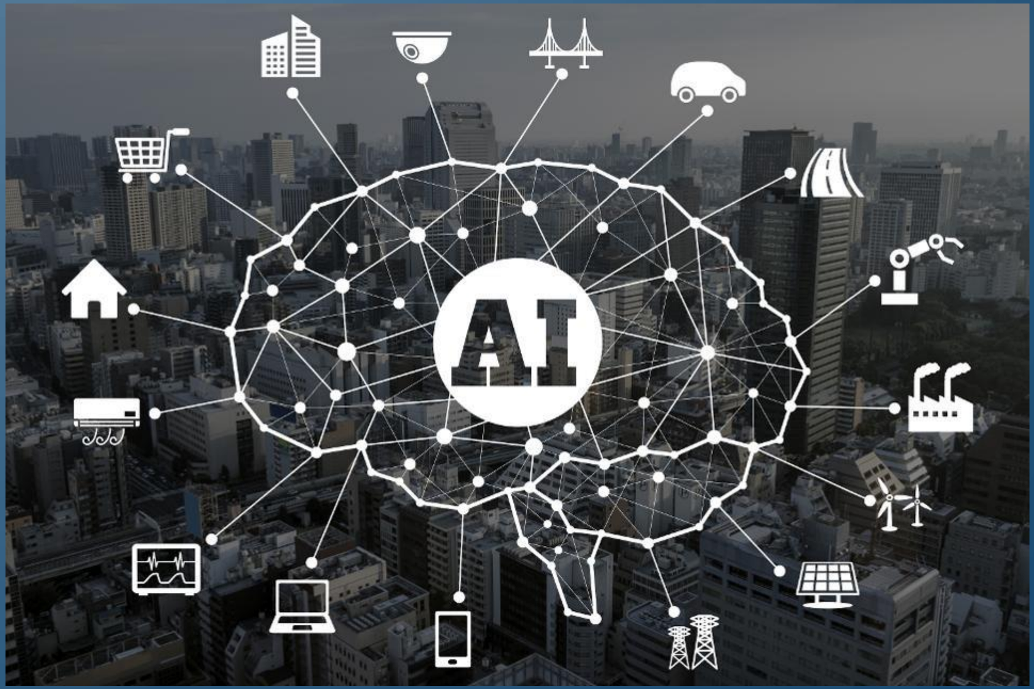
GPAT – CHAPTER 9
ARTIFICIAL INTELLIGENCE

ARTIFICIAL INTELLIGENCE IN GAMES

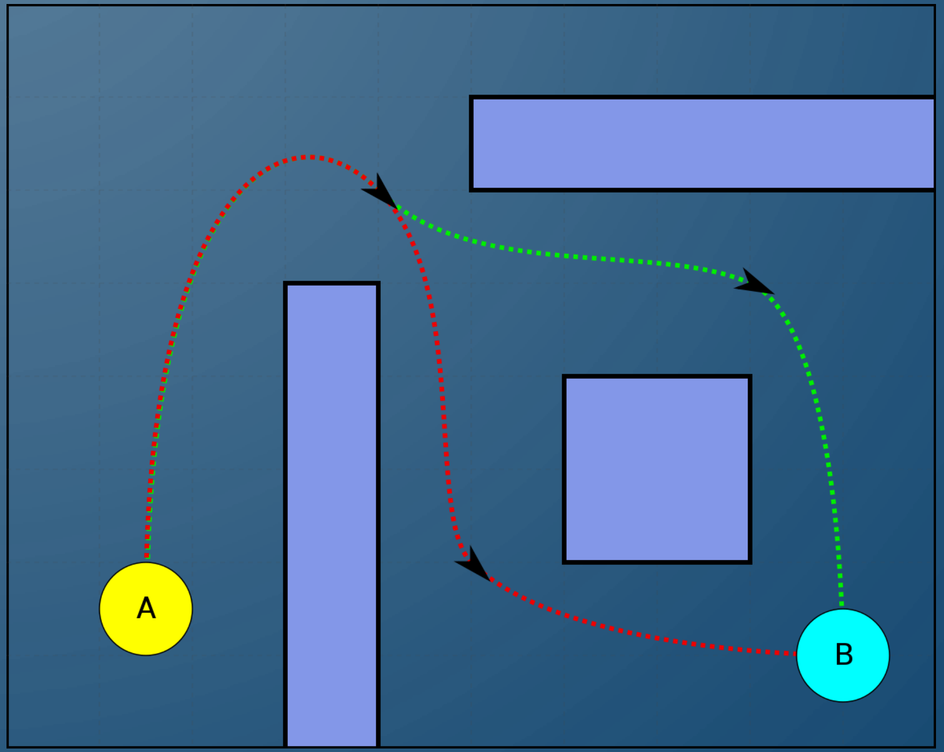
- Artificial Intelligence is a subfield of computer science that attempts to mimic human/animal behavior/intelligence
- Common approaches in AI cannot necessarily be applied to games
  - Need real-time performance
  - Well-defined requirements vs general problems



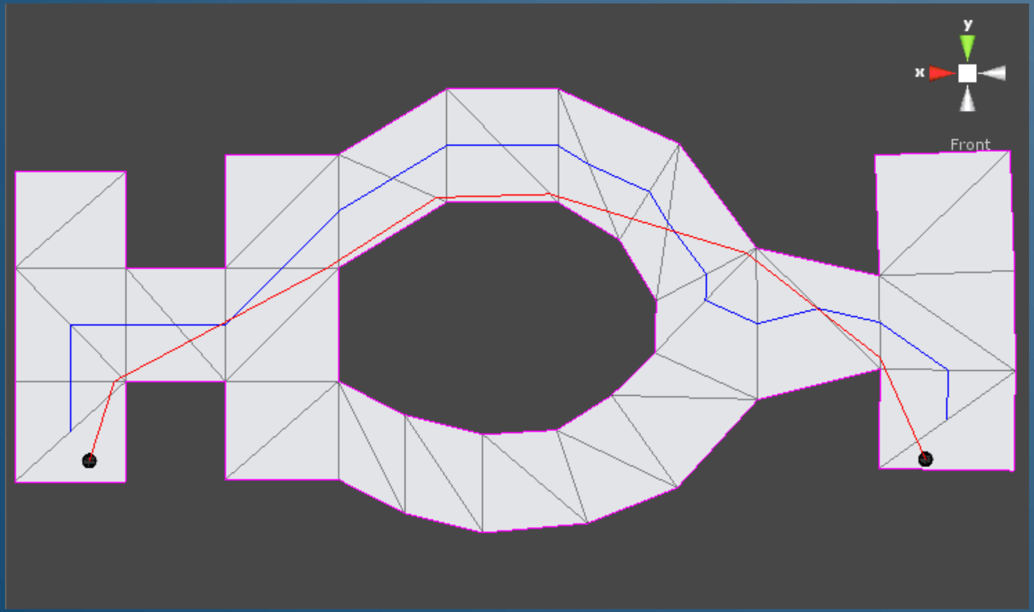
PATHFINDING

PATHFINDING

- Given two points A and B, how do you move intelligently from A to B?
- Sometimes we just want a path, other times we want the best path, etc
- Very complex problem



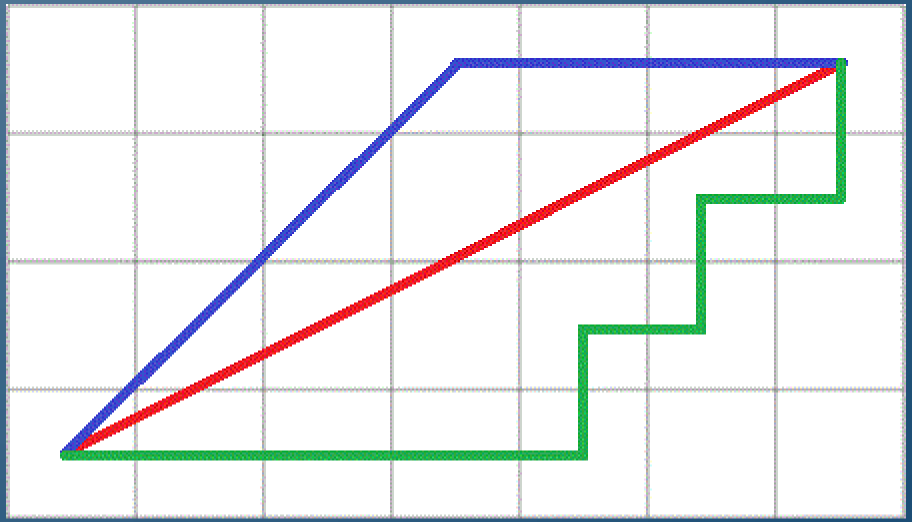
REPRESENTING THE SPACE



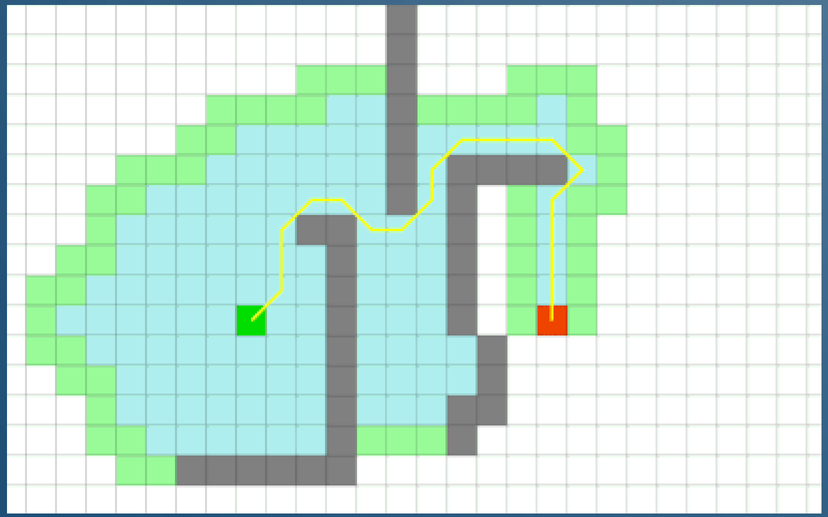
- Always a graph – a set of nodes (things) and edges (relationships between the things)
  - Explicitly, e.g., an adjacency list or a finite grid stored in an array
  - Implicitly, e.g., an infinite grid modeled as a set of equations
- Common graphs
  - Grid of shapes that tessellate in the space (triangles, squares, hexagons)
  - **Path nodes** (artist designed graphs)
  - **Navigation meshes** (geometrically determined)

ADMISSIBLE HEURISTICS

- A **heuristic**, $h(x)$, in path finding will represent an estimated cost from a node to the goal node
- A heuristic is **admissible** if the estimate is always less than or equal to the actual cost
  - Theoretically will guarantee that our algorithm will find the best path
- Examples of admissible heuristics
  - Manhattan distance (city block count) in a planar grid
  - Euclidean distance (straight-line motion)



POSSIBLE SEARCH ALGORITHMS



- Depth-first search – won't find best path
  - Greedy best-first – DFS with priority queue on heuristic
- Breadth-first search – will find path with least amount of edges
  - Dijkstra's algorithm – BFS with priority queue on cost-to-come, and it finds the shortest path
    - A* - Dijksta's algorithm incorporating cost-to-go (heuristic)

A* (A-STAR) ALGORITHM

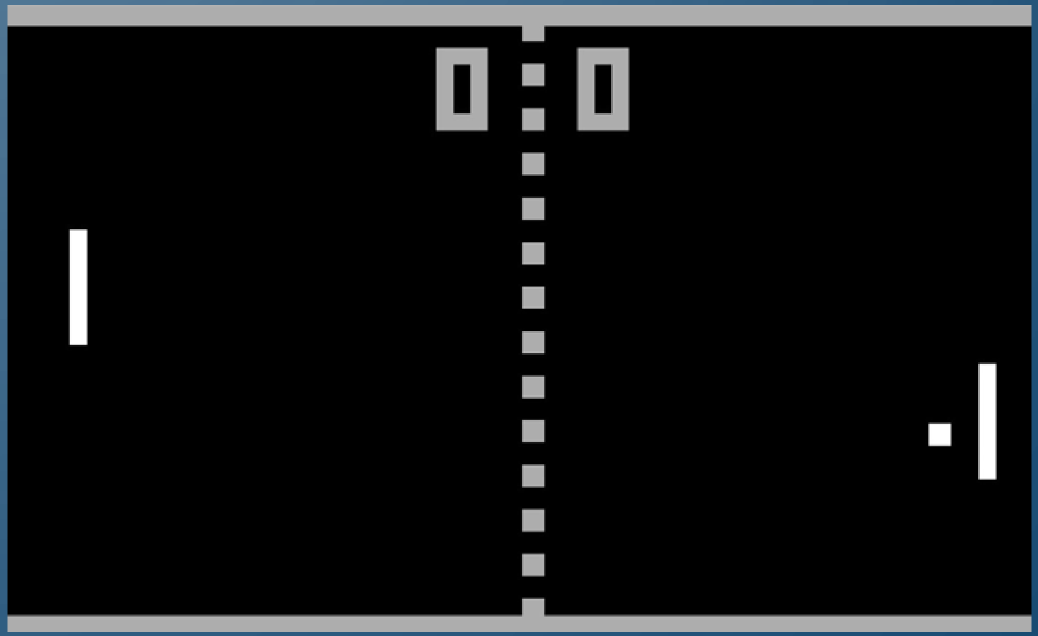
Algorithm A*
Input: Graph G, Nodes $start$ and $goal$
1. Sets $closed \leftarrow \emptyset, open \leftarrow \emptyset$
2. Node $curr \leftarrow start$
3. **repeat**
4. **for each** Node $n \in adjacent(curr)$ **do**
5. **if** $n \in closed$ **then**
6. continue
7. **else if** $n \in open \wedge g(n.curr) < n.g$ **then**
8. $n.parent \leftarrow curr,$ $n.g \leftarrow g(n, curr),$ $n.f \leftarrow n.g + n.h$
9. **else**
10. $n.parent \leftarrow curr,$ $n.g \leftarrow g(n, curr),$ $n.h \leftarrow h(n, goal),$ $n.f \leftarrow n.g + n.h$
11. if $|open| = 0$ then
12. break
13. $curr \leftarrow n \in open$ with smallest f
14. remove $curr$ from $open$ and add to $closed$
15. **until** $curr = goal$

- $g(n, p)$ is cost at n with parent p
- $h(n, g)$ is heuristic cost of n to g
- $f(n) = g(n) + h(n)$
- $closed$ represents visited nodes
- $open$ represents candidates for next visit

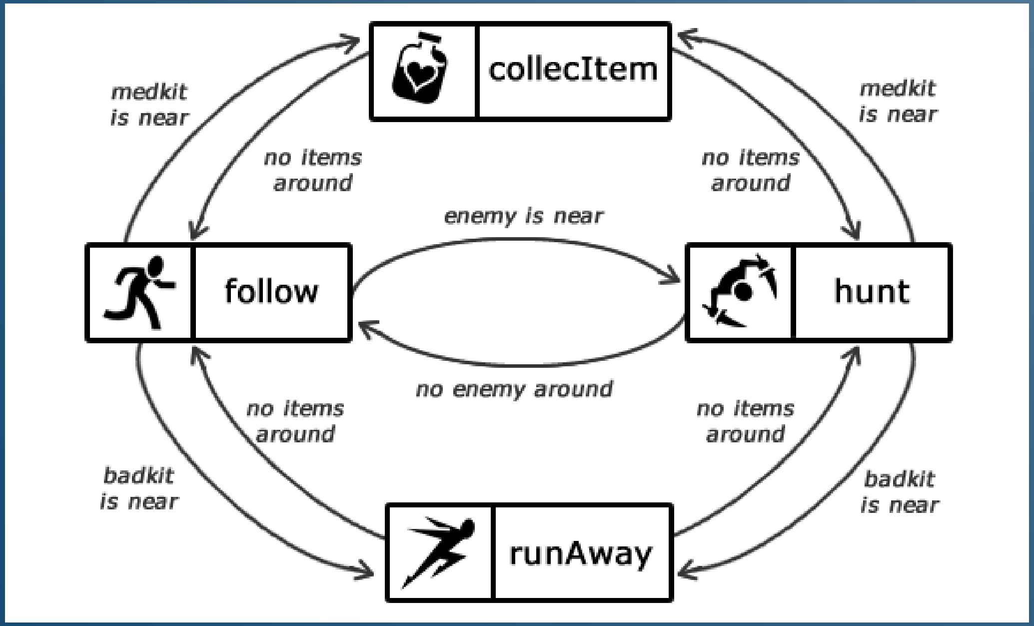
STATE-BASED BEHAVIORS

STATELESS BEHAVIORS

- Some AI can be defined from a simple rule and has no "state"
  - State refers to stored information
- Consider the AI for pong
  - Follow the position of the ball
- A **state-based behavior** behaviors differently (different rules) at different times



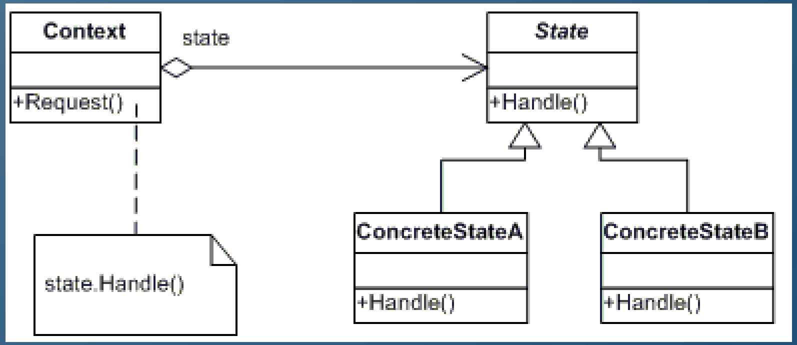
STATE MACHINES



- A state machine contains a set of states (nodes) and conditions for transitions between states (edges)
  - States also would encode actions upon entering/exiting a given state
- Often resembles a complex flow chart (graph)
- Its all a design problem

STATE MACHINE IMPLEMENTATION

- Updates occur in update step of game loop
- Create a polymorphic base class for a state with
  - Update() – update for specific state
  - Enter() – manage entering state
  - Exit() – manage exiting state
- A controller class that stores set of states and rules for transitioning



STRATEGY AND PLANNING

THROUGH THE LOOK OF REAL-TIME STRATEGY GENRE

STRATEGY

- **Strategy** encompasses how an AI should compete (aggressive/defensive)
- **Micro strategy** is per-unit actions implemented through state machines
- **Macro strategy** is overarching strategy (approach to the game)
  - Example would be "rushing"

STRATEGY

- Thought of in terms of goals
  - Example "teching"
  - Example "expanding"
- Prioritization of goals
  - Dynamic weighting of importance of individual goals
- Constructing a plan would create a series of steps to follow to reach a goal

PLANNING

- An algorithm for reaching a goal
  - Example for "expanding"
    - Search for new base
    - Build enough units to defend
    - Send workers and defense to base location
    - Build base
- Possibly implemented through state machine
- Needs to assess feasibility of plan to notify overarching strategy

SUMMARY

- In this chapter, we looked at some basic approaches to artificial intelligence in games
  - Pathfinding
  - State-based behaviors
  - Strategy and planning