Daryl Chang
Addison Weiler

Training BS Agents Using TD Learning

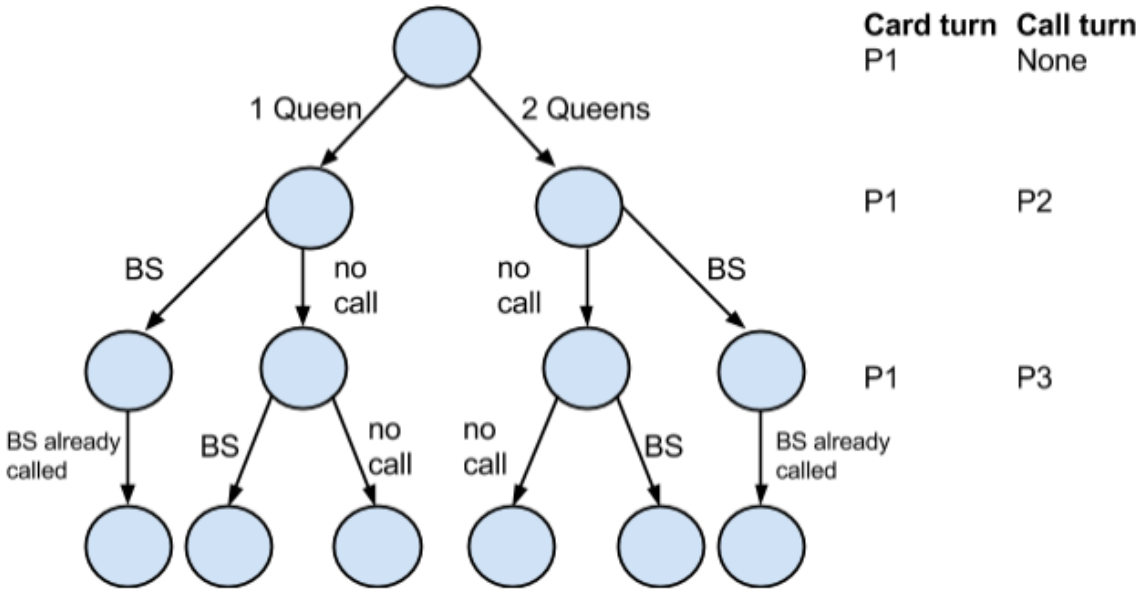
Introduction:

The goal of our project was to create an agent to play the popular card game "BS," also known as "Cheat," using TD learning.

To play BS, a deck of cards is first completely dealt out to three or more players, who then take turns playing up to four cards at a time from their hand to a central discard pile. Each turn, a player is assigned a card value in ascending order that they must play, starting with Aces. The player then has the option to play either cards with the assigned value, or if they do not have any valid cards (or want to get rid of more cards overall), other cards which they claim to be the assigned value. After the player's cards are played face-down, every other player then has the chance to call "BS" on the current player, meaning that they claim the player is lying about what cards they are placing down (i.e. playing a 2 and a 3 while claiming to have played two Aces). If any other players call BS, the played cards are revealed. If the discarding player was lying about the cards they played, they must then take all of the cards in the discard pile. If the discarding player was not lying though, the player that called BS must then take all the cards in the discard pile. The game ends when a player gets rid of all of the cards in their hand.

We modeled a game of BS with a game tree. For simplification of the game, we implemented "turn-based" BS calls, meaning that after a player discarded their cards, we would iterate through the rest of the players in order, and if any player called BS, no other players could call BS for that turn. A BS game tree for one full turn (Card turn and two call turns) is shown in Figure 1 for a game with three players. One other modification we made to our game was to only use card values Ace, 2, 3, 4, and 5 for allowing our models to be tested within a reasonable amount of time.

Figure 1: A representative BS game tree for a single turn involving three players.



Experimentation Overview:

In the following sections, we will introduce the metrics for evaluating our agent, including the baseline simple agent against which the TD agent will be compared. We will then describe the naive TD agent implementation and progressively cover the modifications we made to improve its performance, such as using different weight initializations, creating a model-based reflex agent, and tournament-style training. Finally, we will analyze the reasons behind the performance improvements and make recommendations for future modifications.

Metrics and Simple Evaluation:

For our project, we had to devise a way to test the TD learning agents we created. To do this, we created four separate agents with certain policies that our TD agent could play against. The agents are as follows:

- **Random Agent**: This agent played as randomly as possible, meaning that it chose with a 50% probability whether or not to play a "valid" move or a "lying" move, unless it was forced to play one of the two moves (there were no possible moves for the other option). Similarly, this agent chose with a 50% probability whether or not to call BS on another player.

- **Honest Agent**: This agent always tried to play honestly, meaning it always tried to only play the card value it was assigned, and trusted every other player to do the same (it never called BS). However, in the case where this agent had to play dishonestly, it would choose a single card at random from its hand to play.

- **Dishonest/Greedy Agent**: This agent always played dishonestly, meaning that it always tried to play four random cards at a time from its hand. The agent never called BS on any other player, as it was trying to get rid of cards as fast as possible.

- **Always Call BS Agent:** This agent, true to its name, always calls BS whenever possible. For card turns, the agent then acts like the honest agent, trying to always play honestly unless it is not possible to do so.

While we created these four agents purely as metrics for testing our TD agent, we also created a baseline reflex agent modeled on a simple state evaluation function. This evaluation function returned a score based on the following equation:

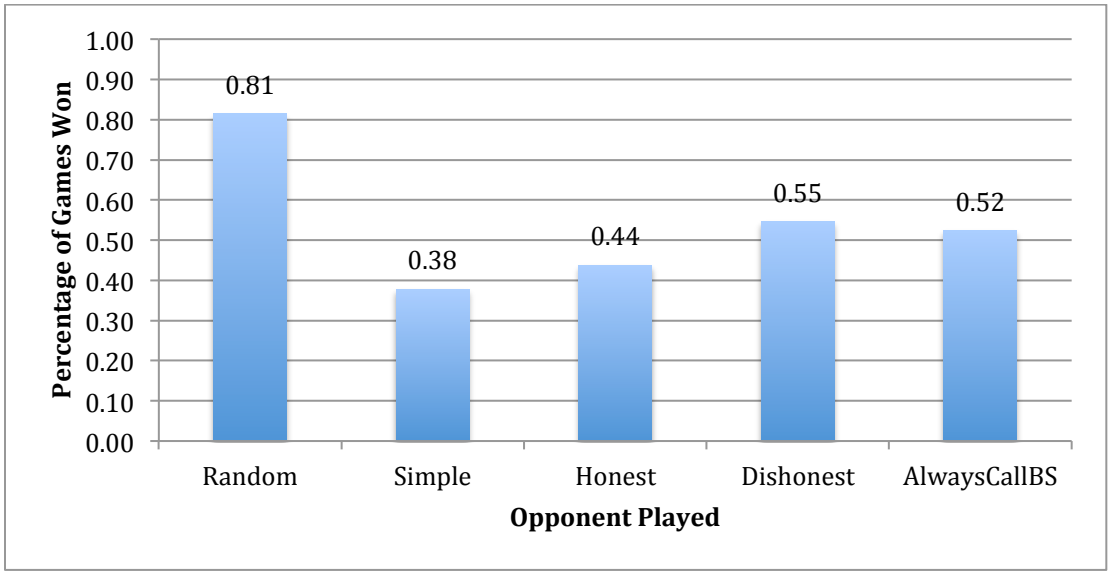
Score = (-1) * (Number of cards in agent's hand) + (Average number of cards in an opponent's hand)

We then played this "simple" agent against both itself and all of our metric agents in order to get baseline scores. The scores are summarized in Figure 2. We also decided to use the simple agent as a fifth metric for testing our TD learning agent.

We fixed several parameters in this project to make testing more convenient. Unless otherwise noted, all results were obtained under the following parameters. First, we set the number of players to be 3, the minimum number needed to play a game of BS. Second, in

calculating the win rate of an agent, we ran 10 runs in which the agent was trained over 50 games (if necessary) and subsequently tested on 100 games. Finally, as mentioned above, we reduced the total card deck size to 20 cards instead of 52 to speed up training and testing.

Figure 2: Simple reflex agent performance (baseline scores)



Overall, the simple reflex agent performed relatively well against the basket of opponents. On average, it calls BS 14.9% of the time. Against the random agent opponents, its BS calls are right 59.6% of the time; however, against simple agent opponents and AlwaysCallBS opponents, its BS calls were only right 33.9% and 10.7% of the time. Hence, some of the variation in the win rates against different opponents can be explained by the accuracy of the BS calls. The remaining variation most likely results from the accuracy of opponents' BS calls and the aggressiveness (i.e. how many cards they play) of their policies.

TD Reflex Agent:

The first agent we created was a TD reflex agent modeled after the agent created in the Backgammon assignment. This agent utilized the log-linear evaluation function with the following features:

- The total number of cards in the agent's hand.
- The number of cards in each of the agent's opponent's hands.
- The size of the discard pile.
- "i of a kind," the number of single cards, pairs, triplets, and quadruplets that the agent has in their hand.
- The number of cards of the next required card value an agent has in their hand (look ahead).
- The number of cards of the "next next" required card value an agent has in their hand (look ahead x 2).
- The number of each specific card in the agent's hand.

Similar to backgammon, a state was defined as a (game, player) combination, and the value of the game state was evaluated from the player's perspective.

Daryl Chang
Addison Weiler

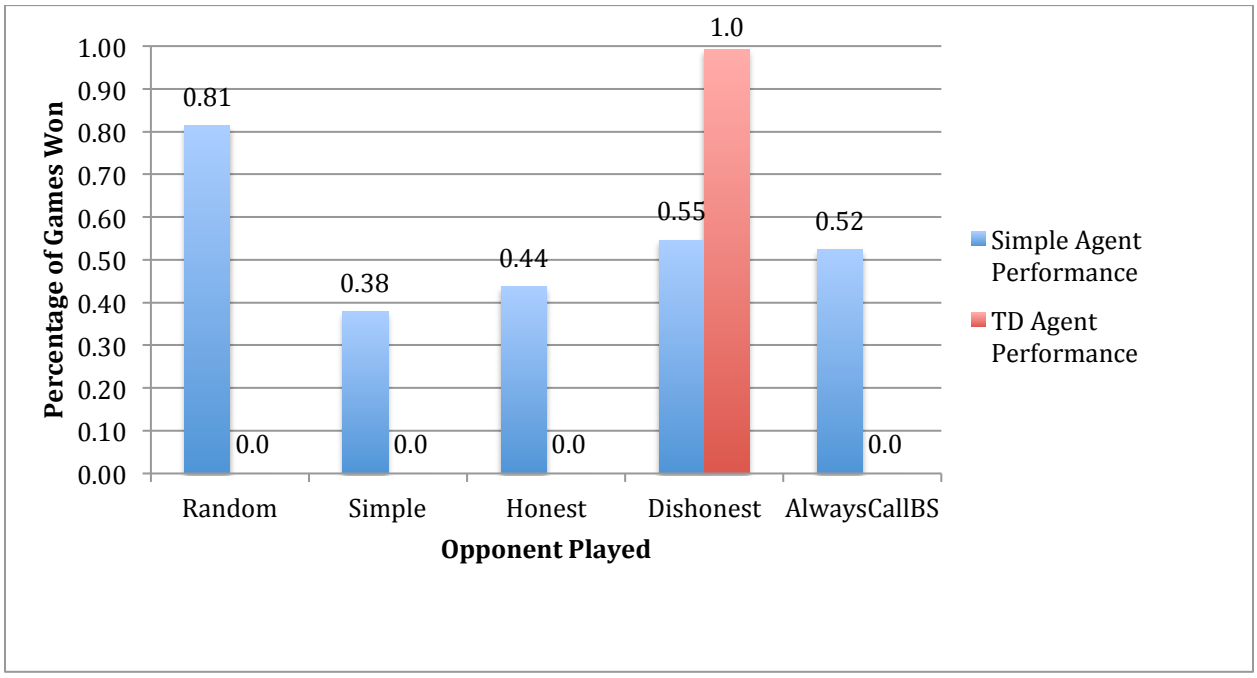
To create our initial weight vector W for training of the TD agent, we selected a Gaussian distribution and initialized one weight vector for each of the three TD agents in training. We kept separate weight vectors in order to avoid cycles when playing the game. Then, we trained the agents by playing all three TD agents against one another for 50 games. In each game, we performed a TD update on a player's weight vector after his turn. After all 50 training iterations, we then took the TD agent with the highest win rate, and played that agent against two Random, two Simple, two Honest, two Dishonest, and two AlwaysCallBS agents to see how well that particular TD agent played overall. We then repeated this entire procedure 10 times to get an average win rate for the TD agent.

One problem we faced when training the TD agents was the creation of cycles that led to non-terminating games. In order to solve this problem, we used separate weight vectors for each TD agent, which we found to create cycles much less frequently. Additionally, we terminated a game as soon as it exceeded 100 turns, and restarted it with a different card deck, since the vast majority of non-cyclic games terminated within 50 turns.

Experiment 1: Varying Weight Initialization

The main question we wanted to answer in this experiment was how varying the initialization of the weights during training would affect the final TD agent performance. To test this, we used different Gaussian distributions to get the initial values for W during training, as specified in Figures 3, 4, and 5. The results are summarized below:

Figure 3: Performance of TD Agent initialized with positive weights (Mean = 0.1, SD = 0.01)



Daryl Chang
Addison Weiler

Figure 4: Performance of TD Agent initialized with mean weight zero (Mean = 0.0, SD = 0.01)

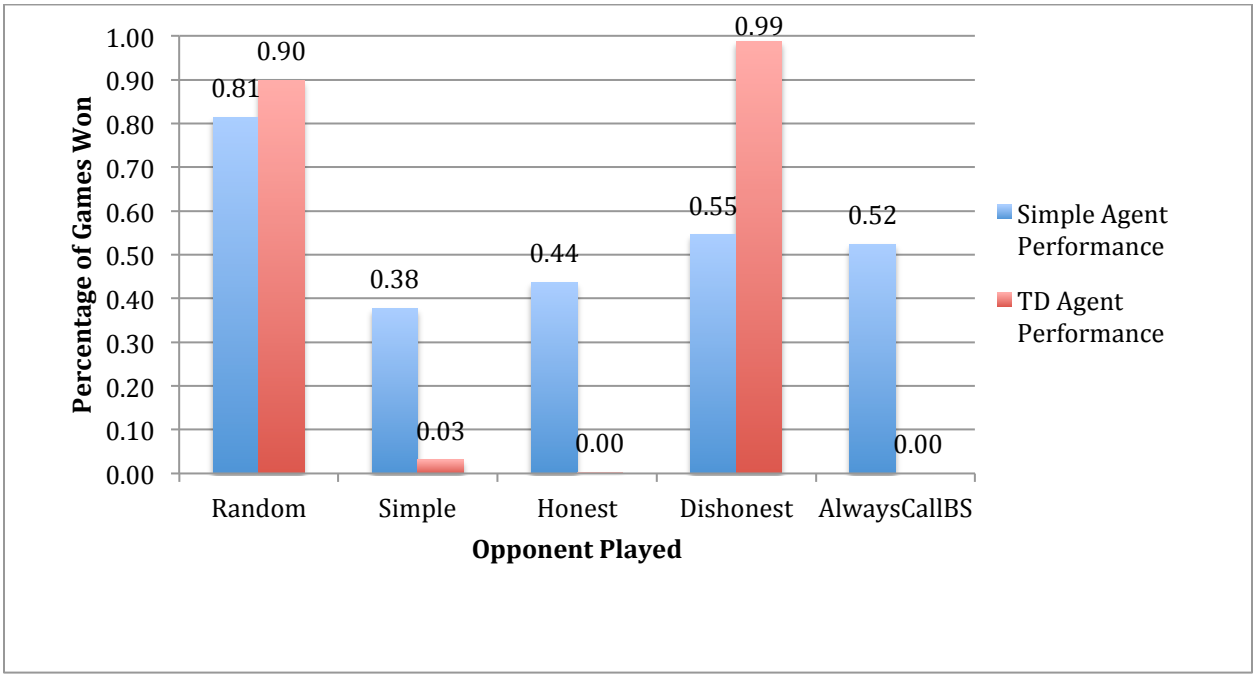
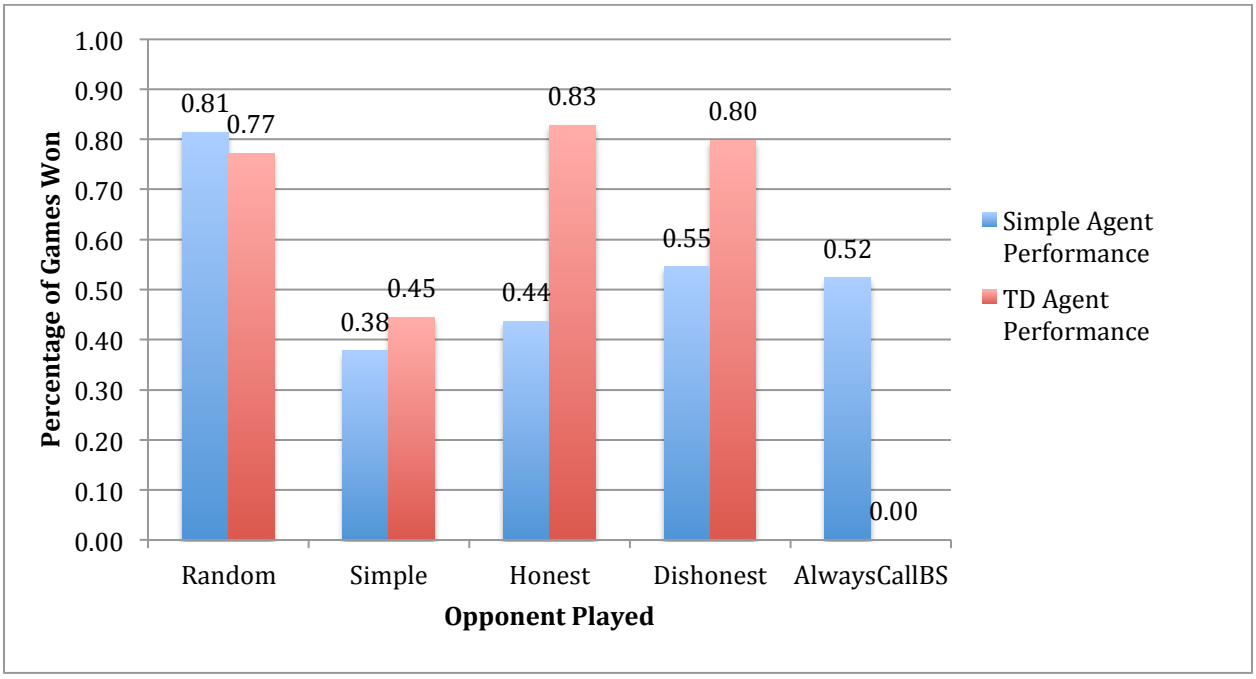


Figure 5: Performance of TD Agent initialized with negative mean weights (Mean = -0.1, SD = 0.01)



As the data show, the TD agent initialized with mean negative weights performed the best overall, although it always lost against AlwaysCallBS agent, regardless of the initialized weights. This is probably due to the fact that many of our features have a negative correlation with winning, such as how many total cards a player has and how many of each card they have.

However, the reason we decided to vary our weight vectors was because we were noticing that the TD agent was converging into one of two policies—"Trusting" or "Distrusting"—based on how the weights were initialized. In the "Trusting" policy, the agent acts almost exactly like the honest agent, and in the "Distrusting" policy, the agent acts like the Dishonest and AlwaysCallBS agent combined in that it always calls BS, but tries to play 4 cards each time it has a card turn. By initializing the weight vector with a negative mean, we would converge into the "Trusting" policy, which would do much better than the "Distrusting" policy overall.

As the data show, the "Distrusting" TD agents initialized with the zero and positive weights do very well against the Dishonest agent, because it has a high rate of calling BS, and thus will always uncover the Dishonest agent's lies. However, they perform poorly against relatively honest agents such as Honest, AlwaysCallBS, and Simple, because they call BS too often and thus end up with too many cards in their hands. The "Trusting" agent, on the other hand, avoids this problem and achieves a higher win rate overall.

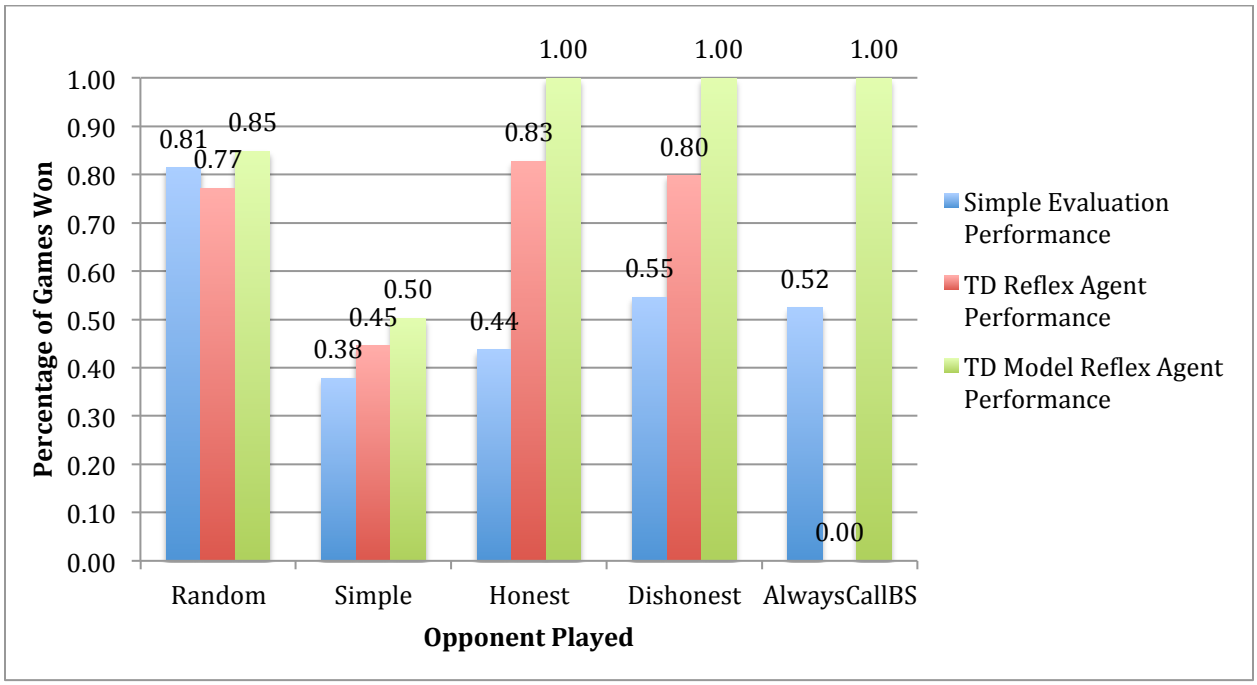
Why do zero and positive initial weights lead to a higher likelihood of calling BS? The final weight vectors produced by training show that the agents positively weight the number of cards in their own hands, effectively making it less bad to call BS incorrectly and add cards to their own hands. This explains the excessive number of BS calls. In contrast, the agents initialized with negative weights will negatively weight the number of cards in their own hands, preventing them from excessively calling BS on other players.

Experiment 2: Implementing a Model-Based Reflex Agent

The game BS involves hidden information, i.e. an agent does not know the cards in his opponents' hand or the policies his opponents are following. When calculating what combination of cards to put down, an agent has to make an assumption about the probability with which his opponents will call BS in order to evaluate the value of that combination. Lying, for example, would be bad if your opponents constantly call BS; similarly, always being honest would be disadvantageous if your opponents never call BS. When deciding whether or not to call BS on an opponent, an agent also has to make an assumption about the probability with which that opponent is lying. If the opponent is a pathological liar, the agent should call BS; if the opponent is honest, the agent should refrain.

In order to model the hidden information in the game, we decided to implement a model-based reflex agent (abbreviated as MBRA) based on the original reflex agent. The main difference is that the MBRA keeps two running probabilities for each player: one representing how likely it is that they will call BS, and another representing how often that player lies. By keeping these two probabilities, the MBRA learns the opponents' policies over the course of a single game. Thus, it is able to make a more informed decision about its own moves. The performance of the MBRA trained with TD learning is shown in Figure 6 below:

Figure 6: MBRA performance with negative mean weight initialization (Mean = -0.1, SD = 0.01)



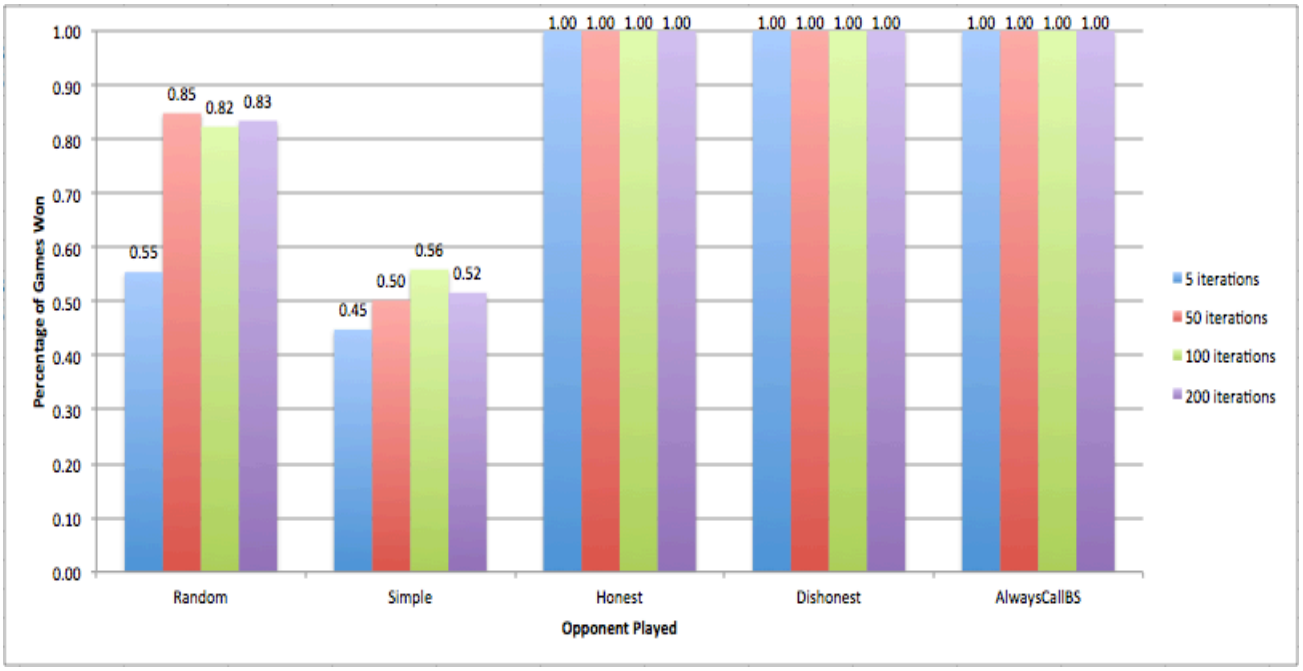
As shown by the data, the TD-trained MBRA does much better than the simple agent and regular TD agent in every metric. In particular, it attains a win rate of 100% against the honest, dishonest, and AlwaysCallBS agents because it is able to learn their fixed policies over the course of each game. In the case of the honest agent opponents, by the end of the game the MBRA puts down 4 cards each turn and does not call BS, since it knows that the honest agent will never call BS and will always try to play honestly. The MBRA adjusts its own policies accordingly for the dishonest and AlwaysCallBS agents as well.

Experiment 3: Varying Number of Training Games

In this experiment, we wanted to find out whether increasing the number of training games would improve the performance of the MBRA. We found that increasing the number of training games helped until around 50-100 games, when we then saw performance of the MBRA decline against the Random Agent. When we then ran the algorithm for 200 training games, we observed a significant drop in the performance of our MBRA against the Simple agent. We believe this occurred because of overtraining in our model: since we were only training our MBRA against other MBRAs, performance decreased when the MBRA played against an opponent whose policy could not be immediately determined. Against the Honest, Dishonest, and AlwaysCallBS agents, the MBRA achieved a win rate of 100% regardless of the number of training games, as it could learn the policies of these agents relatively quickly and respond to them with the best possible course of action.

Daryl Chang
Addison Weiler

Figure 7: MBRA Performance with Varying Number of Training Games

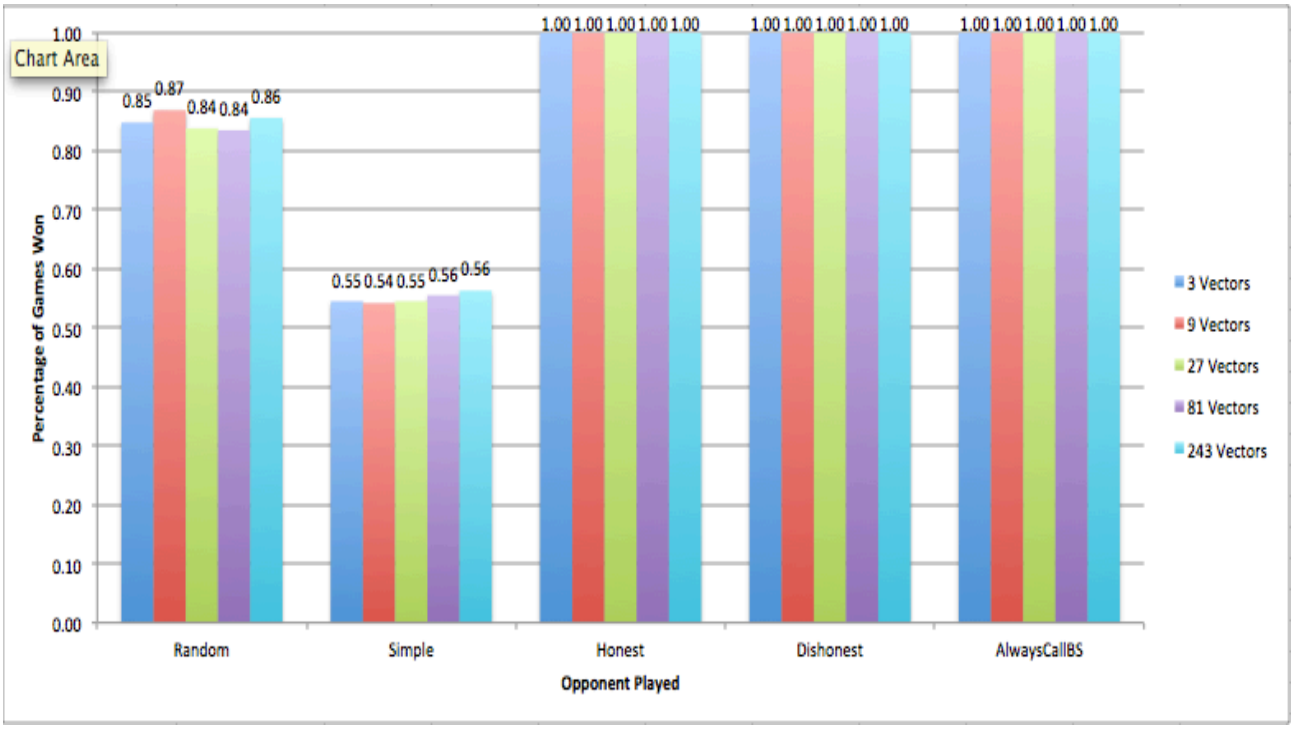


Experiment 4: "Tournament" TD Learning

The final thing we did improve the score of our MBRA was to implement "tournament-style" TD learning. Because there was the chance that our agents still could get in local minima, along with not reaching the best weights with the number of iterations given, to combat this we hypothesized that by running a "tournament" among differing weight vectors, we could get a weight vector that would be the most optimum. Starting with a varying number of players (3, 9, 27, 81, 243), we played "rounds" where three weight vectors would be assigned to model-based reflex agents, then play against one another to determine which one won the most times. The winner would then advance to the next round, until there was only an agent with the best weight vector left. TD updating still took place even in the higher rounds in order to continuously update the weight vectors to get to a better solution. The results for tournament play for 3, 9, 27, 81, and 243 weight vectors are shown below:

Daryl Chang
Addison Weiler

Figure 8: "Tournament" play between MBRA weight vectors
(Initialization for weight vectors: Mean = -0.1, SD = 0.01)



Results show that running a tournament to find the best weight vector did not improve performance of our model, as performance was relatively flat despite the number of weight vectors used. The baseline was 3 vectors, which represents one game of BS between three players. The reason that our "tournament" method could not have worked could be because many of the weights are similar in that they are all initialized with values in a Gaussian distribution with mean -0.1 and standard deviation 0.01, and thus could have all converged into similar minima, which would have made selection of the "best" weight vector arbitrary.

Conclusion:

In this project, we aimed to create an agent for BS that achieves a win rate substantially greater than that of a simple reflex agent. After modeling the game as a game tree and defining a feature set for each game state, we used TD learning to train the agents and evaluated its performance against a basket of opponent agents. We then experimented with four main modifications to improve its performance: varying weight initialization, implementing a model-based reflex agent, varying the number of training games, and implementing tournament-style training for a MBRA. We found that the biggest improvements in performance came from varying the initial weights and incorporating hidden information using the MBRA, while varying the number of training games and using tournament style training did not significantly change performance. Initializing the weights appropriately is important because TD learning will converge on local minima, one of which is a poor-performing subtype. Incorporating hidden information helps the agent learn an opponent's policy, which is especially helpful against agents with a fixed honesty policy. In sum, we have shown that TD learning can be successfully used to train an agent for BS, which has not been published before in literature.

Daryl Chang
Addison Weiler

Future Work:

In the future, we plan to train the MBRA against not just itself, but a variety of agents in order to more accurately reflect the opponents it will play against. For example, we could train it against a basket of simple, random, and honest agents and take the weight vector has the highest average win rate. We also plan to investigate how the number of players affects the win rate of the TD agent, since this project fixed the number of players at 3. We will look at the effect of increasing the deck size on the TD agent performance as well. Lastly, we will investigate the periodicity of a single game (i.e. plot the number of cards a player has over time) and look at how frequently the rank of a player changes. For example, does the rank of a player predict how likely he is to win, and can we incorporate that as a feature in TD learning? Finally, we can look at non-TD learning methods to train the agent, such as Bayesian inference or MDPs. Looking at such modifications will help to improve the agent and give a deeper understanding of its performance.