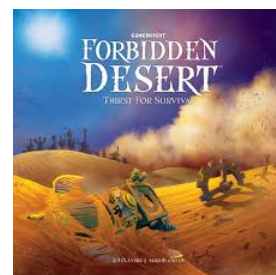
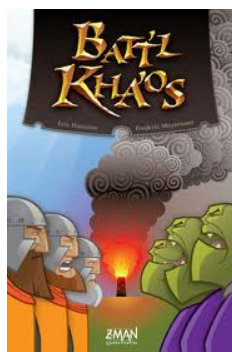


# ADVERSARIAL SEARCH

## Final Project Ideas!



Could do something very interesting using hidden Markov models!

## Today

- Reading
  - ▣ AIMA Chapter Read 5.1 -5.5, Skim 5.7
  
- Goals
  - ▣ alpha-beta pruning
  - ▣ Stochastic games
  - ▣ Real-time decision making

## Recap: Minimax

- Minimax performs DFS of search tree
  - ▣ Time  $O(b^m)$
  - ▣ Not feasible for common real-time games
  
- Alpha-beta pruning
  - ▣ Prunes parts of game tree that won't affect final result
  - ▣ Returns same result as minimax but faster

## Alpha-Beta pruning

- alpha is the best scenario MAX has found so far
    - ▣ MAX can always achieve a utility of alpha (and hopes for higher)
    - ▣ Only MAX modifies alpha
    - ▣ MAX uses beta for pruning
- $[\alpha, \beta]$
- beta is the best scenario MIN has found so far
    - ▣ MIN can always achieve a utility of beta (and hopes for lower)
    - ▣ Only MIN modifies beta
    - ▣ MIN uses alpha for pruning

## Alpha-Beta pruning

```

function ALPHA-BETA-SEARCH(state) returns an action
inputs: state, current state in game

v ← MAX-VALUE(state, -∞, +∞)
return the action in SUCCESSORS(state) with value 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 ← -∞
for a, s in SUCCESSORS(state) do
  v ← MAX(v, MIN-VALUE(s,  $\alpha$ ,  $\beta$ ))
  if v ≥  $\beta$  then return v
   $\alpha$  ← MAX( $\alpha$ , v)
return v

```

---

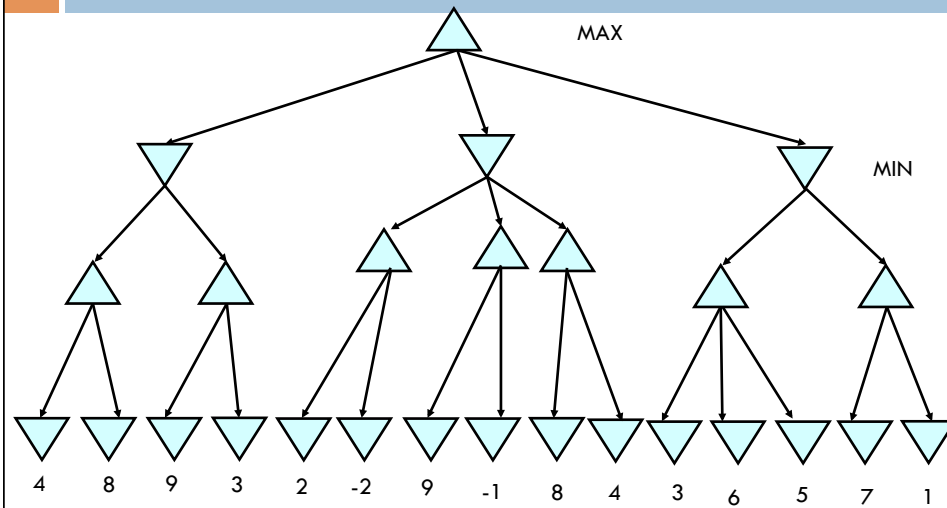
```

function MIN-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 ← +∞
for a, s in SUCCESSORS(state) do
  v ← MIN(v, MAX-VALUE(s,  $\alpha$ ,  $\beta$ ))
  if v ≤  $\alpha$  then return v
   $\beta$  ← MIN( $\beta$ , v)
return v

```

## Alpha-Beta pruning example



## Effectiveness of Alpha-Beta pruning

- Highly-dependent on the order in which the states are examined
- Try to examine those states that are likely to be best
- In practice, the running time for alpha-beta pruning is  $O(b^{m/2})$  as opposed to  $O(b^m)$ 
  - ▣ Effective branching factor is square root of  $b$
  - ▣ Can search twice as deep as minimax

## Stochastic Games

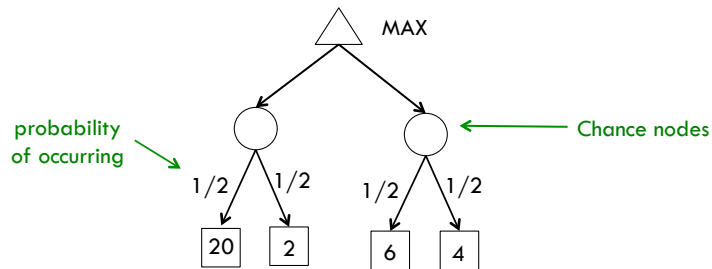
### Stochastic games

- Stochastic games include an element of chance
  - ▣ e.g. dice, unpredictable or random opponents
- Example 1-player and 2-player stochastic games
  - ▣ solitaire, minesweeper, backgammon, pacman

How do we find the optimal strategy in the presence of uncertainty?

## Stochastic games

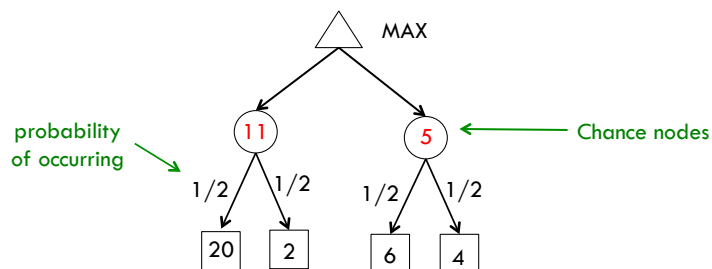
- How do we find the optimal strategy in this case?
  - ▣ Change minimax tree to include chance nodes



- Compute average (expected) utility
  - ▣ e.g.  $\frac{1}{2}(20) + \frac{1}{2}(2) = 11$

## Stochastic games

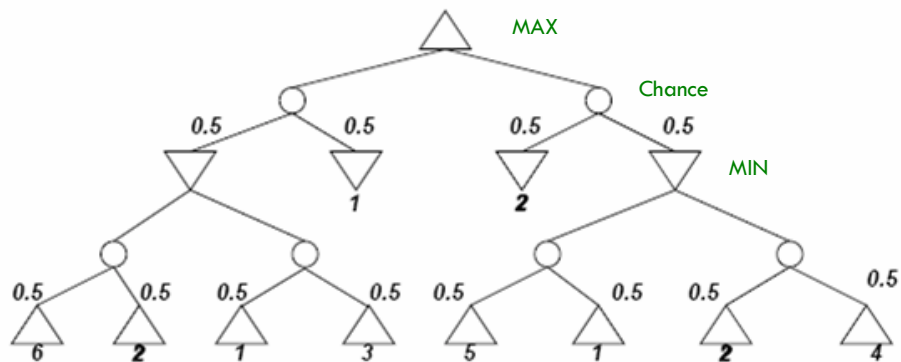
- How do we find the optimal strategy in this case?
  - ▣ Update minimax tree to include chance nodes



- Compute average (expected) utility
  - ▣ e.g.  $\frac{1}{2}(20) + \frac{1}{2}(2) = 11$

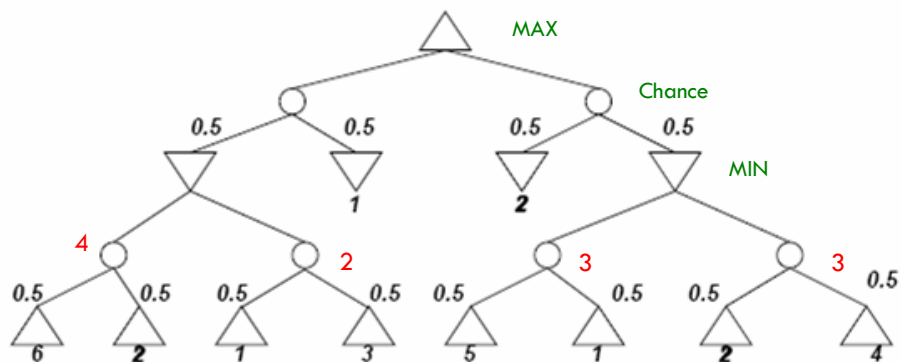
## Stochastic games

- For a 2-person stochastic game: MAX, MIN, chance



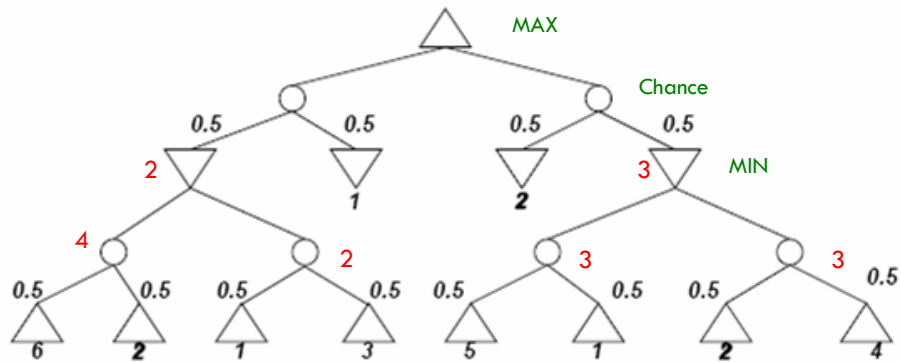
## Stochastic games

- For a 2-person stochastic game: MAX, MIN, chance



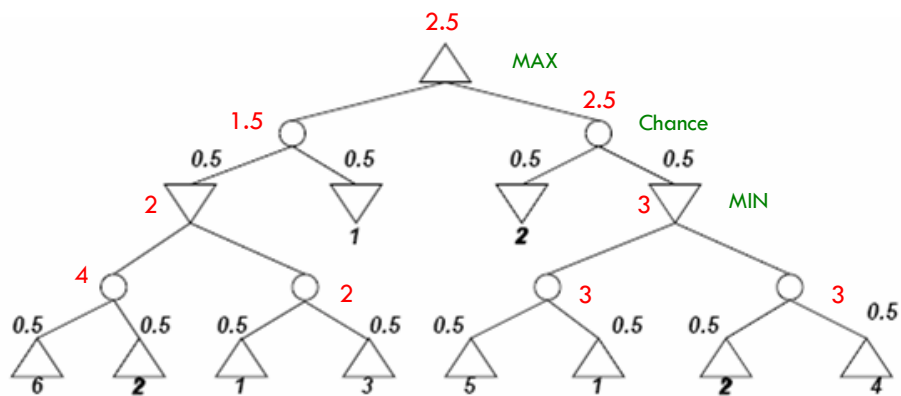
## Stochastic games

- For a 2-person stochastic game: MAX, MIN, chance



## Stochastic games

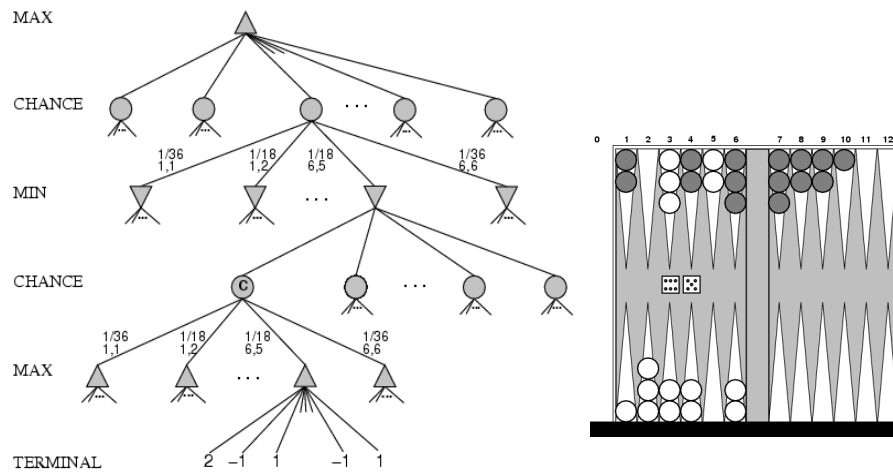
- For a 2-person stochastic game: MAX, MIN, chance





## Example: Backgammon

- White rolls 6-5: (5-10, 5-11), (5-11,19-24), ...



## Expectiminimax

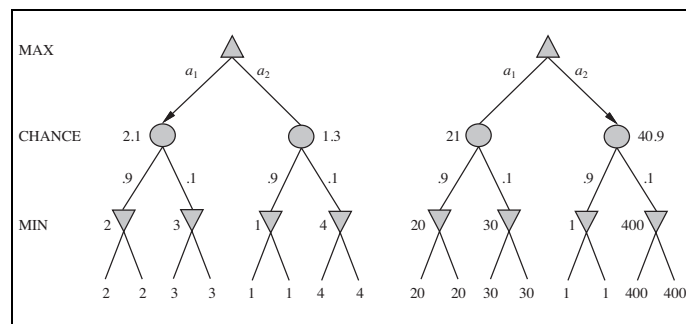
- Update minimax strategy to compute weighted average (expected value) at chance nodes
- Gives the expected value of a position/move

$$\text{EXPECTIMINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if } \text{TERMINAL-TEST}(s) \\ \max_a \text{EXPECTIMINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{PLAYER}(s) = \text{MAX} \\ \min_a \text{EXPECTIMINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{PLAYER}(s) = \text{MIN} \\ \sum_r P(r) \cdot \text{EXPECTIMINIMAX}(\text{RESULT}(s, r)) & \text{if } \text{PLAYER}(s) = \text{CHANCE} \end{cases}$$

- $O(b^m r^m)$  where  $r$  is number of distinct chance outcomes

## ExpectiMinimax in real-time

- Recall, real-time games need to make fast decisions
- For minimax, scale of the evaluation function doesn't matter
- For expectiMinimax, the scale is important



## Real-Time Decision Making

## Real-time decision making

- Alpha-beta pruning still has to search down to the leaf nodes (for part of the search tree)
- Standard approach (Shannon, 1949):
  - ▣ Apply a cutoff test (turn non-leaf nodes into leaves)
  - ▣ Replace utility function by an evaluation function that estimates “desirability” of position



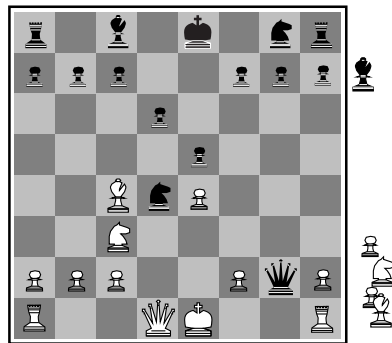
Claude Shannon

## Real-time decision making

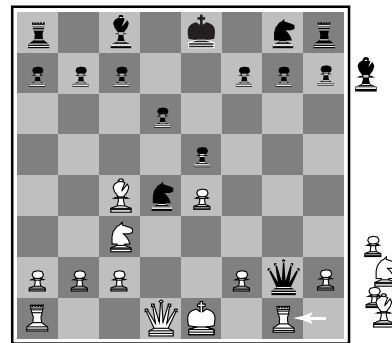
$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if } \text{TERMINAL-TEST}(s) \\ \max_a \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{PLAYER}(s) = \text{MAX} \\ \min_a \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{PLAYER}(s) = \text{MIN} \end{cases}$$

$$\text{H-MINIMAX}(s, d) = \begin{cases} \text{EVAL}(s) & \text{if } \text{CUTOFF-TEST}(s, d) \\ \max_a \text{H-MINIMAX}(\text{RESULT}(s, a), d + 1) & \text{if } \text{PLAYER}(s) = \text{MAX} \\ \min_a \text{H-MINIMAX}(\text{RESULT}(s, a), d + 1) & \text{if } \text{PLAYER}(s) = \text{MIN} \end{cases}$$

## How to construct an evaluation function



(a) White to move



(b) White to move

## How to construct an evaluation function

- Estimates utility of game from truncated position
  - ▣ Order *terminal* states in same manner
  - ▣ Fast to compute
  - ▣ For non-terminal states, correlated with the truth
- Weighted linear combination of features
  - ▣ independence assumption

$$\text{EVAL}(s) = w_1 f_1(s) + \dots + w_n f_n(s) = \sum_{i=1}^n w_i f_i(s)$$

$$\text{EVAL}(s) = 200(K-K') + 9(Q-Q') + \dots + 0.5(D-D' + I-I') + 0.1(M-M')$$

## Cutoff tests – when to stop?

- At a fixed depth
- Iterative deepening
  - ▣ Report the result of the last IDS search that was fully completed
- Horizon effect
  - ▣ Pushing off the inevitable
- Quiescence search
  - ▣ Stop at quiescent (quiet) positions
  - ▣ Focus on non-quiescent positions

## Summary of adversarial games

- Talked mostly about zero-sum games
  - ▣ If I win then you necessarily lose
- Minimax is optimal strategy but often too slow
  - ▣ Alpha-beta pruning can increase max depth by factor of 2
  - ▣ Implement evaluation function and cutoff-tests for early stop and guided search
- Expectiminimax used for games with an element of chance/randomness