Chess: 1997: Deep Blue defeats human champion Gary Kasparov in a six-game match. Deep Blue examined 200M positions per second, used sophisticated evaluation and undisclosed methods for extending some lines of search up to 40 ply. Current programs are even better, if less historic.

Go: Human champions are being beaten. In go, b > 300! Classically use pattern knowledge bases, but big recent advances use Monte Carlo (randomized) expansion methods.

Pacman

Many different kinds of games!
Axes:
- Deterministic or stochastic?
- One, two, or more players?
- Zero sum?
- Perfect information (can you see the state)?

Want algorithms for calculating a strategy (policy) which recommends a move from each state.

Solution for a player is a policy: \( S \rightarrow A \).
Zero-Sum Games

- Agents have opposite utilities (values on outcomes)
- Lets us think of a single value that one maximizes and the other minimizes
- Adversarial, pure competition

General Games
- Agents have independent utilities (values on outcomes)
- Cooperation, indifference, competition, and more are all possible
- More later on non-zero-sum games

Value of a State: utility of best achievable outcome from state.

Non terminal states:
\[ V(s) = \max_{s' \in \text{kids}(s)} V(s') \]

Terminal States:
\[ V(s) = \text{known} \]

Max states:
\[ V(s) = \max_{s' \in \text{succ}(s)} V(s') \]

Min states:
\[ V(s) = \min_{s' \in \text{succ}(s)} V(s') \]

States Under Agent’s Control. Max.
Terminal States.
States Under Opponent’s Control. Min.
Tic-Tac-Toe Game Tree

Adversarial Search (Minimax)

Minimax values computed recursively.

Deterministic, zero-sum games:
- Tic-tac-toe, chess, checkers
- One player maximizes result
- The other minimizes result

Minimax search:
- A state-space search tree
- Players alternate turns
- Compute each node’s minimax value: the best achievable utility against a rational (optimal) adversary

Minimax Efficiency

How efficient is minimax?
- Just like (exhaustive) DFS
- Time: $O(b^m)$
- Space: $O(bm)$

Example: For chess, $b \approx 35$, $m \approx 100$
- Exact solution is completely infeasible.
- But, do we need to explore the whole tree?

Minimax Properties

Optimal against a perfect player. Otherwise?

Demo: $\text{minvsexp}(L6D2, L6D3)$
Problem: In realistic games, cannot search to leaves!

Solution: Depth-limited search.
Instead, search to limited depth in the tree.

Use an evaluation function for non-terminal positions.

Example:
Suppose we have 100 seconds, can explore 10K nodes / sec.
So can check 1M nodes per move.
↓ - reaches about depth 8 – decent chess program

Guarantee of optimal play is gone.

More plies makes a BIG difference.
Use iterative deepening for an anytime algorithm.

Evaluation functions are always imperfect.
The deeper in the tree the evaluation function is buried, the less the quality of the evaluation function matters.

An important example of the tradeoff between complexity of features and complexity of computation.

Demo: depthlimited(L6D4, L6D5)
Evaluation Functions

Evaluation functions score non-terminals in depth-limited search. Ideal function: returns the actual minimax value of the position. In practice: typically weighted linear sum of features:

\[ w_1 f_1(s) + w_2 f_2(s) + \cdots + w_n f_n(s). \]

Example:

\[ f_1(s) = (\text{num white queens} - \text{num black queens}), \text{etc.} \]

Why Pacman Starves

A danger of replanning agents!

- He knows his score will go up by eating the dot now (west, east).
- He knows his score will go up just as much by eating the dot later (east, west).
- There are no point-scoring opportunities after eating the dot (within the horizon, two here).
- Therefore, waiting seems just as good as eating: he may go east, then back west in the next round of replanning!
**Alpha-Beta Pruning**

- General configuration (MIN version)
  - Computing MIN-VALUE at some node $n$
  - Looping over $n$'s children
  - $n$'s estimate of min is dropping
  - Who cares about $n$'s value? MAX
  - $a = \text{"best MAX value on path to root."}$
    - If $n < a$, then MAX will never choose it.
    - So search “prunes” other children.
  - Symmetric for MAX version.

**Alpha-Beta Implementation**

- $\alpha$ - MAX's best option on path to root.
- $\beta$ - MIN's best option on path to root.

```python
def min_value(state, $\alpha$, $\beta$):
    initialize v = $\infty$.
    for each successor $s$ of state:
        v = min(v, value(s, $\alpha$, $\beta$))
    if v $\leq$ $\alpha$ return v
    $\beta$ = min($\beta$, v)
    return v

def max_value(state, $\alpha$, $\beta$):
    initialize v = $-\infty$.
    for each successor $s$ of state:
        v = max(v, value(s, $\alpha$, $\beta$))
    if v $\leq$ $\alpha$ return v
    $\alpha$ = min($\alpha$, v)
    return v
```

**Alpha-Beta Pruning Properties**

- This pruning has no effect on minimax value computed for the root!
- Values of intermediate nodes might be wrong
  - **Important:** root's children may be wrong
- Good child ordering improves effectiveness
  - With “perfect ordering”:
    - Time complexity drops to $O(b^{m/2})$
    - Doubles solvable depth!
  - Full search of, e.g. chess, is still hopeless...

This is a simple example of metareasoning (computing about what to compute)

**Alpha-Beta Quiz**

```
  a
 b   d
  c   e
```

**Alpha-Beta Quiz 2**

```
  a
 b   h
 c   i
```

**Next Time: Uncertainty!**