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.
- 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.
- In November 2003, Deep Fritz drew a four-game match against Garry Kasparov.
- 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.
- On the November 3, 2007 SSDF rating list, Fritz 10 placed fifth with a rating of 2856, points below #1 ranked Rybka.

Artificial Intelligence

Adversarial Search

Readings: Chapter 6 of Russell & Norvig.

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>backgammon, monopoly</td>
</tr>
<tr>
<td>imperfect information</td>
<td>bridge, poker, scrabble</td>
</tr>
</tbody>
</table>
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)

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
  3 12 8 6 4
  14 5 2
MIN
  A1  A2  A3
  A11  A12  A13
  A21  A22  A23
  A31  A32  A33
```

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

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
⇒ exact solution completely infeasible
Cutting Off Search

**MINIMAXCUTOFF** is identical to **MINIMAXVALUE** except

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

Does it work in practice?

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

4-ply \(\approx\) human novice

8-ply \(\approx\) typical PC, human master

12-ply \(\approx\) Deep Blue, Kasparov

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

**α–β** Pruning Example

```
MAX

MIN

3
12 8
2
X
X
```

```
MAX

MIN

3
12 8
2
X
X
```

```
MAX

MIN

3
12 8
2
X
X
```

```
MAX

MIN

3
12 8
2
X
X
```

```
MAX

MIN

3
12 8
2
X
X
```
Properties of $\alpha-\beta$

- Pruning does not affect final result.
- Good move ordering improves effectiveness of pruning.
- With “perfect ordering,” time complexity $= O(b^{m/2})$
  ⇒ doubles depth of search
  ⇒ 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)
The $\alpha-\beta$ algorithm

function $\text{MAX-VALUE}(\text{state}, \text{game}, \alpha, \beta)$ returns the minimax value of $\text{state}$
inputs: $\text{state}$, current state in $\text{game}$
$\text{game}$, game description
$\alpha$, the best score for $\text{MAX}$ along the path to $\text{state}$
$\beta$, the best score for $\text{MIN}$ along the path to $\text{state}$

if $\text{CUTOFF-TEST}(\text{state})$ then return $\text{EVAL}(\text{state})$
for each $s$ in $\text{SUCCESSORS}(\text{state})$ do
    $\alpha \leftarrow \max(\alpha, \text{MAX-VALUE}(s, \text{game}, \alpha, \beta))$
    if $\alpha \geq \beta$ then return $\beta$
end
return $\alpha$

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

if $\text{CUTOFF-TEST}(\text{state})$ then return $\text{EVAL}(\text{state})$
for each $s$ in $\text{SUCCESSORS}(\text{state})$ do
    $\beta \leftarrow \min(\beta, \text{MAX-VALUE}(s, \text{game}, \alpha, \beta))$
    if $\beta \leq \alpha$ then return $\alpha$
end
return $\beta$

Why is it called $\alpha-\beta$?

$\alpha$ is the best value (to $\text{MAX}$) found so far off the current path
If $V$ is worse than $\alpha$, $\text{MAX}$ will avoid it $\Rightarrow$ prune that branch
Similarly, $\beta$ is the best value for $\text{MIN}$.

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, $b > 300$, so most programs use pattern knowledge bases to suggest plausible moves.
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
  $\Rightarrow$ value of lookahead is diminished
- $\alpha-\beta$ pruning is much less effective
- TDGAMMON uses depth-2 search + very good $\text{EVAL} \approx$ world-champion level

Algorithm for Nondeterministic Games

- EXPECTIMINIMAX gives perfect play
- Just like MINIMAX, except we must also handle chance nodes:
  \[ \ldots \]
  \[ \text{if } \text{state} \text{ is a chance node then} \]
  \[ \text{return average of \text{EXPECTIMINIMAX}VALUE of} \]
  \[ \text{SUCCESSORS}(\text{state}) \]
  \[ \ldots \]
- 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

- 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

Diagram:

- MAX
- DICE
- MIN
- 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.