

# Adversarial Search

Hal Daumé III

Computer Science  
University of Maryland

me@hal3.name

CS 421: Introduction to Artificial Intelligence

9 Feb 2012

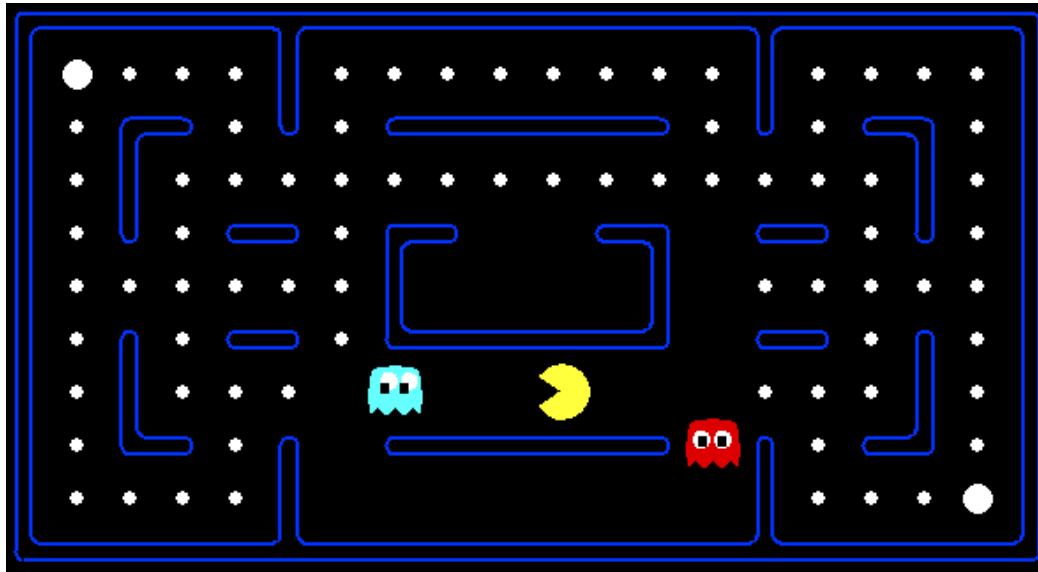


Many slides courtesy of  
Dan Klein, Stuart Russell,  
or Andrew Moore

# Announcements

➤ None

# Adversarial Search



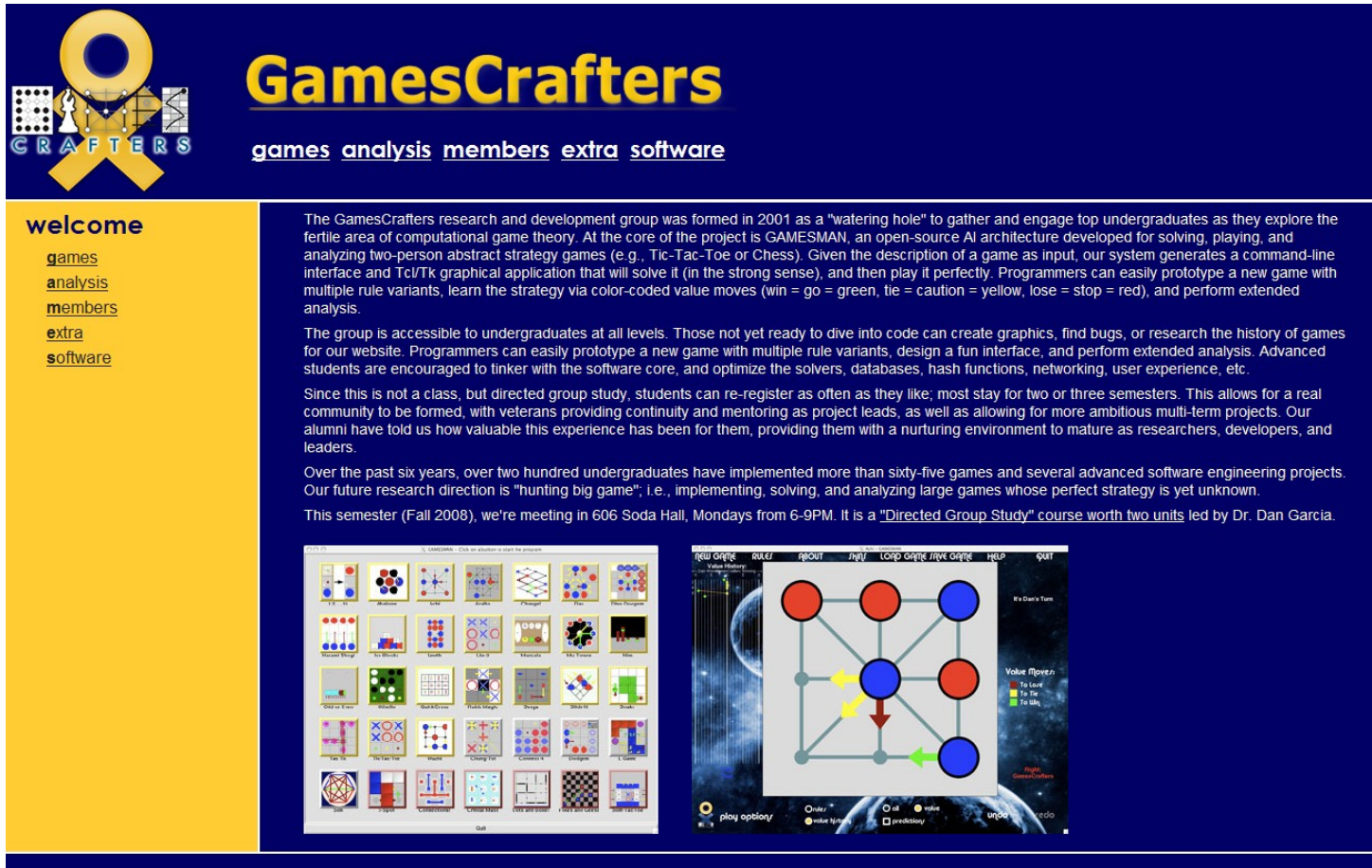
[DEMO: mystery  
pacman]

# Game Playing State-of-the-Art

- **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. Checkers is now solved!
- **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, which are too good.
- **Go:** human champions refuse to compete against computers, which are too bad. In go,  $b > 300$ , so most programs use pattern knowledge bases to suggest plausible moves.
- **Pacman:** unknown

# GamesCrafters

<http://gamescrafters.berkeley.edu/>



The screenshot shows the GamesCrafters website. At the top left is a logo with a yellow ribbon and the text 'GAMES CRAFTERS'. To the right of the logo is the title 'GamesCrafters' in large yellow font, with a navigation menu below it containing 'games', 'analysis', 'members', 'extra', and 'software'. On the left side of the main content area is a yellow sidebar with the word 'welcome' and the same navigation menu. The main content area has a dark blue background with white text. It contains three paragraphs of text describing the group's history, accessibility, and research direction. At the bottom of the main content area are two screenshots: the left one shows a grid of 48 small game icons, and the right one shows a game interface with a grid of nodes and colored arrows, and a legend for 'Value (green)'.

**GamesCrafters**  
games analysis members extra software

welcome  
games  
analysis  
members  
extra  
software

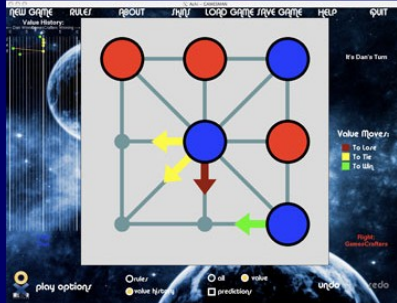
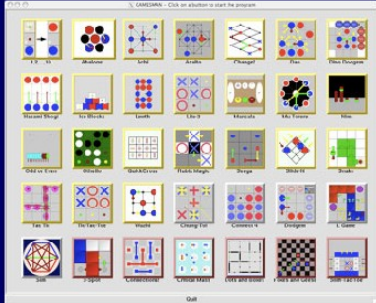
The GamesCrafters research and development group was formed in 2001 as a "watering hole" to gather and engage top undergraduates as they explore the fertile area of computational game theory. At the core of the project is GAMESMAN, an open-source AI architecture developed for solving, playing, and analyzing two-person abstract strategy games (e.g., Tic-Tac-Toe or Chess). Given the description of a game as input, our system generates a command-line interface and Tcl/Tk graphical application that will solve it (in the strong sense), and then play it perfectly. Programmers can easily prototype a new game with multiple rule variants, learn the strategy via color-coded value moves (win = go = green, tie = caution = yellow, lose = stop = red), and perform extended analysis.

The group is accessible to undergraduates at all levels. Those not yet ready to dive into code can create graphics, find bugs, or research the history of games for our website. Programmers can easily prototype a new game with multiple rule variants, design a fun interface, and perform extended analysis. Advanced students are encouraged to tinker with the software core, and optimize the solvers, databases, hash functions, networking, user experience, etc.

Since this is not a class, but directed group study, students can re-register as often as they like; most stay for two or three semesters. This allows for a real community to be formed, with veterans providing continuity and mentoring as project leads, as well as allowing for more ambitious multi-term projects. Our alumni have told us how valuable this experience has been for them, providing them with a nurturing environment to mature as researchers, developers, and leaders.

Over the past six years, over two hundred undergraduates have implemented more than sixty-five games and several advanced software engineering projects. Our future research direction is "hunting big game"; i.e., implementing, solving, and analyzing large games whose perfect strategy is yet unknown.

This semester (Fall 2008), we're meeting in 606 Soda Hall, Mondays from 6-9PM. It is a "Directed Group Study" course worth two units led by Dr. Dan Garcia.



# Game Playing

➤ Many different kinds of games!

➤ Axes:

➤ Deterministic

➤ One, two or

➤ Perfect info

➤ Want algorithm  
which recon

## Examples?

Deterministic, 1 player, perfect information?

Deterministic, 1 player, imperfect information?

Deterministic, >1 player, perfect information?

Deterministic, >1 player, imperfect information?

Stochastic, 1 player, perfect information?

Stochastic, 1 player, imperfect information?

Stochastic, >1 player, perfect information?

Stochastic, >1 player, imperfect information?

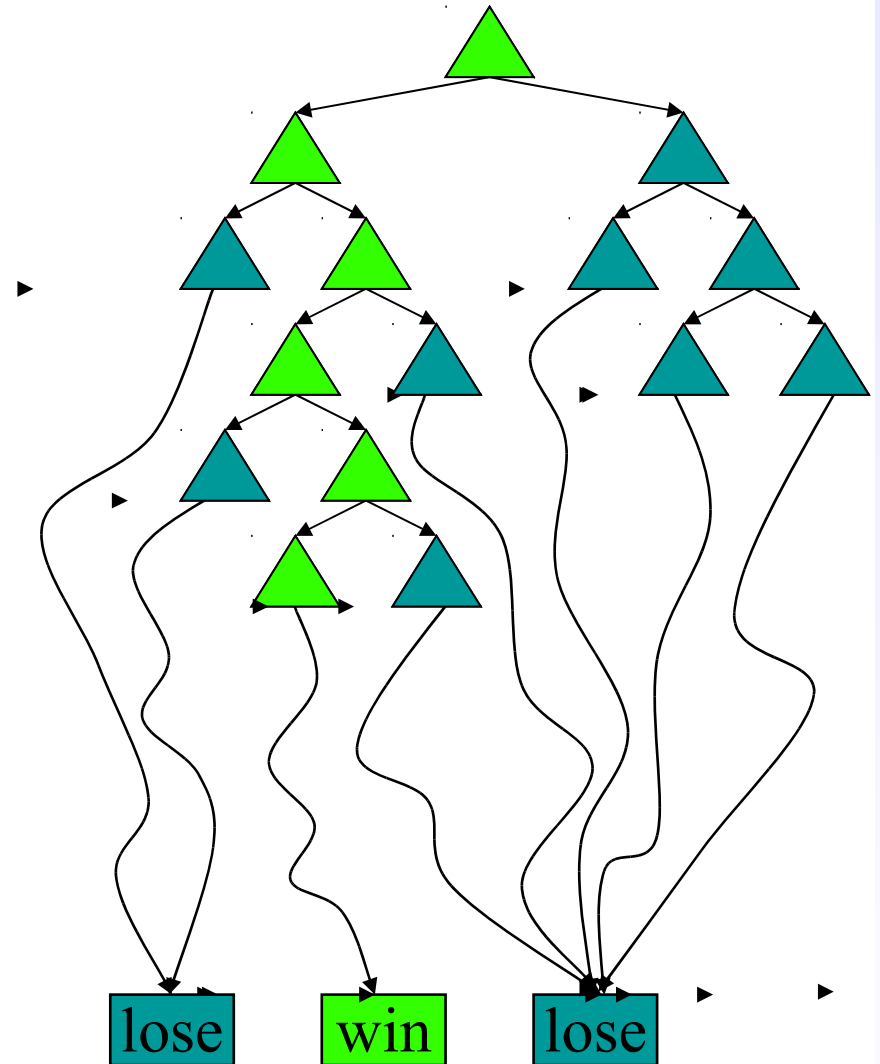
<http://u.hal3.name/ic.pl?q=game>

# Deterministic Games

- Many possible formalizations, one is:
  - States:  $S$  (start at  $s_0$ )
  - Players:  $P=\{1\dots N\}$  (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:  $S \times P \rightarrow R$
  
- Solution for a player is a policy:  $S \rightarrow A$

# Deterministic Single-Player?

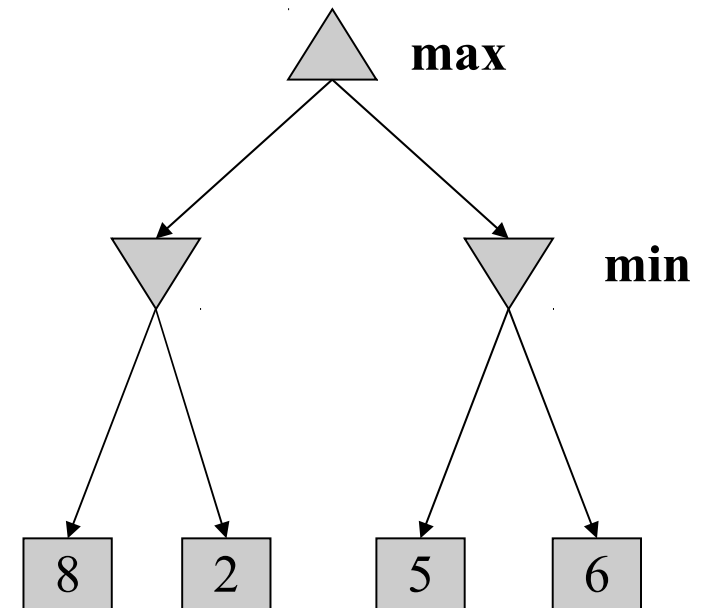
- Deterministic, single player, perfect information:
  - Know the rules
  - Know what actions do
  - Know when you win
  - E.g. Freecell, 8-Puzzle, Rubik's cube
- ... it's just search!
- Slight reinterpretation:
  - Each node stores a **value**: the best outcome it can reach
  - This is the maximal outcome of its children
  - Note that we don't have path sums as before (utilities at end)
- After search, can pick move that leads to best node



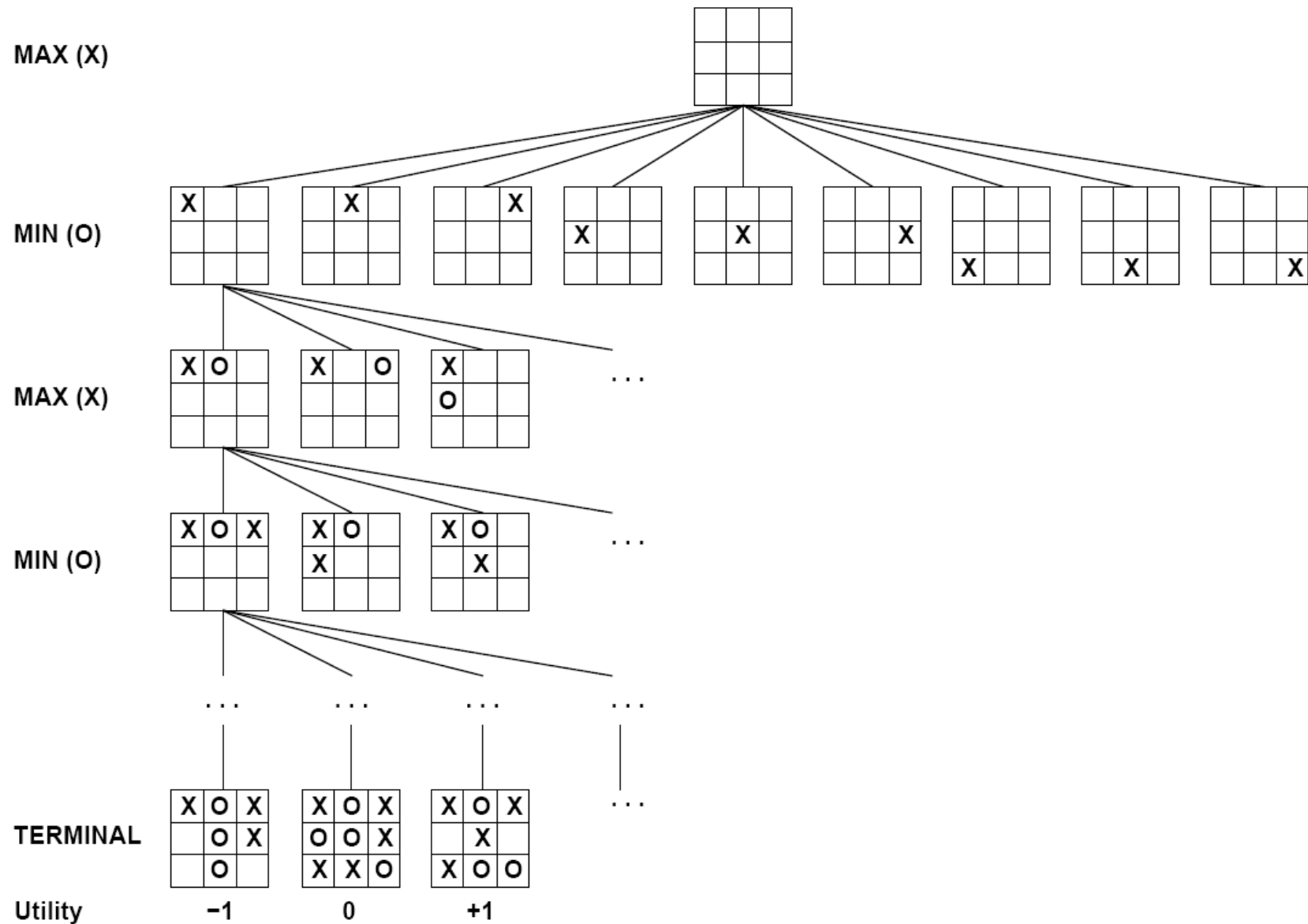


# Deterministic Two-Player

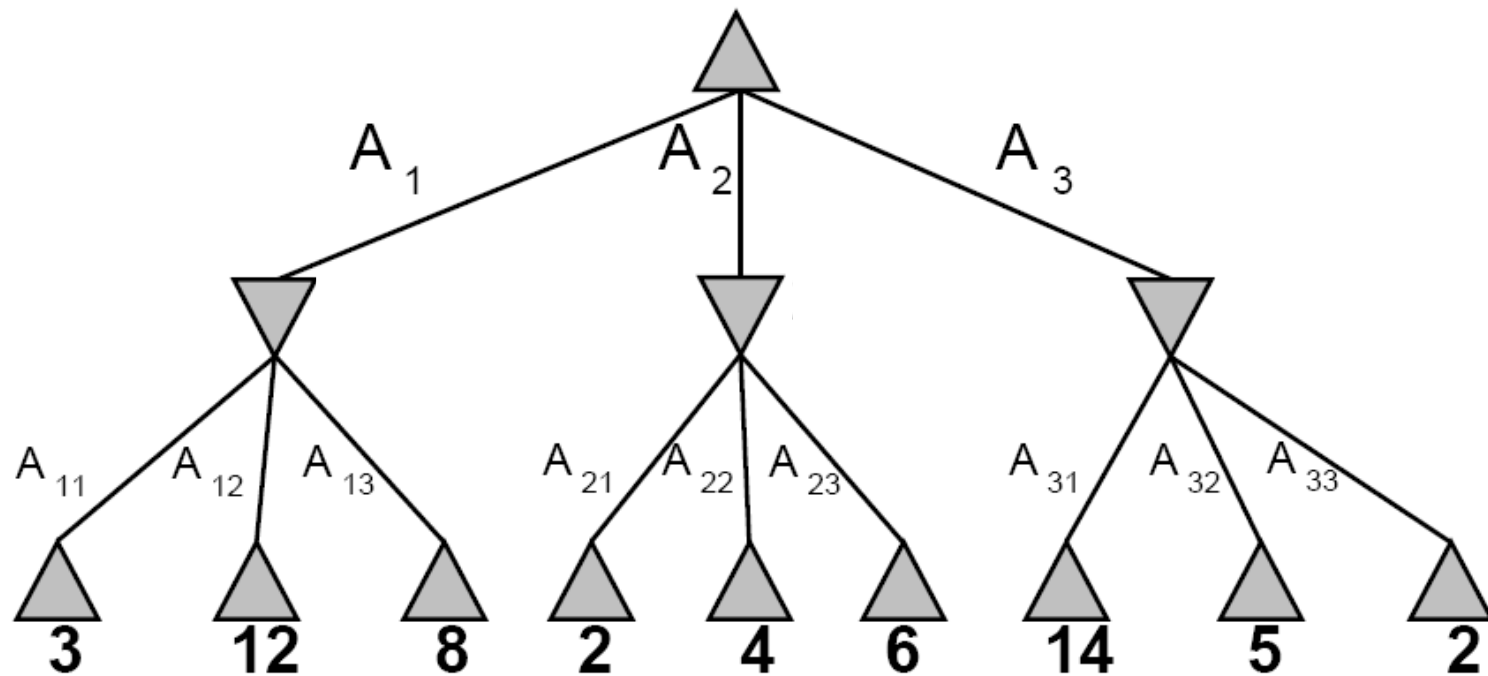
- E.g. tic-tac-toe, chess, checkers
- **Minimax search**
  - A state-space search tree
  - Players alternate
  - Each layer, or ply, consists of a round of moves
  - Choose move to position with highest **minimax value** = best achievable utility against best play
- **Zero-sum games**
  - One player maximizes result
  - The other minimizes result



# Tic-tac-toe Game Tree



# Minimax Example



# Minimax Search

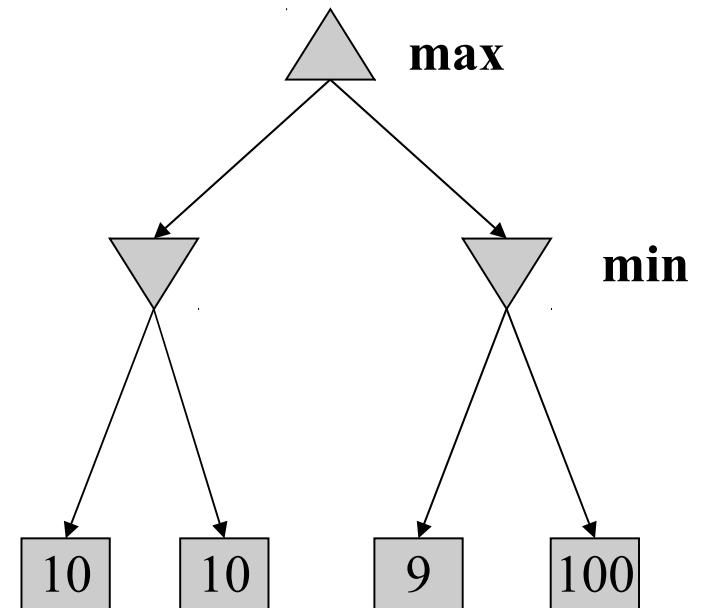
**function** **MAX-VALUE**(*state*) **returns** *a utility value*  
  **if** **TERMINAL-TEST**(*state*) **then return** **UTILITY**(*state*)  
   $v \leftarrow -\infty$   
  **for**  $a, s$  **in** **SUCCESSORS**(*state*) **do**  $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(s))$   
  **return**  $v$

---

**function** **MIN-VALUE**(*state*) **returns** *a utility value*  
  **if** **TERMINAL-TEST**(*state*) **then return** **UTILITY**(*state*)  
   $v \leftarrow \infty$   
  **for**  $a, s$  **in** **SUCCESSORS**(*state*) **do**  $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(s))$   
  **return**  $v$

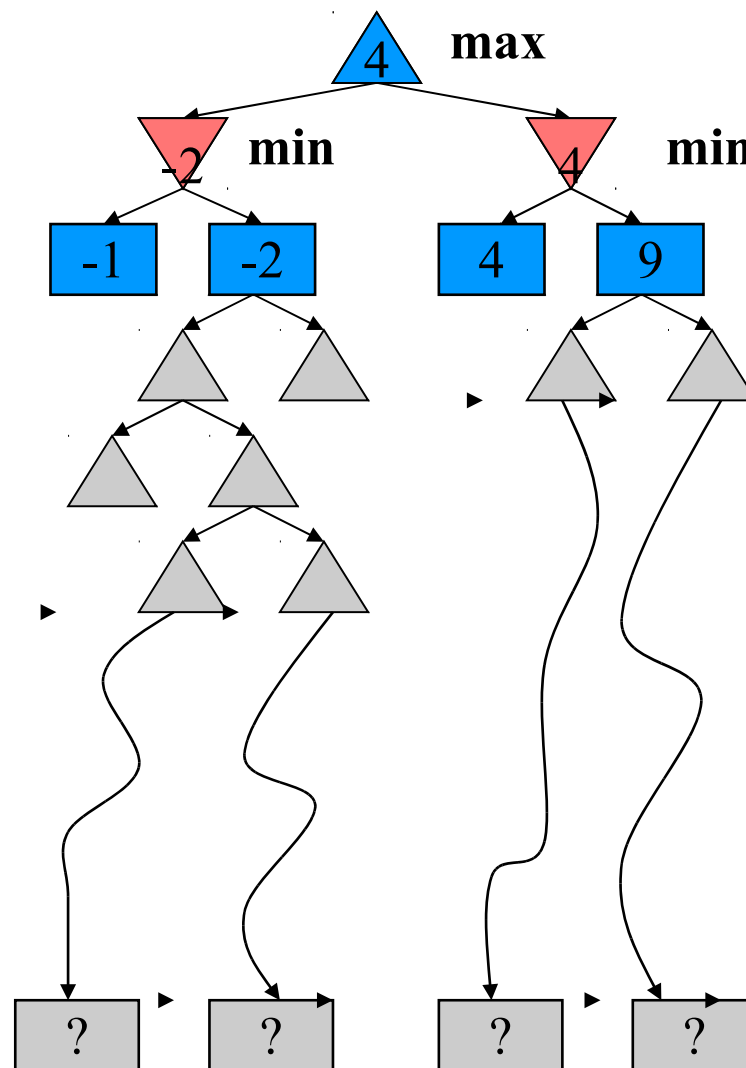
# Minimax Properties

- 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?



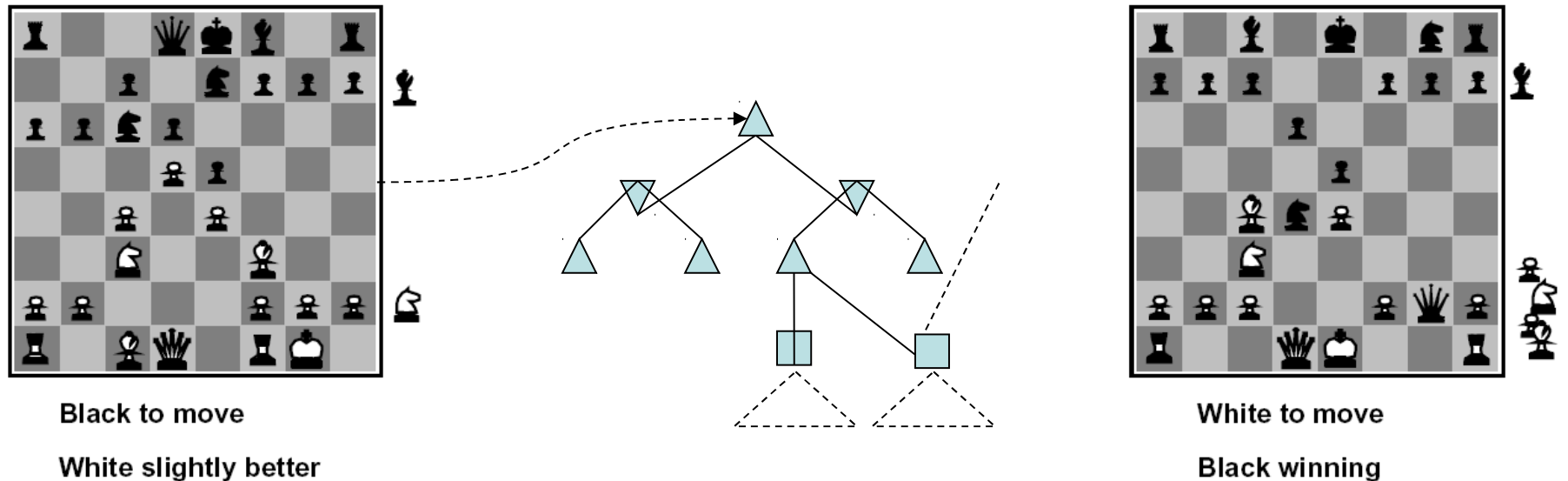
# Resource Limits

- Cannot search to leaves
- Depth-limited search
  - Instead, search a limited depth of tree
  - Replace terminal utilities with an eval function for non-terminal positions
- Guarantee of optimal play is gone
- More plies makes a BIG difference
  - [DEMO: limitedDepth]
- Example:
  - Suppose we have 100 seconds, can explore 10K nodes / sec
  - So can check 1M nodes per move
  - $\alpha$ - $\beta$  reaches about depth 8 – decent chess program



# Evaluation Functions

- Function which scores non-terminals

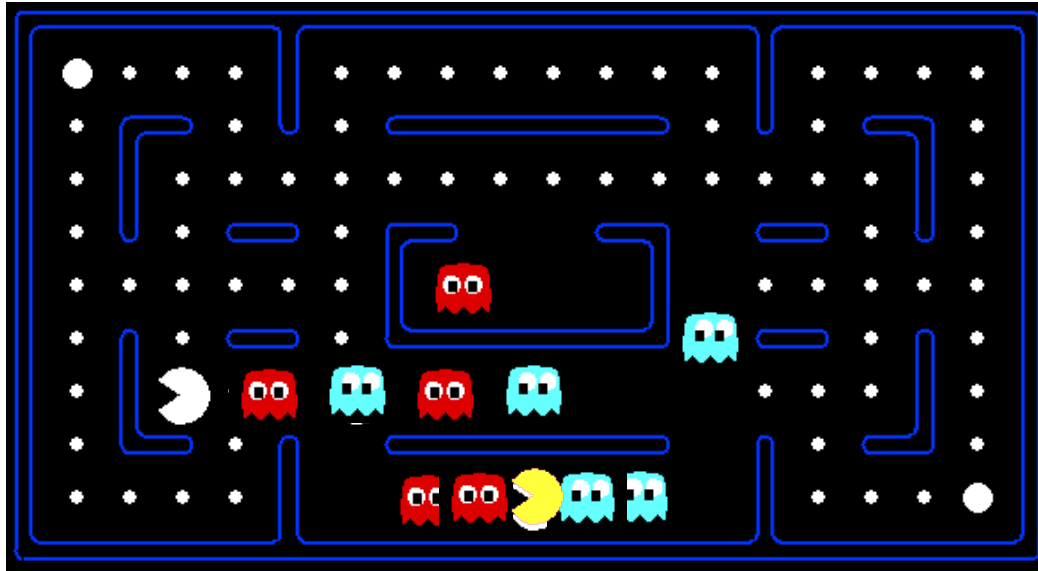


- Ideal function: returns the utility of the position
- In practice: typically weighted linear sum of features:

$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \dots + w_n f_n(s)$$

- e.g.  $f_1(s) = (\text{num white queens} - \text{num black queens})$ , etc.

# Evaluation for Pacman



[DEMO: thrashing, smart ghosts]

$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \dots + w_n f_n(s)$$

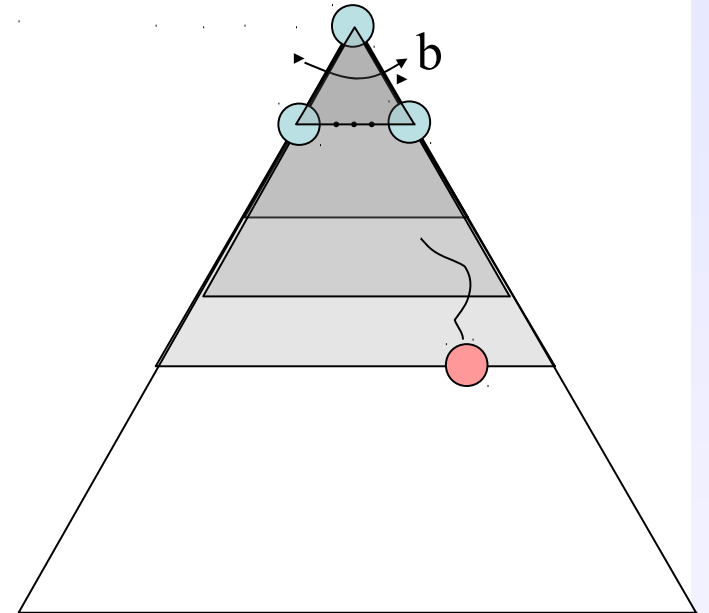


# Iterative Deepening

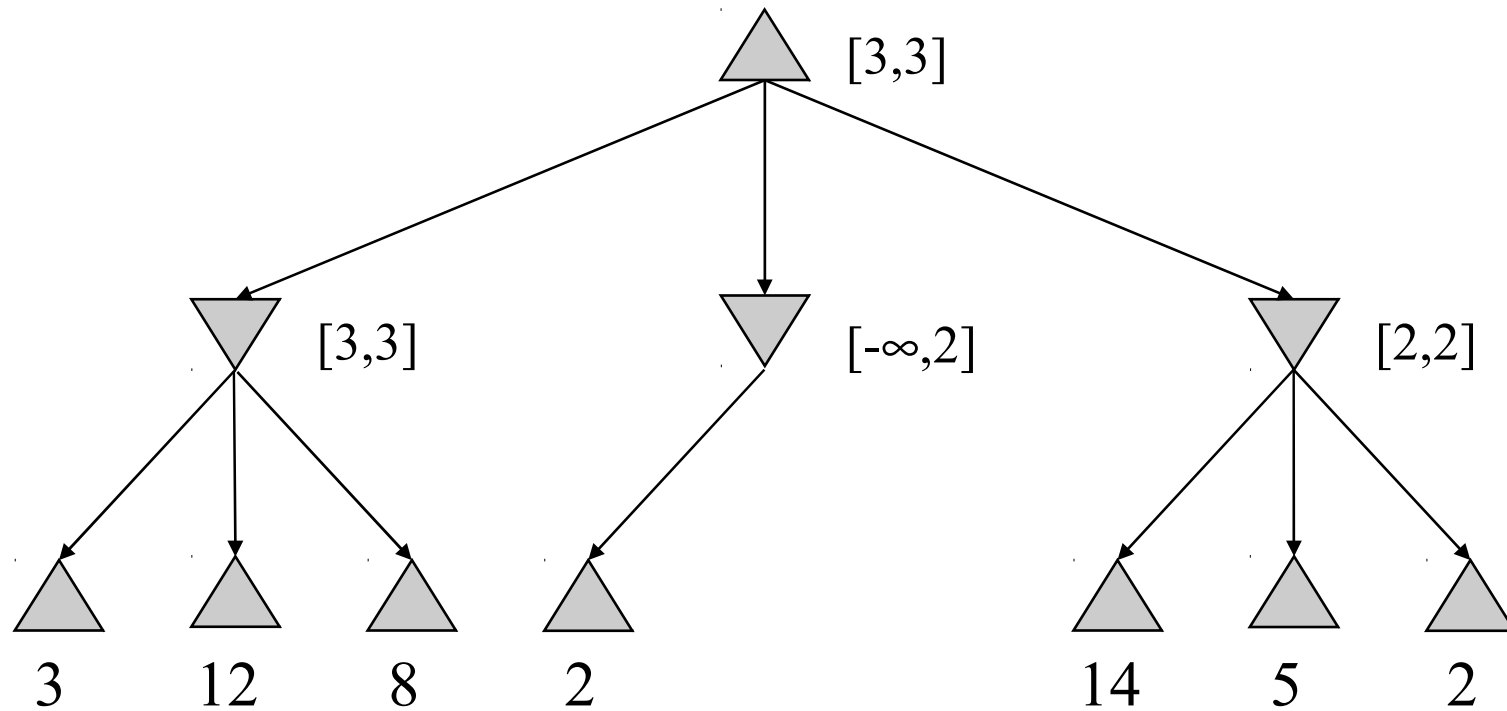
Iterative deepening uses DFS as a subroutine:

1. Do a DFS which only searches for paths of length 1 or less. (DFS gives up on any path of length 2)
2. If “1” failed, do a DFS which only searches paths of length 2 or less.
3. If “2” failed, do a DFS which only searches paths of length 3 or less.  
....and so on.

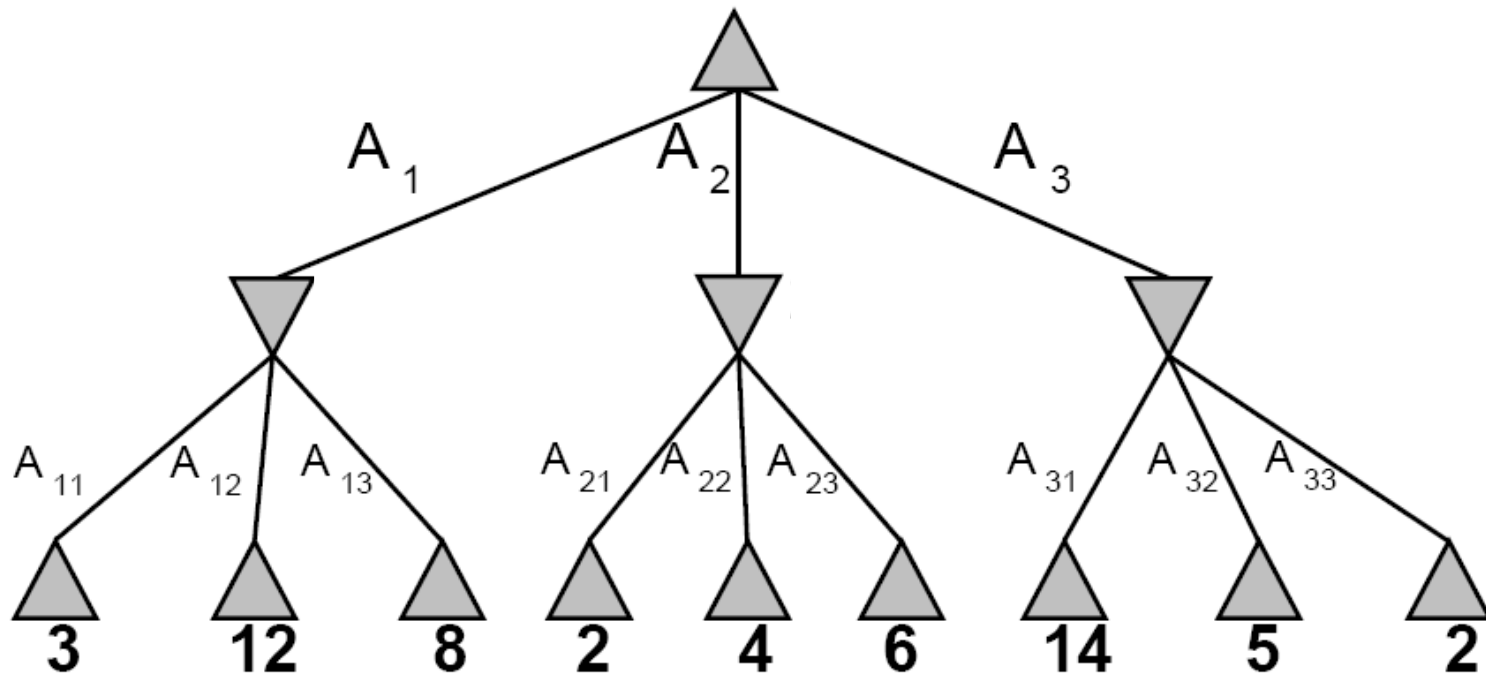
This works for single-agent search as well!  
Why do we want to do this for multiplayer games?



# Pruning in Minimax Search

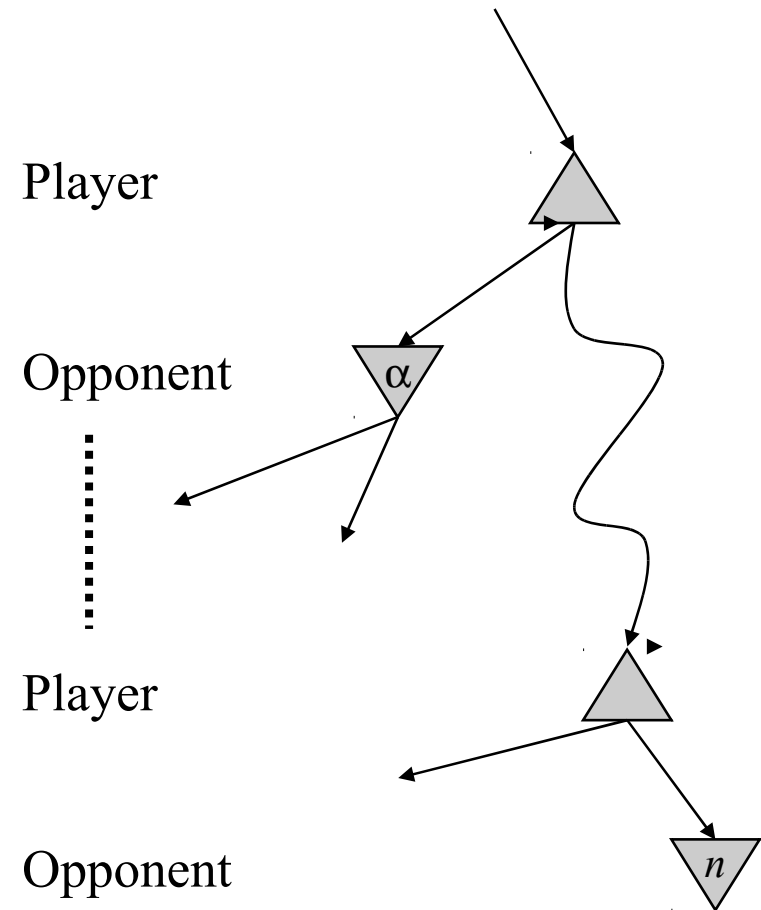


# $\alpha$ - $\beta$ Pruning Example



# $\alpha$ - $\beta$ Pruning

- General configuration
- $\alpha$  is the best value that MAX can get at any choice point along the current path
- If  $n$  becomes worse than  $\alpha$ , MAX will avoid it, so can stop considering  $n$ 's other children
- Define  $\beta$  similarly for MIN

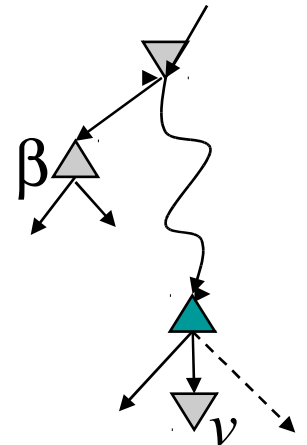


# $\alpha$ - $\beta$ Pruning Pseudocode

```
function MAX-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
   $v \leftarrow -\infty$ 
  for  $a, s$  in SUCCESSORS(state) do  $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(s))$ 
  return  $v$ 
```

---

```
function MAX-VALUE(state,  $\alpha$ ,  $\beta$ ) returns a utility value
  inputs: state, current state in game
            $\alpha$ , the value of the best alternative for MAX along the path to state
            $\beta$ , the value of the best alternative for MIN along the path to state
  if TERMINAL-TEST(state) then return UTILITY(state)
   $v \leftarrow -\infty$ 
  for  $a, s$  in SUCCESSORS(state) do
     $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(s, \alpha, \beta))$ 
    if  $v \geq \beta$  then return  $v$ 
     $\alpha \leftarrow \text{MAX}(\alpha, v)$ 
  return  $v$ 
```

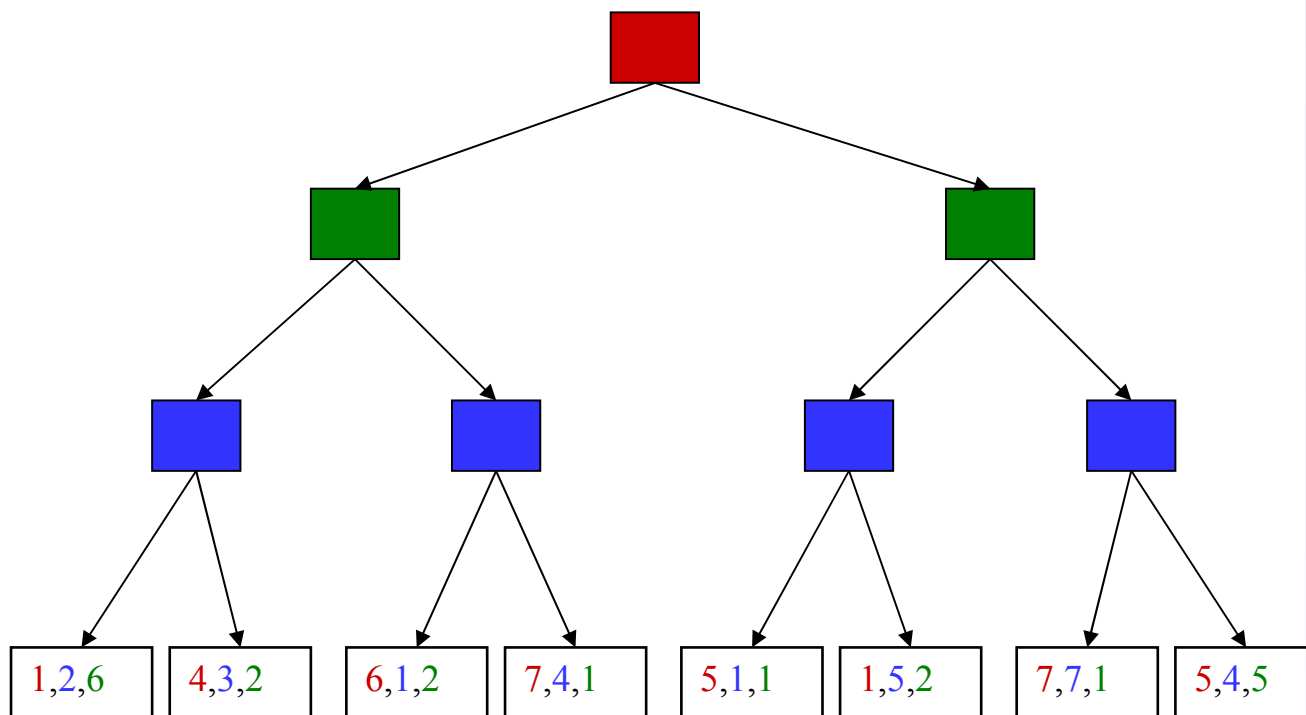


# $\alpha$ - $\beta$ Pruning Properties

- Pruning has **no effect** on final result
- Good move ordering improves effectiveness of pruning
- With “perfect ordering”:
  - Time complexity drops to  $O(b^{m/2})$
  - Doubles solvable depth
  - Full search of, e.g. chess, is still hopeless!
- A simple example of **metareasoning**, here reasoning about which computations are relevant

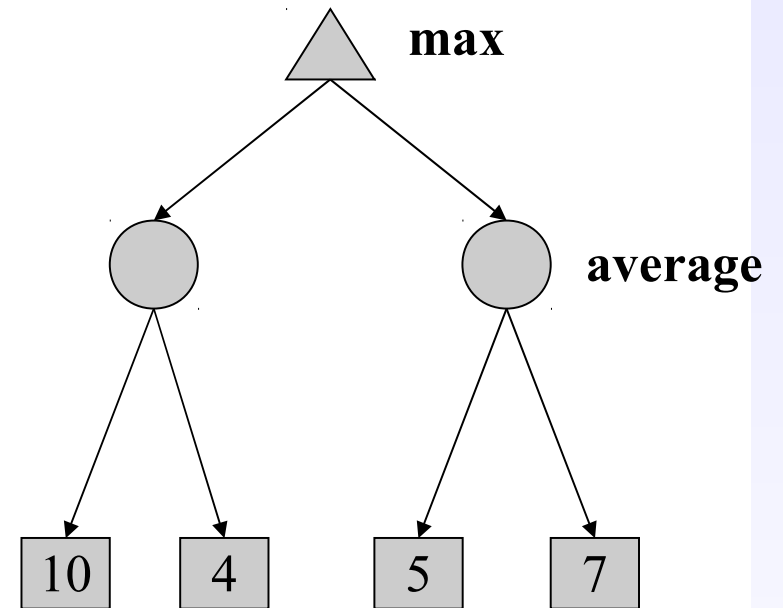
# Non-Zero-Sum Games

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



# Stochastic Single-Player

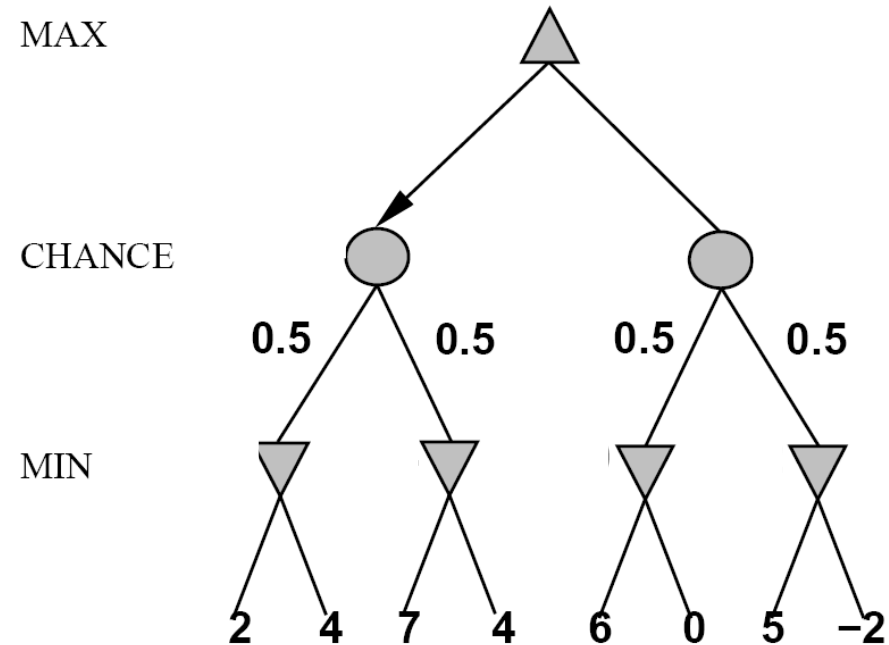
- What if we don't know what the result of an action will be? E.g.,
  - In solitaire, shuffle is unknown
  - In minesweeper, mine locations
  - In pacman, ghosts!
- Can do **expectimax search**
  - Chance nodes, like actions except the environment controls the action chosen
  - Calculate utility for each node
  - Max nodes as in search
  - Chance nodes take average (expectation) of value of children
- Later, we'll learn how to formalize this as a **Markov Decision Process**





# Stochastic Two-Player

- E.g. backgammon
- Expectiminimax (!)
- Environment is an extra player that 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

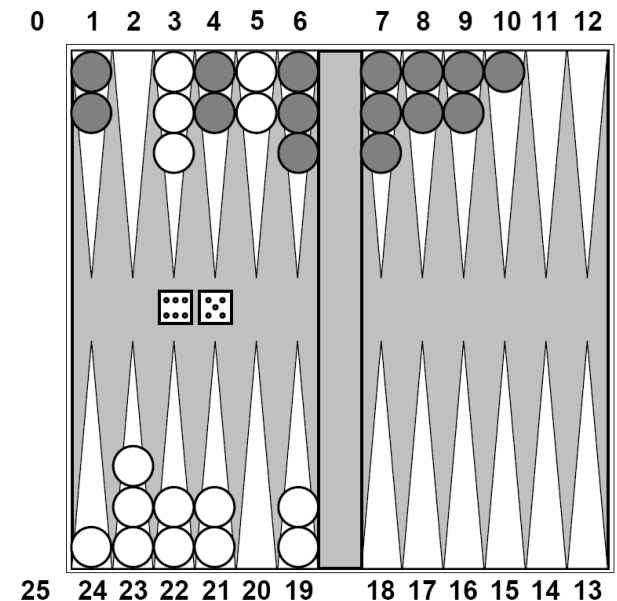
return the lowest EXPECTIMINIMAX-VALUE of SUCCESSORS(*state*)

if *state* is a chance node then

return average of EXPECTIMINIMAX-VALUE of SUCCESSORS(*state*)

# Stochastic Two-Player

- Dice rolls increase  $b$ : 21 possible rolls with 2 dice
  - Backgammon  $\approx$  20 legal moves
  - Depth 4 =  $20 \times (21 \times 20)^3 \approx 1.2 \times 10^9$
- 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



# What's Next?

- Make sure you know what:
  - Probabilities are
  - Expectations are
  - You should be able to do any exercise from:
    - <http://www.cs.umd.edu/class/fall2011/cmsc250-0x0x/hw/HW11.pdf>
    - Username and password are both “250”
  - If you can't, review your probability discrete math!
    - <http://www.cs.umd.edu/class/fall2011/cmsc250-0x0x/notes/CRASH.pdf>
- Next topics:
  - Dealing with uncertainty
  - How to learn evaluation functions
  - Markov Decision Processes