Zewail City of Science, Technology and Innovation
University of Science and Technology
School of Computational Sciences and
Artificial Intelligence
DSAI 402 fall2025

Team Member [ID]

Mariam Mohamed Goda[202201223]

Doctor : Mohamed Fakhry

Rl Algorithms in Different Environment

A concise technical report for this project can follow the structure below; you can put it in a separate `REPORT.md` or as a section in your documentation.

---

1. Introduction

This project implements an interactive **web-based visualization tool** for classical **Reinforcement Learning (RL)** algorithms applied to grid-based environments.
The main goal is to help students and practitioners understand how value functions, policies, and agent behaviour evolve under different algorithms and environment dynamics

2. Objectives

- Provide an educational platform that demonstrates **Dynamic Programming, Monte Carlo, and Temporal Difference** methods on simple grid environments.
-

- Allow users to **interactively tune hyperparameters** (episodes, discount factor, learning rate, exploration rate, slip probability, etc.) and directly see their impact on learning.
- Offer **visual explanations** through grid heatmaps, policy arrows, trajectories, and episode-return curves to bridge theory and implementation.

---

3. System Overview

The system is a **Flask web application** with a **JavaScript frontend** that communicates via JSON APIs. Users interact through a browser UI to select environments and algorithms, while the backend executes RL algorithms and returns structured results.

3.1 High-Level Architecture

- **Backend (Flask):**
  - Serves HTML templates (`index.html`, `env_detail.html`).
  - Exposes API endpoints:
    - `GET /api/run_algorithm` – runs the selected RL algorithm.
    - `POST /api/simulate_policy` – simulates a given policy and returns a trajectory.
- **Core RL Components:**
  - Environments:
    - `GridWorldEnvironment` (deterministic).
    - `FrozenLakEnviroment` (stochastic, FrozenLake-style).
  - Algorithms:
    - Policy Evaluation, Policy Improvement, Policy Iteration.
    - Value Iteration.

        - Monte Carlo Every-Visit and First-Visit.
        - Temporal Difference TD(0).

- **Frontend (HTML + CSS + JS):**
    - `index.html` – environment selection landing page.
    - `env_detail.html` – main dashboard with controls, visualizations, and charts.
    - `static/style.css` – dark, modern UI with grid and cards.
    - `static/script.js` – API calls, grid rendering, policy arrows, charts, and animation control.

---

4. Environments

4.1 GridWorldEnvironment

- Represents a simple grid of size (rows×cols)(\text{rows} \times \text{cols})(rows×cols) with states defined as coordinate pairs (i,j)(i, j)(i,j).
- Provides a fixed action set `["up", "down", "left", "right"]` and a deterministic transition model stored in `env.Prob[state][action]`.
- Rewards:
    - Moving into walls or outside the grid keeps the agent in place with a negative reward.
    - Normal transitions give zero reward, while reaching the goal yields a positive terminal reward

4.2 FrozenLak Enviroment

- Extends the grid structure with **start (S)**, **goal (G)**, **frozen cells (F)**, and randomly generated **holes (H)**.
- Introduces a slip parameter that creates a **stochastic transition model**, mixing intended moves and lateral slips with specified probabilities.
- Rewards:
    - Reaching the goal gives a positive reward.
    - Falling into a hole or leaving the grid yields a negative reward, otherwise the reward is zero

---

5. Implemented Algorithms

5.1 Dynamic Programming Methods

- **Policy Evaluation**
  Iteratively computes the state-value function Vπ(s)V^\pi(s)Vπ(s) for a fixed policy π\piπ using the Bellman expectation equation until the maximum update is below a tolerance threshold.
  The implementation loops over all states, applies the current policy, and backs up expected returns using the environment's transition probabilities.

- **Policy Improvement and Policy Iteration**
  Policy improvement selects, for each state, the action that maximizes the expected return under the current value function.

Policy iteration alternates between evaluation and improvement until the policy stabilizes, yielding an optimal policy and its value function.

- **Value Iteration**
  Combines evaluation and improvement into a single update by directly applying the Bellman optimality operator on the value function.
  The algorithm repeatedly backs up the maximum expected return over actions until convergence, then extracts the greedy policy.

5.2 Monte Carlo Methods

- **Every-Visit Monte Carlo**
  Generates complete episodes by interacting with the environment and records all returns observed after each state–action pair.
  For each pair, it averages all observed returns to estimate Q(s,a)Q(s, a)Q(s,a), allowing policy derivation by taking the best action at each state
- **First-Visit Monte Carlo**
  Similar to Every-Visit, but only the **first occurrence** of each state–action pair in an episode contributes to the return estimate.
  Both implementations optionally track episode returns to visualize convergence and learning dynamics

5.3 Temporal Difference Learning (TD(0))

- Uses an **ε-greedy policy** over current state–action values to balance exploration and exploitation.
- Updates Q(s,a)Q(s, a)Q(s,a) incrementally using one-step bootstrapping with the maximum next-state value, implementing the TD(0) rule.

- Records per-episode returns when tracing is enabled, enabling comparison with Monte Carlo learning curves

---

6. API Design

6.1 `/api/run_algorithm` (GET)

- **Input parameters (query):**
  - Environment: `env` ∈ `{ "frozenlake", "gridworld" }`.
  - Algorithm: `algorithm` ∈ `{ "policy_iteration", "value_iteration", "mc_every", "mc_first", "td" }`.
  - Grid parameters: `rows`, `cols`, `slip`.
  - Algorithm hyperparameters: `gamma`, `theta`, `episodes`, `alpha`, `epsilon`.
  - Flags: `trace`, `stochastic`, `simulate`.
- **Output (JSON):**
  - `policy` and `values` serialized as string-indexed dictionaries.
  - `rewards` and `rewards_grid` for reward visualization.
  - `history` and `episode_returns` (when tracing is enabled).
  - `trajectory` when simulation is requested.
  - Metadata: `iterations`, `time_taken`, `rows`, `cols`, `env`.

6.2 `/api/simulate_policy` (POST)

- Accepts an environment configuration and a serialized policy
-

- Reconstructs the environment and policy, then simulates a trajectory starting from the initial state, returning stepwise transitions and cumulative rewards.

---

7. Frontend and Visualization

- The UI is implemented with **HTML templates**, a shared **CSS theme**, and a dedicated **JavaScript file** that handles dynamic behaviour.
- Environment and algorithm parameters are controlled through sliders and checkboxes, with real-time updates of numeric labels

Key visualization elements:

- Grid view with cell types `S`, `F`, `H`, `G`, and policy arrows over each state.
- An animated agent marker that moves according to the simulated trajectory, optionally with stochastic transitions.
- Chart.js plots displaying episode returns to illustrate learning progress for Monte Carlo and TD algorithms