# CS 188: Artificial Intelligence
## Spring 2006

Lecture 23: Games

4/18/2006

Dan Klein – UC Berkeley

# Game Playing in Practice

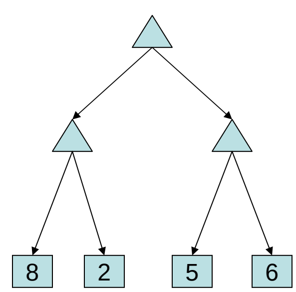
- Checkers: Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used an endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 443,748,401,247 positions. Exact solution imminent.
- Chess: Deep Blue defeated human world champion Gary Kasparov in a six-game match in 1997. Deep Blue examined 200 million positions per second, used very sophisticated evaluation and undisclosed methods for extending some lines of search up to 40 ply.
- 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.

# Game Playing

- Axes:
  - Deterministic or not
  - Number of players
  - Perfect information or not
- Want algorithms for calculating a strategy (policy) which recommends a move in each state

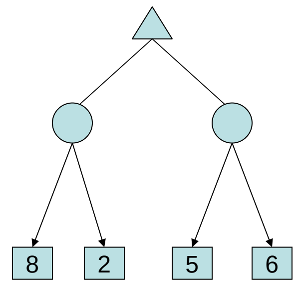
# Deterministic Single Player?

- Deterministic, single player, perfect information:
  - Know the rules
  - Know what moves will do
  - Have some utility function over outcomes
  - E.g. Freecell, 8-Puzzle, Rubik's cube
- … it's (basically) just search!
- Slight reinterpretation:
  - Calculate best utility from each node
  - Each node is a max over children
  - Note that goal values are on the goal, not path sums as before



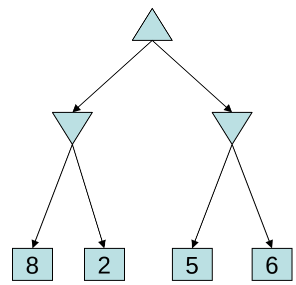
## Stochastic Single Player

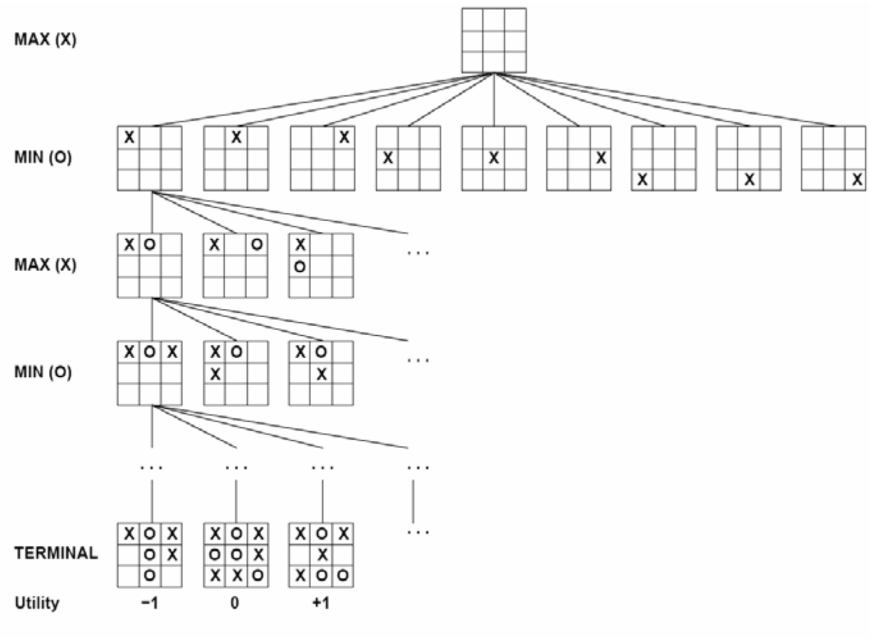
- What if we don't know what the result of an action will be?
  - E.g. solitaire, minesweeper, trying to drive home
- … just an MDP!
- Can also do expectimax search
  - Chance nodes, like actions except the environment controls the action chosen
  - Calculate utility for each node
  - Max nodes as in search
  - Chance nodes take expectations of children



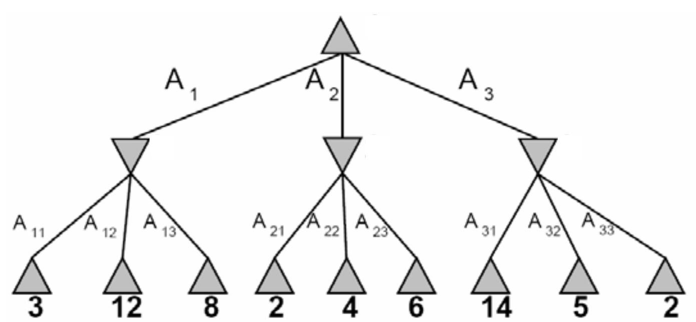

## Deterministic Two Player (Turns)

- E.g. tic-tac-toe
- Minimax search
  - Basically, a state-space search tree
  - Each layer, or ply, alternates players
  - Choose move to position with highest minimax value = best achievable utility against best play
- Zero-sum games
  - One player maximizes result
  - The other minimizes result






# Minimax Example



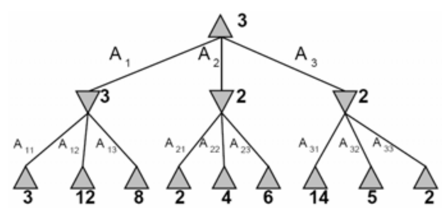
# Minimax Search

```
function MAX-VALUE(state) returns a utility value
   if TERMINAL-TEST(state) then return UTILITY(state)
   v ← −∞
   for a, s in SUCCESSORS(state) do v ← MAX(v, MIN-VALUE(s))
   return v

function MIN-VALUE(state) returns a utility value
   if TERMINAL-TEST(state) then return UTILITY(state)
   v ← ∞
   for a, s in SUCCESSORS(state) do v ← MIN(v, MAX-VALUE(s))
   return v
```

# Minimax Properties

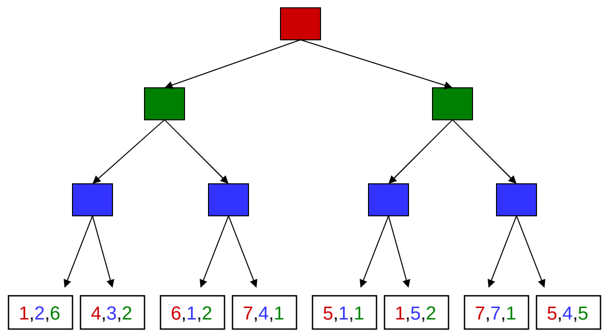
- Optimal against a perfect player. Otherwise?
- Time complexity?
  - $O(b^m)$
- Space complexity?
  - O(bm)




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

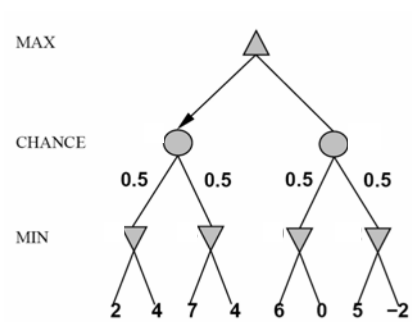
# Multi-Player Games

- Similar to minimax:
  - Utilities are now tuples
  - Each player maximizes their own entry at each node
  - Propagate (or back up) nodes from children




# Games with Chance

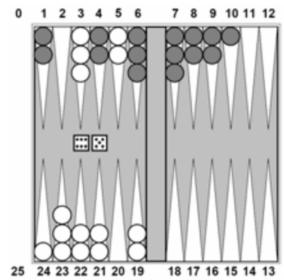
- E.g. backgammon
- Expectiminimax search!
  - Environment is an extra player than moves after each agent
  - Chance nodes take expectations, otherwise like minimax




**if** *state* is a MAX node **then**
  **return** the highest EXPECTIMINIMAX-VALUE of SUCCESSORS(*state*)
**if** *state* is a MIN node **then**
  **return** the lowest EXPECTIMINIMAX-VALUE of SUCCESSORS(*state*)
**if** *state* is a chance node **then**
  **return** average of EXPECTIMINIMAX-VALUE of SUCCESSORS(*state*)

# Games with Chance

- Dice rolls increase b: 21 possible rolls with 2 dice
  - Backgammon ≈ 20 legal moves
  - Depth 4 = 20 x $(21 \times 20)^3$  $1.2 \times 10^9$
- As depth increases, probability of reaching a given node shrinks
  - So value of lookahead is diminished
  - So limiting depth is less damaging
  - But pruning is less possible…
- TDGammon uses depth-2 search + very good eval function + reinforcement learning: world-champion level play




# Games with Hidden Information

- Imperfect information:
  - E.g., 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
- Drawback to this approach?
  - It's broken!
  - (Though useful in practice)

## Averaging over Deals is Broken

- Road A leads to a small heap of gold pieces
- Road B leads to a fork:
  - take the left fork and you'll find a mound of jewels;
  - take the right fork and you'll be run over by a bus.

- Road A leads to a small heap of gold pieces
- Road B leads to a fork:
  - take the left fork and you'll be run over by a bus;
  - take the right fork and you'll find a mound of jewels.

- Road A leads to a small heap of gold pieces
- Road B leads to a fork:
  - guess correctly and you'll nd a mound of jewels;
  - guess incorrectly and you'll be run over by a bus.

## Efficient Search

- Several options:
  - Pruning: avoid regions of search tree which will never enter into (optimal) play
  - Limited depth: don't search very far into the future, approximate utility with a value function (familiar?)

# Next Class

- More game playing
  - Pruning
  - Limited depth search
  - Connection to reinforcement learning!