Q-learning on Games of Imperfect Information

Ryan Holmdahl and Stephanie Chen

October 2, 2016

1 Abstract

We formalize the card game BS as a hidden-state Markov decision process ("HSMDP"), an extension of the standard MDP utilizing a world state not entirely known to the agent. A Q-learning method for improving win rate against estimations of opponents is proposed, tested, and discussed. The results indicate that Q-learning can be effectively applied to some games of imperfect information.

2 Purpose

BS, also known as Cheat, is a card game typically played in large group settings but possible with any group of three or more players. While not a moneymaking enterprise like poker, BS is often the centerpiece of high-stakes competitions with extended family or large friend groups. As such, any possible advantage over one's competition can have a dramatic impact on one's ability to acquire all-important bragging rights. Developing a strategy to the game, however, is heavily dependent on one's expectation of one's opponents and reliant on taxing computation of probabilities. As such, most players enter the game, especially in its usual nonprofessional settings, with some set strategy. We propose that an artificial intelligence approach to BS utilizing Q-learning can take advantage of this common weakness and provide a player with a crucial edge over their opponents.

3 Gameplay

Players form a circle and a full deck of cards, Jokers removed, is shuffled and dealt as evenly as possible among them. Each player's objective is to discard all of their cards.

On each turn, a player discards, face down, a nonzero number of cards of a certain rank, saying aloud the number and the rank. For the first turn, this rank is Ace; for the second, Two; the third, Three; and so on, eventually cycling through Jack, Queen, King, and back to Ace. Players must be honest about how many cards they discard but may lie about the rank of those cards. For example, a player may say "Two Aces" but in actuality play any two cards from their hand. Any of their opponents may, however, call a player out on a lie by saying "BS" after the play has been made but before the next player's turn. If the play was indeed a lie, then the player "busts" and must take the entire discard pile into their hand. If the play was honest, then the caller instead takes the discard pile. Regardless of the outcome, play continues as normal to the next player in the rotation and the next rank in the sequence.

The game ends when a player has no cards in their hand on a turn which is not their own; that is, the player has successfully discarded their entire hand and survived any subsequent "BS" calls.

4 Task Definition

Given black-box input-output estimations of our opponents, the composition of the deck, and the player order, we learn a strategy for one agent player which optimizes for wins against those opponents.

For the purposes of this project, full-size card decks will not be used. Games played with large decks tend to take an infeasible amount of time to simulate, as BS has no set turn limit or guaranteed ending. The time it would take to establish a baseline, let alone learn, on such a deck would be extraordinary. Furthermore, large decks are exponentially more complex than smaller decks. A learning agent would have to learn on an absurd number of iterations to develop a functional policy, which we found unnecessary and impractical for this project. Instead, we use four-person games of six ranks and three cards per rank. We find this to give decent positional equality to the players while capturing the various states of certainty and uncertainty inherent in full games. In a typical game with a full deck and five players, most players would receive ten cards in their starting hand. We can see the hand size relative to the number of ranks is $\frac{10}{13}$, or approximately .77. In our format, two players start with four cards and two players start with five cards. For the four-card hands, the size relative to the number of ranks is approximately .67, while for the five-card hands, the relative size is approximately .83. Averaged among the players, the relative size is .75, which is about the same as that in the full game. As such, we can expect similarly difficult-to-play hands for the players of both.

Beyond its utility in securing BS game victories, we find this task interesting in part because of its incompleteness of information. At any given time, each player is privy only to a small slice of the truth. We more specifically enumerate what exactly is known in

Section 6.2, but crucial unknowns include the contents of opponent hands and the cards actually played by opponents on any given turn. The lack of certainty complicates traditional game modeling techniques such as expectimax, as one would be forced to calculate value against every possible set of unknowns.

While incompleteness is inherent to the genre, BS has the somewhat unique feature of total certainty in union — that is, the sum of every player's knowledge does constitute the entirety of the game's truth. Every card is distributed to the players at the start of the game, so there is no universally unknown deck to act as a source of random chance. This makes the game more difficult to model with a standard Markov decision process, as transition probabilities are not as simple as the likelihood of the appearance of a top card but instead rely on opponent strategies and the unknowns of their hands and knowledge.

In order to evaluate the effectiveness of our learned strategy, we construct a model for a BS game and measure accuracy in terms of win rate against sets of opponents utilizing different policies.

5 Literature Review

There exists a fairly substantial amount of prior work regarding applications of reinforcement learning to games of imperfect information. While the game BS itself has not been extensively studied, most of the literature focuses on variations of the popular Texas Hold'em form of poker, a game similar to BS in its lack of complete information, multiplayer setup, and the need for a skilled player to appropriately manage bluffing strategy [1].

Dahl argues that value-based reinforcement learning methods, such as temporal difference (TD) learning and Q-learning are not applicable to imperfect information games such as poker, primarily because the full state of the game is never known to any one player and thus an accurate value function can never be determined. The "lagging anchor algorithm" utilized instead, which uses prior state history to dampen a simultaneous gradient-descent approach to learning, is however not feasible for necessarily multiplayer games like BS, as potential policies are calculated against all possible opponent policies (Dahl uses a simplified two-player poker game model) [2].

Ishii et al., in analyzing the imperfect information card game Hearts, consider a "partially-observable Markov decision process" (POMDP), in which the game is modeled as a MDP where each state contains a probability distribution of possible complete states, based on state history and game knowledge. They conclude, similarly, that traditional TD- and Q-learning are "naive" approaches to POMDPs, as they assume a direct approximation into an MDP. Instead, they integrate into the learning algorithm explicit, unique function approximators for each opponent agent [3]. This approach is strongly limited by the specificity of the function approximators, creating a strategy that is highly dependent on the opponent policies.

In our model, we aim to strike a balance between the generality but computational difficulty of Dahl's gradient descent and the specificity of Ishii et al.'s opponent-targeted state distributions. This is most similar to the approach presented by Teófilo et al., in which reinforcement learning is used on a single poker player agent competing against predetermined opponent types [4]. Each opponent follows a pre-defined strategy determined by an "aggression factor" derived from standard analysis of human poker players. They conclude that Q-learning is an appropriate approach in this limited case, performing well against a variety of opponent types after training.

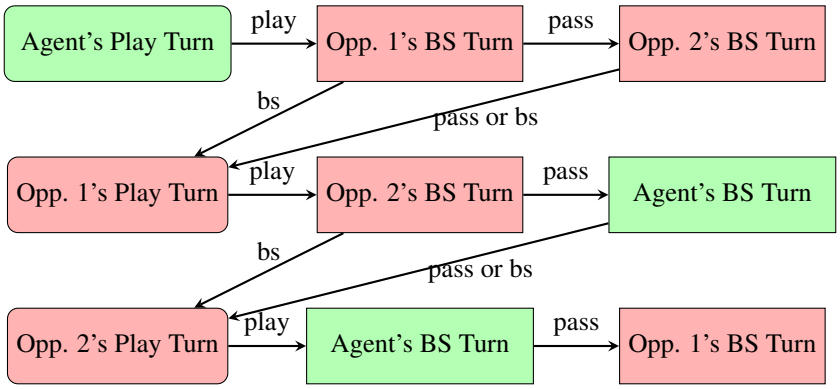
6 Problem Model

6.1 Simulator Design

We choose to structure each simulated game as an instance of what we refer to as a hidden-state Markov decision process. An HSMDP operates similarly to an MDP in that there is a single player agent in a single state which allows for a set of actions that in turn lead non-deterministically to a new state. The key difference between the two is that, in an HSMDP, the state visible to the agent reveals only a portion of the information about the game's full state, the remainder of which is stored in an unsurfaced object. Instead of an action leading to a new state with some set probability, each action changes the hidden game state and triggers the next adversary's turn. A state is created for the adversary using only the information available to it, and the adversary uses this information to act according to a policy function installed in the HSMDP at launch. The agent has no control over this action and no state is surfaced to it on which to act. Instead, the adversaries continue to act and the hidden global game state continues to change until the agent receives a turn, either to make a BS call or play some cards, once again only aware of the limited available information.

As this implies, there are two types of states: a "play" state, in which the player must discard one or more cards, and a "BS" state, in which the player is given the opportunity to call an opponent out on a lie. Each simulation begins with a "play" state surfaced to whichever player is at index 0. That player makes their choice of cards to play, causing the game to transition into a "BS" state for the player at the next index. That player is offered the choice to then make a call or pass. If the player chooses to pass, then the game enters a "BS" state for the following player. If any player makes a BS call, then the appropriate player takes the discard pile and the next player enters a "play" state. Alternatively, if every player passes, play also continues to the next player. At all states where the player calling or playing is our agent, the state is surfaced and the agent may act as it sees fit.

The following diagram shows the state flow for an agent in position 0 in a three-player game:



Throughout our implementation, we record any sort of card list (such as hands, the pile, etc.) in the format of an n-tuple or -list, where n is the number of ranks in the game. Each value in the list or tuple represents the number of that card present in that list. We do not treat cards by identity, as their suit does not matter for this particular game. Using a card count system, as we do, narrows our state space and simplifies learning and decision-making.

One important feature of our particular implementation is the notion of card rotation. Given the cyclical nature of play, from Aces to Kings and back to Aces, the only important feature of any card is its rank relative to the card currently being played. The absolute rank of the card does not matter, as different cards have no different attributes apart from their positions in the cycle. As such, in each "play" state apart from the first, we rotate every card list and tuple, regardless of location in storage, so that the 0th index of every card list represents the current card being played. This allows for much easier policy generalization, as the agent need not learn an entirely new decision for every rank.

6.2 Hidden State Space

The hidden state contains the following data members:

- Player hands. At the beginning of each game, the hidden global state distributes the supplied deck among the players and records how many cards of each rank are held by each player. Whenever plays are made and cards switch hands, the HSMDP card counts are updated. Whenever a visible state is created for a player to act on, the corresponding hand is retrieved.
- Discard pile contents. The discard pile is recorded as a list of lists, with the list at each index n representing those cards contributed by player n. We use this method to facilitate retrieval of each player's pile knowledge when constructing a state.
- Bust knowledge. Consider the case where player 0 has played two Aces. If player 1 busts and takes the discard pile, player 0 now knows that player 1 has just acquired two Aces. While player 1 may at any time play those cards, player 0 can be sure that, until the next bust, no player except player 1 can possess those two Aces. We refer to this awareness as "bust knowledge." In the hidden state, we maintain a list of two-tuples, one per player. Each tuple consists of an (index, card count tuple) pair, where the index refers to the player who busted and the card count tuple indicates which cards they must have acquired.
- The last play. During a round of "BS" turns, the hidden state maintains an (index, card list) tuple, where the index is that of the player of the previous "play" state and the card list represents what they have actually played. This is never surfaced in full to potential callers; instead, callers can only see the player index and the total number of cards played.

In our game model of four players on a six-rank deck, an example of a hidden state could be:

- Hand contents of [[1,0,0,2,0,0], [0,1,1,0,0,1], [0,1,0,1,0,0], [0,1,1,0,2,0]]
- Discard pile contents of [[0,0,0,0,1,0], [2,0,0,0,0,0], [0,0,1,0,0,0], [0,0,0,0,0,2]]
- Bust knowledge of [(1,(1,0,0,0,0,0)), (1,(0,0,0,0,0,0)), (1,(1,0,0,0,0,0)), (1,(0,0,0,0,0,0))]
- Last play of (1, [2,0,0,0,0,0])

6.3 State Space

Each state consists of either a six-tuple, if the turn is a "play" turn, or a seven-tuple, if a "BS" turn. These values, as described below, represent the useful information available to a player selecting an action:

1. State type. This takes on a value of either "play" or "bs" depending on the type of turn being surfaced. This is mostly a utility value for programming convenience, as one can deduce this from the length of the state.
2. Hand contents. An n-tuple, where n is the number of ranks in the deck, representing the counts of each card in the current player's hand. This is one of the most important and most obvious pieces of information surfaced to a player for action selection.

3. Discard pile contribution. A tuple of the same size n representing the counts of cards the current player has contributed to the current discard pile. Knowledge of which cards are entirely out of circulation can help a player make more informed call decisions, as it affects the probability that an opponent has the cards they claim to play. This is reset to zeroes when a player busts and the pile is reset.

4. Pile size. An integer representing the total number of cards in the discard pile. While a player knows only a portion of the cards contained in the pile, every player can see its size. The size of the pile corresponds to the potential loss suffered with a bust or the potential damage dealt to an opponent with a successful call.

5. Bust knowledge. A two-tuple of the form (index, card list), where the index is the last player to bust and the card list is the cards they must have acquired from the discard pile. As with discard pile contribution, this value affects the probability that opponents have certain cards.

6. Hand sizes. A t-tuple, where t is the number of players, whose values represent the hand sizes of the corresponding players.

7. Last play. If the state is a "BS" state, then there is a seventh value which is a two-tuple of the form (index, card count). Index represents the player whose play is under consideration and card count is the number of cards played.

In our example hidden state, the state for player 2 would be: ("bs", (0,1,0,1,0,0), (0,0,1,0,0,0), 6, (1,(1,0,0,0,0,0)), (3,3,2,4), (1,2)).

6.4 Action Space

There are two broad categories of actions:

- Play actions. These are the actions available in "play" states, which take the form of n-tuples, where n is the number of ranks in the deck. The values at each index of the tuple represent how many cards of that rank are to be played. From any "play" state, the set of actions available consists of unique combinations of the player's hand such that the sum of the action (which is equal to the number of cards played) is less than or equal to the count of that rank in the total deck. While a player could, in theory, play more cards of a rank than there are in the deck, such a play could be immediately called by any other player. As such, there is no conceivable justification for making such a play and including such actions would only complicate learning computations. We therefore remove them as possibilities.
- Call actions. As previously explained, players in "BS" states can either "pass" or "bs."

The outcomes of these actions are explained in Section 6.1.

7 Adversaries

7.1 Simple Policy

We have designed two sorts of adversaries for the purpose of training and testing our learning. The first is our **simple policy**, which presents an uneducated, straightforward strategy. In call states, a player following the simple policy has a $1/n$ probability of calling "BS," where n is the number of players in the game. In play states, the player will play all of their cards of the required rank. If the player has no such cards, they will choose a random action from the set of available plays. This policy is simplistic but is a straightforward approximation of the decision-making of a very naive player. As such, we use it as the agent for our baseline against which we can compare the performance of our more refined learning agent.

7.2 Dishonest Policy

Our second policy, the **dishonest policy**, is a more comprehensive estimate of a player's decision-making. The dishonest policy is initialized with a "dishonesty" value d, which represents how likely the player is to cheat unnecessarily. We will break down how the dishonest policy acts by turn type.

On a "BS" turn, the player computes from their bust knowledge and pile knowledge the number of cards of the current rank that the last player certainly does not have. We will call the total number of cards of the current rank t and the number of cards removed from circulation r. We then establish the following other variables:

- N, which is the total number of cards not in the dishonest player's hand;
- $k = t - r$, which is the number of cards of the current rank the last player might have had;
- n, which is the number of cards in the last player's hand before playing; and
- x, which is the number of cards played by that last player.

The dishonest player then makes a decision according to the following rules:

1. If the previous play is impossible with this information — that is, $x + r > t$ — then the player calls BS.
2. If the player *must* have that many cards of the current rank — that is, every possible distribution of the unknown cards gives the player at least x cards of the current rank — then the dishonest player passes.
3. (a) The dishonest player computes their relative change in hand size if it calls incorrectly (changeForCaller) and the relative change in hand size for the last player if it calls correctly (changeForPlayer).
   (b) The dishonest player computes the probability that the player had x cards in hand using a hypergeometric distribution

$$\mathbf{P}(x) = \frac{\binom{k}{x}\binom{N-k}{n-x}}{\binom{N}{n}}$$

   which we will refer to as p.
   (c) The dishonest player decides to call with probability

$$\mathbf{P}(A = \text{call}) = \frac{(1-p) * \text{changeForPlayer}}{\text{changeForCaller} * \text{nplayers}}$$

On a "play" turn, the dishonest player categorizes actions as truthful, if they are entirely honest and consist only of the cards of the current rank; semitruthful, if they contain some cards of the current rank and some other cards; and lies, if they contain no cards of the current rank. The player then makes a decision according to these rules:

1. If the dishonest player has no cards of the current rank, it picks a random lie, weighted against those lies which contain cards of the rank to be played on the player's next turns.
2. If the dishonest player has some truthful actions and either no semitruthful actions or with probability $1 - d$, the player plays all of its cards of the current rank.
3. The player picks a semitruthful action at random, weighing against those actions which contain cards to be played on its upcoming turns.

As we can see, this policy is significantly more informed than the simple policy and takes into account information as a human player might. Moreover, it is less predictable to a learning agent than the simple policy, as it consumes a much larger set of information, operates on that information nonlinearly, and is less deterministic. Given its greater degree of complexity and closer approximation to the actions of a human player, we use the dishonest policy as the adversaries of our baseline, oracle, and learning agents.

8 Baseline

For our baseline, we measure the win rate of a simple-policy agent in each of the three table positions against dishonest-policy agents of various d-values. Across 1000 trials, the results are as follows:

Agent Position	Adversary d-value	Win Probability
0	0.1	0.186
	0.5	0.209
	0.75	0.252
1	0.1	0.190
	0.5	0.197
	0.75	0.213
2	0.1	0.286
	0.5	0.290
	0.75	0.302
3	0.1	0.261
	0.5	0.280
	0.75	0.297

As we can see, the win distribution is somewhat even between the players, favoring the agent when in the latter two positions (which are advantaged by initially receiving one fewer card). The agent performs better against more dishonest opponents, as its random calls are more often validated. Given the simplicity of our simple policy, this agent's performance should be considered the lower bound of our learned agent.

9 Oracle

Our oracle estimates the upper bound of performance by providing the agent with some powerful advantages. The agent can "see" the cards played by its adversaries, perfectly predicting whether they have cheated and calling appropriately. In addition, the agent is only dealt cards that it will play on its first $n/2$ turns, where n is the total number of cards. In the wild, a more beneficial deal would be highly improbable. Across 5000 iterations, a necessarily high number given the very high oracle win rates, the data is as follows:

Agent Position	Adversary d-value	Win Probability
0	0.1	0.997
	0.5	0.999
	0.75	0.999
1	0.1	0.981
	0.5	0.990
	0.75	0.996
2	0.1	0.982
	0.5	0.991
	0.75	0.995
3	0.1	0.964
	0.5	0.985
	0.75	0.992

As is apparent from these results, it is possible to win an extraordinary percentage of games with the ability to correctly predict opponent plays and the proper cards; that is, there is a very low chance that dishonest-policy opponents can get lucky and get an unstoppable hand.

10 Learning Model

We use feature-based Q-learning to develop a policy that optimizes against a given set of opponents.

10.1 Applicability

As discussed in Section 5, there has been some debate about the applicability of Q-learning, and reinforcement learning in general, to adversarial games of imperfect information, since the state surfaced to the player does not represent the entire truth about the game world. We argue that these problems are not applicable to our formalization of BS. Although our states do not entirely represent the game world and there are adversaries to the agent, the game can be flattened into an MDP. The key is that our opponent policies, though non-deterministic, are immutable. The game begins in a world state S determined by the established probabilities of dealing the cards in a particular way. From there, the adversaries act with similarly established probabilities based on their understanding of S. Though difficult to calculate, one could consider each chain of adversary actions (again, whose probabilities are calculable knowing S) as a single transition to a state s surfaced to the agent. By this token, we can construct a set of (p, S') pairs, where p is the probability that S' is the world state at s for each possible surfaced state s. For any action a at s, we can then determine the probability that the new world state is S'', and further follow each chain of adversary actions until eventually reaching all subsequent s'. We therefore have that there are certain transition probabilities, unknown to the players, for each (state, action) pair. As such, we can consider this an MDP with unknown rewards and transitions, which is perfectly applicable for Q-learning.

10.2 Feature Selection

With a state space as large as that of this problem, generalizing states using features is an essential step. Our initial attempts at feature extraction simply broke down the states into their constituent components (hand sizes, current hand, pile size, etc.), along with some very basic derivative features such as the presence of a rank in the hand. This was inefficient and did not efficiently produce satisfactory results, as it failed to capture the important facets of the state and entirely ignored the action being taken. We also found that non-indicator features caused the weights vector to diverge wildly, as our broad state space gave undue weight to features that would always be present. Through an iterative improvement process of adding and removing features, we eventually came to the conclusion that there are two important ideas to capture for each (state, action) pair: the playability of the position and the quality of the action.

In this context, "playability" refers to the strength of the current position and the likelihood of success going forward from it. For each state, we extract features indicating whether cards of the current rank are held and, if so, the number held. In addition, we extract indicators for the presence of cards of the next rank to be played by the agent and the number of such cards held. These features allow the agent to determine whether it can be honest at that state and its successors, which in turn allows it to avoid unnecessary forced lying plays which might result in a bust. As a separate metric of playability, we also create an indicator representing the agent's hand size relative to its opponents. Generally speaking, a relatively smaller hand is beneficial, as fewer moves would be required to reach a win state.

Action quality is an umbrella term for those features determining the probability that an action delivers the agent to the best possible state. For "BS" states, action quality features are typically indicators of the form action+[information], where the information string is some piece of knowledge about the last play or the last player. These pieces of information include the cards the opponent is known to have and the cards the opponent is known not to have. These features allow the learning algorithm to weigh the effect of an action relative to its knowledge and make an educated decision.

For "play" states, features are derived facts about the action being taken. Features include indicators of whether the action included all cards of the current rank and only cards of that rank, whether the agent had no choice but to lie, and if the action included cards of the next rank to be played by the agent. By breaking down the action into general but descriptive components, the algorithm does not need to relearn certain trends for each possible action.

A summary of our features is as follows:

- Indicators on the number of cards of the current rank in hand and a general indicator on whether such cards are present at all.
- Identical indicators for cards of the next rank to be played by the agent.
- In "BS" states, we have indicator features on details about the last play, such as the player, the number of cards played, and the amount of cards the player is known to not have. Each of these features is associated with the action taken by the agent; for example, "pass_nplayed_4" indicates whether the last player played four cards and the agent passes.
- In "play" states, features about the cards played and the truthfulness of the action are extracted, such as whether it was a forced lie, how many false cards were played, and whether or not all of the cards of the current rank were played. Indicators are also created on the number of cards of the agent's next rank that were played.

10.3 Hyperparameter Selection

Our implementation of Q-learning uses a standard exploration probability of 0.2. With experimentation with higher exploration probabilities, up to 0.8, we received some superior performance, but these tests were very inconsistent and occasionally the policy would be far worse than with an exploration probability of 0.2. In the interest of consistency, we opted for slightly worse but stable results as opposed to occasionally higher results.

Our step size for feedback incorporation is equal to $\frac{1}{\sqrt{n}}$, where n is the number of iterations elapsed. This standard value generally gave us good convergence in a low number of trials. Other experiments, including a constant 1, produced worse or divergent results.

We choose to learn for 15000 iterations, a relatively small number given the complexity of the game. From our tests, higher counts did not produce any better results, which we believe is due to our generalized feature selection and forced-convergence step size. With more state-specific features or on larger games, more trials and less harshly tapered step size would likely be necessary.

11 Results

Our learning agent tended to do very well in all positions, as the data below shows. Games against our agent often dragged on for many turns, often in the low hundreds and occasionally extending into the thousands. Our simulations were capped at 1000 agent-turns, so some trials ended incomplete. In the table, we record two win rates. The first, W, marks the win rate among completed games, discarding incomplete trials. The second, W', is a worst-case rate counting each incomplete game as a loss. Data is for 15000 learning iterations and 1000 testing iterations.

Agent Position	Adversary d-value	W	W'
0	0.1	0.921	0.747
	0.5	0.941	0.728
	0.75	0.952	0.780
1	0.1	0.911	0.676
	0.5	0.937	0.771
	0.75	0.959	0.835
2	0.1	0.934	0.797
	0.5	0.976	0.831
	0.75	0.979	0.903
3	0.1	0.915	0.693
	0.5	0.942	0.655
	0.75	0.967	0.855

While these results are certainly noisy because of the relatively low number of testing trials and the inherent randomness of the game, they consistently beat the baseline and approach the oracle, even in worst-case scenarios. Our learning agent tends to do worse against more honest opponents, as did the agent used in the baseline. Unlike our baseline, however, there was no clear positional advantage for the learning agent.

12 Qualitative Policy Analysis

In general, our learned policies push for longer games, oftentimes extending into the thousands of rounds. Taking the policy learned against d-value 0.75 adversaries as an example, we can see the pattern of decision-making taken by the agent in one case. The highest weights are applied to having no cards of the current rank or the next rank, placing a high value on states with fewer cards — essentially states that are closer to winning. The next two highest values are for having those types of cards, which indicate a preference to have at least some of the next two ranks to be played and reducing the need to lie. The policy also has a bias towards taking the "bs" action regardless of other factors, which makes sense against an opponent which lies 75% of the time. Other notable favored traits, in descending order, include passing when the player has no knowledge of the opponent's hand, again avoiding risk; having the smallest hand, or essentially being closest to winning; and preserving cards of the next rank to be played by the agent. These weights seem to indicate a decent understanding of basic strategy by our agent. Features weighted against include passing when three cards of the current rank are out of circulation, indicating a knowledge of when calling BS would be favorable, as well as most actions which are dishonest. Overall, the agent effectively uses the information available to it in predicting lies, favors calling BS out-of-hand against more dishonest opponents, and tries to be honest whenever possible.

13 Error Analysis

While the results of this experiment are certainly promising, there are several sources of error which limit its effectiveness in addressing the given task. From a structural standpoint, the formalization is somewhat limited by its assignment of static or predefined policies to adversaries, though as addressed in Section 5, more fully generalizing our opponent agents would be prohibitively expensive. The notion that an opponent will not adapt to the play-style of our agent, especially over the turn counts that some trials produce, would be naive in a real-world scenario. The turn counts themselves are another issue, as no human players could endure a thousand-turn game of BS. A model that assigns punishment for overly long games or on a per-turn basis might address this problem.

There are also some sources of error in our design of the adversary policies. The underlying assumption that only our agent cares to adapt to its adversaries is tenuous in situations beyond casual games. It is certainly possible that opponents would draw similar estimations of one another as our model and make decisions accordingly, which our policy design does not account for. In addition, we made certain design decisions, such as BS call probability and action weighting, that prevent certain play-styles from being captured. Though we have no reason to believe that any other play-style might perform significantly better against our learning agent, this avenue would need further exploration before use of our model can be considered widely applicable.

Lastly, our reduction of the problem to a four-person game of eighteen cards does not guarantee success in real-world scenarios. While our representation is, as explained in Section 4, a reasonable one, there are numerous realistic play setups, such as those with different player counts, that it does not necessarily capture. Moreover, any solution to a reduced problem cannot be out-of-hand considered a solution to the original, and testing on a full-size deck in more diverse scenarios would be needed to make such a claim.

14 Conclusion and Future Work

We have seen from the results of this experiment that, given a proper formalization of the problem, Q-learning can be an effective tool for solving games of imperfect information. While the strategy adopted by a learning agent can be unusual and likely infeasible in a game where opponents can succumb to exhaustion or analyze the agent's strategy, this is certainly a promising start to the application of these AI principles to this set of games.

To improve upon this particular experiment, a faster language than Python capable of more efficiently running simulations could be used. This would allow larger decks to be used, better representing real-world instances of the game. In addition, one could, for each adversary, map every possible state to an action before simulating and simply used this cached information rather than re-calculating the action every time a state occurs.

As a next step, this particular model of problem formalization and learning could be applied to a different game of the same class as BS. For example, Scum is a card game in which cards are similarly distributed among the players, who then compete to empty their hands by playing cards of increasing value, with some cards having unique characteristics. It would be interesting to see if, by representing the game as an HSMDP as we do here, similarly strong results could be achieved.

Lastly, allowing the adversaries to be more adaptive and learn agent behaviors would make the gameplay more realistic and prevent certain unfair and impractical policies from being developed. We did develop a preliminary system through which adversaries could learn opponent actions, but it did not seem to provide a competitive impact and was not used for the purposes of this project.

References

[1] Billings, D., Papp, D., Schaeffer, J., Szafron, D. (1998). *Opponent Modeling in Poker*. Retrieved from `http://www.aaai.org/Papers/AAAI/1998/AAAI98-070.pdf`.

[2] Dahl, Fredrik A. (30 August 2001). *A Reinforcement Learning Algorithm Applied to Simplified Two-Player Texas Hold'em Poker*. Retrieved from `http://link.springer.com/chapter/10.1007/3-540-44795-4_8`.

[3] Ishii, S., Fujita, H., Yamazaki, T., Matsuda, J., Matusno, Y. (May 2005). *A Reinforcement Learning Scheme for a Partially-Observable Multi-Agent Game*. Retrieved from http://link.springer.com/article/10.1007/s10994-005-0461-8.

[4] Teófilo, Luís F., Passos, N., Reis, Luís P., Cardoso, Henrique L. (June 2012). *Adapting Strategies to Opponent Models in Incomplete Information Games: A Reinforcement Learning Approach for Poker*. Retrieved from http://link.springer.com/chapter/10.1007/978-3-642-31368-4_26