CS 5660 Final Report
for
Reinforced Learning Snake And Ladders Game

Prepared by:

Nihit Sakhuja

Hojat Talkhouncheh

Jacob Valenzuela

Shreyansh Sisodia

Libina Thomas

Project Title: Reinforced Learning Snake and Ladders Game

Responsibilities of Each Team Member:

- **Project Lead:** Shreyansh Sisodia
    - Overall responsibility for the project, oversees the work of all team members, ensures that the project stays on track and meets the deadlines.
- **Team Member 2:** Hojat Talkhouncheh
    - Develop the core logic for state management, action selection, and reward assignment. Also implemented the Bellman equation for updating the Q-table.
- **Team Member 3:** Nihit Sakhuja
    - Set up test cases to evaluate the efficiency of pathfinding, the impact of different rewards and penalties, and the robustness of the algorithm under various game configurations. Analyze data from simulation runs to identify patterns, optimize parameters, and suggest improvements.
- **Team Member 4:** Libina Thomas
    - Design the game layout, including the placement of ladders and snakes, defining start and finish points, and ensuring there are various paths with different challenges. Document the project development process and results. Prepare the presentation slides.
- **Team Member 5:** Jacob Valenzuela
    - Document the project development process and results.Prepare reports and presentations that detail the methodology, results, challenges, and learnings from the project.

Project Description

"Reinforcement Learning for Snake and Ladders Game" is a project aimed at developing an AI agent capable of playing the classic board game Snake and Ladders using reinforcement learning techniques. In Snake and Ladders, players navigate a grid-like board by rolling dice, encountering ladders that allow them to advance and snakes that set them back. The AI agent will learn optimal strategies to navigate the board, avoid snakes, and reach the final square before its opponents.

The project's approach involves creating a simulation of the Snake and Ladders game environment and training the AI agent to make decisions based on its observations and receive rewards. Reinforcement learning algorithms, such as Q-learning or Deep Q-Networks, will be employed to enable the agent to learn from its interactions with the environment over time. By iteratively playing the game and adjusting its strategies based on feedback, the AI agent will develop a policy for making decisions that maximize its chances of winning.

The implementation will involve designing the game environment, including the board layout, dice rolling mechanics, and rules for movement. The AI agent will interact with this environment, selecting actions (e.g., move forward, climb ladders) and updating its policy based on the rewards received. Evaluation of the agent's performance will be based on metrics such as win rate, average game length, and adaptability to different game scenarios.

Overall, this project aims to showcase the effectiveness of reinforcement learning in solving complex decision-making problems in game environments, with potential applications in gaming, robotics, and beyond.

Project Details

Objective: Develop an AI agent capable of playing the Snake and Ladders game autonomously using reinforcement learning techniques.

Approach: Implement reinforcement learning algorithms to enable the AI agent to learn optimal strategies for navigating the game board, avoiding snakes, and climbing ladders to reach the final square.

Implementation:

- Create a simulation of the Snake and Ladders game environment.
- Design the AI agent to interpret game states, select actions, and update its policy based on received rewards.
- Train the AI agent using reinforcement learning techniques, such as Q-learning or Deep Q-Networks.

Evaluation: Assess the performance of the AI agent based on metrics such as win rate, average game length, and adaptability to different game scenarios.

Project Goals

1. Develop an AI Agent:
    - Design and implement an AI agent capable of autonomously playing the Snake and Ladders game.
    - Create a flexible and scalable architecture that can accommodate different reinforcement learning algorithms and strategies.

2. Optimal Strategy Learning:
    - Train the AI agent to learn optimal strategies for navigating the game board, maximizing its chances of reaching the final square.
    - Implement algorithms and techniques for policy learning, exploration-exploitation trade-offs, and reward shaping to facilitate efficient strategy acquisition.

3. Adaptability:
    - Enable the AI agent to adapt its strategies dynamically based on feedback received from the game environment.
    - Implement mechanisms for online learning and continual adaptation to evolving game conditions and opponent behaviors.

4. Generalization:
    - Validate the learned strategies across different board configurations, game scenarios, and player skill levels.
    - Assess the generalization capabilities of the AI agent through cross-validation and testing on unseen data.

5. Interpretability:
    - Provide transparency into the decision-making process of the AI agent, allowing stakeholders to understand its reasoning and behavior.
    - Implement visualization tools and diagnostic mechanisms to inspect and interpret the learned policies and strategies.

6. Evaluation Metrics:
    - Define and measure appropriate evaluation metrics to assess the performance of the AI agent comprehensively.
    - Include metrics such as win rate, average game length, convergence speed, and robustness to perturbations.

7. Comparison with Baselines:
    - Conduct comparative analysis with baseline approaches, including rule-based strategies or handcrafted heuristics, to evaluate the effectiveness and superiority of the reinforcement learning-based AI agent.
    - Benchmark the AI agent against human players or existing AI implementations to gauge its competitiveness and advancement.

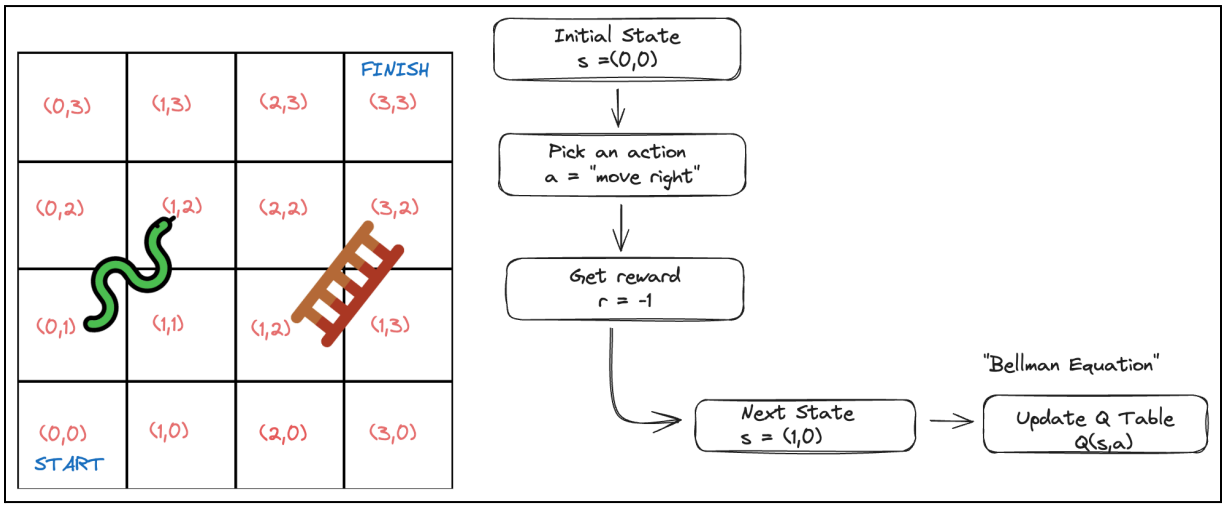
Q-Learning: Modeling Q Values with a Lookup Table

Q-Learning is a model-free RL algorithm, meaning that we don't have to learn a model for the environment with which the agent interacts. Concretely, this means that we don't have to train a model to estimate the transition or reward functions—these are just given to us as the agent interacts with the environment. The goal of Q-Learning is to learn the value of any action at a particular state. We do this through learning a Q function, which defines the value of a state-action pair as the expected return of taking that action at the current state under a certain policy and continuing afterwards according to the same policy.

The algorithm. The first step of Q-learning is to initialize our Q values as zero and pick an initial state with which to start the learning process. Then, we iterate over the following steps (depicted above):

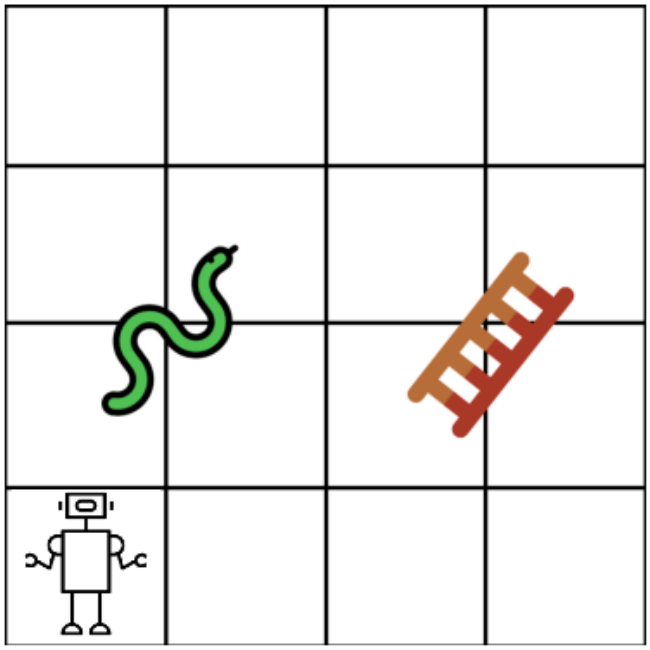
1. Pick an action to execute from the current state (using an ε-Greedy Policy).
2. Get a reward and next state from the (model-free) environment.
3. Update the Q value based on the Bellman equation.

Below is a graphical representation of the algorithm implemented.



Methodology

1. Board Representation:
    a. Grid Size: The board is represented as a 4x4 sized grid, denoting the 16 squares on the game board.
    b. Image Representation: Used Matplotlib to render the game board and the agent on it. The board and agent images are loaded using mpimg.imread() function.
    c. Board Figure:



2. Action Space and Movement:
    a. Action Space: Four actions are created representing the movement directions: up, down, left, and right.
    b. Next Step Calculation: The next_step_reward() function calculates the next position based on the current and chosen action, ensuring the agent stays within bounds of the grid based on a system.
3. Rewards and Positions:
    a. Ladders and Snakes: Special positions on the board are designed as ladder bottoms, ladder tops, snake heads, and snake tails. Moving to these positions triggers rewards or penalties

b. Reward System: Rewards and penalties are assigned based on the movement outcomes.
    i. Ladder bottom provides a positive reward, setting our variable reward to 2.
    ii. Snake head provides a negative reward, setting reward to -2.
    iii. Final goal provides a positive reward, setting reward to 5.
    iv. Standard steps on the board provide a negative reward, setting reward to -1.

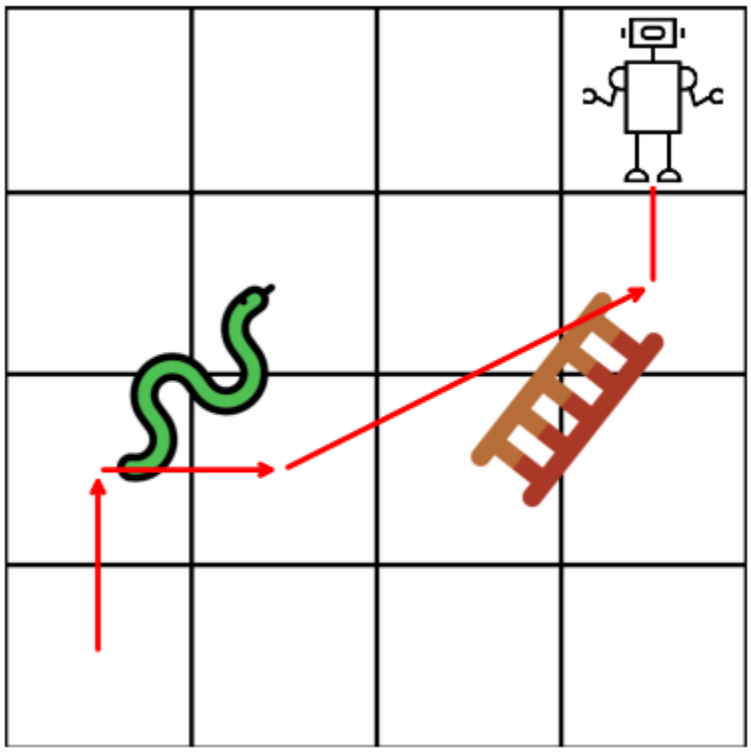
4. Q - Learning:
    a. Q-Table Initialization: The Q-values for each state-action pair are initialized to zero.
    b. Exploration vs. Exploitation: The best_action() function selects the action with the highest Q-value for exploitation, with a random choice for exploration.
    c. Q-Value Update: Q-values are updated using the Q-learning formula, incorporating immediate rewards and the maximum expected future rewards
5. Training:
    a. Game Loop: The agent plays multiple gameplay sessions, each session being referred to as a game. The variable games denote the total number of games the agent will play during the training phase. Within each game, the agent executes a series of actions and updates its strategy based on the feedback given from the game environment. Game is set to 10, due to the size of the board being a 4x4.
    b. Episodes: Within each game, the agent undergoes sequential episodes until it reaches the final square and achieves the goal state. An episode represents a single playthrough from the starting position to the goal state. The variable episodes determine the maximum number of episodes per game, confirming that the agent has sufficient opportunities to learn and refine its strategies based on the game's feedback.
    c. Reward Accumulation: Total rewards earned during each game are tracked to evaluate agent performance.

d. Visualization: Matplotlib is used to visualize the agent's movements on the game board in each step.

6. Optimal Path:
   a. Backtracking: The optimal_path() function backtracks from the final square to the starting square, determining the optimal path based on learned Q-values.

Results

After running through 10 games, our model was able to determine the best path based off of Q-Learning. Our path:



Running our method optimal_path it returns the output:

Optimal Path	(0, 0)	(1, 0)	(1, 1)	(2, 3)	(3, 3)

Conclusion & Learnings

In conclusion, this project successfully demonstrated the application of the Q-learning algorithm in a simulated "Snakes and Ladders" game environment, highlighting the potential of reinforcement learning in navigating complex decision-making scenarios. Throughout the project, we refined our understanding of how to effectively implement and optimize the Q-learning model, integrating components such as the learning rate, discount factor, and reward system to enhance the agent's performance. The challenges encountered, particularly in tuning these parameters to balance between exploration and exploitation, provided deep insights into the dynamic nature of machine learning algorithms. This experience has not only reinforced the importance of meticulous testing and iterative development in artificial intelligence projects but also underscored the value of a clear visual representation in understanding and communicating complex data and algorithms. Moving forward, these learnings will serve as a solid foundation for further exploration into more advanced reinforcement learning strategies and their applications in various real-world settings.

References

1. https://cameronrwolfe.substack.com/p/basics-of-reinforcement-learning
2. Q-Learning Algorithm: From Explanation to Implementation | by Amrani Amine | Towards Data Science
3. https://medium.com/@Kaushik_Dayalan/bellman-equation-value-functions-reinforcement-learning-b8a0a1cad84f