Chinook beat 40-year-reign of champion Marion Tinsley using complete 8-piece endgame. 2007: Checkers solved!

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
Behavior from Computation


Demo: mysterypacman(L6D1)
Video of Demo Mystery Pacman
Adversarial Games

Types of Games

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.
Many possible formalizations, one is:

- States: \( S \) (start at \( s_0 \))
- Players: \( P = \{1...N\} \) (usually take turns).
- Actions: \( A \) (may depend on player / state)
- Transition Function: \( S \times A \rightarrow S \).
- Terminal Test: \( S \rightarrow \{t, f\} \)
- Terminal Utilities: \( S \times P \rightarrow \mathbb{R} \).

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
Adversarial Search
Single-Agent Trees
Value of Game Tree

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:
\[ V(s) = \text{known} \]
Adversarial Game Tree
Minimax Values

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

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

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

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

Terminal values: part of the game.
Minimax Implementation

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

```python
def min-value(state):
    initialize v = +\infty
    for each successor s of state:
        v = min(v, max-value(s))
    return v
```

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

```python
def max-value(state):
    initialize v = -\infty.
    for each successor s of state:
        v = max(v, min-value(s))
    return v
```

```python
def value(state):
    if the state is terminal: return the state’s utility
    if the next agent is MAX: return max-value(state)
    if the next agent is MIN: return min-value(state)
```
Minimax Example
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

Minimax values: computed recursively.

Terminal values: part of the game.

Optimal against a perfect player. Otherwise?

Demo: \( \text{minvsexp}(L6D2, L6D3) \)
Video of Demo Min vs. Exp (Min)
Video of Demo Min vs. Exp (Exp)
Resource Limits
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
Depth Matters

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($L6D4, L6D5$)
Video of Demo Limited Depth (2)
Video of Demo Limited Depth (10)
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: $f_1(s) = (\text{num white queens} - \text{num black queens}),$ etc.
Demo: \(\text{thrashingd} = 2, \text{thrashingd} = 2(\text{fixedevaluationfunction}), \text{smartghostscoordinate}\)
Video of Demo Thrashing (d=2)
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!
Video of Demo Thrashing – Fixed (d=2)
Video of Demo Smart Ghosts (Coordination)
Video of Demo Smart Ghosts (Coordination) – Zoomed In
Game Tree Pruning
Minimax Example
Minimax Pruning
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
- $a = “ best MAX value on path to root.”$
- If $n < a$, than 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
- → most naive version not for action selection

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

The diagram represents a decision-making process with the following nodes:

- **Root Node**: Node a
- **Children Nodes**:
  - Node b
  - Node c
  - Node d
  - Node e
  - Node f

- **Leaves**:
  - Node 10
  - Node 8
  - Node 4
  - Node 50
Next Time: Uncertainty!