Interactive Reinforcement Learning Project

1. Introduction and Objective

This project presents a **fully interactive Reinforcement Learning (RL) learning tool** implemented entirely inside a Jupyter Notebook using **standard Python and `ipywidgets`**.
The objective is to provide an **intuitive, visual, and hands-on environment** for learning core RL concepts through experimentation rather than theory alone.

The tool is designed to:

- Demonstrate **tabular RL algorithms** clearly and transparently.
- Allow **real-time hyperparameter tuning**.
- Visualize **value functions, learning curves, and agent behavior**.
- Behave like a **lightweight web application** without requiring external frameworks or additional Python files.

This makes the notebook a **self-contained interactive RL lab**, suitable for students and beginners in reinforcement learning.

---

2. Implemented Environments

All environments are **finite, tabular Markov Decision Processes (MDPs)** and expose a **Gym-like API**:

- `reset()`
- `step(action)`
- `n_states`, `n_actions`
- Transition model `P` (for DP methods)

2.1 GridWorld 4×4

- Deterministic 4×4 grid.
- Start at the top-left cell, goal at the bottom-right cell.
- Reward:
    - −1 per step
    - 0 upon reaching the goal (terminal).
- Fully observable transition model.
- Ideal for **Dynamic Programming** and policy/value visualization.

2.2 FrozenLake 4×4 (Deterministic)

- Tiles: Start (S), Frozen (F), Hole (H), Goal (G).

- Rewards:
    - −1 when falling into a hole (terminal),
    - +1 when reaching the goal (terminal),
    - 0 otherwise.
- More complex than GridWorld due to mixed terminal rewards.
- Supports both **DP and model-free algorithms**.

2.3 Breakout (Custom Line Environment)

- One-dimensional line of discrete states.
- Agent starts in the center.
- Leftmost state: **hole** (lose, −1 reward).
- Rightmost state: **brick/goal** (win, +1 reward).
- Actions: left, stay, right.
- Small negative step cost to encourage faster solutions.
- Fully tabular with known transition model.

2.4 Gym4ReaL (Custom Line Environment)

- Similar 1D structure to Breakout.
- No step penalty.
- Rewards:
    - +1 at goal,
    - −1 at hole,
    - 0 otherwise.
- Designed as a minimal custom environment for algorithm comparison.

These environments are selectable via the **Environment dropdown** in the UI.

---

3. Implemented Algorithms

All algorithms are implemented in **simple, commented, tabular form**, making the learning process transparent.

3.1 Dynamic Programming (Model-Based)

- **Policy Evaluation**
  Iteratively computes the value function Vπ(s)V^\pi(s)Vπ(s) for a fixed policy using the Bellman expectation equation.

- **Policy Iteration**
  Alternates between:

1. Policy evaluation

2. Greedy policy improvement
   until the policy converges to the optimal policy.

- **Value Iteration**
  Uses the Bellman optimality equation to directly approximate the optimal value function, then extracts a greedy policy.

3.2 Model-Free Methods

- **Monte Carlo Control (Every-Visit)**
  Learns action-value estimates from complete episodes using ε-greedy exploration.

- **TD(0) Prediction**
  One-step Temporal Difference method for estimating state values under a fixed policy.

- **n-step TD Prediction**
  Generalization of TD(0) using n-step returns with a finite horizon (`max_steps`) to avoid numerical instability.

- **SARSA (On-Policy Control)**
  Updates action values using the next action selected by the same ε-greedy policy.

- **Q-learning (Off-Policy Control)**
  Learns an approximation of the optimal action-value function using a greedy target.

All algorithms work across **all four environments** due to the unified interface.

---

4. Interactive User Interface and Hyperparameters

The notebook includes a **web-style interactive panel** built using `ipywidgets`.

UI Components

- **Environment selector**
  GridWorld, FrozenLake, Breakout, Gym4ReaL.

- **Algorithm selector**
  Policy Evaluation, Policy Iteration, Value Iteration, MC, TD(0), n-step TD, SARSA, Q-learning.

- **Hyperparameter sliders**:

    - Discount factor γ (gamma),

    - Learning rate α (alpha),

    - Exploration rate ε (epsilon),

    - Number of episodes,

    - Maximum steps per episode.

Clicking **“Run training”** executes the selected algorithm with the chosen parameters and immediately updates the visual output.

---

5. Visualization and Analysis Tools

To enhance intuition and understanding, several visualization techniques are provided:

5.1 Value Function and Policy Visualization

- Heatmaps of V(s)V(s)V(s) for grid environments.
- Greedy policy arrows overlaid on the grid.

5.2 Learning Curves

- Episode return vs. episode index.
- Used for MC, TD, SARSA, and Q-learning.
- Clearly shows convergence behavior and parameter sensitivity.

5.3 Episode Traces

- Step-by-step greedy episode rollout:
    - State
    - Action
    - Reward
- Helps interpret the learned policy's behavior.

---

6. How to Use the Tool

1. Install required libraries (`numpy`, `matplotlib`, `ipywidgets`).
2. Run all notebook cells from top to bottom.
3. Navigate to the **Interactive UI** section.
4. Select:
    - An environment,
    - An algorithm,
    - Desired hyperparameters.
5. Click **Run training**.
6. Analyze plots and episode traces generated below the UI.

No external configuration or additional files are required.

---

7. Deliverables

- A **single Jupyter Notebook** containing:
    - Environment definitions,

    - Algorithm implementations,
    - Interactive UI,
    - Visualizations,
    - This final report.
- The notebook is pushed to **GitHub** as the sole project artifact.

---

8. Conclusion

This project successfully delivers a **self-contained, interactive RL learning platform** that fulfills all requirements:

- Multiple environments (including custom ones),
- Full coverage of core RL algorithms,
- Adjustable hyperparameters,
- Clear visualizations,
- Web-like interactivity within a notebook.

The tool is well-suited for **education, experimentation, and demonstration** of reinforcement learning fundamentals.