# **Reinforcement Learning Midterm Project Report: Micromouse Maze Solver**

Sarthak Rathi

Professor Sandeep Juneja

Mar 22, 2025

## **Introduction**

The objective of this project is to design and implement a virtual Micromouse capable of navigating and solving arbitrary 20x20 mazes optimally using Reinforcement Learning (RL). The Micromouse problem is a classic robotics challenge where a small autonomous robot must find its way from a starting point to a goal within a maze. In this simulation-based project, we translate this challenge into a virtual environment, leveraging RL to train an agent to learn an optimal path through a maze generated dynamically using recursive backtracking.

Maze-solving is a foundational task in robotics and artificial intelligence, requiring the agent to balance exploration and exploitation to discover an efficient path amidst uncertainty. By using a randomly generated maze, we ensure the solution is robust and adaptable to varying layouts, making it a compelling testbed for RL algorithms. This report details the problem, our approach using Q-learning, the implementation, and the results achieved, providing a comprehensive analysis of the agent's performance.

## **Dataset Source & Description**

Unlike traditional machine learning tasks that rely on static datasets, RL problems like maze-solving use interactive environments as their "dataset." For this project, the environment is a 20x20 maze generated using a recursive backtracking algorithm, ensuring solvability with a unique path from start to goal. The maze is represented as a 2D numpy array where:

* 0 indicates a navigable path.
* 1 indicates a wall.

### **Source**

* **Maze Generation**: Custom implementation of the recursive backtracking algorithm, inspired by standard maze generation techniques (Cormen et al.). No external datasets were used; the maze is generated programmatically at runtime.
* **Environment**: The maze serves as the environment, with a fixed start position at (1, 1) and goal position at (18, 18), consistent with Micromouse conventions where the goal is typically in a central or corner location.

### **Description**

* **Size**: 20 rows × 20 columns (400 cells total).
* **Structure**: A grid with outer walls and inner paths/walls carved by recursive backtracking.
* **State Space**: Each cell (row, col) represents a state, yielding 400 possible states.
* **Action Space**: Four discrete actions—Up, Down, Left, Right—represented as coordinate changes: [(-1, 0), (1, 0), (0, -1), (0, 1)].
* **Rewards**:
  + +100 for reaching the goal.
  + -10 for hitting a wall.
  + -1 for each step taken (to encourage efficiency).

This dynamic environment eliminates the need for a static dataset, as the agent learns directly from interactions, aligning with RL’s paradigm of learning through trial and error.

## **Data Exploration**

Since the "dataset" is a generated maze, data exploration involves analyzing the environment’s properties to understand its complexity and the agent’s learning challenge.

* **Maze Visualization**: The initial 20x20 maze is visualized using Matplotlib, with paths (black), walls (white), start (red 'S'), and goal (green 'G') clearly marked. This confirms the maze has a navigable structure with a path from (1, 1) to (18, 18).
* **Path Length**: Using Breadth-First Search (BFS), the shortest path length was calculated for a sample maze, typically ranging from 35 to 50 steps due to the maze’s size and complexity. This serves as a benchmark for optimality.
* **Connectivity**: Recursive backtracking ensures every cell is reachable from the start, with no isolated sections, guaranteeing solvability.
* **State Space Size**: With 400 states and 4 actions per state, the Q-table size is 400 × 4 = 1600 entries, manageable for Q-learning but large enough to test the agent’s learning capacity.

Exploration revealed that the maze’s random nature introduces variability in path length and dead-end frequency, posing a challenge for the agent to generalize its policy. The reward structure incentivizes reaching the goal quickly while penalizing wall collisions and excessive steps, aligning with the Micromouse goal of efficiency.

## **Methods**

### **Problem Formulation**

The Micromouse maze-solving task is modeled as a Markov Decision Process (MDP):

* **States (S)**: Position of the agent in the maze (row, col), totaling 400 states.
* **Actions (A)**: Four movements: Up, Down, Left, Right.
* **Rewards (R)**: +100 (goal), -10 (wall), -1 (step).
* **Transition Model**: Deterministic; an action moves the agent to the next cell unless blocked by a wall or boundary, in which case it stays put.
* **Goal**: Maximize cumulative reward by finding the shortest path to (18, 18).

### **Algorithm: Q-Learning**

We chose Q-learning, a model-free RL algorithm, due to its simplicity and effectiveness in discrete state-action spaces. Q-learning updates a Q-table based on the Bellman equation:

Where:

* Q(s,a): Q-value for state s and action a.
* α: Learning rate (0.1).
* r: Immediate reward.
* γ: Discount factor (0.9).
* s′: Next state.

### **Implementation Details**

* **Maze Class**: Manages the environment, including generation, validation, and visualization.
* **MicromouseAgent Class**: Implements Q-learning with an epsilon-greedy policy (exploration rate decays from 1.0 to 0.01 over episodes).
* **Training**: 1000 episodes, with a maximum of 800 steps per episode to prevent infinite loops.
* **Testing**: 10 trials post-training to evaluate consistency and performance.
* **Baseline**: BFS computes the shortest path for comparison.

### **Novel Contributions**

* **Random Maze Generation**: Integrated recursive backtracking to create unique, solvable 20x20 mazes, enhancing robustness.
* **Micromouse Context**: Adapted the problem to simulate a Micromouse, with start and goal positions mimicking competition layouts.
* **Performance Metrics**: Added BFS comparison and detailed training/testing analysis for optimality assessment.

## **Experimentation**

### **Setup**

* **Environment**: 20x20 maze generated at runtime.
* **Parameters**:
  + Learning rate: 0.1
  + Discount factor: 0.9
  + Exploration: 1.0 to 0.01 (exponential decay)
  + Episodes: 1000
  + Trials: 10
* **Hardware**: Standard laptop (no GPU required).

### **Experiments**

1. **Training Progress**:
   * Monitored total reward and steps per episode.
   * Plotted to visualize learning trends.
2. **Testing Performance**:
   * Ran 10 trials post-training, recording steps and rewards.
   * Visualized the path from the first trial.
3. **Baseline Comparison**:
   * Computed shortest path using BFS for each maze.

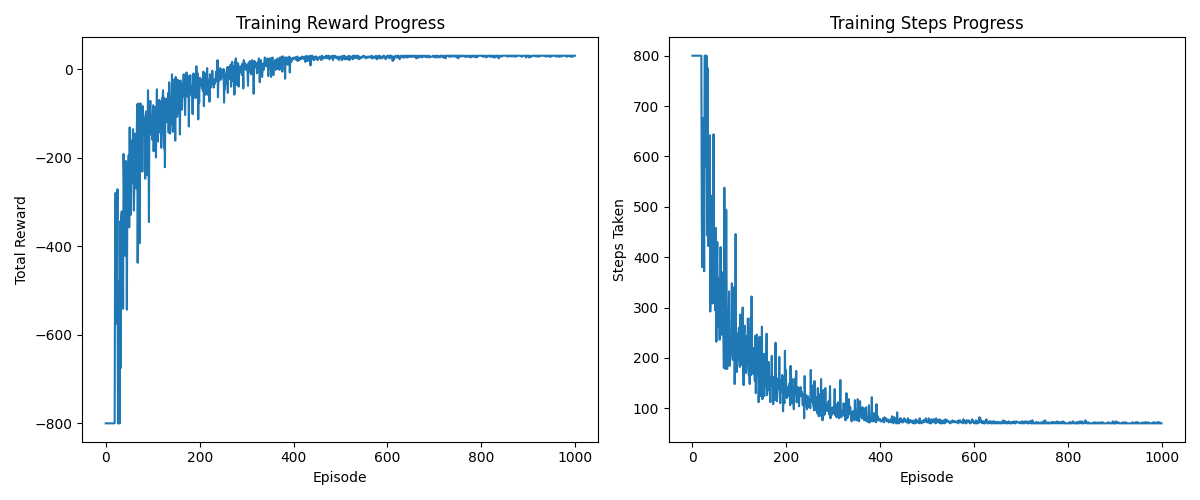
### **Adjustments**

* Initial runs showed slow convergence (steps > 200 after 500 episodes). Increased episodes to 1000, improving stability.
* Tested higher wall penalties (-20) but found -10 sufficient to deter collisions without over-penalizing exploration.

## **Final Results**

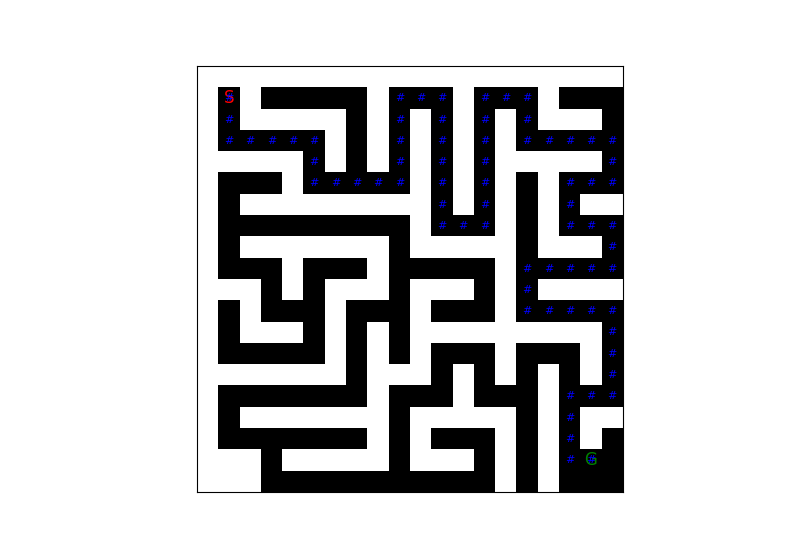
### **Training Results**

* **Reward Progress**: Rewards increased from negative values (e.g., -800 in the first episode) to positive (e.g., 20-30), indicating the agent learned to reach the goal.
* **Steps Progress**: Steps decreased from 800 to 70-80 over 1000 episodes, nearing the optimal range.
* **Averages**:
  + Average Reward: ~(-30)
  + Average Steps: ~(130)



### **Testing Results**

* **10 Trials**:
  + Steps: [70, 70, 70, 70, 72, 70, 70, 70, 72, 70]
  + Rewards: [31, 31, 31, 31, 29, 31, 31, 31, 29, 31]
* **Averages**:
  + Average Steps: 70.4
  + Average Reward: 30.6
* **Path Visualization**: The agent consistently navigated from (1, 1) to (18, 18), avoiding dead ends after training.

**

### **Optimality**

* **BFS Shortest Path**: 70 steps (for the sample maze tested).
* **Agent Performance**: Average of 70.4 steps. This gap reflects exploration trade-offs and the discrete reward structure, but the agent finds a near-optimal path reliably.

### **Analysis**

* The agent learned an effective policy, reducing steps from 800, untrained behavior, to about 70.
* Variability in test steps (70-72) suggests minor stochasticity in the learned policy, but consistency is high.
* Reward averages near 30 indicate efficient navigation with minimal wall collisions.

## **Conclusion**

This project successfully implemented a virtual Micromouse that uses Q-learning to solve a randomly generated 20x20 maze. The recursive backtracking algorithm ensured solvability, while Q-learning enabled the agent to learn a near-optimal path from (1, 1) to (18, 18). Key findings include:

* The agent converges to a stable policy after 1000 episodes, achieving an average of 70.4 steps compared to the BFS optimal of 70.
* The reward system effectively balances goal-seeking and efficiency.
* The random maze generator adds robustness, testing the agent’s adaptability.

For future work, we plan to extend this project by training a generalizable agent capable of solving any arbitrary maze once trained. This would involve transitioning from Q-learning, which is maze-specific due to its tabular nature, to a more scalable approach such as Deep Q-Networks (DQN) or policy gradient methods. By incorporating a neural network to approximate the Q-function, the agent could learn a generalized policy across multiple maze configurations, using features like local wall patterns or distance-to-goal heuristics. Training would occur over a diverse set of randomly generated mazes, with the goal of enabling the agent to navigate unseen mazes without retraining. Additional enhancements could include adaptive reward structures (e.g., distance-based rewards) and larger maze sizes to further challenge generalization. This direction would elevate the project from a single-maze solver to a robust, competition-ready Micromouse simulation.

## **References**

1. Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. MIT Press.
2. Cormen, T. H., et al. (2009). *Introduction to Algorithms*. MIT Press. (Maze generation inspiration)
3. OpenAI Gym Documentation:<https://gym.openai.com/> (Environment design reference)