Value Iteration Agent for FrozenLake-v1

This repository implements a well-known Dynamic Programming Algorithm **Value Iteration-based Reinforcement Learning Agent** for solving the FrozenLake-v1 environment from OpenAI Gym. Dynamic Programming is a mathematical optimization and problem-solving approach. In the context of **Reinforcement Learning (RL)**, DP provides a structured and computationally feasible method for solving **Markov Decision Processes (MDPs)** by leveraging **Bellman equations**.

The agent aims to learn an optimal policy by iteratively estimating the state-value function using Bellman's Optimality Principle. This approach balances exploration and exploitation in a stochastic, discrete environment.

Key Features and Design Choices

1. Agent Design:

The agent uses **three main data structures** implemented as Python's defaultdict:

- **Reward Table (rewards):**
  - Stores the reward associated with (state, action, next_state) transitions.
  - defaultdict(float) ensures that unvisited transitions are initialized with a default reward of 0.0.
- **Transition Probability Table (transits):**
  - Tracks how frequently each (state, action) pair transitions to a given next state, forming the empirical transition probability distribution.
  - Implemented as a defaultdict(collections.Counter) to efficiently count state transitions without requiring pre-definition of states or actions.
- **State-Value Table (state_values):**
  - Stores the estimated value of each state (expected cumulative reward starting from the state).
  - defaultdict(float) initializes unvisited states with a value of 0.0, simplifying computation and avoiding explicit initialization.

Why defaultdict?

- Unlike a standard dictionary, defaultdict avoids KeyError when accessing keys that do not yet exist. This is particularly useful in reinforcement learning, where states, actions, and transitions may not be fully known a priori.

- It simplifies code by implicitly initializing values for previously unseen states, actions, or transitions, making the implementation cleaner and more robust.

2. Value Iteration Algorithm:

Value Iteration is a **dynamic programming** approach to finding the optimal policy. The agent uses Bellman's Optimality Equation:

$$V(s) = \max_a \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma V(s')\right]$$

Here:

- V(s): State-value function.
- R(s,a,s'): Reward for transitioning from state s to s' using action a.
- Γ(gamma): Discount factor, controlling the trade-off between immediate and future rewards.
- P(s'|s,a): Empirical transition probability, estimated from the data.

  in dynamic programming (DP) algorithms, it is essential to have access to the model dynamics of the environment, which include the transition probabilities (P(s'|s,a)) and reward function. However, in many real-world scenarios, these dynamics are not provided explicitly. To address this, the agent performs random exploration by playing n random steps, which helps gather enough data to estimate the dynamics of the Markov Decision Process (MDP) empirically. This involves counting how often each transition occurs for a given (s,a) pair and normalizing these counts to approximate the transition probabilities.

The agent updates its state values iteratively by:

1. Calculating the expected value for each possible action in a state.
2. Selecting the action with the highest value to update V(s).

3. Policy Execution:

- The agent selects actions using the **greedy policy** derived from the updated state values.
- For a given state, it computes the expected value for all possible actions and picks the one with the highest value.

4. Learning Workflow:

1. **Random Exploration:**
   - The agent performs random actions for a fixed number of steps to populate the transition and reward tables.
   - This ensures sufficient exploration of the environment, especially during the early stages.
2. **Value Iteration:**
   - Updates state-value estimates using Bellman's equation until they converge.
3. **Policy Testing:**
   - Evaluates the learned policy by running multiple test episodes and calculating the average reward.

5. FrozenLake-v1 Environment:

- **Description:** FrozenLake-v1 is a discrete-state environment where the agent must navigate a grid to reach a goal while avoiding holes. The environment is stochastic, meaning actions do not always result in deterministic transitions.
- **State Space:** Finite, discrete set of grid positions.
- **Action Space:** Four discrete actions (left, right, up, down).
- **Challenge:** Balancing exploration and exploitation in a stochastic environment to learn an optimal policy.

6. Performance Visualization:

- The agent's learning performance is logged using **TensorBoard**.
- Rewards are visualized over iterations to track the agent’s progress toward solving the environment.

Results

- The agent solves the environment when the average reward over 20 test episodes exceeds **0.95**.
- State-value estimates and learned policies align with the optimal solution for the environment.