CSE 40171: Artificial Intelligence

Adversarial Search: Games and Optimality
What is a game?
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, a deep learning-based system, beat Lee Sedol, a 9-dan professional without handicaps, in a five game match. The win was a major milestone in data driven approaches to game playing.

**Pacman**
Behavior from Computation

Image credit: Dan Klein and Pieter Abbeel, UC Berkeley CS 188
Adversarial Games
Types of 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.
Formal Elements of a Game

- $S_0$: the **initial state**, which specifies how the game is set up at the start
- PLAYER($s$): Defines which player has the move in a state
- ACTIONS($s$): Returns the set of legal moves in a state
- RESULT($s, a$): the **transition model**, which defines the result of a move
Formal Elements of a Game

- TERMINAL-TEST(s): a terminal test, which is true when the game is over and false otherwise. States where the game has ended are called terminal states.

- UTILITY(s, p): a utility function (a.k.a. objective or payoff function) defines the final numeric value for a game that ends in terminal state $s$ for a player $p$. 
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
Two Players

**MAX**
- Moves first
- High values are good for MAX

**MIN**
- Moves after MAX
- High values are bad for MIN
Optimal Decisions in Games
What is different about this compared to basic search?
Adversarial Search
Single-Agent Trees
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:

\[ V(s) = \text{known} \]

Slide credit: Dan Klein and Pieter Abbeel, UC Berkeley CS 188
Adversarial Game Trees

Slide credit: Dan Klein and Pieter Abbeel, UC Berkeley CS 188
Minimax Values

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

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

States Under Agent’s Control:
-8, -5, -10

States Under Opponent’s Control:
+8

Terminal States:
\[ V(s) = \text{known} \]

Slide credit: Dan Klein and Pieter Abbeel, UC Berkeley CS 188
Adversarial Search (Minimax)

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 values: computed recursively

Terminal values: part of the game
Minimax Implementation

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

```python
def min_value(state):
    initialize v = +\infty
    for each successor of state:
        v = min(v, max_value(successor))
    return v
```

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

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

Slide credit: Dan Klein and Pieter Abbeel, UC Berkeley CS 188
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?
Minmax Demo
But we have two of these guys — what do we do?
Multi-player Games

Now what if A and B begin to collaborate?
Multi-player Games