Adversarial Search

CSE 4617: Artificial Intelligence

Isham Tashdeed
Lecturer, CSE

AI in Games

- 1950: First computer player
- 1994: Computer beats the best human player
- 2007: Checkers is solved !

AI in Games

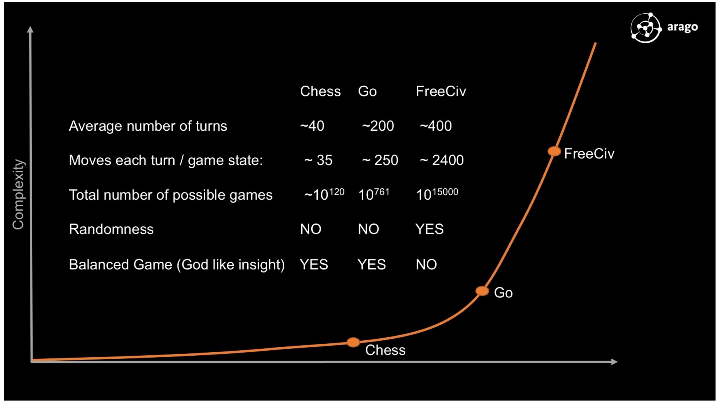
Me after playing a game with Stockish 15

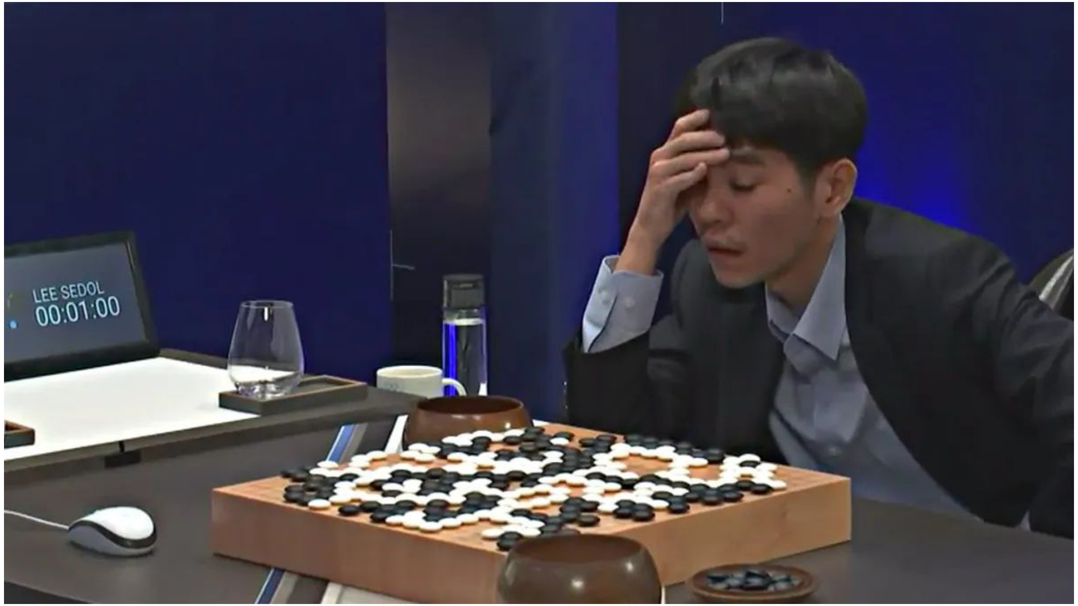
- 1997: Deep Blue defeats human champion
- Chess is still unsolved

"Inevitably the machines must win, but there is still a long way to go before a human on his or her best day is unable to defeat the best computer."

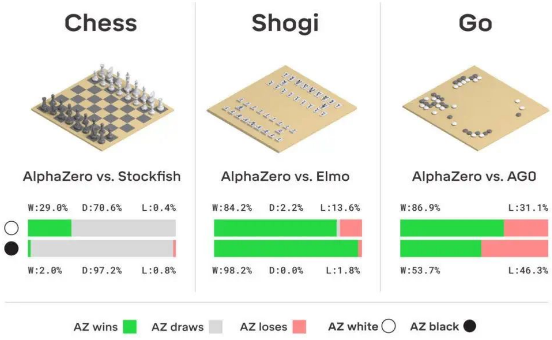
— Garry Kasparov

AI in Games





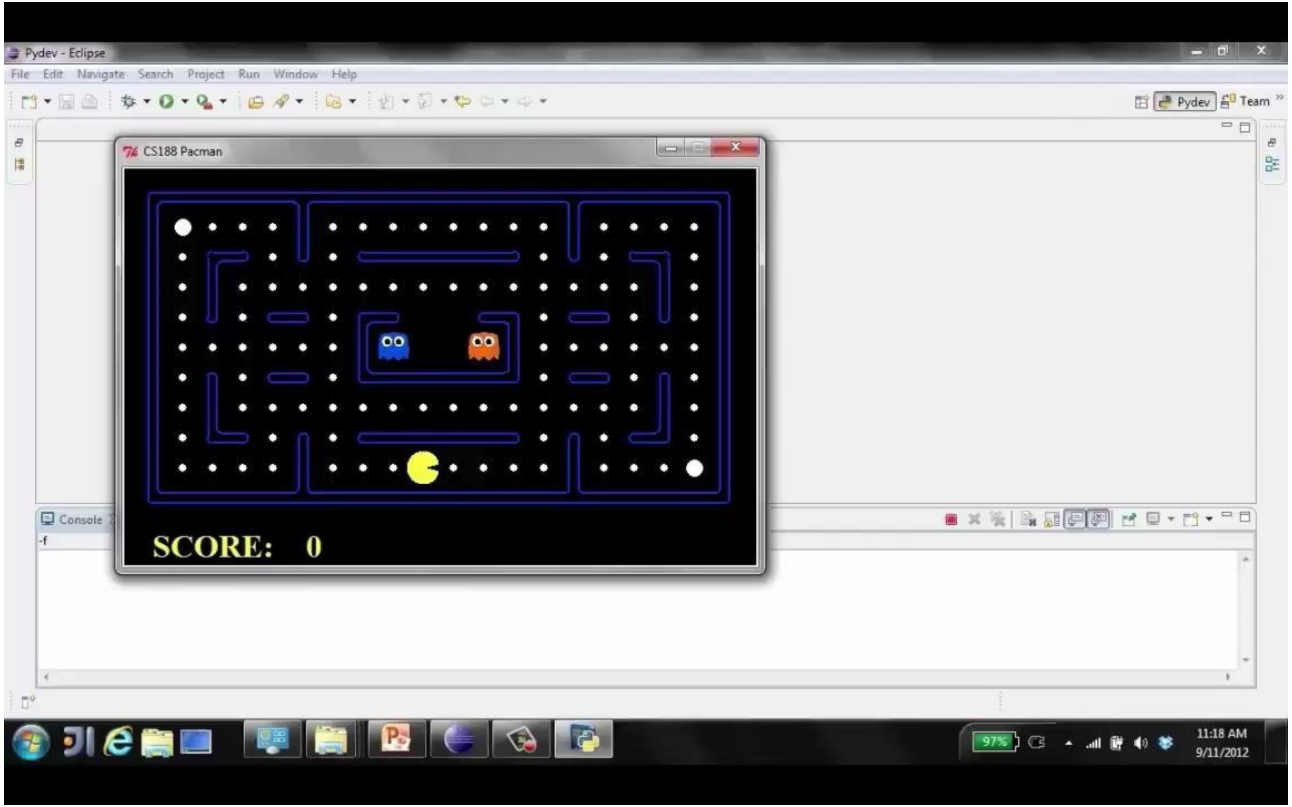
- 2010s: Human champions are now starting to be challenged by machines
- 2016: Alpha GO defeats human champion
- 2017: Introduced alpha zero



AI in Games


30:44 BAR 0-1 PIE
7 GRIEZMANN
1 SZCZĘSNY
DON'T @ ME
Show:
Boxes
Heads
Bones
Armor
Health
FOV Percentage

AI in Games


Pydev - Eclipse
File Edit Navigate Search Project Run Window Help
Pydev
Team
CS188 Pacman
SCORE: 0
Console
11:18 AM
9/11/2012

Types of Games

Types of Games

- Deterministic or Stochastic?
- One, Two or more players?
- Zero sum game?
- Perfect information of states?

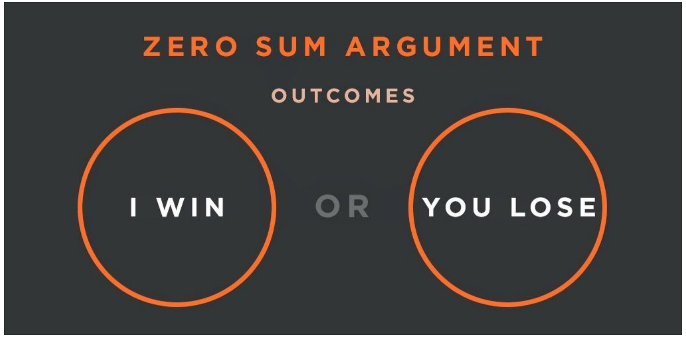
Want algorithms for calculating a strategy (policy) which recommends a move from each state

Solved Game → Can force a draw if both players play optimally

Types of Games

Zero-sum Games:

- Agents have opposite utilities (opposite goals)
- Lets us think of a single value that one maximizes and the other minimizes
- Adversarial, pure competition



General Games:

- Agents have independent utilities
- Cooperation, indifference, competition, and more are all possible
- More later on non-zero-sum games

Formalizing Deterministic Games

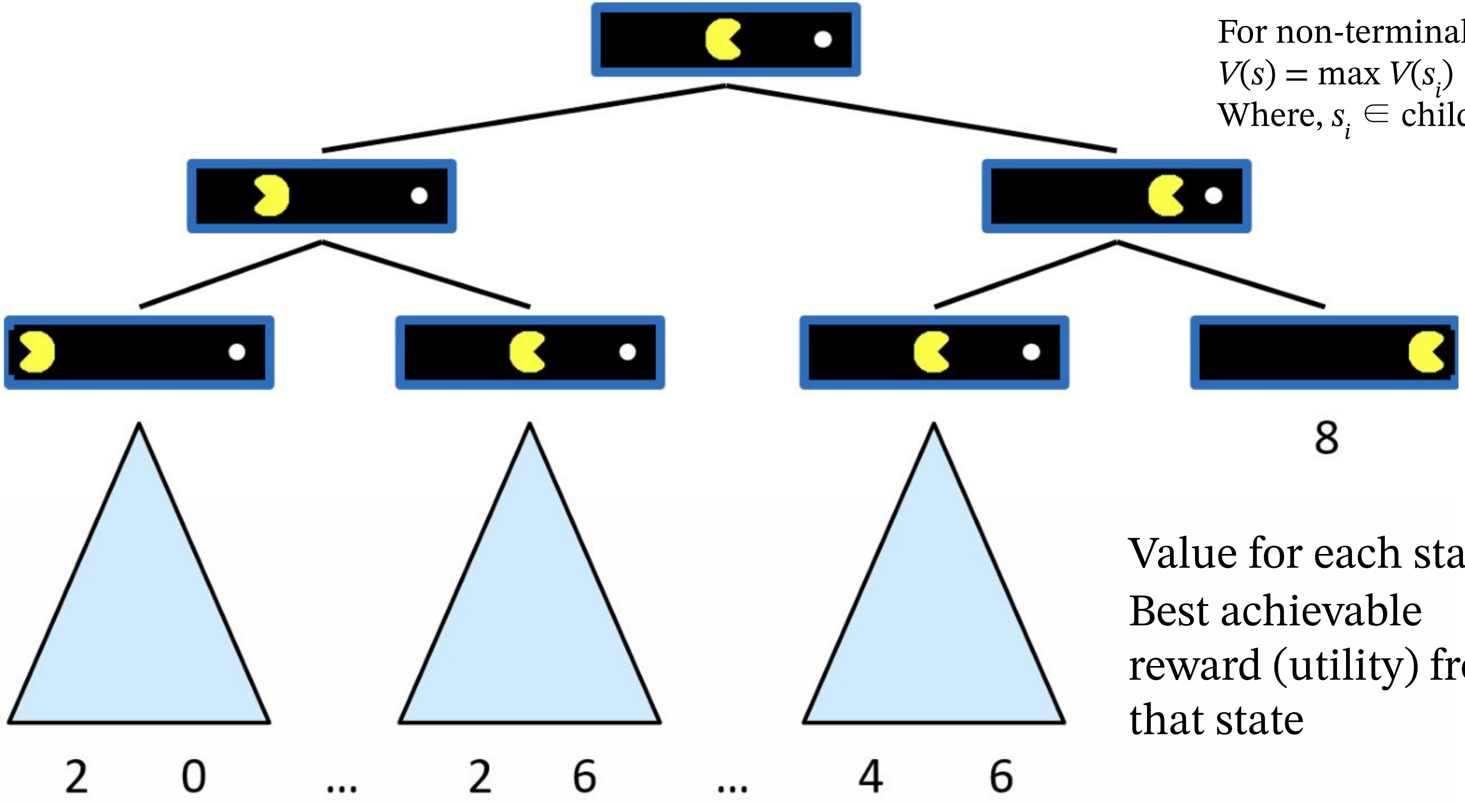
- States → S (Start state = s_0)
- Players → P = {1, 2, …, N} (Usually takes turns)
- Actions → A (Similar to search)
  - May depend on the player/state
- Transition Function → Similar to successor function
- Terminal Test → S = {*true*, *false*}
  - Example terminal states: Win, Lose, Draw
- Terminal Utilities → Assign certain rewards/scores to all outcomes of a game
- Solution for a player is to find a policy

Adversarial Search

Single-Agent Tree

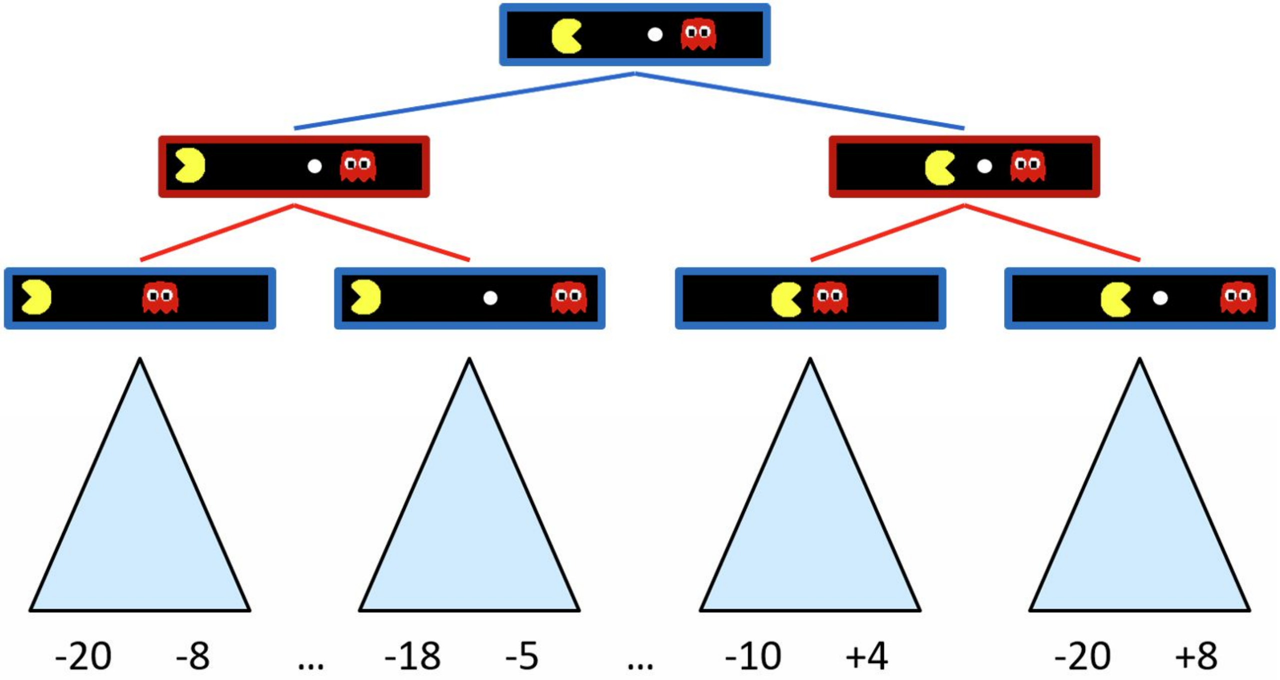
For terminal states:
$V(s)$ = known value

For non-terminal states:
$V(s) = \max V(s_i)$
Where, $s_i \in \text{children}(s)$

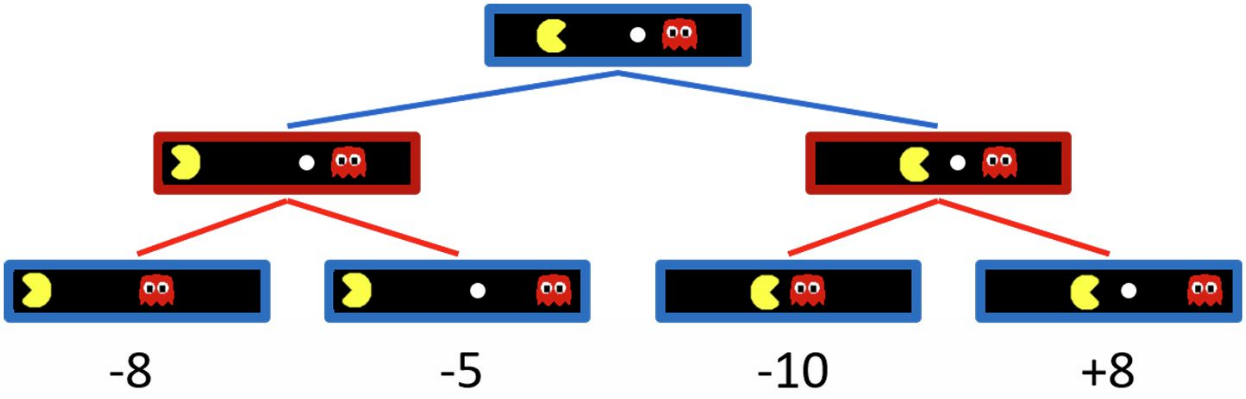


Value for each state:
Best achievable reward (utility) from that state

Adversarial Search Tree



Adversarial Search Tree

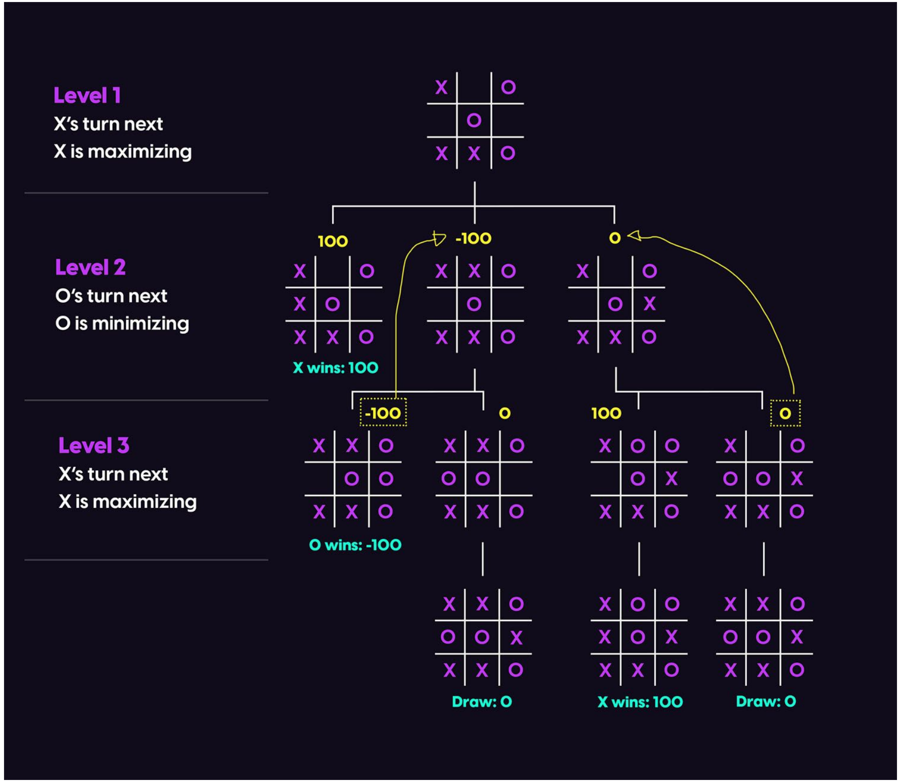


For terminal states:
$V(s) = \text{known value}$

States under opponent's control:
$V(s) = \min V(s_i)$
Where, $s_i \in \text{children}(s)$

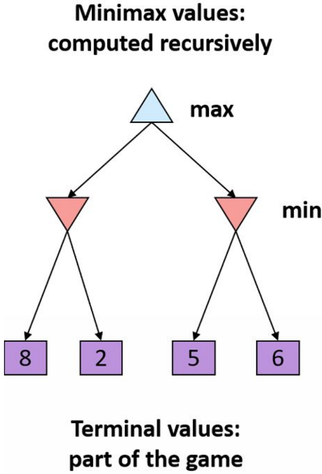
States under agent's control:
$V(s) = \max V(s_i)$
Where, $s_i \in \text{children}(s)$

Adversarial Search



Deterministic, Zero-sum games:

- Tic-tac-toe, chess, checkers…
- One player maximizes result
- The other minimizes result



Minimax Search:

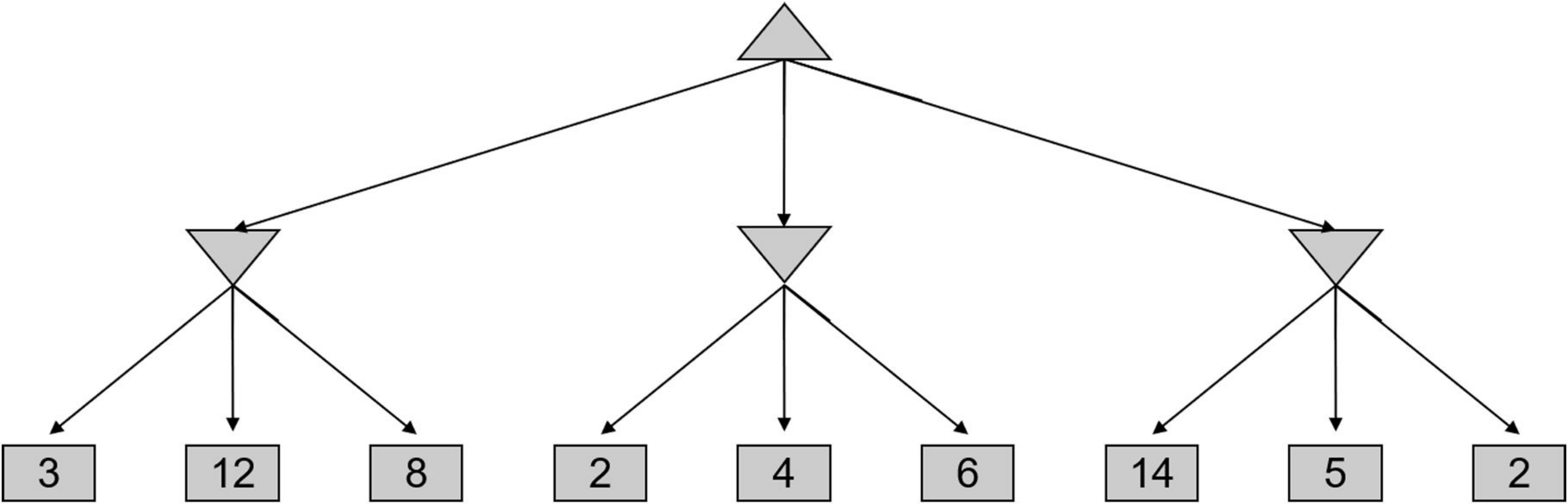
- A state-space search tree
- Players alternate turns
- Compute each node's minimax value → The best achievable utility against a rational (optimal) adversary

Minimax Algorithm

```
VALUE (state) → returns utility value of the state
    if state is TERMINAL: then return utility
    if state is MAX-AGENT: then return MAX-VALUE (state)
    if state is MIN-AGENT: then return MIN-VALUE (state)


MAX-VALUE (state) → returns utility value of a state
    v ← −∞
    for successor in state:
        v = max (v, VALUE (successor))
    return v


MIN-VALUE (state) → returns utility value of a state
    v ← +∞
    for successor in state:
        v = min (v, VALUE (successor))
    return v
```

What is the complexity?

Minimax Algorithm



Formalizing Deterministic Games

What is the time & space complexity of minimax?

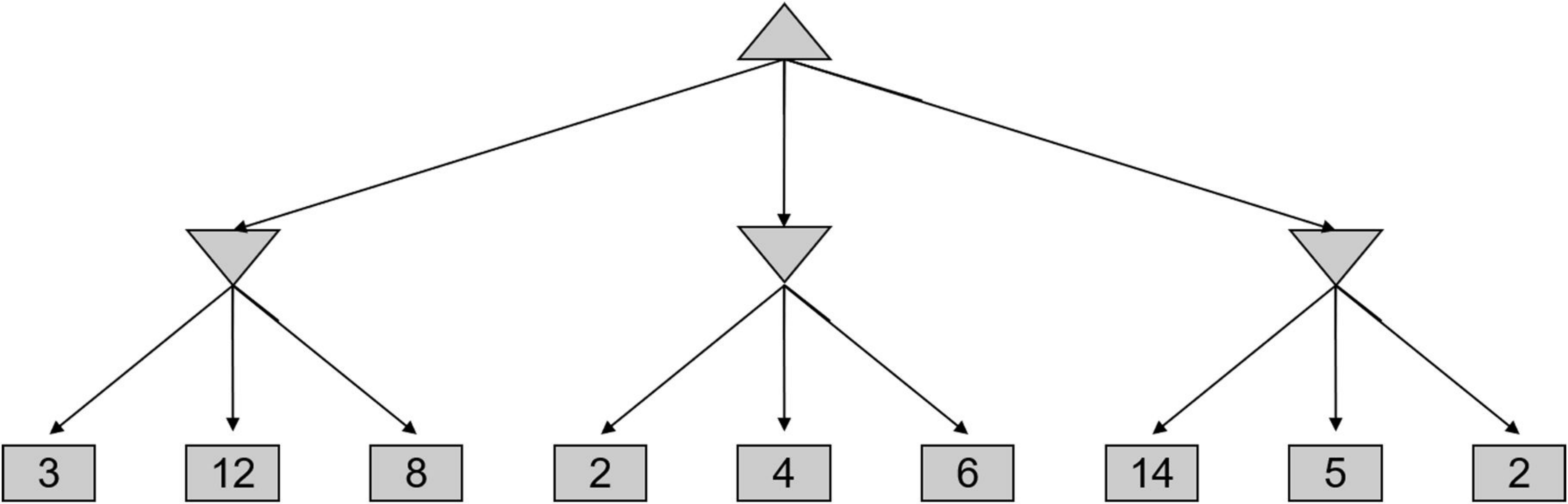
- Similar to DFS
- Time complexity: $O(b^m)$
- Space complexity: $O(bm)$

Here, $b \rightarrow$ branching factor $m \rightarrow$ depth

For chess, $b \approx 35$ and $m \approx 100 \rightarrow$ Finding exact utilities from terminal states is impossible

Game Tree Pruning

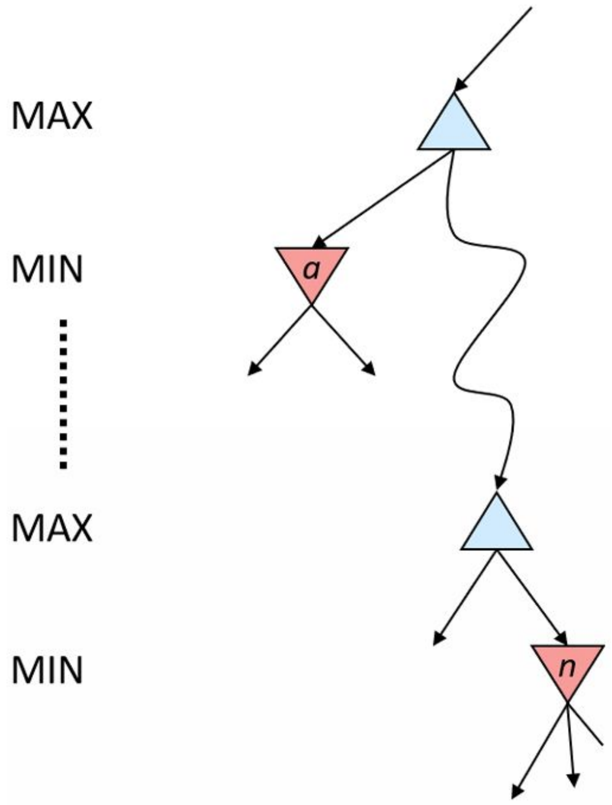
Game Tree Pruning



Alpha-Beta Pruning

From the perspective of MIN:

- Computing the MIN-VALUE at some node n
- Looping over the children of n
- N's estimate of the children's minimum is always dropping
- Let α be the best value that MAX can get at any choice point along the current path from the root
- If n becomes lower than α, MAX will avoid it, so stop considering n's other children.



Alpha-Beta Pruning

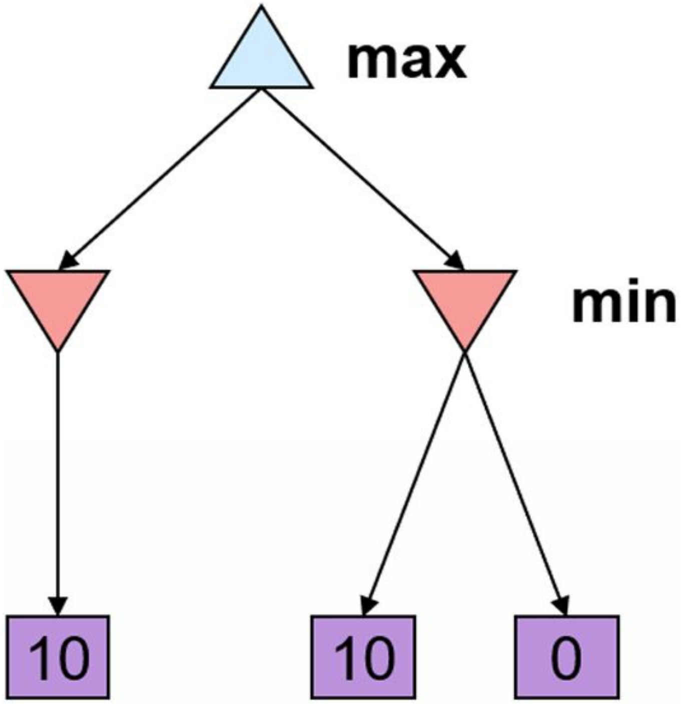
```
VALUE (state, α, β) → returns utility value of the state
    if state is TERMINAL: then return utility
    if state is MAX-AGENT: then return MAX-VALUE (state, α, β)
    if state is MIN-AGENT: then return MIN-VALUE (state, α, β)

MAX-VALUE (state, α, β) → returns utility value of a state
    v ← −∞
    for successor in state:
        v = max (v, VALUE (successor, α, β))
        if v >= β: then return v
        α = max (α, v)
    return v

MIN-VALUE (state, α, β) → returns utility value of a state
    v ← +∞
    for successor in state:
        v = min (v, VALUE (successor, α, β))
        if v <= α: then return v
        β = min (β, v)
    return v
```

$\alpha \longrightarrow$ MAX's best option on path to root

$\beta \longrightarrow$ MIN's best option on path to root

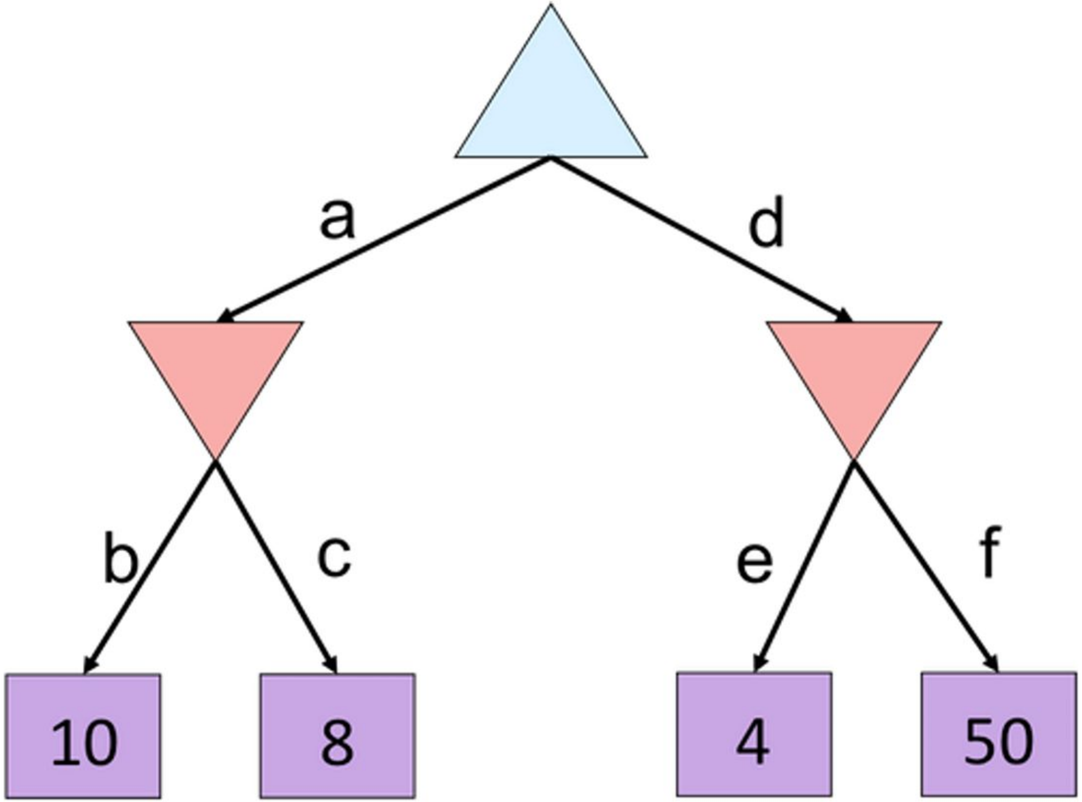
Alpha-Beta Pruning

- Pruning has **no effect** on the minimax value at the root → Why?
- Values of the intermediate nodes may be wrong
  - Prune only on inequality → Not the best
  - Keep track of which one was first
- Good child ordering improves effectiveness
  - Explore good options before bad ones
- With perfect ordering:
  - Time complexity: $O(b^{m/2})$
  - Solvable depth doubles
  - Still no enoch for chess
  - This is an example of meta-reasoning

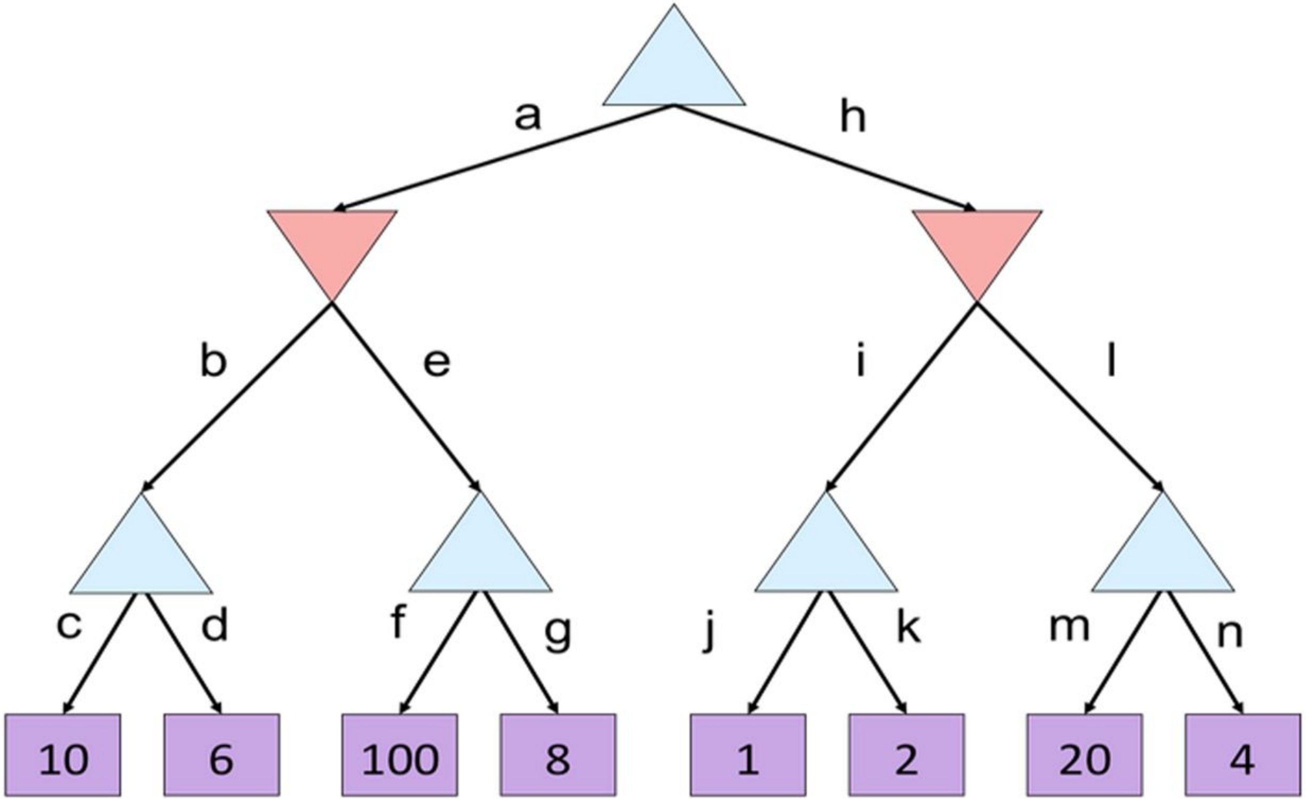


Mini Quiz

Mini Quiz



Mini Quiz



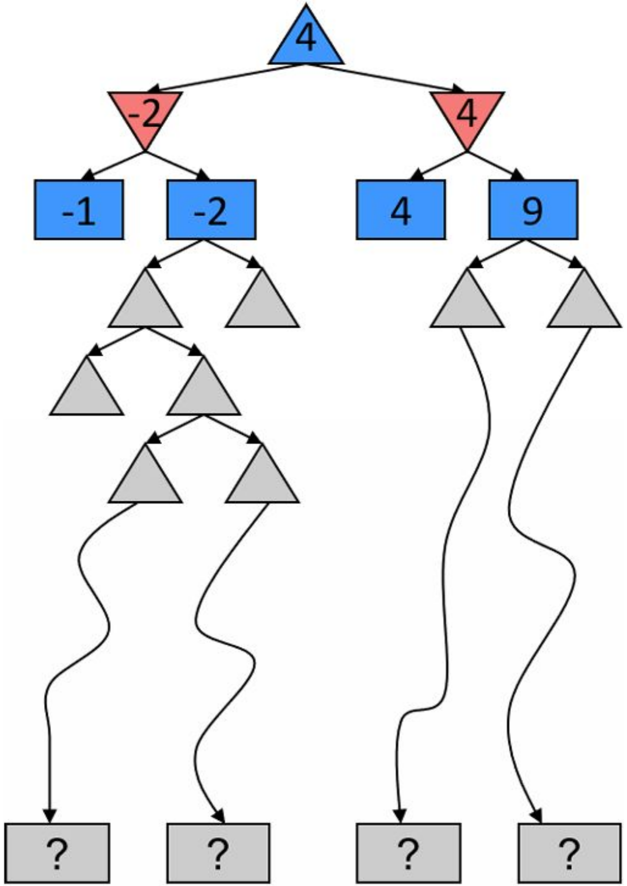
Resource Limits

Resource Limits

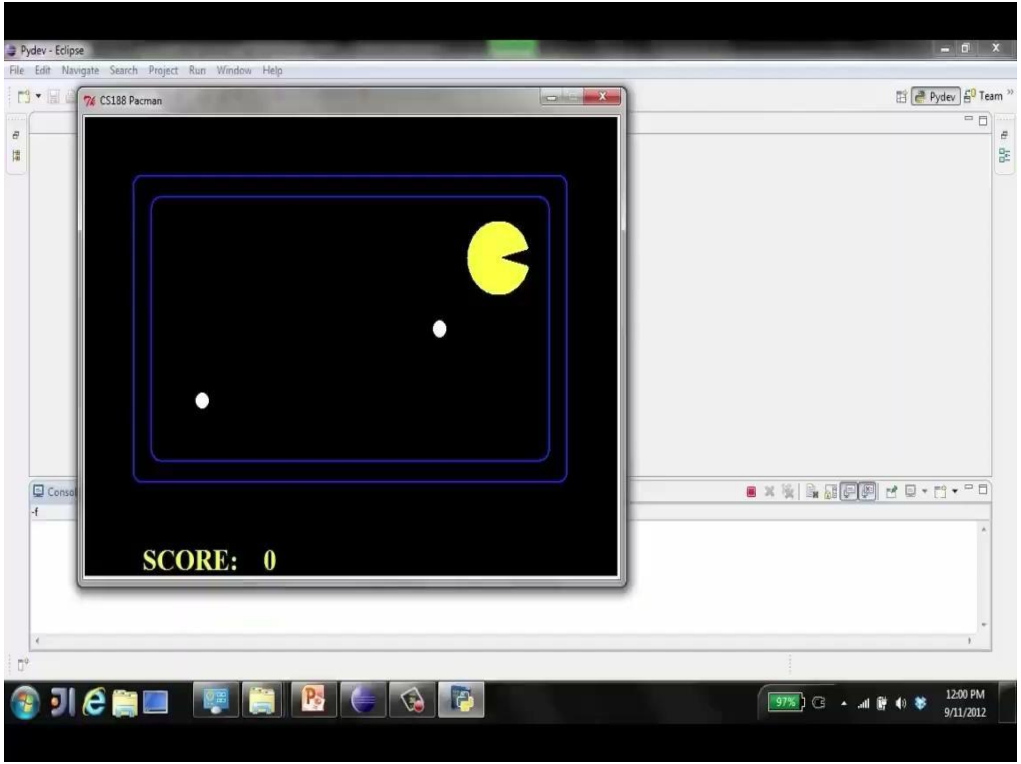
In demanding games like chess, we can not search to the leaves!

Depth-limited Search:

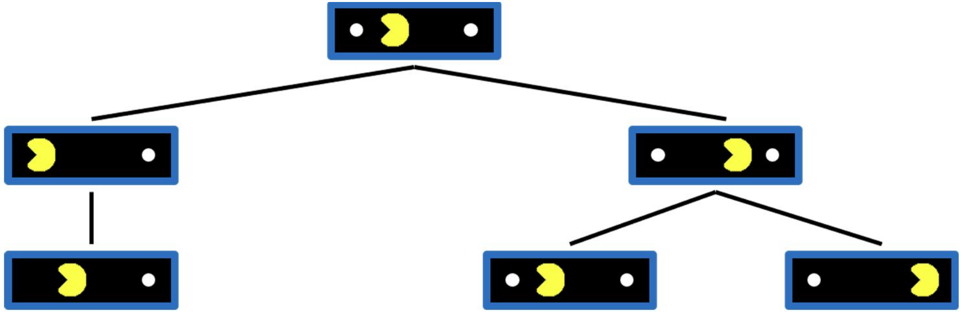
- Search only to a limited depth in the tree
- Replace terminal utilities with an evaluation function for non-terminal positions
- Guarantee of optimal play is gone
- The more depth you check, the better your moves become
- Use iterative deepening or a better outcome



Resource Limits



Adversarial Search

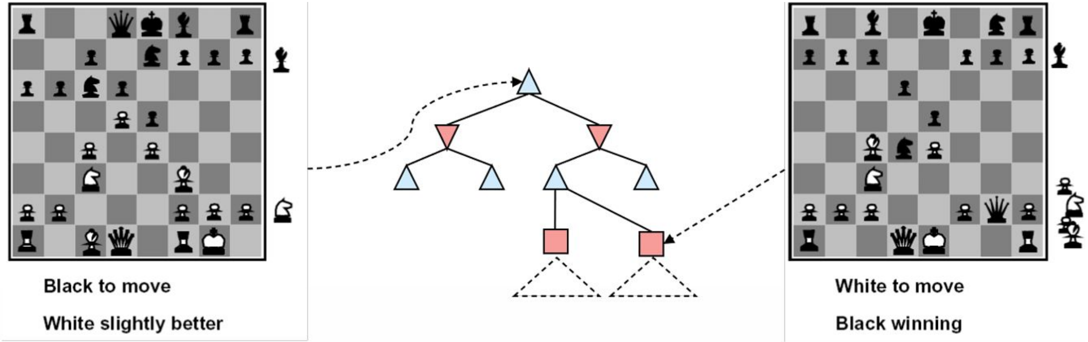


Dangers of replanning:

- 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 limited depth)
- Therefore, waiting seems just as good as eating: he may go east, then back west in the next round of replanning!

Evaluation Function → Used to score non-terminals in depth-limited search

Evaluation Function



- Ideally → returns the actual minimax value of the position
- In practice → typically weighted linear sum of features:
  - $\text{Eval}(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots$
- Evaluation functions are always 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

Additional Resources

- Why is It Difficult to Make Good AI for Games?
- AlphaGo - The Movie
- Training AI to Play Pokemon with Reinforcement Learning
- The Evaluation Function in Board Games

Thank you