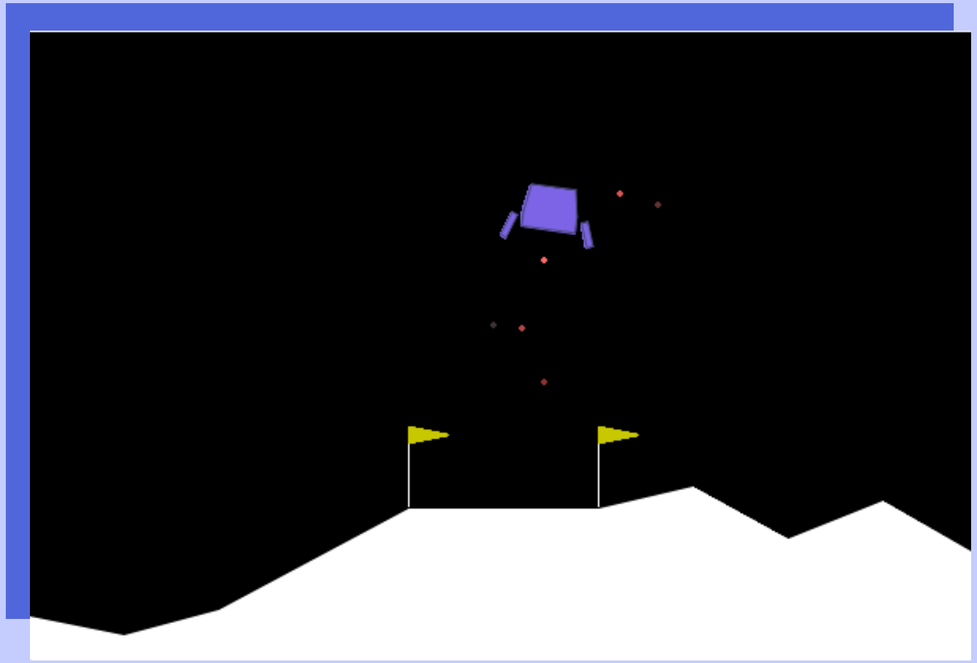

Lunar
Lander
Press Start!

Agenda >>

1. Problem Statement
2. Reinforcement Learning & Environment
3. Trained & Deployed RL Algorithms
4. Results
5. Conclusion

Problem Statement



1. Solve the LunarLander-v3 environment using Reinforcement Learning.
2. Teach an agent to land the lunar module safely between two flags on uneven terrain.

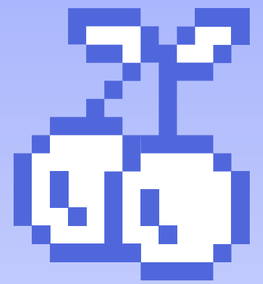
Understanding Reinforcement Learning

Core Concepts

1. State
2. Action
3. Reward

Agent Goals

1. Policy
2. Value Functions

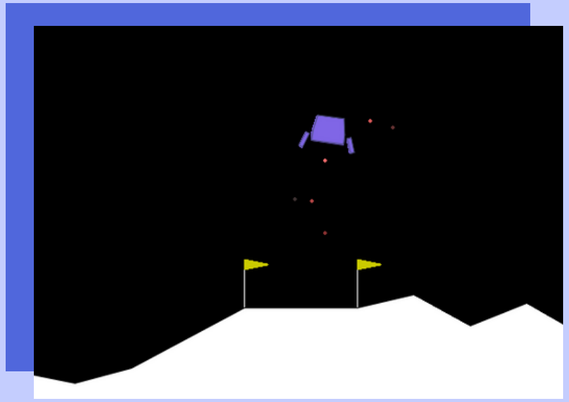


Interaction Loop

The agent observes the state → takes an action → receives a reward → updates its policy.

State

- Represents the current situation of the environment that the agent can observe.
- The agent relies solely on the state to decide its actions.
- A well-defined state space ensures the agent has enough information to learn an optimal policy.

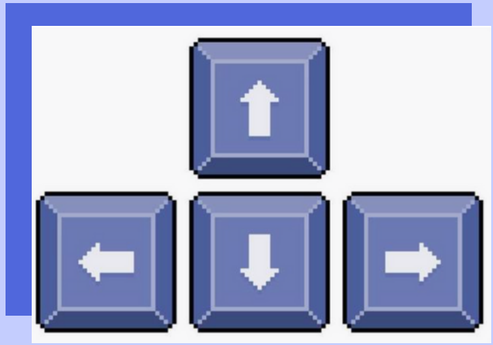


For Lunar Lander, it includes information like the lander's position (x, y), velocity, angle, angular velocity, and whether the legs are touching the ground.

```
//example
[x_position,
 y_position,
 x_velocity,
 y_velocity,
 angle,
 angular_velocity,
 leg_contact_left,
 leg_contact_right]
```

Action

- The set of choices available to the agent to interact with.
- The action chosen directly influences the state transition, subsequent rewards, and is critical for achieving the task's objective.



For LunarLander-v3: Discrete(4) action space, representing:

- 0: Do nothing
- 1: Fire left engine
- 2: Fire main engine
- 3: Fire right engine

```
# example of the action space
action_space = {
    0: "Do nothing",

    1: "Fire left engine",

    2: "Fire main engine",

    3: "Fire right engine"
}
```

Reward

- Represents the feedback signal received by the agent for its actions in the environment.

- Rewards drive the agent's learning process by indicating the quality of its actions, with properly structured rewards enabling faster learning and better performance.

Reward Structure in LunarLander-v3:

1. Positive Rewards:
   a. Successfully landing between the flags: +100 to +200 points.
   b. Moving closer to the target pad.
2. Negative Rewards:
   a. Crashing the lander is -100 points.
   b. Penalty proportional to the amount of fuel burned.
   c. Drifting away from the landing pad.

- Smooth landings receive higher rewards compared to abrupt or unstable descents.

Policy

- Represents the agent’s strategy or decision-making process for choosing actions based on the observed state.
- Deterministic vs. Stochastic:
  - A deterministic policy outputs a specific action for a given state.
  - A stochastic policy outputs a probability distribution over actions, allowing for exploration.
- Role in RL: Learning an optimal policy means the agent consistently chooses actions that maximize rewards.
- Example for LunarLander-v3: At a high altitude with a high descent speed, the policy might choose to fire the main engine (action 2) to slow down.

Value Functions

- Estimates the expected future rewards the agent can achieve from a given state (or state-action pair).

- There are two primary types of value functions:
  - State Value Function (V(s)):
    - Measures the expected cumulative reward from a given state sss if the agent follows a specific policy.
    - Example: "How good is it to be at this position above the landing pad?"
  - Action Value Function (Q(s,a)):
    - Measures the expected cumulative reward from taking an action ‘a’ in state ‘s’, then following a specific policy.
    - Example: "If I fire the left engine now, what is the long-term impact on rewards?"

Interaction Loop

Step 1: Observe State

- The agent perceives the current state of the environment, which contains all necessary information (e.g., position, velocity, and contact status in LunarLander-v3).

Step 2: Take Action

- Based on the observed state, the agent selects an action using its policy (e.g., firing the main engine or doing nothing).

Step 3: Receive Reward

- The environment provides feedback (positive or negative) based on the quality of the action taken.

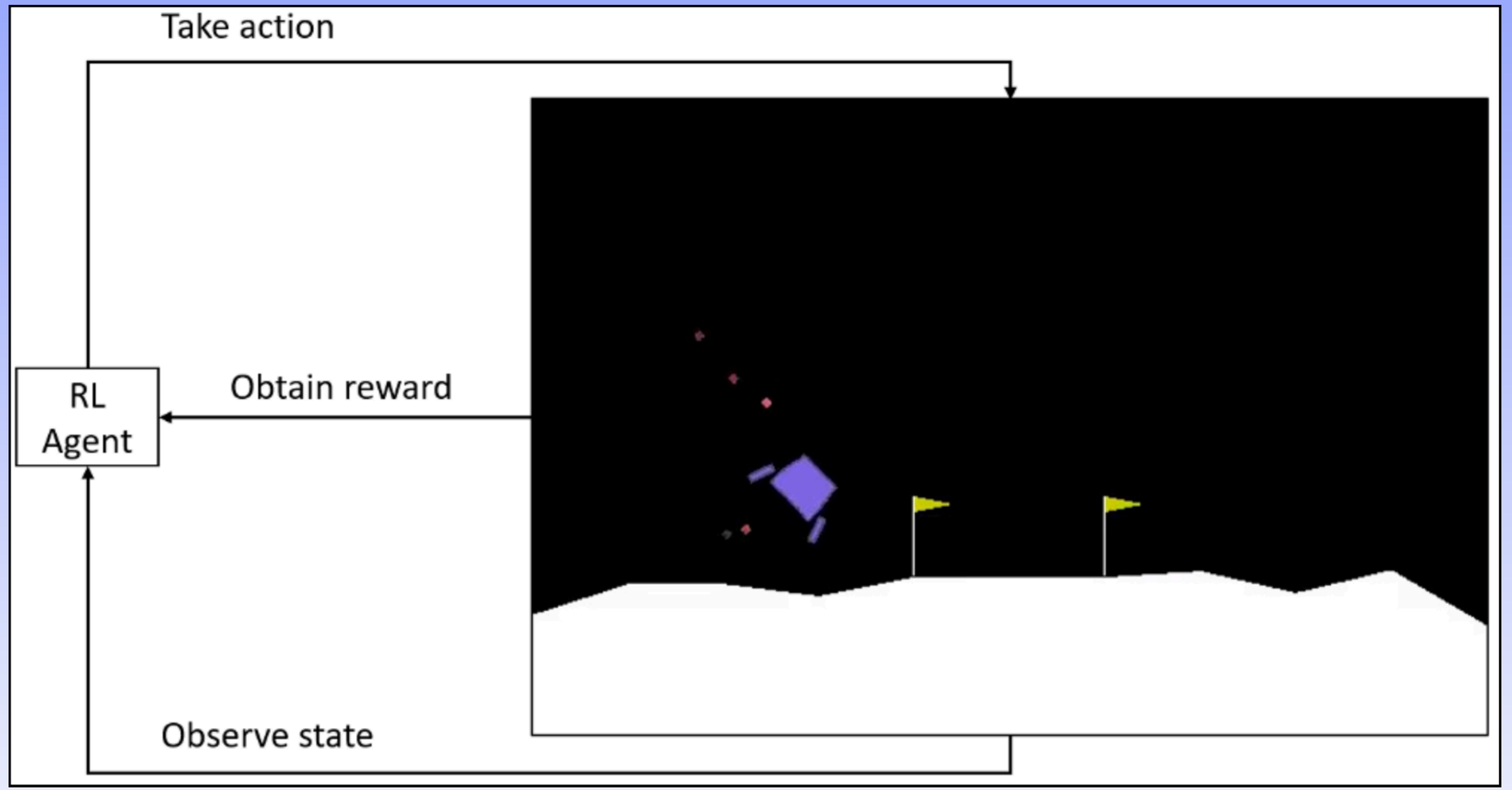
Step 4: Transition to Next State

- The agent enters a new state due to the effects of its action and the environment's dynamics.

Step 5: Update Policy

- Using the reward and new state, the agent improves its policy to increase the likelihood of selecting better actions in the future.

REINFORCE Algorithim



REINFORCE Algorithm

- A policy-gradient method that directly optimizes the policy to maximize cumulative rewards.

- Steps:
  - Collect Trajectory:
    - Run the policy for an episode, storing states, actions, and rewards.
  - Compute Returns:
    - Calculate the total discounted rewards from each timestep

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$$

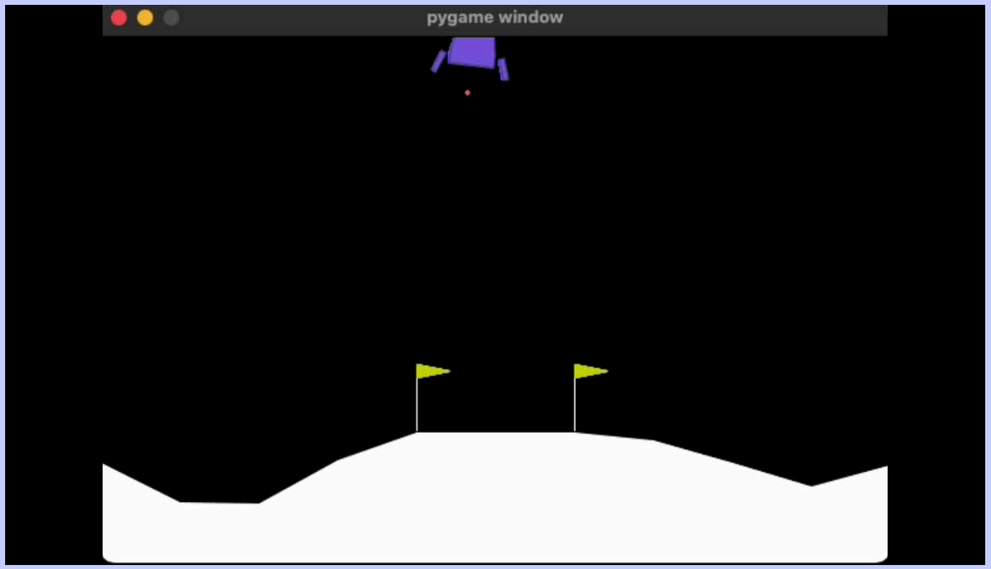
  - Policy Update:
    - Use gradient ascent to adjust the policy parameters

$$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(a_t|s_t) G_t$$

  - Repeat:
    - Update the policy over multiple episodes.

REINFORCE Algorithm

- Advantages:
  - Simple and intuitive.
  - Directly optimizes the policy.
- Challenges:
  - High variance in gradients.
  - Slow convergence.

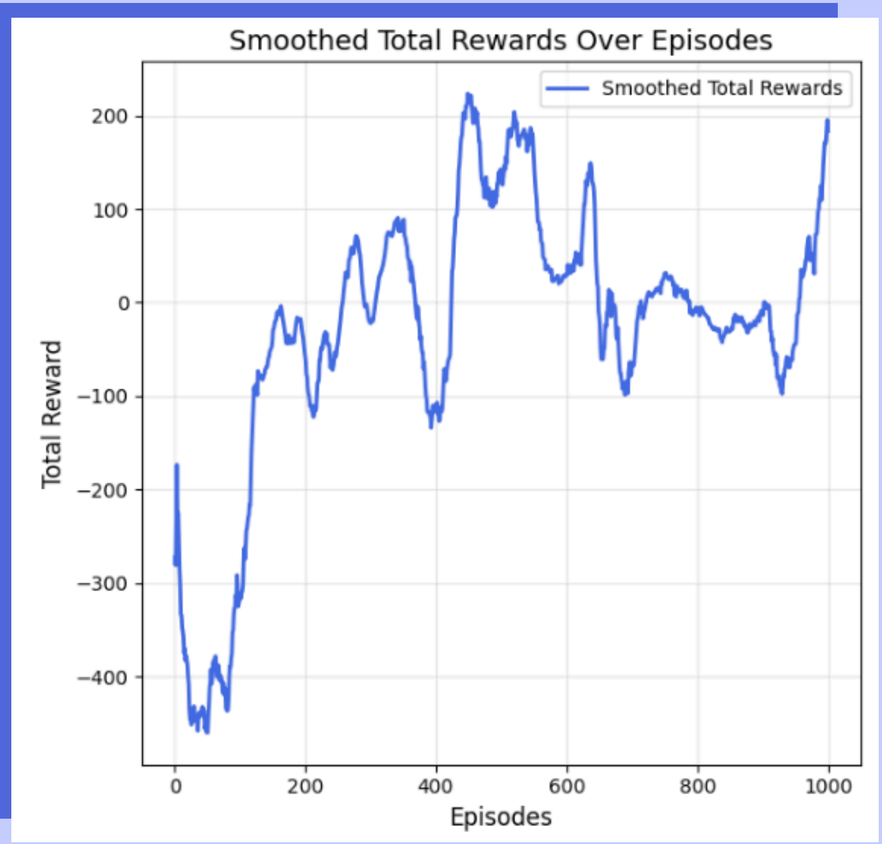


Untrained Environment

REINFORCE Trained Environment
(1000 Episodes)

REINFORCE Algorithm

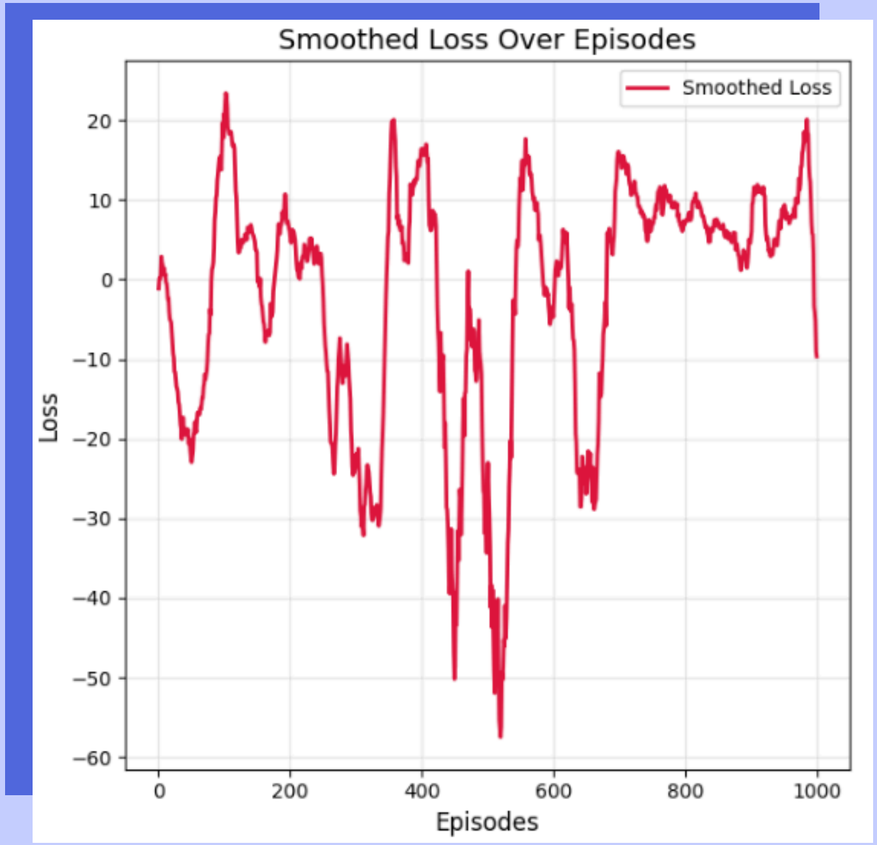
Reward Analysis



1. Initially, the agent performs poorly, receiving large negative rewards due to random actions and crashes.
2. Around episode 400–600, the rewards peak as the agent successfully lands the lunar module consistently.
3. The slight drop in performance toward the middle indicates some instability, but rewards stabilize again towards the end, demonstrating the agent's progress.

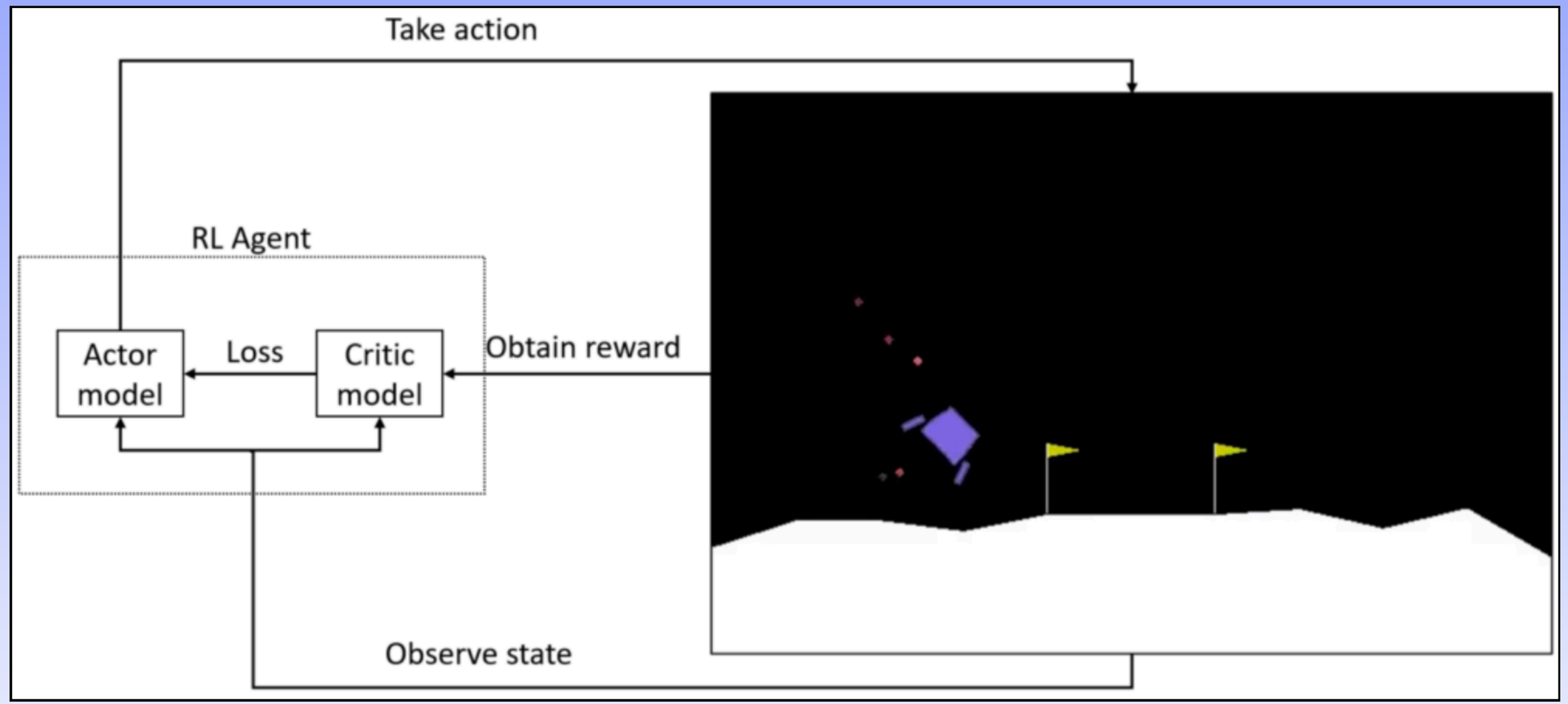
REINFORCE Algorithm

Loss Analysis



1. Initially, the loss fluctuates as the agent explores and updates its policy based on random actions.
2. Around episodes 400–600, the loss shows sharp dips, likely corresponding to significant updates when the agent learns better policies.
3. Toward the end, the loss stabilizes near 0, indicating the agent is approaching convergence, as it has learned a policy that minimizes the REINFORCE loss function.

Actor-Critic Algorithim



Actor-Critic Algorithm

- Combines policy-based (actor) and value-based (critic) methods. The actor decides the actions, and the critic evaluates the actions.

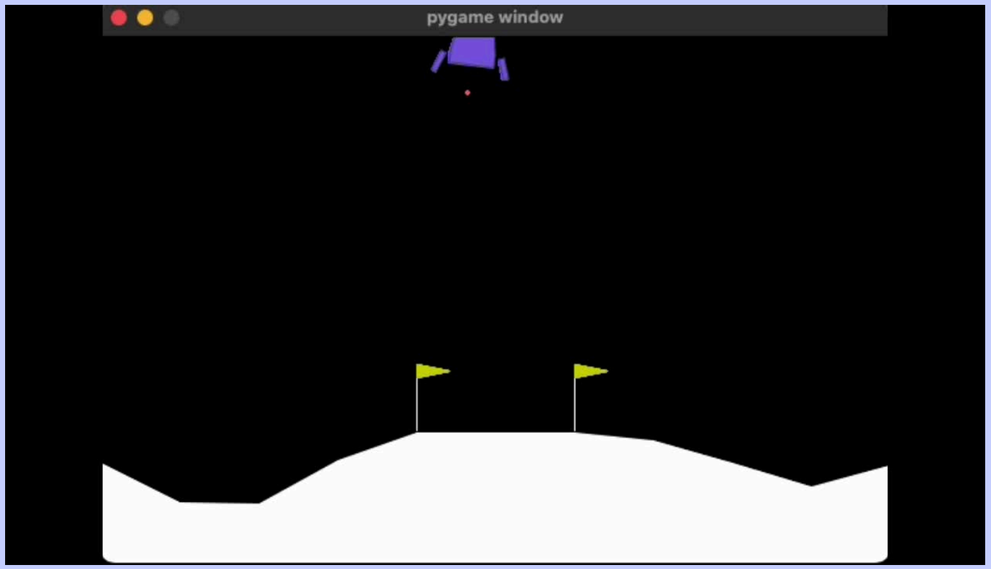
- Key Features:
  - Actor: Learns the policy to select actions.
  - Critic: Estimates the value function V(s) to provide feedback on the actor's actions.
  - Advantage Function: Improves learning stability by reducing variance

$$A(s,a) = Q(s,a) - V(s)$$

- Steps:
  a. Initialize Networks:
     - Actor network (policy) and critic network (value).
  b. Collect Trajectory:
     - Run the current policy to collect states, actions, and rewards.
  c. Critic Update:
     - Minimize the temporal difference
  d. Actor Update:
     - Update the policy using advantage estimates
  e. Repeat:
     - Alternate updates for actor and critic networks.

Actor-Critic Algorithm

- Advantages:
  - Reduces variance compared to policy-only methods like REINFORCE.
  - More sample-efficient than purely value-based methods.
- Challenges:
  - Less stable than off-policy algorithms.
  - Sensitive to hyperparameter tuning.

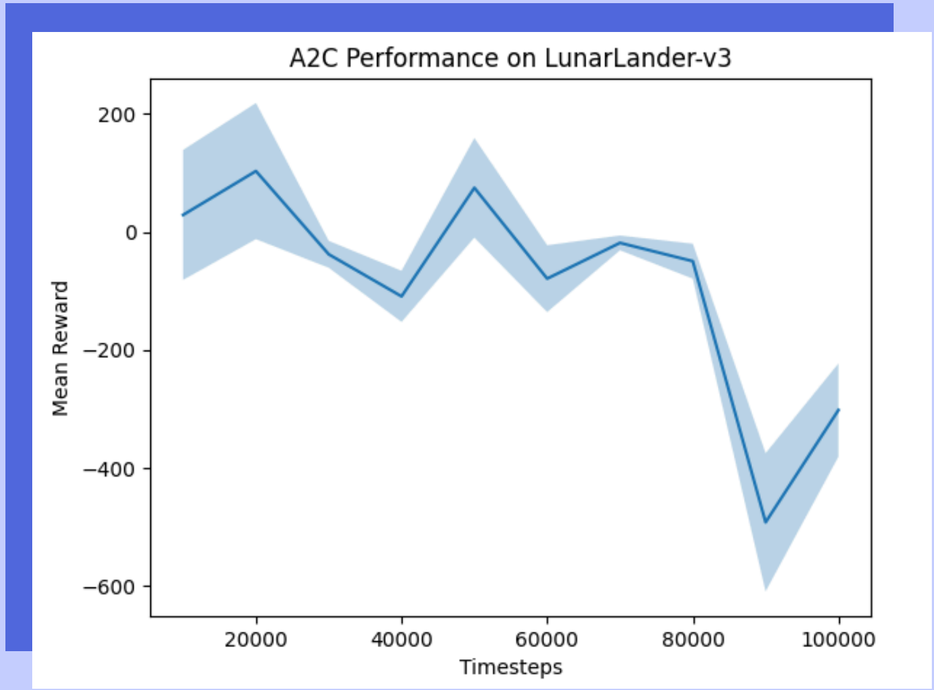


Untrained Environment

A2C Trained Environment
(100000 Timesteps)

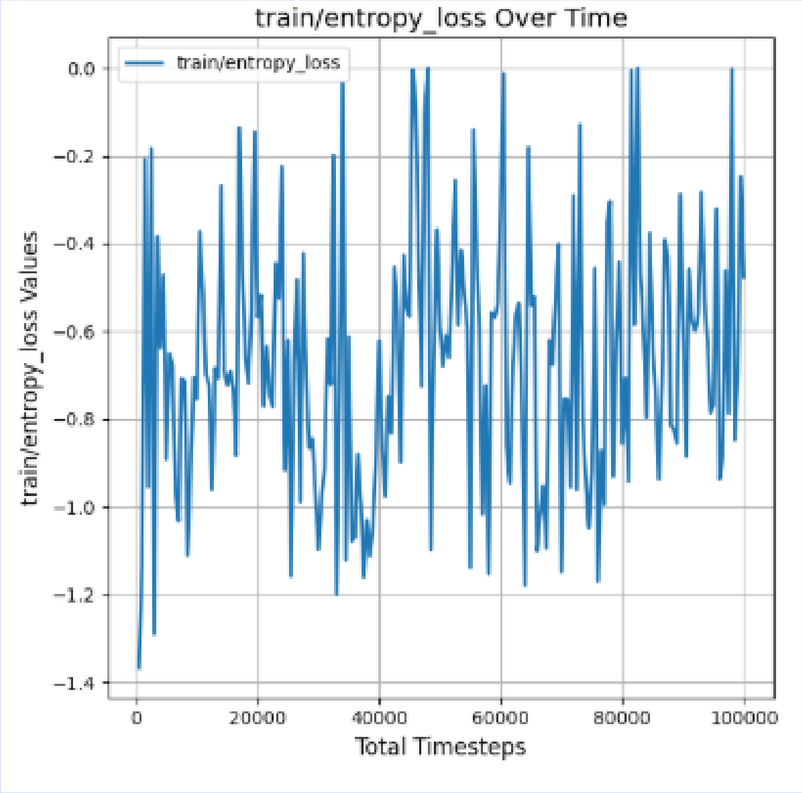
A2C Algorithm

Reward Analysis



- Initial Improvement (0–20,000 Timesteps): The agent starts with negative rewards but shows steady improvement, reaching rewards close to 200 as it learns basic strategies.
- Mid-Training Variance (20,000–60,000 Timesteps):Rewards fluctuate as the agent refines its policy, with increased variability due to exploration and updates.
- Performance Drop (80,000–90,000 Timesteps): A sharp decline occurs, likely due to training instability or suboptimal updates, with high variability in performance.
- Recovery (90,000–100,000 Timesteps): The agent recovers and shows improvement, but performance remains unstable, indicating the need for further training or fine-tuning.

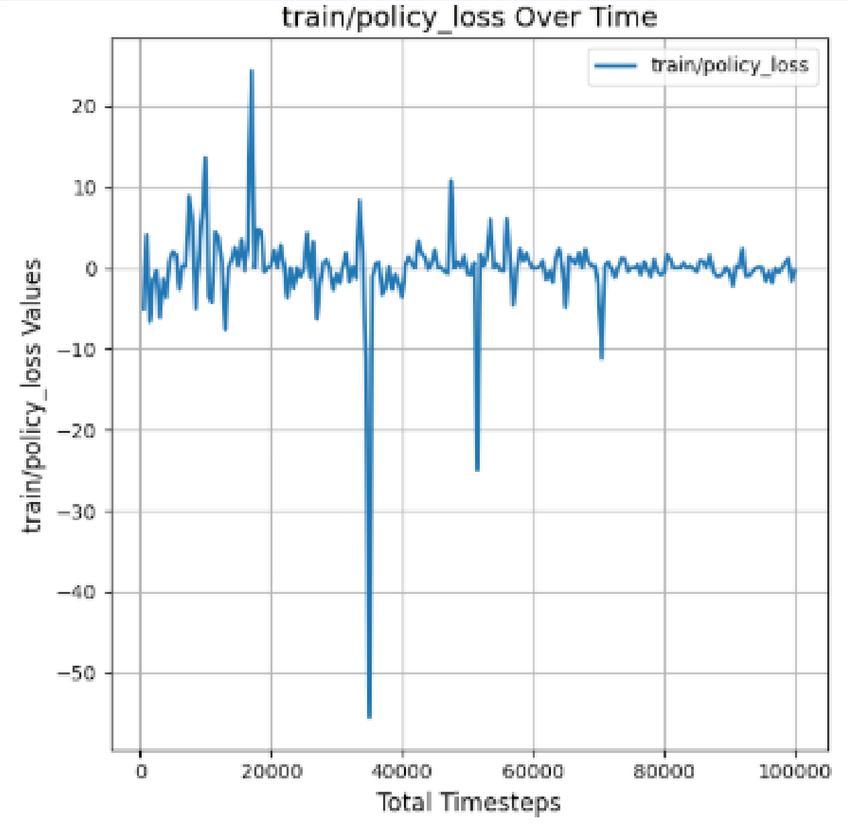
Entropy Loss Analysis



- Entropy Loss
  - The entropy loss fluctuates significantly, indicating the policy is still exploring.
  - While some fluctuation is expected early on, continued instability suggests the agent hasn't fully converged to a deterministic policy.
  - It indicates an insufficient training period or suboptimal hyperparameters.

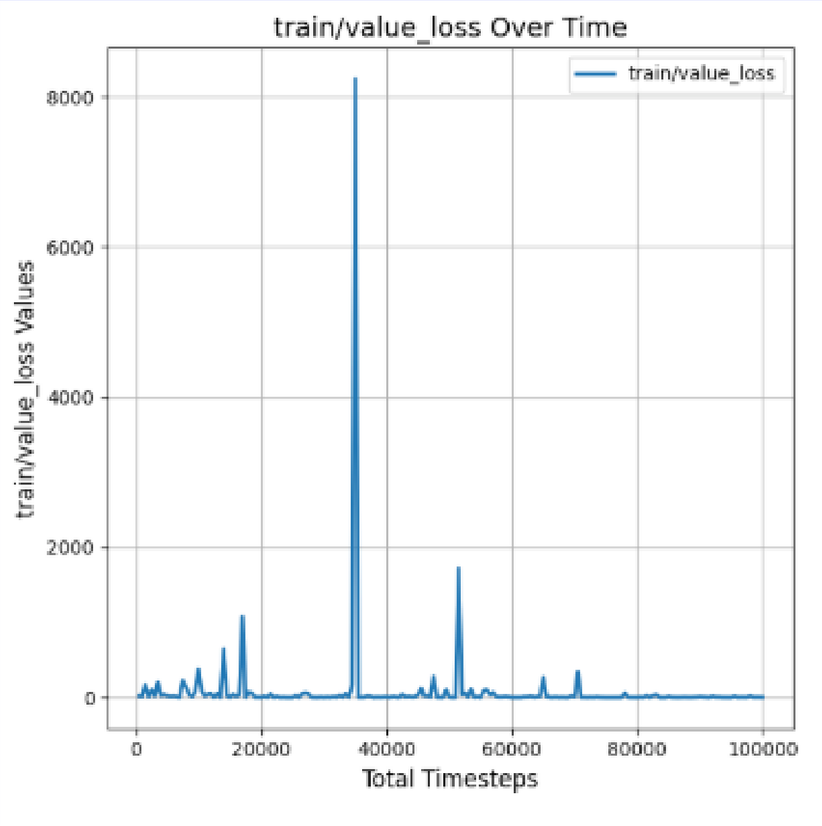
A2C Algorithm

Policy Loss Analysis



- Policy Loss
  - Sharp drops around 40,000 timesteps suggest sudden updates caused by large gradients or reward changes.
  - The loss stabilizes toward the end, indicating the policy is improving and exploration is decreasing.

Value Loss Analysis



- Value Loss
  - Large spikes, particularly around 40,000 timesteps, indicate difficulties in predicting returns for some states.
  - The value loss decreases over time, showing the value network has improved its estimation of expected rewards.

Conclusion

Stability:

- REINFORCE: Shows instability early on but becomes more stable toward the end as the policy converges.
- A2C: Demonstrates greater stability from the start, thanks to the critic providing more consistent feedback.

Learning Speed:

- REINFORCE: Slower convergence due to high variance in gradient estimates and reliance on full episode returns.
- A2C: Faster learning as the critic minimizes temporal difference errors, making updates more sample-efficient.

Exploration and Variance:

- REINFORCE: Entropy loss fluctuates significantly, indicating prolonged exploration and potential instability.
- A2C: Balances exploration and exploitation better, with smaller fluctuations in policy and value loss.

GAME OVER