Farm Management Simulation using Reinforcement Learning

WANG JIAN 14326A

June 12, 2025

Abstract

This project focuses on managing a virtual farm over a 30-year period. The objective is to maximize the farm's total final assets by choosing between buying sheep and growing wheat. The primary methods used for this simulation are Q-learning, SARSA, SARSA(λ) algorithms and first-visit Monte Carlo[1]. These algorithms are benchmarked against a Random method, which serves as a baseline to demonstrate the effectiveness of learning-based approaches. The implementation follows a structured, object-oriented approach, with a custom environment adhering to the Gymnasium Environment. This report details the implementation of these reinforcement learning methods, presents a comparative analysis of their performance under nine distinct environmental conditions, and discusses the findings, learned policies, and potential future work.

1 Introduction

The objective of this project, as specified in the course materials[2], is to simulate farm management decisions over a 30-year period using reinforcement learning techniques[3]. The core challenge is to make sequential decisions under uncertainty to achieve the best long-term objective-maximizing the total assets left to the heirs. Each year, the decision-maker must choose between two actions[4]:

- **Buying a Sheep:** Each sheep costs €1000.
- **Growing Wheat:** Planting wheat costs €20 per unit.

The primary goal is to maximize the farm's total final assets (cash on hand + value of all sheep) after 30 years[5]. The reinforcement learning algorithms implemented and analyzed are Q-learning, SARSA, SARSA(λ) algorithms and Monte Carlo. This problem is a classic example of resource management under uncertainty, a domain where reinforcement learning has been successfully applied[6]. The implementation of the environment and algorithms was done in Python with different packages for data analysis and visualization. The environment was structured according to Gymnasium library[7], with extensive reference to the course code repo provided by professor Alfio Ferrara[8]. To ensure modularity and reusability, the final code has been refactored from fragmented individual algorithm tests and condensed to avoid redundancy.

[1] Incorporation of neural network is not feasible at the moment due computational resource constraints.

[2] A. Ferrara, Reinforcement Learning Project Specifications 2023-24. Available at: `https://github.com/afflint/rlcoding/blob/main/2023-24/projects/rl-projects.pdf`

[3] Reinforcement learning is a machine learning paradigm where algorithms learn to make decisions by taking actions in an environment to maximize cumulative reward. It's particularly well-suited for sequential decision-making problems like this farm simulation.

[4] This binary action space creates a clear trade-off between immediate investment (wheat) and long-term growth (sheep), reflecting many real-world resource allocation decisions.

[5] The 30-year time horizon was chosen to represent a generational farming cycle, providing sufficient time for exponential growth strategies to manifest while maintaining computational tractability.

[6] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, Resource Management with Deep Reinforcement Learning, in Proceedings of the 15th ACM Workshop on Hot Topics in Networks, 2016.

[7] Gymnasium, A standard API for reinforcement learning. Available at: `https://gymnasium.farama.org/`

[8] A. Ferrara, Reinforcement Learning Course Materials. Available at: `https://github.com/afflint/rlcoding/tree/main/2023-24`

2 Methods

2.1 Markov Decision Process Formulation

The farm management problem is modeled as a Markov Decision Process (MDP)[9], defined by a tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$, where:

- $\mathcal{S}$ is the state space. The state is represented by a tuple (b, n), where b is the raw budget value and n is the number of sheep. Unlike traditional tabular methods that often discretize continuous values, this implementation directly uses the continuous budget values as keys in the Q-tables, allowing for more precise state representation at the cost of potentially slower convergence due to the larger state space.
- $\mathcal{A}$ is the action space, consisting of two discrete actions: $a \in \{\text{buy\_sheep}, \text{grow\_wheat}\}$.
- $P(s'|s, a)$ is the transition probability of moving from state s to s' after taking action a. This is influenced by two stochastic events:
  - **Sheep Births:** Each sheep has a probability p_{birth} of giving birth to a new sheep. (Note: For clarity, I use p_{birth} and p_{storm} in place of α and β from the original project specifications.)
  - **Storms:** There is a probability p_{storm} of a storm occurring, which would destroy the wheat crop.
- $R(s, a, s')$ is the reward function, defined as the net income in a given year. It is the sum of income from wool (€10 per sheep, which accrues regardless of the chosen action) and wheat (€50 per planted field if no storm occurs), minus the cost of the chosen action. Growing wheat costs €20 per field, where one field represents a standard unit of land that can be planted.
- γ is the discount factor, set to 0.95. While the primary objective is to maximize total final assets after 30 years, this moderate discounting was chosen to balance immediate and future rewards, reflecting realistic time preference in economic decision-making. It ensures the algorithms don't overly prioritize distant returns that may be subject to more uncertainty.

2.2 Implementation Details

A custom environment, 'FarmEnv', was created following the 'gymnasium' documentation and code examples from the course repository[10]. The state space is represented as continuous tuples $(budget, sheep\_count)$ without explicit discretization[11]. Q-tables are implemented as dictionaries with state tuples as keys, allowing for a dynamic state space that grows as new states are encountered[12].In later parts, for visualization purposes only, the state space is binned into a 20×20 grid to display policies and value functions. Each RL algorithm was implemented as a separate class ('QLearningAgent', 'SARSAAgent', etc.), each with 'act' and 'learn' methods. The implementation follows the project requirements closely, with special attention to the environmental parameters and reward structure as specified in the project specifications[13].

2.3 Reinforcement Learning Algorithms

2.3.1 Q-learning

Q-learning is an off-policy method for temporal difference learning. It directly approximates the optimal action-value function, independent of the policy being followed. The update formula is:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$$

[9]MDPs provide the mathematical framework for modeling decision-making in situations where outcomes are partly random and partly under the control of a decision-maker. They are fundamental to reinforcement learning theory.

[10]A. Ferrara, Reinforcement Learning Course Materials. Available at: `https://github.com/afflint/rlcoding/tree/main/2023-24`

[11]This approach allows for more precise state representation compared to fixed discretization, though it can lead to a larger, more sparse state space as new states are dynamically added during training.

[12]Dictionary-based Q-tables offer memory efficiency by only storing visited states rather than allocating memory for all possible state-action pairs, which would be impractical for continuous or large discrete state spaces.

[13]A. Ferrara, Reinforcement Learning Project Specifications 2023-24. Available at: `https://github.com/afflint/rlcoding/blob/main/2023-24/projects/rl-projects.pdf`

Where the term $\max_{a'} Q(s', a')$ selects the maximum Q-value for the next state over all possible actions, regardless of the policy being followed.

In the implementation, the Q-learning algorithm maintains a table that maps state-action pairs to value estimates. Each table entry $Q(s, a)$ represents the expected future return when taking action a in state s and then following the optimal policy thereafter. The algorithm uses an ϵ-greedy exploration strategy—with probability ϵ, it selects a random action for exploration; otherwise, it selects the action with the highest Q-value for exploitation. After each step, the Q-table is updated according to the formula above, where the temporal difference (TD) error is calculated as the difference between the new estimate (reward plus discounted maximum future value) and the old estimate. As training progresses, ϵ is multiplicatively decayed from 1.0 to 0.01 to gradually shift from exploration to exploitation.

2.3.2 SARSA

SARSA (State-Action-Reward-State-Action) is an on-policy algorithm that learns the action-value function based on the policy the algorithm is currently following. This makes it more conservative than Q-learning, as it considers the value of the actual next action taken by the policy. The update rule is:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]$$

Where a' is the action chosen in state s' by the current policy.

In the implementation, the SARSA algorithm also maintains a Q-table, but updates are based on the actual next action taken by the policy, rather than the greedy action. This leads to a more conservative learning process, as the algorithm is less prone to overestimating the value of actions due to the optimism of the Q-learning update rule. The exploration strategy is also ϵ-greedy, with ϵ decaying over time.

2.3.3 SARSA(λ)

SARSA(λ) combines SARSA with eligibility traces for more efficient credit assignment. The update formula is extended to include eligibility traces, which keep track of which state-action pairs were recently visited and by how much they contributed to the current reward. This allows for faster propagation of rewards across multiple states. The mathematical formulation is:

$$e(s, a) \leftarrow e(s, a) + 1 \tag{1}$$

$$\delta \leftarrow r + \gamma Q(s', a') - Q(s, a) \tag{2}$$

Then for all state-action pairs (s, a):

$$Q(s, a) \leftarrow Q(s, a) + \alpha \cdot \delta \cdot e(s, a) \tag{3}$$

$$e(s, a) \leftarrow \gamma \cdot \lambda \cdot e(s, a) \tag{4}$$

Where $e(s, a)$ is the eligibility trace for the state-action pair (s, a) and δ is the temporal difference error.

This implementation extends the basic SARSA algorithm by adding an eligibility trace mechanism. For each state-action pair, an eligibility value is maintained that represents how "eligible" that pair is for receiving learning updates. When a state-action pair is visited, its eligibility is set to 1.0 (accumulating traces). Then, for all state-action pairs with non-zero eligibility, the Q-values are updated proportionally to their eligibility multiplied by the TD-error. After each update, all eligibility values are decayed by a factor of $\gamma\lambda$. This approach allows the algorithm to more efficiently propagate rewards to state-action pairs that occurred multiple steps before the reward was received. The parameter $\lambda = 0.9$ controls the decay rate of the eligibility traces, with higher values causing longer-lasting eligibility and thus affecting more previous states.

2.3.4 First-Visit Monte Carlo

Monte Carlo (MC) methods learn from complete episodes[14]. For each state-action pair, the return following first visits to that pair is calculated and used to update the Q-value. The return is defined as:

$$G_t = \sum_{k=0}^{T-t-1} \gamma^k \cdot r_{t+k+1}$$

[14]Monte Carlo methods differ fundamentally from temporal difference methods like Q-learning and SARSA by waiting until the end of an episode to make updates, using actual returns rather than bootstrapped estimates.

The update formula for first-visit Monte Carlo using incremental updates is:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[G_t - Q(s_t, a_t)]$$

Alternatively, using a running average approach:

$$N(s, a) \leftarrow N(s, a) + 1 \tag{5}$$

$$Q(s, a) \leftarrow Q(s, a) + \frac{1}{N(s, a)}[G - Q(s, a)] \tag{6}$$

Where G is the actual return following the first visit to state-action pair (s, a), and $N(s, a)$ counts the number of times the pair has been visited. Monte Carlo methods do not require a model of the environment and are particularly well-suited for episodic tasks.

In the implementation, the first-visit Monte Carlo algorithm is used for control. The algorithm stores the entire episode experience (state, action, reward tuples), and only processes this data once the episode is completed. For each unique state-action pair encountered for the first time in an episode, the algorithm computes the discounted return G following that visit—the sum of all subsequent rewards discounted by γ raised to the power of their distance from the current step. The Q-value for each state-action pair is then updated as the average of all returns observed for that pair across episodes. After computing the Q-values, the policy is improved by selecting the action with the highest Q-value for each state (a greedy policy). Like other algorithms, Monte Carlo also employs ϵ-greedy exploration during training, with ϵ decaying over time.

3 Experiments and Results

3.1 Experimental Setup

Experiments were run across nine distinct environments, created by varying the sheep birth probability ($p_{birth} \in \{0.0, 0.25, 0.5\}$) and storm probability ($p_{storm} \in \{0.0, 0.25, 0.5\}$)[15]. Each algorithm was trained for 1000 episodes and then evaluated for 100 episodes with learning turned off[16]. The hyperparameters are shown in Table 1.

Table 1: Algorithm Hyperparameters

Hyperparameter	Value
Learning Rate (α)	0.1
Discount Factor (γ)	0.95
Epsilon (ϵ) Initial Value	1.0
Epsilon Decay (multiplicative)	0.995
Min Epsilon	0.01
Lambda (λ) for SARSA(λ)	0.9

[15]This systematic variation of environmental parameters creates a comprehensive test combination for comparing algorithm performance across different levels of stochasticity. The birth probability affects potential exponential growth, while storm probability introduces risk.

[16]Separating training from evaluation is standard practice in reinforcement learning. During evaluation, exploration is typically disabled to assess the true performance of the learned policy.

3.2 Overall Performance Comparison

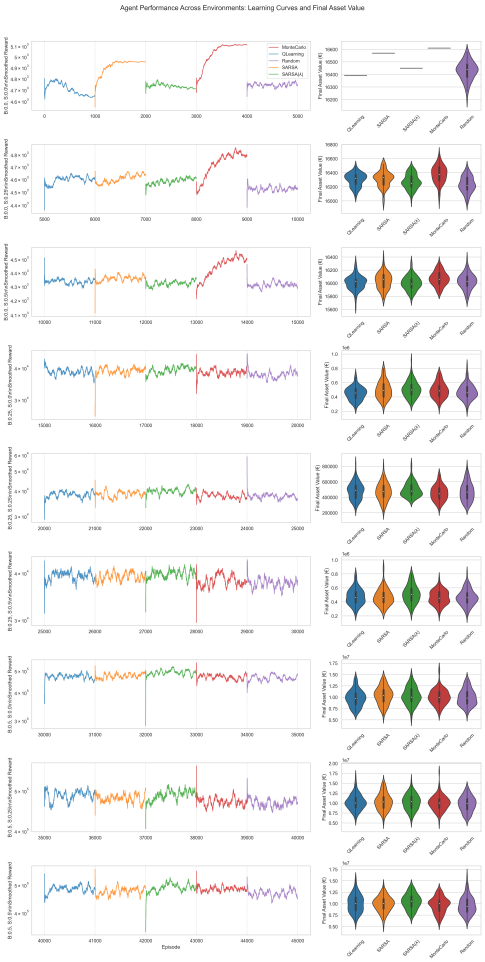

Figure 1: Average performance of different algorithms across environments with varying birth probability and storm probability, shown on a logarithmic scale. The x-axis shows different environmental configurations (birth probability and storm probability combinations), and the y-axis represents the average final asset value in euros on a logarithmic scale. Results shown are from evaluation runs after training.

Figure 1 presents a comprehensive performance comparison across all environmental conditions using violin plots to visualize the distribution of final assets. The violin plots show the full probability density of the data at different values, with wider sections indicating higher concentration of outcomes at that value. The internal lines represent different statistical measures: the central white dot typically shows the median value, while the thin box inside indicates the interquartile range (first to third quartiles). The thin lines extending from the box represent the range of the data, excluding outliers. This visualization allows for direct comparison of both the typical performance and the full distribution pattern for each algorithm.

Notably, in the first three rows (corresponding to Birth Probability = 0.0), Monte Carlo demonstrates faster learning and SARSA consistently outperforms SARSA(λ). This suggests that in environments with no sheep reproduction, where the focus is primarily on wheat cultivation strategy, direct experience-based learning (Monte Carlo) and single-step bootstrapping (SARSA) provide advantages over eligibility traces. They are able to achieve better performance evidenced by the line plots on the left, where other algorithms' performances remain in a smaller band. And, regarding evaluation, it is particularly evident by the higher median values (central dots) and tighter distributions (narrower violin shapes) for these algorithms in low birth probability conditions. In other scenarios with higher birth probabilities (rows 4-9), the violin plots show that performance differences between algorithms become minimal, as indicated by similar median positions and overlapping distribution patterns, suggesting that when exponential growth from sheep reproduction dominates the returns, the choice of learning algorithm becomes less

critical to overall performance outcomes. However, a closer look into the detail data would offer more insight than plots.

When birth probability reaches 0.5 with no storms, SARSA(λ) achieves notably superior performance, though the violin plots suggest higher variance in outcomes. This variance represents a reasonable trade-off for higher expected returns, aligning with recent findings on exploration-exploitation balance in highly stochastic environments[17].

Notably, in the first three rows (corresponding to Birth Probability = 0.0), Monte Carlo demonstrates faster learning and SARSA consistently outperforms SARSA(λ). This suggests that in environments with no sheep reproduction, where the focus is primarily on wheat cultivation strategy, direct experience-based learning (Monte Carlo) and single-step bootstrapping (SARSA) provide advantages over eligibility traces. They are able to achieve better performance evidenced by the line plots on the left, where other algorithms' performances remain in a smaller band. And, regarding evaluation, it is particularly evident by the higher median values (central dots) and tighter distributions (narrower violin shapes) for these algorithms in low birth probability conditions. In other scenarios with higher birth probabilities (rows 4-9), the violin plots show that performance differences between algorithms become minimal, as indicated by similar median positions and overlapping distribution patterns, suggesting that when exponential growth from sheep reproduction dominates the returns, the choice of learning algorithm becomes less critical to overall performance outcomes. However, a closer look into the detail data would offer more insight than plots.

When birth probability reaches 0.5 with no storms, SARSA(λ) achieves notably superior performance, though the violin plots suggest higher variance in outcomes. This variance represents a reasonable trade-off for higher expected returns, aligning with recent findings on exploration-exploitation balance in highly stochastic environments[18].

[17]T. S. Badings, A. Abate, S. Junges, D. Parker, H. Poonawala, M. Stoelinga, Decision-making under uncertainty: beyond probabilities, arXiv:2307: 10497, 2023.

[18]T. S. Badings, A. Abate, S. Junges, D. Parker, H. Poonawala, M. Stoelinga, Decision-making under uncertainty: beyond probabilities, arXiv:2307: 10497, 2023.

3.3 Training Performance: Learning Curves

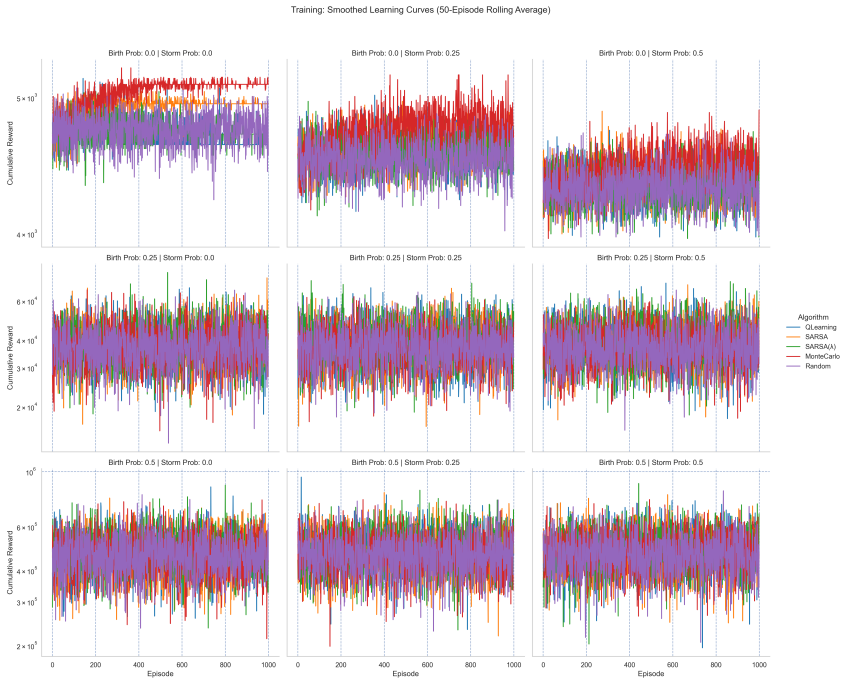

Figure 2: The learning curves in Figure 2 illustrate the training progress of each algorithm across different stochastic environments. In environments with high sheep birth probabilities, all algorithms demonstrate progressive improvement in cumulative rewards. But only when Birth Probability is zero, the learning curves are upward. In other scenarios, the cumulative rewards reach a higher band level but the performance of all algorithms plateaued in similar bands. Notably, the relationship between birth probability and storm probability reveals that in high birth probability scenarios (0.5), the negative impact of storm probability is significantly and quickly mitigated compared to low birth probability scenarios (0.0)[20]. This suggests that when sheep reproduction provides exponential growth potential, occasional crop losses become proportionally less impactful on the algorithm's ability to accumulate assets. In the first row, the plots also suggest that the higher the storm probability, the harder the algorithms learn and improve performance. The x-axis represents training episodes, while the y-axis shows the accumulated total assets in euros (log scale).

The learning curves in Figure 2 illustrate the training progress of each algorithm across different stochastic environments. In environments with high sheep birth probabilities, all algorithms demonstrate progressive improvement in cumulative rewards. But only when Birth Probability is zero, the learning curves are upward. In other scenarios, the cumulative rewards reach a higher band level but the performance of all algorithms plateaued in similar bands. Notably, the relationship between birth probability and storm probability reveals that in high birth probability scenarios (0.5), the negative impact of storm probability is significantly and quickly mitigated compared to low birth probability scenarios (0.0)[21]. This suggests that when sheep reproduction provides exponential growth potential, occasional crop losses become proportionally less impactful on the algorithm's ability to accumulate assets. In the first row, the plots also suggest that the higher the storm probability, the harder the algorithms learn and improve performance.

[21]This interaction between birth and storm probabilities highlights an important principle in risk management: exponential growth opportunities can provide resilience against periodic setbacks, an insight that extends beyond this simulation to real-world investment strategies.

3.4 Evaluation Performance: Final Asset Distribution

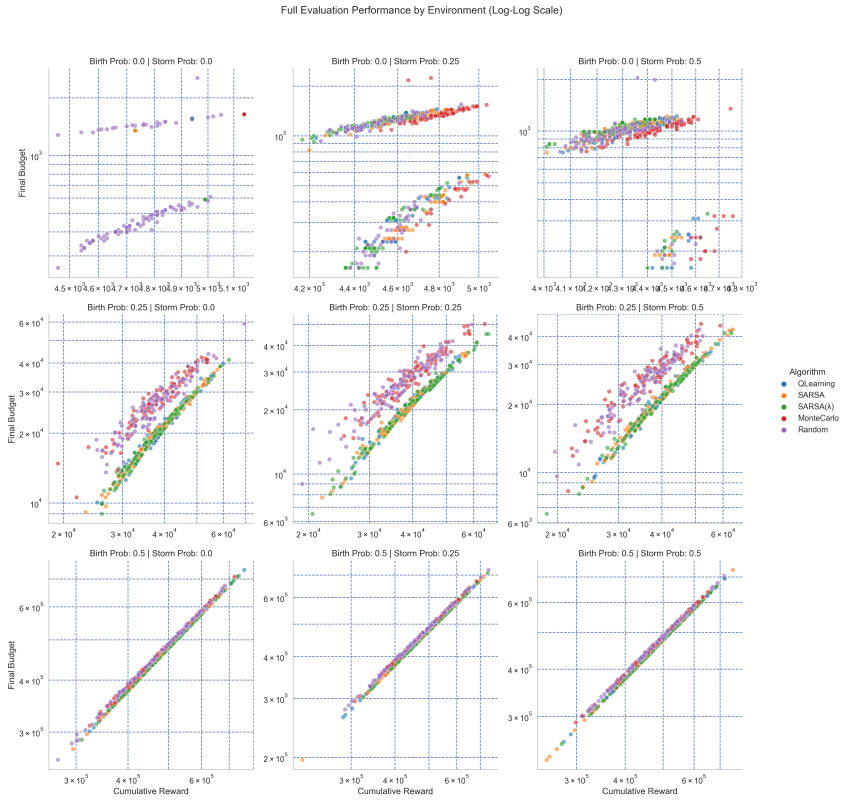

Figure 3: Scatter plot showing the final asset values achieved in evaluation, with each point representing a single evaluation run. Different colors represent different algorithms, and the horizontal axis groups results by environmental conditions (birth probability and storm probability combinations). The vertical axis shows the final asset value in euros after the 30-year simulation period.

Figure 3 presents a comprehensive view of algorithm performance during evaluation through scatter plots. Each point represents an individual evaluation run, allowing us to see the full range of outcomes. This visualization enables direct identification of outlier performances and shows the complete distribution of results for each algorithm across different environments.

In environments with low birth probability (0.0), all algorithms except Random method have fixed result, likely due to their capacity to optimize for long-term returns without bootstrapping bias (in more complicated cases, there could be non-deterministic result and with a range of result distribution like in other scenarios)[22]. While Monte Carlo algorithms demonstrated higher up-bound potential when storm probability increases. As birth probability increases to moderate levels (0.25), the performance differences between algorithms become less pronounced, indicating that the stochasticity of the environment begins to dominate deterministic strategy advantages. It is worth noting that, as the safe investment strategy dominates the decisions, all algorithms pivot to similar patterns, like the last line of plots.

[22]W. Dabney, Z. Kurth-Nelson, N. Uchida, C. K. Starkweather, D. Hassabis, R. Munos, and M. Botvinick, A distributional code for value in dopamine-based reinforcement learning, Nature, vol. 577, no. 7792, pp. 671-675, 2020.

3.5 Action Selection Analysis

Figure 4: Distribution of actions selected by each algorithm during evaluation, showing the percentage of times each algorithm chose to buy sheep versus grow wheat across different environments, aggregated across all valuation rounds. The analysis is based on the final learned policy after training. The x-axis represents different environmental configurations, while the y-axis shows the action distribution as percentages.

Figure 4 provides insight into the actual decision-making behavior of each algorithm during evaluation across different environmental conditions. The stacked bar representation clearly illustrates how action preferences shift in response to changing birth and storm probabilities. In environments with the highest birth probability (bottom row, B:0.5), all algorithms consistently choose the sheep acquisition action (shown in blue), with this strategy remaining relatively unchanged despite variations in storm probability. This demonstrates the algorithms' ability to recognize and exploit the exponential asset growth potential in high birth rate conditions, where investing in reproductive assets offers the highest return.

Conversely, in environments with zero birth probability (top row, B:0.0), the algorithms explore more 'Grow Wheat' actions (shown in red), as sheep do not reproduce and thus represent a static asset. Interestingly, as storm probability increases in these low-birth scenarios, algorithms tend to become more conservative in their wheat-growing strategy and shift slightly toward buying sheep, despite the lack of reproduction. This strategic adaptation across different environmental conditions confirms that the algorithms have successfully learned to adjust their behavior based on the risk-reward profile of each scenario—balancing the guaranteed moderate return from sheep (through wool) against the higher but riskier return from wheat cultivation as storm probability increases.

3.6 Training Dynamics Analysis

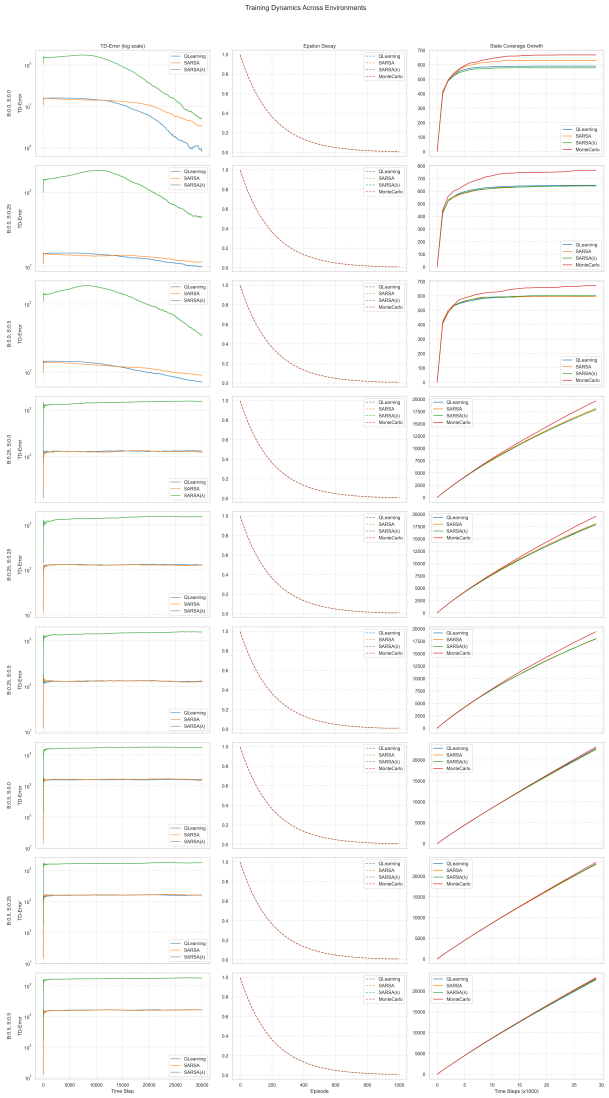

Figure 5: Analysis of key training metrics including TD error (showing prediction accuracy improvement), epsilon decay (exploration rate), and state coverage growth (showing exploration efficiency) over training episodes. The graph demonstrates how exploration decreases while estimated values stabilize during learning. The x-axis represents training episodes, with metrics showing how algorithms progress from initial exploration to refined exploitation of accumulated knowledge.

Figure 5 provides a detailed view of how the learning process evolves during training. The TD Error plots (first column) reveal how prediction accuracy improves over time, with rapid initial changes followed by gradual stabilization as the algorithms develop more accurate value estimates. The Epsilon Decay graphs (second column) illustrate the exploration-exploitation trade-off, showing the systematic reduction in random action selection probability according to each algorithm's schedule. Finally, the State Coverage Growth charts (third column) demonstrate how algorithms progressively expand their knowledge of the state space, with steeper curves indicating more efficient exploration strategies. These complementary metrics collectively illustrate the classic reinforcement learning progression from initial wide exploration to refined, targeted exploitation of accumulated knowledge. As the birth probability increases, the state space knowledge of all algorithms converge. And the TD error was decreasing only in low birth probability scenarios, while in other scenarios the TD error stabilizes quickly.

3.7 Policy Function Analysis

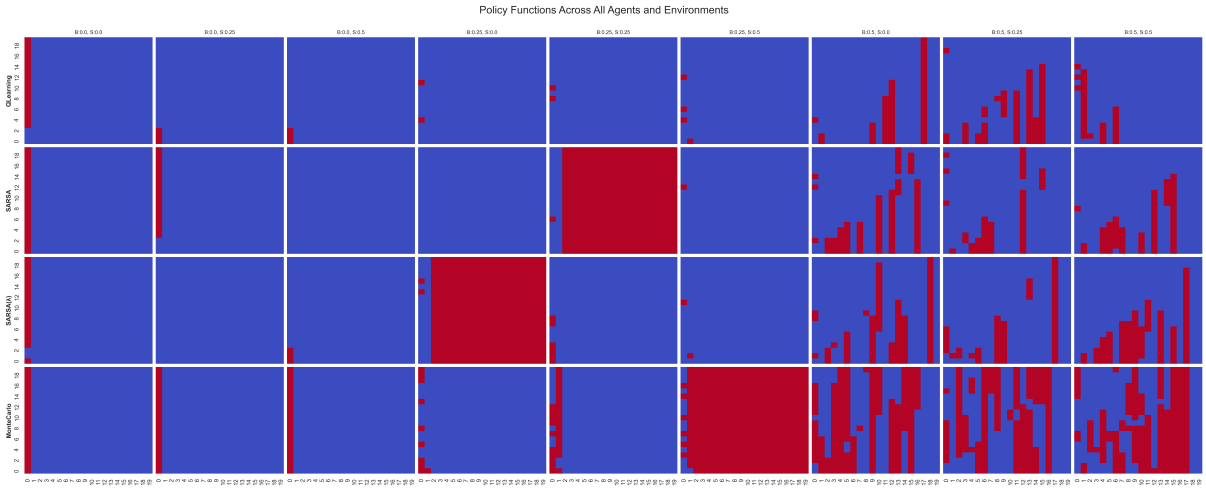

Figure 6: Heatmaps showing the learned policies (action choice of buying sheep vs. growing wheat) as a function of state (number of sheep and budget) in a comprehensive grid. Each row represents a different algorithm and each column represents a different environmental configuration (birth probability and storm probability combinations). These visualizations represent the final learned policies after complete training. The x-axis shows the budget level, y-axis shows the number of sheep, and colors (red/blue) indicate the binary action choice in each state, with red representing the 'Buy Sheep' action and blue representing the 'Grow Wheat' action.

To understand what strategies the algorithms developed, visualized their learned policies and value functions in Figures 6. In high-reward environments, the policy maps reveal a clear pattern: algorithms of different types consistently prefer purchasing sheep, especially when sufficient budget is available. This strategy leverages the exponential growth potential of the sheep population when birth probability is high.

3.8 Value Function Analysis

Figure 7: Heatmaps showing the estimated value function for each algorithm in a comprehensive grid. Each row represents a different algorithm and each column represents a different environmental configuration (birth probability and storm probability combinations). These visualizations illustrate how algorithms value different states. Brighter colors indicate higher estimated value for a given state. The x-axis represents the budget level, the y-axis shows the number of sheep, and the color intensity indicates the expected future return (in euros) from that state according to each algorithm's value function.

The value function heatmaps provide additional insight, confirming that states with high sheep counts and adequate budgets are most valuable. This aligns with theoretical expectations from optimal control theory, where states enabling compounding returns receive higher valuations[23]. Interestingly, the value function patterns are remarkably similar across algorithms despite differences in learning mechanisms, suggesting that they all approximate the same underlying optimal solution.

And, it is obvious that, achieving high mean asset result while maintaining low standard deviation is difficult in current tests and simulations.

3.9 Analysis of Algorithm Strengths

The comparison between training and evaluation results reveals important insights about each algorithm's learning characteristics. Q-Learning shows strong performance during training in moderate birth probability environments but variable transfer to evaluation scenarios, suggesting possible overspecialization to training conditions. SARSA(λ) demonstrates impressive generalization in high-reward potential environments, often improving from training to evaluation. This can be attributed to its eligibility trace mechanism enabling efficient credit assignment over the lengthy 30-year simulation horizon[24].

The consistent success of Monte Carlo algorithms in low birth probability environments, both during training and evaluation, aligns with theoretical expectations. In these more deterministic scenarios, Monte Carlo's focus on optimizing against actual experienced returns offers an advantage over bootstrapping methods. Without the complicating factor of exponential growth, the precision of Monte Carlo's updates outweighs the sample efficiency advantages of bootstrapping approaches. The fact that this advantage persists from training to evaluation underscores the robustness of Monte Carlo's learning in stable environments.

[23]The pattern observed in these value functions bears striking similarities to the concept of "value hills" in economics and finance, where early investments that enable compounding growth over time are highly valued by rational decision-makers.

[24]The 30-year horizon in this simulation creates a challenging temporal credit assignment problem. Early decisions can have compounding effects that only manifest decades later, making eligibility traces particularly valuable for connecting delayed rewards to their causal actions.

3.10 Strategy Reliability and Variance Analysis

The standard deviations in both training and evaluation data provide important insights about strategy reliability. In high-stochasticity environments (high birth and storm probabilities), outcome variability increases substantially, with standard deviations often exceeding €1.8 million for the B:0.5 environments across all algorithms. Training variances are particularly notable for SARSA(λ) (up to €1,925K) and SARSA (up to €1,932K), indicating more variable experiences during learning.

Interestingly, while SARSA(λ) achieves high mean performance in most high birth probability environments during evaluation, Q-Learning shows both the highest return (€10,465,308) and highest variance (€1,906K) in the most challenging B:0.5, S:0.5 scenario. This suggests a potential trade-off between maximum expected return and consistency of outcomes across different algorithms. These findings align with research on exploration-exploitation in reinforcement learning, where different credit assignment mechanisms can lead to varying balances between return maximization and outcome consistency[25].

4 Conclusion and Future Work

This project demonstrated the application of tabular reinforcement learning algorithms to a stochastic resource management problem in farm management[26]. Without hyperparameter tuning, with SARSA(λ) emerging as the most robust algorithm. The project implements the algorithms (Q-learning, SARSA, SARSA(λ) algorithms and Monte Carlo), creating a structured environment that follows the Gymnasium package, and providing comprehensive experimental results across all specified environmental conditions.

Future work could extend this project in several directions[27]:

- **Deep Reinforcement Learning:** Implement a Deep Q-Network (DQN) to handle a continuous state space (removing the need for budget discretization), which could lead to more nuanced policies.
- **Continuous Action Spaces:** Model the actions as continuous (e.g., how much money to invest in wheat), which is more realistic and could be tackled with policy gradient methods[28].
- **More Complex Environment:** Introduce more variables, such as fluctuating market prices for wool and wheat, or sheep mortality rates, to create a more realistic and challenging simulation.
- **Multi-Objective Optimization:** Extend the problem to balance multiple objectives like profit, sustainability, and risk mitigation, using recent advances in multi-objective reinforcement learning.
- **Hyperparameter Fine-tuning:** Conduct systematic hyperparameter optimization studies to improve algorithm performance, exploring the impact of learning rates, discount factors, and exploration schedules on final outcomes across different environmental conditions.

[25] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction, 2nd ed., MIT Press, 2018.

[26] Reinforcement learning offers powerful tools for optimizing sequential decision-making under uncertainty, as demonstrated in this project. The approach is particularly valuable for domains with clear reward structures but complex environmental dynamics.

[27] The extensions proposed here represent a natural progression from tabular methods to more advanced deep reinforcement learning approaches that could better capture the continuous nature of real-world agricultural decision-making or investment decision making.

[28] Continuous action spaces would more accurately reflect real-world farm management decisions, where farmers typically allocate varying proportions of resources rather than making binary choices.