Chess: Computer vs. Human

Deep Blue is a chess-playing computer developed by IBM.

- On February 10, 1996, Deep Blue became the first machine to win a chess game against a reigning world champion (Garry Kasparov) under regular time controls.
- On 11 May 1997, the machine won a six-game match by two wins to one with three draws against world champion Garry Kasparov.

Rybka: Computer Chess

Rybka is a computer chess engine designed by International Master Vasik Rajlich.

- As of February 2011, Rybka is one of the top-rated engines on chess engine rating lists and has won many computer chess tournaments.
- Rybka won four consecutive World Computer Chess Championships from 2007 to 2010, but it was stripped of these titles in June 2011 because that Rybka plagiarized code from both the Crafty and the Fruit chess engines. Others dispute this.
- Rybka 2.2n2 is available as a free download and Deep Rybka 3 is ranked first among all computer chess programs in 2010.
- Rybka uses a bitboard representation, and is an alpha-beta searcher with a relatively large aspiration window. It uses very aggressive pruning, leading to imbalanced search trees.

Chess: Computer vs. Human

Deep Fritz is a German chess program developed by Frans Morsch and Mathias Feist and published by ChessBase.

- In 2002, Deep Fritz drew the Brains in Bahrain match against the classical World Chess Champion Vladimir Kramnik 4-4.
- On June 23, 2005, in the ABC Times Square Studios, Fritz 9 drew against the then FIDE World Champion Rustam Kasimdzhanov.
- From 25 November-5 December 2006 Deep Fritz played a six game match against Kramnik in Bonn. Fritz was able to win 4-2.
Games vs. Search problems

- "Unpredictable" opponent: Solution is a contingency plan
- Time limits: Unlikely to find the best step, must approximate
- Game types:

<table>
<thead>
<tr>
<th>deterministic</th>
<th>chance</th>
</tr>
</thead>
<tbody>
<tr>
<td>perfect information</td>
<td>chess, checkers, go, othello</td>
</tr>
<tr>
<td>imperfect information</td>
<td>bridge, poker, scrabble, blackjack</td>
</tr>
</tbody>
</table>

Zappa: Computer Chess

Zappa is a chess engine written by Anthony Cozzie, a graduate student at the University of Illinois at Urbana-Champaign.

- The program emphasizes sound search and a good use of multiple processors. Zappa scored an upset victory at the World Computer Chess Championship in August, 2005, in Reykjavík, Iceland. Zappa won with a score of 10 out of 11, and beat both Junior and Shredder, programs that had won the championship many times.
- Zappa’s other tournament successes include winning CCT7 and defeating Grandmaster Jaan Ehlvest 3-1. In Mexico in September 2007 Zappa won a match against Rybka by a score of 5.5 to 4.5.
- In March 2008 Anthony Cozzie announced that "the Zappa project is 100% finished", which includes both tournaments and future releases.
- In June 2010, Zach Wegner announced that he had acquired the rights to maintain and improve the Zappa engine. The improved engine competed in the 2010 WCCC under the name Rondo, achieving second place behind Rybka.

Minimax

Perfect play for deterministic, perfect-information games

Idea: choose move to position with highest minimax value
= best achievable payoff against best play

E.g., 2-ply game:

```
MAX

A_1  A_2  A_3
3  2  1
```

Tic-Tac-Toe

```
MAX (X)
```

```
MIN (O)
```

```
TERMINAL
```

Utility

-1 0 +1
Properties of Minimax

- **Complete**: Yes, if tree is finite (chess has specific rules for this)
- **Optimal**: Yes, against an optimal opponent. Otherwise??
- **Time complexity**: $O(b^m)$
- **Space complexity**: $O(bm)$ (depth-first exploration)

For chess, $b \approx 35$, $m \approx 100$ for “reasonable” games ⇒ best exact solution infeasible

Minimax Algorithm

**Function MINIMAX-DECISION(game)** returns an operator

```
for each op in OPERATORS[game] do
    VALUE[op] ← MINIMAX-VALUE(APPLY(op, game), game)
end
return the op with the highest VALUE[op]
```

**Function MINIMAX-VALUE(state, game)** returns a utility value

```
if TERMINAL-TEST[game](state) then
    return UTILITY[game](state)
else if MAX is to move in state then
    return the highest MINIMAX-VALUE of SUCCESSORS(state)
else
    return the lowest MINIMAX-VALUE of SUCCESSORS(state)
```

Digression: Exact values don’t matter

- Behaviour is preserved under any monotonic transformation of $EVAL$
- Only the order matters: payoff in deterministic games acts as an ordinal utility function

Resource Limits

Suppose we have 100 seconds, explore $10^4$ nodes/second ⇒ $10^6$ nodes per move

Standard approach:
- **cutoff test** e.g., depth limit
- **evaluation function** = estimated desirability of position and explore only (hopeful) nodes with certain values
Cutting Off Search

MINIMAX_CUTOFF is identical to MINIMAX_VALUE except

1. TERMINAL? is replaced by CUTOFF?
2. UTILITY is replaced by EVAL

Does it work in practice?

\[ b^m = 10^6, \quad b = 35 \quad \Rightarrow \quad m = 4 \]

4-ply \( \approx \) human novice
8-ply \( \approx \) typical PC, human master
12-ply \( \approx \) Deep Blue, Kasparov
The $\alpha-\beta$ algorithm

$$\text{function } \text{MAX-VALUE}(state, game, x, y) \text{ returns the minimax value of } state$$

$$\text{inputs: } state, \text{ current state in game}$$

$$\text{game, game description}$$

$$\alpha, \text{ the best score for } \text{MAX} \text{ along the path to } state$$

$$\beta, \text{ the best score for } \text{MIN} \text{ along the path to } state$$

$$\text{if CUTOFF-TEST}(state) \text{ then return } \text{EVAL}(state)$$

$$\text{for each } s \text{ in } \text{SUCCESSORS}(state) \text{ do}$$

$$\alpha \leftarrow \text{MAX}(\alpha, \text{MIN-VALUE}(s, game, x, y))$$

$$\text{if } \alpha \geq \beta \text{ then return } \beta$$

$$\text{end}$$

$$\text{return } \alpha$$

$$\text{function } \text{MIN-VALUE}(state, game, x, y) \text{ returns the minimax value of } state$$

$$\text{if CUTOFF-TEST}(state) \text{ then return } \text{EVAL}(state)$$

$$\text{for each } s \text{ in } \text{SUCCESSORS}(state) \text{ do}$$

$$\beta \leftarrow \text{MIN}(\beta, \text{MAX-VALUE}(s, game, x, y))$$

$$\text{if } \beta \leq \alpha \text{ then return } \alpha$$

$$\text{end}$$

$$\text{return } \beta$$

Good Move Ordering Is Important
Properties of $\alpha-\beta$

- Pruning does not affect final result.
- Good move ordering improves effectiveness of pruning.
- With "perfect ordering," time complexity = $O(b^{d/2})$.
- With good move ordering, can easily reach depth 8 and play good chess.

A simple example of the value of reasoning about which computations are relevant (a form of metareasoning).

Deterministic Games in Practice

- **Checkers**: Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used an endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 443,748,401,247 positions.
- **Othello**: Human champions refuse to compete against computers, who are too good.
- **Go**: Human champions refuse to compete against computers, who are too bad. In go, $b \geq 300$, so most programs use pattern knowledge bases to suggest plausible moves.

Nondeterministic games

- In backgammon, the dice rolls determine the legal moves.
- Simplified example with coin-flipping instead of dice-rolling.
Nondeterministic games in practice

- Dice rolls increase $b$: 21 possible rolls with 2 dice
- Backgammon $\approx 20$ legal moves (can be 6,000 with 1-1 roll)
  \[
  \text{depth 4} = 20 \times (21 \times 20)^3 \approx 1.2 \times 10^9
  \]
- As depth increases, probability of reaching a given node shrinks
  \[\Rightarrow\text{value of lookahead is diminished}\]
- $\alpha-\beta$ pruning is much less effective
- TDGAMMON uses depth-2 search + very good $\text{EVAL}$
  \[\approx \text{world-champion level}\]

Algorithm for Nondeterministic Games

- $\text{EXPECTMINIMAX}$ gives perfect play
- Just like $\text{MINIMAX}$, except we must also handle chance nodes:
  \[
  \begin{align*}
  &\text{if } \text{state is a chance node then} \\
  &\quad \text{return average of } \text{EXPECTMINIMIXVALUE of SUCCESSORS(state)}
  \end{align*}
  \]
- A version of $\alpha-\beta$ pruning is possible but only if the leaf values are bounded.

Games of imperfect information

- Examples include card games, where opponent’s initial cards are unknown
- Typically we can calculate a probability for each possible deal. Seems just like having one big dice roll at the beginning of the game
- Idea: compute the minimax value of each action in each deal, then choose the action with highest expected value over all deals
- Special case: if an action is optimal for all deals, it's optimal.
- GIB, current best bridge program, approximates this idea by
  1) generating 100 deals consistent with bidding information,
  2) picking the action that wins most tricks on average

Exact values DO matter

- Behaviour is preserved only by positive linear transformation of $\text{EVAL}$
- Hence $\text{EVAL}$ should be proportional to the expected payoff
Games are fun to work on! They illustrate several important points about AI.

- perfection is unattainable ⇒ must approximate
- good idea to think about what to think about
- uncertainty constrains the assignment of values to states

Games are to AI as grand prix racing is to automobile design.