Maze Game

Abstract

This project is a Maze game in which we have 2 distinct modes: one where users can play as the Hider against the Seeker and another where both the Hider and Seeker are RL agents engaged in a dynamic pursuit trying to collect maximum rewards. The job of the Hider is to reach the goal(the exit of the maze) .The job of Seeker is to find the Hider before the Hider reaches the goal.The agent navigates a grid-based environment, learning to make optimal decisions through reinforcement learning. This project showcases the application of Reinforcement learning in game development, providing a foundation for more complex AI-based games.

Task Definition and Modelling

Goal: In this project, we have two agents: Hider and Seeker. The Hider aims to reach the goal while evading the Seeker, whereas the Seeker's objective is to locate the Hider before the Hider reaches the goal.

State Space: The state space consists of the positions of both the Hider and Seeker on the grid, represented as tuples of their respective coordinates. Additionally, the state includes information about the layout of the maze, such as the locations of walls and the goal.

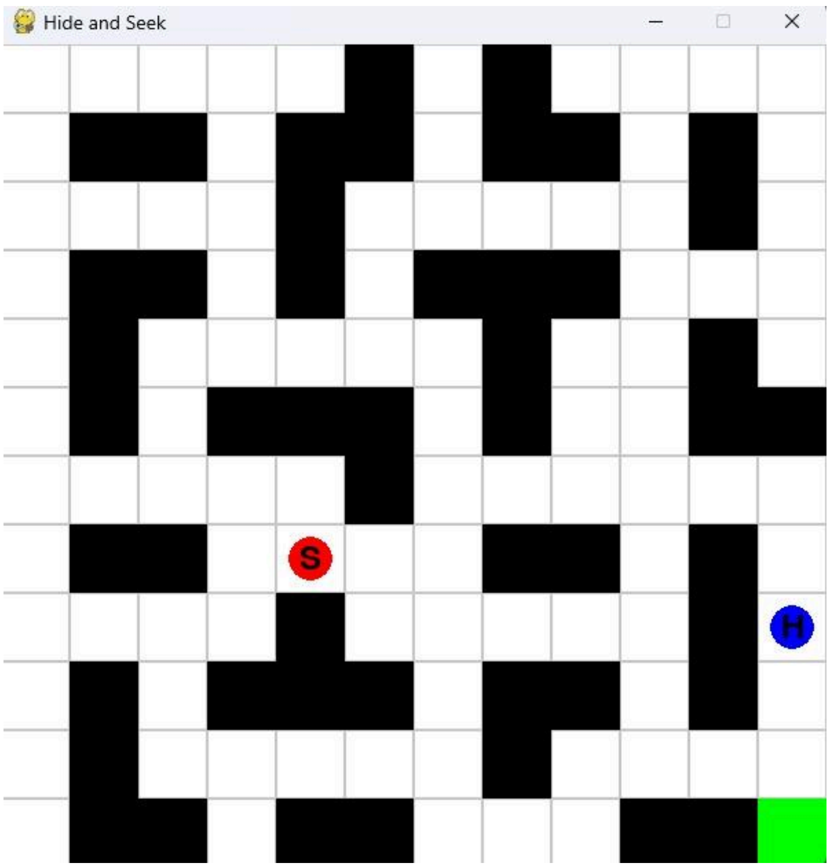
Action Space: Both agents have four possible actions: up, down, left, and right. These actions determine the movement of the agents on the grid, constrained by the boundaries of the maze and any obstacles.

Dynamics: The dynamics of the environment are governed by the rules of the maze and the movement capabilities of the agents. Each action taken by an agent results in a transition to a new state based on the agent's current position and chosen action. The environment updates the positions of both agents simultaneously, considering valid moves within the maze structure. Collisions with walls result in no movement for that action.

Rewards: The rewards are calculated based on the relative positions of the Hider and the Seeker. If they occupy the same position, the Seeker wins, resulting in a -1 reward for the Hider and a +1 reward for the Seeker. If the Hider manages to move far away from the Seeker (beyond half the grid size) and reaches the goal, the Hider wins, receiving a +1 reward, while the Seeker gets a -1 reward. Both agents receive a +0 reward if they have not completed their respective tasks within the 100 steps.

Methodology

1. We define a Maze Structure:



,

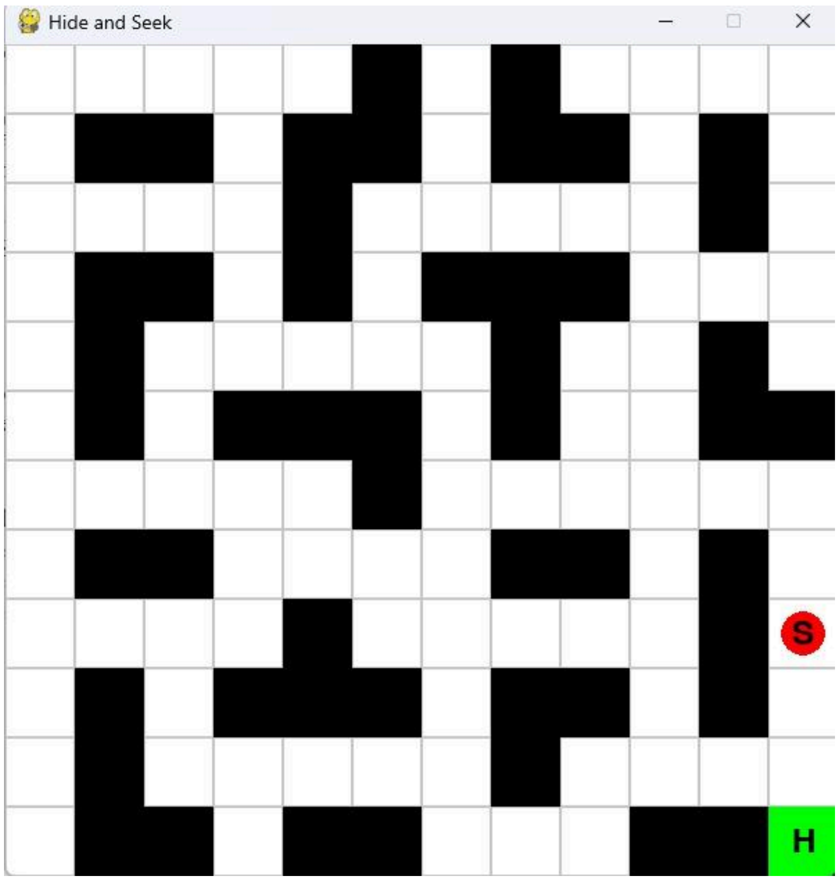
S→Seeker

H→Hider

Green box→Goal or exit

2. We have defined a reward function :
    - Hider:
        - +1 reward for reaching the goal.
        - -1 reward if it fails to reach the goal within 100 steps.
    - Seeker:
        - +1 reward for finding the Hider.
        - -1 reward if it fails to find the Hider within 100 steps.

Both the Hider and the Seeker receive +0 reward if they fail to complete their respective tasks within 100 steps.



Hider reaching the goal

3. Implement the Algorithm and evaluate it:

   We are comparing between 4 algorithms to find which one is suitable for our project.

   1.Policy Gradients: Policy gradient methods update the policy parameters directly to maximize the expected cumulative reward. The objective function to be optimized is:

$$J(\theta) = \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t R_t\right],$$

where R, is the cumulative reward at time step t.

2.Q-Learning: The Q-learning algorithm updates the action-value function

Q(s, a) iteratively based on the observed rewards and transitions.

The Q-value update equation is given by:

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

where a is the leaming rate and 7 is the discount factor

3.SARSA:SARSA updates the action-value function using the state-action-reward-state-action (SARSA) tuples.

The update rule is:

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

where (s, a, r, s', a') is the observed transition.

4. **Deep Q-Networks (DQN)**: DQN employs a neural network to approximate the action-value function Q(8, a;0). It minimizes the mean squared error between the predicted Q-values and the observed rewards:

$$\mathcal{L}(\theta) = \mathbb{E}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta) \right)^2 \right],$$

where θ are the parameters of the neural network and θ^- are the target network parameters.

4.We will choose an Algorithm suitable for it and train it

5. Evaluation:

- **Performance Metrics**: The performance of the trained agents is evaluated by running multiple episodes and tracking the number of wins for both agents.
- **Steps to Completion**: The number of steps taken to complete each episode is recorded to assess the efficiency of the agents.

6.After this we will save the model and load it for each game.

We have 2 modes:

- **User_Mode**:Where the user can play as Hider against Seeker Agent
- **Agent_Mode**:Where the Hider and Seeker Agent play against each other trying to accumulate maximum reward

7.Visualization:

- **Pygame Integration**: The environment and agents' actions are visualized using the Pygame library.
- **Grid Display**: The grid, including walls, agents, and the goal, is rendered.
- **Agent Movements**: The agents' movements are animated to show the progress of each episode.

Experiments and Results

Details of our experiments.

To evaluate the model, now we check the performance over 1000 episodes

1.Policy Gradients:

Average Steps per episode:22.816

Total episodes where Hider wins:663

Total episodes where Seeker wins:140

Episodes where Hider and Seeker both failed their task: 197

2.Q-Learning:

Average Steps per episodes:77.652

Total episodes where Hider wins:138

Total episodes where Seeker wins:95

Episodes where Hider and Seeker both failed their task: 767

3.SARSA:

Average Steps per episodes:34.429

Total episodes where Hider wins:583

Total episodes where Seeker wins:103

Episodes where Hider and Seeker both failed their task: 317

4.Deep Q-Networks (DQN):

Average Steps per episodes:50.985

Total episodes where Hider wins:404

Total episodes where Seeker wins:164

Episodes where Hider and Seeker both failed their task: 432

Analysis

1. **Policy Gradients**: Policy gradient methods directly optimize the policy parameters to maximize the expected cumulative reward. They are suitable when the action space is continuous or large, as they learn a parameterized policy. In this case, Policy Gradients achieve the lowest average steps per episode, indicating efficient exploration and exploitation of the environment. However, they require careful tuning of hyperparameters and can be computationally intensive due to the need for gradient estimation.

2. **Q-Learning**: Q-Learning is a fundamental reinforcement learning algorithm that learns action-values directly and is model-free. While Q-Learning achieves a higher average steps per episode compared to Policy Gradients, it may still be a suitable choice in scenarios where the environment is relatively simple and the action space is discrete. However, it struggles in this scenario due to the large action space and complexity of the environment, as indicated by the high number of episodes where both agents fail. Q-Learning also tends to converge slowly and may require extensive exploration to achieve optimal performance.

3. **SARSA**: SARSA is an on-policy temporal difference learning algorithm that updates action-values based on state-action-reward-state-action transitions. SARSA performs better than Q-Learning in this scenario, achieving a lower average steps per episode and a higher number of successful Hider wins. It's suitable for scenarios where the agent's policy needs to be refined based on observed experiences, as it directly learns from the actions taken by the policy. SARSA's performance indicates a good balance between exploration and exploitation, but it may still face challenges in highly dynamic environments.

4. **Deep Q-Networks (DQN)**: DQN employs neural networks to approximate the action-value function, enabling it to handle complex and high-dimensional state spaces. While DQN achieves a moderate average steps per episode and a reasonable number of successful Hider wins, it may be a preferred choice when dealing with environments where raw sensory inputs are required to make decisions. DQN's ability to generalize across similar states is beneficial in complex environments, but the training process can be computationally expensive and sensitive to hyperparameter settings. The introduction of techniques such as experience replay and target networks helps stabilize training, but further improvements could enhance robustness and efficiency.

Scalability and Robustness: The scalability of these algorithms varies, with DQN showing potential for handling larger and more complex state spaces due to its use of neural networks.

Policy Gradients also scale well with large action spaces but may require more computational resources. Q-Learning and SARSA, while effective in simpler environments, may struggle with scalability and robustness in more complex settings. Future work could explore combining these methods (e.g., Actor-Critic approaches) to leverage their strengths and mitigate weaknesses. s

Computational Efficiency: In terms of computational efficiency, Q-Learning and SARSA are generally faster to train in simpler environments but may require more episodes to converge in complex settings. Policy Gradients and DQN, while potentially more computationally intensive, offer greater flexibility and scalability. Optimizing the training process and exploring more efficient architectures could improve their applicability to real-time game scenarios. Potential Improvements: Potential improvements include incorporating advanced techniques such as reward shaping, hierarchical reinforcement learning, or meta-learning to enhance agent performance. Additionally, experimenting with different neural network architectures and regularization techniques could further improve the robustness and efficiency of DQN and Policy Gradient methods.

The Agents are able to play their respective roles properly and provide a smooth interaction with the enivornment.

Conclusion

The project highlights the use of different reinforcement learning algorithms in a multi-agent environment. Policy Gradients emerged as the most effective method for this particular task, efficiently balancing exploration and exploitation to maximize rewards. SARSA also showed promising results, offering a viable alternative to Q-Learning and DQN in scenarios requiring policy refinement based on direct experiences. Future work could explore more sophisticated algorithms and variations, such as combining Policy Gradients with DQN (e.g., Actor-Critic methods), to further enhance the performance and adaptability of the agents. Additionally, extending the environment to include more complex mazes and dynamic obstacles could provide further insights into the scalability and robustness of these algorithms in more challenging settings. Overall, this project serves as a foundational step towards integrating reinforcement learning in game development, showcasing its potential to create intelligent and adaptive agents capable of navigating and interacting within complex environments.

References

1. Generating intelligent agent behaviors in multi-agent game AI using deep reinforcement learning algorithm

2. Reinforcement Learning: An Introduction by Richard S. Sutton and Andrew G. Barto