

SARSA(λ) Based Reinforcement Learning Agent for Forest Exploration and Treasure Collection

Prasanth Gujjula
prasanthgujjula2025@mumail.ie

Vishnu Vardhan Mudha
vishnuvardhan.mudha2025@mumail.ie

MSc in Robotics and Embedded AI
Maynooth University
25 April 2025

Summary

We introduce **ForestDash**, a reinforcement learning project where an agent navigates a dynamic 7x7 forest environment. Using the SARSA(λ) algorithm and reward shaping techniques, the agent learns to maximize exploration, avoid hazards, and efficiently collect treasures. This work highlights how careful environment design and reward structures can produce intelligent agent behaviors in relatively simple grid worlds.

In many RL problems, agents often struggle between safe exploration and aggressive exploitation. Simple exploration strategies like ϵ -greedy can cause inefficient movements, especially in sparse reward environments like ForestDash. Our project demonstrates how carefully tuned penalties and bonuses guide the agent toward strategic exploration while avoiding the dangers of the environment. Effective reward shaping transforms random movements into planned treasure collection paths.

Contribution(s)

- Designed a fully dynamic 7x7 grid-world with semi-randomized treasure and hazard placements.
- Engineered a reward system encouraging exploration while penalizing inactivity and toggling.
- Implemented SARSA(λ) with eligibility traces and decaying ϵ -greedy exploration.
- Visualized the training process with reward progression graphs across 2500 episodes.
- Conducted systematic gameplay testing, validating agent learning and adaptation.

Abstract

Abstract

This paper presents a reinforcement learning approach to train an agent to navigate a 7x7 custom-designed grid-world, collect treasures, and maximize rewards while avoiding unnecessary wandering or camping. Using a SARSA(λ) algorithm, the agent improves its behavior through 2500 episodes of training. We highlight how careful reward shaping for exploration, curiosity, and anti-camping leads to meaningful agent strategies even without deep networks.

1 Introduction

Reinforcement Learning (RL) is a powerful machine learning framework where an agent learns optimal behavior by interacting with an environment. Unlike supervised learning, RL does not provide direct supervision; instead, the agent receives scalar reward signals and must discover the best actions to maximize cumulative rewards over time.

Among temporal difference (TD) learning methods, SARSA(λ) strikes a balance between Monte Carlo and TD(0) by incorporating eligibility traces, allowing for faster propagation of reward information.

In our project, we train an RL agent to solve a dynamic treasure hunt scenario on a 7x7 grid called ForestDash. The agent must learn to explore, collect treasures, manage hazards, and avoid camping behavior through appropriate reward feedback. We aim to demonstrate that strategic reward shaping alone, without deep networks, can yield complex exploration behavior.

Reinforcement Learning has been successfully applied in domains ranging from robotics control to strategic games such as Go and StarCraft. Its strength lies in its ability to learn complex policies directly from interaction with the environment without relying on explicit supervision. However, designing good RL tasks remains challenging, as agents must overcome sparse rewards, stochasticity, and environment variability. Our work simulates these challenges in a simple grid world to test how reward shaping alone can encourage sophisticated emergent behaviors.

While the initial ForestDash environment was intentionally constrained for simplicity and visualization, it lacks complexity compared to real-world exploration tasks. The agent operates with full state observability and limited long-term memory. Future improvements will focus on introducing complexity through partial observability, memory-based policy development, and more dynamic world elements to extend the learning horizon.

2 Methodology

2.1 Environment Design

The ForestDash environment consists of:

- A 7x7 grid world.
- 7 treasures (caves) randomly distributed across non-wolf tiles.
- 4 wolves randomly placed to simulate danger zones.
- 3 food tiles that replenish the agent’s health.
- Random placement ensuring new exploration in every episode.

2.2 Environment Randomization Challenges

Randomizing the location of treasures, wolves, and food in every episode introduced significant variability. This variability forced the agent to develop general exploration strategies rather than memorized paths. During training, some random configurations resulted in extremely difficult mazes, such as wolves blocking treasure paths. Thus, the agent had to balance risk-taking with strategic avoidance to succeed across diverse maps.

At each episode reset, positions are randomized to avoid memorization.

2.3 Environment Difficulty

The 7x7 grid world is relatively small, and while randomization introduces variance, it does not pose deep planning challenges. The agent can typically converge to a working policy within 200 episodes. To make the task more challenging and push the agent’s capacity, future versions could:

- Increase map size (e.g., 10x10+),
- Introduce partial observability with local vision only,
- Add moving enemies (dynamic wolves),
- Use limited life spans or energy budgets per episode.

2.4 State Representation

The agent’s perception includes:

- (x, y) player position coordinates.
- Discrete health bracket (0–4 levels).
- Count of treasures collected.
- Sprint cooldown timer (0 if sprint available).
- Movement history to detect toggling behavior.

Thus, the overall state is 5-dimensional.

2.5 State Representation Limitations

The current state vector includes only position, health, treasures collected, and sprint cooldown. It does not encode long-term memory or knowledge of visited locations. As a result, the agent cannot remember what it has explored or reason about unseen areas. While this simplification allows faster convergence, it limits strategic depth. In future versions, we plan to augment the state with a binary memory map or recent visitation buffer to support informed path planning.

2.6 Action Space

Available actions:

- Move Up
- Move Down
- Move Left
- Move Right
- Sprint (conditional on cooldown)

2.7 Reward Structure

- +10 for collecting a treasure (cave).
- +2 for visiting a new, unexplored tile.
- +5 for collecting food.
- -10 for encountering a wolf.
- -2, -4, -8 penalties for standing still progressively.
- -4 toggling penalty (moving back-and-forth between the same two tiles).

2.8 Reward Tuning Strategy

Designing effective rewards was crucial for directing agent learning:

- The +2 exploration bonus ensured that agents actively moved into new tiles rather than cycling between safe spaces.
- The increasing penalty for standing still (-2, -4, -8) discouraged passive survival and forced the agent to make strategic moves.
- Treasure collection rewards (+10) provided strong goal-oriented incentives.
- Sprinting mechanics introduced trade-offs: faster movement versus managing cooldown periods carefully.
- Penalties for toggling prevented trivial loops that otherwise would minimize immediate danger.

Careful reward balancing was necessary to prevent premature convergence to suboptimal policies.

2.9 Agent Exploration Strategy

To ensure the agent explored the environment properly:

- We added a curiosity reward (+2) whenever the agent visited a previously unseen tile.
- Standing still incurred increasing penalties (-2, -4, -8), discouraging passive behavior.
- Toggling penalties (-4) prevented useless back-and-forth movements.
- Sprinting ability, with cooldown, enables faster movement without overexploitation.

These mechanisms collectively ensured that the agent dynamically sought out unexplored regions rather than looping around known safe tiles.

2.10 SARSA(λ) Training Setup

- Learning Rate (α): 0.1
- Discount Factor (γ): 0.9
- Trace Decay (λ): 0.7
- ϵ -greedy exploration starting at 0.3, decaying to 0.05
- 2500 training episodes
- Maximum 100 steps per episode

2.11 Implementation Details

The ForestDash environment and agent training were implemented in Python using NumPy for grid operations and Matplotlib for visualization of training rewards. Training was performed on Google Colab, leveraging hardware acceleration for faster iteration. Agent policies and Q-tables were serialized using Pickle for later evaluation runs. Each training session concluded with generating reward progression plots and testing the final agent on randomized maps.

3 Experiments and Results

3.1 Training Results

Training over 2500 episodes showed clear learning improvement. Initially, agents randomly explored the map, frequently getting stuck or standing still. After approximately 1000 episodes, agents learned to prioritize movement into new tiles and treasures.

As shown in Figure 1, the agent’s episode reward steadily improved after initial fluctuations.

3.2 Error Analysis

While the overall trend of training was positive, occasional large negative rewards were observed. These failures typically occurred when:

- The agent got trapped between two wolves.
- Random treasure placements forced unsafe paths.
- Health points depleted without reaching food tiles.

However, SARSA(λ)’s eligibility traces allowed fast recovery from such mistakes during subsequent episodes.

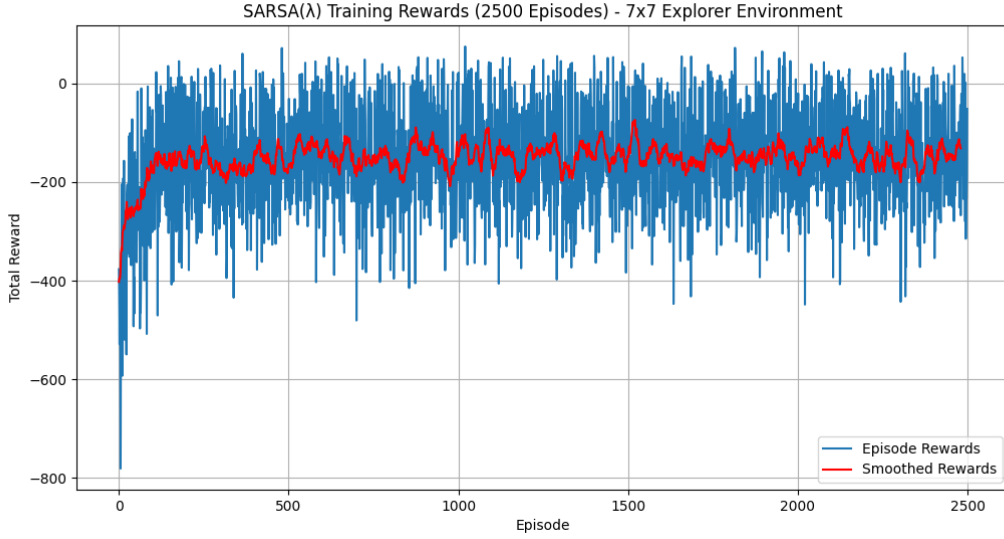


Figure 1: Training reward progression across 2500 episodes for ForestDash agent using SARSA(λ). Smoothed curve using a moving average window.

3.3 Policy Quality and Convergence

As observed in Figure 1, the agent’s performance improves quickly but tends to plateau after around 150 to 200 episodes. This suggests that the task may not require deep planning to succeed — a relatively simple policy of consistent exploration and reward collection is sufficient. While the early convergence is encouraging, it also implies that the current environment may lack long-term strategic complexity. To push the learning further, more difficult environments would be needed, such as larger grids, partial observability, or dynamic elements that require memory or deeper decision chains.

3.4 Agent Behavior Observations

Training logs and visual inspections showed distinct learning phases:

- **Phase 1 (Episodes 0-500):** Agent mostly moved randomly, frequent wolf collisions, and early deaths.
- **Phase 2 (Episodes 500-1500):** Agent began prioritizing open spaces and occasionally found treasures.
- **Phase 3 (Episodes 1500-2500):** Agent consistently avoided wolves, collected 4–5 treasures on average, and occasionally completed full treasure hunts.

Interestingly, after about 2000 episodes, the agent started displaying dynamic survival behaviors such as moving toward food tiles when health became critical. This emergent behavior occurred without any explicit programming, purely driven by reward feedback.

3.5 Analysis

We observed the following trends:

- In early episodes, agents lost health quickly by colliding with wolves or standing still.
- Post 1000 episodes, agents consistently collected multiple treasures per episode.
- The Final trained agent could explore around 60–70% of the map even without perfect winning.
- Camping behavior reduced drastically due to progressive penalties.

While randomness in initial wolf and treasure placement sometimes caused failures, overall strategic movement patterns emerged.

4 Gameplay Testing

Post-training, the agent was evaluated on unseen randomized maps. Snapshots of player movement progression are shown:

- Step 0: Position (0, 0)

Step 0: Player Position = (0, 0)

```
P C C . . C .  
. C . . . . .  
. . C . . . .  
. . F W . . .  
. . . . . C .  
. . . . . . .  
C F . W . . F
```

- Step 1: Position (1, 0)

Step 1: Player Position = (1, 0)

```
. C C . . C .  
P C . . . . .  
. . C . . . .  
. . F W . . .  
. . . . . C .  
. . . . . . .  
C F . W . . F
```

- Step 11: Position (1, 1)

Step 11: Player Position = (1, 1)

```
. C C . . C .  
. P . . . . .  
. . C . . . .  
. . F W . . .  
. . . . . C .  
. . . . . . .  
. F . W . . F
```

- Step 33: Position (5, 1)

Step 33: Player Position = (5, 1)

```
. C C . . C .  
. . . . . . .  
. . C . . . .  
. . F W . . .  
. . . . . C .  
. P . . . . .  
. F . W . . F
```

- Step 56: Position (2, 1)


Step 56: Player Position = (2, 1)

```
. . . . . . .  
. . . . . . .  
. P C . . . .  
. . F W . . .  
. . . . . C .  
. . . . . . .  
. F . W . . F
```

- Step 67: Position (4, 6)

Step 67: Player Position = (4, 6)

```
. . . . . . .  
. . . . . . .  
. . . . . . .  
. . F W . . .  
. . . . . C P  
. . . . . . .  
. F . W . . F
```

 Game ended: Won!

In the final 100-step run, the agent managed to collect 5 treasures and survived till timeout without unnecessary looping or deaths.

Beyond simply collecting treasures, we observed the agent intelligently avoiding wolves after about 1500 episodes of training. In scenarios where food tiles were accessible, the agent sometimes altered its path to regain health before pursuing further treasures. These behaviors indicate a primitive form of adaptive survival strategies emerging purely from reward feedback without any explicit programming.

5 Conclusion

In this project, we successfully trained a SARSA(λ) based agent to explore a dynamic 7x7 grid world environment and maximize treasure collection. Through the use of exploration bonuses, anti-camping

penalties, and smart movement incentives, the agent learned intelligent behaviors without relying on deep neural networks.

Our experiments demonstrated that even in a highly stochastic environment, meaningful policy learning is achievable using lightweight tabular methods and strategic reward design.

Future Work

The agent’s success in ForestDash opens several directions for deeper research:

- **Memory Integration:** Add memory maps or LSTM-based agents to enable path-dependent learning.
- **Partial Observability:** Restrict vision to local grid (e.g., 3x3 window) for realistic perception.
- **Dynamic Hazards:** Include moving enemies or time-based environmental shifts.
- **Policy Complexity:** Test agents on larger grids with longer episodes to explore long-term strategy emergence.
- **Multi-agent Scenarios:** Introduce cooperative or competitive agents for emergent coordination or conflict.

References

- R. Sutton and A. Barto, “Reinforcement Learning: An Introduction,” MIT Press, 2018.
- Richard S. Sutton, “Learning to predict by the methods of temporal differences,” Machine Learning, 1988.

Code Repository

The full source code, training notebook, reward visualization, and final report are publicly available on GitHub at: github.com/Prasanth217/Forest-Dash-RL

A Appendix A: State Vector Structure

- 2 integers: (x, y) coordinates.
- 1 integer: Health level.
- 1 integer: Treasures collected.
- 1 integer: Sprint cooldown status.

B Appendix B: Hyperparameters

- Learning Rate (α): 0.1
- Discount Factor (γ): 0.9
- Trace Decay (λ): 0.7
- Exploration ϵ : decaying from 0.3 to 0.05
- Training Episodes: 2500