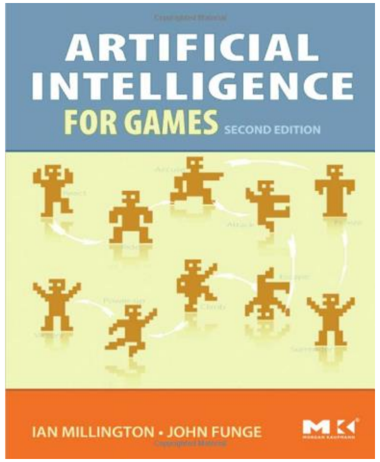
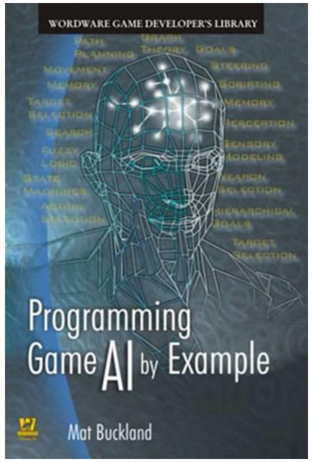
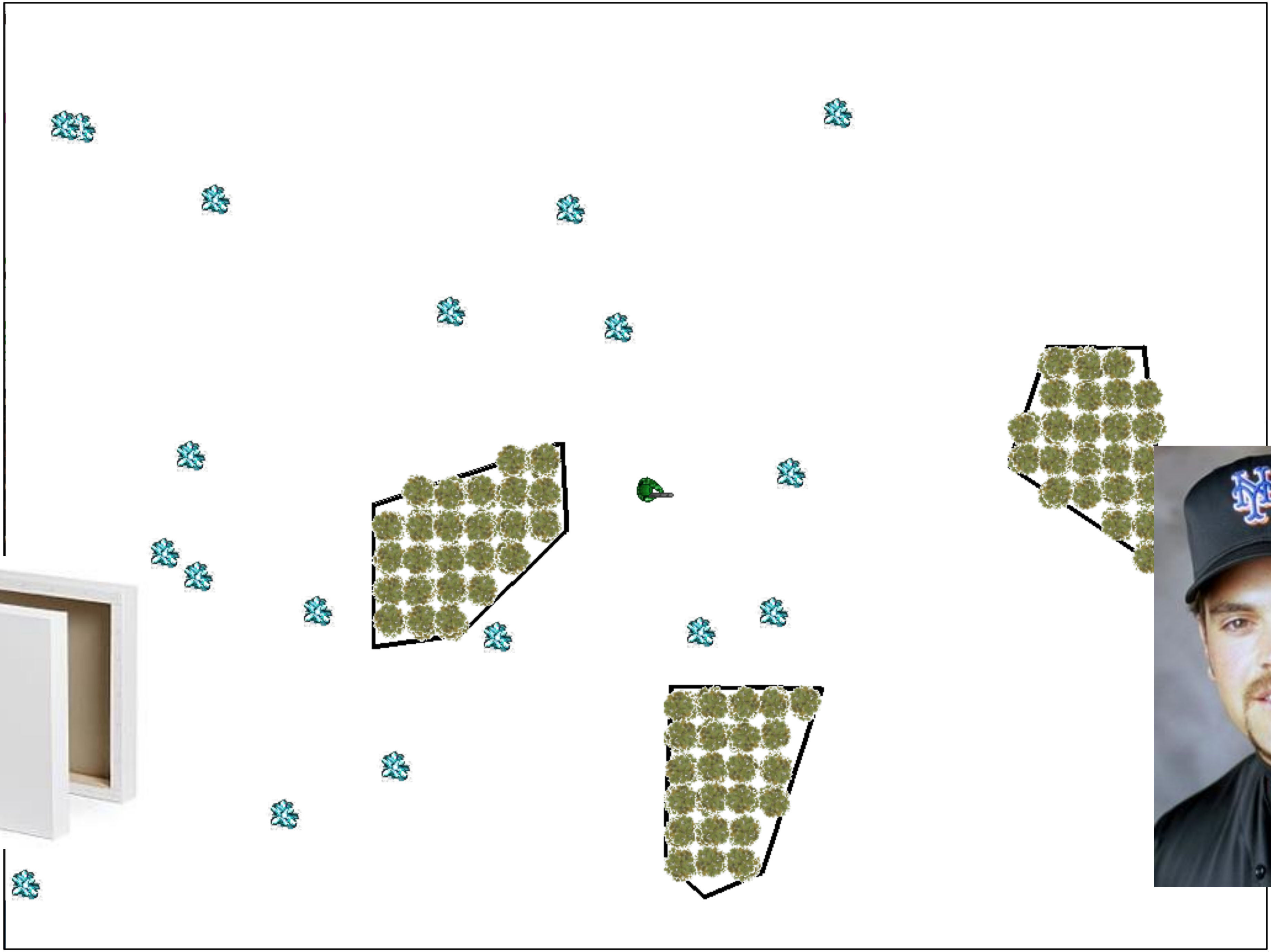
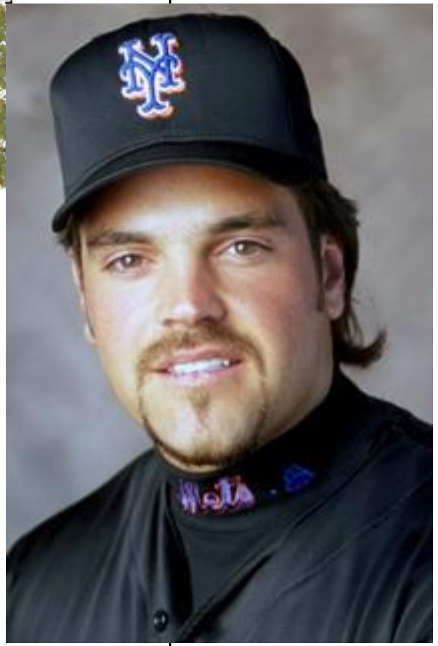




"the question of whether computers can think is like the question of whether submarines can swim" -- Dijkstra

Game AI: The set of algorithms, representations, tools, and tricks that support the creation and management of real-time digital experiences





PREVIOUSLY ON…

What is AI?

Goals of AI

Systems that **think** like humans	Systems that **think** rationally
Systems that **act** like humans	Systems that **act** rationally

Artificial Intelligence

- Getting a computer to do something that a “reasonable person” would think requires intelligence
  - Complexity fallacy
    - Simple can be good, Complex can be bad.
  - Illusion of complexity & Illusion of intelligence: Behaviorism
    - M&F: “We are not interested in the nature of reality or mind; we want characters that look right. In most cases, this means starting from human behaviors and trying to work out the easiest way to implement them in software.”

- Automation

Why AI in games?

Automation—because you need other people to do things, but don’t always have those people

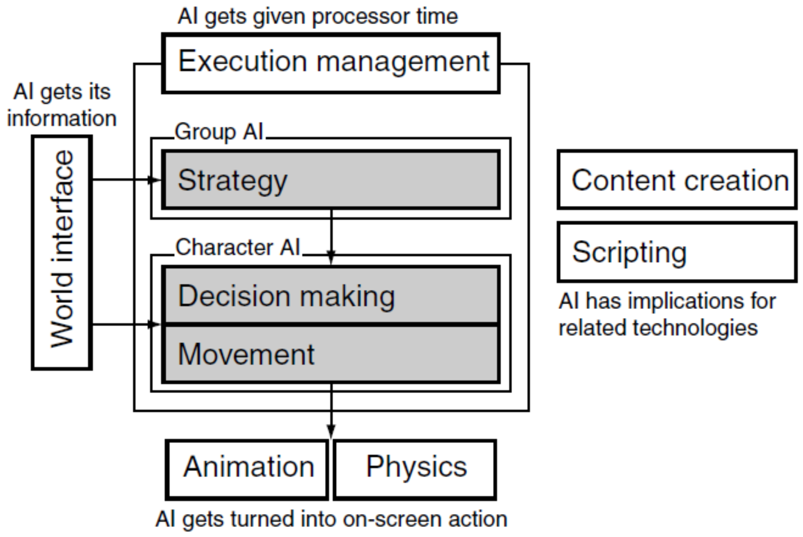
- Opponents
- Companions
- NPCs (shopkeepers, farmers, villains)
- Dungeon master?
- Plot writer?
- Game designer?

What this class is about

- AI **for** games
  - Ways in which AI can—and is used to—enhance game play experiences
  - **Game dev industry: Set of algorithms, representations, tools, tricks, and techniques that support the creation and management of real-time digital experiences**
  - Goals include:
    - enhancing the player's engagement, enjoyment, and experience
      - End behavior is the target
      - Do better than random
    - doing things the player or designer cannot do or don't want to do
      - replace real people when they are unwilling or unavailable to play
      - aid for designers and developers
    - making the entities, opponents, agents, companions, etc. in games appear intelligent
    - believable characters / looking convincing

Major ways GameAI is used…

- In game
  - Movement
  - Decision making
  - Strategy
  - Tailoring/adapting to player individual differences
  - Drama Management
- Out of game
  - PCG
  - Quality control / testing



M&F Fig 1.1

Why AI is important for games

- Why is it essential to the modeled world?
  - NPC’s of all types: opponents, helpers, extras, …
- How can it hurt?
  - Unrealistic characters → reduced immersion
  - Stupid, lame behaviors → reduced fun
  - Superhuman behaviors → reduced fun
- Until recently, given short shrift by developers. Why?
  - Graphics ate almost all the resources
  - Can’t start developing the AI until the modeled world was ready to run
    - AI development always late in development cycle
- Situation rapidly changing / changed. How?
  - AI now viewed as helpful in selling the product
  - Still one of the key constraints on game design

Credit: Dr. Ken Forbus

What this class is about

- “Game AI is (a fundamental part of) game design”
  - How a game design can be brought into existence through the application of algorithms that are often thought of as intelligent
  - Pac-man?

- Not a substitute for an Intro to AI course
- Not going to teach good game design

At its core

- What is it for?
  - Suspension of disbelief (aka “Magic Circle”)
    - making the entities/opponents/agents/companions/etc. in games appear intelligent
  - Easing the cost of development
  - Tailoring/adapting to player individual differences
  - (Data-driven insights)

What this class is NOT about

- AI **in** games
  - John Laird and Michael van Lent (2000): Games are perfect test-beds for "human level" AI
  - AI should play games as if human
    - Vision
    - Decision making in real-time
    - Handling uncertainty
    - Learning
    - Opponent modeling
  - Demonstrated with an AI agent that played Quake
- https://en.wikipedia.org/wiki/Artificial_general_intelligence
  - https://openai.com/five/

How/Why distinct from “academic AI”

- Supporting the player experience
- Good game AI == matching right behaviors to right algorithms
- Product is the target, not clever coding – ends justify means. FUN
- Illusion of intelligence
- “Magic Circle” (Rules of play: game design fundamentals)
- Elegance in simplicity & the complexity fallacy
- Quality control & resource limits
- Fun vs smart: goal is not always to beat the player
- Optimal/rational is rarely the right thing to do

Common (game) “AI” Tricks?

- Move before firing – no cheap shots
- Be visible
- Have horrible aim (being Rambo is fun)
- Miss the first time
- Warn the player
- Attack “kung fu” style (Fist of Fury; BL vs School)
- Tell the player what you are doing (especially companions)
- React to own mistakes
- Pull back at the last minute
- Intentional vulnerabilities or predictable patterns

Liden, “Artificial Stupidity: The Art of Intentional Mistakes”. *AI Game Programming Wisdom*.

Common Heuristics

- Most Constrained / Most Difficult
  - Given current state make choice / select action least available.
  - Aka: Do the most difficult thing first
    - E.g. Attacking enemy with special weapon
    - E.g. Assigning large characters to squads first
- Try the most promising thing first
  - Give each option a rough score and attempt in decreasing order

Intelligent vs. random

Graphs, Search, & Path Planning

2019-08-21

Graphs

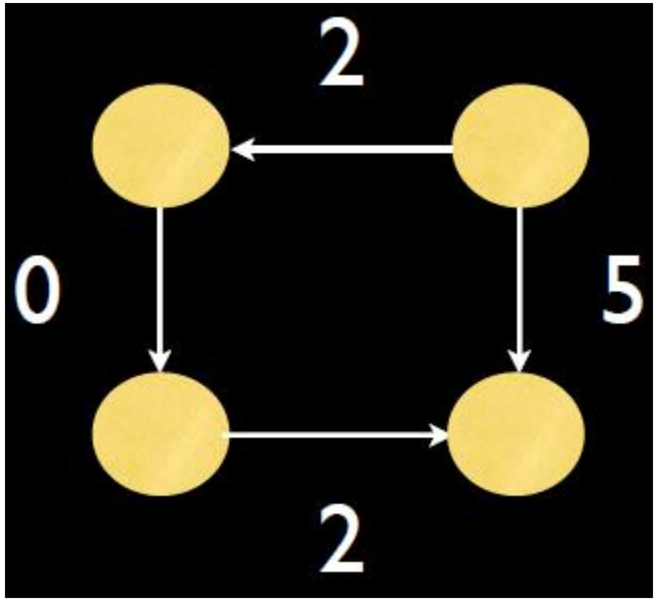
- What is a graph?
- What defines a graph?
- How can we represent them?
- How does representation effect search?

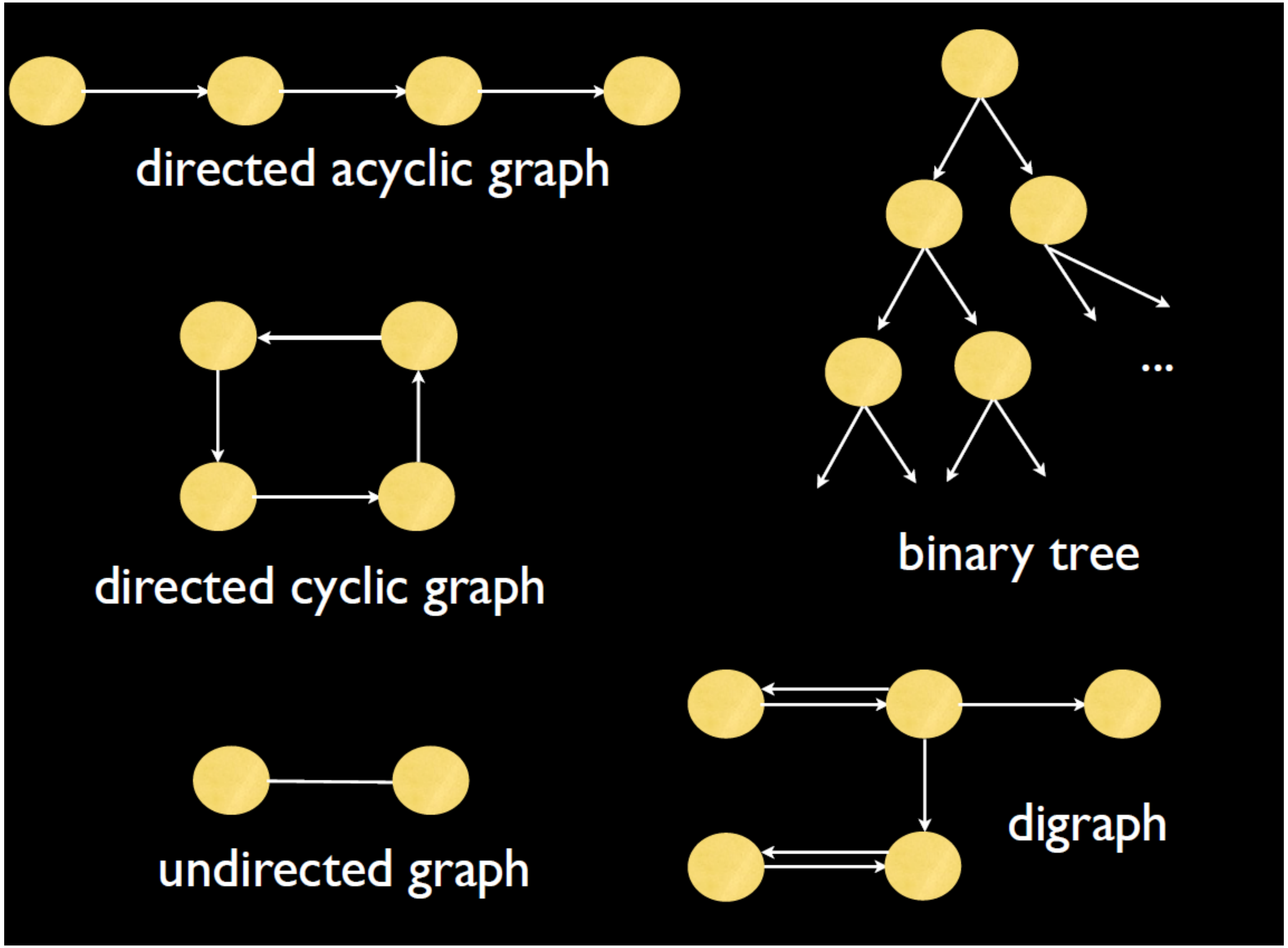
- **Applications to GAI?**

- See Buckland CH 5 for a refresher

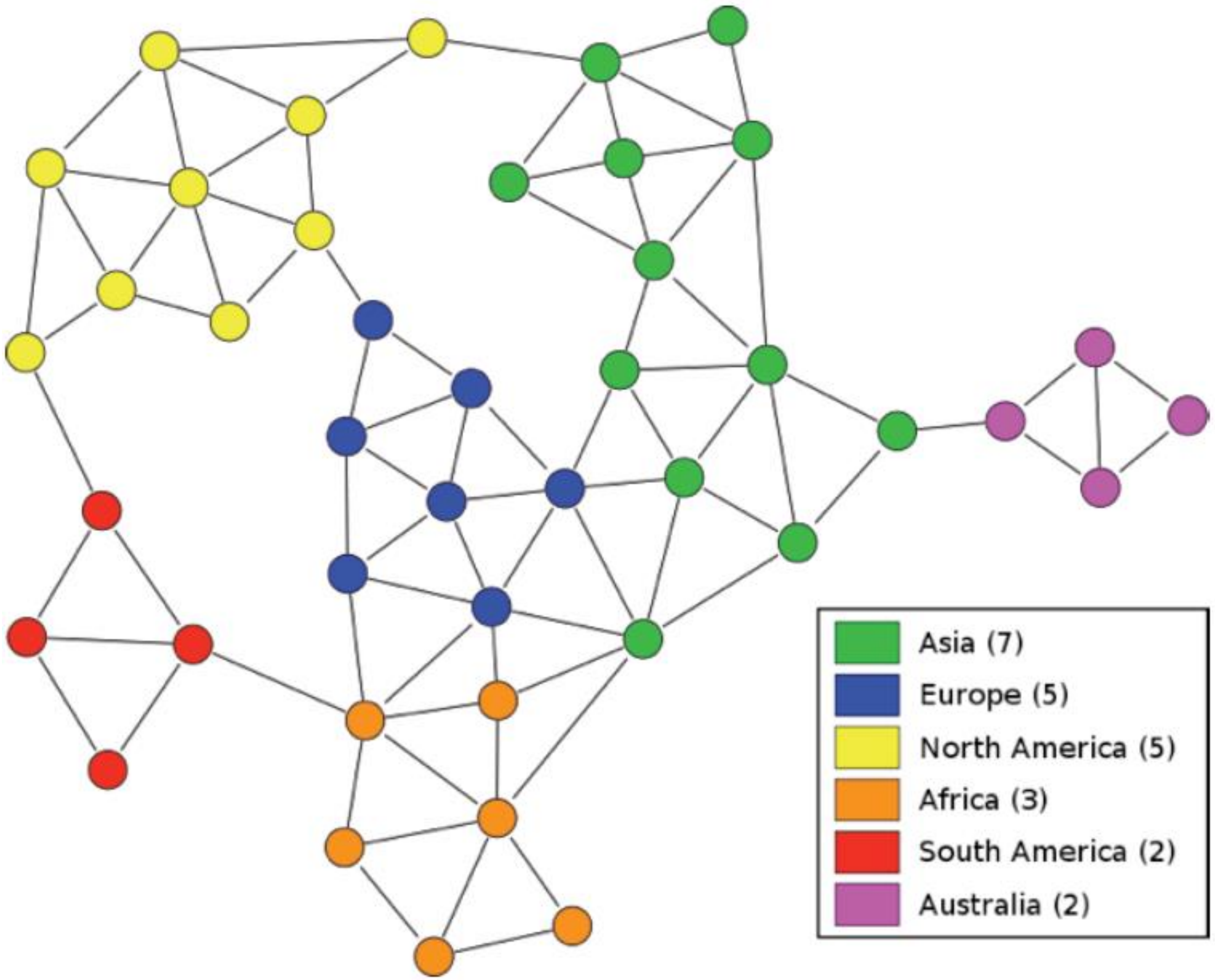
Graphs (2)

- G = {N,E}, N: Nodes, E: Edges (with cost)

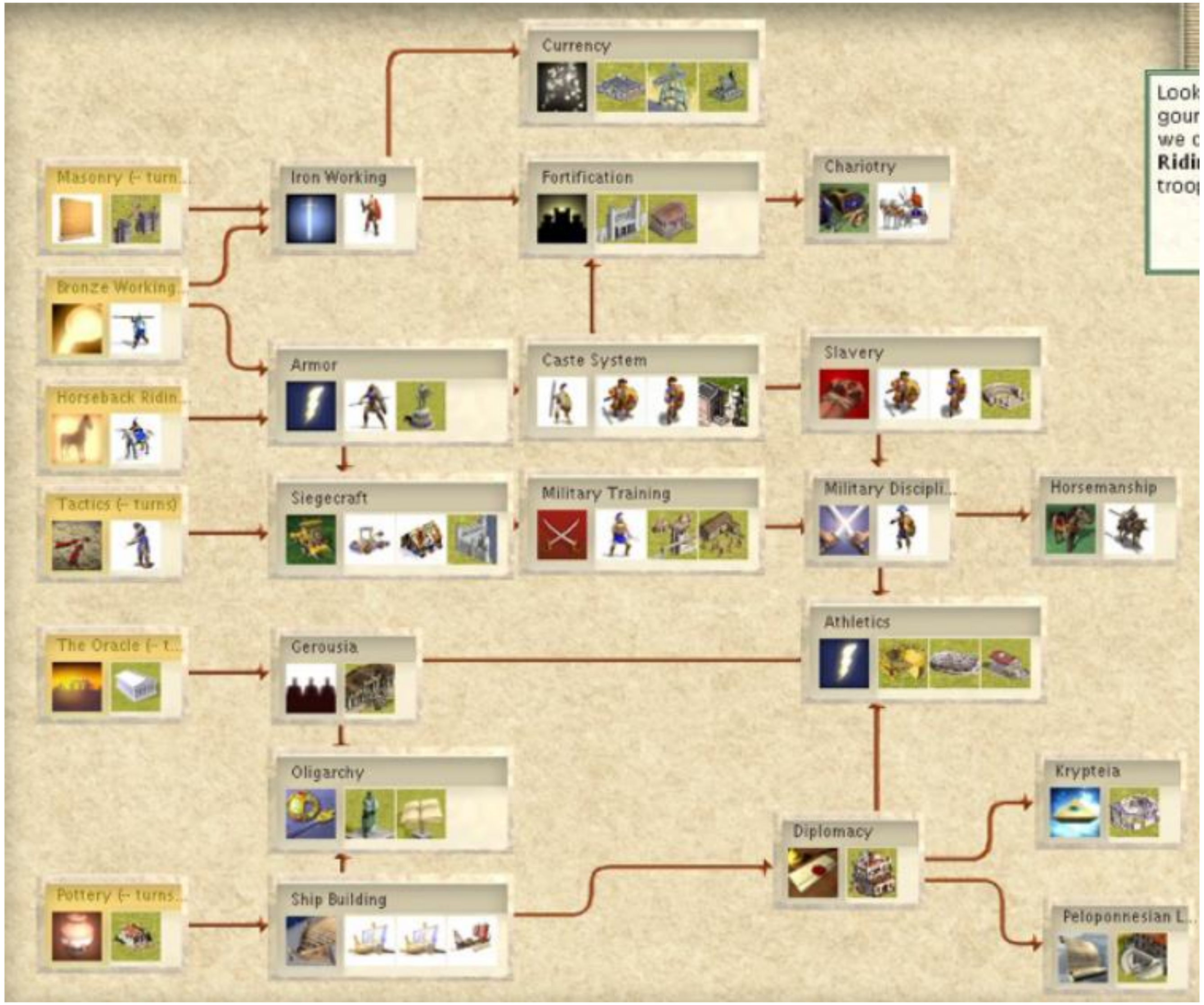



directed acyclic graph
binary tree
...
directed cyclic graph
undirected graph
digraph

Risk


Asia (7)
Europe (5)
North America (5)
Africa (3)
South America (2)
Australia (2)

RTS Dependency Tree

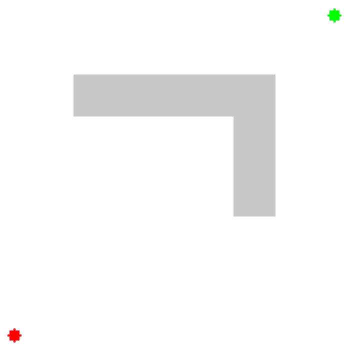


Graphs: Killer App in GAI

- Navigation / Pathfinding
- Navgraph: abstraction of all locations and their connections
- Cost / weight can represent terrain features (water, mud, hill), stealth (sound to traverse), etc
- What to do when …
  - Map features move
  - Map is continuous, or 100K+ nodes?
  - 3D spaces?

Graph Search

- Uninformed (all nodes are same)
  - DFS (stack – lifo), BFS (queue – fifo)
  - Iterative-deepening (Depth-limited)
- Informed (pick order of node expansion)
  - Dijkstra – guarantee shortest path ($E\log_2 N$)
  - A* (IDA*)…. Dijkstra + heuristic
  - D*
- Hierarchical can help



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

What problem are all of these approaches solving?

or

What differentiates these approaches?

Sidebar: Iterative Deepening

- mid 1970s
- Idea: perform depth-limited DFS repeatedly, with an increasing depth limit, until a solution is found.
- Each repetition of depth-limited DFS needlessly duplicates all prior work?
  - Duplication is not significant because a branching factor $b > 1$ implies that the number of nodes at depth k exactly is much greater than the total number of nodes at all depths $k-1$ and less.
- That is: most nodes are in the bottom level.

Number of nodes

- Full, complete, balanced binary tree, height *h*
  - Total number of nodes *N* = 2^{h+1} – 1
    - Leaf nodes at height 0: 2^0 = 1
    - Leaf nodes at height 1: 2^1 = 2
    - Leaf nodes at height 2: 2^2 = 4
    - Leaf nodes at height 3: 2^3 = 8
    - N = 1 + 2 + 2^2 + 2^3 + … + 2^h
      = (2^{h+1} – 1) / (2 – 1) = 2^{h+1} - 1
  - Number of leaves *L* = 2^h
  - Height 42, *N = 8,796,093,022,207*
    L = 4,398,046,511,104

“Informed” Search: Heuristics

- [dictionary]*“A rule of thumb, simplification, or educated guess that reduces or limits the search for solutions in domains that are difficult and poorly understood.”*
- *h(n)* = estimated cost of cheapest path from n to goal (with goal == 0)

Path finding problem solved, right?

- Compilation
  - http://www.youtube.com/watch?v=lw9G-8gL5o0
- Sim City (1, 2 … 5)
  - https://www.youtube.com/watch?v=zHdyzx_ecbQ
- Half-Life 2
  - http://www.youtube.com/watch?v=WzYEZVI46Uw
- Fable III
- DOTA (Defense of the ancients) 1+2
  - https://www.youtube.com/watch?v=p585DHI0qh4
- WoW (World of Warcraft)
- Minecraft Bedrock Edition
  - https://www.youtube.com/watch?v=qR5M5v0XDM0
- Fallout 4
  - https://www.youtube.com/watch?v=M7TicvLXrQo
- DARPA robotics challenge:
  - https://www.youtube.com/watch?v=g0TaYhjpOfo

Path finding models

1. Tile-based graph – “grid navigation”
2. Path Networks / Points of Visibility NavGraph
3. Expanded Geometry
4. NavMesh

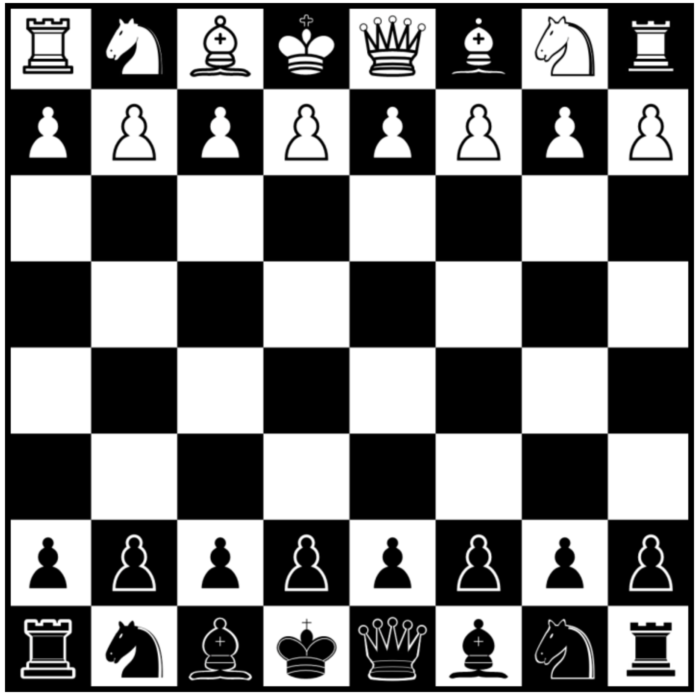
Path finding models

1. **Tile-based graph – "grid navigation"**
   - Simplest topography
   - Assume static obstacles
   - Imaginary latice of cells superimposed over an environment such that an agent can be in one cell at a time.
   - Moving in a grid is relatively straightforward: from any cell, an agent can traverse to any of its four (or eight) neighboring cells
2. Path Networks / Points of Visibility NavGraph
3. Expanded Geometry
4. NavMesh

Model 1: Grid Navigation

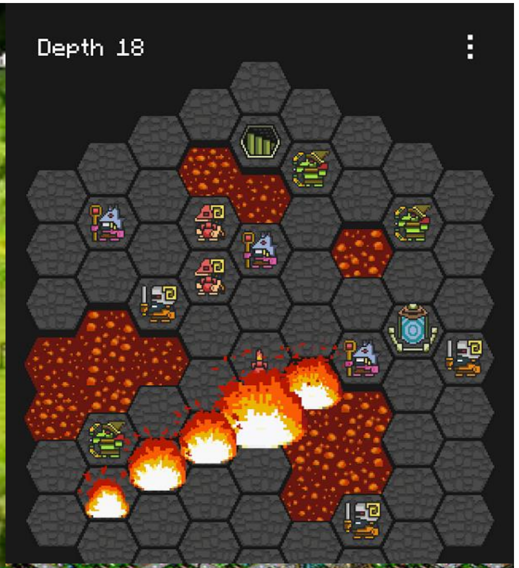
- 2D tile representation mapped to floor/level
  - Squares, hex; 8 or 6 neighbors / connectivity
- Mainly RTS games
- One entity/unit per cell
- Each cell can be assigned terrain type
- Bit mask for non-traversable areas
- Navigation: A* (or perhaps greedy), Dijkstra, D*, IDA*
  - http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html

Grids



Also Grids

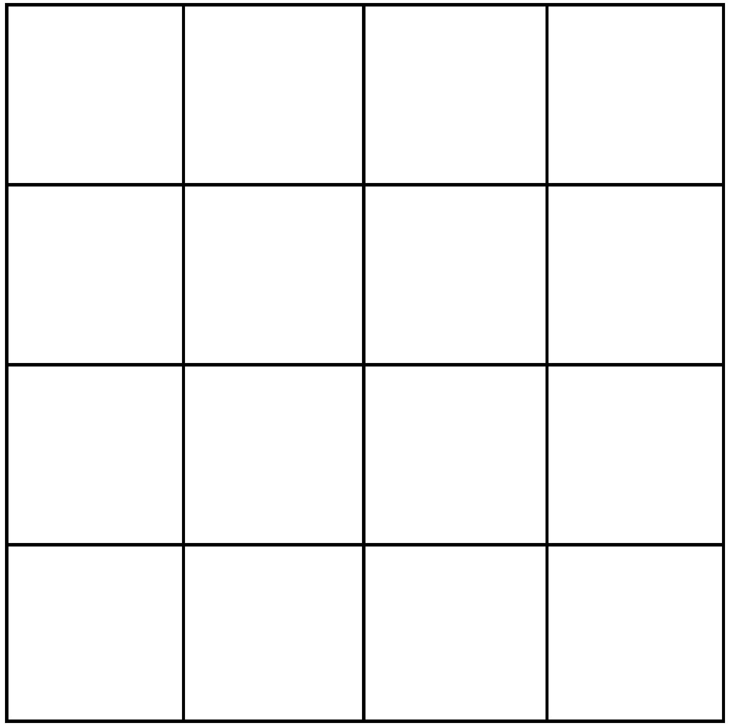

8h
Demkina
37/ 37
43/ 43
0 TO
10/11


Depth 18


Centering Tool
Water Shortage Reported
RCI

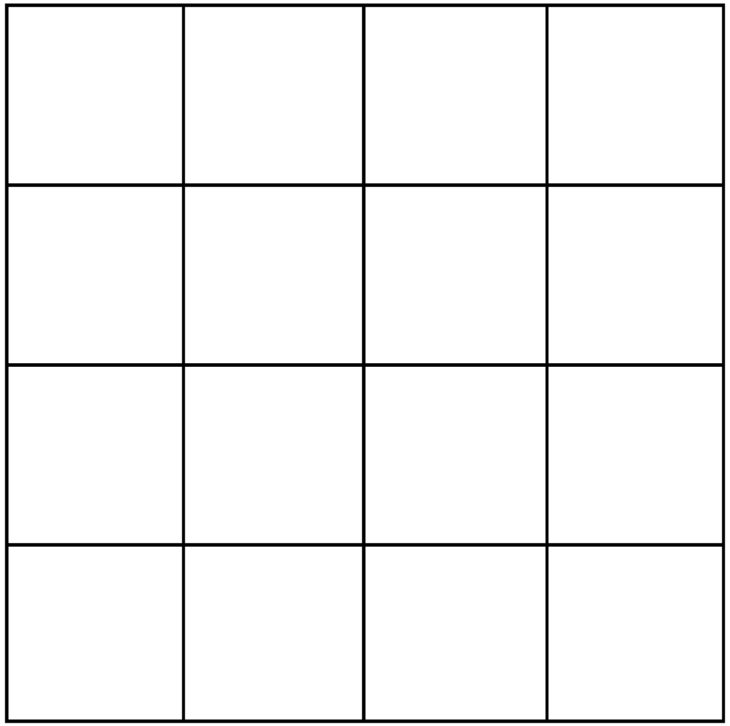
Grids

- 2D tile representation mapped to floor/level
  - Squares/hex cells
  - 8 or 4 neighbors / connectivity
  - Simplify the space
- At most one entity/unit per cell

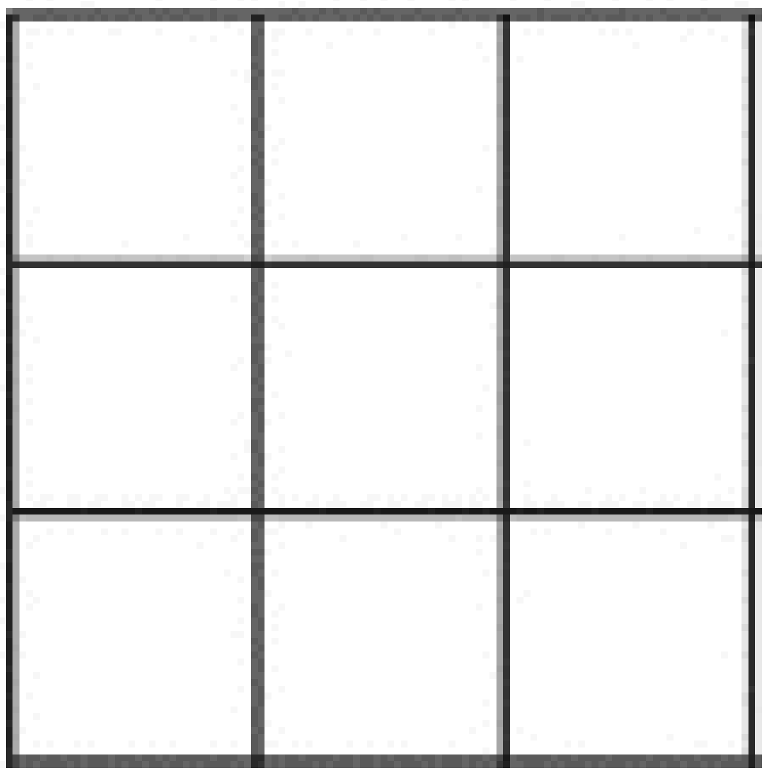


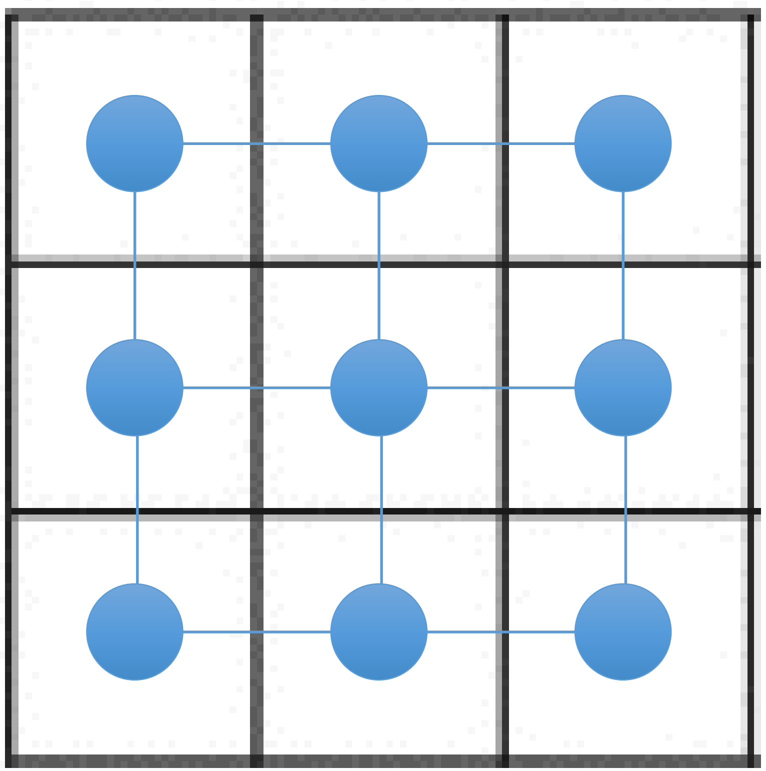
Movement through Grids

- Continuous or Discrete?
- If continuous, we need a path of cells from a current cell to a goal cell
  - Navigate from one cell center to the next



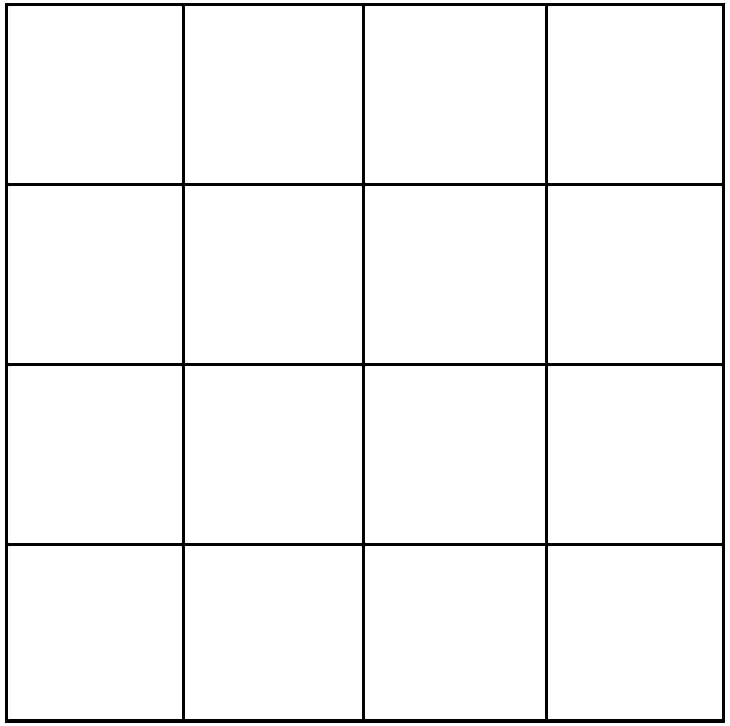
Grid as Graph



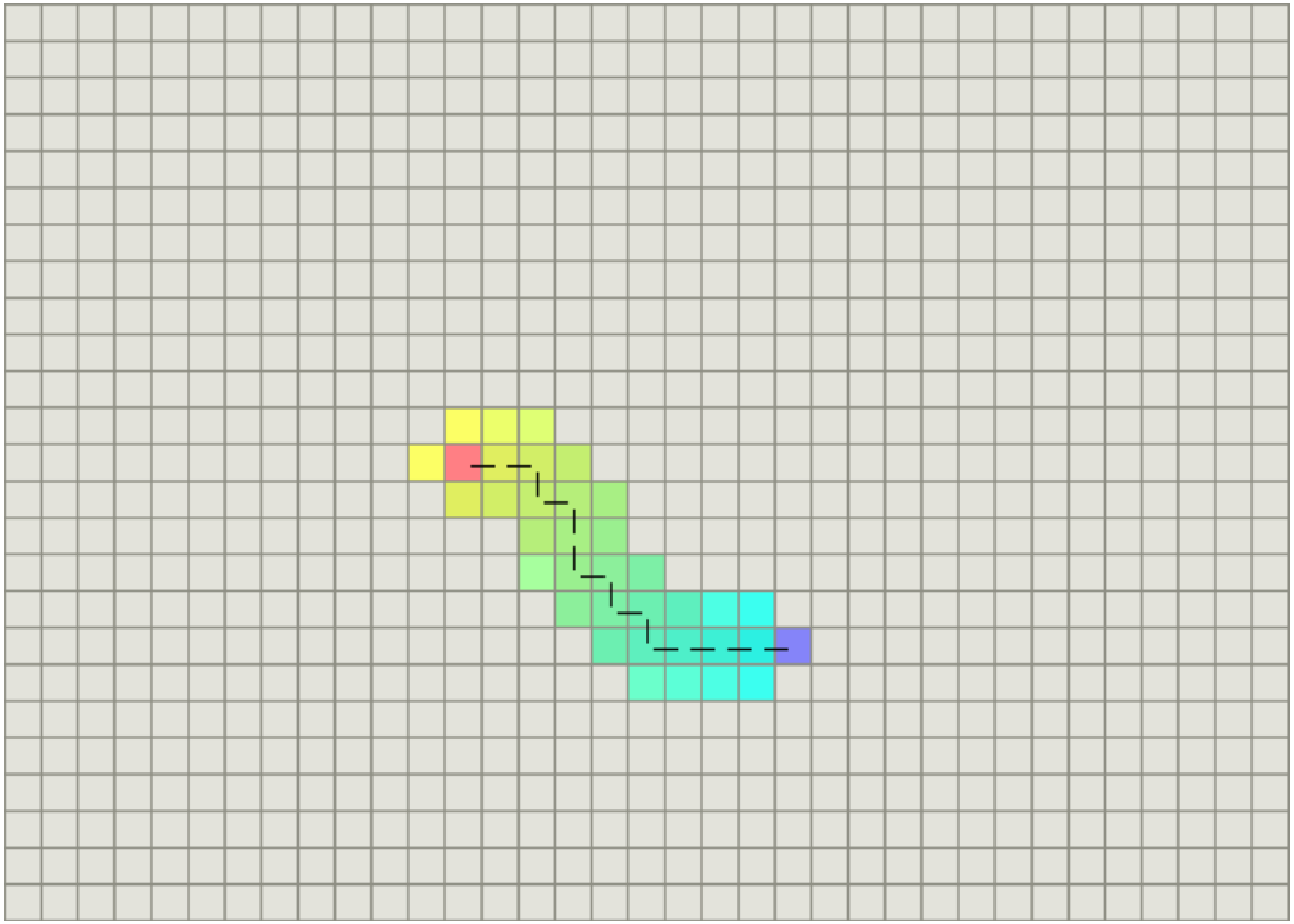


Greedy Path Planning

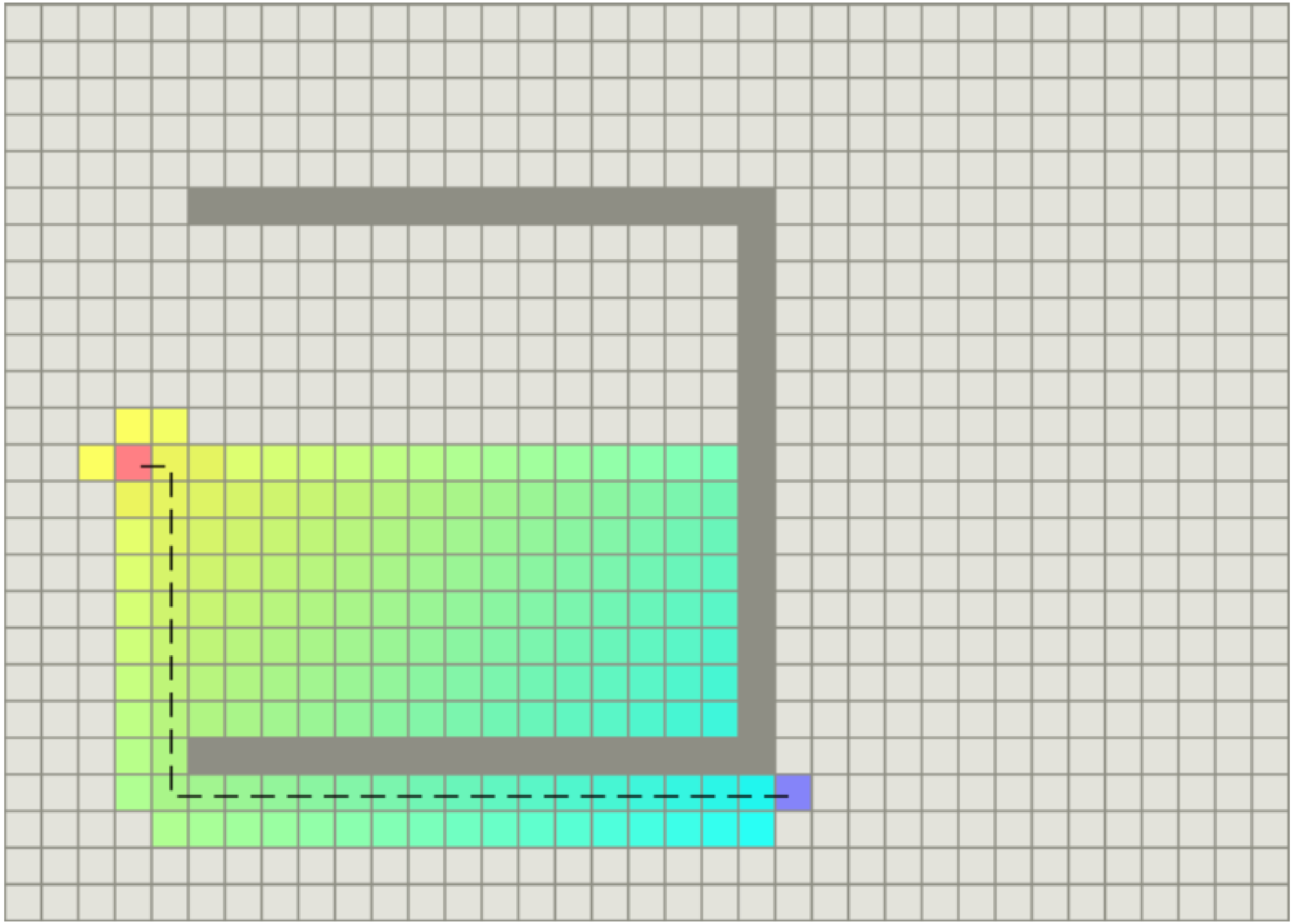
- Given current cell/node, pick the next cell that is closest to the goal cell according to some heuristic
- Once goal cell is reached, backtrack to the initial cell



Grid Path planning can be fast



Grid path planning can be very slow



Path Planner

- Initial state (cell), Goal state (cell)
- Each cell is a state agent can occupy
- Sort successors, try one at a time (backtrack)
- Heuristic: Manhattan or straight-line distance
- Each successor stores who generated it

Question

What are pros and cons of a grid representation of space in terms of character movement?

Grid navigation: pros

- Discrete space is simple
- Can be generated algorithmically at runtime (Hw1)
- Good for large number of units
- A*/greedy search works really well on grids (uniform action cost, not many tricky spots)

Grid navigation: cons

- Discretization “wastes” space
- Agent movement is jagged/awkward/blocky, though can be smoothed
- Some genres need continuous spaces
- Partial-blocking hurts validity
- Search must visit a lot of nodes (cells)
- Search spaces can quickly become huge
  - E.g. 100x10 map == 100k nodes and ~78k edges

New Problems

- Generation
- Validity
- Quantization
  - Converting an in-game position (for yourself or an object) into a graph node
- Localization
  - Convert nodes back into game world locations (for interaction and movement)
- Awkward agent movement
- Long search times