מכון טכנולוגי חולון
Holon Institute of Technology

Reinforcement Learning Assignment

Monte Carlo Control & TD Learning in Treasure Hunt Environment

---

Overview

In this assignment, you will build a custom reinforcement learning environment, implement Monte Carlo and Temporal-Difference (TD) control algorithms, and compare their performance. This hands-on task is designed to deepen your understanding of model-free control methods.

Part 1: Implement the Treasure Hunt Environment

Design a **grid-based environment** where an agent must navigate through obstacles, collect treasures, and reach a goal state. This will be your testing ground for RL algorithms.

Environment Specifications:

- At least **5x5 grid**
- **Start state**, **goal state**
- At least **2 treasures** (+ reward)
- At least **1 trap** (− reward)
- **Empty cells** with small/zero reward
- Actions: up, down, left, right
- Deterministic or slightly stochastic transitions (optional)

Required Interface:

```
class TreasureHuntEnv:
  def reset(self) -> Tuple[state, info]: ...
  def step(self, action) -> Tuple[next_state, reward, done, info]: ...
  def render(self): ...
  def get_state_space(self): ...
  def get_action_space(self): ...
```

You may choose to represent the state as an (x, y) coordinate or use another encoding. Provide a method to **visualize** the environment and learned policies.

Part 2: Monte Carlo Control

Tasks:

1. **Monte Carlo Prediction**:
   - Use **First-Visit Monte Carlo** to estimate the state-value or action-value function.
2. **Monte Carlo Control with Exploring Starts**:
   - Implement a policy improvement loop using **exploring starts**.
3. **Monte Carlo Control with ε-soft policies**:
   - Use an **ε-greedy** policy for both exploration and improvement.

Suggested Functions:

```
def monte_carlo_es(env, num_episodes): ...

def monte_carlo_epsilon_soft(env, num_episodes, epsilon): ...
```

Part 3: TD Learning – SARSA & Q-learning

Choose **at least one** TD algorithm to implement and compare with Monte Carlo Control:

SARSA (on-policy):

```
def sarsa(env, num_episodes, alpha, gamma, epsilon): ...
```

Q-learning (off-policy):

def q_learning(env, num_episodes, alpha, gamma, epsilon): ...

You can explore different values of:

- Learning rate alpha
- Discount factor gamma
- Exploration rate epsilon

Part 4: Evaluation & Report

Visualizations:

- Value function heatmaps
- Policy maps (arrows showing actions)
- Return vs. episodes plot for each algorithm

Report (inside the notebook):

Include:

- Design of your environment
- Description of algorithms implemented
- Comparison of Monte Carlo vs. TD methods
- Insights and observations (e.g., convergence, stability, sample efficiency)

Bonus (Optional)

- Add **stochastic transitions** (e.g., “windy” gridworld).

Submission Instructions

Submit all in jupyter notebook (code, graphs, results, explanations)

- Your explanations are a central part of the submission. We expect well-reasoned analysis and interpretation of your results. Efforts to improve results and critically evaluate outcomes will be rewarded accordingly.
- You may work individually or in pairs.
- Submissions that are heavily aided by generative AI tools will be reviewed carefully and may require additional justification.
- It is highly recommended that you write your code and report independently and with full understanding. Work that is not well understood by the student may negatively affect your learning and evaluation.
- Ensure timely submission according to the specified deadline in Moodle.

Important: You are expected to be able to explain all parts of your code and report. Questions about your assignment may be asked during the final exam to assess your understanding. Work that appears unfamiliar to the student will not be accepted.

Tips

- Keep episode lengths bounded to avoid infinite loops.
- Use random seeds for reproducibility.
- Start with small grids to debug faster.
- Use print/debugging or render() to understand agent behavior.