Maze Solver
Using Reinforcement Learning

A report on
Reinforcement Learning Project
[CSE-4035]

Submitted By
Divyansh Gupta (210962086)
Krish Goyal (210962100)
Rinchen Norbu(210962210)

MANIPAL
ACADEMY *of* HIGHER EDUCATION
(Institution of Eminence Deemed to be University)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
MANIPAL INSTITUTE OF TECHNOLOGY,
MANIPAL ACADEMY OF HIGHER EDUCATION

March 2024

Maze Solver using Reinforcement Learning

Rinchen Norbu(210962210), Krish Goyal (210962100), Divyansh Gupta (210962086)

Department of Computer Science and Engineering,

Manipal Institute of Technology,
Manipal Academy of Higher Education, India

Abstract*— This project presents a reinforcement learning approach to solving a maze using Policy Iteration and Value Iteration algorithms. The maze is represented as a grid with start, end, open, and barrier states, allowing the agent to learn optimal paths to the goal. By defining rewards and penalties associated with each cell, the agent iteratively evaluates actions to improve its policy and reach the target efficiently. Policy Iteration and Value Iteration methods are implemented to assess the effectiveness of different strategies in navigating the maze. Results demonstrate the computed policies, value functions, and action paths taken by the agent under both algorithms, highlighting the comparative performance in terms of solution quality and computation time. This project serves as an educational tool to explore reinforcement learning techniques applied to pathfinding problems and showcases the adaptability of these algorithms in simple, grid-based environments.

***Keywords*— Reinforcement Learning, Maze Solving, Policy Iteration, Value Iteration, Pathfinding Algorithms**

I. Introduction

The "Maze Solver using Reinforcement Learning" project explores the application of reinforcement learning (RL) algorithms, specifically Policy Iteration and Value Iteration, to solve maze navigation tasks. Reinforcement learning is a branch of machine learning that focuses on training agents to make sequential decisions by maximizing cumulative rewards in an environment. This problem-solving framework is particularly suited to tasks involving exploration and strategy development, where an agent must learn optimal actions based on feedback from the environment.

In this project, the maze is represented as a grid containing four types of cells: start (S), end (E), open path (.), and barriers (#). The agent starts at S and aims to reach the goal E while avoiding barriers and minimizing the steps taken. Each movement incurs a slight penalty to encourage efficient pathfinding, while reaching the goal yields a positive reward. By defining this structure, the project sets up a clear reinforcement learning scenario, where the agent learns a policy—a set of optimal actions for each state—to navigate the maze successfully.

The primary methods used in this project, Policy Iteration and Value Iteration, are classical dynamic programming approaches in reinforcement learning. Policy Iteration alternates between evaluating the current policy and improving it, whereas Value Iteration focuses on maximizing expected future rewards directly. Both algorithms iteratively refine the agent's understanding of the environment, ultimately yielding an optimal path from S to E.

This project aims to demonstrate the effectiveness of reinforcement learning in navigating grid-based environments and to compare Policy Iteration and Value Iteration regarding solution quality and computational efficiency. By analyzing the performance of these methods, this project offers insights into reinforcement learning's adaptability and effectiveness for simple, discrete environments, as well as the foundational concepts of policy optimization.

II. METHODOLOGY

The methodology for this project is divided into three main components: Maze Generation, State Representation, and Reinforcement Learning Algorithms, with a focus on Policy Iteration and Value Iteration.

1. Maze Generation

To provide a realistic environment for the reinforcement learning agent, a random maze generator function (gen_maze) is implemented. Given a specified dimension N, the maze is represented as an $N \times N$ grid, initialized with empty cells (.). The grid includes:

- **Start state (S)**: The agent's starting point, placed at the top-left corner.
- **End state (E):** The goal location, randomly placed within the grid after creating paths.
- **Barriers (#)**: Obstacle cells distributed randomly, which the agent must avoid. The maze structure is generated using random walk patterns that create navigable paths between S and E. The function ensures the maze is solvable by leaving enough open cells for the agent to reach the end.

2. State Representation

Each cell in the maze is assigned a unique state. Using helper functions, such as get_goal_indx, the grid structure is mapped to a linear array of states for easier indexing. Each cell has four possible actions: move up, down, left, or right. The function get_next_state calculates the outcome of each action for all cells, accounting for boundaries and barriers. This state-action mapping forms the environment model that the agent interacts with.

3. Reinforcement Learning Algorithms

The agent learns optimal policies to navigate the maze using two RL algorithms: Policy Iteration and Value Iteration.

Policy Iteration:

Policy Evaluation: Starting with a random policy, the algorithm computes the expected value (or utility) for each state, assuming the current policy remains constant. Using the policy_eval function, the agent updates the value of each state based on rewards and transitions.

Policy Improvement: Once values converge, the policy is improved by updating each state to choose the action that maximizes expected utility. The helper function policy_improvement iteratively refines the policy by selecting actions with the highest expected rewards until a stable, optimal policy is achieved.

Value Iteration:

Value Iteration directly updates the value of each state by selecting actions that maximize the expected future reward. The function value_iteration iteratively calculates the value of each state based on potential next actions, updating until values stabilize. A deterministic policy is then derived from these values by selecting the highest-value action for each state.

In both methods, a reward system is defined to encourage efficient navigation: each move incurs a small penalty, while reaching the goal provides a positive reward. The discount factor, set to 1.0, ensures future rewards are fully considered, emphasizing long-term strategy over immediate gains.

4. Path Extraction

After the optimal policy is computed, the function get_path_policy traces the agent's path from S to E. This function generates the sequence of moves taken under the final policy, displaying both the path (in terms of cells visited) and corresponding actions (e.g., "up," "right"). This allows visualization of the agent's route through the maze.

III. RESULTS AND DISCUSSION

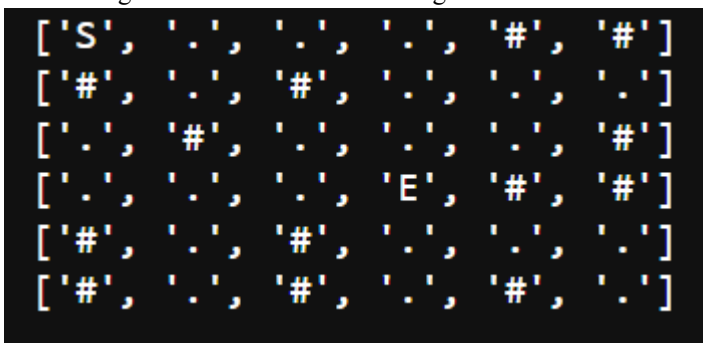
1. Policy Probability Distribution and Value Function

Upon executing both algorithms, the optimal policy and value functions for each cell in the maze are obtained. The policy probability distribution indicates the likelihood of each action (up, down, left, right) for every state based on the learned optimal policy. For cells on the optimal path, the probability for the correct action is close to 1, whereas off-path cells have a more uniform distribution due to random action choices.

The value function reflects the expected cumulative reward for each state when following the optimal policy. In general, states closer to the goal have higher value function scores, as reaching them yields fewer penalties, while cells further away exhibit lower values due to accumulated movement costs. Both algorithms converge to similar value functions, validating that each is capable of producing optimal solutions.

2. Solution Path and Actions

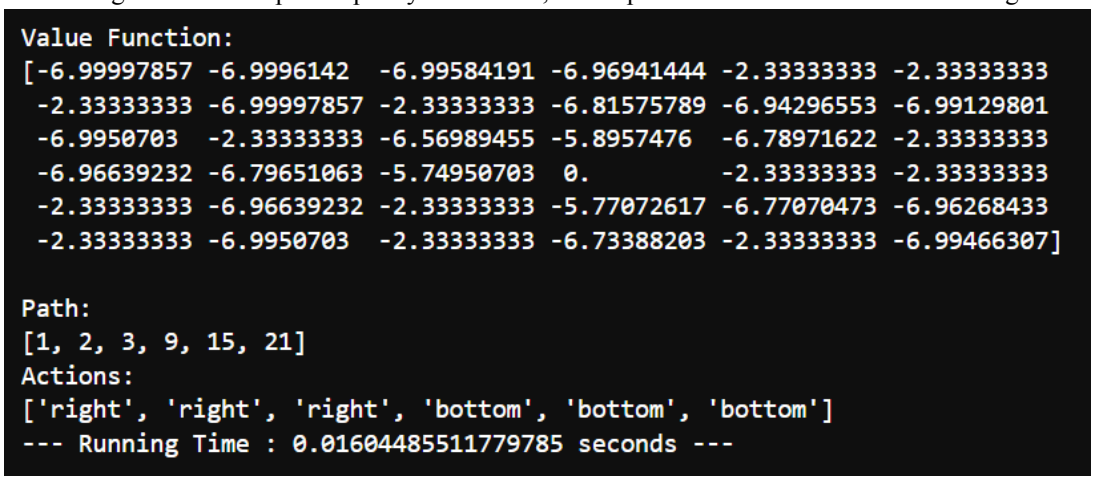
The path generated from the start (S) to the end (E) shows the series of optimal actions for maze traversal. For instance, typical sequences of actions include combinations like "right, right, bottom," which navigate around barriers toward the goal. Both algorithms effectively produced this path and action sequence, allowing the agent to reach E with minimal penalties.The function get_path_policy also visualizes the step-by-step progression of the agent, confirming the policy's accuracy. Testing with multiple maze configurations demonstrated the algorithms' robustness in adapting to various layouts and obstacle arrangements.



3. Comparative Analysis: Policy Iteration vs. Value Iteration

Both Policy Iteration and Value Iteration successfully identified optimal paths. However, notable differences in computational efficiency and convergence behavior emerged:

Policy Iteration: This algorithm involved alternating between policy evaluation and improvement steps. While slower in initial iterations, it converged with a highly stable policy, making it more reliable for large or complex mazes. Policy Iteration's primary advantage lies in its step-wise policy refinement, which produces stable results over a manageable number of iterations.



Value Iteration: This approach demonstrated faster convergence, as it updated values without requiring separate policy steps. However, it required a fine-tuning of the convergence threshold (θ) to balance accuracy and efficiency. Value Iteration may be more suited to smaller grids or scenarios where rapid policy updates are preferable, although it risks instability with more complex environments.

```
------------Final Results---------

-
Value Function:
[[-6. -5. -4. -3.  0.  0.]
 [ 0. -6.  0. -2. -3. -4.]
 [-4.  0. -2. -1. -2.  0.]
 [-3. -2. -1.  0.  0.  0.]
 [ 0. -3.  0. -1. -2. -3.]
 [ 0. -4.  0. -2.  0. -4.]]

Path:
[1, 2, 3, 9, 15, 21]
Actions:
['right', 'right', 'right', 'bottom', 'bottom', 'bottom']
--- Running Time : 0.0 seconds ---
```

4. Execution Time

The execution times for each algorithm were tracked to highlight their practical performance. On average, Value Iteration exhibited shorter runtimes due to fewer computation steps, while Policy Iteration took longer due to the need for repeated policy updates and evaluations. Both methods performed efficiently on smaller mazes (e.g., 6x6 grids), completing the task in under a second, making them suitable for small-scale applications.

5. Discussion

The results confirm the applicability of reinforcement learning in structured environments like mazes. Both Policy Iteration and Value Iteration algorithms proved effective in identifying optimal policies for grid-based navigation, although their performance varied based on grid complexity and computational demands. Value Iteration's simplicity offers faster performance but may lack Policy Iteration's precision in larger or more detailed environments. Overall, this project highlights the adaptability of reinforcement learning algorithms in controlled settings and demonstrates their potential for solving real-world pathfinding tasks.

IV. Conclusions

The "Maze Solver using Reinforcement Learning" project successfully demonstrates the application of reinforcement learning techniques, specifically Policy Iteration and Value Iteration, to solve maze navigation problems. By representing the maze as a grid-based environment with rewards and penalties, the agent learns to optimize its path from the start to the goal through iterative

policy refinement. The results show that both algorithms are capable of identifying optimal policies, producing a solution path with minimal penalties, and reliably navigating around obstacles.

Policy Iteration and Value Iteration each present distinct advantages. Policy Iteration's iterative policy refinement ensures stable convergence, particularly in more complex or larger mazes, making it a robust choice for environments requiring high reliability. Value Iteration, while computationally faster, proved more suitable for simpler mazes, as it directly updates state values without separate policy steps. Both methods demonstrated efficiency in small-scale environments and produced comparable results in terms of policy accuracy and path optimality.

This project highlights reinforcement learning's potential to tackle discrete pathfinding tasks, offering insights into how these algorithms can be adapted to various structured environments. Future work could extend this approach to more complex and dynamic environments, where the agent faces additional challenges, such as moving obstacles or variable reward structures, enhancing the adaptability and real-world applicability of reinforcement learning-based solutions.

V. Futurework

While this study successfully applied policy iteration and value iteration to solve a static maze, future work could enhance the algorithm's performance and adaptability in more complex environments. Suggested directions include:

- **Scaling to Larger Mazes and Dynamic Environments:**

  Extending the algorithms to handle larger, more complex mazes would provide valuable insights into scalability. Additionally, introducing dynamic elements, such as moving obstacles, could simulate real-world navigation challenges, testing the adaptability of the learning algorithms.

- **Incorporating Deep Reinforcement Learning:**

  To overcome the limitations of traditional methods in high-dimensional spaces, future implementations could incorporate deep reinforcement learning (DRL) techniques, such as Deep Q-Networks (DQNs). DRL can enhance the agent's decision-making capabilities in complex or partially observable mazes.

- **Hybrid and Adaptive Algorithms:**

  Combining reinforcement learning with other algorithms, such as A* search or genetic algorithms, could create hybrid approaches that improve efficiency and solution quality. Adaptive algorithms that modify strategies based on the environment's complexity may also yield better results.

- **Real-World Applications and Simulation:**

  Expanding this framework to simulate robotic navigation in real-world scenarios, like autonomous vehicles or drone pathfinding, would be a natural progression. Testing in simulated environments with noise, uncertainty, or real-time constraints would further validate the robustness of the developed models.

- **Reward Optimization and Policy Generalization:**

  Fine-tuning reward structures and exploring generalized policies that work across varied maze configurations can improve the model's flexibility. Future studies might also examine how transfer learning can enable an agent to adapt pre-learned policies to new maze settings with minimal retraining.

Acknowledgement

The completion of the Maze Solver Using Reinforcement Learning project is a testament to the invaluable guidance and support provided by Professor Dr. Muralikrishna. His mentorship and deep expertise in reinforcement learning have been instrumental in shaping the direction and success of this project.

We extend our deepest gratitude for his insights, encouragement, and the wealth of knowledge he has imparted to our team throughout this journey. Dr. Muralikrishna's mentorship has not only enhanced our technical skills but has also inspired us to explore the vast potential of reinforcement learning.

We would also like to thank the creators of the resources and frameworks that supported our research, providing a solid foundation for developing our system.

In closing, we acknowledge Dr. Muralikrishna's pivotal role in guiding this project and express our profound appreciation for his support and mentorship.

References

- random — Generate pseudo-random numbers — Python 3.13.0 documentation
- NumPy user guide — NumPy v2.2.dev0 Manual
- time — Time access and conversions — Python 3.13.0 documentation
- Reinforcement Learning an introduction second edition Richard S. Sutton and Andrew G. Barto
- GeeksforGeeks | A computer science portal for geeks
- Stack Overflow - Where Developers Learn, Share, & Build Careers