Games and adversarial search
(Chapter 5)

World Champion chess player Garry Kasparov is defeated by IBM’s Deep Blue chess-playing computer in a six-game match in May, 1997

(link)
Why study games?

• Games are a traditional hallmark of intelligence
• Games are easy to formalize
• Games can be a good model of real-world competitive or cooperative activities
  – Military confrontations, negotiation, auctions, etc.
## Types of game environments

<table>
<thead>
<tr>
<th>Perfect information (fully observable)</th>
<th>Deterministic: Chess, checkers, go</th>
<th>Stochastic: Backgammon, monopoly</th>
</tr>
</thead>
<tbody>
<tr>
<td>Imperfect information (partially observable)</td>
<td>Deterministic: Battleships</td>
<td>Stochastic: Scrabble, poker, bridge</td>
</tr>
</tbody>
</table>
Alternating two-player zero-sum games

• Players take turns
• Each game outcome or terminal state has a utility for each player (e.g., 1 for win, 0 for loss)
• The sum of both players’ utilities is a constant
Games vs. single-agent search

• We don’t know how the opponent will act
  – The solution is not a fixed sequence of actions from start state to goal state, but a *strategy* or *policy* (a mapping from state to best move in that state)

• Efficiency is critical to playing well
  – The time to make a move is limited
  – The branching factor, search depth, and number of terminal configurations are huge
    • In chess, *branching factor* \( \approx 35 \) and *depth* \( \approx 100 \), giving a search tree of \( 10^{154} \) nodes
      – Number of atoms in the observable universe \( \approx 10^{80} \)
  – This rules out searching all the way to the end of the game
Game tree

- A game of tic-tac-toe between two players, “max” and “min”
COMPLETE MAP OF OPTIMAL TIC-TAC-TOE MOVES

YOUR MOVE IS GIVEN BY THE POSITION OF THE LARGEST RED SYMBOL ON THE GRID. WHEN YOUR OPPONENT PICKS A MOVE, ZOOM IN ON THE REGION OF THE GRID WHERE THEY WENT. REPEAT.

MAP FOR X:

http://xkcd.com/832/
MAP FOR O:

http://xkcd.com/832/
A more abstract game tree

Terminal utilities (for MAX)

A two- ply game
Game tree search

- **Minimax value of a node**: the utility (for MAX) of being in the corresponding state, assuming perfect play on both sides
- **Minimax strategy**: Choose the move that gives the best worst-case payoff
Computing the minimax value of a node

- **Minimax**\((\text{node})\) =
  - \(\text{Utility}(\text{node})\) if \(\text{node}\) is terminal
  - \(\max_{\text{action}} \text{Minimax}(\text{Succ}(\text{node}, \text{action}))\) if \(\text{player} = \text{MAX}\)
  - \(\min_{\text{action}} \text{Minimax}(\text{Succ}(\text{node}, \text{action}))\) if \(\text{player} = \text{MIN}\)
Optimality of minimax

- The minimax strategy is optimal against an optimal opponent
- What if your opponent is suboptimal?
  - Your utility can only be higher than if you were playing an optimal opponent!
  - A different strategy may work better for a sub-optimal opponent, but it will necessarily be worse against an optimal opponent

Example from D. Klein and P. Abbeel
More general games

- More than two players, non-zero-sum
- Utilities are now tuples
- Each player maximizes their own utility at their node
- Utilities get propagated (*backed up*) from children to parents
Alpha-beta pruning

• It is possible to compute the exact minimax decision without expanding every node in the game tree.
Alpha-beta pruning

• It is possible to compute the exact minimax decision without expanding every node in the game tree.

```
MAX

MIN

3

12

8

≥3
```
Alpha-beta pruning

- It is possible to compute the exact minimax decision without expanding every node in the game tree.
Alpha-beta pruning

- It is possible to compute the exact minimax decision without expanding every node in the game tree
Alpha-beta pruning

• It is possible to compute the exact minimax decision without expanding every node in the game tree.
Alpha-beta pruning

- It is possible to compute the exact minimax decision without expanding every node in the game tree
Alpha-beta pruning

- $\alpha$ is the value of the best choice for the MAX player found so far at any choice point above node $n$
- We want to compute the MIN-value at $n$
- As we loop over $n$'s children, the MIN-value decreases
- If it drops below $\alpha$, MAX will never choose $n$, so we can ignore $n$'s remaining children
- Analogously, $\beta$ is the value of the lowest-utility choice found so far for the MIN player
**Alpha-beta pruning**

**Function** \( \text{action} = \text{Alpha-Beta-Search}(\text{node}) \)
- \( \nu = \text{Min-Value}(\text{node}, -\infty, \infty) \)
- return the \textit{action} from \textit{node} with value \( \nu \)

\( \alpha \): best alternative available to the Max player
\( \beta \): best alternative available to the Min player

**Function** \( \nu = \text{Min-Value}(\text{node}, \alpha, \beta) \)
- if Terminal(\text{node}) return Utility(\text{node})
- \( \nu = +\infty \)
- for each \textit{action} from \text{node}
  - \( \nu = \text{Min}(\nu, \text{Max-Value}(\text{Succ}(\text{node}, \textit{action}), \alpha, \beta)) \)
  - if \( \nu \leq \alpha \) return \( \nu \)
  - \( \beta = \text{Min}(\beta, \nu) \)
- end for
- return \( \nu \)
**Alpha-beta pruning**

**Function** \( action = \text{Alpha-Beta-Search}(node) \)

\[
v = \text{Max-Value}(node, -\infty, \infty)
\]

return the \( action \) from \( node \) with value \( v \)

\( \alpha: \) best alternative available to the Max player

\( \beta: \) best alternative available to the Min player

**Function** \( v = \text{Max-Value}(node, \alpha, \beta) \)

if Terminal(\( node \)) return Utility(\( node \))

\( v = -\infty \)

for each \( action \) from \( node \)

\[
v = \max(v, \text{Min-Value}(\text{Succ}(node, action), \alpha, \beta))
\]

if \( v \geq \beta \) return \( v \)

\( \alpha = \max(\alpha, v) \)

end for

return \( v \)
Alpha-beta pruning

• Pruning does not affect final result
• Amount of pruning depends on move ordering
  – Should start with the “best” moves (highest-value for MAX or lowest-value for MIN)
  – For chess, can try captures first, then threats, then forward moves, then backward moves
  – Can also try to remember “killer moves” from other branches of the tree
• With perfect ordering, the time to find the best move is reduced to $O(b^{m/2})$ from $O(b^m)$
  – Depth of search is effectively doubled
Evaluation function

• Cut off search at a certain depth and compute the value of an evaluation function for a state instead of its minimax value
  – The evaluation function may be thought of as the probability of winning from a given state or the expected value of that state
• A common evaluation function is a weighted sum of features:

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

  – For chess, \( w_k \) may be the material value of a piece (pawn = 1, knight = 3, rook = 5, queen = 9) and \( f_k(s) \) may be the advantage in terms of that piece
• Evaluation functions may be learned from game databases or by having the program play many games against itself
Cutting off search

• **Horizon effect:** you may incorrectly estimate the value of a state by overlooking an event that is just beyond the depth limit
  – For example, a damaging move by the opponent that can be delayed but not avoided

• **Possible remedies**
  – **Quiescence search:** do not cut off search at positions that are unstable – for example, are you about to lose an important piece?
  – **Singular extension:** a strong move that should be tried when the normal depth limit is reached
Advanced techniques

• **Transposition table** to store previously expanded states
• **Forward pruning** to avoid considering all possible moves
• **Lookup tables** for opening moves and endgames
Chess playing systems

• Baseline system: 200 million node evaluations per move (3 min), minimax with a decent evaluation function and quiescence search
  – 5-ply ≈ human novice
• Add alpha-beta pruning
  – 10-ply ≈ typical PC, experienced player
• Deep Blue: 30 billion evaluations per move, singular extensions, evaluation function with 8000 features, large databases of opening and endgame moves
  – 14-ply ≈ Garry Kasparov
• More recent state of the art (Hydra, ca. 2006): 36 billion evaluations per second, advanced pruning techniques
  – 18-ply ≈ better than any human alive?