Uncertainty and Randomness (Expectimax)

- Good for explicit randomness, unpredictable opponents, or actions that can fail
- Values should now reflect average-case (expectimax) outcomes, not worst-case (minimax) outcomes
- Expectimax ≜ compute the average scores under optimal play
  - Max nodes as in minimax search
  - Chance nodes are like min nodes, but the outcome is uncertain
  - Calculate their expected utilities ≜ weighted average (expectation) of children

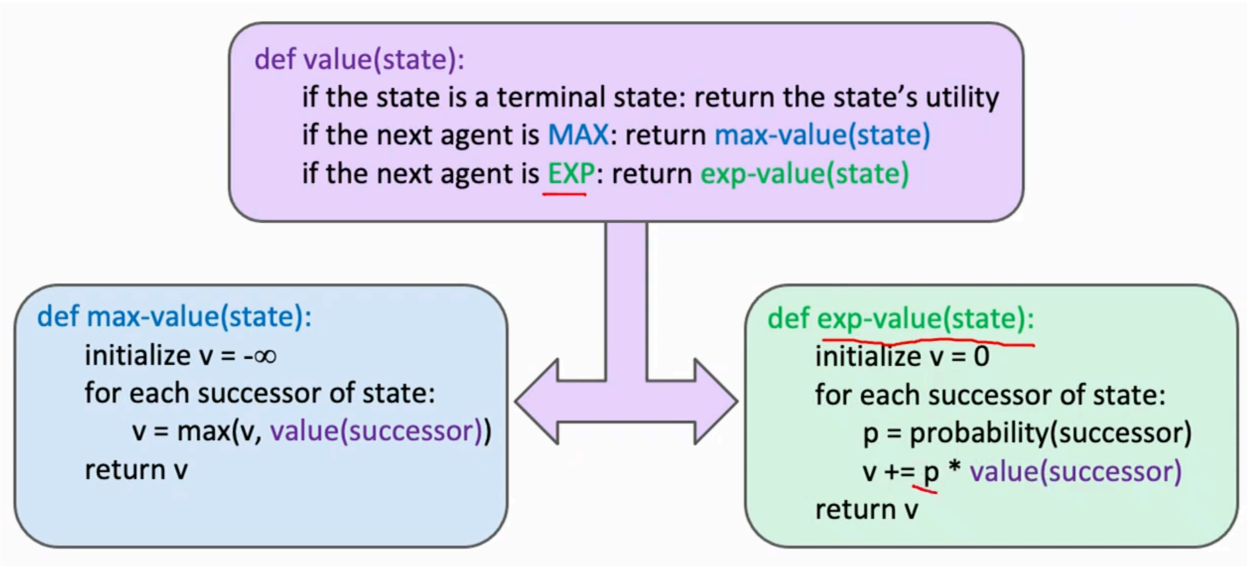

Figure 1: Screenshot_2023-09-20_at_6.51.31_PM.png

-
- Cannot prune
- Depth-limitation works
- Expected value of a function of a random variable ≜ the average weighted by the probability distribution over outcomes
- Can be challenging to get the optimal weights for an expectimax model

Mixed Layer Types

- Expectiminimax
  - Environment is an extra 'random agent' player that moves after each min/max agent
  - Each node computes the appropriate combination of its children

Multi-Agent Utilities

- Minimax
  - Have layers for each player
  - Have to propagate each value for each player up the tree
  - Terminals have utility tuples
  - Each player maximizes its own value

Difficulties with Search

- Even with alpha-beta pruning and limited depth, large b is an issue ($O(b^{\frac{m}{2}})$)
- Limiting depth requires us to design good evaluation functions
  - Not a general-purpose method: need to design new evaluation function for each new problem
  - Bad evaluation function may lead to inefficient or biased solutions
  - The trend in AI is to prefer general method and less human tweaking

Monte Carlo Tree Search (MCTS)

- Evaluation by rollouts: estimate value of a state by playing many games from this state by taking random actions (or some other fast policy) and count wins and losses
  - Allocate more rollouts to promising or uncertain nodes
- Selective search: explore parts of the tree that will help improve the decision at the root, regardless of depth

Upper Confidence Bound (UCB) Heuristic

$$\text{UCB1(n)} = \frac{\text{U(n)}}{\text{N(n)}} + \text{C} \times \sqrt{\frac{\text{log(n)(parent(n))}}{\text{N(n)}}} \Bigg|$$

$C \mathrel{\widehat{=}}$ parameter we chose to trade off between two terms
$N(n) \mathrel{\widehat{=}}$ number of rollouts from node n
$U(n) \mathrel{\widehat{=}}$ total utility of rollouts for player of parent(n)

- The total utility of rollouts $\triangleq$ the number of wins - Have to keep track of both N and U for each node - Combines ‘promising’ and ‘uncertain’

Monte Carlo Tree Search (MCTS) Algorithm

- Repeat until end of time:
  1. Selection: recursively apply UCB to choose a path down and leaf node n
  2. Expansion: add a new child c to n
  3. Simulation: run a rollout from c
  4. Back-propagation: update U and N counts from c back up to the root
- Choose the action leading to the child with highest N
- Most common tool for solving hard search problems
- Time complexity independent of b and m
- No need to design evaluation functions (general-purpose and easy to use)
- Solution quality depends on number of rollouts N
  - Theorem: as $N \to \infty$, MCTS selects the minimax move
- Maps well to parallel hardware