Two-Player Games

- A special category of multiagent Environment
- One other agent
- Other agent is adversarial
  - objectives are exactly opposed to the primary agent
  - (zero-sum game)
- Two agents move in alternating turns
Game as Search Problem

- States (as usual)
  - E.g., board position
  - Player to move
- Actions: legal moves in state
- Result function:
  - State resulting from choosing given (legal) move in given state
- Terminal-test: determines when game over
- Utility (payoff) function: assigns numeric values to terminal states

These together define the game tree (analogous to state-space graph)

A Game Tree

![Game Tree Diagram]

- x wins
- tie

...
Winning Strategies

- A winning strategy for tic-tac-toe:
  - A move 1 for X, such that
    - for any move 1 for O, there is
      - a move 2 for X, such that
        - for any move 2 for O, there is
          - ... such that X wins.

“Grundy’s Game”

- Two players, one or more piles of pennies.
- Players take turns splitting one of the piles into two of unequal size.
- Piles of one or two cannot be split.
- Winner is last player to successfully split a pile.
Grundy, n = 7

Propagating Game Values
**Minimax**

- First player **MAX**
- Second player **MIN**
- Label leaves with utility of result (e.g., win = 1, lose = -1)
- Minimax algorithm
  - Traverse depth-first to label nodes with *minimax* value
  - MAX or MIN nodes, depending on whose turn

**Minimax Values**

- Leaf node
  - Utility of result of game (e.g., 1 for win)
- MAX node
  - Max of minimax value of successors
- MIN node
  - Min of minimax value of successors
**Minimax Algorithm**

**Function** `Minimax-decision(state)` \textbf{returns} an action
\begin{itemize}
  \item \textit{inputs:} current state in game
  \item `v` ← `Max-value(state)`
  \item \text{\textit{return the action in Actions(state) with value}} \text{v}
\end{itemize}

---

**Function** `Max-Value` \textbf{returns} a utility value
\begin{itemize}
  \item if terminal-test(state) \textbf{then return} payoff(state)
  \item `v` ← – infinity
  \item \textbf{for} `a` in Actions(state) \textbf{do}
  \item \hspace{1em} `v` ← \text{Max(`v, Min-value(Result(s,a)))}
  \item \textit{return} \text{v}
\end{itemize}

---

**Function** `Negamax-Value` \textbf{returns} a utility value
\begin{itemize}
  \item if terminal-test(state) \textbf{then return} payoff(state)
  \item `v` ← – infinity
  \item \textbf{for} `a` in Actions(state) \textbf{do}
  \item \hspace{1em} `v` ← \text{\textit{Max(`v, Negamax-value(Result(s,a)))}}
  \item \textit{return} \text{v}
\end{itemize}

---

**Negamax Trick**

Rather than MIN, just treat opponent as MAX with negated score.

**Function** `Minimax-decision(state)` \textbf{returns} an action
\begin{itemize}
  \item \textit{inputs:} current state in game
  \item `v` ← `Negamax-value(state)`
  \item \text{\textit{return the action in Actions(state) with value}} \text{v}
\end{itemize}

---

**Function** `Negamax-Value` \textbf{returns} a utility value
\begin{itemize}
  \item if terminal-test(state) \textbf{then return} payoff(state)
  \item `v` ← – infinity
  \item \textbf{for} `a` in Actions(state) \textbf{do}
  \item \hspace{1em} `v` ← \text{\textit{Max(`v, Negamax-value(Result(s,a)))}}
  \item \textit{return} \text{v}
Search Complexity

- Time: $O(b^m)$
  - $b$ branching factor, $m$ max depth
- For chess, $b = 35$ and $m = 100$
- $35^{100} \approx 10^{154}$
- Suppose we had $10^{100}$ processors, each exploring $10^{18}$ nodes/second

Multi-Player Games

Max$^N$ algorithm
- conceptually similar to Minimax
- value of terminal node: an $N$-vector
- value of intermediate node: that of best successor for currently-moving player
Static Evaluation

- Provide direct evaluation of intermediate game states
- Should represent expected utility, e.g., probability of winning
- Typically, searching deeper yields better evaluations
- But horizon and other effects suggest searching to quiescence

Quiescence Search

- Horizon effect
  - Searching to fixed depth misses important moves just past stopping point
  - Paths may look good because they delay inevitable disaster
- Remedy
  - Continue searching until no more “violent” (big impact) moves possible
  - Must be selective to avoid blowup (e.g., just search captures)
Pruning Game Trees

Alpha-Beta Search

- Maintain at each node $n$:
  - $\alpha$: value of best choice found so far for any MAX node on path to $n$
  - $\beta$: value of best choice found so far for any MIN node on path to $n$
- During minimax search, prune subtree (terminate recursive call) whenever worse than current $\alpha$ or $\beta$ value.
Example

Search can be stopped below any MIN node having a beta value less than or equal to the alpha value of any of its MAX ancestors.

Example

Search can be stopped below any MAX node having $\alpha \geq \beta$ of any of its MIN ancestors.
\( \alpha-\beta \) Pruning Effectiveness

- Order of generating successors matters
- In worst-case, still \( O(b^m) \)
  - (how?)
- Best case:
  - examine successors in most favorable order
  - \( O(b^{m/2}) \)
- Average case:
  - examine successors in random order
  - \( O(b^{3m/4}) \)

Transposition Table

- Hash table of previously seen positions, with cached values (or bounds)
- Avoids repeating search on multiple branches of search tree
- Can dramatically deepen search
- But expensive therefore must be used selectively
Games of Chance

- Uncertainty also relevant for games
- Can extend turn-playing model to include chance elements
- E.g., backgammon

Backgammon Tree Fragment

Can solve using expectiminimax algorithm
**Expectiminimax**

Expectiminimax(n) = 
Payoff(n) if n is terminal state
max\(_s\) in Successors(n) Expectiminimax(s) if n is MAX node
min\(_s\) in Successors(n) Expectiminimax(s) if n is a MIN node
\(\sum_{s\in \text{Successors}(n)} \text{Pr}(s) \text{ Expectiminimax}(s)\) if n is CHANCE node

**State-of-Art (Chess)**

- **Deep Blue** defeats world champion (Kasparov), May 1997
- Today: PC chess programs beat almost everybody
- **Deep Fritz** defeats world champion (Kramnik), Nov 2006
  - Multiprocessor version of commercial program Fritz
- **Rybka**: current World Computer Chess champion
  - Off-the-shelf 8-core 3.2 GHz processor
Checkers is Solved

- Announced in Science, July 2007
  - Schaeffer et al., U Alberta
- State space > $10^{20}$: Largest game solved to date
- Massive parallel search
  - Endgame databases computed 1989–2005
  - Forward search 2005–2007
  - 50+ dedicated processors over much of search time
- 18 years: Longest successful computation?

Search Process

Forward search to endgame databases
- Solve selected 3-move openings, construct overall proof from these
- Proof-number search, using Chinook evaluations, iterating on error tolerances

39T