Objective:

The project aims to develop a reinforcement learning agent able to determining the most effective methods to irrigate crops, thereby avoiding farmer water waste. While overirrigation leads to degradation of the soil, water loss, and lower crop yields, underwatering reduces output. By simulating different climatic conditions, this study aims to maximize irrigation schedules. Especially in places where water is limited and weather is erratic, this is crucial for improving sustainable farming practices. The main objective here is to:

1. Keep soil moisture balanced (not too wet or dry).
2. Grow healthy crops without wasting water.
3. Achieve bonuses by consistently maintaining optimal moisture levels.

Environment:

Within this simulated agricultural setting, players take on the role of agents tasked with water management in response to changes in the weather, soil, and crop development. The agent learns to use water effectively to keep the crop healthy. Daily, the incentive mechanism directs the agent to make the right choices. Visualizing the crop's growth and soil moisture lets one track the agent's development. We created a custom gym environment simulating an irrigation scenario with different temperatures, humidity, crops, and climates to train a reinforcement learning on. The simulation's length is 120 days, which is sufficient to span a whole growing season.Following this, the environment is reset and the agent is given the option to add water daily using one of eleven various irrigation levels, ranging from zero to the maximum permitted per day. A specific amount of water is immediately affected by the action chosen. At each day, the agent sees the current state (observations), the below observations help the agent make decisions.

1. Soil moisture (how wet the soil is, scaled 0 to 1).
2. Crop water-loss rate.
3. Crop Coefficient: Indicates crop's water needs.
4. Leaf Area Index (LAI): Measures crop's leaf density.
5. Growth progress: How far along the crop is in the growing season.
6. Day number in the season.
7. How much irrigation was done the previous day.

The agent tries to maximize this value. The reward is calculated based on below factors, where agent's goal is to maximize this reward by balancing irrigation and crop health over the course of the growing season.

1. The whole 120 days is divided into 3 stages of 40 days each- seedling, vegetation and mature. If the span goes less than 30% (means less water optimality) days, then crop become stressed.
2. Soil moisture (theta_norm): The closer the soil moisture is to the optimal range (0.6 to 0.9), the better the reward.
    a. If the soil is too dry (below 0.6), it gets a negative reward.

b. If it is too wet (above 0.9), it also gets a negative reward.

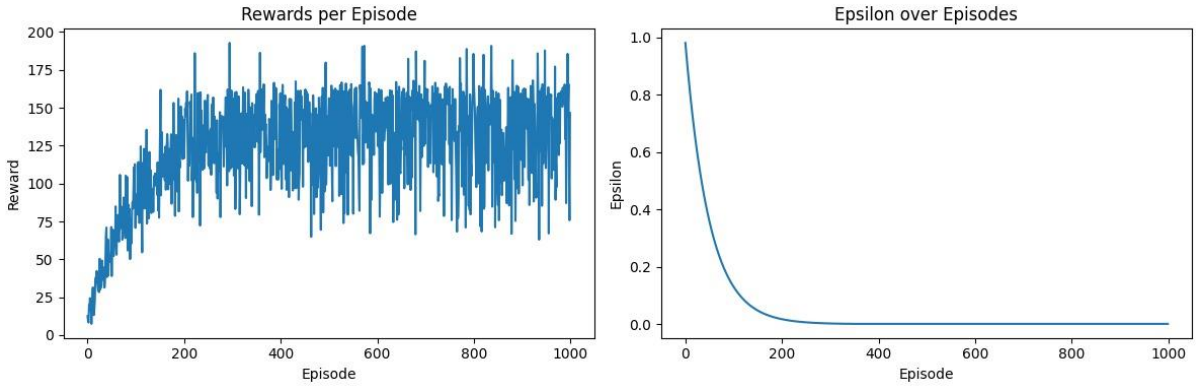
3. Positive reward: Soil moisture is in the optimal range (0.6- 0.9), crop stress is low, and water usage is efficient.
4. Negative reward: If the soil is too dry or too wet, if crop stress is high, or if water is wasted.
5. Crop stress factor: Based on soil moisture, the agent is penalized if the crop is stressed due to low moisture.
6. Water usage: The agent gets a penalty if it uses too much water. If the agent uses a lot of water (high irrigation), it gets a negative reward for wasting resources.
7. Temperature penalties: If the temperature is too high (over 35°C) or too low (below 5°C), the agent receives a penalty.

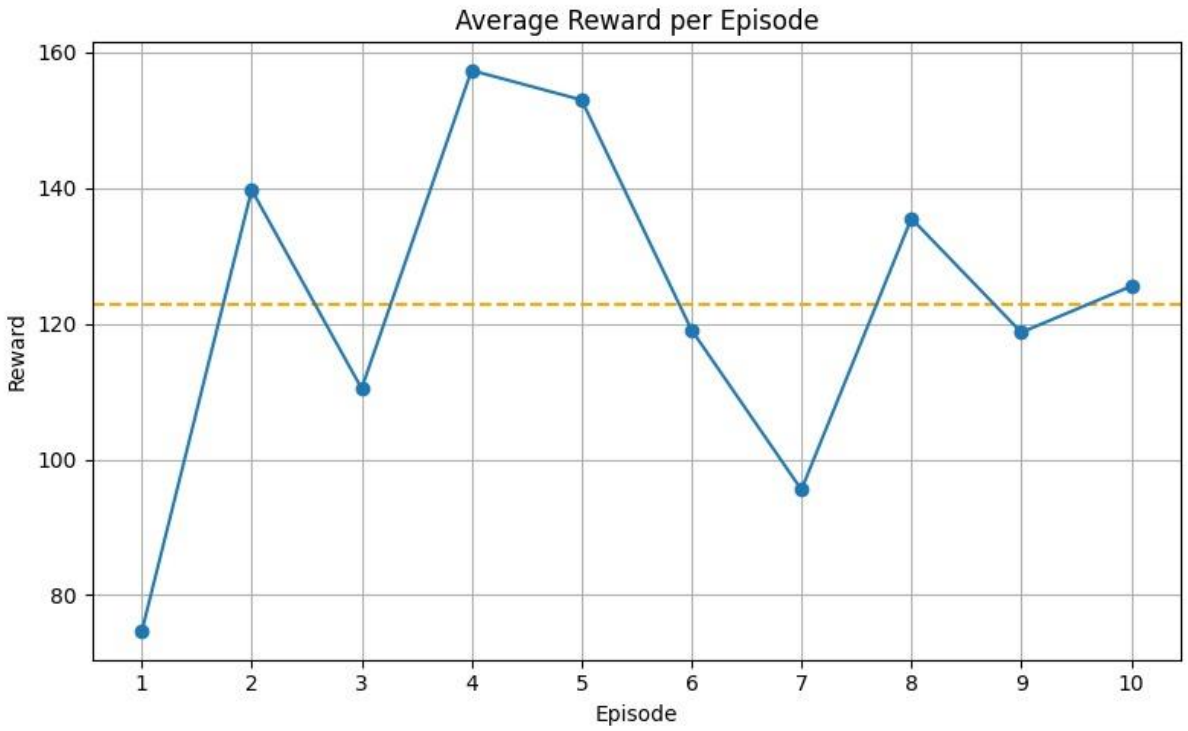
Custom Agent Implementations:

DQN:

The DQN agent uses a deep neural network to learn about its environment and obtain an estimate of Q-values. It updates its Q-values using the Bellman equation and experience playback during training. The agent can learn to control irrigation effectively and enhance its decision-making skills by interacting with the simulated environment.

The overall reward during evaluation shows that the agent can be trained to make well-informed irrigation decisions resulting in effective water use and optimal crop development.





The graph shows the agent's reward for every training session. A rising trend in the reward over time indicates that the agent is improving at making decisions while figuring out how to maximize the task. The graph also shows some reward variation, which can indicate that the agent is still trying out various strategies and not fully using the acquired policy just yet.

DQN Implementation using SB3:

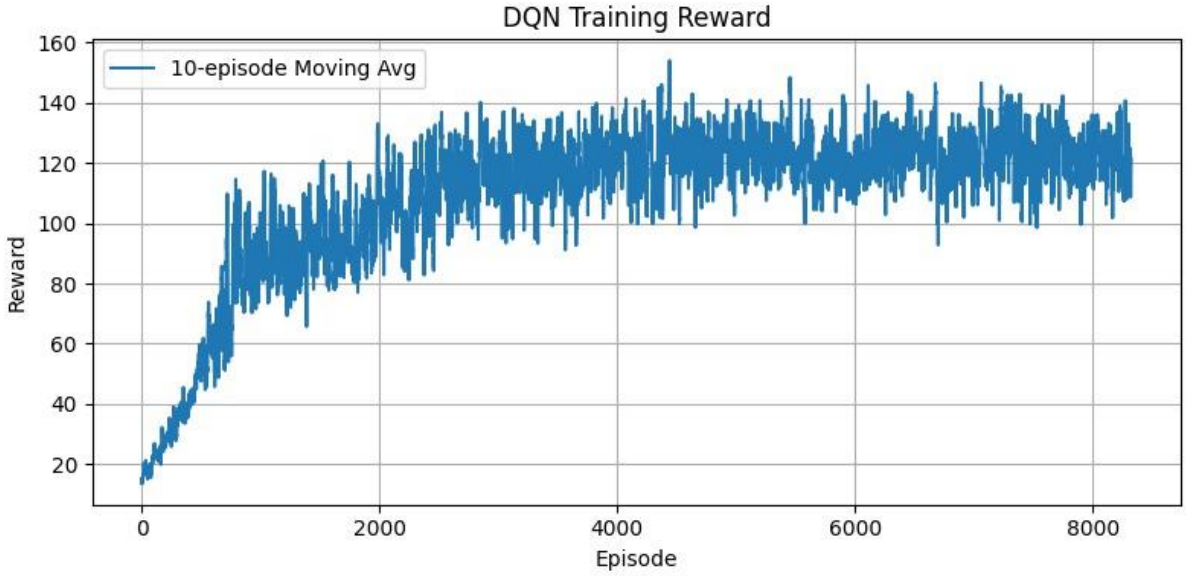
The DQN model uses deep neural networks to approximate Q-values and guide action selection in reinforcement learning.

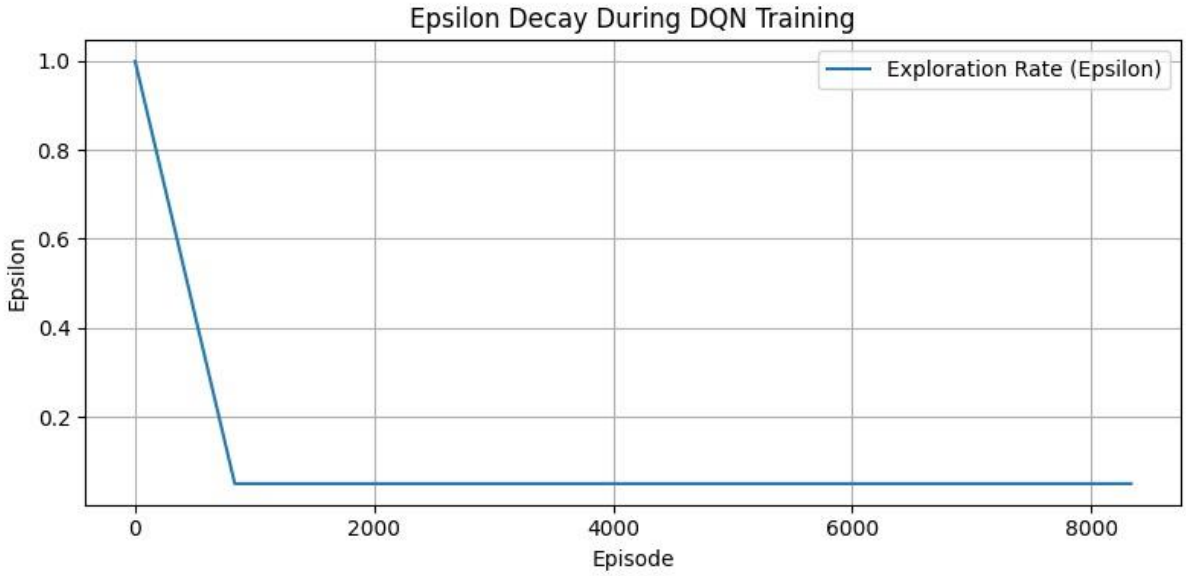
Model Parameters:

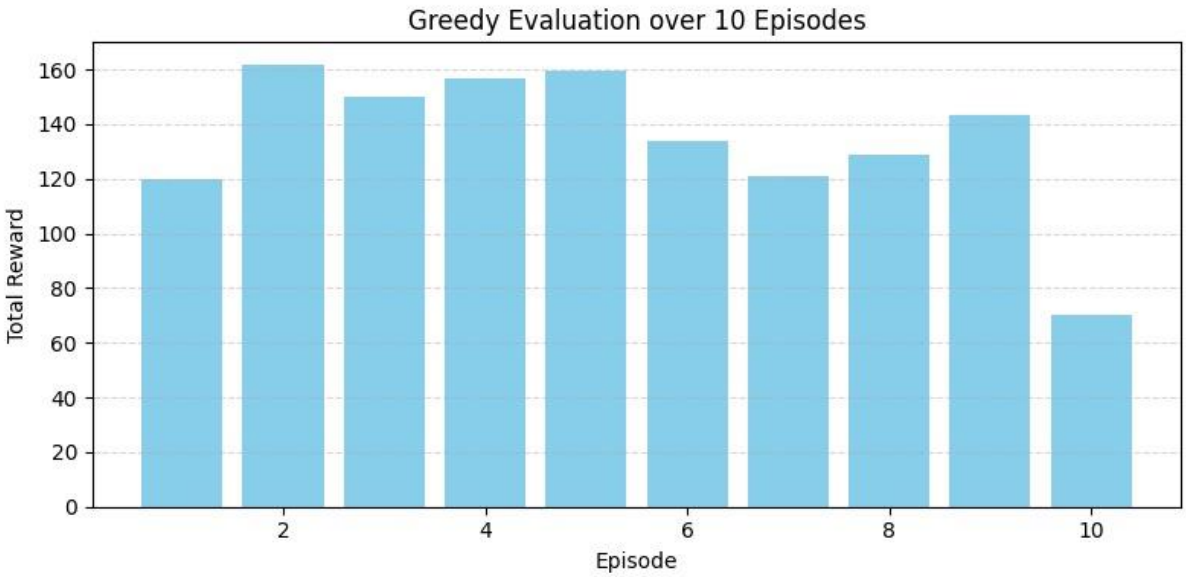
1. policy: The neural network architecture to use for decision-making (in this case, MlpPolicy means a Multi-Layer Perceptron).
2. learning_rate: The learning rate for training.
3. buffer_size: The size of the experience replay buffer.
4. learning_starts: The number of steps before starting to learn.
5. batch_size: The size of batches sampled from the replay buffer.
6. target_update_interval: How often the target network is updated with the model's weights.
7. train_freq: The frequency at which the model is trained.
8. The model is trained for 10,000 timesteps using the learn method.

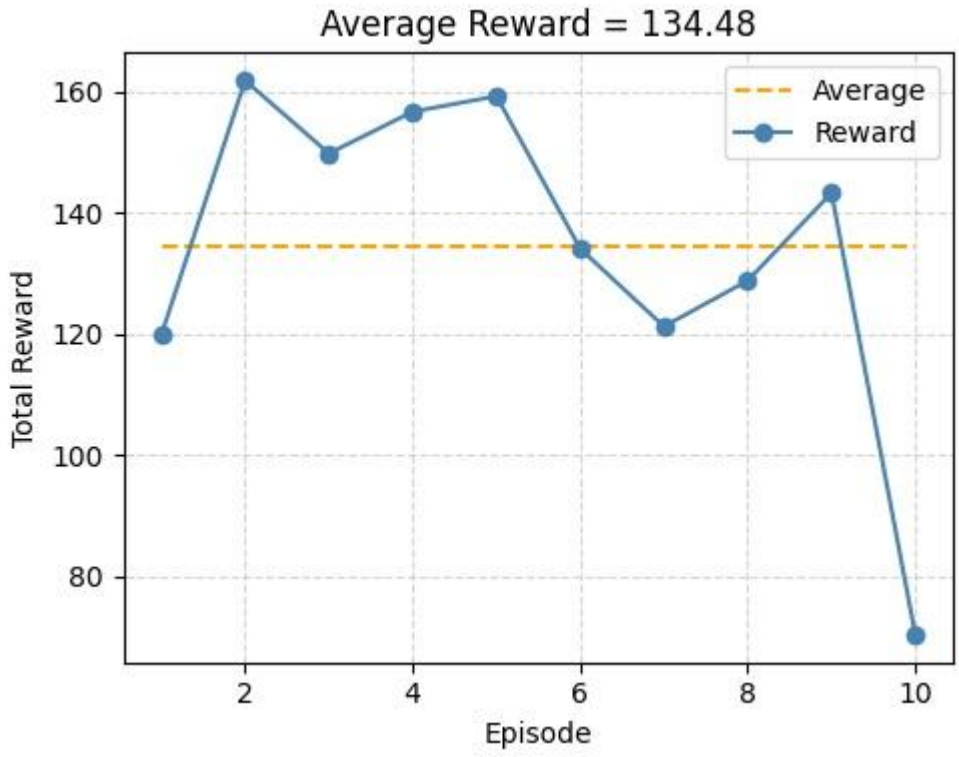
Workflow:

1. Training: The agent is trained in the SimpleIrrigationEnv environment, logging rewards and epsilon decay during training. The training results (reward curve and epsilon decay) are visualized.
2. Evaluation: After training, the agent is evaluated using a greedy policy. The evaluation rewards are plotted, and the average reward is calculated.
3. Rendering: A single episode is run with the trained model, and the environment is rendered to visualize the agent's actions.







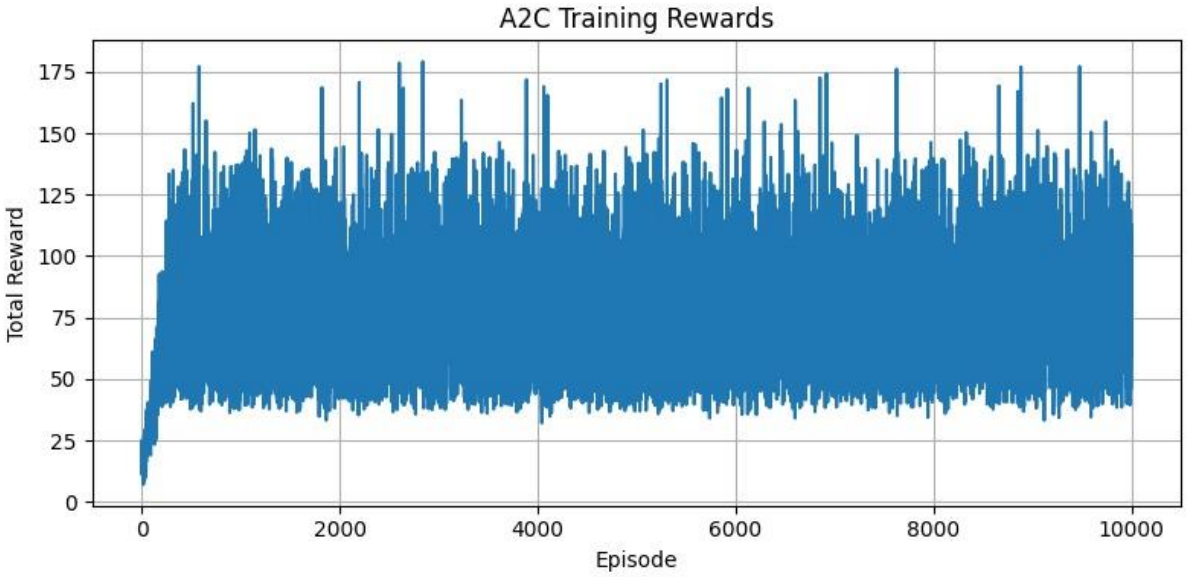


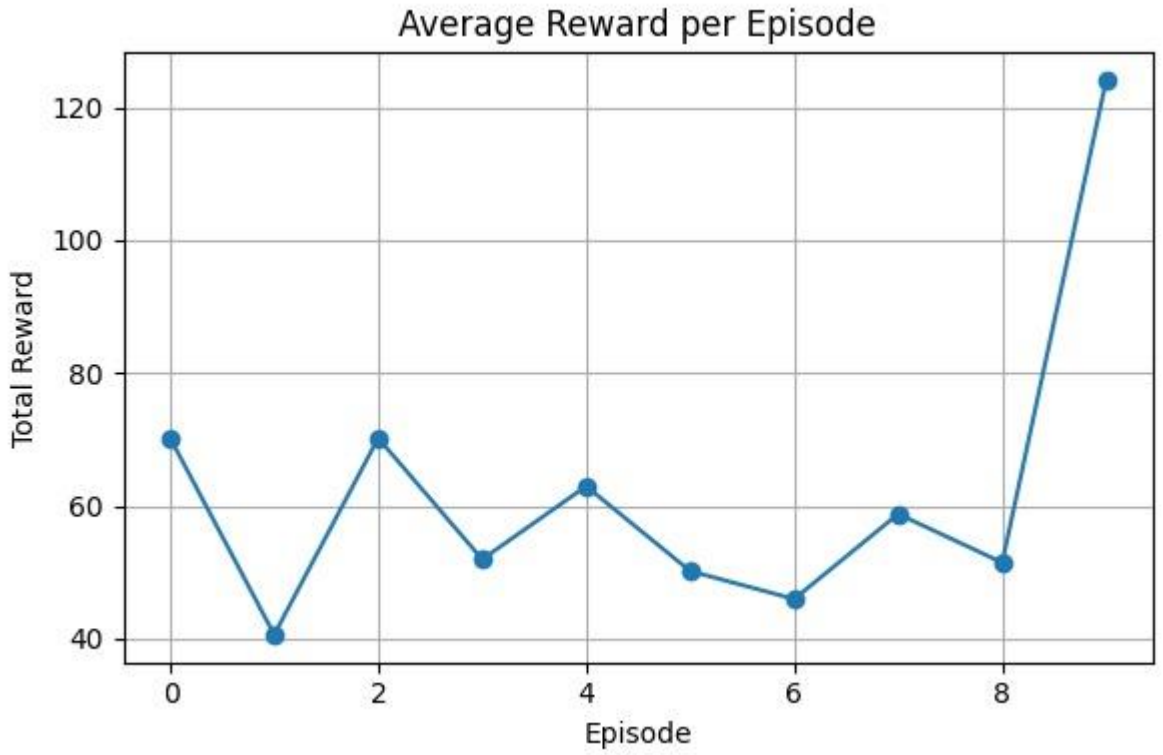
The performance of the agent improves and the incentives increase as it discovers the optimal policy, as seen in the DQN training graph. The 10-episode average movement helps to smooth out fluctuations, therefore the agent is maintaining its effectiveness. At first, the agent experiments with several strategies, which makes its rewards somewhat erratic. But, as training goes on, it begins to apply what it has picked up and its rewards get more steady. At the conclusion of the 8,000 episodes, the agent has finished the training and is seeing more gains.

A2C:

The Actor-Critic method combines two key components, the actor and the critic, which work together to optimize decision-making through learning from the environment.

1. Actor: The actor decides which action to take based on the current state of the environment. It outputs a probability distribution over possible action.
2. Critic: The critic evaluates the action taken by the actor by estimating the value of the state, helping the agent learn how good its actions are.
3. Hidden Layer: The input state (e.g., the current moisture level of the soil) is passed through a hidden layer to learn complex features.
4. Policy Layer (Actor): The policy layer outputs the probability distribution over the available actions (discrete actions in this case). The actor will sample actions from this distribution.
5. Value Layer (Critic): The value layer outputs a single value representing the predicted value of the current state (how good the current state is for achieving the goal).
6. Discount Factor: This is a hyperparameter that determines how much the agent should prefer immediate rewards over future rewards. Typically, values close to 1.0 (like 0.99) indicate that future rewards are almost as important as immediate rewards.





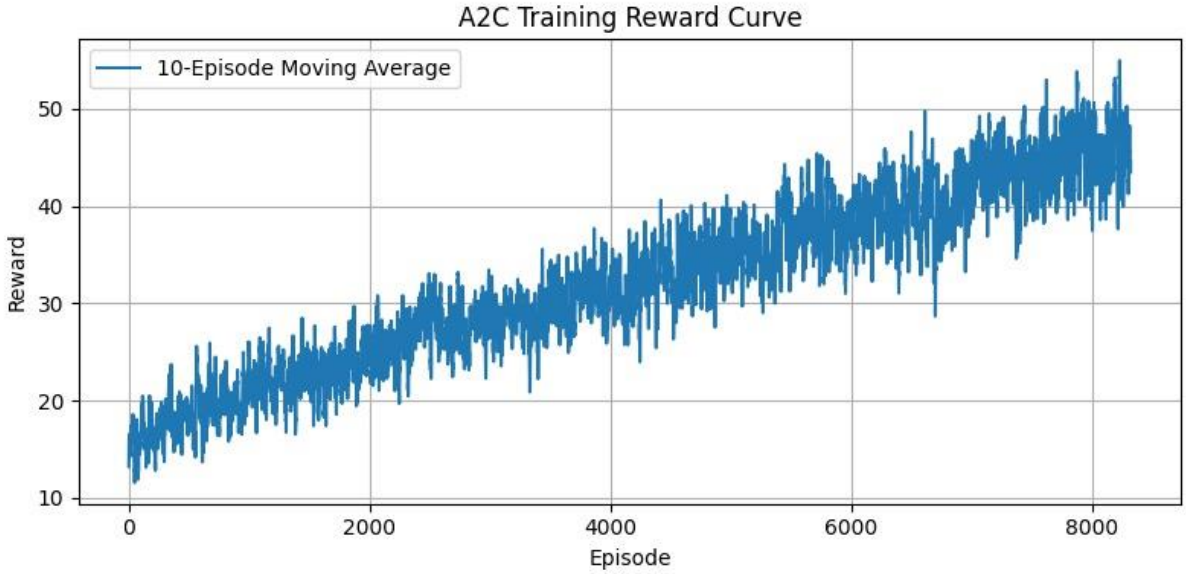
The total award is shown on the graph as you go through the training episodes. An agent is beginning to learn to enhance its actions when, after a certain number of episodes, the reward variability increases from a low starting point. The agent's reward starts to level out after completing a few thousand episodes, which indicates that they continuously perform well on the task.

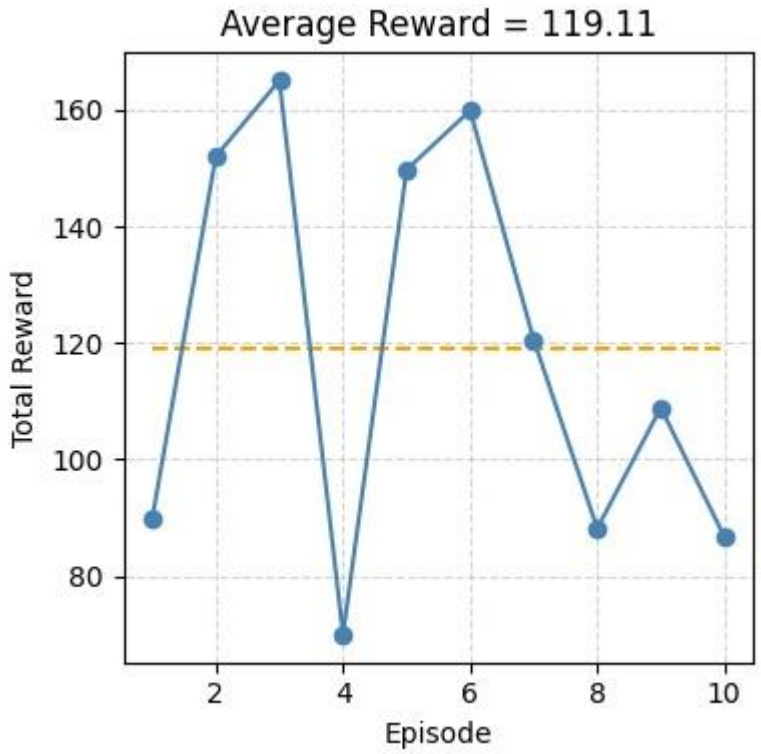
A2C Implementation using SB3:

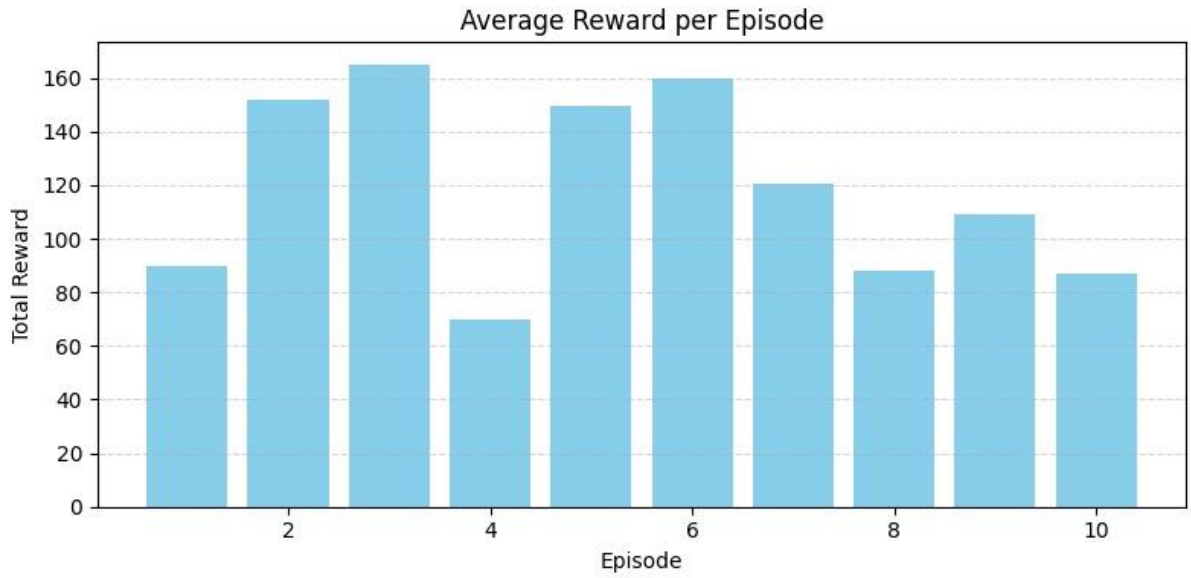
The A2C model is built using Stable Baselines 3, which simplifies reinforcement learning tasks. The A2C model:

1. Policy: Uses a multi-layer perceptron (MLP) policy network ('MlpPolicy') to predict actions.
2. Learning Parameters: It uses a learning rate of 1e-4, and the gamma (discount factor) is set to 0.99, meaning the agent values future rewards highly. The gae_lambda (Generalized Advantage Estimation) is set to 0.95 to control the smoothing of the advantage estimate.
3. Entropy Coefficient: The entropy coefficient (ent_coef=0.01) is added to encourage exploration by penalizing the agent for becoming too deterministic.
4. Stores rewards for each episode (ep_rewards), which is used to visualize the agent's learning progress.
5. Logs rewards at each training step and appends them to ep_rewards for later use.
6. Model Training: The model is trained for 10,000 timesteps using the model.learn() function. During training, the model interacts with the environment and adjusts its policy based on the rewards it receives.

7. Callback Usage: The TrainLoggerCallback is passed as a callback during training to collect and log episode rewards.
8. After training, the model is saved to disk (a2c_logs_10000.zip) for later use. The evaluate_greedy function evaluates the trained model over multiple episodes.





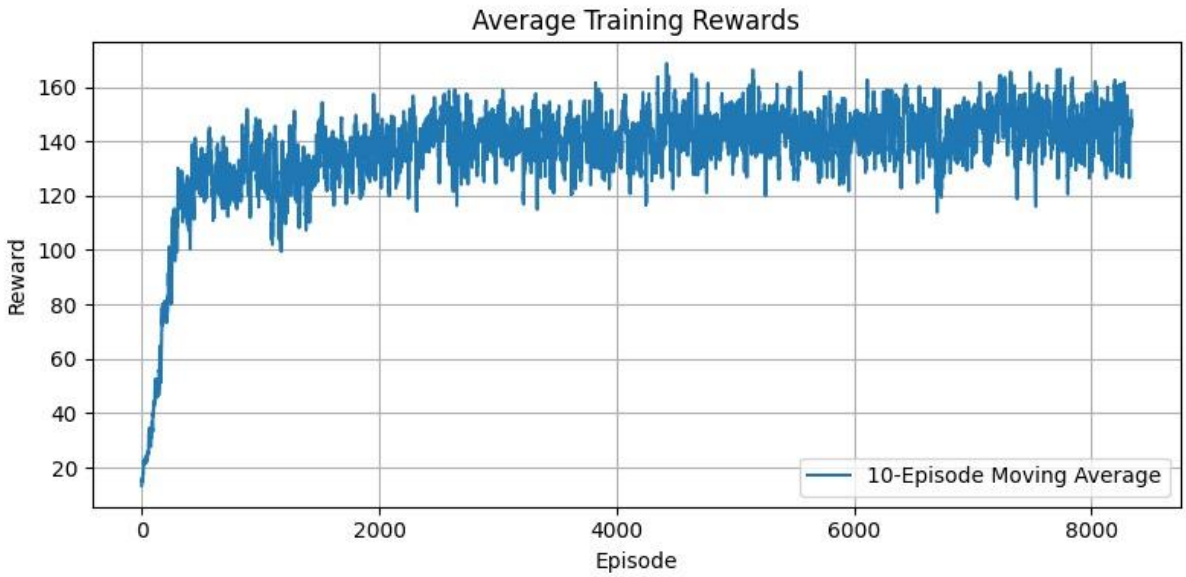


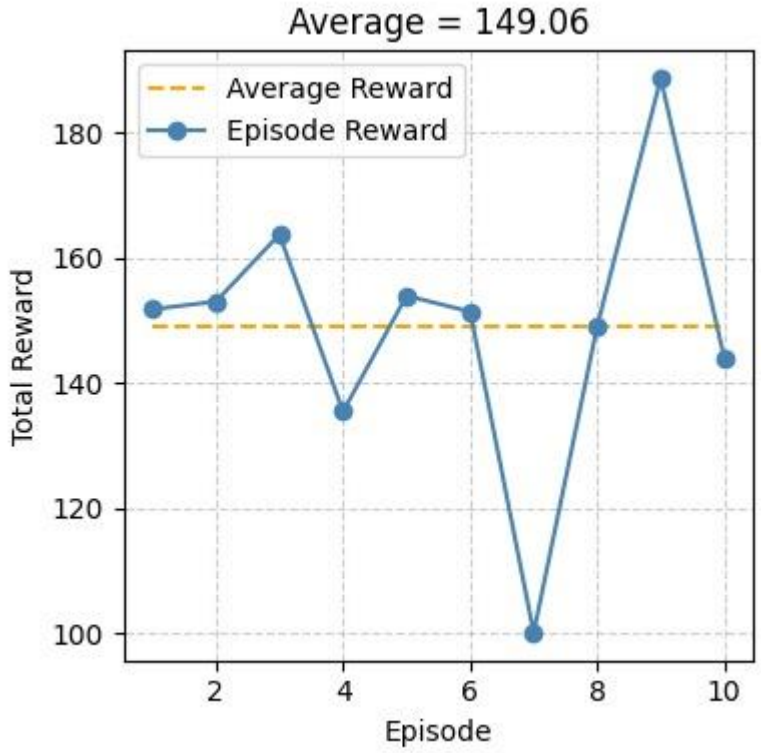
The A2C training plot shows increasing incentives as the agent acquires knowledge and experience. At first, exploration creates a significant amount of reward variation; but, when the agent starts to apply its learned policy, the rewards stabilize. The 10-episode moving average shows consistent progress, which suggests the agent is learning effectively overall. The agent consistently receives better incentives by the time training is finished.

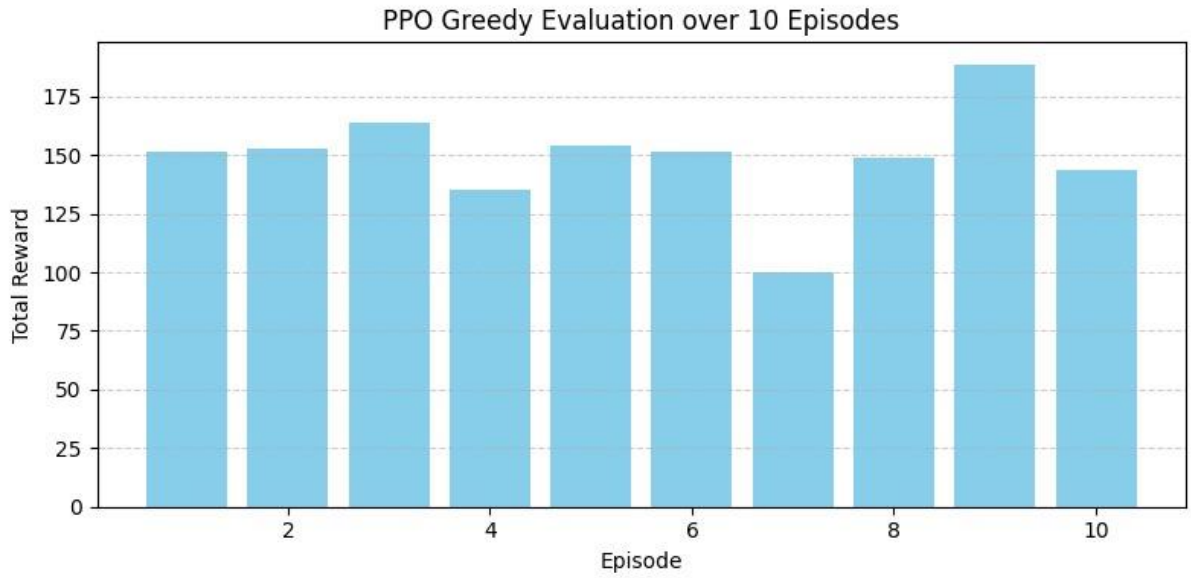
PPO:

Training the agent to complete a synthetic irrigation task aims to optimizing rewards (healthy crops, least water use) by means of irrigation management optimization (e.g., watering crops). Among the most modern reinforcement learning techniques, Proximal Policy Optimization (PPO) is popular for its consistency and efficiency in sampling. A policy gradient approach maximizes predicted rewards and guarantees that policy changes do not stray too far from the present policy, hence optimizing the policy.

1. Greedy Action: The agent selects actions based on the learned policy and takes steps in the environment until the episode ends.
2. Episode Execution: The agent interacts with the environment, and the environment is rendered after each action, showing the agent's decision-making in real time.
3. Reward Display: After completing the episode, the total reward for the episode is displayed in the console.
4. Model Training: The PPO model is trained using the SyntheticIrrigationEnv, and episode rewards are logged with the TrainLoggerCallback.
5. Model Evaluation: After training, the model is evaluated using a greedy policy (always taking the best action according to the learned policy).
6. Rendering the Environment: A single episode is run with the trained model, and the environment is rendered to visualize the agent's actions.

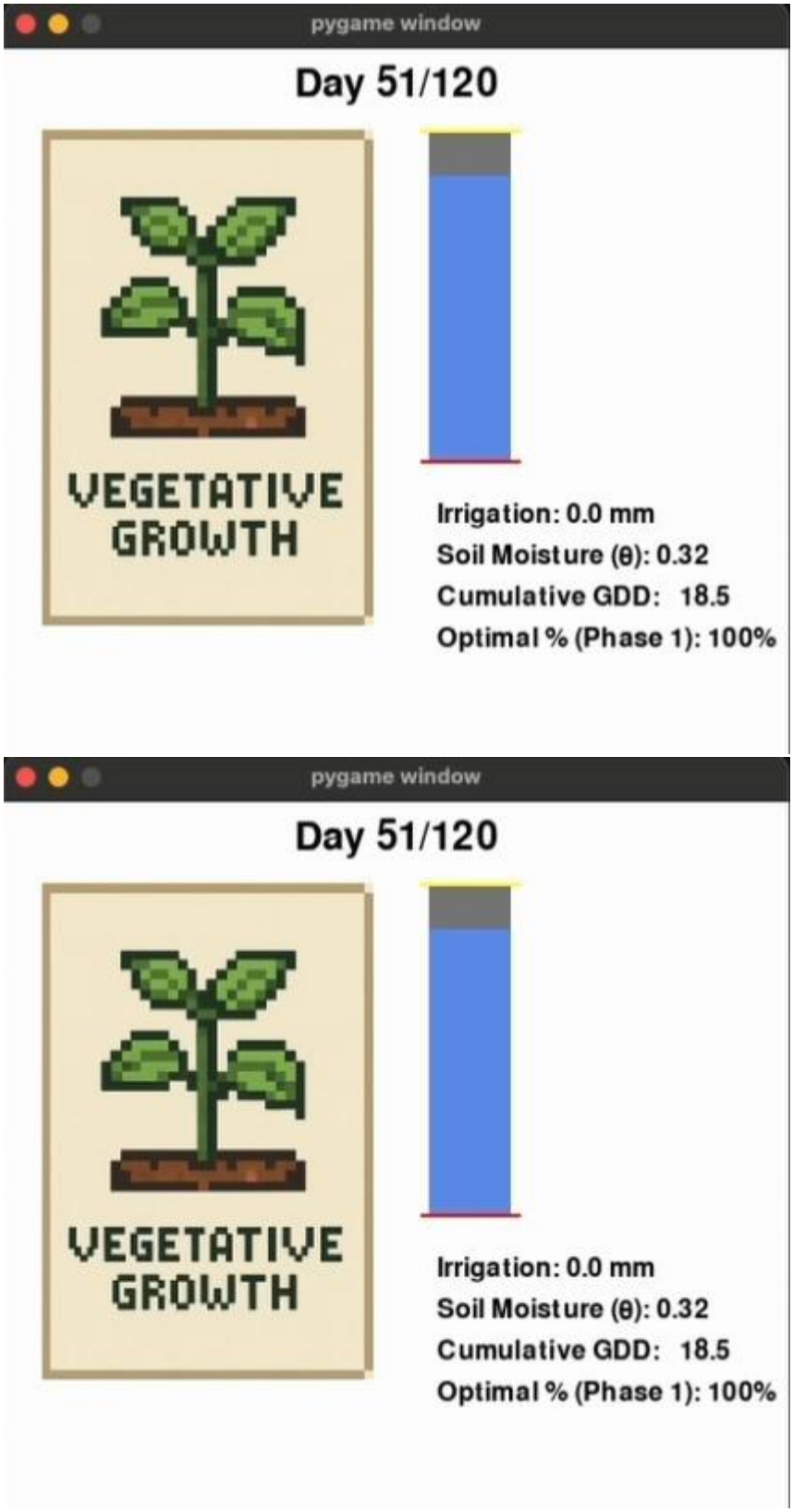

Average Training Rewards
Reward
Episode
10-Episode Moving Average
160
140
120
100
80
60
40
20
0
2000
4000
6000
8000


Average = 149.06
Average Reward
Episode Reward
Total Reward
Episode
180
160
140
120
100
2
4
6
8
10

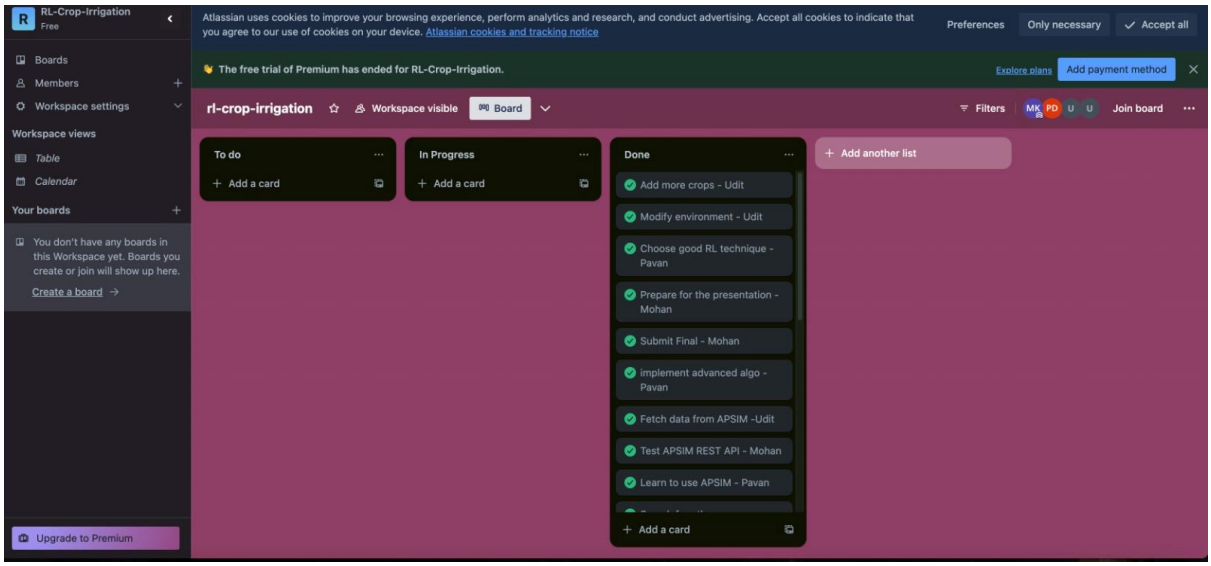


The fast first rise in rewards shown in the PPO training graph indicates that the agent quickly adopts a clever strategy. The rewards level out after the first phase assuming the agent has discovered a consistent policy. Consistent progress is highlighted by the 10-episode mean flattening down the payout curve. The story shows that the PPO agent could maximize its policy and receive regular and high rewards.

Results:


pygame window
Day 51/120
VEGETATIVE GROWTH
Irrigation: 0.0 mm
Soil Moisture (θ): 0.32
Cumulative GDD: 18.5
Optimal % (Phase 1): 100%
pygame window
Day 51/120
VEGETATIVE GROWTH
Irrigation: 0.0 mm
Soil Moisture (θ): 0.32
Cumulative GDD: 18.5
Optimal % (Phase 1): 100%

Trello:



https://trello.com/b/IG519nTW/rl-crop-irrigation

References:

https://www.apsim.info

https://apsimnextgeneration.netlify.app

https://apsimnextgeneration.netlify.app/usage/server/

https://apsimnextgeneration.netlify.app/usage/server/

https://trello.com

rol.pdf

https://sites.ucmerced.edu/files/wdu/files/ipsn_2022_drlic_deep_reinforcement_learning_for_irrigation_cont

https://www.sciencedirect.com/science/article/pii/S0378377422000270

https://github.com/WUR-AI/crop-gym?tab=readme-ov-file

https://arxiv.org/pdf/2104.04326

https://www.mdpi.com/2227-7390/13/4/595

https://github.com/alkaffulm/aquacropgymnasium?tab=readme-ov-file