Lecture 6
Games & Adversarial Search

CS 520: Intro AI  Jingjin Yu | Rutgers
Games

- Tic-tac-toe
- Backgammon
- Monopoly
- Chess
- The Chinese version
- The Japanese version
Just This Past Week (January 28th)...
Why are We Fascinated with Games?

- Games are “benchmarks” of human intelligence
- Model real world competitive and cooperative behaviors
  - Monopoly games: bargaining, cooperation, competition
  - Chess, go: competition, strategy
- But much simplified
  - E.g., chess has 32 pieces and a board with 64 positions
  - Ideal for mathematical study as well as applying computational techniques

- This lecture will cover
  - The MinMax algorithm (also known as minimax, MM...)
  - Alpha-beta pruning
  - Stochastic games & partially observable games
Focus: Alternating 2-Player Games

⇒ Such games are **sequential** with the players taking turns
⇒ The game ends with a **terminal state** with **utilities** for both players
⇒ **Zero-sum games**: one player’s utility is the negation of the other player’s utility – hence summed utility is zero
⇒ **Non zero-sum games**: total utility is non-zero; e.g., in soccer qualifying matches, 3 points for win, 1 point for draw, 0 for loss
Game as a Type of Search Problem

%!Games fall into a category of search problems that we have touched on – those with non-deterministic actions

%!In fact, we may assume we always face the worst outcome after we make a choice

%!This is known as adversarial search

%!The process of playing a two player game

%!Assuming Max and Min are playing a zero-sum game

%!At a step $k$ of the sequential game, Max wants to maximize her utility

%!At step $k + 1$, Min wants to maximize his utility, which means minimizing Max’s utility

%!The process continues until the game reaches a terminal state

%!This gives us a game (search) tree for the game
Example: Game Tree for Tic-Tac-Toe

Basically, the MinMax algorithm seeks to go from the leaves and collect the best choice at the top.
A Two-Ply Game Tree Example

⇒ Suppose Max and Min play according to the following game tree

⇒ The last layer shows the terminal states with utilities for Max
⇒ From here, Min minimizes the utility
⇒ Max then maximizes among Min’s choices
⇒ This is the gist of the MinMax algorithm
The MinMax (or Minimax) Algorithm

function MINIMAX-DECISION(state) returns an action
    return \( \arg \max_a \in \text{ACTIONS}(s) \) MIN-VALUE(Result(state, a))

function MAX-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    \( v \leftarrow -\infty \)
    for each \( a \) in ACTIONS(state) do
        \( v \leftarrow \max(v, \text{MIN-VALUE}(\text{RESULT}(s, a))) \)
    return \( v \)

function MIN-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    \( v \leftarrow \infty \)
    for each \( a \) in ACTIONS(state) do
        \( v \leftarrow \min(v, \text{MAX-VALUE}(\text{RESULT}(s, a))) \)
    return \( v \)

⇒ Basically, do \( \max \left\{ \min \left\{ \max \left\{ \min \{ \ldots \} \right\} \right\} \right\} \) until hits terminal states
The MinMax (or Minimax) Algorithm

⇒ To summarize, MinMax does

\[
\text{MINIMAX}(s) = \begin{cases} 
\text{UTILITY}(s) & \text{if } \text{TERMINAL-TEST}(s) \\
\max_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{PLAYER}(s) = \text{MAX} \\
\min_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{PLAYER}(s) = \text{MIN}
\end{cases}
\]

⇒ This can be used to compute optimal strategies when the game tree is reasonably shallow

⇒ An optimal strategy is one that gives the best utility if playing against an optimal opponent (i.e., always making utility-minimizing decisions)

⇒ Using an optimal strategy against a non-optimal opponent will not yield lower utility for Max

⇒ Example of an optimal strategy for a game...
Optimal Strategy for Tic-Tac-Toe, Visualized

http://xkcd.com/832/
Three Player Games

⇒ A three player game is rather different, e.g.,

- Each player has an individual utility
- Not possible for player $C$ to minimize utility of both $A$ and $B$
- Natural strategy is for $C$ to maximize his/her own utility
- Same for other players
Alpha-Beta Pruning (Exact) Heuristic

- Visit all terminal states in MinMax is unnecessary
- Keep track of
  - $\alpha$ – the value of the best choice (highest value) so far for Max
  - $\beta$ – the value of the best choice (lowest value) so far for Min
Alpha-Beta Pruning, General Case

- Keep track of
  - $\alpha$ – the value of the best choice (highest value) so far for Max
  - $\beta$ – the value of the best choice (lowest value) so far for Min

- Say Min node $m$ has the best choice for Max, which is $\alpha$
  - If another Min node $n$ has successor with utility $v \leq \alpha$
  - Then Max cannot do better than $v \leq \alpha$ on $n$
  - This is because Min will minimize over $n$
  - Such branches (under node $n$) can be safely truncated
  - Optimality is not affected

- Similarly, if a Max node has the best choice $\beta$ for Min
  - If a later Max node has a larger utility $\nu$, the node is a useless branch
  - An upper stream Min node will only pick $\beta$, not $\nu
MinMax with Alpha-Beta Pruning

function ALPHA-BETA-SEARCH(state) returns an action
    $v \leftarrow \text{MAX-VALUE}(state, -\infty, +\infty)$
    return the action in ACTIONS(state) with value $v$

function MAX-VALUE(state, $\alpha$, $\beta$) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    $v \leftarrow -\infty$
    for each $a$ in ACTIONS(state) do
        $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(\text{RESULT}(s,a), \alpha, \beta))$
        if $v \geq \beta$ then return $v$ // Pruning
        $\alpha \leftarrow \text{MAX}(\alpha, v)$ // Update
    return $v$

function MIN-VALUE(state, $\alpha$, $\beta$) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    $v \leftarrow +\infty$
    for each $a$ in ACTIONS(state) do
        $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(\text{RESULT}(s,a), \alpha, \beta))$
        if $v \leq \alpha$ then return $v$ // Pruning
        $\beta \leftarrow \text{MIN}(\beta, v)$ // Update
    return $v$
Properties of Alpha-Beta Pruning

⇒ Alpha-beta pruning does not affect optimality

⇒ With proper ordering (big if here) of node expansion, alpha-beta pruning examines $O(b^m)$ nodes instead of $O(b^m)$ for MinMax

⇒ This means much larger problems can be tackled with the same amount of computation time

⇒ The best order is generally not possible in practice
Imperfect Real-Time Decisions

\(\Rightarrow\) MinMax and alpha-beta pruning are optimal

\(\Rightarrow\) Requires full game tree to reach all terminal states

\(\Rightarrow\) Impractical in practice, e.g., chess game

\(\Rightarrow\) Possible positions \(\sim 10^{43}\)

\(\Rightarrow\) Complexity of \(10^{120}\) assuming 40 pairs of moves (typical game)

\(\Rightarrow\) This is known as the Shannon number (after Claude Shannon)

\(\Rightarrow\) Impossible to remember or compute for moderately complex games

\(\Rightarrow\) To play actual games, one needs to make imperfect real-time decisions

\(\Rightarrow\) Human players can evaluate the situation of a game board

\(\Rightarrow\) This may be abstracted as a heuristic value function
Evaluation function

- Replaces terminal states with evaluation functions
- These functions must be computed quickly
- A common evaluation function

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

- E.g., for chess, a naïve evaluation function may have \( w_i f_i(s) \) for a game piece \( i \).
  - \( w_i \) may represent the “power” of the piece, e.g., \( w_{\text{pawn}} = 1 \), \( w_{\text{queen}} = 10 \)
  - \( f_i \) may represent the current advantage of the piece. E.g., a pawn near opponent’s base may have much larger \( f \) than a starting pawn

- Evaluation function may be learned from game databases
  - AlphaGo uses random (Monte Carlo) tree search + deep learning
Issues with Evaluation Functions

- Cutting off game tree and applying evaluation function may cause the **horizon effect**
  - Evaluation function may have big variations at nearby search nodes
  - E.g.,

- Can be alleviated by **selectively adding more search depth**

(a) White to move

(b) White to move
Stochastic Games

⇒ Some games have a stochastic component (i.e., non-rational decisions on the game tree)
⇒ Backgammon is such a game
Handling Stochastic Games

⇒ Same MinMax + alpha-beta pruning + evaluation function approach
⇒ Replaces utility with expected utility to handle randomness

\[
\text{EXPECTIMINIMAX}(s) = \\
\begin{cases} 
\text{UTILITY}(s) & \text{if } \text{TERMINAL-TEST}(s) \\
\max_a \text{EXPECTIMINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{PLAYER}(s) = \text{MAX} \\
\min_a \text{EXPECTIMINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{PLAYER}(s) = \text{MIN} \\
\sum_r P(r) \text{EXPECTIMINIMAX}(\text{RESULT}(s, r)) & \text{if } \text{PLAYER}(s) = \text{CHANCE}
\end{cases}
\]

⇒ Basically, making “chance” a third player who plays by expectation instead of min or max
“Chance” an Expectation Based Player
Games with Imperfect Information

⇒ In many games, e.g., bridge, players have only partial information about the current state of the game

⇒ Strategies in such cases
  ⇒ Remembering past moves to infer the state of the game
  ⇒ Carry out Monte Carlo simulation to simulate the possible state of the game
AI for Games: a Brief Incomplete History

⇒ 1912: Ernst Zermelo published MinMax (minimax) algorithm
⇒ 1949: Claude Shannon – proposed evaluation function for chess playing
⇒ 1956: John McCarthy – alpha-beta pruning
⇒ 1956: Arthur Samuel – checkers program playing against itself
⇒ 1958-1972: early chess programs with alpha-beta
⇒ 1975: Knuth & Moore – correctness and complexity of alpha-beta
⇒ 1982: Pearl – showing alpha-beta’s asymptotic optimality
⇒ Many interesting development afterwards...
⇒ Most recently Monte Carlo tree + deep learning
  ⇒ Devil seems to be now in the engineering details
  ⇒ More interesting theory and algorithms ahead?
AI for Games: the Best

⇒ Better than human
  ⇒ Checkers: proven by Schaeffer’s team in 2007 that the best a human player can do against a program (Chinook) is a draw
  ⇒ Chess:
    ⇒ First victory of computer over top human player in 1997 (Deep Blue v.s. Kasparov)
    ⇒ Human rarely wins afterwards

⇒ On par with top human players
  ⇒ Backgammon
  ⇒ Bridge

⇒ Not just yet...
  ⇒ Texas hold’em
    ⇒ Limit Texas hold’em is statistically solved
    ⇒ No-limit, the more common version, still pretty difficult
  ⇒ Go
    ⇒ AlphaGo uses Monte Carlo tree + deep learning to beat a ~600 ranked player
    ⇒ A match against a top player Lee Sedol, will be held in March of this year
    ⇒ Current estimate is that AlphaGo will not win
Additional Resources and Exercises


- *Take a look at paper about AlphaGo
  - [http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html](http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html)
  - You should be able to download from Rutgers network

* suggests it’s optional