Adversarial Search
Good morning colleagues!

- Past due:
  - HW1 - Search
  - 5 reading responses: AI100 report; 4 Textbook readings

- Upcoming EdX Homeworks
  - HW2: CSPs - due Monday 2/15 at 11:59 pm
  - HW3: Games – due Monday 2/22 at 11:59 pm

- Upcoming programming projects
  - P1: Search - due Wednesday 2/10 at 11:59 pm
  - P2: Games – due Wednesday 2/24 at 11:59 pm

- Readings: Markov Decision Processes
  - **NOT** just the usual textbook
  - Due Monday 2/15 at 9:30 am
Game Playing State-of-the-Art

- **Checkers**: 1950: First computer player. 1994: First computer champion: Chinook ended 40-year-reign of human champion Marion Tinsley using complete 8-piece endgame. 2007: Checkers solved!

- **Chess**: 1997: Deep Blue defeats human champion Gary Kasparov in a six-game match. Deep Blue examined 200M positions per second, used very sophisticated evaluation and undisclosed methods for extending some lines of search up to 40 ply. Current programs are even better, if less historic.

- **Go**: 2016: AlphaGo, created by Google DeepMind beat 9-dan professional Go player Lee Sedol 4-1 on a full sized 19 x 19 board. AlphaGo combined Monte Carlo Tree Search with deep neural networks, improving via reinforcement learning through self-play.

- **OpenAI Five (DOTA)**: getting close to world-class
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
Many different kinds of games!

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

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

- Zero-sum game
- Practice problem in breakout rooms
- Work for a couple of minutes independently, but then quickly start comparing progress – even if you’re not done yet.
Tic-Tac-Toe Game Tree
### Adversarial Search (Minimax)

- **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 Search Diagram](image-url)

**Minimax values:** computed recursively

**Terminal values:** part of the game
def max-value(state):
    initialize v = -∞
    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 = +∞
    for each successor of state:
        v = min(v, max-value(successor))
    return v

V(s') = \min_{s \in \text{successors}(s')} V(s)
Minimax Implementation (Dispatch)

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 = -∞
    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 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?
Minimax Properties

Optimal against a perfect player. Otherwise?
Minimax vs Expectimax (Min)

End your misery!
Minimax vs Expectimax (Exp)

Hold on to hope, Pacman!
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
  - \( \alpha-\beta \) 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
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:

\[
Eval(s) = w_1f_1(s) + w_2f_2(s) + \ldots + w_nf_n(s)
\]

- e.g. \( f_1(s) = \text{(num white queens} - \text{num black queens)} \), etc.
Some of Your Questions

- Difference between utility function; evaluation function; heuristic
- Are there pruning methods other than alpha-beta?
- How does one come up with a utility function? (Lorenzo Martinez)
  - Can an agent learn its own utility function? (Theodore Venter)
- Maximize expected utility vs. sacrificing players (Michael Rodriguez-Labarca)
- What should you do against non-optimal opponents? (Kelsey Zhan)
- CSPs vs. game trees (Daniel Kim)
- How do we arrive at different difficulty levels in online games? (Omar Dadabhoy)
- 1950s algorithms – so why so long to beat humans at chess? (Thomas Norman)
- Game trees possible for games more complex than chess? (Colette Montminy)
Minimax Example
Minimax Pruning
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
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

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

α: MAX's best option on path to root
β: MIN's best option on path to root
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!
  - Full search of, e.g. chess, is still hopeless...

- This is a simple example of **metareasoning** (computing about what to compute)
Next Time: Uncertainty!