AI Project

Hide-and-Seek with q-learning

Team:
Shaik rahul-12041350
Jagadeesh-12240810

1**. Introduction:**

Welcome to our AI-powered Hide-and-Seek game! Experience the thrill of intelligent agents, both hiders and seekers, navigating a dynamic arena, learning and adapting their strategies through the power of reinforcement learning. This project showcases the fascinating intersection of playful competition and strategic decision-making, where every move unlocks new possibilities and challenges.

2. **State Space:**

Our agents perceive the world through a state representation defined as a tuple: (x_coordinate, y_coordinate, distance_from_start, cumulative_reward). This provides information about their location, progress, and accumulated rewards.

3. Algorithm:

We utilize the Q-learning algorithm to guide our agents' decision-making process. Q-learning enables agents to learn optimal actions by associating Q-values with state-action pairs. These values represent the expected cumulative reward for taking a specific action in a given state.

The agents explore the environment and update their Q-values based on the following equation:

$$Q(s, a) \leftarrow (1 - \alpha) * Q(s, a) + \alpha * (R + \gamma * \max_{a'} Q(s', a'))$$

where:

Q(s, a): Q-value for the current state-action pair.

α: Learning rate (controls how much new information influences Q-values).

R: Immediate reward obtained after taking action a in state s.

γ: Discount factor (values future rewards less than immediate ones).
s': Next state reached after taking action a.
a': Possible actions in the next state s'.
Page 2

4. Tech Stack:

Programming Language: Python
GUI Development: Tkinter (for creating the user interface)
3D Visualization: Raycast (for simplified rendering of visual elements)
Data Storage: Pickle files (for saving and loading trained game states)
Algorithm: Q-learning

5. Training Process:

Our training process involves iteratively refining the behavior of hiders and seekers using reinforcement learning.

5.1 Policies and Rules:

The policy.py code defines the decision-making strategies for both seekers and hiders. The Q-learning algorithm updates Q-values based on rewards and penalties, guiding agents towards optimal actions. We employ an epsilon-greedy approach to balance exploration and exploitation, allowing agents to discover new strategies while leveraging learned knowledge.

5.2 Reward System:

The reward system incentivizes desirable actions like successful hiding or capturing, while penalizing undesirable actions like wall collisions. This feedback mechanism guides the Q-learning algorithm, helping agents learn effective strategies.

5.3 Iterative Improvement:

Through thousands of iterations, agents learn and adapt their behavior based on experiences and rewards. The Q-learning

algorithm continuously updates Q-values, leading to increasingly sophisticated strategies for both hiders and seekers. Trained data is stored in pickle files, ensuring that agents retain learned knowledge across iterations.

5.4 State-Action Exploration:

Agents explore various state-action pairs, continuously refining their Q-values stored in pickle files. As the learning rate (α) increases beyond 0.1, the agents rely more on learned values, demonstrating more informed and effective decision-making.

6. Pickle Files:

Pickle files are used to store the agents' learned knowledge, specifically the Q-values associated with state-action pairs.

6.1 Initial Pickle File Content:

The initial content of the hider pickle file includes state-action pairs with their
corresponding Q-values. For example:
Key(state) :: (384, 184, 0.0, 0.0)
Value(action) :: {'LEFT': 0, 'RIGHT': 0, 'UP': 0.060156786186756 92, 'DOWN':0.13186272464634627}
This represents the initial Q-values for specific states and actions, providing a foundation
for the hider's decision-making process.
.

6.2 Training and Update:

During training, the pickle files are updated after each iteration. Q-values are adjusted based on the agents' experiences, reinforcing successful actions and discouraging unsuccessful ones. For example, a hider that successfully evades the seeker will have the Q-value for the chosen action in that state increased.

6.3 Iterative Improvement:

The iterative process of updating Q-values is crucial for refining the agents' strategies over time. As they accumulate more experiences and feedback, their decision-making becomes more sophisticated, leading to improved performance in the game.

7. Results:

7.1 Untrained Game:

Initially, without any training, both hiders and seekers exhibit random and ineffective behavior. Their movements lack strategy, resulting in inefficient hiding and seeking.

7.2 Trained Game:

After sufficient training, the agents demonstrate significant improvement in their strategies. Hiders become more adept at finding effective hiding spots and evading the seeker, while seekers develop better pursuit techniques and search patterns.

8. Experiments:

To explore alternative approaches and further enhance the game, we can consider the following experiments:

8.1 Heuristic-Based Approach:

Implement and evaluate the effectiveness of heuristic-based strategies, such as hiders prioritizing areas with more obstacles and seekers focusing on areas with fewer obstacles.

8.2 Deep Q-Networks:

Investigate the use of Deep Q-Networks (DQNs) to approximate Q-values using neural networks, potentially leading to more complex and adaptive strategies.

8.3 Monte Carlo Methods:

Explore the application of Monte Carlo methods, where agents learn from simulating multiple episodes and averaging rewards to estimate action values.