Reinforcement Learning for Simplified Poker: A Study on Adaptive Strategies in Leduc Hold'em

Li Ge
2022533011
`lige2022@`

Long Yuxuan
2022533034
`longyx2022@`

Zeng Yihe
2022533161
`zengyh2022@`

Abstract

This paper explores the application of reinforcement learning in a multi-agent environment, focusing on decision-making strategies within the game of Leduc Hold'em poker. The project involves the design and implementation of a flexible environment class to facilitate agent interaction, state tracking, and modular support for various learning algorithms. We develop and evaluate multiple reinforcement learning agents, including Q-Learning, model-based methods, and Bayesian approaches, to address the dynamic and competitive nature of the game. Through adversarial training, agents iteratively refine their policies by learning from each other, enhancing their decision-making capabilities. Experimental results demonstrate the effectiveness of these methods in achieving strategic adaptability and highlight the potential of reinforcement learning to solve complex multi-agent problems. This study provides valuable insights for advancing research in multi-agent systems, game theory, and artificial intelligence.

1 Introduction

Poker is a classic example of an imperfect information game, requiring players to make decisions under uncertainty while anticipating their opponents' strategies. This complexity makes poker a valuable domain for testing artificial intelligence (AI) techniques, especially in reinforcement learning and multi-agent systems. Among various poker variants, **Leduc Hold'em** stands out for its simplified rules and strategic depth, offering a manageable environment for studying decision-making under incomplete information.

In this project, we explores the development of AI agents for Leduc Hold'em, focusing on their ability to learn and adapt to dynamic environments. We introduce four types of agents: Random Agent, Model-Based Agent, Q-Learning Agent, and Bayesian Agent. Each agent follows a distinct decision-making paradigm.

2 Background and Problem Formulation

2.1 Original Problem

Leduc Hold'em is a simplified poker game often used in artificial intelligence and game theory research due to its strategic depth and manageable complexity. The game is designed to simulate the core elements of poker, such as betting strategies, hidden information, and decision-making under uncertainty.

The game uses a small deck of six cards, consisting of three unique ranks (e.g., Jack, Queen, King), with two cards for each rank. Two players participate in the game, and the objective is to maximize the number of chips won by making optimal decisions. Each player is dealt one private card, which is hidden from their opponent, and one public card is revealed during the game to influence decision-making.

Leduc Hold'em consists of two rounds:

- **First Round:** Each player is dealt one private card, which only they can see. Players then take turns deciding whether to bet, call, or fold based on their private card and their assumptions about the opponent's possible card.
- **Second Round:** A single public card is revealed. Players again take turns deciding their actions based on their private card, the public card, and the opponent's previous actions. This round concludes with a showdown if no player folds.

The winner of the game is determined by comparing the ranks of the cards:

- If one player folds during any round, the other player automatically wins the pot (the total chips bet by both players).
- If both players reach the end of the second round without folding, the player with the highest combination of private and public cards wins the pot.

One of the key challenges in Leduc Hold'em is its **partial information**. Players can only see their private card and the public card but have no knowledge of the opponent's private card. This makes it essential to make decisions based on probabilities and predictions about the opponent's strategy. Additionally, players must balance between **exploitation**—capitalizing on strong hands—and **bluffing** to mislead the opponent and force favorable outcomes.

Overall, Leduc Hold'em captures the essence of decision-making in poker, including uncertainty, risk management, and dynamic strategy adjustment. These features make it an ideal testbed for reinforcement learning and multi-agent systems.

2.2 Problem Formulation

This project focuses on developing an artificial intelligence agent for a simplified version of Texas Hold'em poker, known as **Leduc Hold'em**. Leduc Hold'em is a widely studied variant in the field of game theory and artificial intelligence due to its manageable complexity and strategic depth, which make it suitable for research on decision-making under uncertainty and multi-agent interactions.

In Leduc Hold'em, the game is played using a small deck of six cards, with two suits and three ranks, such as King, Queen, and Jack. twictwic This small deck size significantly reduces the state space of the game while maintaining the core challenges of poker, including incomplete information and probabilistic reasoning.

The game typically involves two players, making it ideal for head-to-head competitions. Each player starts with an equal number of chips and competes to maximize their winnings through strategic betting and decision-making. The game proceeds in multiple stages, beginning with the *pre-flop phase*. During this phase, each player is dealt one private card aka hole hand, which remains hidden from the opponent. This hidden information creates the primary uncertainty that players must account for in their strategies.

After the private cards are dealt, the game moves to the *flop phase*, where a single public card is revealed. This card is shared by both players and can be combined with their private cards to form a hand. The public card introduces new information into the game, allowing players to adjust their strategies based on the potential hand combinations available to both themselves and their opponent.

The betting rounds are a critical component of Leduc Hold'em. There are two betting rounds: one after the private cards are dealt and another after the public card is revealed. In each betting round, players can take one of five possible actions:

- **Bet**: Place a wager based on the perceived strength of their hand.
- **Call**: Match the opponent's bet to stay in the game.
- **Raise**: Increase the amount of the current bet to exert pressure on the opponent.
- **Fold**: Forfeit the current hand and concede the pot to the opponent.

- **Check**: Pass the action to the next player without placing a bet, while still maintaining the right to call, raise, or fold later in the round

Players must balance between aggressive betting, which can force the opponent to fold, and conservative strategies, which reduce the risk of losing chips. The limited betting rounds and fixed chip amounts further constrain the decision-making process, requiring precise calculations and well-timed actions.

If neither player folds during the betting rounds, the game proceeds to the *showdown phase*, where both players reveal their private cards. The winner is determined based on the hand strength, with higher-ranked combinations taking precedence. In Leduc Hold'em, a pair (one private card and the public card of the same rank) beats a high card, and ties occur as far as two player share the same private card. When both players have a high-card hand, they compare their cards from highest to lowest. For example, if the public card is a K, a player with a Q in their hole cards beats a player with a J in theirs.

The simplified structure of Leduc Hold'em retains the essential strategic elements of Texas Hold'em while significantly reducing the complexity of the game. The reduced card space makes it easier for players to estimate their opponent's possible hands, yet the hidden private cards maintain the challenges of decision-making under incomplete information. This combination of simplicity and depth makes Leduc Hold'em an excellent platform for studying reinforcement learning and game-theoretic strategies.

In this project, we leverage the unique properties of Leduc Hold'em to develop and evaluate artificial intelligence agents. By focusing on this simplified rule set, we aim to explore the effectiveness of various reinforcement learning methods, including value-based, model-based, and probabilistic approaches, in handling dynamic decision-making and partial information environments.

3 Random Agent

3.1 Overview

The Random Agent is the simplest form of artificial intelligence in this project. It interacts with the Leduc Hold'em environment by selecting one action at random from the set of legal actions provided by the game state. The agent does not evaluate the game state, opponent strategy, or future consequences of its actions. Its primary purpose is to:

- Serve as a baseline to compare the performance of other, more complex agents.
- Test the robustness and correctness of the game environment.
- Provide a controlled and non-adaptive opponent during training and evaluation of other agents.

3.2 Algorithm

The Random Agent operates by adhering to a simple algorithm:

1. Obtain the current game state, which includes the set of legal actions.
2. Randomly select an action from the legal actions using a uniform probability distribution.
3. Execute the selected action in the environment.

This straightforward algorithm ensures that the agent always selects a valid action while introducing no strategic bias.

3.3 Implementation

The implementation of the Random Agent is straightforward and leverages Python's random sampling functionality. The following pseudocode illustrates the agent's decision-making process:

Algorithm 1 Random Agent Decision Process

```
1: **Input:** Current game state $S$
2: Extract the set of legal actions $A \leftarrow S$.legal_actions
3: Randomly select an action $a \sim$ Uniform$(A)$
4: **Return:** Selected action $a$
```

4 Model-Based Agent

Unlike model-free methods such as Q-Learning, which directly learn a value function or policy from experience, a Model-Based Agent *explicitly* constructs and maintains an internal model of the environment's dynamics. Specifically, it learns or estimates:

- A **transition model**, capturing how the environment state evolves in response to actions.
- A **reward model**, indicating the immediate reward for each state-action pair.

By integrating these models, the agent can *simulate* future outcomes of different actions and thereby plan ahead to select actions that maximize long-term return.

4.1 Core Principles of Model-Based Reinforcement Learning

In model-based RL, the agent maintains:

- **Transition model:** $T(s, a, s')$, representing the probability of moving from state s to state s' when taking action a.
- **Reward model:** $R(s, a)$, use the chip we bet in the current round to represent the immediate reward.

Once these models are learned from data, the agent can **plan** by simulating different actions and predicting future returns.

4.2 Algorithm

The Model-Based Agent in Leduc Hold'em typically proceeds as follows:

1. **Initialize**: the transition model T and reward model R to empty or uniform distributions.
2. **Collect experience**: Run entire episodes in the Leduc environment. After each action:
   - Observe the current state s, chosen action a, immediate reward r, and the resulting next state s'.
   - **Update the model**: Store counts in T and use the chip we bet in this round to update R.
3. **Estimate transition probabilities and rewards**: After collecting enough data,

$$P(s\prime \mid s, a) \approx \frac{T(s, a, s\prime)}{\sum_{s\prime\prime} T(s, a, s\prime\prime)}, \quad \hat{R}(s, a) \approx \text{mean}\big(R(s, a)\big).$$

   (a) **Plan or simulate**: When choosing an action in state s, the agent proceeds as follows:
      i. Use the transition model derived from past experiences.
      ii. Use the learned average reward estimate $\hat{R}(s, a)$ to approximate the immediate return of each action a.
      iii. Apply a softmax operation to the average rewards $\hat{R}(s, a)$ for all legal actions to obtain a probability distribution (if multi-step lookahead is not required, one can skip more complex simulation or search).
      iv. • **Training mode**: Randomly sample an action from the softmax distribution.
          • **Evaluation mode**: Select the action with the highest average reward, i.e., $\arg\max_a \hat{R}(s, a)$.

(b) **Repeat**: Continue collecting gameplay data to further refine (or fine-tune) the transition and reward models, thereby improving strategy quality over time.

Algorithm 2 Model-Based Agent Training Algorithm

```
1:  Initialize Model: $T(s, a, s') \leftarrow 0$, $R(s, a) \leftarrow []$ for all $(s, a, s')$.
2:  for each training episode do
3:      Reset environment, get initial state $s$.
4:      while episode not terminated do
5:          Choose action $a$.
6:          Execute $a$ in $s$, observe $r, s'$.
7:          Update the counts: $T(s, a, s') \leftarrow T(s, a, s') + 1$.
8:          Append $r$ into $R(s, a)$.    // store reward sample
9:          $s \leftarrow s'$.
10:     end while
11: end for
12: Compute approximate distributions:
```

4.3 Implementation Details

1. **State Representation and Action Selection**

   Each state is obtained by converting the observations returned by the RLCard environment (such as the player's private cards, public cards, betting rounds, and chip information) into a tuple to enable hashing. Each action is represented by an integer index (e.g., `fold`, `call`, `raise`). In each training round, the agent gathers trajectory data from the environment, namely the complete record of each episode (including every state, action, reward, and the subsequent state). Using these trajectory data, we can update our model.

2. **Updating the Transition and Reward Models**

   After each game ends, the agent iterates over all state-action-reward-next-state tuples for that game and updates its model based on the actual rewards observed. We maintain two dictionaries in the updating process:

   - `transition_model`: Records the transition frequencies for each state-action pair.
   - `reward_model`: Records the reward history for each state-action pair.

   The updating procedure is carried out by the `_update_model` method, which traverses each state-action pair and updates the model based on the reward information.

3. **Decision Making and Strategy Selection**

   When making decisions, the agent uses its current state-action-reward model and adopts a Softmax strategy to choose actions. This approach assigns a probability to each action based on its expected reward (calculated as the running mean of historical rewards). The agent then randomly selects an action from the set of legal actions according to these probabilities.

   During training, the agent iteratively optimizes its decision-making strategy by simulating multiple games. In each decision step, it calculates the expected reward for each legal action and applies Softmax sampling, thereby ensuring adequate exploration.

4.4 Experimental Observations

We trained the Model-Based Agent for several thousand hands of Leduc Hold'em against various baseline opponents. While this approach can ultimately yield more stable planning than a purely random strategy once the transition and reward models converge, the agent's performance heavily depends on the types of opponents it trains against, which in turn makes its overall learning process more volatile. In other words, the style and strategy of the training opponents substantially shape the agent's behavior, causing training outcomes to vary across different matchups. Moreover, because the environment is partially observable (the agent cannot see the opponent's cards) and the game tree can branch significantly, deeper lookahead or more sophisticated opponent belief modeling can further improve performance. However, before the agent's models become sufficiently accurate, training can be relatively unstable, as it must accumulate enough experience to reliably predict transitions and rewards.

Overall, the Model-Based Agent demonstrates how learning explicit models of the environment allows for multi-step reasoning and adaptable strategies in Leduc Hold'em, distinguishing it from simpler reactive methods.

5 Q-learning Agent

The Q-Learning Agent applies model-free reinforcement learning to develop an effective decision-making strategy in the Leduc Hold'em environment. Unlike random agents, which select actions blindly, the Q-Learning Agent learns from interactions with the environment to optimize its policy over time. This is achieved by estimating the long-term reward for each state-action pair and using this knowledge to make informed decisions.

5.1 Core Principles of Q-Learning

Q-Learning is based on the concept of a Q-function, $Q(s, a)$, which represents the expected cumulative reward for taking action a in state s and following the optimal policy thereafter. The agent updates this Q-function iteratively during training using the Temporal Difference (TD) learning rule:

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

where:

- α is the learning rate, controlling how quickly the agent updates its estimates.
- γ is the discount factor, determining the importance of future rewards relative to immediate rewards.
- r is the reward received for performing action a in state s.
- s' is the state reached after taking action a.

This equation balances the immediate reward with the estimated value of future actions, ensuring that the agent learns to prioritize long-term gains.

5.2 Algorithm

The Q-Learning Agent interacts with the environment in episodes, each consisting of multiple games. In each episode, it performs the following steps:

1. Observe the current state s and identify the set of legal actions.
2. Select an action a using the ϵ-greedy policy:
    - With probability ϵ, choose a random action to explore the environment.
    - With probability $1 - \epsilon$, choose the action that maximizes $Q(s, a)$.
3. Execute the action a, observe the resulting state s' and the reward r.
4. Update the Q-value for the state-action pair (s, a) using the TD rule.
5. Repeat until the episode terminates.

The training process consists of multiple episodes, during which the agent iteratively refines its Q-values. Once training is complete, the agent uses the learned Q-table to make decisions without further updates.

Algorithm 3 Q-Learning Agent Training Algorithm

```
1: Initialize Q-table $Q(s, a)$ to zero for all state-action pairs.
2: for each training episode do
3:     Initialize state $s$.
4:     while episode not terminated do
5:         Select action $a$ using $\epsilon$-greedy policy.
6:         Execute action $a$, observe reward $r$ and next state $s'$.
7:         Update Q-value:
               $$Q(s, a) \leftarrow Q(s, a) + \alpha \cdot \left[ r + \gamma \cdot \max_{a'} Q(s', a') - Q(s, a) \right].$$
8:         Set $s \leftarrow s'$.
9:     end while
10: end for
```

5.3 Implementation Details

In our implementation, the Q-Learning Agent uses a Q-table stored as a dictionary, where keys are state-action pairs and values are Q-values. The training is conducted in an environment with a fixed seed to ensure reproducibility. Hyperparameters such as the learning rate ($\alpha = 0.01$), discount factor ($\gamma = 0.8$), and exploration rate ($\epsilon = 0.1$) were chosen through experimentation to balance learning efficiency and stability.

The agent alternates between training and evaluation phases. During training, it explores the environment extensively to gather diverse experiences. For evaluation, the agent uses a fixed policy derived from the Q-table and competes against baseline agents, such as the Random Agent and Rule-Based Agent.

6 Rule Agent

The Rule Agent mimics human intuition by following predefined, simple rules to evaluate hand strength and make decisions in Leduc Hold'em. When humans play poker, they tend to maximize the pot size when holding strong hands and minimize losses when holding weak hands. Similarly, the Rule Agent uses straightforward heuristics based on experience and practice to decide its actions based on the private and public cards available in the current game state.

The Rule Agent makes decisions solely based on the cards in hand and on the table without requiring any training. Before the public card is revealed (pre-flop), the agent chooses to **raise** if it holds a **King (K)**. Otherwise, it chooses **call** or **check** to gather more information about the game state. After the public card is revealed (post-flop), if the private card matches the public card, forming a pair, the agent chooses to **raise**. Otherwise, the Rule Agent evaluates its hand strength as a single card: if the private card is **King (K)**, it randomly decides between **call** and **check** (or **fold**) with equal probability; otherwise, it opts for **check** (or **fold**).

6.1 Decision Rules

The decision-making process of the Rule Agent can be divided into two phases:

- **Pre-flop:** If the private card is **King (K)**, the agent selects **raise**. Otherwise, it chooses **call** if available; if not, it selects **check**.
- **Post-flop:**
  - If the private card matches the public card, forming a pair, the agent selects **raise**.
  - Otherwise:
    - If the private card is **King (K)**, the agent randomly decides between:
      - **call** with a 50% probability.
      - **check** (or **fold**) with a 50% probability.
    - If the private card is not **King (K)**, the agent selects **check** (or **fold**).

6.2 Algorithm

The following pseudocode summarizes the decision-making process of the Rule Agent:

Algorithm 4 Rule Agent Decision-Making Process

```
1: Input: Game state $S$ with private card $hand$, public card $public\_card$, legal actions $legal\_actions$.
2: Output: Chosen action $action$.
3: if $public\_card$ is available then
4:     if $hand.rank == public\_card.rank$ then
5:         $action \leftarrow$ raise
6:     else
7:         if $hand.rank == K$ then
8:             $rand \leftarrow$ random number in $[0, 1)$
9:             if $rand < 0.5$ then
10:                $action \leftarrow$ check
11:            else
12:                $action \leftarrow$ call
13:            end if
14:        else
15:            $action \leftarrow$ check
16:        end if
17:    end if
18: else
19:    if $hand.rank == K$ then
20:        $action \leftarrow$ raise
21:    else
22:        if call is in $legal\_actions$ then
23:            $action \leftarrow$ call
24:        else
25:            $action \leftarrow$ check
26:        end if
27:    end if
28: end if
29: return $action$
```

6.3 Analysis

The rule agent operates on simple predefined rules without requiring any training. It uses the information available in the game state, such as the private hand and public cards, to select the next action. These rules are intuitive and align with basic human decision-making strategies in poker. Despite its simplicity, the Rule Agent demonstrates surprisingly effective performance in practice, making it a solid benchmark for evaluating other agents. The detailed results will be discussed in the experimental analysis section.

The primary strength of the Rule Agent lies in its straightforward logic, which mirrors human intuition. It chooses aggressive actions like **raise** when holding strong hands, aiming to maximize the pot size. Conversely, it opts for conservative actions like **check** or **fold** when holding weaker hands, minimizing potential losses. This balance between aggression and caution enables the Rule Agent to function as a competitive opponent in Leduc Hold'em, despite the absence of any learning or adaptability.

By following these predefined strategies, the Rule Agent highlights the importance of hand evaluation in poker while serving as a useful baseline for comparison against more complex agents. Its effectiveness in real-world scenarios will be analyzed further in subsequent sections through performance metrics and direct comparisons with learning-based agents.

7 Bayesian Agent

Bayes Agent is an intelligent agent based on Bayesian updating strategies, which utilizes probability and statistical methods to infer opponents' behaviors and hand distributions in reinforcement learning environments. The core idea is to continuously update the opponent's strategy model using their actions and prior knowledge (such as the distribution of their hands) to make optimal decisions for itself.

In Texas Hold'em, the commonly used Game Theory Optimal (GTO) strategy aims to construct a balanced strategy that cannot be easily exploited by opponents. Bayesian Agent dynamically adjusts the estimation of the opponent's hand and flexibly applies probabilistic models, enabling it to progressively approximate GTO strategies.

"Unlike conventional GTO solvers, in the rules of Leduc Hold'em, due to its significantly smaller state space compared to the complexity of Texas Hold'em, Bayes Agent can perform online real-time calculations without relying on offline pre-computation like traditional GTO solvers. This feature further enhances the practicality and efficiency of Bayes Agent in simplified environments."

Compared to traditional fixed strategies or heuristic methods, Bayes Agent excels in handling incomplete information scenarios. Its core advantage lies in updating the opponent's hand distribution through behavioral observation, combining this with expected value calculations. This allows the agent to make decisions that balance offensive and defensive strategies, thereby achieving a closer approximation to GTO equilibrium strategies.

7.1 Core Elements of Bayesian Agent

EV (Expected Value) is the core element of the Bayesian Agent. EV represents the expected payoff of the current action, calculated based on the player's hole cards, public cards, and the range of the opponent's hole cards.

$$EV = \mathbb{P}(win) \times pot + \frac{\mathbb{P}(tie) \times pot}{2} - \mathbb{P}(loss) \times \text{chip-to-bet}$$

which indicates that we only care about benefit in the current round no matter the chip we bet in former round.

This approach allows the agent to adapt its actions depending on the game context, making its overall strategy more robust. By dynamically recalculating EV throughout the game, the agent can execute a mixed-frequency strategy that integrates bluffing, value betting, and folding.By leveraging Bayesian updates and continuously refined EV estimates, the agent balances aggression and caution, ensuring a well-rounded decision-making process that aligns with Game Theory Optimal (GTO) principles.

In a Bayesian agent, we assume that the opponent is also an experienced player who makes decisions based on optimal choices or a certain optimal strategy. Therefore, we can infer the opponent's range of hole cards based on their actions. Using the information from our own hole cards and the public cards, we can establish a prior probability distribution for the opponent's hole cards.

When observing the opponent's actions, we calculate their EV (expected value) under the current distribution of hole cards. By evaluating the EV and their chosen action, we determine the likelihood of the opponent taking that specific action. Finally, using Bayes' theorem, we update the prior to derive the posterior probability distribution of the opponent's hole cards. This process allows the Bayesian agent to dynamically adjust its understanding of the opponent's strategy and make more informed decisions.

$$P(H|A) = \frac{P(A|H) \cdot P(H)}{\sum_{H'} P(A|H') \cdot P(H')}$$

Where:

- $P(H|A)$: The posterior probability of the opponent holding hand H after observing action A.
- $P(H)$: The prior probability of the opponent holding hand H.
- $P(A|H)$: The likelihood of the opponent choosing action A given they hold hand H.
- The denominator $\sum_{H'} P(A|H') \cdot P(H')$: The normalization term, representing the total probability across all possible hands.

7.2 Algorithm

Algorithm 5 Bayesian Agent Decision-Making Process

```
1: Input: Game state $S$ with private card $hand$, public card $public\_card$, legal actions $legal\_actions$, opponent action history.
2: Output: Chosen action $action$.
3: Initialize: Prior hand distribution for opponent $P(H) = [P(J), P(Q), P(K)]$.
4: Update: Calculate posterior distribution for opponent's hand based on observed action history using Bayes' Theorem.
5: Compute: Compute expected value (EV) for each possible action based on $hand$, $public\_card$, and opponent's posterior hand distribution.
6: Choose: Calculate action probabilities using softmax function on EV values.
7: Action Selection:
8: if Opponent's action is observed then
9:     Update opponent's hand distribution $P(H|A)$ using Bayes' Theorem based on the action $A$.
10: end if
11: Select action: Choose action $action \leftarrow$ sample from action probabilities.
12: return $action$
```

7.3 Analysis

The Bayesian Agent demonstrates exceptional decision-making capabilities in environments with incomplete information. By leveraging Bayesian inference, the agent dynamically updates its opponent's strategy model, enabling responsive decisions while reducing the risk of exploitation. Through iterative calculations of Expected Value (EV), the Bayesian Agent effectively integrates bluffing, value betting, and folding into a cohesive strategy, adapting to evolving game dynamics.

Additionally, the Bayesian Agent infers opponents' hand distributions based on observed actions and employs probabilistic models to approximate Game Theory Optimal (GTO) strategies. This inference ability is particularly effective in games like Leduc Hold'em, which have smaller state spaces, allowing the agent to perform real-time online calculations and quickly adjust strategies, maintaining competitiveness in complex games.

Comparison with Other Reinforcement Learning Algorithms

Compared to Model-Based Methods:

- Model-based methods rely on learning the environment's dynamics and reward functions to plan decisions, making them well-suited for deterministic environments.
- In contrast, the Bayesian Agent focuses on observing and inferring opponent behavior, dynamically adjusting decisions without explicitly modeling the environment's dynamics, making it more robust in incomplete information games.

Compared to Q-Learning:

- Q-Learning updates the value of state-action pairs (Q-values) through trial-and-error exploration but struggles with high exploration costs in large state spaces or incomplete information environments.
- The Bayesian Agent, leveraging probabilistic reasoning and prior knowledge, achieves efficient decision-making even under limited exploration by dynamically adjusting strategies.

Limitations

The current design of the Bayesian Agent focuses primarily on modeling opponent behavior within a single game, without considering their long-term strategy or playing style. This limitation may result in locally optimal decisions in the short term but fails to account for global strategies, impacting performance in multi-game scenarios.

For instance, the Bayesian Agent cannot identify opponents' long-term tendencies, such as aggressive or conservative play styles, leading to potential missteps against different types of opponents. Future improvements could include extending the model to capture multi-game tendencies and learning long-term opponent behavior through repeated interactions, thereby enhancing performance in global gameplay scenarios.

Conclusion

The Bayesian Agent strikes a balance between theoretical rigor and practical applicability. By integrating Bayesian inference and EV computation, the agent makes robust decisions in environments with incomplete information, progressively approximating GTO strategies.

Key Advantages:

- Dynamically updates opponents' hand distributions for real-time precise decisions.
- Combines probabilistic reasoning with EV computation, providing high adaptability.
- Outperforms traditional algorithms (e.g., model-based methods and Q-Learning) in addressing incomplete information games.

Despite its limitations in modeling long-term strategies, the Bayesian Agent can further enhance its capabilities through improvements, making it a powerful tool for complex environments with incomplete information.

8 Experimental Results and Analysis

We use two distinct criteria to analyze and compare the performance of the models.

1. **Match-Based Evaluation**: Inspired by the competition format in Texas Hold'em, we measure performance by recording the number of times each player wins 200 big blinds first across 10 matches.
2. **Profitability Assessment**: To evaluate profitability, we use the net number of big blinds won over 100 hands as a metric to assess the model's performance.

	Random	**Model-based**	**Q-learning**	**Rule-based**	**CFR**
Random	–	0	0	0	0
Model-based	10	–	10	10	0
Q-learning	10	0	–	9	0
Rule-based	10	0	1	–	10
CFR	10	10	10	0	–

Table 1: Pairwise results of five agents over 10 matches each. Entry (i, j) denotes how many matches agent i won against agent j.

	Random	**Model-based**	**Q-learning**	**Rule-based**	**CFR**
Random	–	-62.25	-44.50	-23.75	-35.75
Model-based	62.25	–	-26.00	14.00	-21.00
Q-learning	44.50	26.00	–	33.50	6.00
Rule-based	23.75	-14.00	-33.50	–	6.00
CFR	35.75	21.00	0.50	-6.00	–

Table 2: Pairwise average results of five agents' rewards of 100 hands. Entry (i, j) denotes how many rewards agent i won against agent j.

Table 1: Pairwise Results Over 10 Matches

- *Interpretation:* The table indicates how many matches each agent won against every other agent (with 10 matches in each pairing).

  - **Random** lost all matches against every opponent.
  - **Model-based** achieved full victories over **Random** but lost all matches to **Q-learning**.
  - **Q-learning** also won all of its matches against **Random**, yet performed poorly against **Rule-based** and **Model-based**.
  - **Rule-based** won ten matches each against **Q-learning** and **CFR**, but lost to **Model-based**.
  - **CFR** secured full victories over **Random** and **Rule-based**, but had no wins against **Model-based** or **Q-learning**.

Table 2: Pairwise Average Rewards Over 100 Hands

- *Interpretation:* The table shows the average reward obtained by each agent after 100 hands against each opponent.
  - **Random** receives negative rewards against all agents, indicating generally poor performance.
  - **Model-based** achieves high rewards against **Random** and **Q-learning**, but posts weaker results against **Rule-based** (14.00) and **CFR** (-21.00).
  - **Q-learning** gains positive returns when facing **Random** (44.50) and **Model-based** (26.00), but suffers substantial losses against **Rule-based** (-33.50) and only slightly positive outcomes against **CFR** (6.00).
  - **Rule-based** sees favorable results against **Random** (23.75), yet negative returns in matchups with **Model-based** and **Q-learning**.
  - **CFR** performs relatively well overall, posting positive averages against **Random** and **Q-learning**, while slightly negative against **Rule-based** (-6.00) and notably negative against **Model-based** (-21.00).

Key Observations

- The **Model-based** agent performs strongly in many scenarios, particularly versus **Random** and **Q-learning**. However, it is less effective against **Rule-based** and **CFR**, suggesting it may require more targeted strategies for those specific opponents.
- **Random** is the weakest agent, consistently producing negative returns against all others.
- **Q-learning** exhibits decent performance in certain matchups (e.g., against **Random** and **Model-based**) but struggles significantly with **Rule-based** and **CFR**.
- **Rule-based** manages good gains against **Random**, but suffers in encounters with more adaptive or learning-based agents.
- **CFR** demonstrates relatively balanced outcomes and can compete strongly in several pairings. However, it still shows weaknesses against some specific agents (e.g., **Model-based**).

9 Conclusion

n this project, we explored various reinforcement learning techniques and rule-based strategies within the context of Leduc Hold'em—a simplified poker game that preserves the essential dynamics of hidden information, betting, and bluffing. We introduced and investigated multiple agents, including a Random Agent, Model-Based Agent, Q-Learning Agent, Bayesian Agent, and a handcrafted Rule-Based Agent. Each agent exhibited distinct strengths in handling incomplete information and adapting to opponent behaviors:

Random Agent: served as a baseline with no strategic bias, illustrating the environment's core functionality.

Model-Based Agent explicitly learned transition and reward models, showing the potential of planning and simulation but also requiring substantial interaction data to achieve stable performance.

Q-Learning Agent leveraged a value-based, model-free learning approach to iteratively refine its policy through temporal difference updates.

Rule-Based Agent provided a heuristic-driven strategy inspired by simple human poker logic, performing surprisingly well despite its lack of learning.

Bayesian Agent dynamically updated its beliefs regarding the opponent's possible cards, integrating probabilistic reasoning for decision-making and demonstrating strong performance in uncertain scenarios.

Experimental results demonstrated that agents employing more sophisticated inference or learning mechanisms tended to outperform simpler ones. However, each approach also highlighted trade-offs between complexity, convergence speed, and robustness in partially observable settings. Furthermore, outcomes depended heavily on the nature of opponents, underscoring the importance of diverse training experiences and robust adaptive strategies.

Overall, this study underscores the value of Leduc Hold'em as a compelling environment for investigating multi-agent reinforcement learning, game theory, and decision-making under incomplete information. Future work may focus on integrating opponent modeling, extending multi-step lookahead, and combining multiple strategies (e.g., Bayesian updates with deep learning methods) to handle more complex or dynamic versions of poker and other multi-agent tasks.

External Resources

https://github.com/datamllab/rlcard

We utilized these external resources to build a gaming environment, which includes the game flow, the determination of victory or defeat, and the assessment of winning chips.