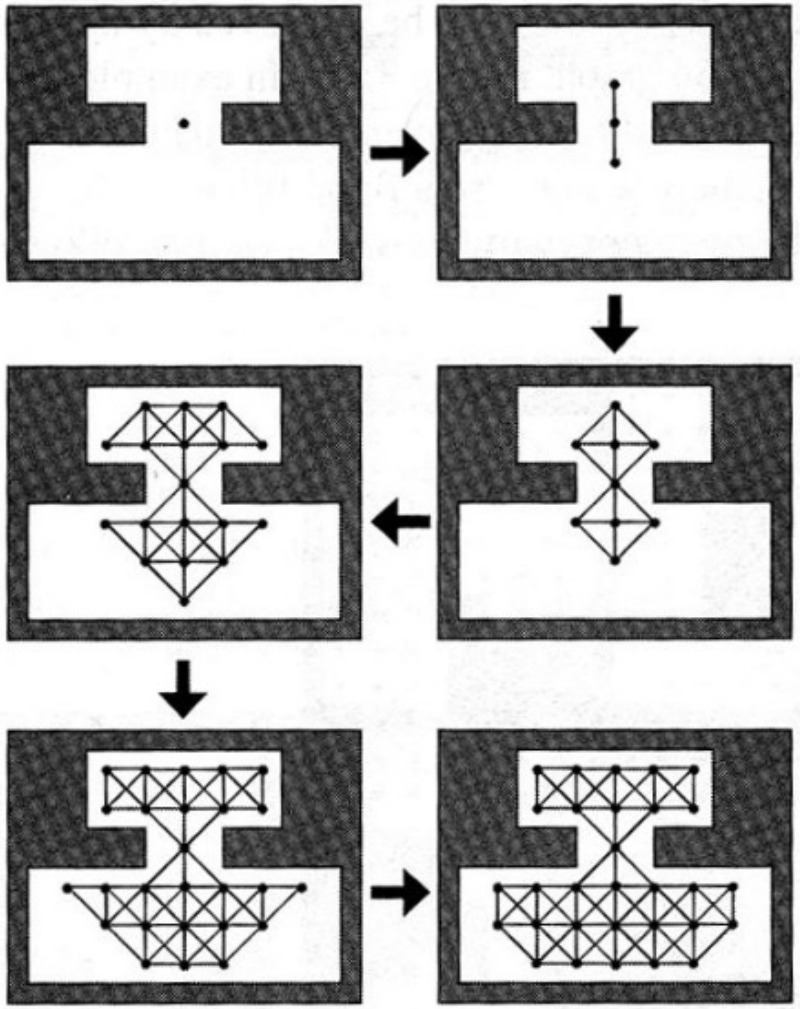
Grids can be thought of as just very uniform, dense path networks.

A	B	C	D	E	F	G	H
I	J	K	L	M		N	O
P	Q	R	S	T		U	V

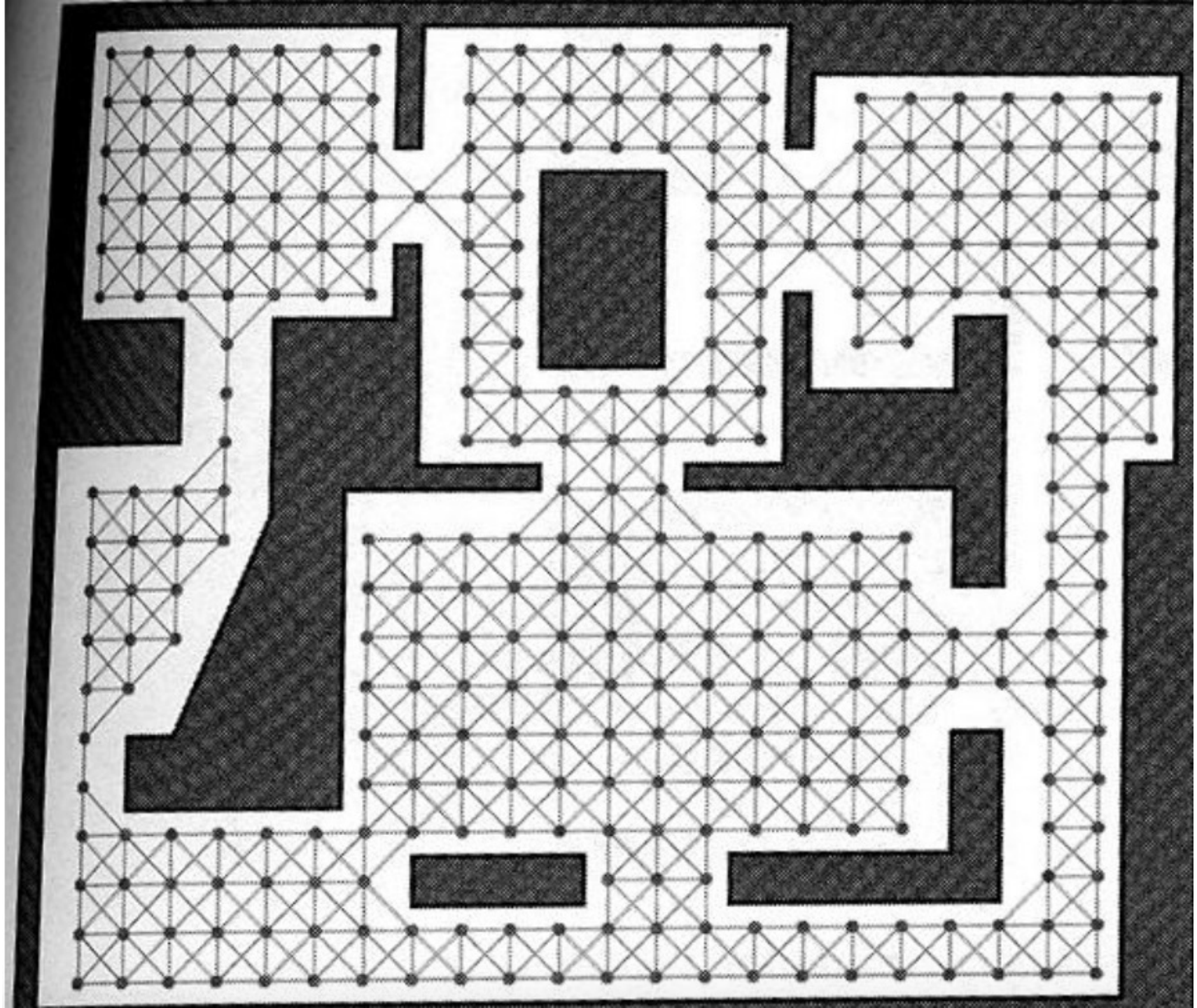
Grids can be thought of as just very uniform, dense path networks.



We can generate them! (Flood Fill)



- Start with “seed”
- “grow” graph
- Ensure all nodes and edges are at least as far from walls as agents bounding radius

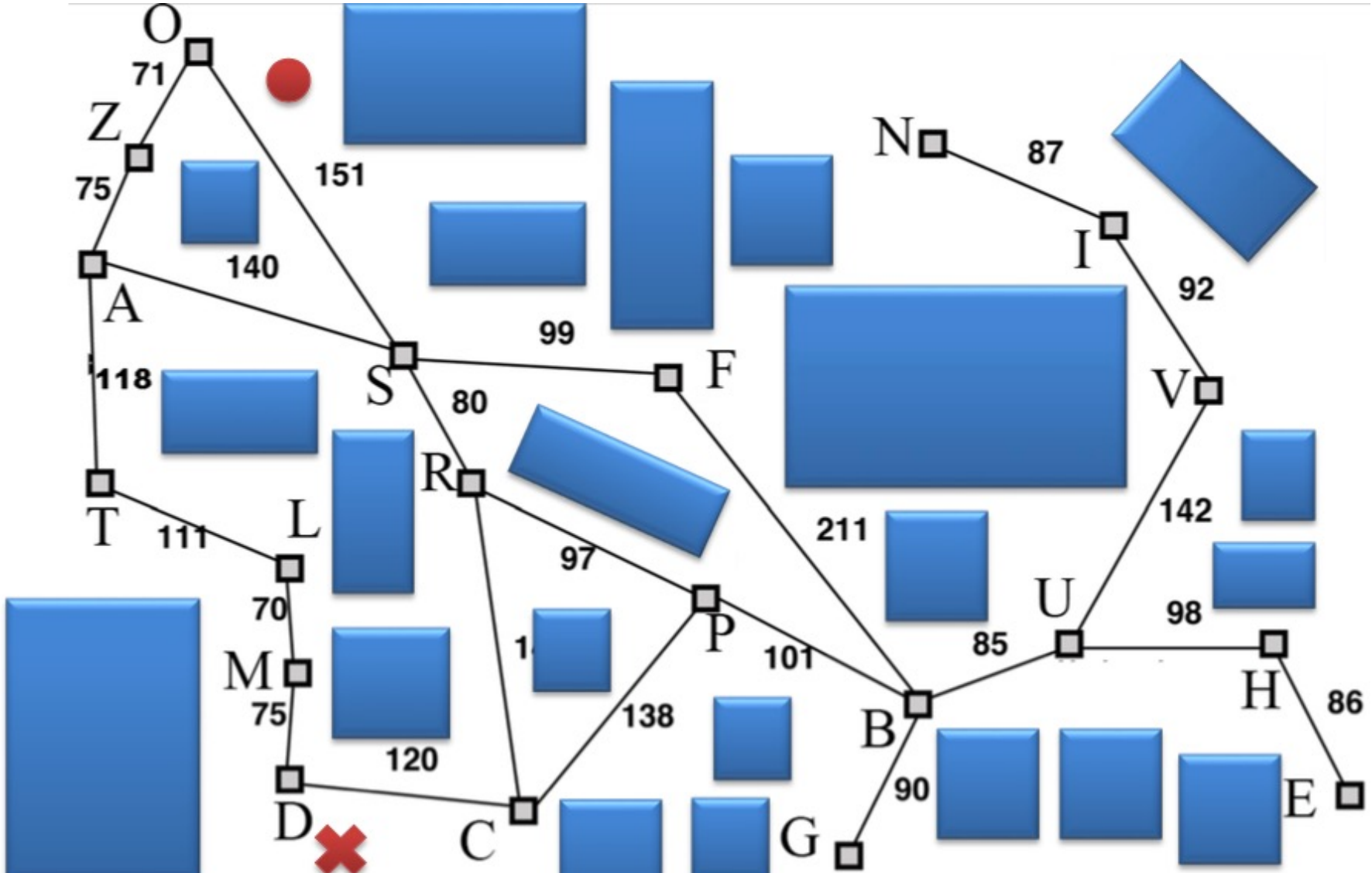


Using the path network

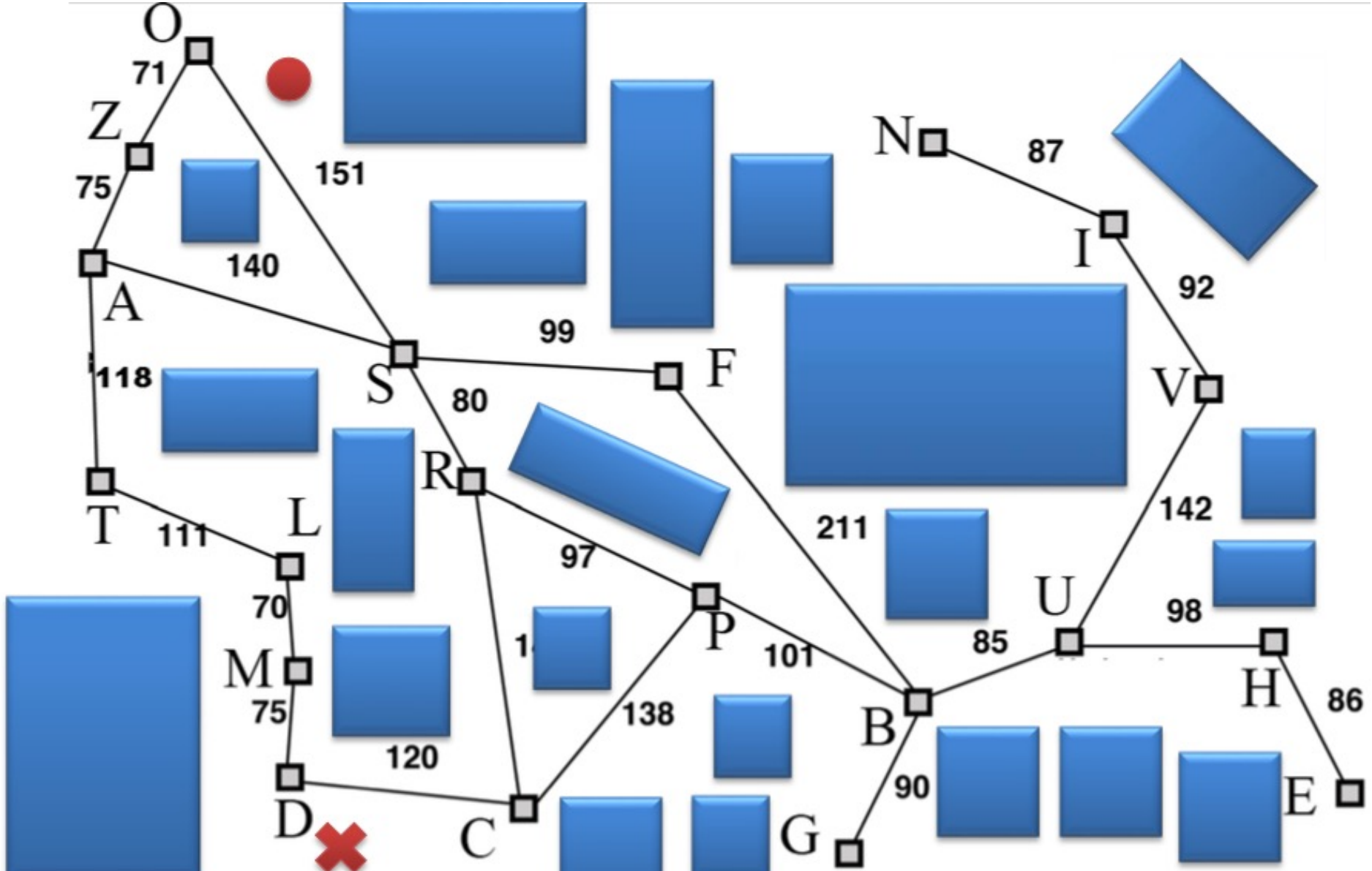
Basic AI steps when told to go to target X

1. Find the closest visible graph node (A)
2. Find the closest visible graph node to X (B)
3. Search for lowest cost path from A to B
4. Move to A
5. Traverse path
6. Move from B to X

Question 2 List current nodes from calculating a path from ● to ✖ with greedy pathing. What is the shortest path overall?
https://forms.gle/tmePfBSEsM7m4Q5C6
https://tinyurl.com/guz-pq3b



Answer: O, S, R, C, D are current nodes for greedy path planning. S->R->C is shortest.



Path network: pros

- Compact representation (at runtime, but must be stored if authored).
- Does not require an agent to be at one of the path nodes at all times (unlike grid).
- Continuous, non-grid movement in local area
- Plays nice with “steering” behaviors
- Can indicate hand-designed, special locations (nodes) and paths (edges).

Path network: cons

- https://youtu.be/e0WqAmuSXEQ
- Getting on and off the network can be awkward
- Path node placement
  - Difficult for complex maps
  - May have invisible spots
- Dynamic pathing doesn't work
  - Doesn't fit well with map-generation features or obscured areas (Fog-of-war) in games

Path Networks vs. Grids?

- When is it more appropriate to use one or the other?
  - Memory (on-disc vs. runtime)
  - Game Design
  - Types of Computation
  - Desired agent behavior
  - Etc.

Summary

- Path Networks probably feel like a weird in-between approach, because they are!

- On Monday we'll talk about Nav Meshes, with all the benefits of path networks and more (everything needed for HW1)

Practice Quiz Time!

This is **practice and will not be graded.**

I can answer questions about practice quiz questions, but please don’t ask a question that gives away an answer.

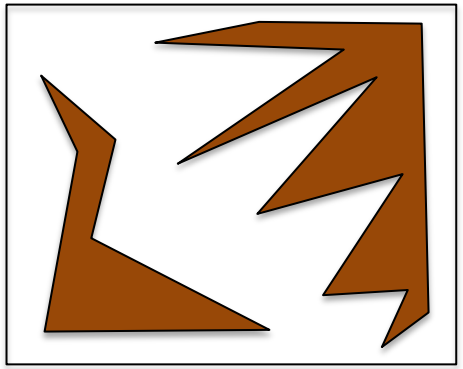
You may use any notes, slides, or videos that you like. But I’d prefer that you not talk to another student.

(PRACTICE) Quiz Link

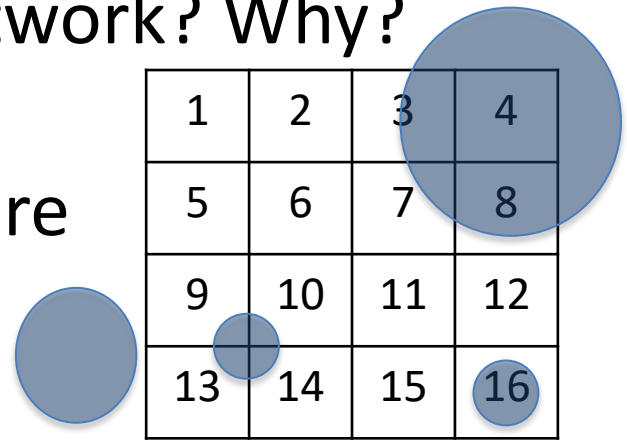
https://forms.gle/Fpcr33ci73eiB7G29

https://tinyurl.com/guz-quiz1

1. Which of the following best defines game AI for this class?
   a) Path finding
   b) NPC behaviours
   c) Non-human game development decisions
   d) Non-stochastic decisions in the game



2. For the environment to the right, would you use a grid or path network? Why?
3. For the grid to the right, which grid cells are valid? (Blue blobs are obstacles).



4. Let's say we have a game with *different grid cell sizes* (some grids are twice as large as other grid cells). What would need to change in terms of generating and using the grid?

1. C
2. Either could work, it depends upon the “Why” answer given.
   - Grids: If this was for a tactics or puzzle game, if there would be many units, or if clunky movement was preferred.
   - Path Network: (Likely better answer) because of the harsh environment making many grid cells invalid, if in the game continuous movement is important.
3. 1,2, 5, 6, 11, 12, and 15
4. (Things in black could be said, things in red are instructions as to what not to say)
   - During generation we would need to check whether a 2x2 cell could fit/be valid, then if a 1x1 cell could fit/be valid.
   - Data representation for storing grids would need to change.
   - The validity check for a 2x2 cell would be unchanged.
   - The neighbors of each grid cell would differ (1x1 would have 4, 2x2 would have 8).
   - Visualization between the cell types would differ.
   - Pathing would differ as some cells would now cost twice as much.