Non-Deterministic Search

- Maze-like problem
  - Agent lives in a grid
  - Walls block the agents' path
- Noisy movement: actions do not always go as planned
- Agent receives rewards at each time step
  - Small 'living' reward
  - Big reward at end (good or bad)
- Goal: maximize sum of rewards

Markov Decision Process (MDP)

- No one fixed state
- Goal $\triangleq$ find policy that tells you the best action for any state that you might find yourself in
- Set of states $s \in S$
- Set of actions $a \in A$
- Transition function $T(s, a, s')$
  - Probability that a from s leads to s'
  - Also called the model or the dynamics
- Reward function $R(s, a, s')$
  - Sometimes just $R(s)$ or $R(s')$
- A start state
- Maybe a terminal state
- 'Markov' $\triangleq$ 'given the current state, the past and present are independent'
- Want a policy π^*: $S \to A$
  - Policy $\triangleq$ choice of action for each state
  - A policy π gives an action for each state
  - An optimal policy is one that maximizes expected utility if followed
  - An explicit policy defines a reflex agent
- Can project into expectimax-like search tree

Time Value Decisions

- Decides to take rewards early or later
- Have to weight them to take the rewards earlier or later
- Value $\longrightarrow$ value $\times$ γ $\longrightarrow$ value $\times$ γ^2 $\longrightarrow$ ...
- Sooner rewards probably do have higher utility than later rewards
- Help our alg converge

Infinite Utilities

- Problem: what if the game lasts forever? Do we get infinite rewards
- Finite Horizon (similar to depth-limited search)
  - Terminate episodes after a fixed T steps (eg: life)
  - Gives non-stationary policies (π depends on time left)
- Discounting: use $0 < \gamma < 1$
  - Smaller γ means smaller 'horizon' – shorter term focus
  - $$U([r_0, ..., r_\infty]) = \sum_{t=0}^{\infty} \gamma^t r_t \leq \frac{R_{\max}}{1-\gamma}$$
  - Absorbing state: guarantee that for every policy, a terminal state will eventually be reached
  - Utility $\triangleq$ sum of discounted rewards
- Expectimax evaluation

- Problem: states are repeated
- Solution: only compute needed quantities once, cache the rest in a lookup table
- Idea: do a depth-limited computation, but with increasing depths until change is small (deep parts of the tree eventually don't matter if $\gamma < 1$)

- Optimal quantities
  - The value (utility) of a state s: $V^*(s) \triangleq$ expected utility starting in s and acting optimally
  - The value (utility) of a q-state (s, a): $Q^*(s, a) \triangleq$ the expected utility starting out having taken action from state s and (thereafter) acting optimally
  - The optimal policy $\pi^*(s) \triangleq$ optimal action from state s
  - Recursive definition of value (similar to expectimax):
    - $$V^*(s) = \max_a (Q^*(s, a))$$
    - $$Q^*(s, a) = \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V^*(s')]$$

    ### Bellman Equation

$$V^*(s) = \max_a \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V_k(s')]$$

- Solves how to get optimal values in terms of optimal values of other states - $T(s, a, s') \hat{=} P(s' \mid s, a)$ - $O(S^2 A)$ - Key idea: time-limited values - Compute time limited values from bottom up (calculate and cache all V_0) then apply bellman equation to get $V_{k+1}(s)$ and repeat until convergence which yields V^* - Define $V_k(s)$ to be the optimal value of s if the game ends in k more time steps $\iff$ what a depth k expectimax would yield