Interactive RL Learning Web Tool - Project Report

1. Executive Summary

This project provides a comprehensive, web interactive application to explore the differences between the theoretical concepts of Reinforcement Learning (RL) and their practical applications. The tool was constructed with the help of Streamlit as the frontend and Gymnasium as the environment-simulation software, providing the user with a broad set of algorithms, including basic variants of Dynamic Programming and n-step Temporal Difference control techniques.

The application consists of three distinctive modules: Foundation (DP and Prediction), Control (Q-Learning and SARSA), and a Custom Environment (Battery GridWorld). It has actual-time optimisation of hyperparameters, visualisation of value functions through a heatmap, and a custom-written discrete environment that requires agents to optimise a limited set of resources (battery) on a grid.

2. Implemented Environments

The tool combines common Gymnasium settings, as well as a custom-made setting, to conduct experimentation on the various types of RL algorithms.

2.1 Standard Environments

- **GridWorld Variants:** There are used Frozen Lake-v1 and Cliffwalking-v0 are used as examples of discrete state space algorithms.
- **Continuous Environments:** CartPole-v1 and MountainCar-v0.
    - *Implementation Note:* Q-Learning and SARSA implementations in this case are based on tabular mechanisms and, therefore, a custom Discretizer wrapper (located in utils.py) is utilized on these continuous environments. This privateer maps a continuous valued observation (e.g., Cart Position, Pole Angle) to a discrete bin space, creating a size-easy state space (ex, a 104 state space) upon which tabular updates may be applied.

2.2 Custom Environment: Battery GridWorld

My custom environment, based on BatteryEnv, was created as custom_env.py, where I replaced the generic requirement, Gym4Real, with BatteryEnv.

- **Objective:** The agent has to move using a 5x5 grid to a goal at location (4, 4).
- **Dynamics:**
    - **State Space:** A Tuple (x, y, battery_level), where battery ranges from 0 to 15.
    - **Action Space:** Discrete (Up, Down, Left, Right).

    - **Reward Function:** +50 at the goal, -1 per step, and -10 at the point when the battery goes dead.
- **Rendering:** Renders images with the help of pygame, which will be used to show the battery of the agent through changes in color (Blue = High, Red = Low).

3. Algorithm Implementations

The main processing is contained in the algorithm.py, which applies 8 algorithms in three classes.

3.1 Dynamic Programming (DP)

Applied to model-based settings (FrozenLake) in which the transition dynamics are known.

- **Policy Evaluation (eval_pol):** This is an iterative update of the Value Function, and incrementally computes the Value Function by totaling future expected rewards until the magnitude of the change in the Value Function is less than a tolerance level.
- **Policy Iteration (policy_iter):** Alternation between policy_iter and a greedy policy improvement step. When there are no actions changed, the policy is terminated.
- **Value Iteration (value_iter):** This algorithm operates by bypassing estimation and improving in the same update step through the value update by one step; that is, it maximizes the action-values, i.e., maximizes over the entries of a table. It gives out a deterministic policy.

3.2 Prediction Methods

Approximated values of V(s) are used to approximate the time-varying variance with no model and a random policy.

- **Monte Carlo (MC) Prediction** (**mc_pred):** Predicts MC by a first-visit method. It produces complete episodes, archives the history, and takes a step backward to determine the return. The value is updated when the latter is the first time that has been used in the episode.
- **Temporal Difference (TD(0)) (td_pred):** This learning model changes the value, instead of the target, as the input in Reward after a step using the bootstrap target.

3.3 Control Algorithms (n-step TD)

The unified Agent class does Q-Learning and also SARSA with the capability of n-step lookahead.

- **N-step Implementation:** The trainloop makes use of three buffers: sbuf, abuf, and rbuf.
    - The algorithm will stabilize when step tau = 0.
    - It obtains the truncated point of return, G.
    - In case the episode was not completed yet (tau + n < T), the value of state S is bootstrapped with S.
- **Q-Learning (Off-Policy):** A bootstrap value is based on the maximum-over-next: maxa

Q(s t + n, a).
- **SARSA (On-Policy):** Other values of next action are the actual next action.

4. Parameter Adjustment & Interactive Features

The interface is app.py, where the user can change the hyperparameters in real-time, and the training configuration of the agent is re-run or updated immediately.

- **Discount Factor:** The Foundation module has a slider (between 0.0 and 1.0) that lets users see how short-sighted and far-sighted assessment alters the Value Function heatmap.
- **Learning Rate:** Learning rates are adjustable when using MC, TD, and Control algorithms, which affect the convergence and the rate of convergence.
- **Reward Shaping (Real-time):** In FrozenLake, a control known as a Hole Penalty slider gives a user an opportunity to adjust the internal transition probabilities of the environment env.P in real-time. It shows how raising the stakes in case of failure changes the duty of the best policy (turning the agent to be less risky/tailored).
- **N-step:** There is a slider (1-5) that allows the user to switch between TD(0) (1-step) and multi-step bootstrapping and observe the impact on the learning process.

5. Visualization & User Interface Design

UI is interested in the concepts of clearness and instant visual feedback.

- **Value Function Heatmaps:** In the DP part of our code, there is an operation by seaborn.heatmap like V(s) on the visualization. This enables users to be able to intuitively view safe lanes (dark red/blue) and hazardous places (holes/cliffs).
- **Convergence Plots:**
  - **Prediction:** st.line chart predicts the comparison of MC and TD estimates of the Start State per episode. This vividly shows that MC variance is so great compared to the bias/speed of TD.
  - **Control:** It has a rolling average (smoothed) reward plot that shows the reward curve of the agent as a function of 1000 or more episodes.
- **Custom Environment Rendering:**
  - In the case of Battery GridWorld, the tool is not a simple number plotter. It makes use of image_spot.image() in cadencing a simulation loop.
  - The BatteryEnv.render() takes advantage of PYG and displays a challenge grid on which the color of the agent dynamically changes depending on the battery level, and visualized feedback upon the state tuple (x, y, bat) is immediate.

6. Conclusion

This tool manages to deploy the needed collection of RL algorithms and environments, and packages them in an interactive interface. The project will deliver on the goal of becoming an educational platform through its ability to allow the manipulation of its core parameters, such as, but not limited to, gamma, n-steps, and reward functions, and have the effect of

visualization through heatmaps and simulations immediately to demonstrate the overall knowledge of the Reinforcement Learning mechanics.