Lecture 17

Games and Adversarial Search

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

Slides by Stuart Russell and Peter Norvig

Course Overview

- ✔ Introduction
  - ✔ Artificial Intelligence
  - ✔ Intelligent Agents
- ✔ Search
  - ✔ Uninformed Search
  - ✔ Heuristic Search
- ✔ Uncertain knowledge and Reasoning
  - ✔ Probability and Bayesian approach
  - ✔ Bayesian Networks
  - ✔ Hidden Markov Chains
  - ✔ Kalman Filters
- ✔ Learning
  - ✔ Supervised
    Decision Trees, Neural Networks
    Learning Bayesian Networks
  - ✔ Unsupervised
    EM Algorithm
- ✔ Reinforcement Learning
- ▶ Games and Adversarial Search
  - ▶ Minimax search and Alpha-beta pruning
  - ▶ Multiagent search
- ▶ Knowledge representation and Reasoning
  - ▶ Propositional logic
  - ▶ First order logic
  - ▶ Inference
  - ▶ Plannning

Outline

- Games
- Perfect play
  - minimax decisions
  - α–β pruning
- Resource limits and approximate evaluation
- Games of chance
- Games with imperfect information

Outline

1. Introduction

2. Minimax

3. α–β Algorithm

4. Stochastic Games

Multiagent environments

Multiagent environments:

- cooperative
- competitive ➨ adversarial search in games

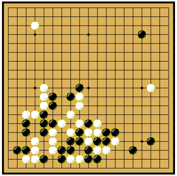
AI game theory (combinatorial game theory)

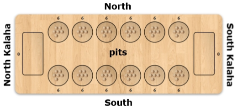
- deterministic/stochastic
- turn taking
- two players
- zero sum games = utility values equal and opposite
- perfect/imperfect information
- agents are restricted to a small number of actions described by rules

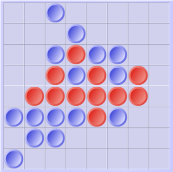
“Classical” (economic) game theory includes cooperation, chance, imperfect knowledge, simultaneous moves and tends to represent real-life decision making situations.

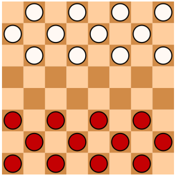
Types of Games

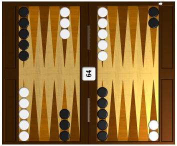
	deterministic	chance
perfect information	chess, checkers, kalaha go, othello	backgammon, monopoly
imperfect information	battleships, blind tictactoe	bridge, poker, scrabble

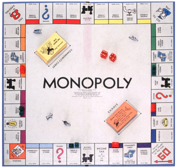












Games vs. search problems

"Unpredictable" opponent ⇒ solution is a strategy/policy
specifying a move for every possible opponent reply ➨ contingency strategy

Optimal strategy: the one that leads to outcomes at least as good as any other strategy when one is playing an infallibile opponent

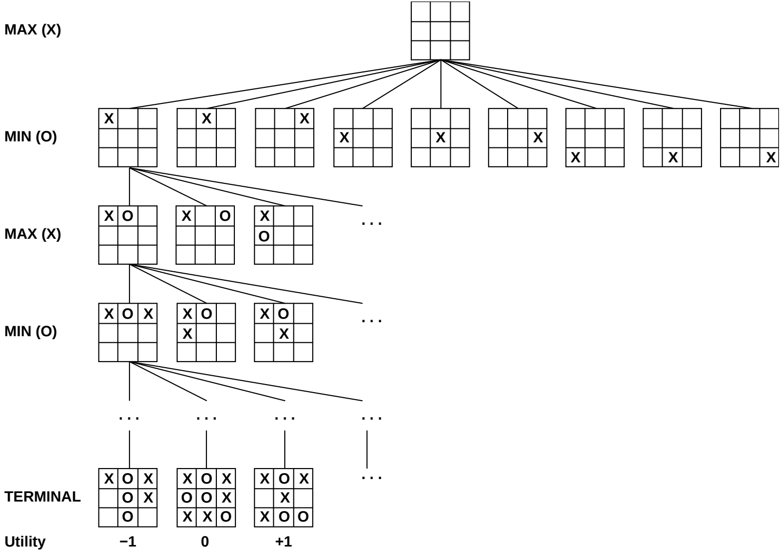
Search problem ⇝ game tree

- initial state: root of game tree
- successor function: game rules/moves
- terminal test (is the game over?)
- utility function, gives a value for terminal nodes (eg, +1, -1, 0)

Terminology:

- Two players called MAX and MIN.
- MAX searches the game tree.
- Ply: one turn (every player moves once) from "reply". [A. Samuel 1959]

Game tree (2-player, deterministic, turns)



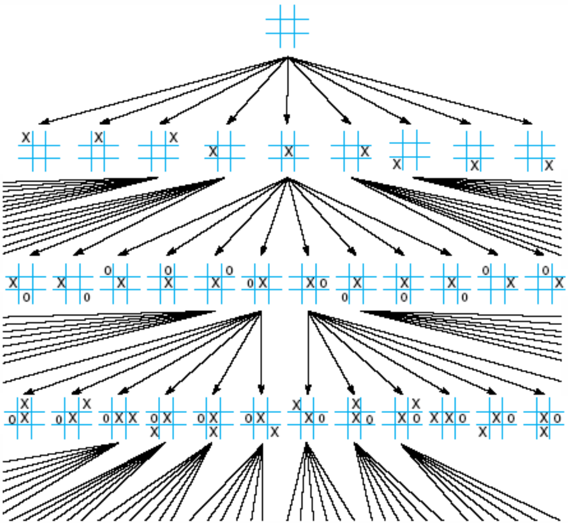
Measures of Game Complexity

- state-space complexity: number of legal game positions reachable from the initial position of the game.

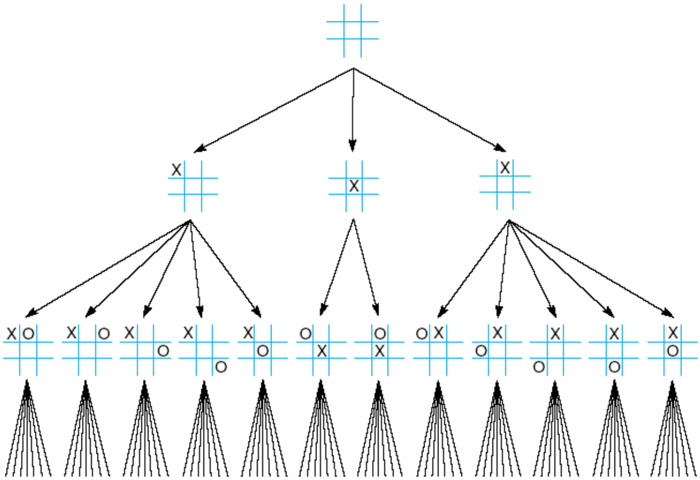
  an upper bound can often be computed by including illegal positions
  Eg, TicTacToe:
  $3^9 = 19.683$
  5.478 after removal of illegal
  765 essentially different positions after eliminating symmetries

- game tree size: total number of possible games that can be played: number of leaf nodes in the game tree rooted at the game's initial position.

  Eg: TicTacToe:
  $9! = 362.880$ possible games
  255.168 possible games halting when one side wins
  26.830 after removal of rotations and reflections



First three levels of the tic-tac-toe state space reduced by symmetry: $12 \times 7!$



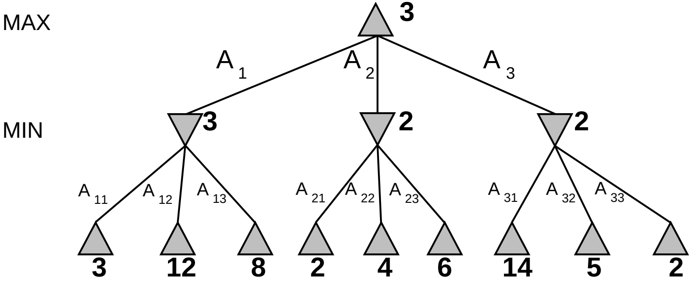
Outline

Minimax

Perfect play for deterministic, perfect-information games

Idea: choose move to position with highest minimax value ($\rightsquigarrow$utility for MAX)
= best achievable payoff against best play

E.g., 2-ply game:



Minimax algorithm

Recursive Depth First Search:

function MINIMAX-DECISION(*state*) **returns** *an action*
  return $\arg\max_{a \in \text{ACTIONS}(s)}$ MIN-VALUE(RESULT(*state*, *a*))

---

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

---

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

Properties of minimax

Complete?? Yes, if tree is finite (chess has specific rules for this)
Time complexity?? $O(b^m)$
Space complexity?? $O(bm)$ (depth-first exploration)

But do we need to explore every path?

Measures of Game Complexity

- game-tree complexity: number of leaf nodes in the smallest full-width decision tree that establishes the value of the initial position.
  A full-width tree includes all nodes at each depth.
  estimates the number of positions to evaluate in a minimax search to determine the value of the initial position.

  approximation: game's average branching factor to the power of the number of plies in an average game.
  Eg.: chess For chess, $b \approx 35$, $m \approx 100$ for "reasonable" games
  $\Rightarrow$ exact solution completely infeasible

- computational complexity applies to generalized games (eg, $n \times n$ boards)
  Eg: TicTacToe:
  $m \times n$ board k in a row solved in $DSPACE(mn)$ by searching the entire game tree

Historical view

Time limits $\Rightarrow$ unlikely to find goal, must approximate

Plan of attack:

- Computer considers possible lines of play (Babbage, 1846)
- Algorithm for perfect play - MINIMAX - (Zermelo, 1912; Von Neumann, 1944)
- Finite horizon, approximate evaluation (Zuse, 1945; Wiener, 1948; Shannon, 1950)
- First chess program (Turing, 1951)
- Machine learning to improve evaluation accuracy (Samuel, 1952–57)
- Pruning to allow deeper search - $\alpha - \beta$ alg. - (McCarthy, 1956)

Resource limits

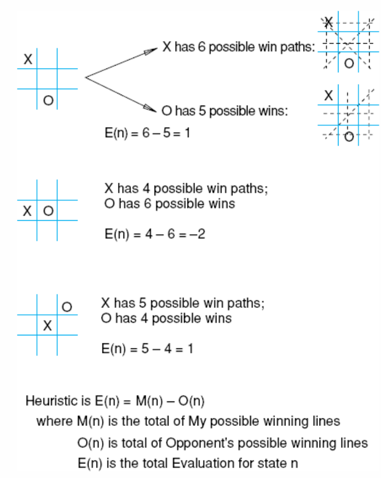
Standard approaches:

- n-ply lookahead: depth-limited search
- heuristic descent
- heuristic cutoff
  1. Use Cutoff-Test instead of Terminal-Test
     e.g., depth limit (perhaps add quiescence search)
  2. Use Eval instead of Utility
     i.e., evaluation function that estimates desirability of position

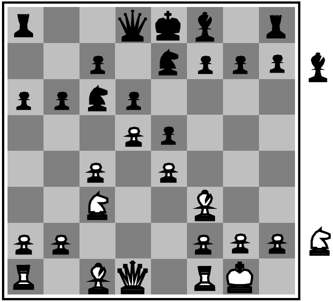
Suppose we have 100 seconds, explore 10^4 nodes/second
$\Rightarrow 10^6$ nodes per move $\approx 35^{8/2}$

Heuristic Descent

Heuristic measuring conflict applied to states of tic-tac-toe

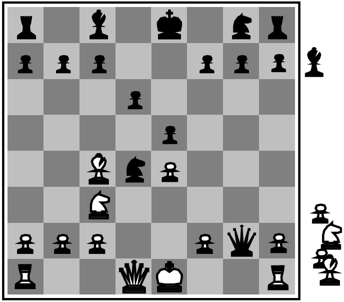


Evaluation functions



Black to move

White slightly better



White to move

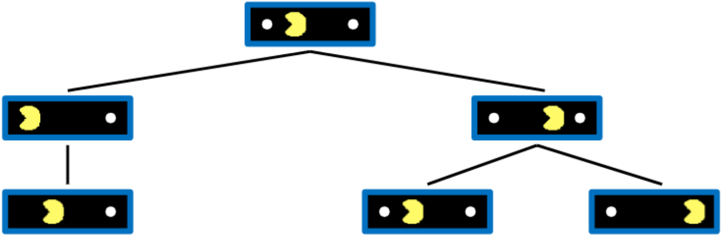
Black winning

For chess, typically linear weighted sum of features

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

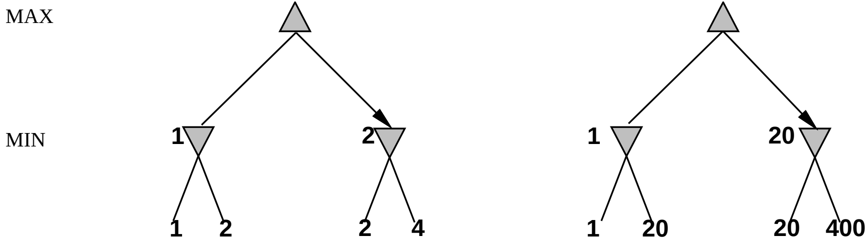
e.g., $w_1 = 9$ with
$f_1(s)$ = (number of white queens) – (number of black queens), etc.

Thrashing



- 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!

Digression: Exact values don't matter



Behaviour is preserved under any **monotonic** transformation of Eval
Only the order matters:
payoff in deterministic games acts as an ordinal utility function

Outline

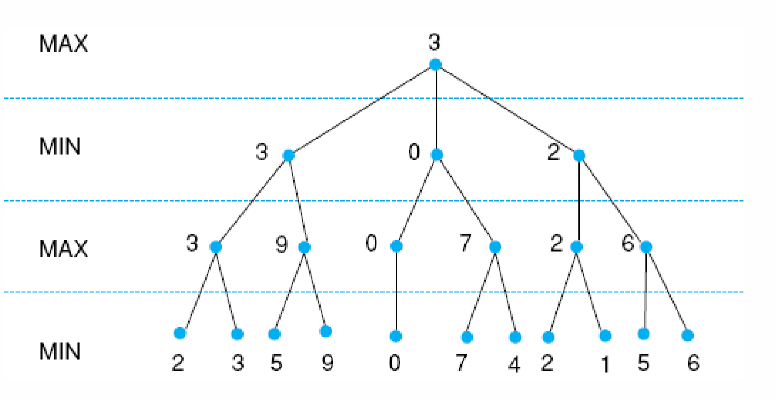
1. Introduction

2. Minimax

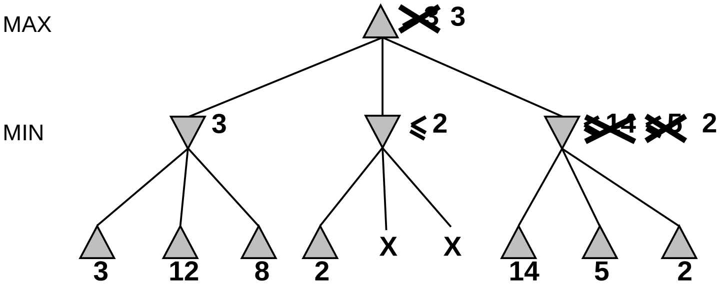
3. α–β Algorithm

4. Stochastic Games

Example

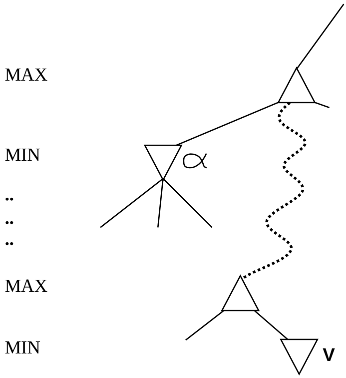


α–β pruning example



$$\text{Minimax}(\mathit{root}) = \max\{3, \min\{2, x, y\}, \min\{\ldots\}\}$$

Why is it called α–β?



α is the best value (to MAX) found so far along the current path
If V is worse ($<$) than α, MAX will avoid it $\Rightarrow$ prune that branch
Define β similarly for MIN

The α–β algorithm

α is the best value to MAX up to now for everything that comes above in the game tree. Similar for β and MIN.

```
function ALPHA-BETA-SEARCH(state) returns an action
  v ← MAX-VALUE(state, −∞, +∞)
  return the action in ACTIONS(state) with value v
```

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

```
function MIN-VALUE(state, α, β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← +∞
  for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(s,a) , α, β))
    if v ≤ α then return v
    β ← MIN(β, v)
  return v
```

Properties of α–β

- Pruning **does not** affect final result
- Good move ordering improves effectiveness of pruning
- With "perfect ordering," time complexity = $O(b^{m/2})$
  $\Rightarrow$ **doubles** solvable depth
- if b is relatively small, random orders leads to $O(b^{3m/4})$
- Unfortunately, 35^{50} is still impossible!

Deterministic games 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.
- Kalaha (6,6) solved at IMADA in 2011
- Chess: Deep Blue defeated human world champion Gary Kasparov in a six-game match in 1997. Deep Blue searches 200 million positions per second, uses 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.

Outline

Stochastic Games

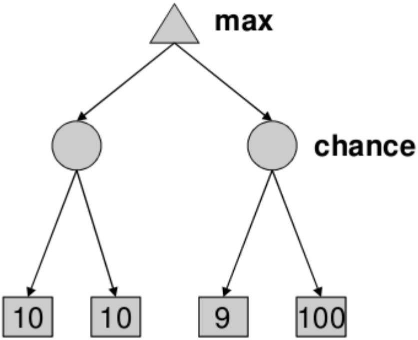
Uncertainty in the result of an action. Examples:

- In solitaire, next card is unknown
- In minesweeper, mine locations
- In pacman, the ghosts act randomly

Can do expectimax search to maximize average score

- Max nodes as in minimax search
- Chance nodes, like min nodes, except the outcome is uncertain
- Calculate expected utilities I.e. take weighted average (expectation) of values of children

Note, they can be formalized as Markov Decision Processes

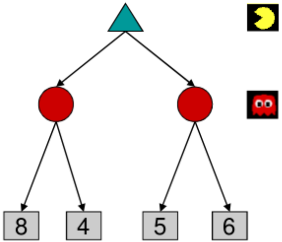


Expectimax Pseudocode

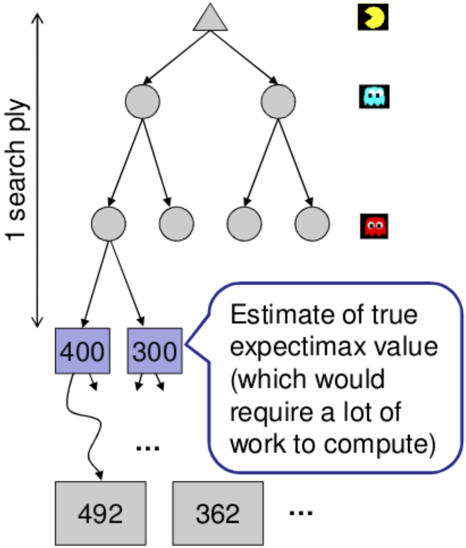
```
def value(s)
    if s is a max node return maxValue(s)
    if s is an exp node return expValue(s)
    if s is a terminal node return evaluation(s)

def maxValue(s)
    values = [value(s') for s' in successors(s)]
    return max(values)

def expValue(s)
    values = [value(s') for s' in successors(s)]
    weights = [probability(s, s') for s' in successors(s)]
    return expectation(values, weights)
```

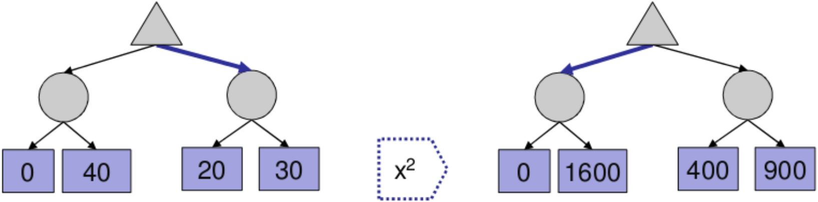


Depth-Limited Expectimax

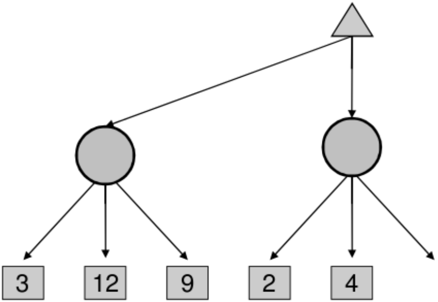


Digression: magnitudes matter

For expectimax, we need magnitudes to be meaningful



Expectimax-pruning

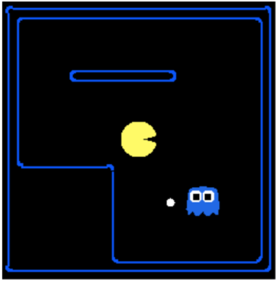


Expectimax for Pacman

- Ghosts are not anymore trying to minimize pacman's score
- Instead, they are now a part of the environment
- Pacman has a belief (distribution) over how they will act
- World assumptions have impact

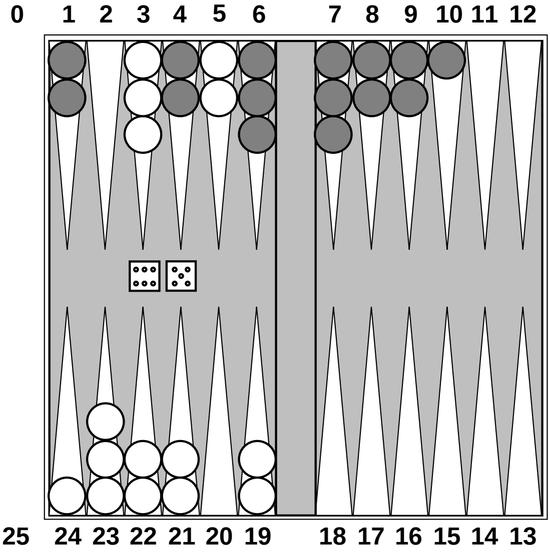
Results from playing 5 games

	Minimizing Ghost	Random Ghost
Minimax Pacman	Won 5/5 Avg. Score: 483	Won 5/5 Avg. Score: 493
Expectimax Pacman	Won 1/5 Avg. Score: -303	Won 5/5 Avg. Score: 503



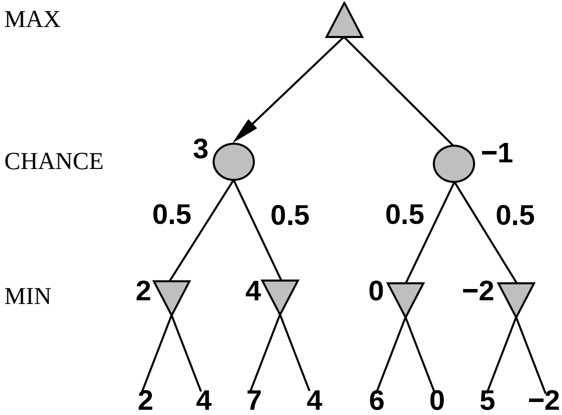
Pacman used depth 4 search with an eval function that avoids trouble
Ghost used depth 2 search with an eval function that seeks Pacman

Nondeterministic games: backgammon



Nondeterministic games in general

In nondeterministic games, chance introduced by dice, card-shuffling
Simplified example with coin-flipping:



Algorithm for nondeterministic games

Expectiminimax gives perfect play
Just like Minimax, except we must also handle chance nodes:

...
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*)
...

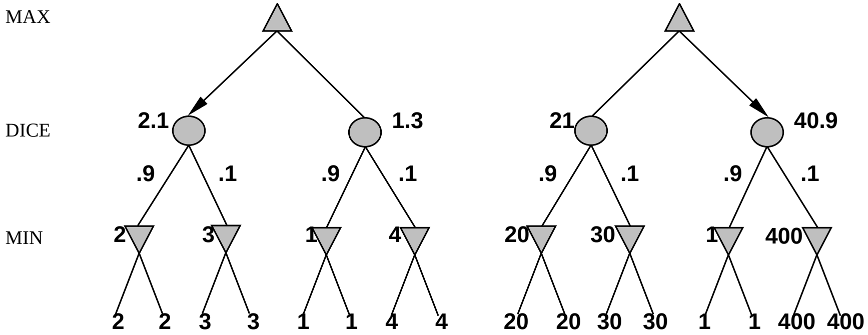
Nondeterministic games in practice

Dice rolls increase b: 21 possible rolls with 2 dice
Backgammon $\approx$ 20 legal moves

$$\text{depth } 4 = 20 \times (21 \times 20)^3 \approx 1.2 \times 10^9$$

- As depth increases, probability of reaching a given node shrinks
  $\Rightarrow$ value of lookahead is diminished
- α–β pruning is much less effective
- Temporal Difference Learning Gammon uses depth-2 search + very good Eval
  $\approx$ world-champion level

Digression: Exact values DO matter



Behaviour is preserved only by positive linear transformation of Eval
Hence Eval should be proportional to the expected payoff

Games of 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