
CS 232: Artificial Intelligence

Fall 2026

Prof. Carolyn Anderson
Wellesley College

Reminders

- ♦ I have help hours today (W323) and Friday from 3-4 (W423)
- ♦ HW 4 will be released today
- ♦ Tutor hours:
 - Aeka: Thursdays 7:30-8:30 in H103
 - *This week: Saturday*
 - Amber: ~~Fridays 1:30-2:30~~ in Whiteboard Alley
 - Coco: Mondays 7-8 in Whiteboard Alley
- ♦ No class on 10/16

Homework 4

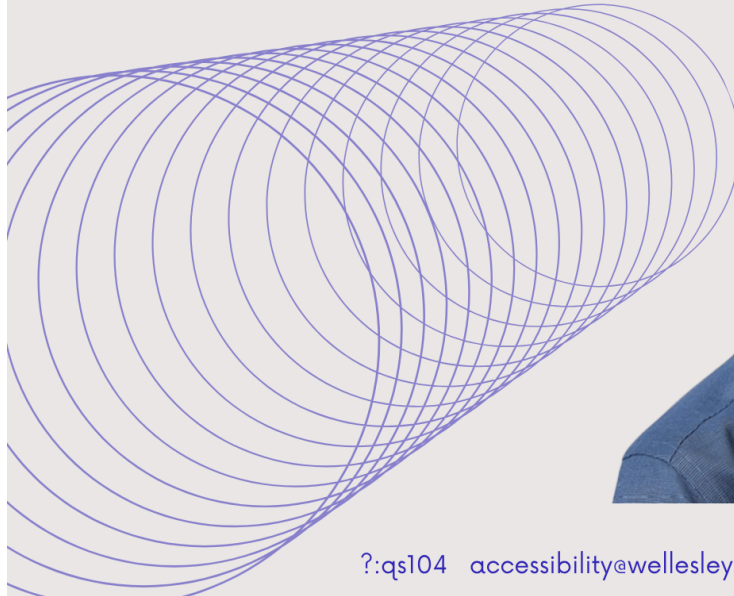
In Homework 4, we're introducing ghosts into our Pacman game. You'll implement minimax search. One catch: there are multiple ghosts and only 1 Pacman! So *each ghost* will do a round of minimizing before Pacman gets to move again.

THURSDAY, OCTOBER 8
12:45PM TO 2PM
H105



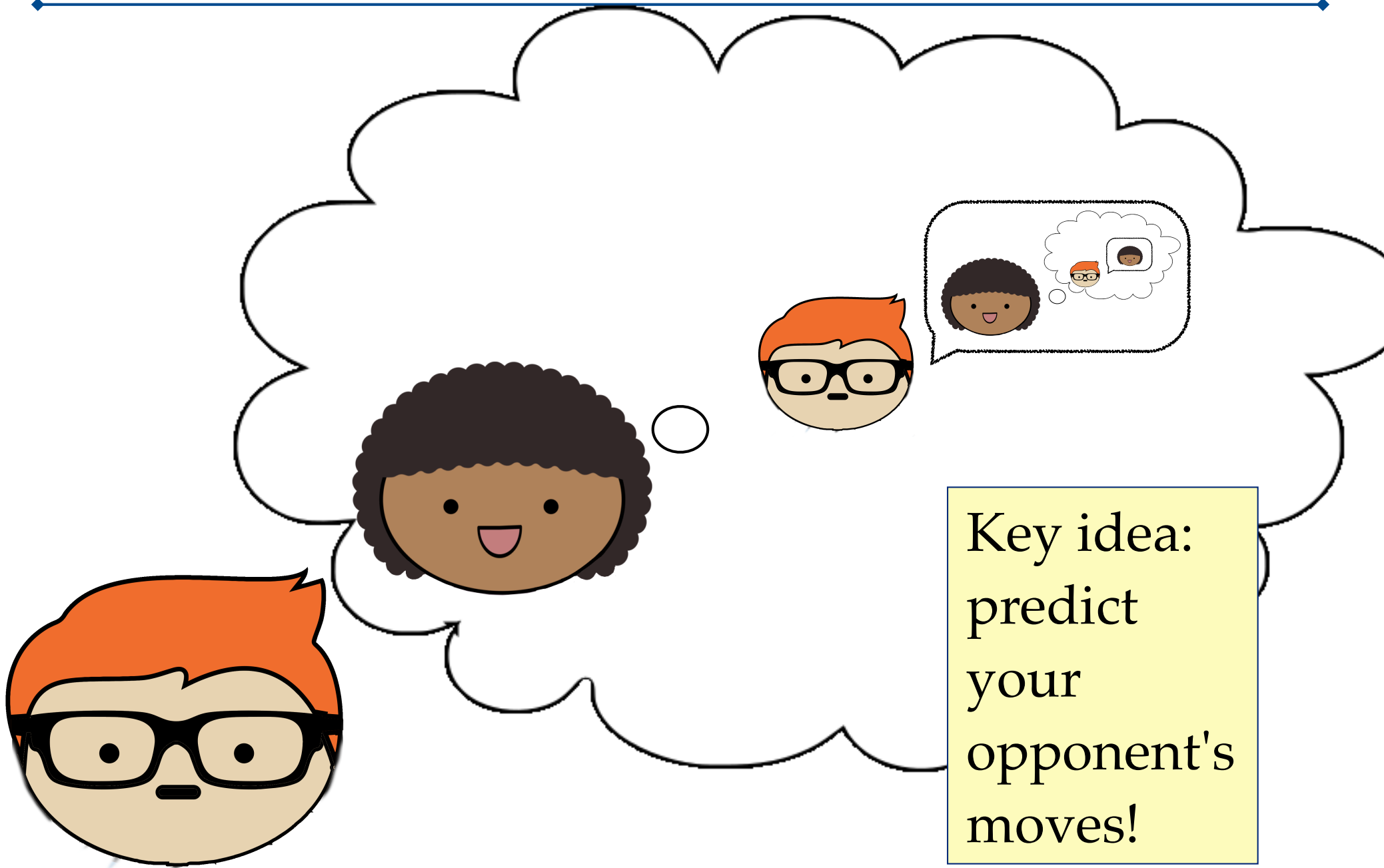
<https://bit.ly/3V2EJLd>

CS Colloquium:
**Tech Ethics and
College Chaplaincy:
Lessons from MIT
Radius**

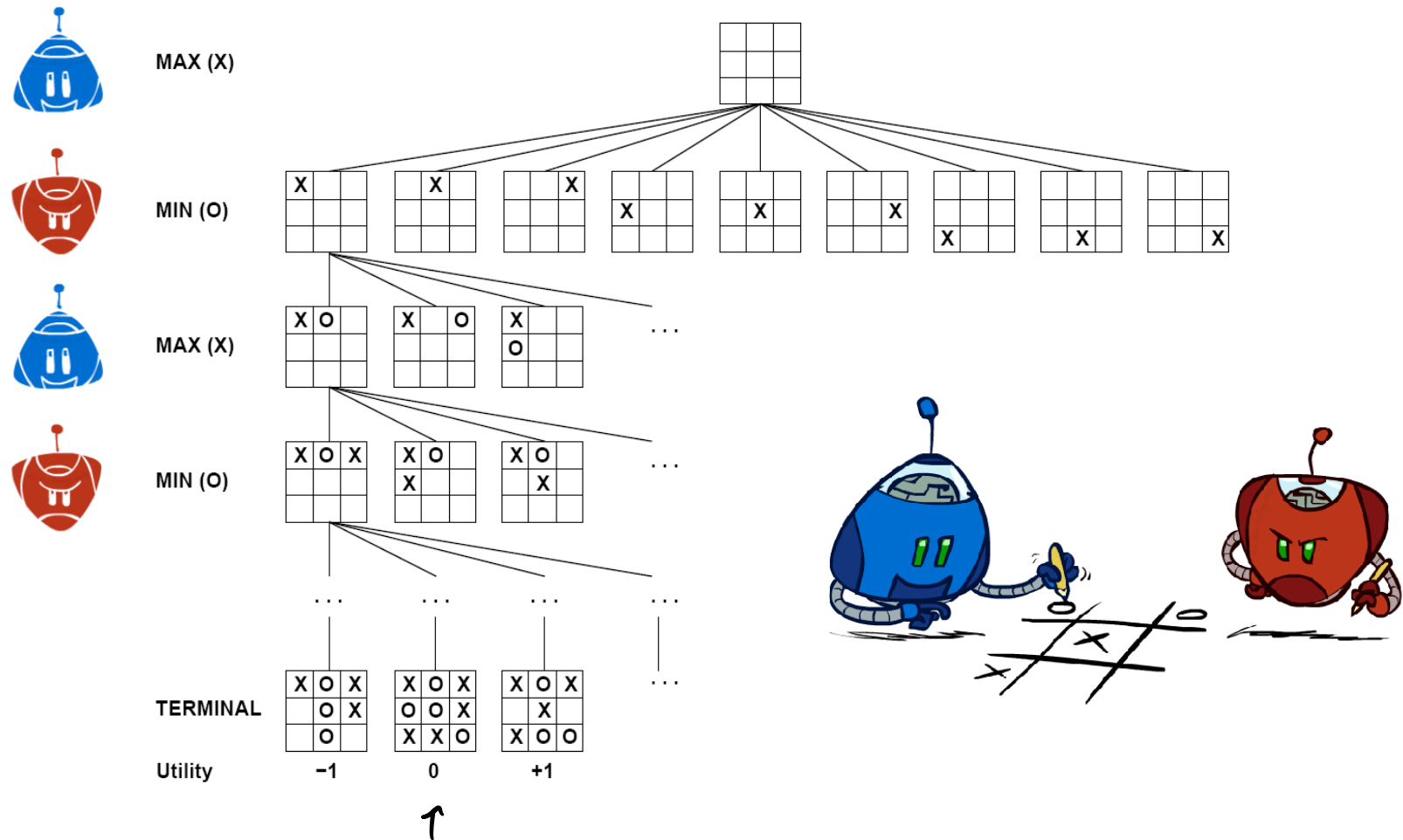


Adversarial Games

Deterministic Adversarial Games



Tic-Tac-Toe Game Tree



Minimax Search

In Minimax, we seek to optimize our score at the expense of our opponent.

We do this by reasoning recursively to predict their moves and compute the **expected utility** of various states we could reach.

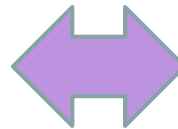
Minimax Algorithm

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

$$V(s) = \max_{s' \in \text{successors}(s)} V(s')$$

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

$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$



Minimax Algorithm

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 MIN: return min-value(state)

def max-value(state):

initialize $v = -\infty$

for each successor of state:

$v = \max(v, \text{min-value}(\text{successor}))$

return v

$$V(s) = \max_{s' \in \text{successors}(s)} V(s')$$

def min-value(state):

initialize $v = +\infty$

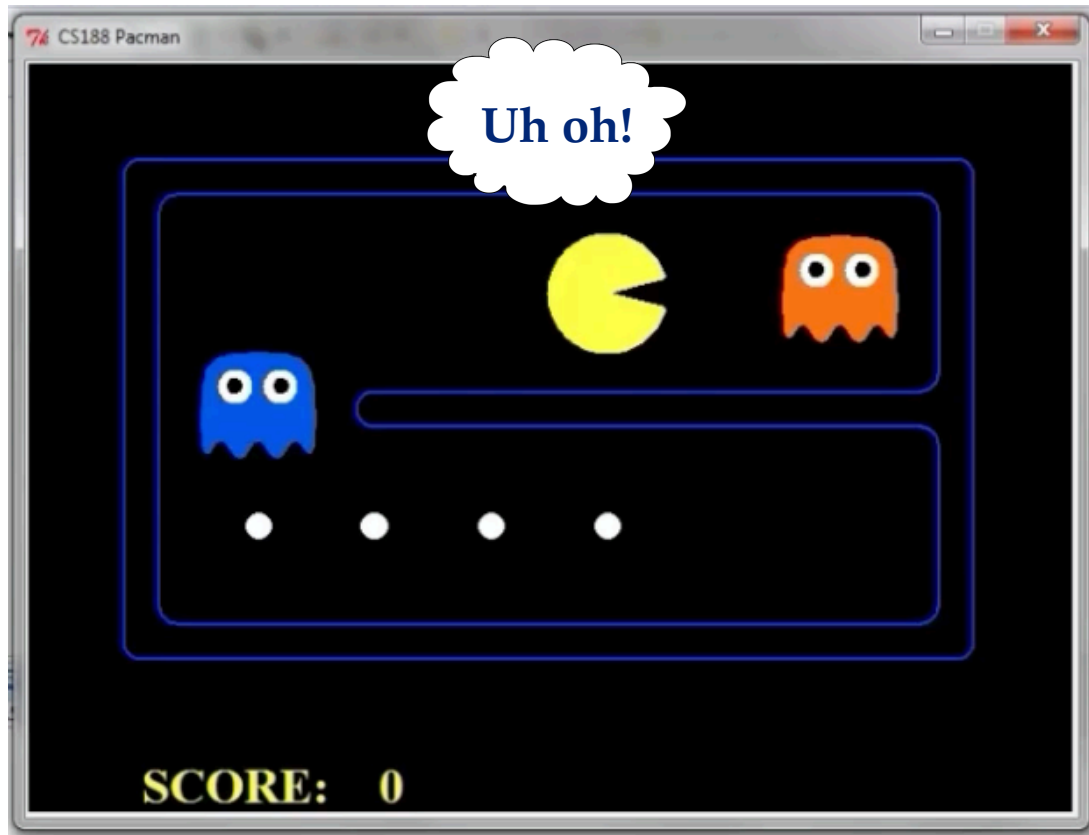
for each successor of state:

$v = \min(v, \text{max-value}(\text{successor}))$

return v

$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$

Expectations v Reality: Pacman



Minimax Summary

- ✦ Rank final game states by their final scores (for tic-tac-toe or chess: win, draw, loss).
- ✦ Rank intermediate game states by whose turn it is and the available moves.
 - If it's X's turn, set the rank to that of the *maximum* move available. If a move will result in a win, X should take it.
 - If it's O's turn, set the rank to that of the *minimum* move available. If a move will result in a loss, X should avoid it.

Efficiency

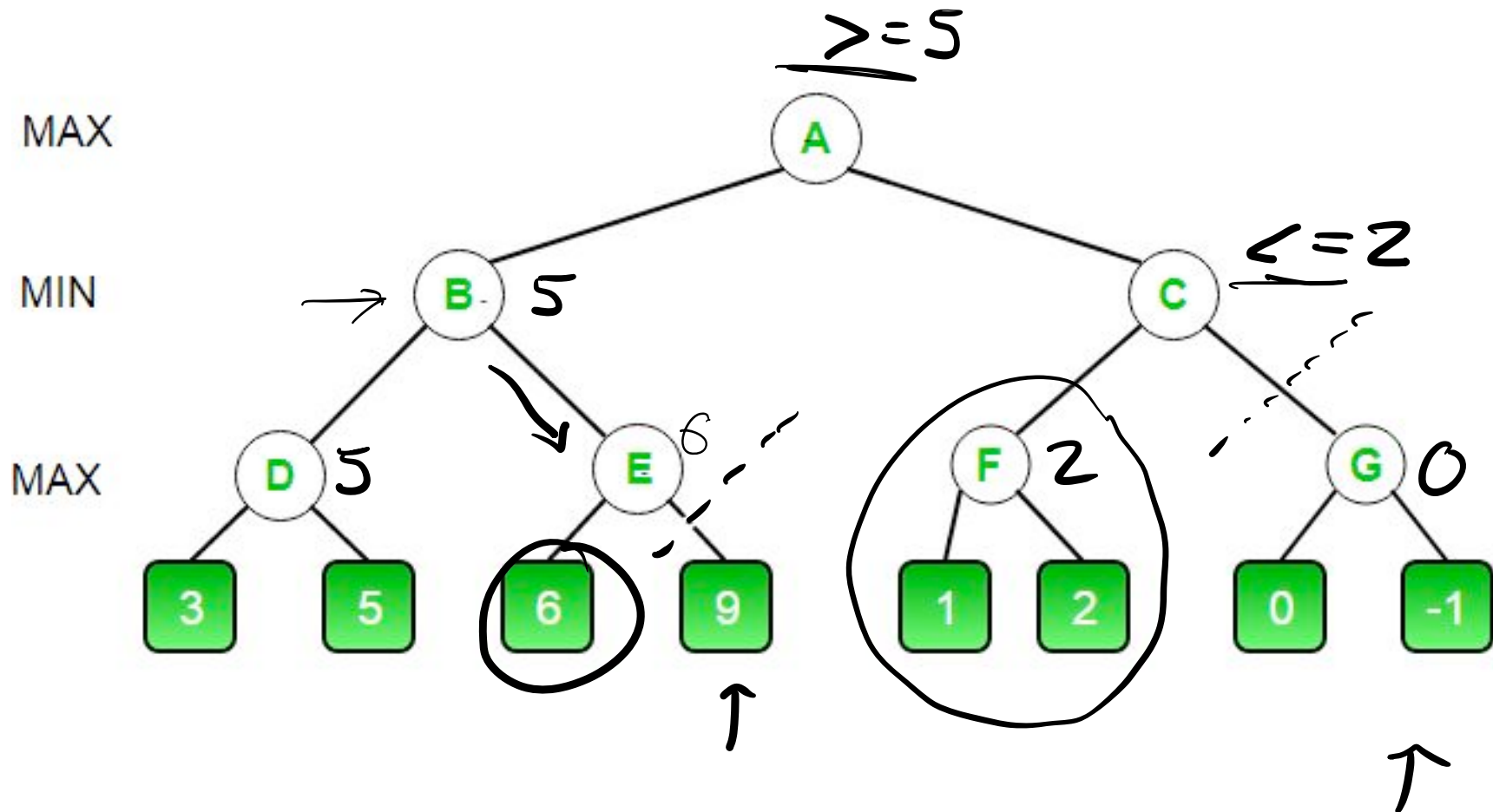
Minimax Efficiency

How efficient is minimax?

- Just like (exhaustive) DFS
- Time: $O(b^m)$
- Space: $O(bm)$
- For chess, $b \approx 35$, $m \approx 100$

So, the exact solution is infeasible. But do we need to explore the whole tree?

Minimax Example



Pruning

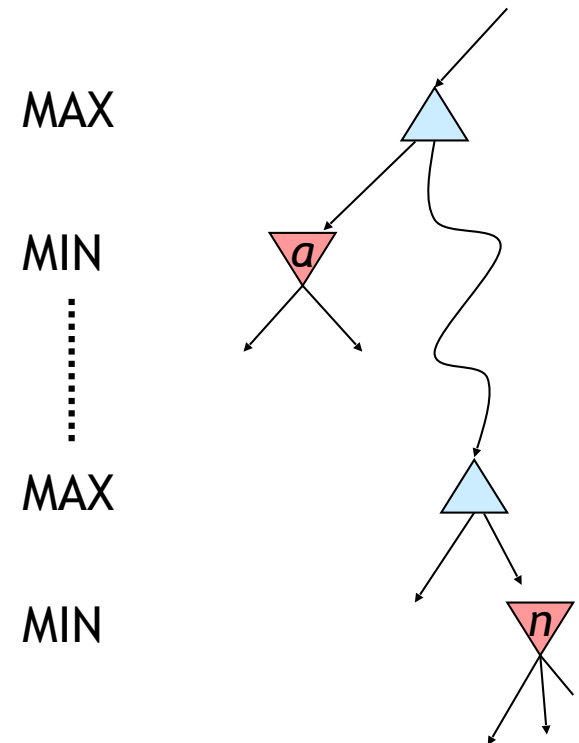
Key idea: give up on paths when you realize that they are worse than options you've already explored .

- ♦ Track the maximum score possible for the minimizing player (beta)
- ♦ Track the minimum score possible for the maximizing player (alpha)

Whenever the **maximum score for beta** becomes less than the **minimum score for alpha**, the maximizing player can stop searching down this path, because it will never be reached.

Alpha-Beta Pruning

- General configuration (MIN version)
 - We're computing the MIN-VALUE at some node n
 - We're looping over n 's children
 - n 's estimate of the childrens' min is dropping
 - Who cares about n 's value? MAX
 - Let a be the best value that MAX can get at any choice point along the current path from the root
 - If n becomes worse than a , MAX will avoid it, so we can stop considering n 's other children (it's already bad enough that it won't be played)
- MAX version is symmetric



Alpha-Beta Implementation

α : MAX's best option on path to root
 β : MIN's best option on path to root

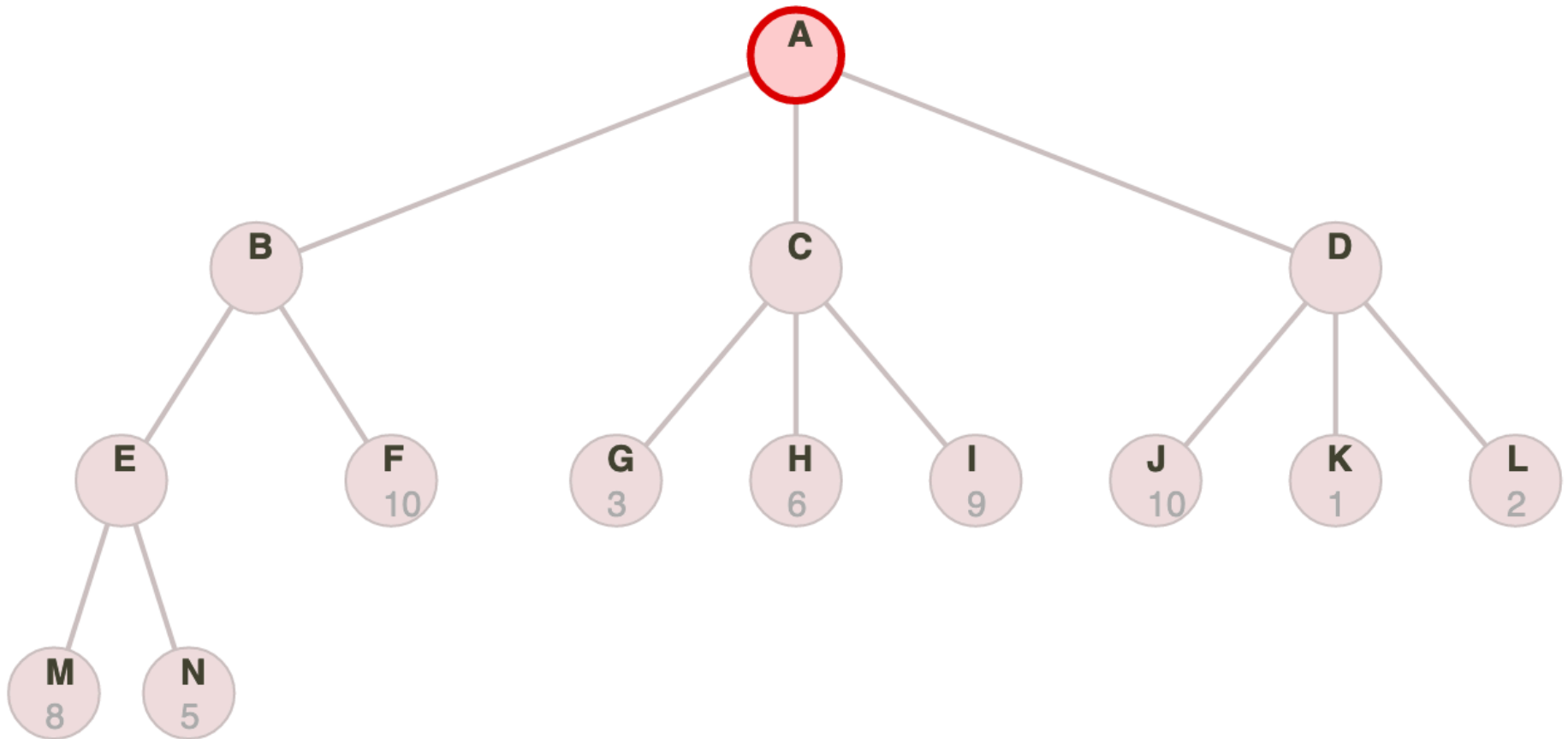
```
def max-value(state,  $\alpha$ ,  $\beta$ ):  
    initialize  $v = -\infty$   
    for each successor of state:  
         $v = \max(v,$   
             $\text{value}(\text{successor}, \alpha, \beta))$   
        if  $v \geq \beta$  return  $v$   
         $\alpha = \max(\alpha, v)$   
    return  $v$ 
```

```
def min-value(state,  $\alpha$ ,  $\beta$ ):  
    initialize  $v = +\infty$   
    for each successor of state:  
         $v = \min(v,$   
             $\text{value}(\text{successor}, \alpha, \beta))$   
        if  $v \leq \alpha$  return  $v$   
         $\beta = \min(\beta, v)$   
    return  $v$ 
```

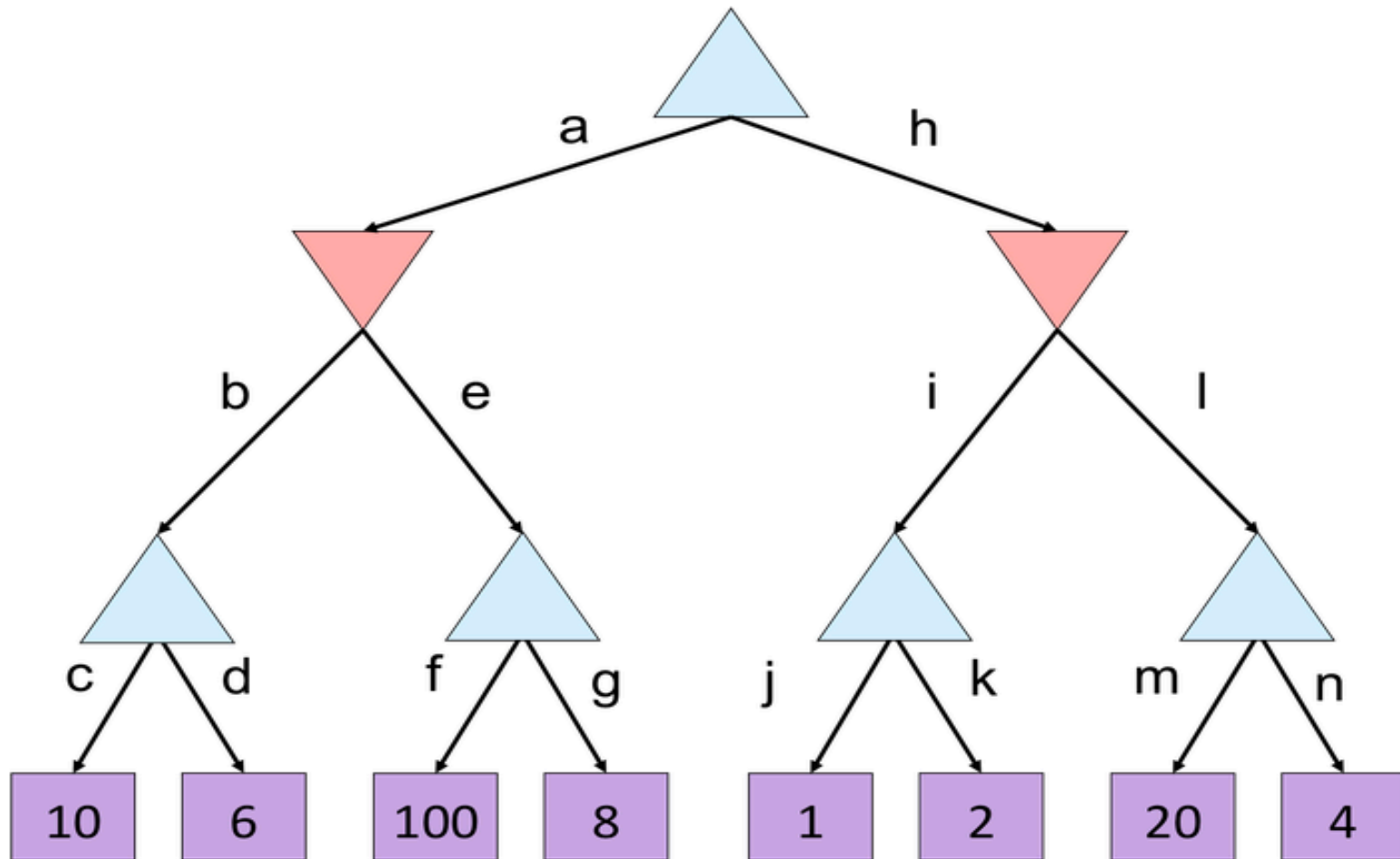
Alpha-Beta Pruning Properties

- Pruning doesn't effect minimax value
- Values of intermediate nodes might be wrong
- Good child ordering increases efficiency
- With perfect ordering:
Time complexity drops to $O(b^{n/2})$
Doubles our searchable depth!

Another Demo

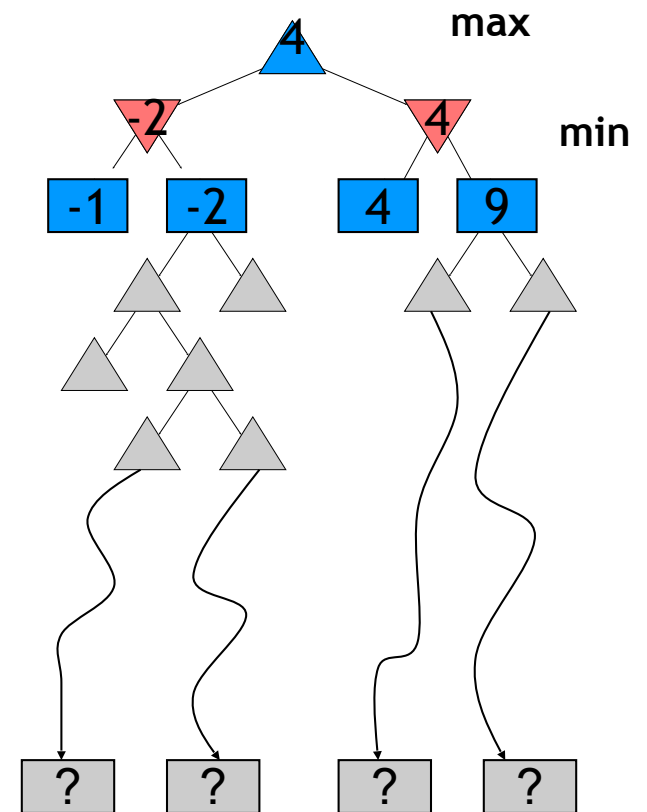


Alpha-Beta Pruning Exercise

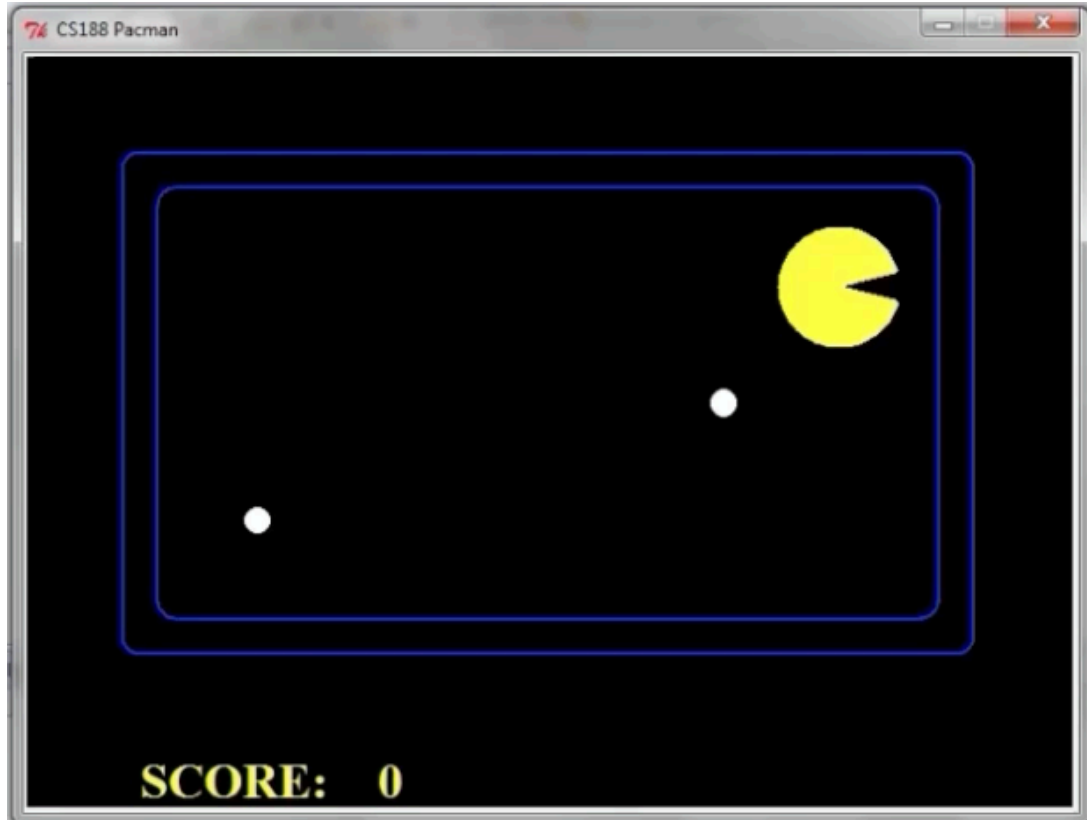


Resource Limits

- ♦ 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



Resource Limits



Ask a friend:
If we set the depth
limit to 2, what
will Pacman do?

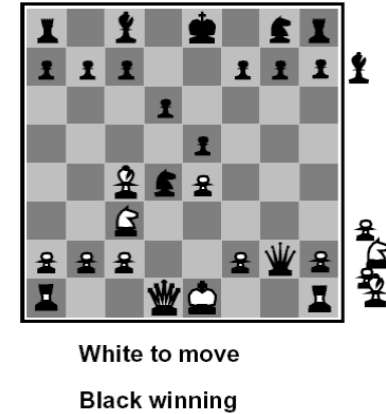
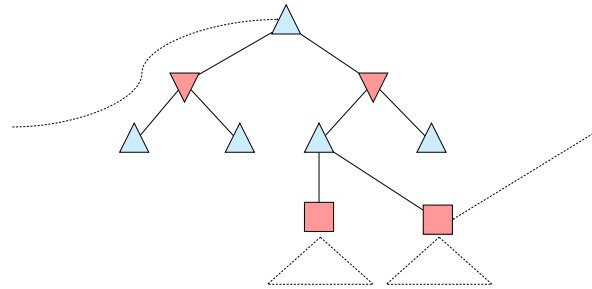
Why Pacman Starves

- A danger of replanning agents!
 - He knows his score will go up by eating the dot now (west, east)
 - He knows his score will go up just as much by eating the dot later (east, west)
 - There are no point-scoring opportunities after eating the dot (within the horizon, two here)
 - Therefore, waiting seems just as good as eating: he may go east, then back west in the next round of replanning!

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:
- e.g. $f_1(s) = (\text{num white queens} - \text{num black queens})$, etc.

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

Evaluation for Pacman

Talk to a friend: what would be a good evaluation function for Pacman?