Alpha-Beta Implementation

α: MAX's best option on path to root
β: MIN's best option on path to root

```
def max-value(state, α, β):
    initialize v = -∞
for each successor of
    state:
    v = max(v, min-value(successor, α,β)) if v
        ≥ β
            return v
        α = max(α, v)
    return v
```

```
def min-value(state , α, β):
    initialize v = +∞
for each successor of
    state:
    v = min(v, max-value(successor, α, β)) if v
        ≤ α
            return v
        β = min(β, v)
    return v
```

Algorithm

Week 5.

→ Minimax + Utility with Depth Cutoff Pseudocode

```
def value(state, depth):
    if the state is a terminal state or depth > cutoff: return the state's
    utility value
    if the next agent is MAX: return max-value(state)
    if the next agent is MIN: return min-value(state)
```

```
def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor))
    return v
```

$$V(s) = \max_{s' \in \text{successors}(s)} V(s')$$

```
def min-value(state):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor))
    return v
```

$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$

→ Uncertain Outcome

- Average-Case ⟷ Expectimax
- Worst-Case ⟷ Minimax

↳ Expectimax Search

- Calculate expected utilities

Expectimax Pseudocode

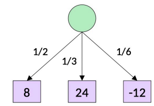
```
def value(state):
    if the state is a terminal state: return the state's utility
    if the next agent is MAX: return max-value(state)
    if the next agent is EXP: return exp-value(state)
```

```
def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor))
    return v
```

```
def exp-value(state):
    initialize v = 0
    for each successor of state:
        p = probability(successor)
        v += p * value(successor)
    return v
```

算 Probability

```
def exp-value(state):
    initialize v = 0
    for each successor of state:
        p = probability(successor)
        v += p * value(successor)
    return v
```



v = (1/2) (8) + (1/3) (24) + (1/6) (-12) = 10

Tips: no pruning

Instead → Depth-Limited Expectimax

$\boxed{\text{深度 limit}}$ → 有限时间做 optimal

1. depth d
2. Max node
3. Chance node
4. Depth-Limit

→ Heuristic evaluation function

→ Markov Decision Processes

○ **Dangers**
- Dangerous Optimism
- Dangerous Pessimism

○ **Utility** → functions from outcome to real number of <u>agent's preference</u>

○ Maximum expected utility:

A rational agent chose action → Maximizes its expected utility, given its knowledge

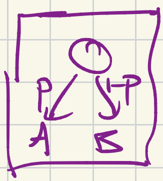
↳ Utilities: Uncertain Outcome

↳ Preferences

- Prizes, A, B . . .
- Lotteries, $L = [p, A; (1-p), B]$

○ **Notion**: **$A \succ B$ Preference**

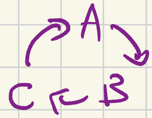
$A \sim B$ Indifferrence



○ Rational Preferences

↳ Constraint: **Axiom of Transitivity**:

intransitive preference



The Axioms of Rationality

Orderability

$(A \succ B) \vee (B \succ A) \vee (A \sim B)$

Transitivity

$(A \succ B) \wedge (B \succ C) \Rightarrow (A \succ C)$

Continuity

$A \succ B \succ C \Rightarrow \exists p\ [p, A;\ 1-p, C] \sim B$

Substitutability

$A \sim B \Rightarrow [p, A;\ 1-p, C] \sim [p, B; 1-p, C]$

Monotonicity

$A \succ B \Rightarrow$
$(p \geq q \Leftrightarrow [p, A;\ 1-p, B] \succeq [q, A;\ 1-q, B])$

Maximum Expected Utility
MEU Principle:

$U(A) \geq U(B) \Longleftrightarrow A \succ B$

$U([p_1, S_1 \ldots; p_n, S_n]) = \sum_i p_i U(S_i)$

→ Utility Scale

- Normalized Utilities: $U_+ = 1.0$ $\quad U_- = 0.0$
- behavior is invariant under positive linear transformation

  $U'(x) = k_1 U(x) + k_2$ $\quad$ where $k_1 > 0$

→ Human Utility

- Standard Lottery (eg. $p + (1-p) = 1$)

→ Money

- expected monetary value: $EMV(L) = p \times A + (1-p) \times$

$[\ \overset{\text{游戏}}{U(L)} < \overset{\text{金钱}}{U(EMV(L))}\ ]$

- risk-averse & risk prone

⇒ Non-Deterministic Search

- Markov Decision Processes
  - ↳ A set of states $s \in S$
  - ↳ A set of actions $a \in A$
  - ↳ Transition Function $T(s, a, s') \sim P(s'|s,a)$
    (also called the model or dynamics)
  - ↳ Reward function $R(s,a,s')$
  - ↳ Start state / terminal state

- MPS

  only care present state (action outcomes depends only the current state)

$$P(S_{t+1} = s' \mid S_t = s_t, A_t = a_t, S_{t-1} = s_{t-1}, A_{t-1}, \ldots S_0 = s_0)$$

$$= P(S_{t+1} = s' \mid S_t = s_t, A_t = a_t)$$

accumutate
||
Present

- Policies
  - ↳ plan, or Sequence of actions, from start to goal
  - ↳ For MDPs, want to optimal policy $\pi^* : S \to A$
    - one that maximizes Expected Unitity if followed
  - ↳ π : gives an action for each states
  - (Expectimax didn't compute entire policies)

- MDP Search Trees
  - Each MDP state ⟶ expectimax-like search tree

△ s

○ s, a   (s, a, s') called a transition

△ s, a, s' $T(s,a,s') = P(s' | s,a)$ $R(s,a,s')$

- Utilities of Sequence
  - More > less Utilities / first come U > latter come U

- Discounting
  - Value of rewards decay exponentially

$$1 \to \gamma \to \gamma^2 \ldots$$

$1 \left\{ \begin{array}{l} \Delta \\ \circ \\ \Delta \end{array} \right.$ → Sooner reward probably > later rewards
utility

$\gamma \left\{ \begin{array}{l} \Delta \\ \circ \\ \Delta \\ \vdots \end{array} \right.$ → help algorithms Converge

eg. $U([1,2,3]) = 1 \times 1 + 0.5 \times 2 + 0.25 \times 3$

$U([1,2,3]) < U([3,2,1])$

- Stationary Preferences

$$[a_1, a_2, \dots] \succ [b_1, b_2, \dots]$$
$$\Updownarrow$$
$$[r, a_1, a_2, \dots] \succ [r, b_1, b_2, \dots]$$

↳ Additive utility: $U([r_0, r_1, r_2 \dots]) = r_0 + r_1 + r_2 + \dots$

↳ Discounted Utility: $U([r_0, r_1, r_2 \dots]) = r_0 + \gamma r_1 + \gamma^2 r_2 + \dots$

- Infinite Utilities

game last forever → infinite reward ?

↳ Finite horizon (similar to depth limited search)

↳ Terminate episodes after a fixed T steps

↳ Gives nonstationary policies

↳ Discounting use $0 < \gamma < 1$ (smaller γ means smaller "horizon")

$$U([r_0 \dots r_\infty]) = \sum_{t=0}^{\infty} \gamma^t r_t \leq R_{max} / (1-\gamma) \quad \text{最大累积奖励}$$

- Optimal Quantities

↳ The Value (utility) of a state s:

$V^*(s)$ = expected utility starting in s and acting optimally

↳ The Value (utility) of a q-state (s,a)

$Q^*(s,a)$ = expected utility starting out having taken action a from state s and acting optimally

Optimal policy

$\pi^*(s)$ = optimal action from state S

- Values of States

Fundamental operation:

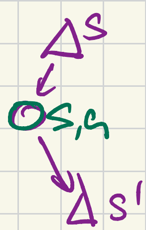
Expected utility under optimal action + Average Sum of reward (discount)

= Expectimax Caculation

- Recursive definition of value:

$V^*(s) = \max_a Q^*(s,a)$ eg. $\max(-10, 10)$

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



$$V^*(s) = \max_a \underbrace{\sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma V^*(s')\right]}_{Q^*}$$

- Time-Limited Value

通过深度为 K 的 expectimax 计算

$V_k(s)$ ⎫ k 为步数到多少为止

如果 k 步结束时,状态 s 的最优值 ⎭ S 是当前 state

- Value Iteration

↳ Bellman equation characterize the optimal value

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

↳ Value iteration computes:

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma V_k(s')\right]$$

↳ Start with $V_0(s) = 0$: → expected reward sum

↳ 给定 $V_k(s)$ 值,对每个状态执行一次 expectimax

↳ 重复直到 Converges → states 变化足够小,已经 converges optimal

↳ Complexity Iteration $O(S^2A)$ time $S \to A \to S$

Converge ⟹ unique optimal value

- Convergence
- 基础：1. 最大深度 M 的树
  2. 折扣因子小于 1
- V_k & V_{k+1} → 深度 k+1 的 expectimax 结果
  V_k：k 层之后, reward → 0
  V_{k+1}：k+1 层, reward → 实际奖励 (V_k)

→ Fixed Policies

- Fixed Policy's Expectimax tree
  $s \to \pi(s) \to s'$
  固定于某个特定 policy, 可 not optimal, 但 follow policy π

- Utility for a Fixed Policy
  Basic operation: Compute the Utility of a state s under a fix (non-optimal) policy

$$V^{\pi}(s) = \sum_{s'} T(s, \pi(s), s')\left[R(s, \pi(s), s') + \gamma V^{\pi}(s')\right]$$

$V^\pi(s) = \sum_{s'} T(s, \pi(s), s') [R(s, \pi(s), s' + \gamma V^\pi(s')]$

◦ Utilities for a fixed Policy

$$V^\pi(s) = \sum_{s', r} P(s', r | s, \pi(s)) [r + \gamma V^\pi(s')]$$

- $P(s', r \mid s, \pi(s))$ 是在状态 s 下，根据策略 π 采取行动 $\pi(s)$ 后，转移到状态 s' 并获得即时奖励 r 的概率。
- $\pi(s)$ 是策略 π 在状态 s 下推荐的行动。
- r 是即时奖励。
- γ 是折扣因子，用于减少未来奖励的当前价值。
- $V^\pi(s')$ 是遵循策略 π 时状态 s' 的期望效用。

Utility

◦ Policy Evaluation

How to calculate V's for fixed policy π?

→ 1. recursive Bellman equations into updates (like iteration)

$V_0^\pi(s) = 0$

$\Sigma(P \quad + \quad R)$

$$V_{k+1}^\pi(s) \leftarrow \sum_{s'} T(s, \pi(s), s') [R(s, \pi(s), s') + \gamma V_k^\pi(s')]$$

→ Q

- $V_{k+1}^\pi(s)$：这是在策略 π 下，状态 s 在 $k+1$ 次迭代后的价值。它代表从状态 s 出发，遵循策略 π 所期望获得的总折扣奖励的估计。
- $T(s, \pi(s), s')$：这是在状态 s 下，按策略 π 执行行动 $\pi(s)$ 后转移到状态 s' 的概率。
- $R(s, \pi(s), s')$：这是执行行动 $\pi(s)$ 从状态 s 转移到状态 s' 时获得的即时奖励。
- γ：这是折扣因子，它决定了未来奖励相对于即时奖励的重要性。
- $V_k^\pi(s')$：这是在策略 π 下，状态 s' 在 k 次迭代后的价值。

公式中的求和操作将每个可能的下一个状态 S'的概率乘以在该状态下可能获得的奖励，然后将所有这些乘积累加起来。这样做是为了计算当前状态s 在遵循固定策略 π 下的期望总折扣奖励。

→ 2. linear system

Matrix

→ Policy Extraction

- Computing Actions from Values

  每个行动的期望回报：

动作函数

$$Q(s,a) = \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma U^*(s')\right]$$

  最佳行动：

$$\pi^*(s) = \underbrace{\arg\max_a}_{\text{最大}} \underbrace{Q(s,a)}_{s \text{ state do } a\text{，所期望得到的总回报}}$$

- Computing Actions from Q-Value

$$\pi^*(s) = \arg\max Q^*(s,a)$$

→ Policy Iteration

- Problem with Value Iteration
  - → slow – $O(S^2A)$ per iteration
  - → The "max" at each state rarely changes
  - → Policy converges before the value

○ Approach for Optimal Value

○ Policy Evaluation: Calculate utilities for fixed policy until Converge

$$V_{k+1}^{\pi_i}(s) \leftarrow \sum_{s'} T(s, \pi_i(s), s') \left[R(s, \pi_i(s), s') + \gamma V_k^{\pi_i}(s') \right]$$

+

○ Improvement:

update policy using one step look-ahead with resulting converged utilities as future value

Repeat steps until policy converges

$$\pi_{i+1}(s) = \arg\max \sum_{s'} T(s, a, s') \left[R(s, a, s') + \gamma V^{\pi_i}(s') \right]$$

→ optimal, Converge faster

→ Comparsion (Value & policy)

Value
- ○ Every Value iteration updates both value and (implicitly) policy
- ○ Don't track policy, but take max over actions

Policy
- ○ Policy iteration
  - evaluation – do several time, Everytime do one action
  - Improvement – chose new policy