Games

- Games: multi-agent environment
  - What do other agents do and how do they affect our success?
  - Cooperative vs. competitive multi-agent environments.
  - Competitive multi-agent environments give rise to adversarial search a.k.a. games

- Why study games?
  - Fun!
  - They are hard
  - Easy to represent and agents restricted to small number of actions

Relation of Games to Search

- Search – no adversary
  - Solution is (heuristic) method for finding goal
  - Heuristics and CSP techniques can find optimal solution
  - Evaluation function: estimate of cost from start to goal through given node
  - Examples: path planning, scheduling activities

- Games – adversary
  - Solution is strategy (strategy specifies move for every possible opponent reply).
  - Time limits force approximate solutions
  - Examples: chess, checkers, Othello, backgammon

Types of Games

<table>
<thead>
<tr>
<th>Perfect information</th>
<th>Deterministic</th>
<th>Chance</th>
</tr>
</thead>
<tbody>
<tr>
<td>chess, go, checkers, othello</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Imperfect information</th>
<th>Bridge, hearts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Poker, canasta, scrabble</td>
<td></td>
</tr>
</tbody>
</table>

Our focus: deterministic, turn-taking, two-player, zero-sum games of perfect information.

- Zero-sum game: utility values at the end of the game are equal and opposite
- Perfect information: fully observable

Game setup

- Two players: MAX and MIN
- MAX moves first and they take turns until the game is over.
- Games as search:
  - Initial state: e.g. starting board configuration
  - Successor function: list of (move, state) pairs specifying legal moves and resulting states.
  - Terminal test: game over? (terminal states)
  - Utility function: value of terminal states. E.g. win (+1), lose (-1) and draw (0) in chess.
- Players use search tree to determine next move.

Partial Game Tree for Tic-Tac-Toe
The Tic-Tac-Toe Search Space

- Is this search space a tree or graph?
- What is the minimum solution depth?
- What is the maximum search depth?
- What is the branching factor?

Optimal strategies

- Find the best strategy for MAX assuming an infallible MIN opponent.
- Assumption: Both players play optimally.
- Given a game tree, the optimal strategy can be determined by using the minimax value of each node:

\[
\text{MINIMAX-VALUE}(n) = \begin{cases} 
\text{UTILITY}(n) & \text{if } n \text{ is a terminal} \\
\max_{s \in \text{successors}(n)} \text{MINIMAX-VALUE}(s) & \text{if } n \text{ is a max node} \\
\min_{s \in \text{successors}(n)} \text{MINIMAX-VALUE}(s) & \text{if } n \text{ is a min node}
\end{cases}
\]

Two-Ply Game Tree

Definition: ply = turn of a two-player game

The minimax decision makes MAX choose the option that maximizes the worst-case outcome for MAX.

Minimax Algorithm

- function MINIMAX-DECISION(state) returns an action
  - inputs: state, current state in game
  - return the action in SUCCESSORS(state) with value v
- function MAX-VALUE(state) returns a utility value
  - if TERMINAL-TEST(state) then return UTILITY(state)
  - for a in SUCCESSORS(state) do
    - v ← MAX(v, MIN-VALUE(a))
  - return v
- function MIN-VALUE(state) returns a utility value
  - if TERMINAL-TEST(state) then return UTILITY(state)
  - for a in SUCCESSORS(state) do
    - v ← MIN(v, MAX-VALUE(a))
  - return v
Properties of Minimax

- Minimax explores tree using DFS.
- Therefore:
  - Time complexity: $O(b^m)$
  - Space complexity: $O(bm)$

Problem of minimax search

- Number of game states is exponential in the number of moves.
  - Solution: Do not examine every node
  - $\Rightarrow$ Alpha-beta pruning
    - Remove branches that do not influence final decision
    - General idea: you can bracket the highest/lowest value at a node, even before all its successors have been evaluated

Alpha-Beta Pruning

- $\alpha$: the highest (i.e. best for Max) value possible
- $\beta$: the lowest (i.e. best for Min) value possible
- Initially $\alpha$ and $\beta$ are $(-\infty, +\infty)$.

Alpha-Beta Example

- Range of possible values
- $\alpha$ and $\beta$ update as the search progresses
- Example tree showing how $\alpha$ and $\beta$ are updated.
Alpha-Beta Algorithm

function ALPHA-BETA-SEARCH(state) returns an action
inputs:
state, current state in game
\[ v \leftarrow \text{MAX-VALUE}(\text{state}, -\infty, +\infty) \]
return the action in SUCCESSORS(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 \( a, s \) in SUCCESSORS(state) do
\[ v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(s, \alpha, \beta)) \]
if \( v \geq \beta \) then return \( v \)
\[ \alpha \leftarrow \text{MAX}(\alpha, v) \]
return \( v \)

function MIN-VALUE(state, \( \alpha \), \( \beta \)) returns a utility value
if TERMINAL-TEST(state) then return UTILITY(state)
\[ v \leftarrow +\infty \]
for \( a, s \) in SUCCESSORS(state) do
\[ v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(s, \alpha, \beta)) \]
if \( v \leq \alpha \) then return \( v \)
\[ \beta \leftarrow \text{MIN}(\beta, v) \]
return \( v \)

Alpha-beta pruning

- When enough is known about a node \( n \), it can be pruned.

Final Comments about Alpha-Beta Pruning

- Pruning does not affect final results
- Entire subtrees can be pruned, not just leaves.
- Good move ordering improves effectiveness of pruning
  - With "perfect ordering," time complexity is \( O(b^{m/2}) \)
  - Effective branching factor of \( \sqrt{b} \)
  - Consequence: alpha-beta pruning can look twice as deep as minimax in the same amount of time

Is this practical?

- Minimax and alpha-beta pruning still have exponential complexity.
- May be impractical within a reasonable amount of time.
- SHANNON (1950):
  - Terminate search at a lower depth
  - Apply heuristic evaluation function EVAL instead of the UTILITY function

Cutting off search

- Change:
  - if TERMINAL-TEST(state) then return UTILITY(state) into
  - if CUTOFF-TEST(state, depth) then return EVAL(state)
  - Introduces a fixed-depth limit
  - Selected so that the amount of time will not exceed what the rules of the game allow.
  - When cutoff occurs, the evaluation is performed.
Heuristic EVAL

- Idea: produce an estimate of the expected utility of the game from a given position.
- Performance depends on quality of EVAL.
- Requirements:
  - EVAL should order terminal-nodes in the same way as UTILITY.
  - Fast to Compute.
  - For non-terminal states the EVAL should be strongly correlated with the actual chance of winning.

Heuristic EVAL example

In chess:
$$\text{Eval}(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

How good are computers…

- Let’s look at the state of the art computer programs that play games such as chess, checkers, othello, go…

Checkers

- Chinook: the first program to win the world champion title in a competition against a human (1994)

Chinook

- Components of Chinook:
  - Search (variant of alpha-beta). Search space has $10^{20}$ states.
  - Evaluation function
  - Endgame database (for all states with 4 vs. 4 pieces; roughly 40 billion positions).
  - Opening book - a database of opening moves
  - Chinook can determine the final result of the game within the first 10 moves.
  - Author has recently shown that several openings lead to a draw.

Chess

- 1997: Deep Blue wins a 6-game match against Garry Kasparov
  - Searches using iterative deepening alpha-beta; evaluation function has over 8000 features; opening book of 4000 positions; end game database.
  - FRITZ plays world champion, Victor Kramnik; 8-game draw.
The best Othello computer programs can easily defeat the best humans (e.g. Logistello, 1997).

Games that include chance
- Possible moves (5-10,5-11), (5-11,19-24), (5-10,10-16) and (5-11,11-16)

Possible moves (5-10,5-11), (5-11,19-24), (5-10,10-16) and (5-11,11-16)
- [1,1],…,[6,6] probability 1/36, all others - 1/18
- Can not calculate definite minimax value, only expected value

Expected minimax value
- **UTILITY(n)**
- **MAX** ∈ `successors(n)` EXPECTMINMAX(s)
- **MIN** ∈ `successors(n)` EXPECTMINMAX(s)
- **∑** ∈ `successors(n)` P(s) EXPECTMINMAX(s)

P(s) is the probability of a chance state s (e.g. a roll of the dice).
These equations can be propagated recursively in a similar way to the MINIMAX algorithm.

TD-Gammon (Tesauro, 1994)
- World class program based on a combination of reinforcement Learning, neural networks and alpha-beta pruning to 3 plies.
- Move analyses by TD-Gammon have lead to some changes in accepted strategies.

White’s turn, with a roll of 4-4
Summary

- Games are fun
- They illustrate several important points about AI
  - Perfection is unattainable -> approximation
  - Uncertainty constrains the assignment of values to states