CIS 471/571(Fall 2020): Introduction to Artificial Intelligence

Lecture 1 Search

Thanh H. Nguyen

Source: http://ai.berkeley.edu/home.html

Reminders

- Project 2:
  - Deadline: Oct 27th, 2020

- Written assignmen
  - Deadline: Oct 24th, 2020

Adversarial Games

Types of Games

- Many different kinds of games!

- Axes:
  - Deterministic or stochas
  - One, two, or more player
  - Zero sum?
  - Perfect information (can you see the state)?

- Want algorithms for calculating a strategy (policy) which recommends a move from each state

Deterministic Games

- Many possible formalizations, one is:
  - States: S (start at s_0)
  - Players: P={1...N} (usually take turns)
  - Actions: A (may depend
  - Transition Function: Sx
  - Terminal Test: $S \rightarrow \{t,f\}$
  - Terminal Utilities: $SxP \rightarrow R$

- Solution for a player is a policy: $S \rightarrow A$

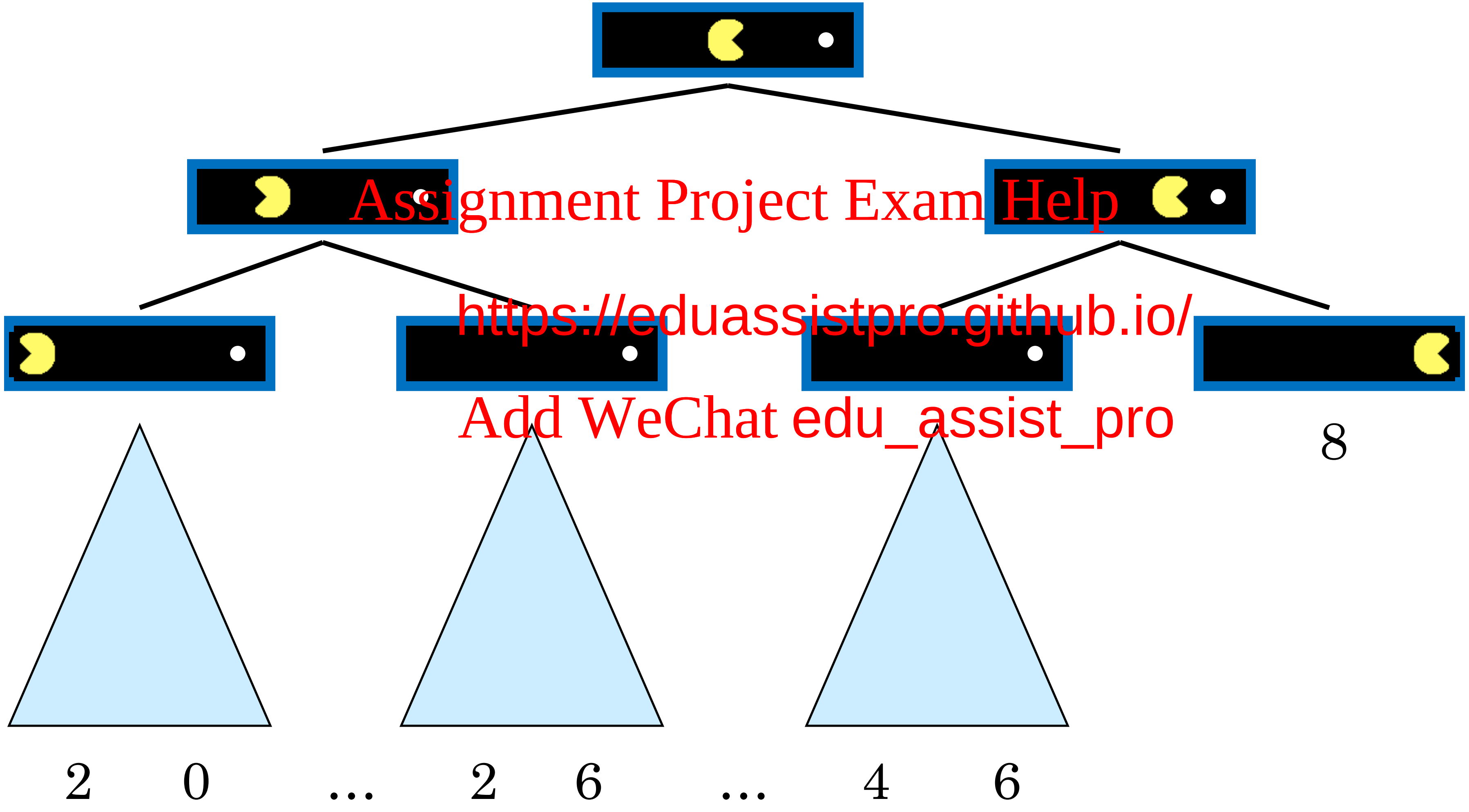
Zero-Sum Games

- Zero-Sum Games
  - Agents have opposite utilities (values on outcomes)
  - Lets us think of a single value that one maximizes and the other minimizes
  - Adversarial, pure competition

- General Games
  - Agents have independent utilities (values on outcomes)
  - Cooperation, indifference, competition, and more are all possible
  - More later on non-zero-sum games

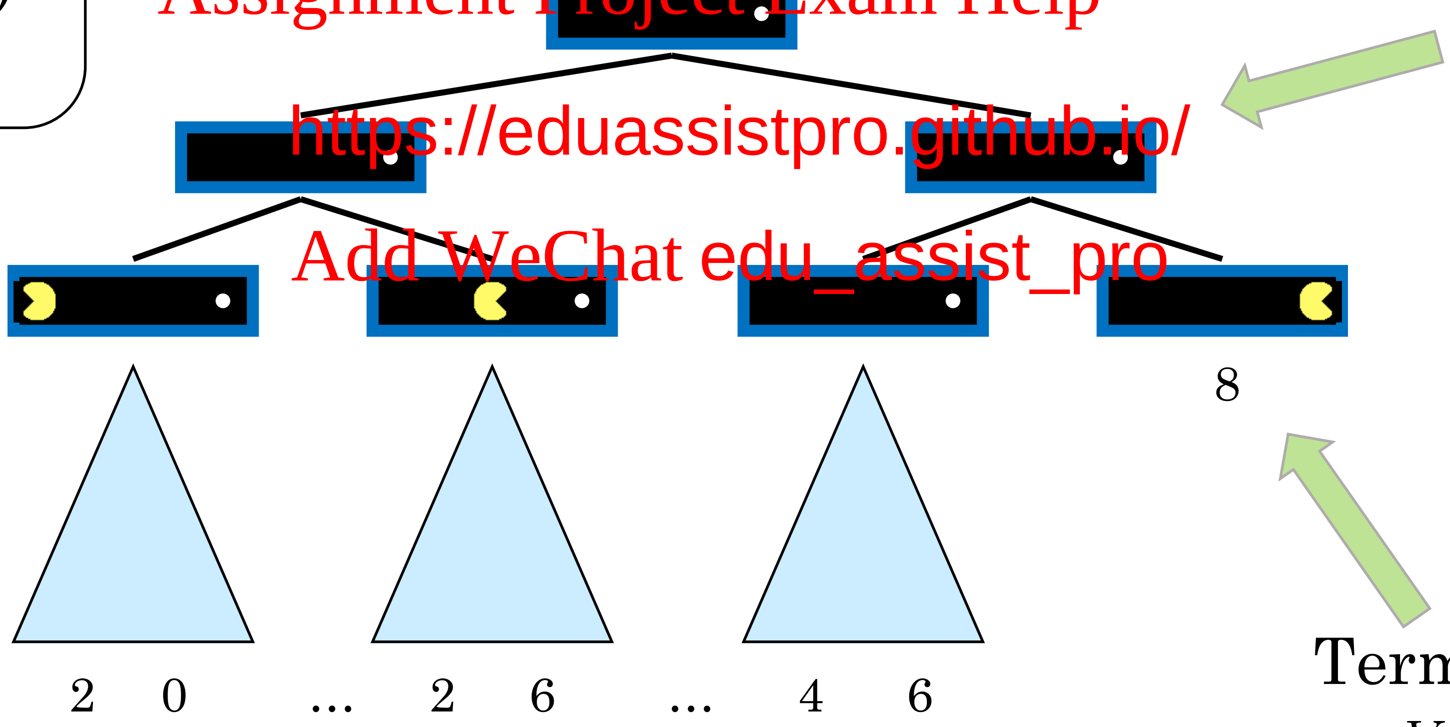
Adversarial Search

Single-Agent Trees



Value of a State

Value of a state:
The best achievable outcome (utility) from that state



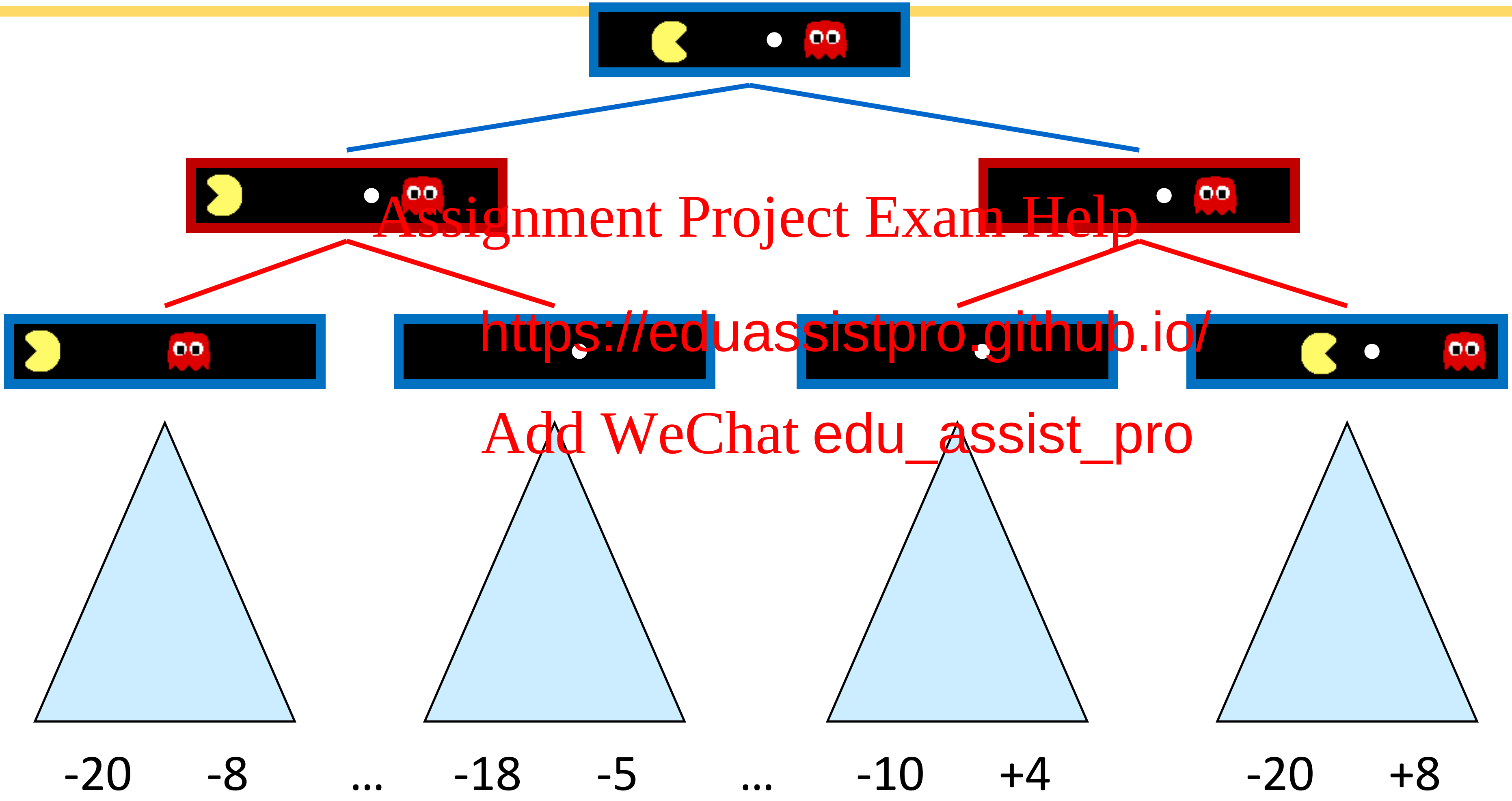
Non-Terminal States:

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

Terminal States:

$$V(s) = \text{known}$$

Adversarial Game Trees



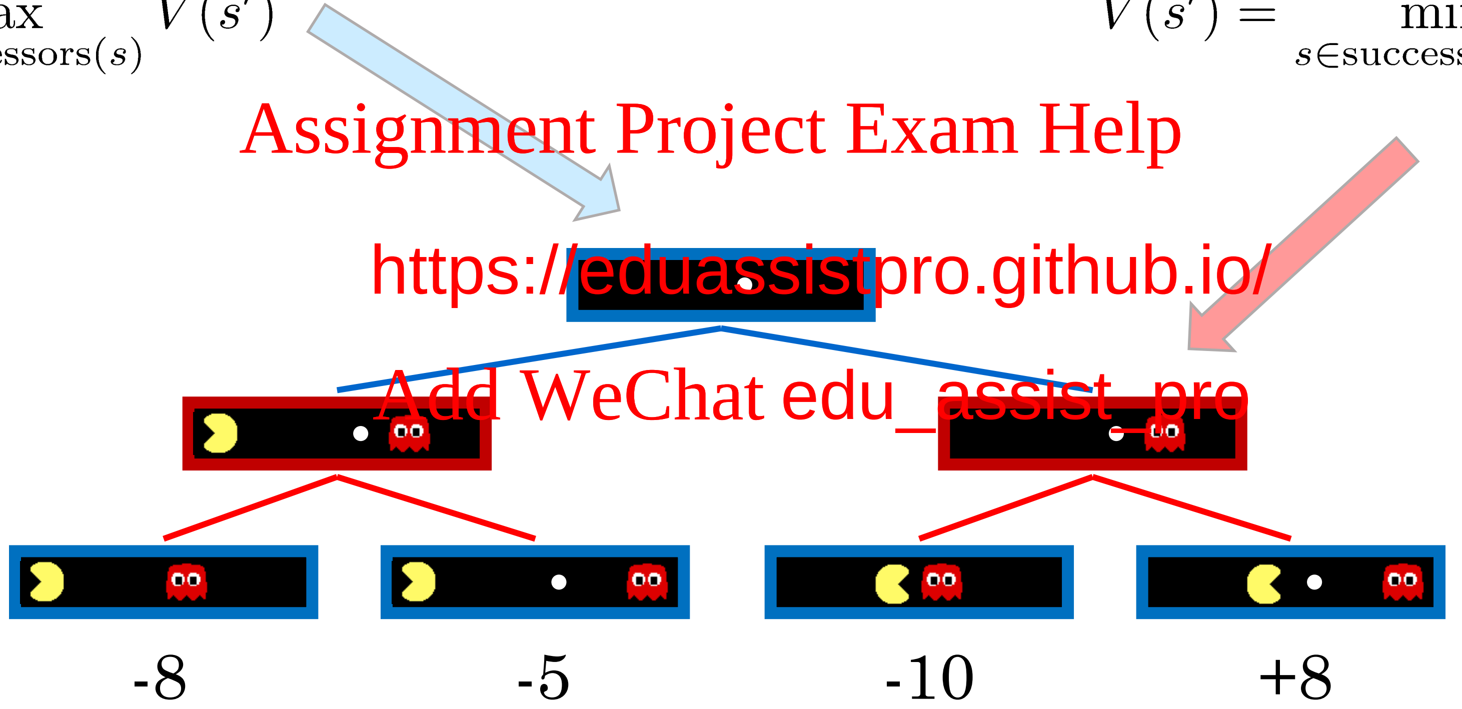
Minimax Values

States Under Agent's Control:

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

States Under Opponent's Control:

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



Terminal States:

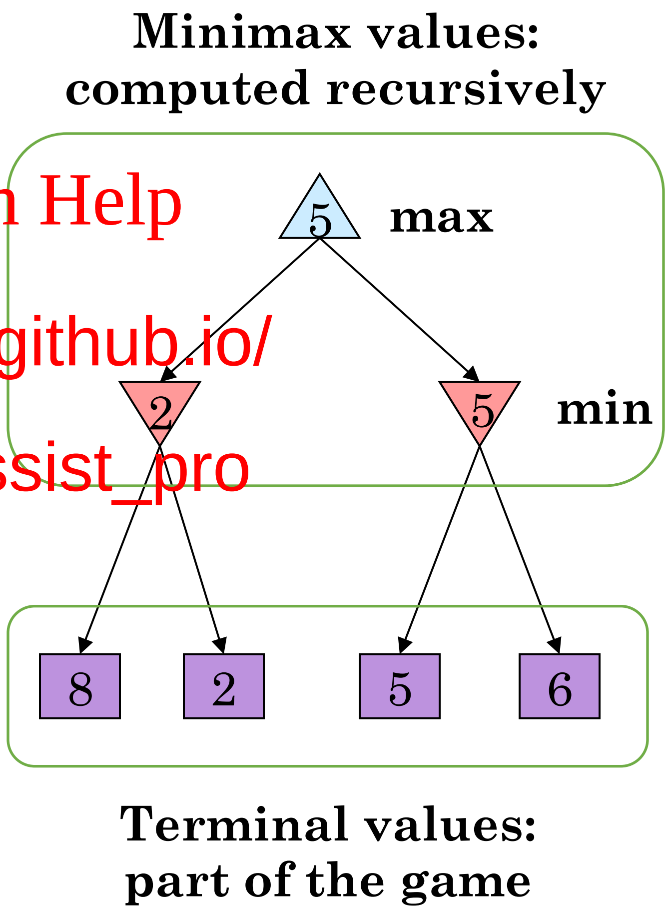
$$V(s) = \text{known}$$

Tic-Tac-Toe Game Tree

Adversarial Search (Minimax)

- Deterministic, zero-sum games:
  - Tic-tac-toe, chess, checkers
  - One player maximizes result
  - The other minimizes

- Minimax search:
  - A state-space search tree
  - Players alternate turns
  - Compute each node's minimax value: the best achievable utility against a rational (optimal) adversary



Minimax Implementation

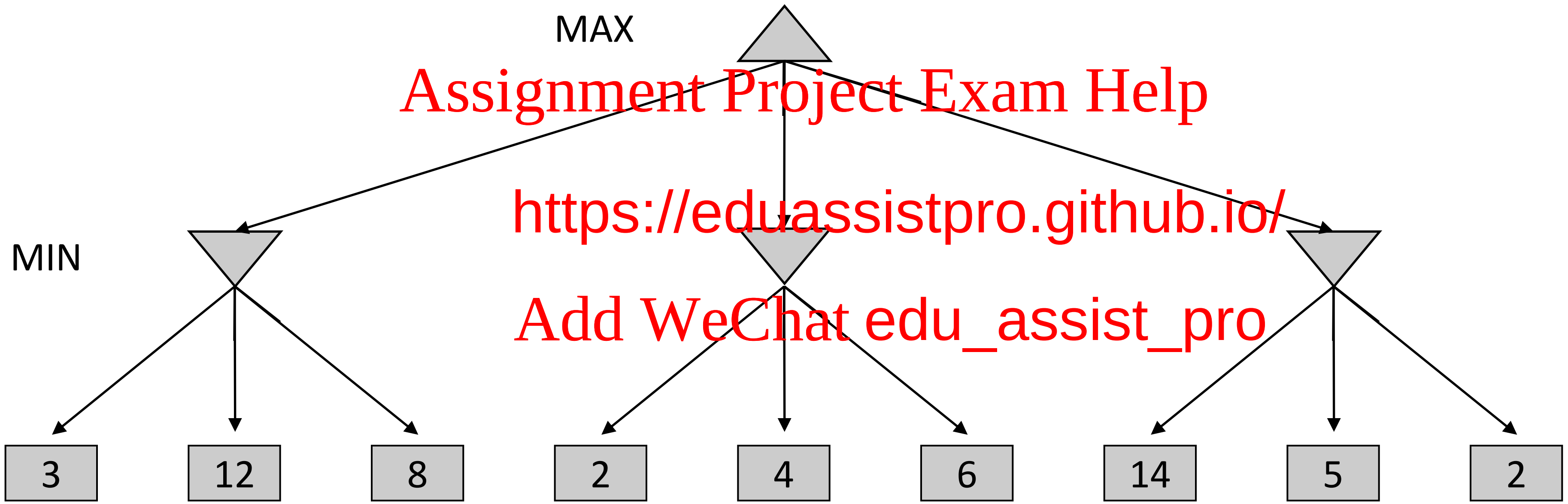
```
def value(state):
    if the state is a terminal state: return the state's utility
    if the next agent is MAX: return max-value(state)
    if the next age                         alue(state)
```



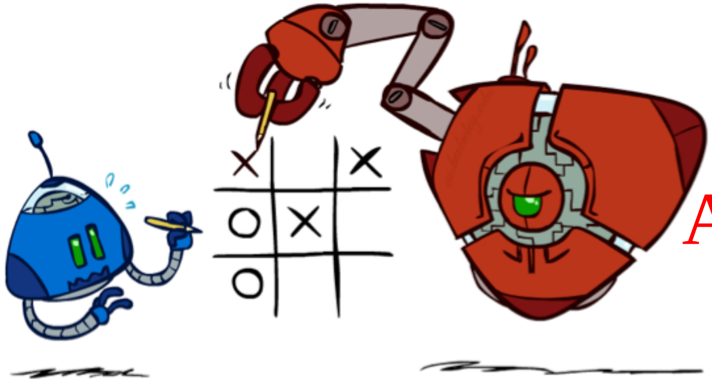
```
def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor))
    return v
```

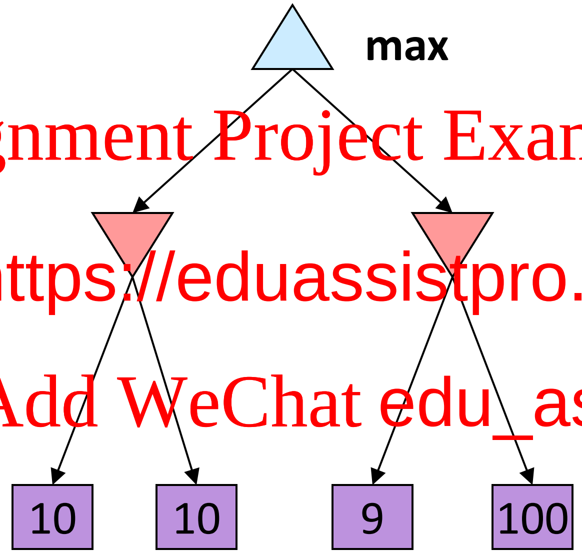
```
def min-value(state):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor))
    return v
```

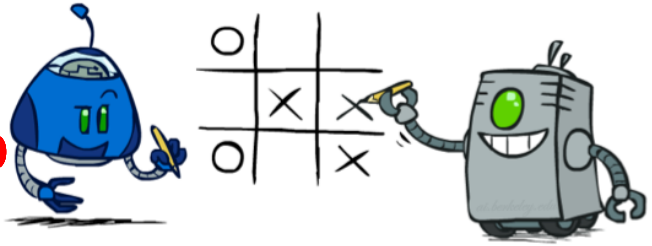
Minimax Example



Minimax Properties







Optimal against a perfect player. Otherwise?

Minimax Efficiency

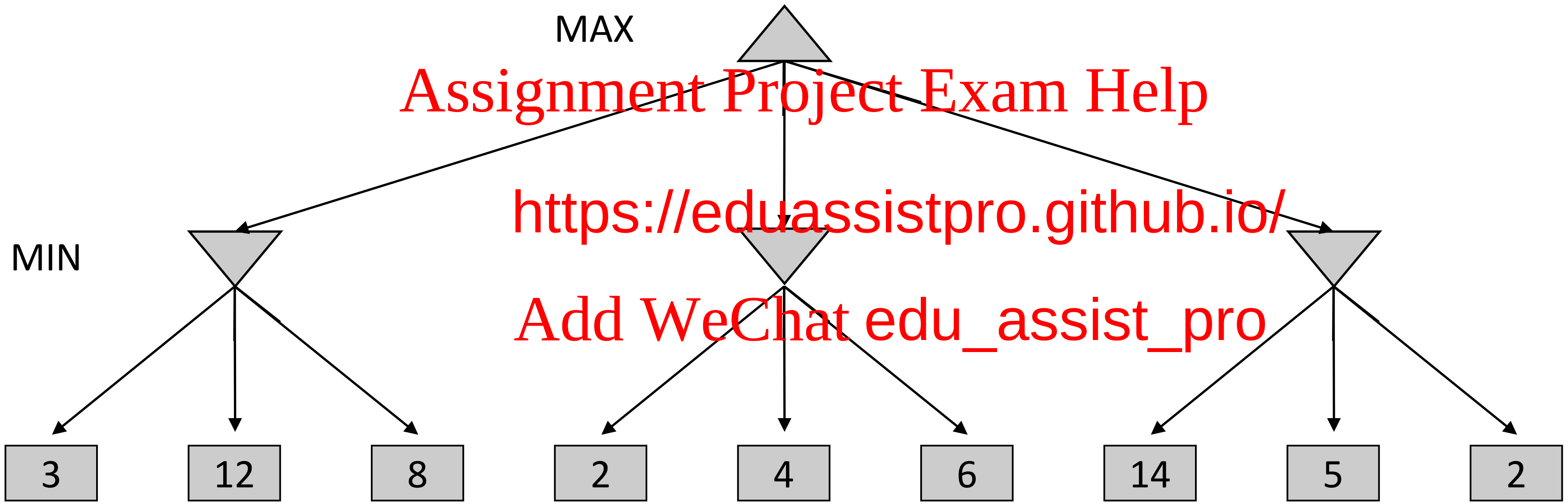
- How efficient is minimax?
  - Just like (exhaustive) DFS
  - Time: $O(b^m)$
  - Space: $O(bm)$

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

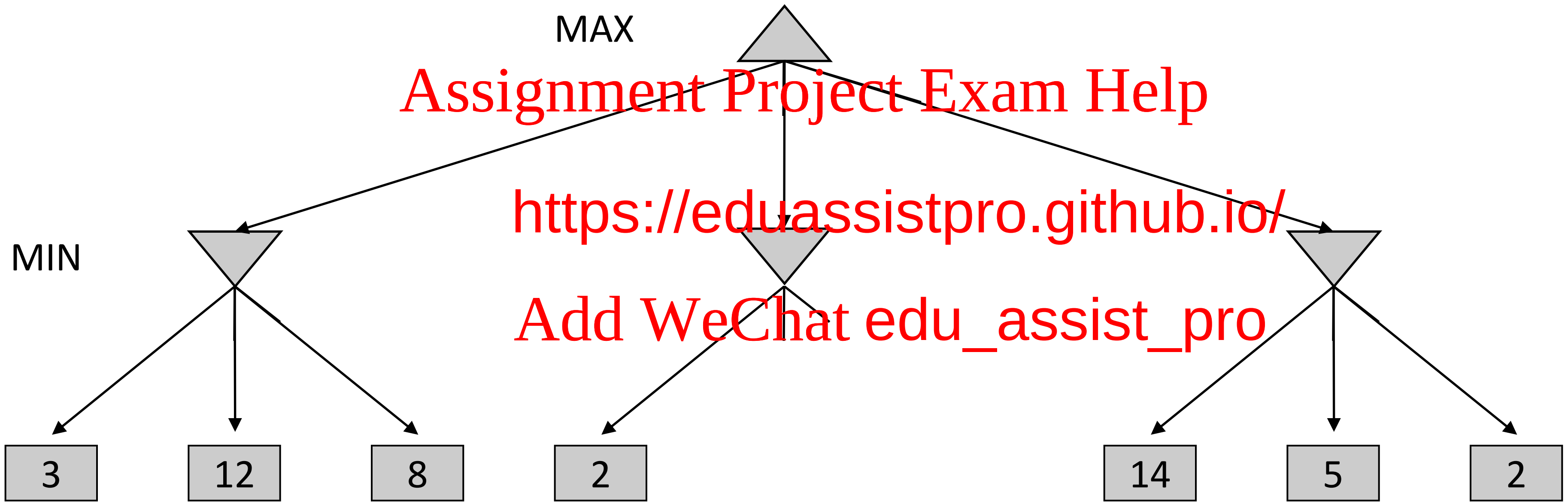
Resource Limits

Game Tree Pruning

Minimax Example



Minimax Pruning



Alpha-Beta Pruning

- **Alpha** α: value of the best choice so far for MAX (lower bound of Max utility)
- **Beta** β: value of the best choice so far for MIN (upper bound of Min utility)
- Expanding at MAX node **n**: update α

  - If a child of **n** has value great … he MAX node **n**

  - Reason: MIN parent of **n** wo … ich leads to **n**

- At MIN node **n**: update β
  - If a child of **n** has value less than α, stop expanding the MIN node **n**
  - Reason: MAX parent of **n** would not choose the action which leads to **n**

Alpha-Beta Implementation

```
def value(state, α, β):
    if the state is a terminal state: return the state's utility
    if the next agent is MAX: return max-value(state, α, β)
    if the next age                                    alue(state, α, β)
```



```
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
```

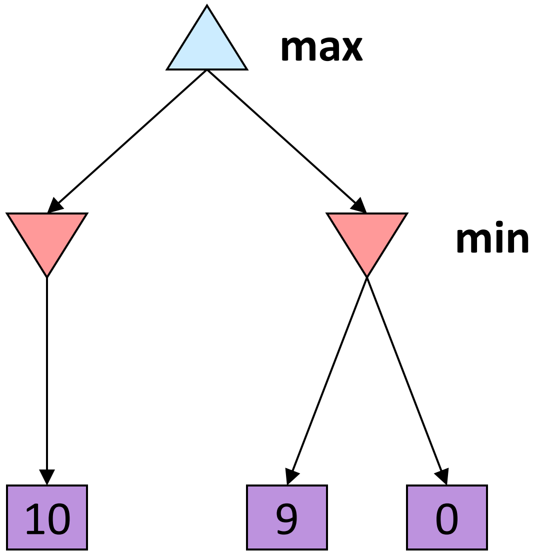
```
    lue(state , α, β):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor, α, β))
        if v ≤ α return v
        β = min(β, v)
    return v
```

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 ro e
  - So the most naïve version wo on

- Good child ordering improves effectiven ing



Alpha-Beta Quiz

max $[\alpha, \beta]$=$[-\infty, +\infty]$

min

Alpha-Beta Quiz

max $[\alpha, \beta] = [-\infty, +\infty]$

min $[\alpha, \beta] = [-\infty, +\infty]$

Alpha-Beta Quiz

max $[\alpha, \beta]=[-\infty, +\infty]$

min $[\alpha, \beta]=[-\infty, 10]$

Alpha-Beta Quiz

max $[\alpha, \beta] = [-\infty, +\infty]$

min $[\alpha, \beta] = [-\infty, 8]$

Alpha-Beta Quiz

max $[\alpha, \beta] = [-\infty, +\infty]$

min 8 $[\alpha, \beta] = [-\infty, 8]$

Alpha-Beta Quiz

max $[\alpha, \beta]$=$[8, +\infty]$

min 8 $[\alpha, \beta]$=$[-\infty, 8]$

Alpha-Beta Quiz

max $[\alpha, \beta]$=$[8, +\infty]$

min 8 $[\alpha, \beta]$=$[-\infty, 8]$ $[\alpha, \beta]$=$[8, +\infty]$

Alpha-Beta Quiz

max $[\alpha, \beta]$=$[8, +\infty]$

min 8 $[\alpha, \beta]$=$[-\infty, 8]$ $[\alpha, \beta]$=$[8, +\infty]$

Alpha-Beta Quiz

max $[\alpha, \beta]$=$[8, +\infty]$

min 8 $[\alpha, \beta]$=$[-\infty, 8]$ 4 $[\alpha, \beta]$=$[8, +\infty]$

Alpha-Beta Quiz 2

max

min

max

Alpha-Beta Quiz 2

max $[\alpha, \beta]$=$[-\infty, +\infty]$

min

max $[\alpha, \beta]$=$[-\infty, +\infty]$

Alpha-Beta Quiz 2

max $[\alpha, \beta]$=$[-\infty, +\infty]$

min

max $[\alpha, \beta]$=$[-\infty, +\infty]$

Alpha-Beta Quiz 2

max $[\alpha, \beta]=[-\infty, +\infty]$

min

max $[\alpha, \beta]=[10, +\infty]$

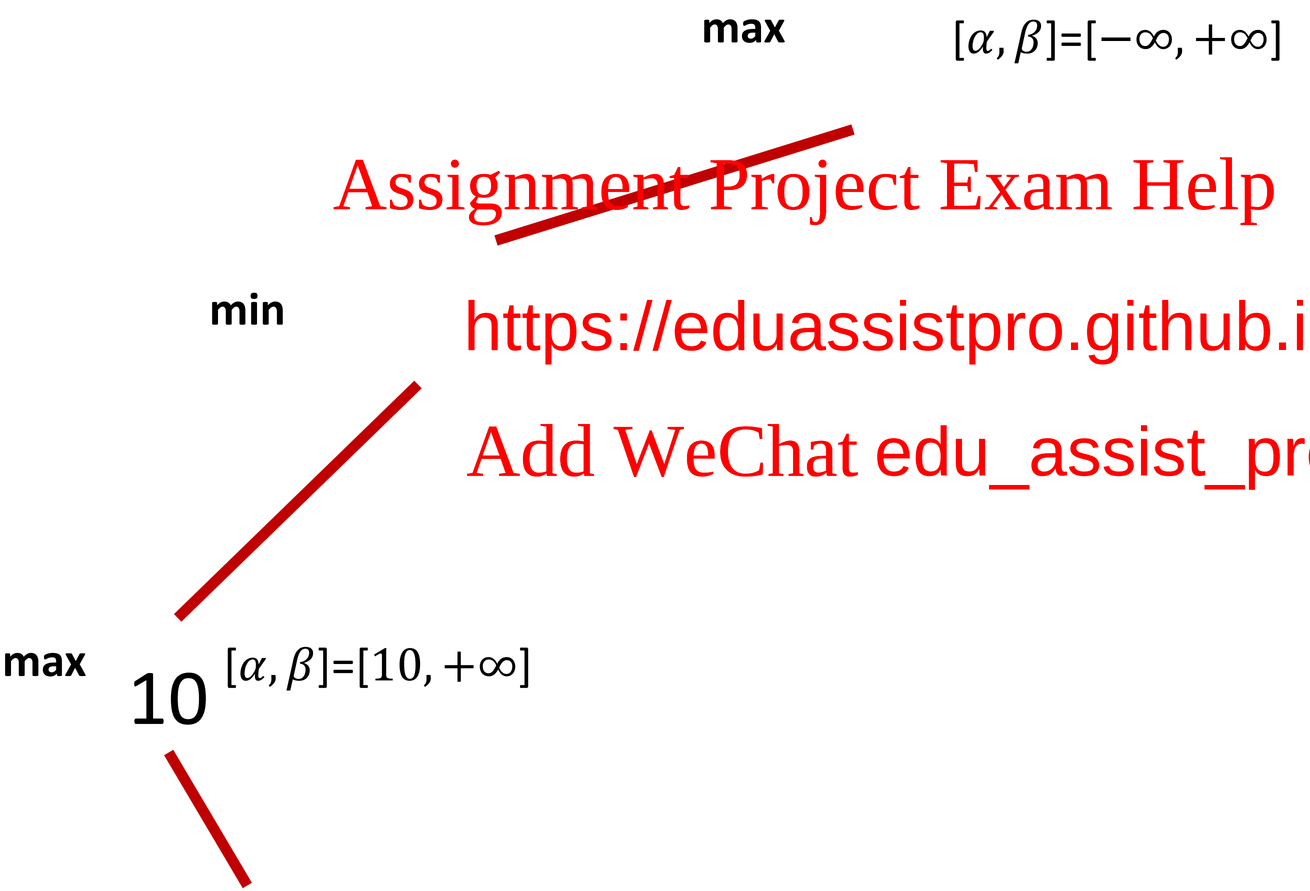
Alpha-Beta Quiz 2

max $[\alpha, \beta]$=$[-\infty, +\infty]$

min

max $[\alpha, \beta]$=$[10, +\infty]$

Alpha-Beta Quiz 2



Alpha-Beta Quiz 2

max $[\alpha, \beta]$=$[-\infty, +\infty]$

min

max 10 $[\alpha, \beta]$=$[10, +\infty]$

Alpha-Beta Quiz 2

max $[\alpha, \beta]$=$[-\infty, +\infty]$

min

max 10 $[\alpha, \beta]$=$[10, +\infty]$ $[\alpha, \beta]$=$[-\infty, 10]$

Alpha-Beta Quiz 2

max $[\alpha, \beta]$=$[-\infty, +\infty]$

min

max 10 $[\alpha, \beta]$=$[10, +\infty]$ $[\alpha, \beta]$=$[-\infty, 10]$

Alpha-Beta Quiz 2

max $[\alpha, \beta]=[-\infty, +\infty]$

min

max 10 $[\alpha, \beta]=[10, +\infty]$ 100 $[\alpha, \beta]=[-\infty, 10]$

Alpha-Beta Quiz 2

max $[\alpha,\beta]=[-\infty,+\infty]$

min 10

max 10 $[\alpha,\beta]=[10,+\infty]$ 100 $[\alpha,\beta]=[-\infty,10]$

Alpha-Beta Quiz 2

max $[\alpha, \beta]$=$[10, +\infty]$

min 10 $[\alpha, \beta]$=$[10, +\infty]$

max 10 $[\alpha, \beta]$=$[10, +\infty]$ 100 $[\alpha, \beta]$=$[-\infty, 10]$ $[\alpha, \beta]$=$[10, +\infty]$

Alpha-Beta Quiz 2

max $[\alpha, \beta]$=$[10, +\infty]$

min 10 $[\alpha, \beta]$=$[10, +\infty]$

max 10 $[\alpha, \beta]$=$[10, +\infty]$ 100 $[\alpha, \beta]$=$[-\infty, 10]$ $[\alpha, \beta]$=$[10, +\infty]$

Alpha-Beta Quiz 2

max $[\alpha, \beta]$=$[10, +\infty]$

min 10 $[\alpha, \beta]$=$[10, +\infty]$

max 10 $[\alpha, \beta]$=$[10, +\infty]$ 100 $[\alpha, \beta]$=$[-\infty, 10]$ $[\alpha, \beta]$=$[10, +\infty]$

Alpha-Beta Quiz 2

max $[\alpha, \beta]$=$[10, +\infty]$

min 10 $[\alpha, \beta]$=$[10, +\infty]$

max 10 $[\alpha, \beta]$=$[10, +\infty]$ 100 $[\alpha, \beta]$=$[-\infty, 10]$ 2 $[\alpha, \beta]$=$[10, +\infty]$

Alpha-Beta Quiz 2

max $[\alpha, \beta]$=$[10, +\infty]$

min 10 2 $[\alpha, \beta]$=$[10, +\infty]$

max 10 $[\alpha, \beta]$=$[10, +\infty]$ 100 $[\alpha, \beta]$=$[-\infty, 10]$ 2 $[\alpha, \beta]$=$[10, +\infty]$

Alpha-Beta Quiz 2

max 10 $[\alpha, \beta]=[10, +\infty]$

min 10 2 $[\alpha, \beta]=[10, +\infty]$

max 10 $[\alpha, \beta]=[10, +\infty]$ 100 $[\alpha, \beta]=[-\infty, 10]$ 2 $[\alpha, \beta]=[10, +\infty]$

Resource Limits

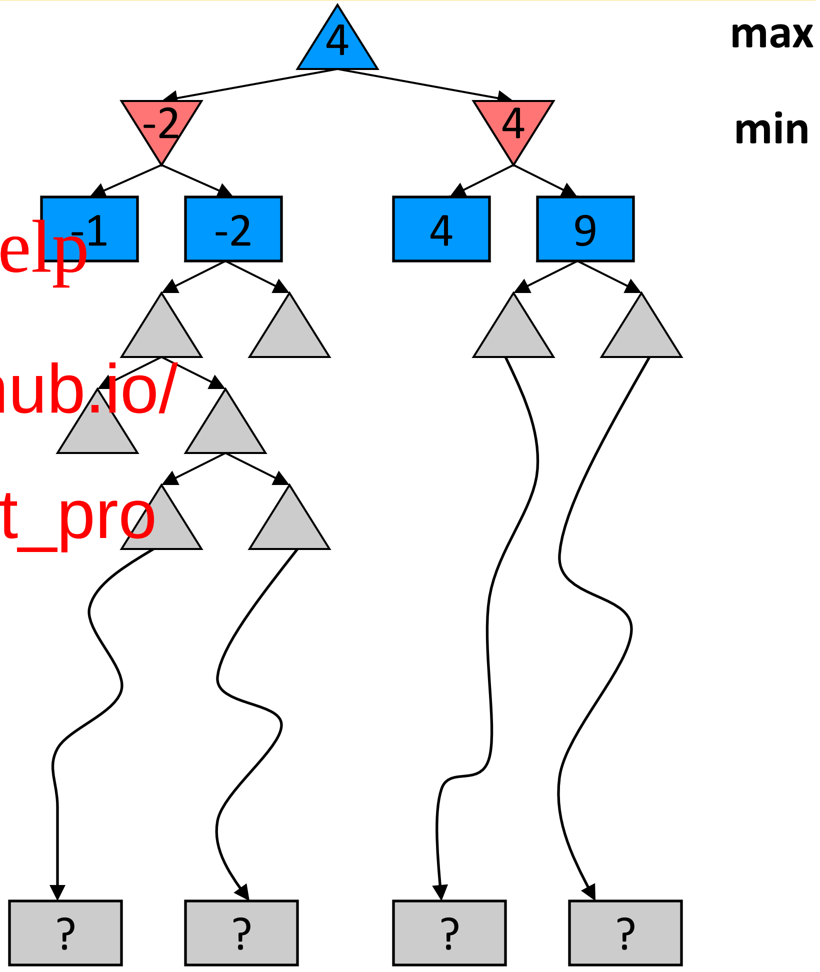
Resource Limits

- Problem: In realistic games, cannot search to leaves!

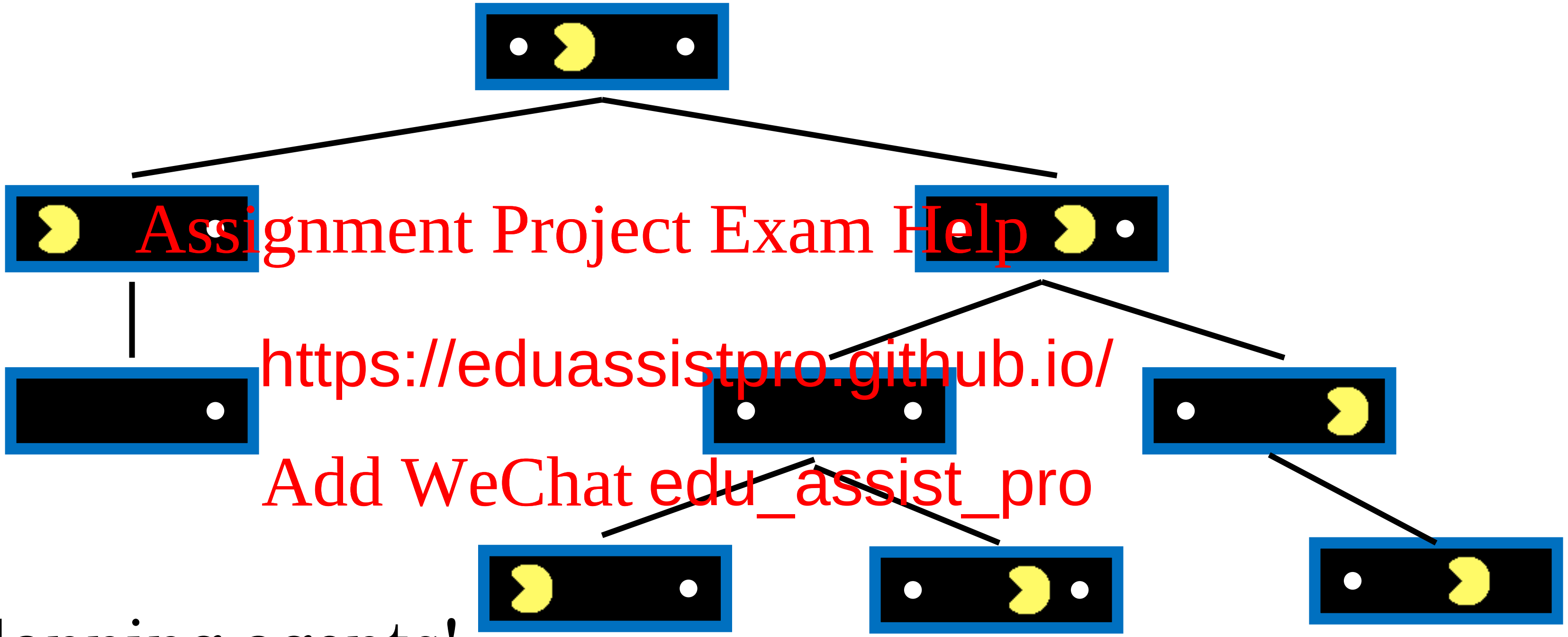
- Solution: Depth-limited search
  - Instead, search only to a limited depth in the tree
  - Replace terminal utilities with an e
    terminal positions

- Example:
  - Suppose we have 100 seconds, can explore 10K nodes /
  - 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



Why Pacman Starves

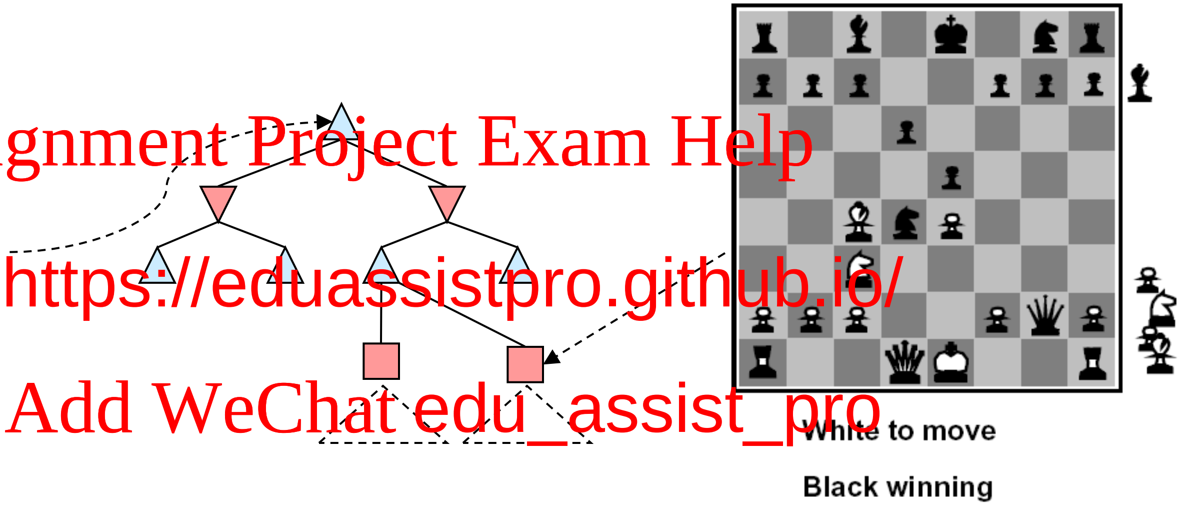


- A danger of replanning agents!
  - He knows his score will go up by eating the dot now (west, east)
  - He knows his score will go up just as much by eating the dot later (east, west)
  - There are no point-scoring opportunities after eating the dot (within the horizon, two here)
  - Therefore, waiting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

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

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

Evaluation for Pacman

Depth Matters

- Evaluation functions are always imperfect
- The deeper in the tree the evaluation function is the less the quality of evaluation function matters
- An important example of the tradeoff between complexity of features and complexity of computation

[Demo: depth limited (L6D4, L6D5)]

Synergies between Evaluation Function and Alpha-Beta?

- Alpha-Beta: amount of pruning depends on expansion ordering
  - Evaluation function can provide guidance to expand most promising nodes first (which later makes it more likely there is already a good alternative on the path to the root)
    - (somewhat similar to role of)

- Alpha-Beta: (similar for roles of min ed)
  - Value at a min-node will only keep going down
  - Once value of min-node lower than better option for max along path to root, can prune
  - Hence: IF evaluation function provides upper-bound on value at min-node, and upper-bound already lower than better option for max along path to root THEN can prune