# Reinforcement Learning in Competitive Blackjack

Param Damle, Sid Burre, Kelvin Peng

## Task I: Exploration

We first explored the performance of an epsilon-greedy algorithm in the provided tic-tac-toe source code. By playing against the trained models, we could evaluate their performance. We found the following results for 1000, 5000, and 10000 epoch trained models vs. a human player:

| Epochs | Human Win/Tie/Loss |
|---|---|
| 1000 | 1/8/1 |
| 5000 | 1/8/1 |
| 10000 | 1/7/2 |

## Task II: Applications to Blackjack

### Game Setup

For our entertainment game, we chose to slightly modify the classic game of Blackjack. In traditional Blackjack, the player is dealt 2 cards and decides to successively hit (get another card) or stand (end his turn), trying to attain the highest hand total value less than or equal to 21. Going over 21 is called "busting", and you automatically lose. In regular Blackjack, the player only faces the dealer, who is programmed to keep hitting until a value 17 or greater, and then stand. The values are determined as such:

- Number cards 2-10 count for their own value
- Jack, Queen, and King count for 10
- Ace counts for 1 or 11, whatever is most beneficial to the hand

To allow multi-agent competition, we modified Blackjack to include any number of players playing in succession, where the players with the highest value hands ($\leq 21$) win. For the majority of the experimentation, we will be playing a game with 2 players (to eventually attain human vs bot gameplay) and 1 dealer. We include the dealer, who is hard programmed to keep hitting until 17, so the game retains some incentive structure to force all players to make some risky hits. Without a reason to try to get a hand 17 or greater, the game may degenerate into a battle of hands with values around 12 to avoid busting at all costs.

We modeled our simulation as such:

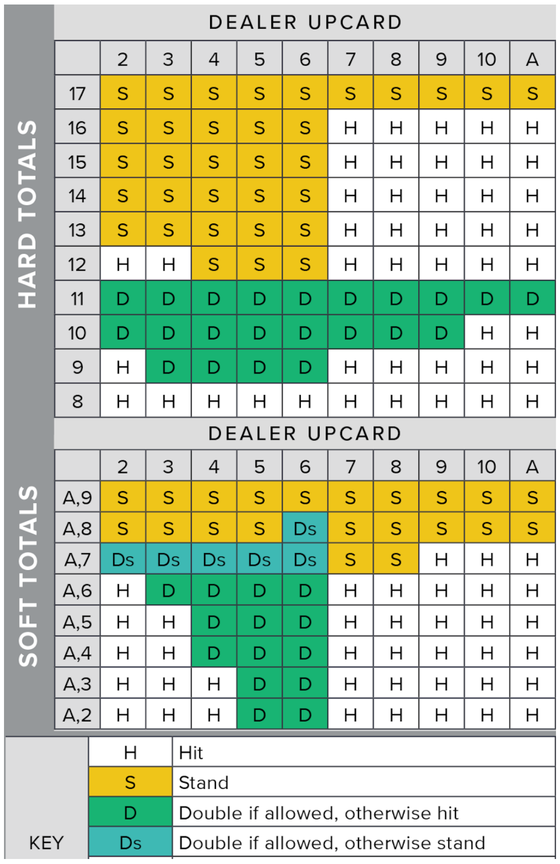
1. 2 cards are dealt to every player, and 1 card is dealt to the dealer. All players have knowledge of all cards dealt prior to the start of their turn.
2. Player 1 goes, and has 2 actions: hit (request another card) or stand (end turn). If he busts, his turn is automatically ended.
3. Player 2 goes after Player 1, and can see all the dealt cards. This player-by-player run continues until all players have gone.
4. The dealer plays last, and hits until a hand value 17 or greater is reached.
5. In the end, the winners are the participants with hand totals the closest to (but not over) 21. Net reward for each winner is equal to the number of participants (number of players + 1 dealer) divided by the number of winners (to distribute winnings evenly) minus 1 (for initial bet). Net reward for all losers is -1 for losing initial investment.
6. Agents use net reward to adjust their probabilities of hitting or standing in the next game. This is where reinforcement learning comes in.

## Agent Setup

When creating agents to play this game, we considered two variables when adopting our strategies:

- **State-Specificity**: A non-state-specific strategy tries to learn 1 action profile: a general probability for hitting as opposed to standing. This probability is the same regardless of the current hand or dealer's hand. A state-specific strategy has different probabilities for hitting or standing depending on the current hand total, represented by 26 states.
    - The states correspond to hands that contain an Ace and those that don't. Any Ace beyond the first Ace in a hand must count for 1, as two 11's in a hand would make it bust (over 21). Thus, the states align with the following rules:
        - States 0-16 correspond to hands *without an Ace* that have a total value of 4-20, respectively. Anything under 4 is impossible without an Ace, given each player is dealt 2 cards in the beginning. Any hand totaling 21 is optimal, and thus an automatic stand. Any hand above 21 is an automatic bust, so no state is needed.
        - States 17-25 correspond to hands *with an Ace* where the other cards total 1-9, respectively. Any other total greater than 9 is either an automatic stand or a bust.
    - For our simulation, we ignored the Blackjack rules on hand splitting or doubling down, to allow for a more streamlined learning and playing experience.
- **Strategy**: Our strategy relates to how the agents generate action profiles (probabilities for standing vs hitting) based on their belief probabilities. The strategies are:

- 0. Risky/Naive Strategy: Based on the classic EXP3 weight update, action_prob = $(1-\gamma)*$*belief* + $(\gamma * 1/2)$
- 1. Cautious/Game-Programmed [Only State-Specific]: Using the cards face up on the table (cards dealt up till this point in the game), figure out the probability of busting by drawing the next card and do not go over this. action_prob = $(1-P^{bust})*$*belief* + (with probability $P^{bust}$, *stand*)
- 2. Mixed Strategy [Only State-Specific]: A mix of both earlier strategies. action_prob = $(1-\gamma)*((1-P^{bust})*$*belief* + (with probability $P^{bust}$, *stand*)) + $(\gamma * 1/2)$
- 3. Hardcoded Strategy [Only State-Specific]: We included this to show our learned model could outperform standard strategies preexisting. Our pick was based on the standardized Strategy Chart for Blackjack, available widely online, depicted to the right.



| HARD TOTALS | DEALER UPCARD | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | A |
| 17 | S | S | S | S | S | S | S | S | S | S |
| 16 | S | S | S | S | S | H | H | H | H | H |
| 15 | S | S | S | S | S | H | H | H | H | H |
| 14 | S | S | S | S | S | H | H | H | H | H |
| 13 | S | S | S | S | S | H | H | H | H | H |
| 12 | H | H | S | S | S | H | H | H | H | H |
| 11 | D | D | D | D | D | D | D | D | D | D |
| 10 | D | D | D | D | D | D | D | D | H | H |
| 9 | H | D | D | D | D | H | H | H | H | H |
| 8 | H | H | H | H | H | H | H | H | H | H |

| SOFT TOTALS | DEALER UPCARD | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | A |
| A,9 | S | S | S | S | S | S | S | S | S | S |
| A,8 | S | S | S | S | Ds | S | S | S | S | S |
| A,7 | Ds | Ds | Ds | Ds | Ds | S | S | H | H | H |
| A,6 | H | D | D | D | D | H | H | H | H | H |
| A,5 | H | H | D | D | D | H | H | H | H | H |
| A,4 | H | H | D | D | D | H | H | H | H | H |
| A,3 | H | H | H | D | D | H | H | H | H | H |
| A,2 | H | H | H | D | D | H | H | H | H | H |

| KEY | | |
|---|---|---|
| | H | Hit |
| | S | Stand |
| | D | Double if allowed, otherwise hit |
| | Ds | Double if allowed, otherwise stand |

Above, $\gamma$ is a learning parameter that scales down with the training iteration, *belief* is a state vector that specifies the stand/hit probabilities for each hand value, and *stand* is a vector featuring a 100% probability of standing. The $P^{bust}$ value was calculated by inspecting both the hand the player possessed as well as all the other face up cards at the table, and represents the probability that the next card drawn will bust the current hand.

In total this gave us 1 non-state-specific agent and 4 state-specific agents. Our reasoning for making the states dependent on the value or presence of an Ace is this is what affects whether a new card will cause a bust (loss) or get the player closer to 21 (win). We hoped to show dominance of a mixed, state-specific strategy over both non-state-specific and Our hypothesis was that the mixed strategy (3) would ultimately be the most optimal, and consistently beat the other strategies.

## Training

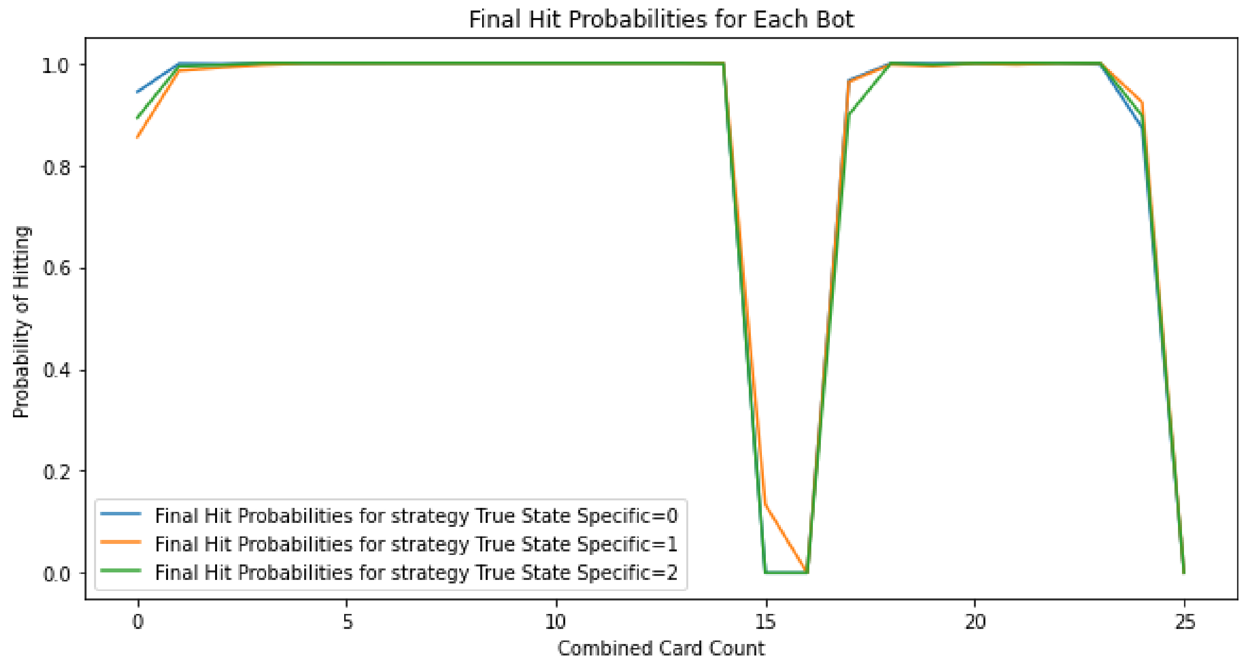
Although we had 5 different settings of bots, we only had to train 4 of these bots as the fifth bot featured a hard-coded strategy that did not require tuning. We employed three different learning mechanisms total:

Type I: bold, epsilon greedy, naive, Strategy 0

Type II: Cautious, Game-Programmed, Strategy 1
Type III: Mix of cautious and risky, Strategy 2

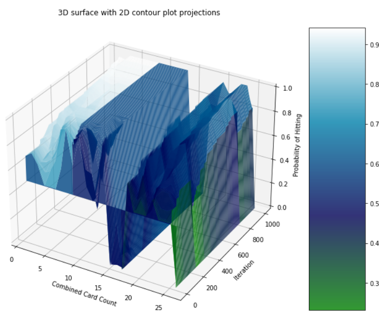
Type 1 featured a non-state specific implementation that had an identical stand/hit probability, regardless of the current hand.

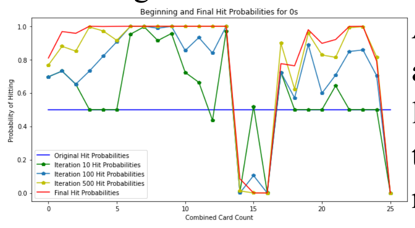
Bots would train at a 2-player table (along with a dealer) for ~4000 epochs, gaining a reward equivalent to 1 if they beat the dealer, -1 if they lost to the dealer, and a fractional value equivalent to the number of other players that tied the dealer if the player tied. Each of the bots would have identical policies (belief) that were consulted and updated on a turn by turn basis. This was an advantageous training schema as it trained bots to create a generalized strategy for all positions at the table. Of the four bots we *trained*, only three had state specific strategies and the following comparative final hit probabilities are displayed below:

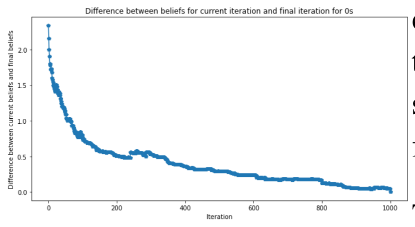


As one can see,  each bot had nearly identical strategies, differing primarily along the combined card count values of 15, 17, and 24. Interestingly enough, the Type II strategy that was labeled a conservative strategy featured a more aggressive hit rate than the other strategies, hitting even where the other strategies chose to stand. The policy described here differs significantly from that of a traditional blackjack strategy where players are continually less likely to hit after the card hand exceeds 15. This may be due to the fact that the bots could condition their hit probabilities based on the state of the table, relying on state-information not encapsulated in the hand to inform their actions. Also, if a bot is less likely to be risky with its hits, it is less likely to face repercussions after a hit and thus less likely to lower its chance of hitting again.

Pictured below are iteration based training information for the three state-based bots:

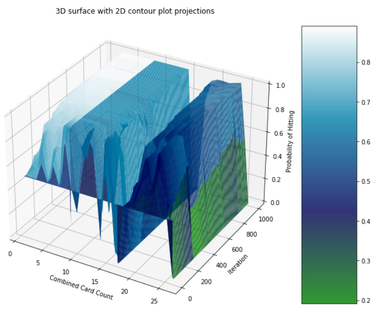


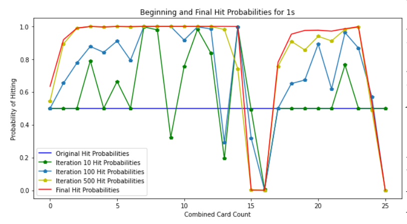


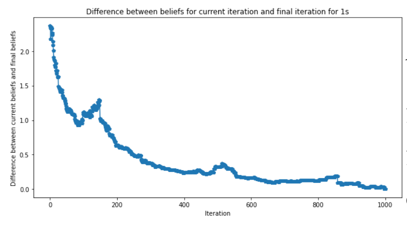


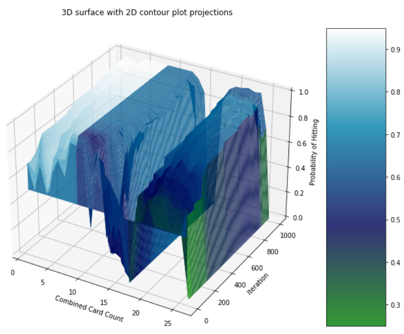
As one can see, each of the bots accomplished most of their learning relatively early on into their training, achieving the majority of their learning by epoch 600 out of 1000. Further training was tested but no significant improvements were found.

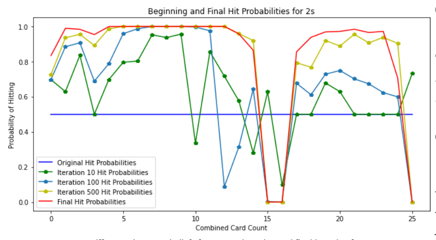
Take notice, the jumpiness featured in the policies within the first couple of iterations. Each bot features different convergence behaviors prior to descending on its final stand probabilities. Learning method 1 presents a steady, gradual iteration on its beliefs until it finally decides on a final belief. Strategies 2 and 3 feature wild oscillations before descending on a final policy, particularly for make-or-break combined card counts between 12 and 17 where variability with the layout of the table and behavior of other players matters most.

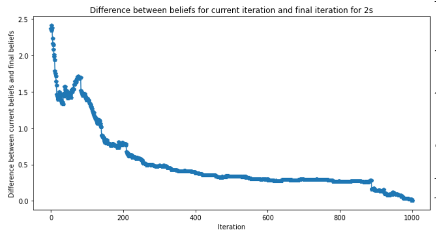










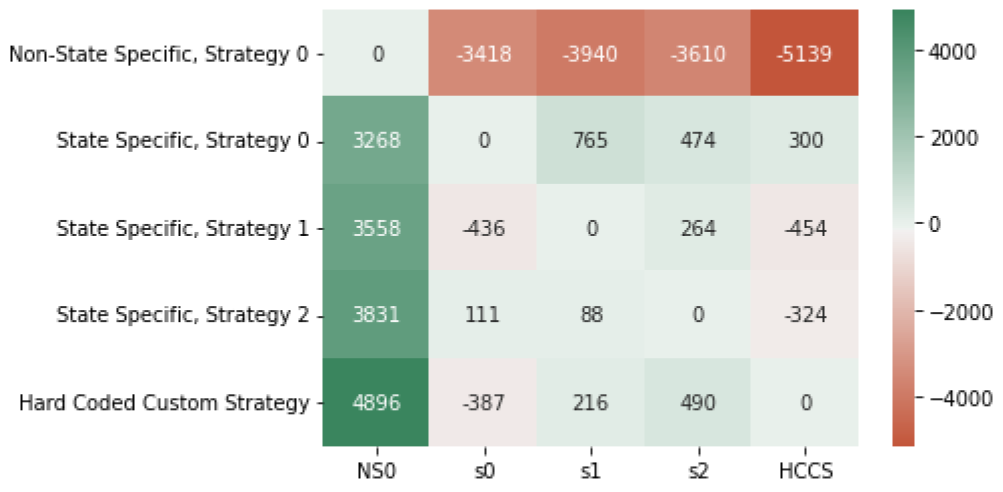


Finally, the learning rates of a all bots was calculated by computing the vectorized distance between the policy at the current iteration and the final policy. This produced interesting results, similar to those observed by inspecting the top right graphs. Both strategies 2 and 3 feature an increase in distance from their final policy near iteration 175. This is an interesting deviation as they dramatically correct the course, in an almost identical fashion. This is indicative of the consideration of the P[Bust] value that is unique to strategies 2 and 3.

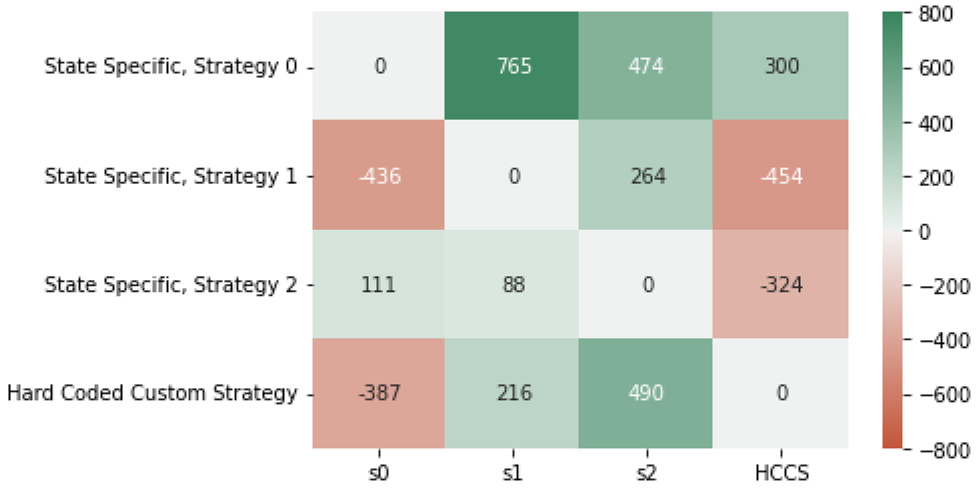
## Testing

We played each agent against every other agent, round-robin style, with both agents taking turns going first. This was important because the player who goes second has the advantage of knowing where the earlier players stood or bust. Each permutation pair matching was run for 1000 epochs, and the total net rewards of the first player were compared to that of the second.

The value in each cell of the heatmap represents a proxy for winrate. As explained above, winning in Blackjack cannot be reduced to win rate as winning in blackjack is effectively a doubling of the money betted. Instead, a proxy for winning was used to represent the matchup based outcomes. Each of the matchup tables below has a specific layout. The rows represent the games where the bot listed to the left of the column went first and the columns represent games where the bot listed at the bottom of the column went second. The bots are in the same order regardless of row or column (top-bottom, left-right) so abbreviations are used for the columnal labels.



As one can see the non-state specific strategy loses handedly to all the other bots, this is expected as a blackjack strategy based solely on a single probability is likely to lose.



After removing the games with the non-state specific bot, we get the bottom table. In this table we see that no strategy is strictly dominant, and all strategies display weak dominance. Surprisingly enough, we find that State Specific, Strategy 0 (Type 1) demonstrates strict dominance over state specific strategy 1 and the hard coded custom strategy and weak dominance when playing State Specific, Strategy 2, the strategy that was deemed to be mixed. Interestingly, the only place where weak dominance is expressed is when Strategy 0 plays Strategy 2, which makes some sense. Both of these strategies are highly dependent on the information offered by the cards at the table. So by definition, if one player gets more information than the other (by going last), then it is less likely that it will win, solely because it has less information to go off of.

## Conclusion

In conclusion, we found that only our Strategy 0 bot could pull off any meaningful wins against the benchmark hard-coded strategy that followed the strategy table to the left. This is an interesting conclusion that is highly unexpected. If anything, Strategy 3 or Strategy 2 should've been the likely winners. But because they are not winners, displays something very interesting about Blackjack; it is more efficient to play risky and loose rather than to stand. We believe that strategies 2 and 3 were simply too conservative in their final play, most likely a product of being trained against identically strategizing conservative opponents.

Future experimentation should focus primarily around three variables; the training schema used, the epsilon/gamma training parameters, and training epochs. Manipulating training schema would focus primarily on the bots that are learned against. In our current method bots are trained homogeneously, playing against 4 other copies identical to them. But would training results change if bots were exposed to different strategies early on? Or even if they were trained against rotating adversarial strategies? Epsilon/gamma and epochs are primarily for inter-bot competition. Does each successive epoch truly better a bot or not? This question would be addressed by having bots of different epochs committed. This is similar for epsilon/gamma manipulation. Do different epsilon/gamma values make bots more or less effective?

## Resources

The source code for the project can be found here: https://github.com/paramsdamle/blackjack

A video demonstration of the project can be found here: https://youtu.be/qSKaci_2B64