Assignment: Tic-Tac-Toe with Value Iteration

Veljko Radojičić Nikolina Živanović

Objective

The goal of this exercise is to implement a **value iteration** algorithm for the game of Tic-Tac-Toe, allowing an agent to learn an **optimal policy** against different types of opponents. It is required to:

- Construct a value function $V(s)$ for all game states.
- Derive an optimal policy from the learned values.
- Train the agent against a **random opponent** and a **fixed heuristic opponent**.
- Consider **both playing roles**: agent goes first (X) and agent goes second (O).

Rules

Tic-Tac-Toe is a two-player, turn-based game with the following rules:

- Board size: 3x3
- Players: X and O
- Goal: get three symbols in a row, column, or diagonal
- Game ends in a win for X, a win for O, or a draw

Assignment Requirements

1. Value Iteration Algorithm

Implement value iteration to compute $V(s)$ for all legal board states:

1. Initialize $V(s)$ for all states
2. Iteratively update $V(s)$ using the Bellman equation:

$$V(s) \leftarrow \max_a \sum_{s^+} P(s^+ \mid s, a) \left[R(s^+) + \gamma V(s^+) \right]$$

3. Continue iterations until convergence: $|V_{new}(s) - V(s)| < \delta$
4. Derive optimal policy $\pi(s)$ from the converged values:

$$\pi(s) = \arg\max_a \sum_{s^+} P(s^+|s,a)\big[R(s^+) + \gamma V(s^+)\big]$$

2. Opponents

Train and evaluate the agent against two types of opponents:

1. **Random Policy:** opponent chooses any available move with uniform probability
2. **Fixed (Heuristic) Policy:** a simple deterministic strategy. This is an example of defined priority of moves. It is allowed for a fixed policy to be significantly simpler than this.
   (a) Win immediately if you have a winning move
   (b) Block opponent's immediate winning move
   (c) Create a fork if possible
   (d) Block opponent's fork
   (e) Take center if available
   (f) Take opposite corner if opponent took a corner
   (g) Take any free corner
   (h) Take any free side (non-middle or non-corner square)

   Another suggestion for a fixed policy: MiniMax policy

3. Playing Roles

The agent must consider both scenarios:

1. **Agent plays first (X)**: agent moves first, opponent follows
2. **Agent plays second (O)**: opponent moves first, agent follows

4. Simulation and Evaluation

- Simulate multiple (say 1000) games to verify policy performance:
  - Against random policy
  - Against fixed policy
- Record results for both first-player and second-player scenarios
- There is no need for any kind of visualisation.

States and Transitions

- **State** $s :=$ current board configuration (9-character string, e.g., "X.O..XO..")
- **Action** $a :=$ placing X or O in an empty cell. Notice: *opponent is part of an environment*
- **Transition** $s \to s^+ :=$ the resulting state *after the agent moves and the opponent responds.*
- **Reward**

$$R(s^+) := \begin{cases} +1 & \text{if agent wins} \\ -1 & \text{if agent loses} \\ 0 & \text{if draw or non-terminal state} \end{cases}$$

- **Probability** $P(s^+ \mid s, a)$:
  - Deterministic opponent $\Rightarrow$ probability $= 1$
  - Random opponent $\Rightarrow$ uniform probability over all legal opponent moves
- **Reward**: deterministic, depends only on the next state s^+.

Example: Agent plays first

$$\underbrace{\begin{bmatrix} X & O & . \\ . & . & . \\ . & . & . \end{bmatrix}}_{\text{observed state } s_0} \xrightarrow{\text{action } a} \underbrace{\begin{bmatrix} X & O & X \\ . & . & . \\ . & . & . \end{bmatrix}}_{\text{"intermediate" state } s_1} \xrightarrow{\text{opponent response}} \underbrace{\begin{bmatrix} X & O & X \\ . & O & . \\ . & . & . \end{bmatrix}}_{\text{next observed state } s_2}$$

Generally speaking, $V(s)$ is updated using:

$$V(s_0) \leftarrow \max_a \sum_{s_2} P(s_2 \mid s_0, a)\big[R(s_2) + \gamma V(s_2)\big]$$

Note: If agent's move immediately ends the game, the board is already terminal, so that specific term in the sum reduces to $R(s_1)$. Another way to treat this issue is through *afterstates*. More on that you can find in `https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf`, page 160.

Why we use $P(s^+ \mid s, a)$ instead of $P(s^+, r \mid s, a)$

- General MDP formulation: $P(s^+, r \mid s, a)$ accounts for stochastic rewards
- In Tic-Tac-Toe, **reward is deterministic** and fully determined by s^+
- Therefore, we can write:

$$V(s) \leftarrow \max_a \sum_{s^+} P(s^+ \mid s, a)\big[R(s^+) + \gamma V(s^+)\big]$$

- This is simpler and equivalent, no need to include r in the transition probability

Why we write $R(s^+)$ instead of $R(s, a)$

In Tic-Tac-Toe, the reward is fully determined by the resulting state s^+: wins, losses, and draws can be detected by examining the board after the transition.

If the opponent's policy is deterministic, then the transition $s, a \to s^+$ is also deterministic. In that case, the reward may equivalently be viewed as a deterministic function of the pair (s, a), since (s, a) uniquely determines s^+ and thus the corresponding reward.

Conversely, if the opponent policy is stochastic, the same pair (s, a) can lead to multiple different next states s^+ with different probabilities. Since the reward depends on the outcome, and these outcomes differ across possible next states, the reward cannot be determined uniquely from (s, a) alone. Instead, it is defined jointly with the transition to s^+, which is why the correct representation here is $R = R(s^+)$ or, more generally, $P(s^+, r \mid s, a)$.

Implementation Tips

- Represent board states as 9-character strings: "." = empty, "X" = agent, "O" = opponent
- Use functions for: checking winner, listing legal moves, applying moves, opponent policies, checking whether a state is terminal

Evaluation Criteria

- Correct implementation of value iteration
- Consideration of both agent roles (first and second)
- Learning against both random and fixed opponents
- Evidence of policy performance through simulations