WiLDCARD: Winning Liar's Dice with Calculative Agents and Reinforced Decision-Making

Stanford University - CS 238: Decision Making Under Uncertainty

Eric Lee
Department of Computer Science
Stanford University
ericlee7@stanford.edu

Ryan Lian
Department of Computer Science
Stanford University
ryanlian@stanford.edu

Brian Xu
Department of Computer Science
Stanford University
brianxu@stanford.edu

Abstract

Liar's Dice is a complex game involving strategy, probability, and bluffing, which traditionally presents a significant challenge for AI agents due to its bidding structure and game flow. In this paper, we introduce **WiLDCARD**, a reinforcement learning agent designed to achieve superhuman performance in a variant of Liar's Dice requiring deeper strategic analysis, where all dice rolled as 1s serve as wildcards. To address the intractability of the game's state space, we leveraged Deep Q-Networks for function approximation, training a neural network to estimate optimal actions effectively. Training began with fixed-policy dummy agents, followed by self-play to refine strategies. Reward shaping and opponent belief modeling were implemented to encourage favorable behaviors and increase win rates. Results showed the final agent was able to win against all combinations of 1 and 2 players, demonstrating the agent's ability to learn a strong basic strategy while adapting to opponents. We conclude with a discussion of limitations and future work.

1 Problem

Our objective is to develop a reinforcement learning agent capable of devising a winning strategy for Liar's Dice against multiple baseline-strategy opponents. Liar's Dice, or Dudo, is a game of incomplete information in which players roll several dice and place bids on the total quantity of specific dice values across all rolls. Limited knowledge of other players' dice introduces strategic uncertainty. Players lose dice when their bids are successfully challenged or when they incorrectly call bluffs, and the last player with dice remaining wins. To increase the complexity and uncertainty within the game, we focus on a variant where rolls of 1 are treated as wildcards, counting as any value. This addition significantly expands the strategic depth of the game.

Specifically, this game poses intriguing challenges in game theory and artificial intelligence due to its elements of incomplete information, environmental stochasticity, and intractably large game tree. To enable exact computation of Nash equilibria, prior work addressing Liar's dice has primarily focused on reduced versions of the game with either 1 opponent or limited dice per player. However, this approach leaves a gap in exploring the more complex scenarios common in real-world gameplay with multiple agents, many dice, and varied strategies. We address this gap by developing a reinforcement learning approach to learn and approximate strategies against both 1 and 2 opponents with 5 dice, while incorporating the wildcard variation.

2 Related Work

Liar's Dice has been studied extensively using reinforcement learning, with many approaches addressing the intractability of its state space through simplifications. Greaves and Tay (2018) reduced the game to two players with five dice, applying Q-learning to this simplified setting, while Dodge (2017) used fixed-policy dummy agents to simulate gameplay and generate data for Q-learning. While insightful, these approaches fail to comprehensively capture the complexities of real-world gameplay, which often involves varied scenarios and multiple players.

To build upon these methods, we note that value function approximation techniques, particularly neural networks, have proven effective for handling large state spaces. For instance, counterfactual regret minimization and neural network-based Q-value approximation have successfully produced Liar's Dice agents (Ahle, 2022). While promising, we aim to enhance these methods by incorporating heuristics to constrain the state space, enabling faster convergence to reasonable policies. Specifically, Deep Q-Networks (DQNs) have shown notable success in domains such as Atari (Mnih et al., 2013) and poker (Heinrich and Silver, 2016). These successes, along with the structural similarities between Liar's Dice and these games—such as turn-based actions and stochastic elements—inform our decision to use Q-learning with a neural network for Q-value approximation.

Belief modeling has also been explored to improve decision-making in bluffing games like poker (Zhang et al., 2024) and Skull (Lee, 2017). However, the computational cost of Bayesian updating not only hinders scalability but often yields diminishing returns in terms of performance improvements. To address this, we propose a simpler belief heuristic that updates dynamically during gameplay and feeds directly into the DQN, achieving a balance between computational efficiency and strategic depth. By integrating DQN-based value approximation, belief modeling, and state-space heuristics, our work addresses the challenges posed by the wildcard variant of Liar's Dice, providing a scalable and effective solution for this complex domain.

3 Approach

To address the problem, we implemented the following techniques to build our agent:

3.1 Interpreting Liar's Dice as a Partially Observable Markov Decision Process (POMDP)

Given the inherent sequential decision-making component of Liar's Dice with probabilistic outcomes, we modeled the game as a POMDP. Concretely, we defined our states, rewards, actions, and observations as follows:

- $\mathcal{S}$ is the set of states, defined as all tuples of the form (number of dice between all players, number of dice of our agent, the results our agent rolled of form $[\#, \#, \#, \#, \#, \#]$, the current bid, belief of opponent dice rolls, boolean for winning the game).
- $\mathcal{A}$ is the set of actions to be taken by the agent, which consist of calling bluff on the current bid, or placing a bid of N many Ks, where $N \in \{1, \ldots, \text{total number of dice across players}\}$ and $K \in \{1, 2, 3, 4, 5, 6\}$. Not all actions are possible depending on current bid.
- Rewards $\mathcal{R}$ were defined as follows (prior to reward shaping): winning the game (+5000), losing the game (-5000), placing a bid that is not called (+100), bluffing and getting called (-500), incorrectly calling bluff (-1000), correctly calling bluff (+500).
- The space of observations $\mathcal{O}$ include all possible bids from opponents, which provide a signal as to the dice they hold within each round.

3.2 Deep Q-Networks (DQN)

As seen in the formulation of $\mathcal{S}$, the state space is intractably large. Take the simple case when 2 players each have 5 dice: there are already $({}_{10}C_5)^2$ configurations of dice rolls with 76 possible actions, totaling 4826304 states. Then, when we consider that each player could have between 0 to 5 dice for up to 3 players, it becomes clear that it is infeasible to enumerate each state in $\mathcal{S}$.

Therefore, we implement deep Q-networks (DQN) to approximate action values with neural networks (Mnih et al., 2013). In particular, we initialize two neural networks which predict Q-values: a training network Q_{train} which is constantly updated during training, and a target network Q_{target} that is periodically updated from Q_{train} to prevent Q-values from oscillating during training. To update Q-values, we apply the Bellman equation with Q_{train} to minimize loss L with

$$L = \left(r + \gamma \cdot \max_{a' \in A} Q_{\text{target}}(s', a') - Q_{\text{train}}\right)^2$$

such that eventually the target network Q_{target} accurately represents the maximum discounted reward for taking action a from state s.

3.3 Replay Buffer and Simulations

To allow the agent to explore strategies and implement updated policies as it plays games, we implemented a replay buffer R: a log of (state, action, reward, next state) tuples from a given game simulation. After the agent plays a set number of simulated games, the target network Q_{target} is updated with a set number of tuples sampled from R. This replay buffer helps shuffle the data presented, stabilize training, and mitigate correlations between consecutive experiences.

3.4 α-Greedy Data Generation

To guide the agent toward a sensible basic policy at the start of training, we defined a heuristic rule-based policy π and had the agent follow policy π (instead of choosing an action from Q_{target}) with probability α. This technique allows the agent to immediately explore and train on promising reward tuples. Note that α is decreased over time so the agent eventually follows its own policy.

3.5 ε-Greedy Exploration

After the agent learned a sensible baseline policy, we implemented ε-greedy exploration, where the agent would choose an action uniformly at random with probability ε. This technique allows the agent to explore the state and action space, potentially unlocking new strategies from such data.

3.6 Dummy Agents and Self Play

In order to simulate games and create baselines for comparison, we created dummy agents with different rule-based policies. This allowed for the learning of basic gameplay and early data gathering. Then, after the agent had refined its policy with some training, we employed self-play against earlier versions of the agent with inspiration from the successes of AlphaGo (Silver et al., 2016) and AlphaZero (Silver et al., 2017). Such self-play exposes the agent to different strategies with greater complexity than the fixed policies of the dummy agents.

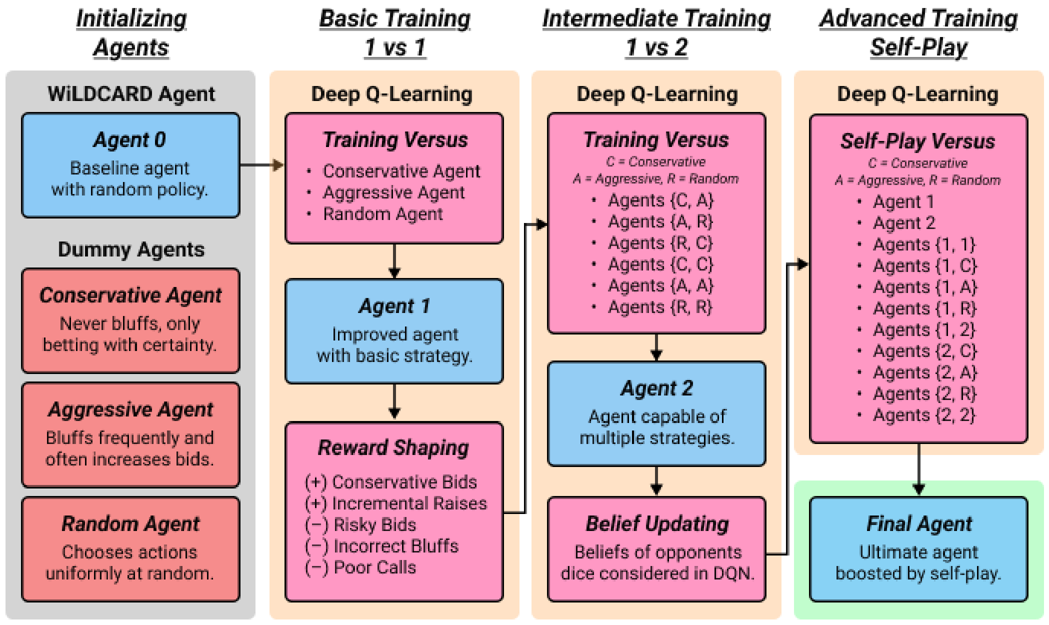
3.7 Reward Shaping

In addition to the basic reward structure defined in $\mathcal{R}$, we implemented reward shaping. As we iterated through different scenarios, we added intermediate rewards to incentivize or punish behaviors specific to that stage of training. For example, we heavily discouraged bluffing in early training to build a stable policy, while encouraging more risky play in later stages of training.

3.8 Beliefs for Opponents' Dice

One innovation in our approach is the incorporation of probabilistic beliefs about the dice rolls of each opposing agent. Instead of storing an extensive history of bids, which is computationally expensive, or treating each bid in isolation, which loses information from previous bids, our agent develops a belief distribution over the opponents' dice. In particular, we initialize a uniform prior as [α_1, α_2, α_3, α_4, α_5, α_6] with $\alpha_i = 1, \forall i \in \{1, \ldots, 6\}$ for each opposing agent. As opponents make bids on a die value i, we iteratively increment the relevant parameter α_i in this prior. During training, we sample $Dirichlet(\alpha)$ for $\alpha \in \mathbb{R}^6$, using this prior to generate a probabilistic estimate of the opponents' dice rolls. This estimate is then incorporated into the state input to the target network. This approach is

Figure 1: Training framework through simulation, deep Q-learning, reward shaping, and self-play.



not only more computationally efficient than tracking all bids, but also allows for exploration. The inherent randomness of the Dirichlet distribution generates reasonable estimates while exposing the neural network to a range of slightly different states, promoting generalization.

4 Experiments

To train our final agent, we followed the process outlined in Figure 1.

4.1 Agent 0: Initialization

To begin, we created the simulation environment, where we were capable of simulating games between 2 or 3 players while building the replay buffer and training the target network. Our environment supported modes for training and evaluation across many sets of agents. We built Q_{target} and Q_{train} as 3-layer fully connected feedforward neural networks, with a size of 128 for each hidden layer. Training was conducted after every 10 game simulations, with each training sessions run with discount factor $\gamma = 0.9$, learning rate $\eta = 0.01$, and 50 epochs.

We began by defining three dummy agents, with our Agent 0 being a Random Agent:

- **Random Agent:** At each state, chooses an action from all legal actions uniformly at random.
- **Aggressive Agent:** At each state, the agent raises the bid by one dice count or one face count 50% of the time, and calls bluff on the current bid the remaining 50% of the time.
- **Conservative Agent:** At each state, the agent only takes actions consistent with their knowledge of their own dice, calling bluff on the current bid if no such actions exist.

4.2 Agent 1: Basic Training (1-vs-1)

We cycled between 2-player training games against each of the three dummy agents, employing:

- **Conservative α-Greedy Data Generation:** We began with $\alpha = 1.0$ to follow the Conservative Agent policy [3.3], and reduced α by 0.1 with each iteration. This pushes the agent toward a conservative baseline policy in early training.

- **1-vs-1 Training:** While Agent 1 started with very nonsensical large betting patterns, its policy began stabilizing after learning from the conservative reward tuples. The agent clearly errs toward the side of aggression, often calling bluffs and placing large raises.

4.3 Agent 2: Intermediate Training (1-vs-2)

Then, we proceeded with training on 3-player games across all combinations of dummy agents:

- **Reward Shaping for Complex Strategies:** To guide the agent toward less aggressive actions, we augmented the reward space with intermediate rewards. Specifically, we offered large rewards for making small bids with incremental raises (by 1 or 2), while heavily punishing risky bluffs and calls (with large bid values).
- **1-vs-2 Training:** We trained in cycles across all six combinations of 2 opponents, shuffling the order across iterations. Agent 2 showed significant improvement in adhering to more conservative strategies, effectively countering Random and Aggressive agents. This agent struggled the most against Conservative agents, failing to use the predictability of their bids.

4.4 Final Agent: Advanced Training (Self-Play with Beliefs)

Finally, we trained our final agent with self-play against weaker versions of itself (Agent 1 and 2) in different 2 and 3 player configurations:

- **ε-Greedy Exploration:** To allow exploration on top of the agent's successful baseline policy, we allowed exploration uniformly across legal actions with $\varepsilon = 0.2$.
- **Opponent Dice Estimation:** We implemented the belief estimation discussed in [3.8] for each opposing agent. To incorporate these dice estimates into Q_{target}, we concatenated the belief to the neural network input and adjusted the size of the first layer accordingly.
- **Self-Play Against Prior Agents:** We conducted training in cycles against every new combination of 2 and 3 players, including either Agent 1, Agent 2, or both. This self-training and exploration demonstrated new strategic understanding such as predictive calling and well-timed bluffs, showing the most change in play against Conservative Agents.

5 Results

To evaluate our agent, we simulated 1000 games against each 1 and 2 dummy agent opponent configurations, where the main metric is the win percentage of our agent. Note that we evaluated the performance of each dummy agent (where Agent 0 is the Random Agent) as a benchmark to compare the performance of our trained agents with.

As seen in Table 1, our final agent outperformed every other agent in terms of win rate in 2-player games. Performance increased with each additional stage of our training. Additionally, note that the final agent was the only agent capable of beating each of the dummy agents by winning more than 50% of the time.

Table 1: 1-vs-1 games won by each agent, simulating 1000 games per combination of agent players.

Agent	vs Random	vs Aggressive	vs Conservative
Aggressive (Benchmark)	93.2%	50.1%	12.2%
Conservative (Benchmark)	99.8%	88.6%	48.2%
QAgent 0	47.8%	8.4%	0.2%
QAgent 1	**100%**	89.7%	47.9%
QAgent 2	**100%**	90.7%	49.2%
Final QAgent	**100%**	**92.1%**	**52.4%**

Similarly, in Table 2, our final agent boasted the highest win rate against other agents in 3-player games, with improvements in performance as training progressed and winning a plurality of the time (more than 33%) against 2 other opponents.

Table 2: 1-vs-2 games won by each agent, simulating 1000 games per combination of agent players. We denote Aggressive as A, Conservative as C, Random as R, and Benchmark as [B].

Agent	vs{R, A}	vs{A, C}	vs{C, R}	vs{R, R}	vs{A, A}	vs{C, C}
Aggressive [B]	70.2%	4.2%	14.9%	83.3%	34.2%	7.7%
Conservative [B]	99.7%	39.1%	52.1%	99.8%	78.1%	33.5%
QAgent 0	15.1%	0%	0.2%	32.9%	3.2%	0%
QAgent 1	98.4%	34.8%	43.3%	99.9%	63.8%	26.8%
QAgent 2	99.1%	39.5%	45.4%	**100%**	72.6%	27.9%
Final QAgent	**99.8%**	**48.1%**	**52.5%**	**100%**	**85.8%**	**36.4%**

6 Discussion

The performance of the baseline Agent 0 was notably poor, as it followed a random policy and struggled significantly against the dummy agents. This highlights the importance of structured training to develop effective strategies. After Basic Training, we observed substantial improvements across all scenarios. Agent 1 dominated against random agents and outperformed aggressive agents by calling their bad bids. Its policy resembled that of the Conservative Agent, which can be attributed to the α-greedy generation approach. However, Agent 1 still struggled against conservative agents, as its occasional bluffs were often called out.

Following Intermediate Training, we saw marked improvements in win rates, particularly in 3-player scenarios. Win rates increased by over 5% in most aggressive agent configurations, as the training with 3-player simulations helped the agent adapt its policy to perform against multiple opponents. Reward shaping proved to be particularly effective, as the agent began to prefer incremental bids that were more likely to be true. This shift in behavior was critical in achieving higher win rates against aggressive agents, who typically attempted to out-bid or call the agent's bluffs.

Finally, after Advanced Training, we observed the strongest performance yet. The model was now able to beat conservative agents a majority of the time in both 2-player and 3-player scenarios. This success can be attributed to the ε-greedy exploration strategy and the self-play component, which allowed the agent to explore a wider range of strategies catered toward more conservative agents. Additionally, belief modeling improved the agent's ability to exploit the predictability of conservative agents' bids, leading to more successful outcomes. This combination of exploration, self-play, and belief modeling ultimately enabled the agent to outplay more conservative strategies and achieve a stronger, more adaptive approach.

7 Conclusion

In this paper, we introduced **WiLDCARD** to train a reinforcement learning agent designed to play a variant of Liar's Dice with wild-card dice. By leveraging Deep Q-Networks for function approximation, our agent was able to effectively estimate optimal actions in an intractably large game space. The training process, which involved starting with fixed-policy agents and progressing through self-play, was essential in refining the agent's strategy and improving its performance across a variety of opponents. Our results demonstrate that our agent was able to outperform random and aggressive agents and achieve a strong baseline strategy. After later training phases, including the incorporation of reward shaping, opponent belief modeling, and self-play, the agent showed significant improvements, including the ability to defeat conservative agents.

Despite these successes, we note limitations and avenues for future work. One limitation is the agent's continued difficulty in consistently outperforming conservative agents, suggesting further refinement in exploitative and nuanced bluffing strategies. Further, given our success of using neural networks for function approximation, it is promising to investigate more sophisticated neural network architectures to generate more refined action value estimates. Future research could also focus on improvements in the agent's belief model, enabling more effective counter-strategies.

Overall, by successfully training an agent to play Liar's Dice at such a level, we have demonstrated the potential for DQNs to handle complex decision-making tasks with partial observability, and we hope these findings garner further research in similar partial information games.

8 Team Contributions

- **Eric Lee** implemented the reward shaping, belief estimation, and greedy exploration strategies to led the multi-stage training process.
- **Ryan Lian** implemented the training and target networks, the capturing of reward tuples within a replay buffer, and the control flow for continuous training.
- **Brian Xu** implemented classes specifying agent policies and developed the game simulation environment with game configuration and evaluation capabilities.

References

Thomas Dybdahl Ahle. 2022. Liar's dice by self-play with counterfactual regret and neural networks. *Toward Data Science, Medium.* URL: https://towardsdatascience.com/lairs-dice-by-self-play-3bbed6addde0.

Matt Dodge. 2017. A statistical look at liar's dice - the gambler vs the mathematician. *Code-Burst, Medium.* URL: https://codeburst.io/a-statistical-look-at-liars-dice-the-gambler-vs-the-mathematician-42d3a96a88d1.

Dylan Greaves and Kenneth Tay. 2018. Winning liar's dice. *Stanford University, CS 221.* URL: https://web.stanford.edu/class/archive/cs/cs221/cs221.1192/2018/restricted/posters/dyl/poster.pdf.

Johannes Heinrich and David Silver. 2016. Deep reinforcement learning from self-play in imperfect-information games. URL: https://arxiv.org/abs/1603.01121.

Hana Lee. 2017. Reinforcement learning applied to a game of deceit: Theory and reinforcement learning. *Stanford University, CS 229.* URL: https://cs229.stanford.edu/proj2017/final-reports/5216365.pdf.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. URL: https://arxiv.org/abs/1312.5602.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, and et al. Laurent Sifre. 2016. Mastering the game of go with deep neural networks and tree search. *Nature.* URL: https://www.nature.com/articles/nature16961.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, and Matthew Lai et al. 2017. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv.* URL: https://arxiv.org/abs/1712.01815.

Zuyuan Zhang, Mahdi Imani, and Tian Lan. 2024. Modeling other players with bayesian beliefs for games with incomplete information. URL: https://arxiv.org/abs/2405.14122.