CS 348
Intro to Artificial Intelligence

Day 5
Games

(slides based on Downy, Sood, Dan Klein, Pieter Abbeel)

Mike Rubenstein

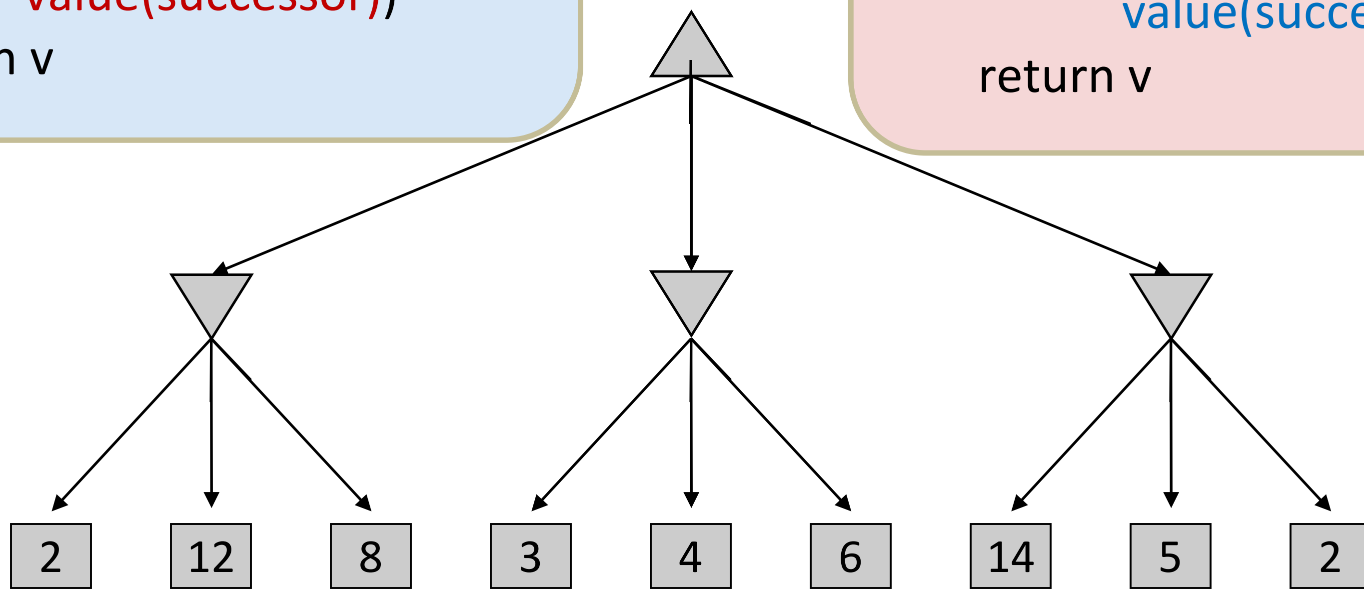
Today:

- Class business
  - Lab 1 Due today, 7pm
- Adversarial search (chapter 5.1 - 5.4)
  - Alpha-beta pruning
  - Expecta-minimax
  - Evaluation functions
- Labs:
  - Intro Lab 3 (A*)
  - Discuss lab 2
  - Answer questions about lab 1

Minimax Example

```
def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, min-value(successor))
    return v
```

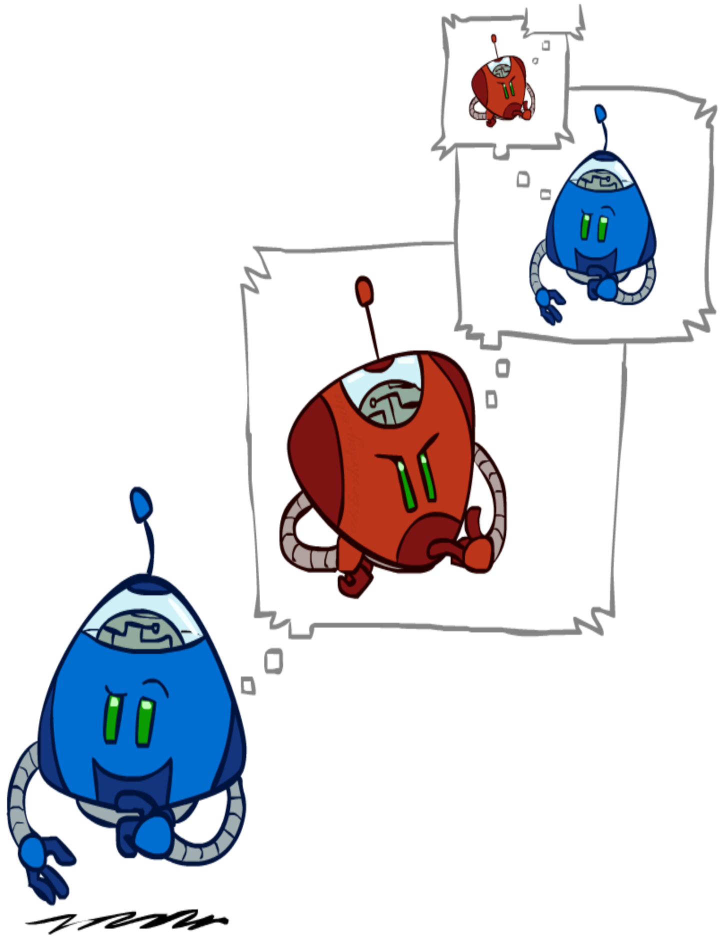
```
def min-value(state):
    initialize v = +∞
    for each successor of state:
        v = min(v, max-value(successor))
    return v
```



Minimax Efficiency

- How efficient is minimax?
  - Just like (exhaustive) DFS
  - Time: $O(b^m)$
  - Space: $O(bm)$

- Example: For chess, $b \approx 35$, $m \approx 100$
  - Exact solution is completely infeasible
  - But, do we need to explore the whole tree?



Alpha-Beta Implementation

α: MAX's best option on path to root

β: MIN's best option on path to root

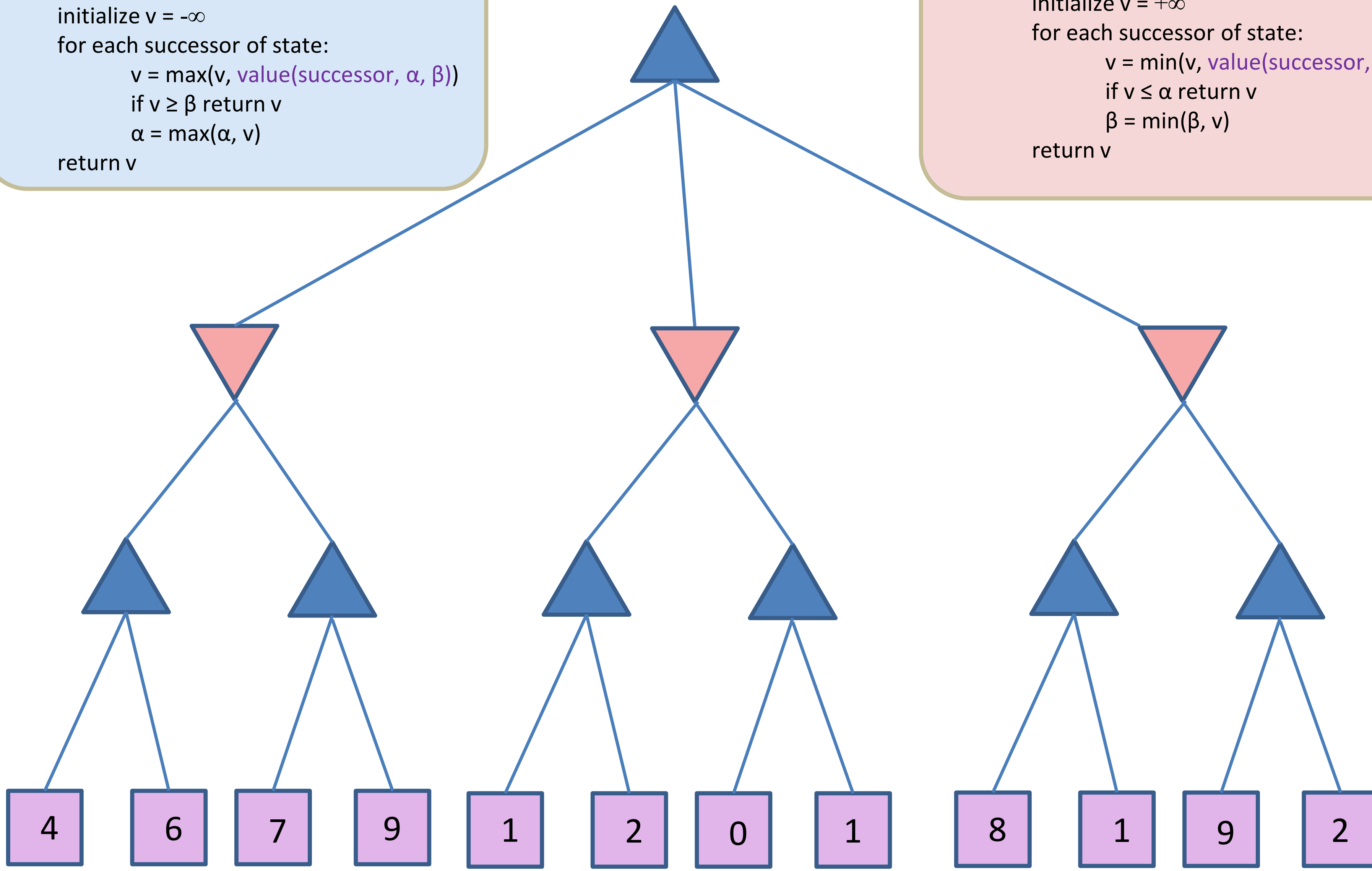
```
def max-value(state, α, β):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor, α, β))
        if v ≥ β return v
        α = max(α, v)
    return v
```

```
def min-value(state , α, β):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor, α, β))
        if v ≤ α return v
        β = min(β, v)
    return v
```

Alpha-beta pruning example

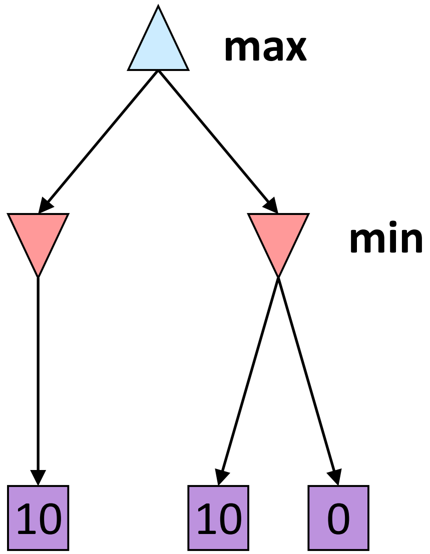
```
def max-value(state, α, β):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor, α, β))
        if v ≥ β return v
        α = max(α, v)
    return v
```

```
def min-value(state , α, β):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor, α, β))
        if v ≤ α return v
        β = min(β, v)
    return v
```

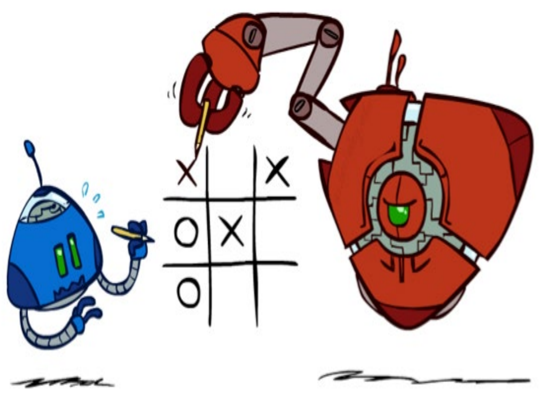


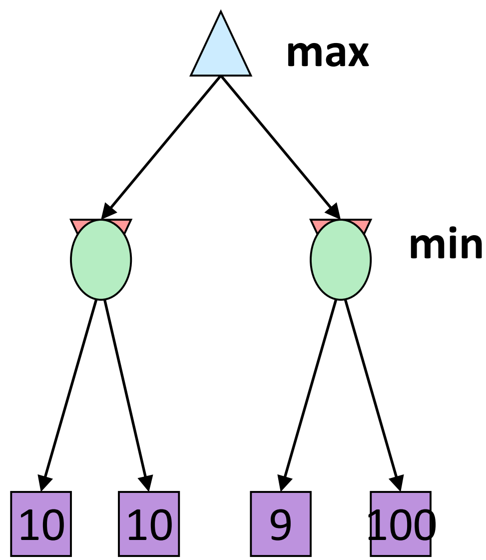
Alpha-Beta Pruning Properties

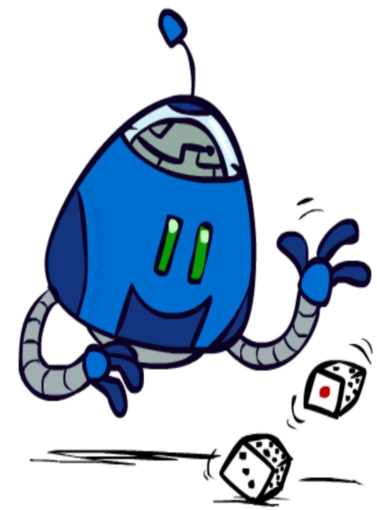
- This pruning has no effect on minimax value computed for the root!
- Values of intermediate nodes might be wrong
  - Important: children of the root may have the wrong value
  - So the most naïve version won't let you do action selection
- Good child ordering improves effectiveness of pruning
- With "perfect ordering":
  - Time complexity drops to $O(b^{m/2})$
  - Doubles solvable depth!
- With random ordering
  - Time complexity is $O(b^{3m/4})$
- This is a simple example of metareasoning (computing about what to compute)



Worst-Case vs. Average Case



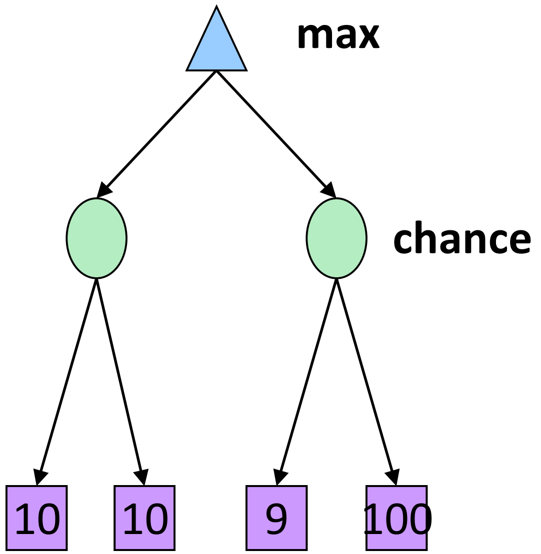




Idea: Uncertain outcomes controlled by chance, not an adversary!

Expectimax Search

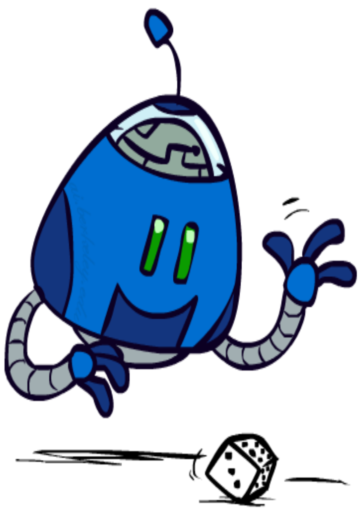
- Why wouldn't we know what the result of an action will be?
  - Explicit randomness: rolling dice
  - Unpredictable opponents: the pacman ghosts respond randomly
  - Actions can fail: when moving a robot, wheels might slip

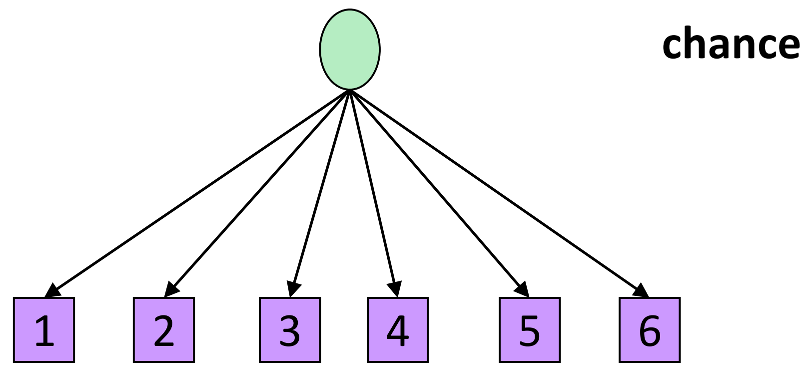


- Values should now reflect average-case (expectimax) outcomes, not worst-case (minimax) outcomes

- Expectimax search: compute the average score under optimal play
  - Max nodes as in minimax search
  - Chance nodes are like min nodes but the outcome is uncertain
  - Calculate their expected utilities
  - I.e. take weighted average (expectation) of children

Expectimax Search





Expectimax Pseudocode

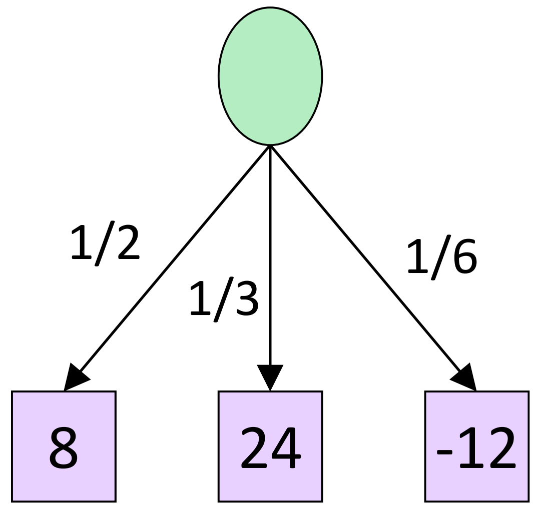
```
def value(state):
    if the state is a terminal state: return the state's utility
    if the next agent is MAX: return max-value(state)
    if the next agent is EXP: return exp-value(state)
```

```
def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor))
    return v
```

```
def exp-value(state):
    initialize v = 0
    for each successor of state:
        p = probability(successor)
        v += p * value(successor)
    return v
```

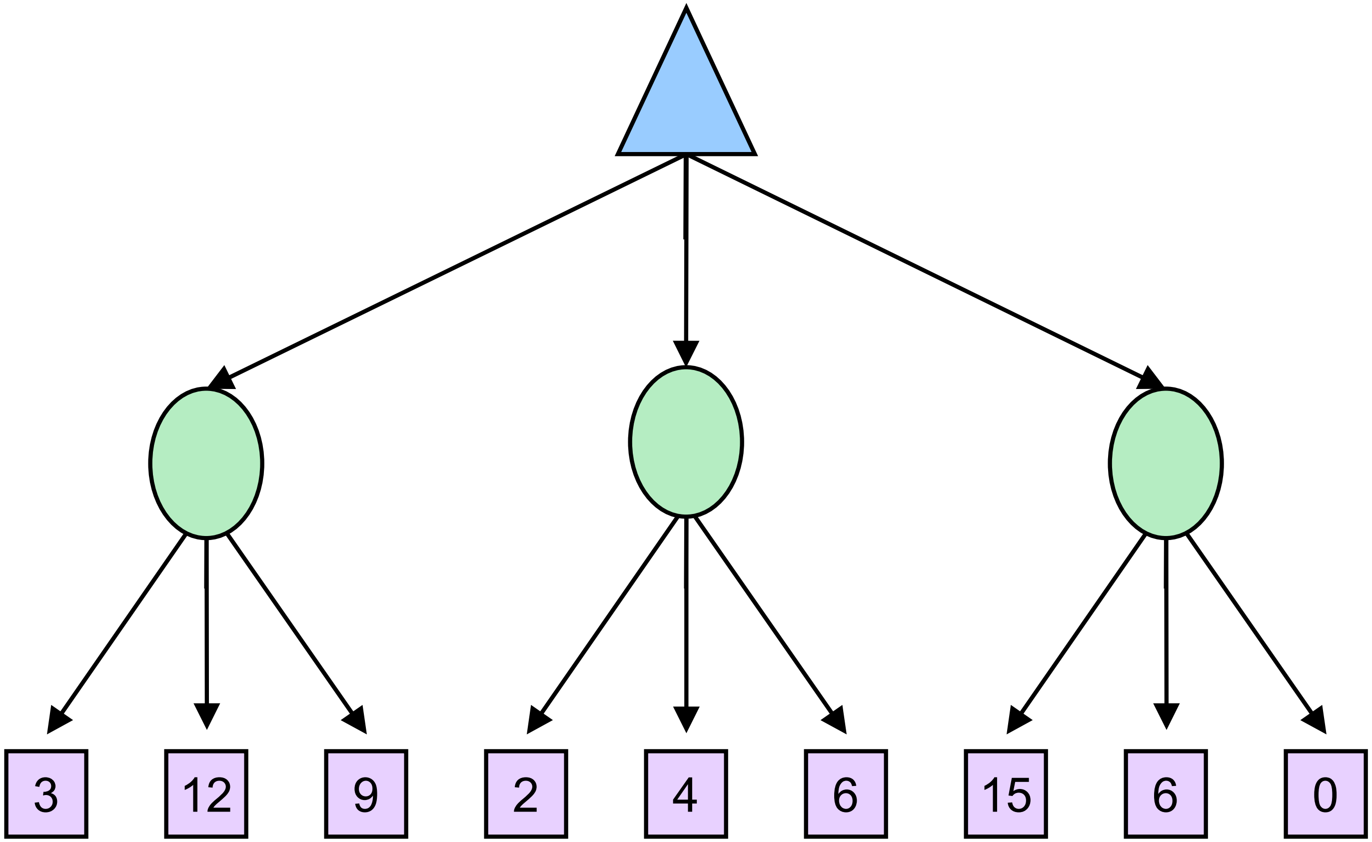
Expectimax Pseudocode

```
def exp-value(state):
    initialize v = 0
    for each successor of state:
        p = probability(successor)
        v += p * value(successor)
    return v
```

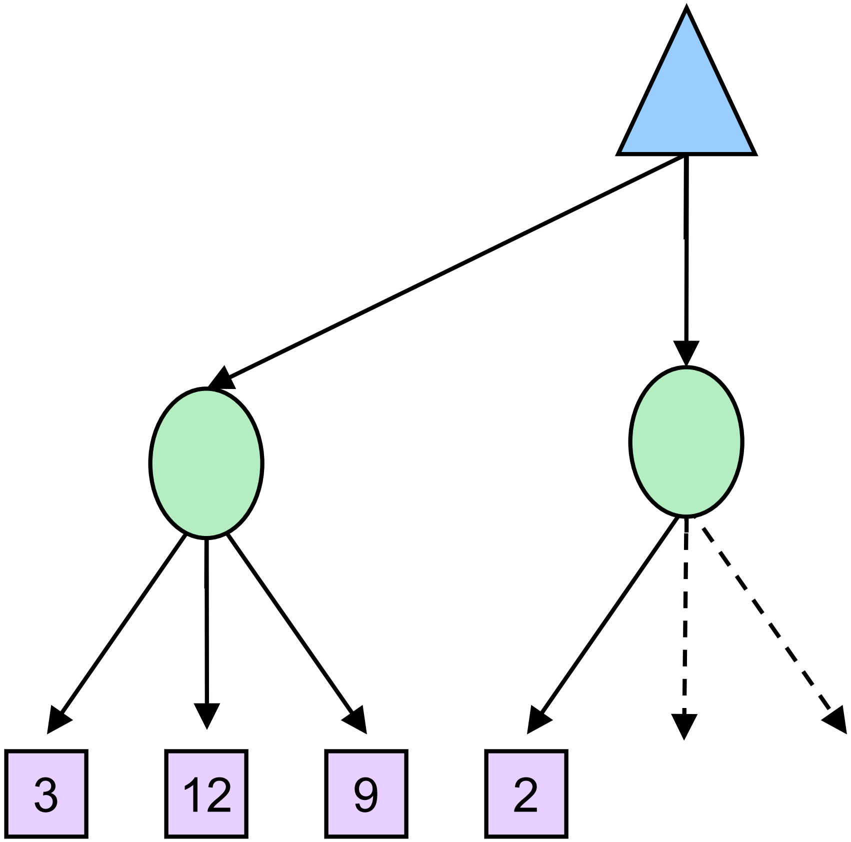


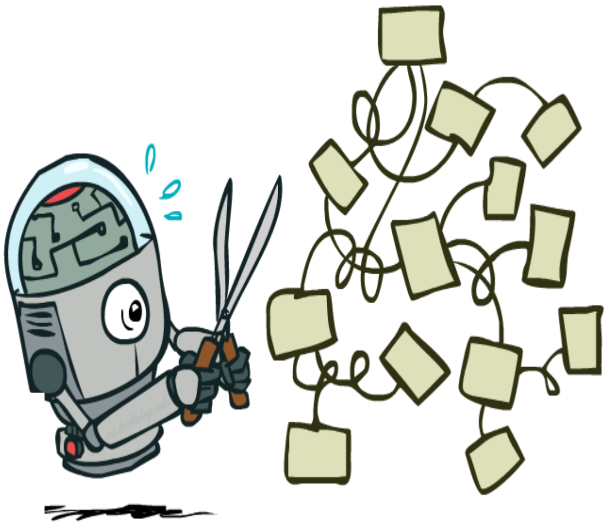
$v = (1/2)\ (8) + (1/3)\ (24) + (1/6)\ (-12) = 10$

Expectimax Example

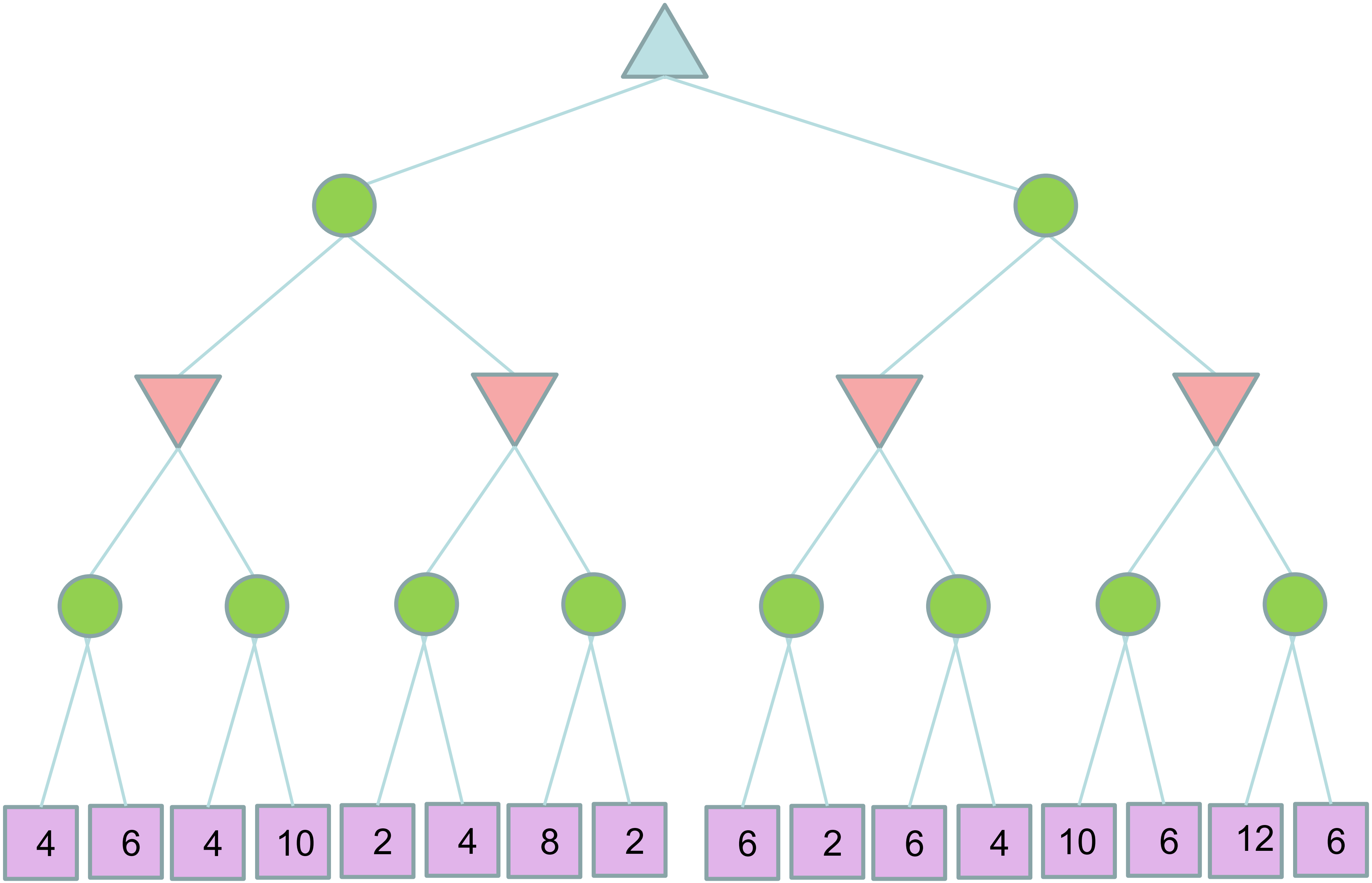


Expectimax Pruning?



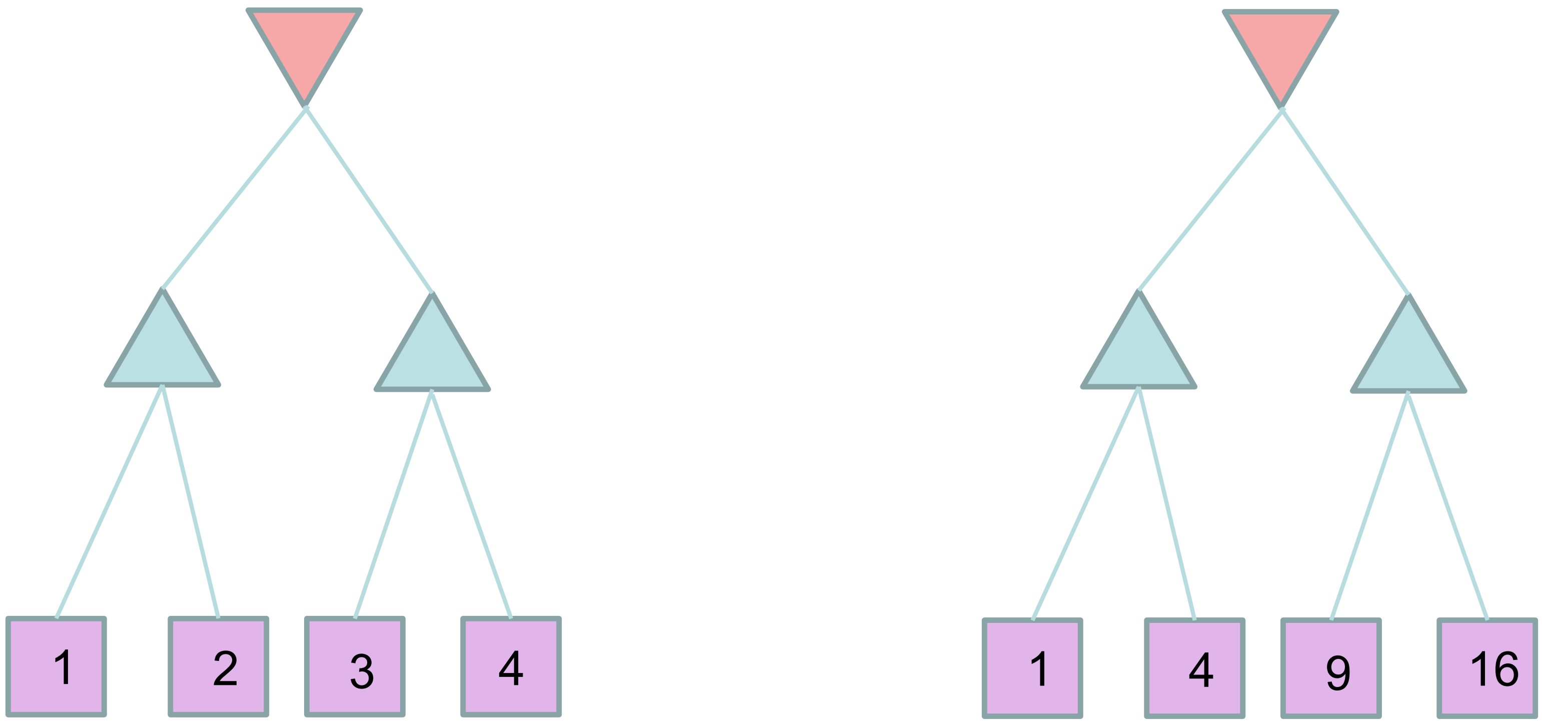


Expecti-minimax



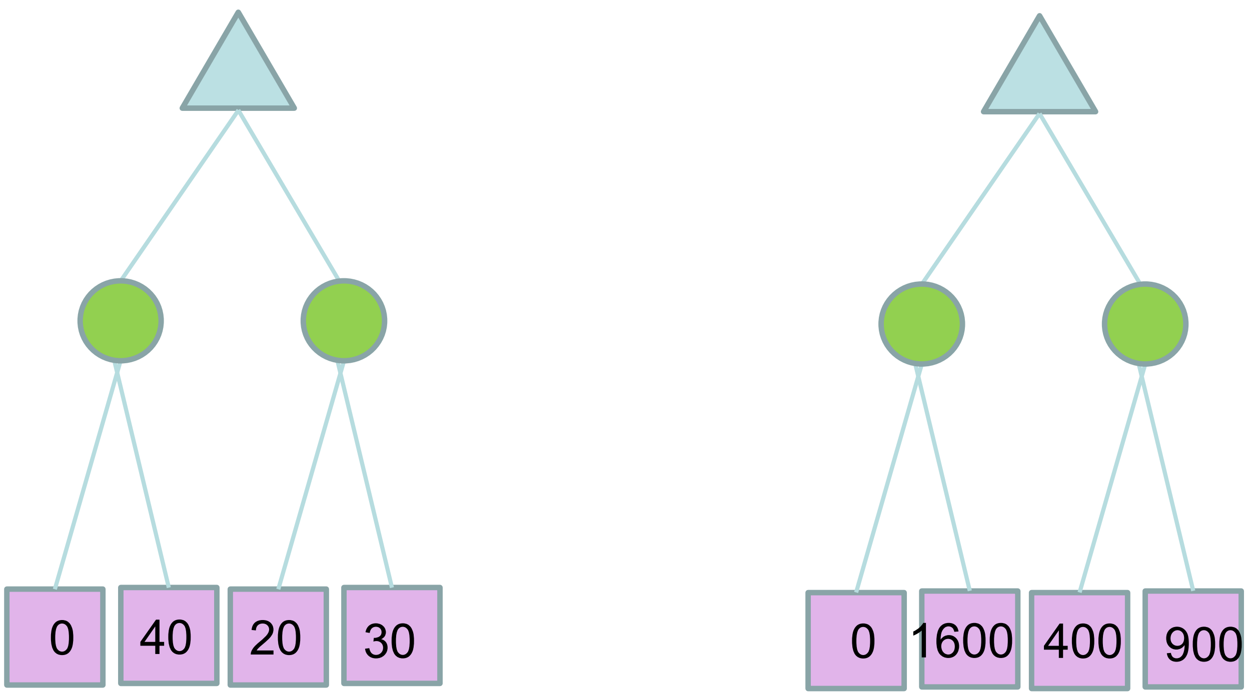
Utility values

- For worst-case minimax reasoning, terminal function scale doesn't matter
  - We just want better states to have higher evaluations (get the ordering right)
  - We call this insensitivity to monotonic transformations



Utility values

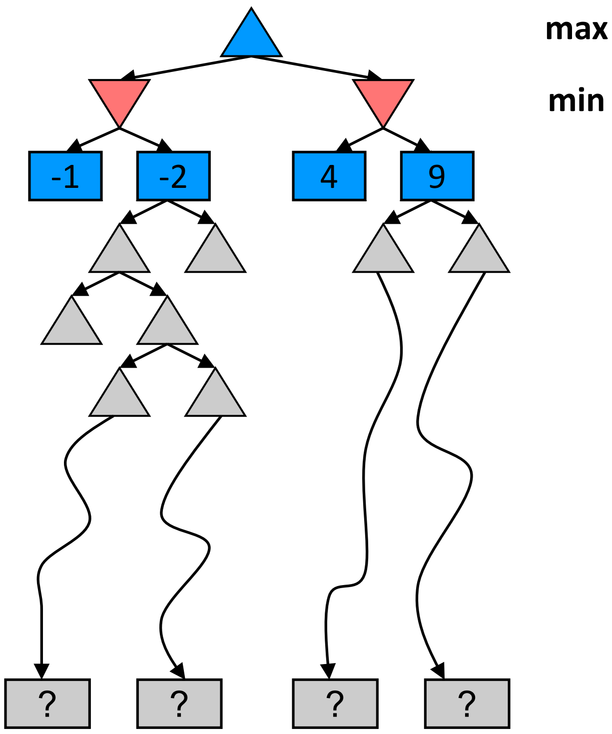
- For worst-case minimax reasoning, terminal function scale doesn't matter
  - We just want better states to have higher evaluations (get the ordering right)
  - We call this insensitivity to monotonic transformations
  - For average-case expectimax reasoning, we need *magnitudes* to be meaningful



Resource Limits

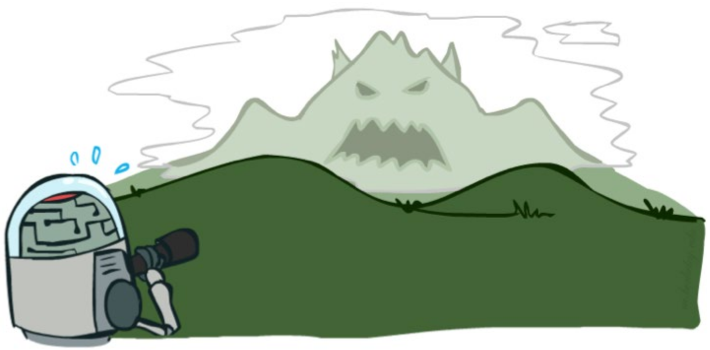
Resource Limits

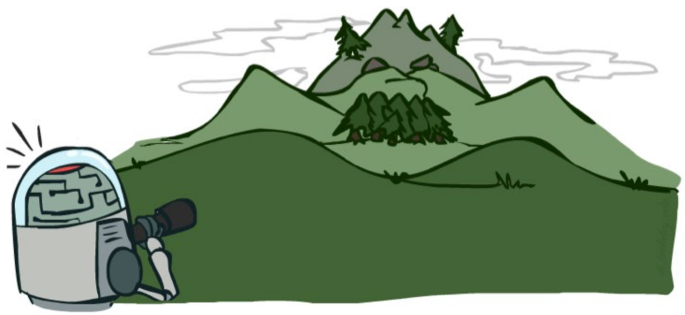
- Problem: In realistic games, cannot search to leaves!
- Solution: Depth-limited search
  - Instead, search only to a limited depth in the tree
  - Replace terminal utilities with an evaluation function for non-terminal positions
- Example:
  - Suppose we have 100 seconds, can explore 10K nodes / sec
  - So can check 1M nodes per move
  - α-β reaches about depth 8 – decent chess program
- Guarantee of optimal play is gone
- More plies makes a BIG difference
- Use iterative deepening for an anytime algorithm



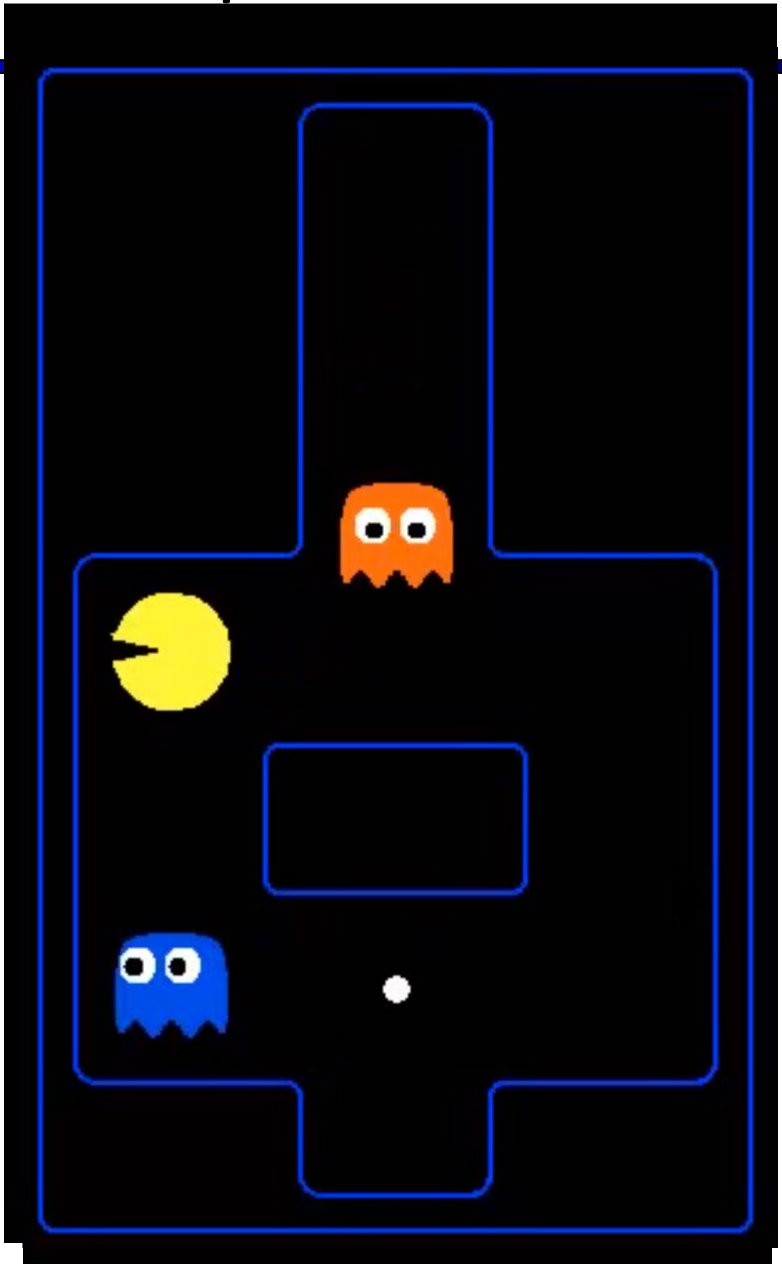
Depth Matters

- Evaluation functions tend to be imperfect
- The deeper in the tree the evaluation function is buried, the less the quality of the evaluation function matters
- An important example of the tradeoff between complexity of features and complexity of computation

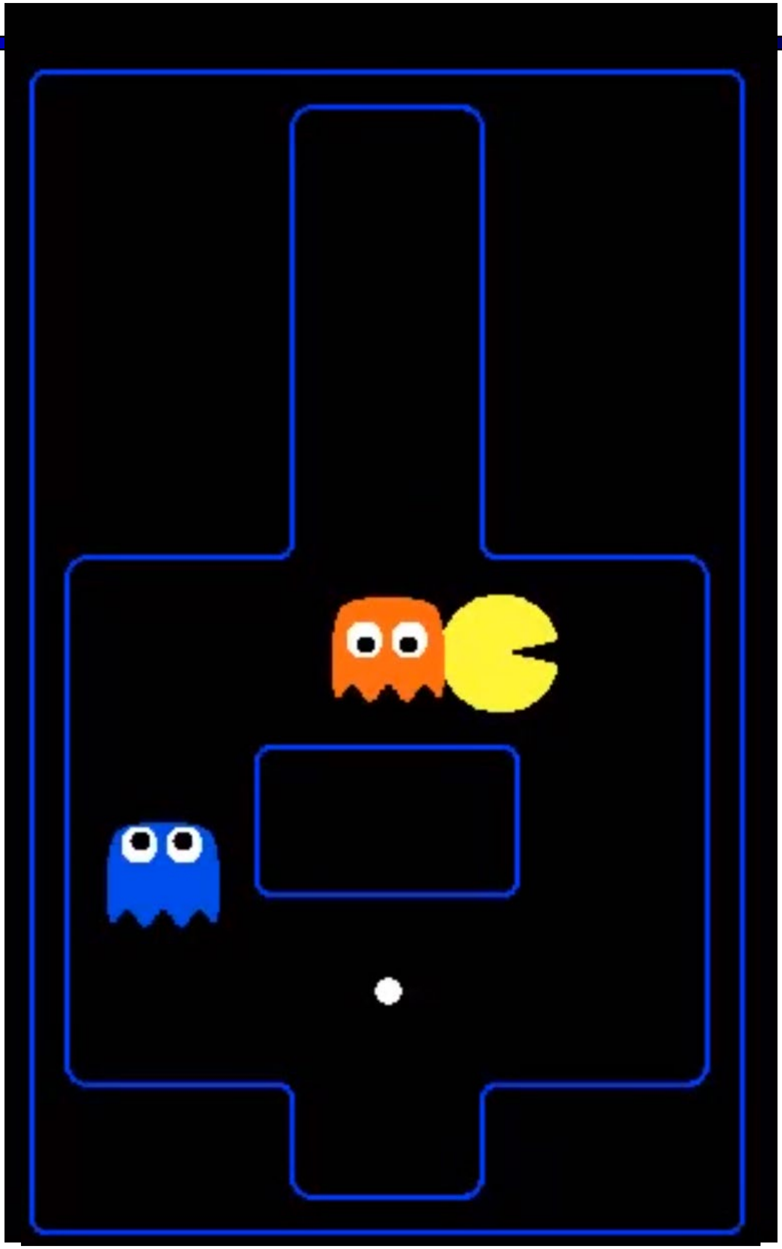




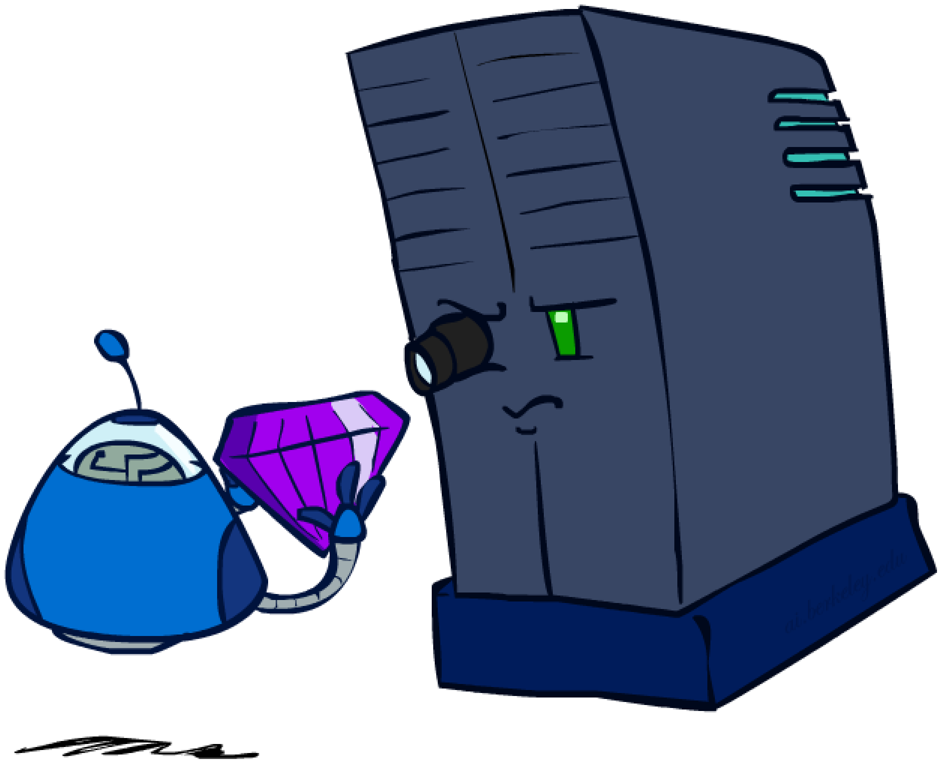
Depth 2 search



Depth 10 search

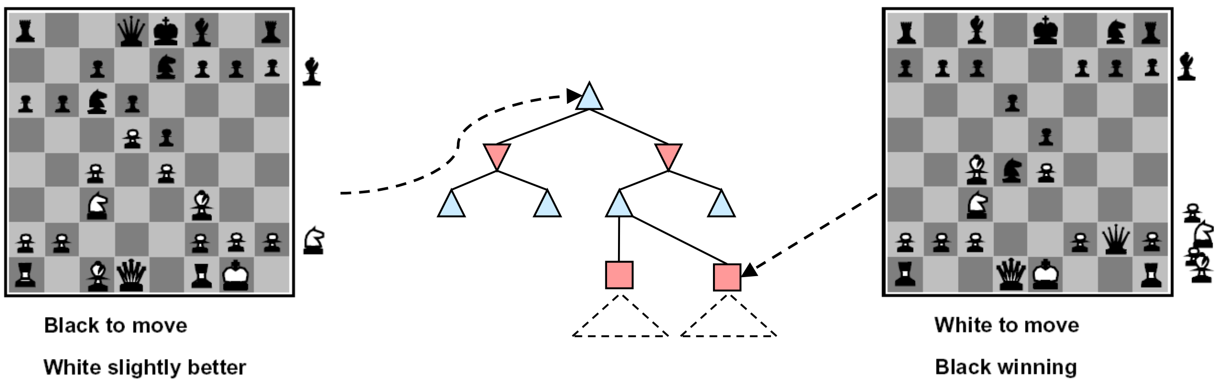


Evaluation Functions



Evaluation Functions

- Evaluation functions score non-terminals in depth-limited search

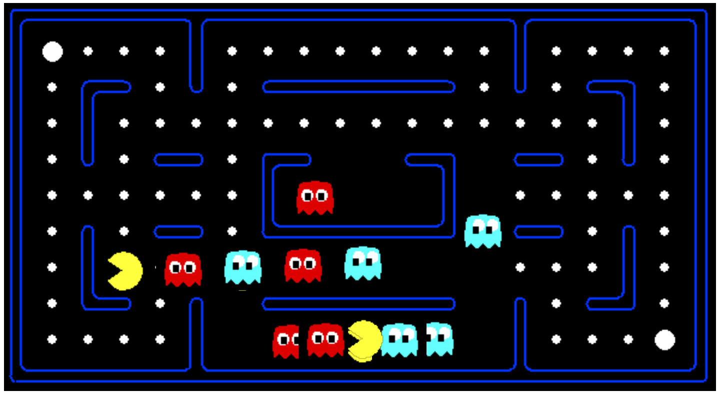


- Ideal function: returns the actual minimax value of the position
- In practice: typically weighted linear sum of features:

$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

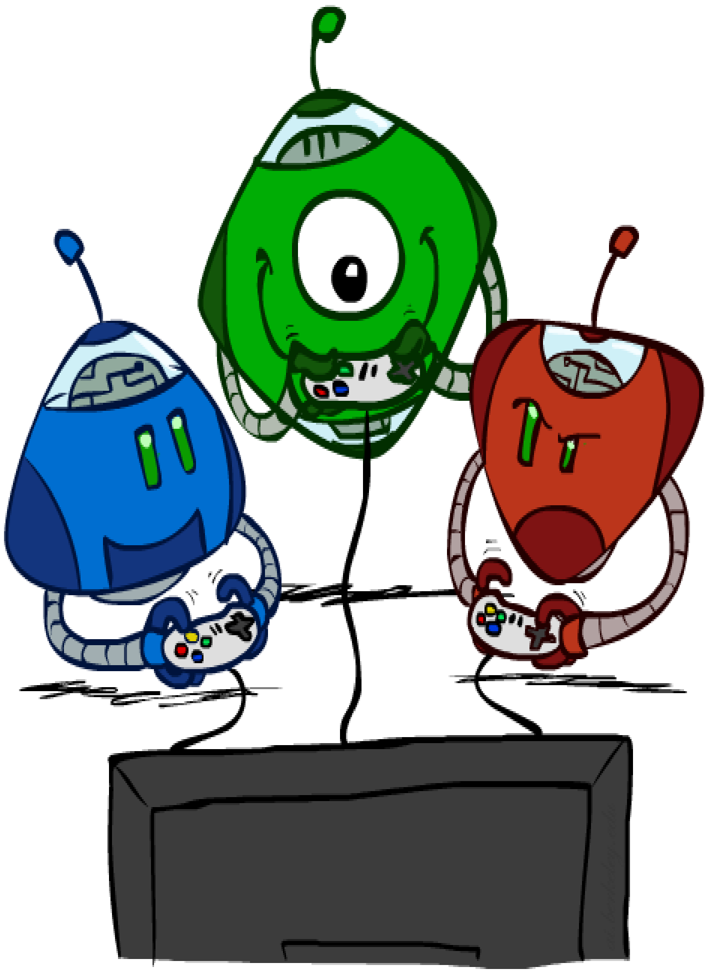
- e.g. $f_1(s)$ = (num white queens – num black queens), etc.

Evaluation for Pacman



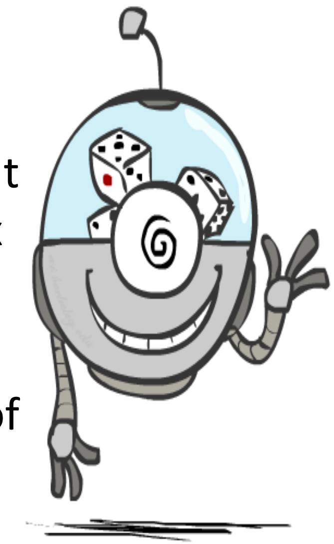
What features are we using?
Eval(n)=

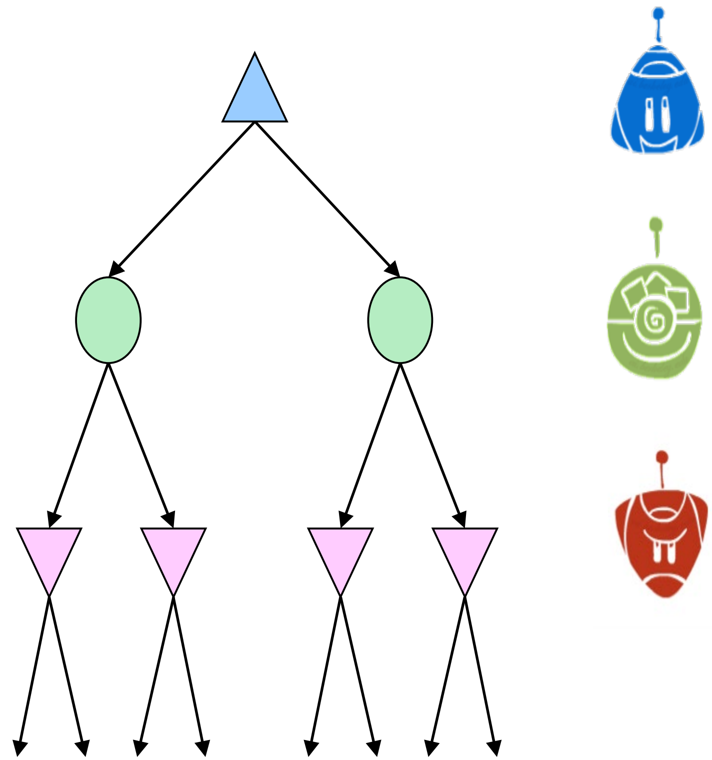
Other Game Types



Mixed Layer Types

- E.g. Backgammon
- Expectiminimax
  - Environment is an extra "random agent" player that moves after each min/max agent
  - Each node computes the appropriate combination of its children



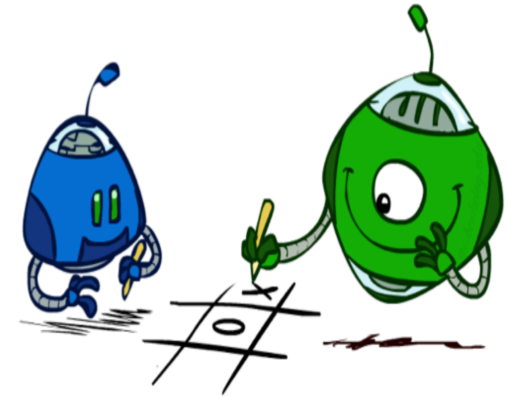


Example: Backgammon

- Dice rolls increase *b*: 21 possible rolls with 2 dice
  - Backgammon ≈ 20 legal moves
- As depth increases, probability of reaching a given search node shrinks
  - So usefulness of search is diminished
  - So limiting depth is less damaging
- Historic AI: TDGammon uses depth-2 search + very good evaluation function + reinforcement learning: world-champion level play
- 1st AI world champion in any game!

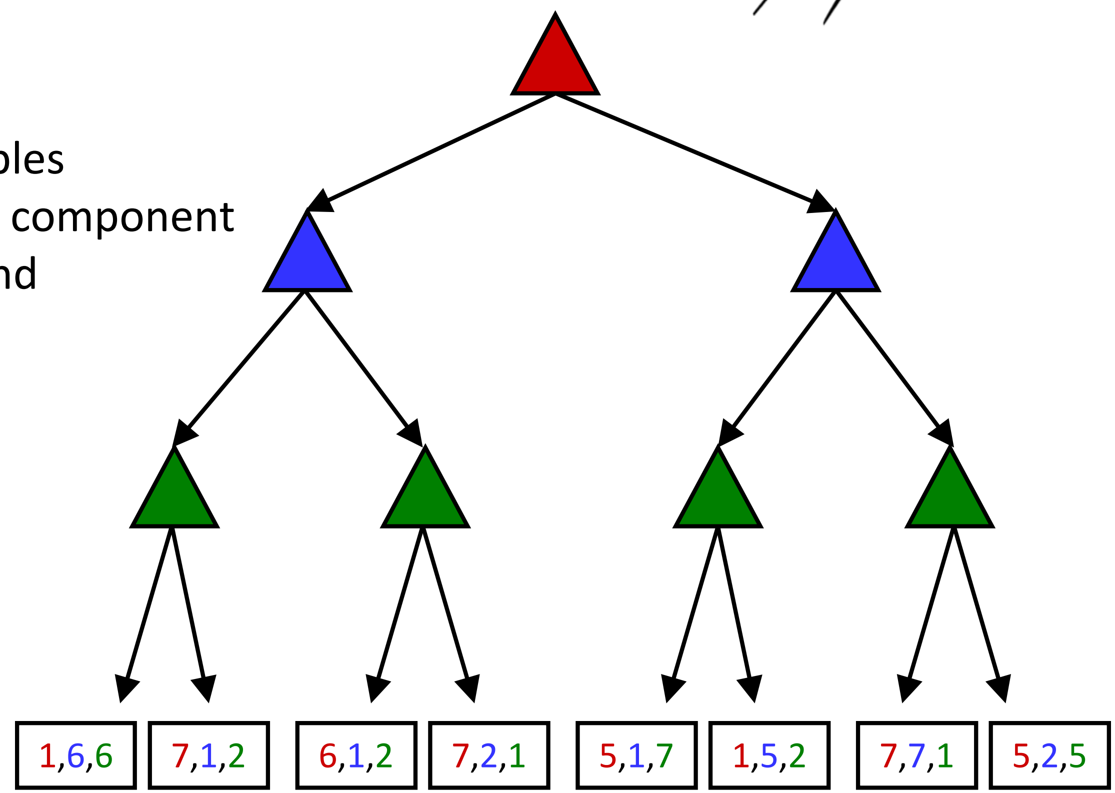
Image: Wikipedia

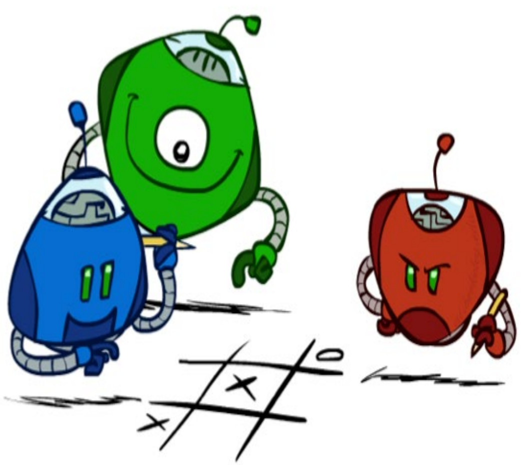
Multi-Agent Utilities



- What if the game is not zero-sum, or has multiple players?

- Generalization of minimax:
  - Terminals have utility tuples
  - Node values are also utility tuples
  - Each player maximizes its own component
  - Can give rise to cooperation and competition dynamically…





Summary

- Adversarial search
  - Mini-Max
  - Alpha-Beta pruning
  - Expecti-mini-max
  - Limited depth search

Lab 3 Intro

- A* search on grid world
  - Due April 28,7pm
- Use Manhattan distance as H(n)

1	1	1	1	0
0	2	0	0	0
0	1	1	1	1
0	1	1	3	1
0	0	0	0	0

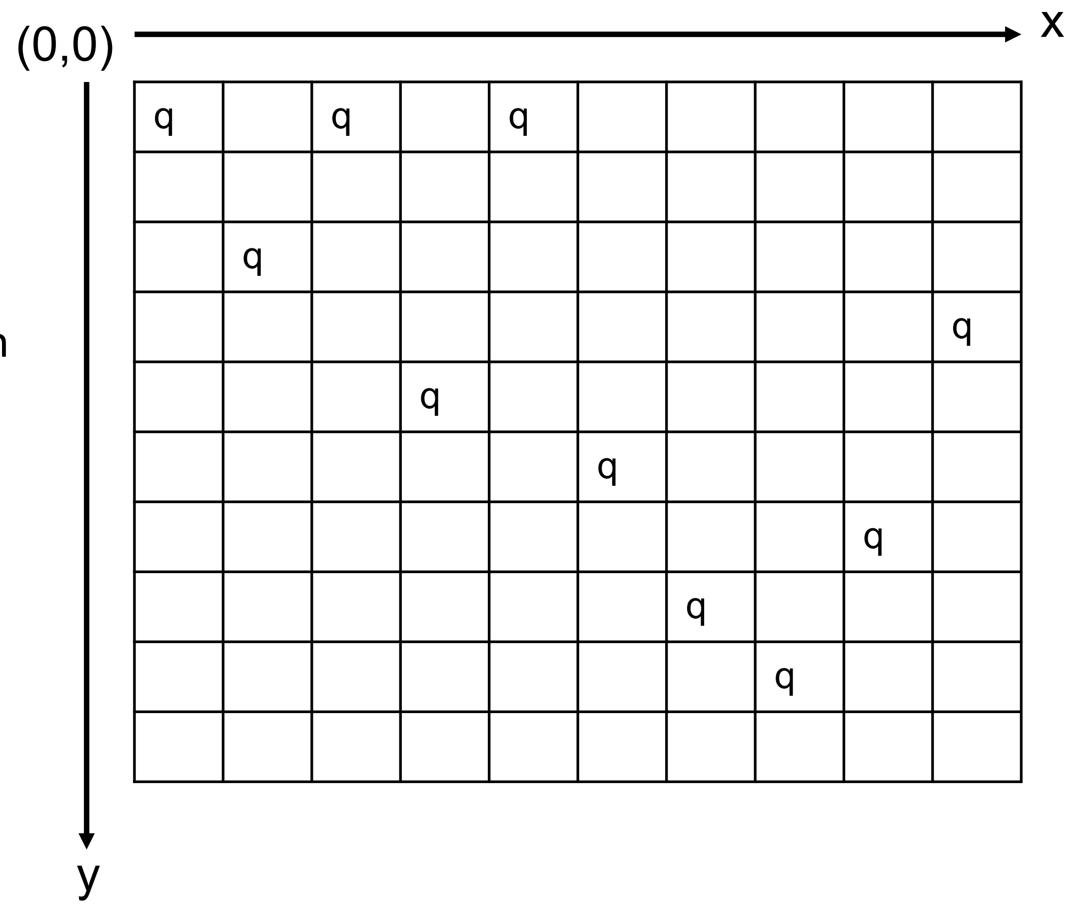
- In case of multiple nodes in frontier with lowest F(n)
  - Expand node with lowest x, if still tied, then expand node with lowest y

- No loops
- Exactly 1 starting position, 1 goal position

- OK to reuse portions of your code from lab 1

Lab 2, Due April 21, 7pm

- 10-queens problem
  - Queens constrained to column
  - Find local minima using greedy search
    - Each step is motion of one queen in her column
      - 81 possible next moves
    - Follow tie breaking rules



Lab 2 discussion

Lab 2 discussion