Tragedy-of-the-Commons RL Environment

December 5, 2025

Background and Motivation

Commons and resource-management problems are common in ecology, economics, and social systems.

In these settings, a shared resource is harvested by multiple agents. If all agents harvest aggressively, they may collapse the resource; if agents harvest too little, they may starve individually.

Reinforcement Learning (RL) provides a way to model and study how an agent might behave in a population of non-learning (or randomly acting) agents, and whether the agent can learn sustainable harvesting strategies.

This assignment represents the first step in my larger project which is to experiment with whether multiple reinforcement-learning agents can coexist sustainably in a commons-type environment without explicit coordination or predefined collaboration. In particular, I'm trying to see if a population of N agents can learn to converge on an equilibrium harvesting strategy, one where each agent extracts enough resources to survive over time, yet where the group collectively avoids over-exploitation and prevents collapse of the shared resource.

In this assignment, you will work with a simplified simulation environment inspired by the "tragedy of the commons." The environment is described below.

Environment Description

- There is a single renewable resource with maximum capacity $R_{\max}$.
- At the start of each episode, the resource is initialized at $R_{\max}$.
- In each "round" (time step):
  1. The order of all agents (the learning agent plus K non-learning "non-agents") is shuffled uniformly at random.
  2. Each non-agent before the learning agent extracts a random amount uniformly in $[0, \text{max\_nonagent\_take}]$.
  3. The learning agent then chooses how much to extract (via a discrete action). The agent *observes only*:

    - Current resource level R, and
    - Number of non-agents who acted before it in this round.

4. After the agent, the remaining non-agents extract resource (random as above).
5. After all extractions, the resource regenerates according to a logistic growth rule:

$$\Delta R = \text{regen\_rate} \times R \times \left(1 - \frac{R}{R_{\text{max}}}\right).$$

6. If resource drops to zero (or below), the environment collapses; episode terminates with negative reward; the agent dies.
7. If the agent extracts below a minimal survival threshold, the agent starves; episode terminates with negative reward.

- Otherwise, the episode continues.

This environment captures key aspects of commons dilemmas:

- renewable but limited resource (via logistic growth),
- competition (multiple extractors),
- partial observability (agent does not know future extraction),
- decision tradeoff between short-term gain and long-term sustainability.

Tasks

attempt the following tasks.

1. **Implement the reward function.** In the `TocEnv` class, complete the method `_reward(self, agent_action)`. Here is my idea for the reward function:

    - If the agent's extraction is below `survival_min_take`, the episode ends with a negative reward (e.g. -1).
    - If resource is ≤ 0 (after any extraction), the episode ends with a negative reward (e.g. -1).
    - Otherwise, the agent receives a positive reward for surviving (e.g. +1), and the episode continues.

    answer: see `_reward(self, agent_action)` function in anwserCode/TocEnv.py

2. **Choose number of buckets for the discretization of the observation space.** Because the first observation in the observation vector is continuous this discretization method will allow for the use of Q-learning and SARSA as they require discrete state spaces. e.g.:

$$\text{state} = \left(\lfloor R/(R_{\text{max}}/N)\rfloor,\ n_{\text{before}}\right)$$

where n_{before} is number of non-agents before the agent in the current round. The more buckets N the larger the state space becomes. But it also allows for the observation to be more accurate. So there is a trade-off.

answer: I chose 15 "buckets" as it allowed for less information loss than 10 which was what I originally had but 15 "buckets" still gives me have a relatively small state space.

$$stateSize = (\text{num\_buckets}) \times (K + 1)$$

$$stateSize = 15 \times (5 + 1)$$

$$stateSize = 15 \times 6 = 90$$

3. **Implement a tabular Q-learning algorithm.** Write a function or class that uses the environment + discretization + Q-learning to learn a policy. Use ε-greedy exploration, learning rate α, discount γ, etc. Track total reward per episode over training.

   answer: see `train(self)` function in answerCode/Learning.py/Qlearning

4. **Implement a tabular SARSA algorithm.** Similarly, implement SARSA (on-policy TD), using the same discretization. Track reward per episode over training.

   answer: see `train(self)` function in answerCode/Learning.py/Sarsa

5. **Compare performance via plots.** After training both algorithms for a sufficient number of episodes (e.g. 5,000–20,000), produce:

   - A plot of average (or smoothed) reward per episode vs episodes, comparing Q-learning and SARSA.
   - A plot of ending resource level per episode (to evaluate ecological collapse or stability).

   Answer: see plot photos in

   `Tragedy_of_commons_SRL_Midterm2`

   named:

   `ending_resource_level_QLearning`

   `ending_resource_level_Sarsa`

   `reward_stabilization`

6. **Write a short reflection.** Reflect on:

   - Differences between Q-learning and SARSA behavior (e.g. risk-taking vs conservative harvesting).
   - Whether the learned policies tend to sustain the resource or collapse it.

- How partial observability (unknown future extractions) affects strategy.
- Why might the resource graphs show a lot of variance.
- What modifications (e.g. reward shaping, environment changes) might encourage better and/or different outcomes.

Answer: Both methods behave pretty similarly in this environment. If you take a look at the plots Q-Learning on average converges to a higher reward value than SARSA which makes sense as Q-Learning is off-policy and updates the Q-table as if it always takes the optimal action. I am a little surprised by the fact that Q-learning performed as well as it did, given the fact the this environment is very stochastic. I would've thought that the Q-learning updates would make it miss information a lot more than it did. I did think that Sarsa would be the better algorithm for this environment given its on policy learning but both performed way in any case so who's complaining.

The resource pool never collapsed in my experiments, which is encouraging in the sense that the learned policy supports sustainable behavior. However, from a learning dynamics perspective, I would have wanted to see early episodes where the resource does collapse, followed by mid to late episode behavior in which the agent gradually discovers a strategy that preserves the pool. That transition from unsustainable exploitation to stable equilibrium is common in many Tragedy of The Commons RL experiments. This behavior is something you do see in my larger project.

In this particular setup, collapse dynamics are unlikely to emerge. There is only one learning agent, and the environment is structured so that neither the agent nor any single non-learning actor can collapse the environment in a single step. This was intentional: the extraction limits were chosen so that collapse must result from collective over-harvesting over multiple steps, not from a single catastrophic action. Because the non-learning actors behave randomly and are individually bounded in how much they can extract, and because the learning agent is likewise bounded, the system tends to hover in a regime where resource levels fluctuate but rarely reach zero.

As a consequence, the learning problem is relatively "safe": the agent is not exposed to large negative feedback from the environment, which limits the of the emergent dynamics of the environment. To observe meaningful collapse-and-recovery behavior, the environment would likely need 1. multiple learning agents whose combined actions can meaningfully stress the resource which is what my larger project explores.

Something I would improve is to increase extraction limits that would make collapse possible with a single agent. This seems like a pretty simple fix, so why did I not try this. Well I did, and When I tried to increase the limits I could never get out of a collapse. This means once the environment collapsed the agent could never figure out how to get out of the collapse in a later episode. I don't know why this was happening. But I think if I had more time I could figure out a solution to this problem.

Another improvement would be coming up with a different reward function. My reward function is very basic which is good as this environment is meant to be basic but I think with stronger reward function I could uncover more interesting behavioral dynamics.