Introduction:

Blackjack is marketed as a simple game, one that casino novices should pick up with ease. The premise is simple: draw cards until your total card value is as close to 21 as possible and you beat the dealer’s hand. Past analysis on the game has been done, and it’s been found that blackjack has the best odds of any table game in the casino, with around a 0.5% house edge when played properly.

While casino novices often decide to risk their money on the blackjack table, they often don’t play with the optimum policy. Blackjack is a game with many individual states (hundreds of states for just a 52-card deck), and knowing how to maneuver through all of them to give yourself the best chance to win can be confusing.

Some companies try to capitalize on this by selling ‘Basic Strategy Cards’, which claim to give the player information on how to approach betting at every state. However, these cards don’t account for differences in state such as the number of cards left in the deck and the card count at certain points. I believe that I can make a program that can improve the basic strategy of the starting player, by providing information on how to bet, and what their odds are in every situation on the board.

Project Goal:

The goal of this project is to evaluate the card counting strategy and decide how useful it is as a metric to gain advantage playing a game of blackjack. Doing this will first involve deriving an optimal policy by which to play card counting blackjack, and the testing this policy through simulation. Different betting policies will also be tested, to vie

Summary:

An MDP was created for a simplified blackjack game using a card counting model. The simplification is with regards to an assumption made regarding the probability of the cards in the game. For a given true count value, the probability of a certain value card being drawn is fixed. Using value iteration, an optimum policy and expected utility were calculated for all states.

A more realistic Blackjack MDP was created. This MDP accounts for the total cards remaining in the deck and adjusts the probability of an individual card being drawn after every move. Using simulation, a Monte Carlo Model Fixed Simulation was run using the fixed policy learned via the simplified blackjack model. The expected utilities of the simulated cases were compared to those produced by the value iteration process.

A further probabilistic analysis was conducted using the expected values learned via the value iteration process and the PDF distribution of true card count values. Using a further Monte Carlo simulation, a distribution of card count values expected for a full deck blackjack game were found. Using these pieces of information, the approximate value of different betting policies could be approximated.

Simplified Card Couting MDP:

Purpose: This simplified model was created in hope of producing an optimum policy for how to play BlackJack. This MDP operates under the assumption that the probability of drawing a low-card (2, 3, 4, 5, 6), mid-card (7, 8, 9), or high-card (10, J, K, Q, A) is completely dependent on the true card count. Thus, for this MDP, the transition probability for drawing a certain card doesn’t change, regardless of what previous cards have been drawn. Though this may seem counter intuitive, because this is stuff. However, this MDP isn’t make to

This MDP

State: (PlayerState, DealerState, CardCount)

PlayerState: {##}\*{keys}

The PlayerState is a string that held information about the player value, as well as information about what possible transitions the player could take in the next turn.

{##} – Numbers that tell the program the current value in the players hand. For example, a 6 and a 7 would generate a value of 13. We do not pass information about the previous cards other than the value in order to simplify the state space.

{keys} – Keys are letters added onto the string that provide information about possible next moves. If ‘D’ or ‘S’ are in the key portion of the state, then this means that a double or split move is legal on the next turn. If ‘A’ is in keys, this mean the player has an ace in their hand with value 11. When the ace has been incorporated into the value with value 1, then the ‘A’ key is deleted.

Actions: Actions are relatively limited for a game of Blackjack:

* Stay: Player stops drawing cards. The dealer will now draw cards from the deck until there hand value is in range 17 to 21.
* Draw: The player draws 1 card. If the player exceeds 21, he or she loses.
* Double: The player doubles there bet and draws one card. On the next move, they must stay.
* Begin: The player draws 2 cards and the dealer draws one. If the player loses

Transitions: The transition probabilities for the simplified MDP are directly related to the card-count and mid-card value. These transition probabilities do not change throughout the MDP. If more than one card is drawn per action, then the transition probability would involve multiplying the individual probability of P(cardA) \* P(cardB) … \* P(cardN).

Rewards: Rewards are only earned at the end of the MDP. Typically, if the player wins, they earn betting 1 unit, and if they lose, they lose one betting unit. All ties result in a push, meaning there is no reward. The bet can be altered in the initialization of the MDP. Some edge cases includes ‘BLACKJACK’, where the player gets a score of 21 in the first turn and the dealer does not, the player receives 1.5 times their bet value. Additionally, if the player chooses to ‘double down’ on a hand, the reward (whether win or lose) will be doubled.

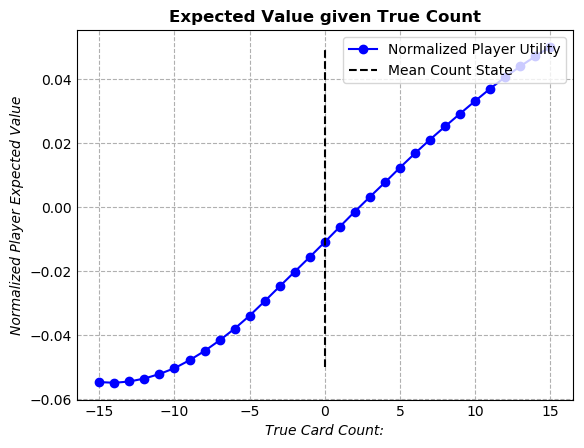
Variable Inputs:

* Count Value: An initial count value can be specified for the MDP. This count value is constant and will not change in any successor states.
* Mid-Card Value: The mid-card value is defined as the number of mid-cards (7, 8, 9) per 52 card deck. After a shuffle and reset, the number of mid-cards would be 12. However, once a blackjack game begins, this number is variable. The mid-card value affects the probability of drawing any card, and is a necessary value as count doesn’t provide any information about the concentration of mid-cards in the deck.

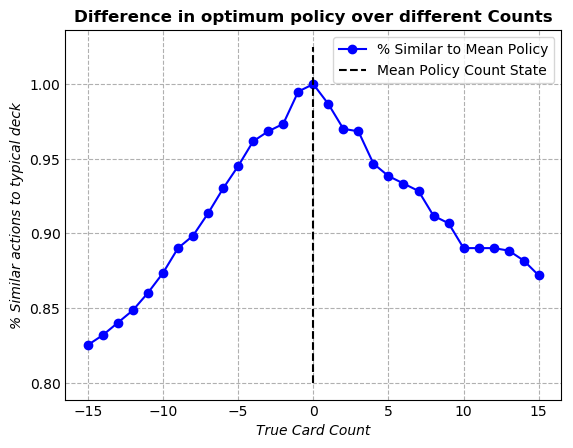
Algorithm: Value iteration was used to solve this MDP and produce an expected utility and optimal policy. This algorithm was run many times for different count start states.

Card Count Analysis: The MDP was run for 31 different values of true card count (-15 to 15 incremented by 1). The optimum policy and expected utility was saved for each MDP.

The expected value for the starting state of each MDP trial was extracted and plotted. As the true card count increases, the expected value of a hand increases approximately linearly, showing that the player achieves greater returns when playing optimally with a higher count deck. An important feature is the point where the expected utility changes from negative to positive, which is the point where the player now has the edge over the casino. This is the point in time where the player should want to bet big, since they should win more money than they lose. When the count approaches the large number of 15, the behavior of the expected value curve seems to hit an asymptotic maximum.



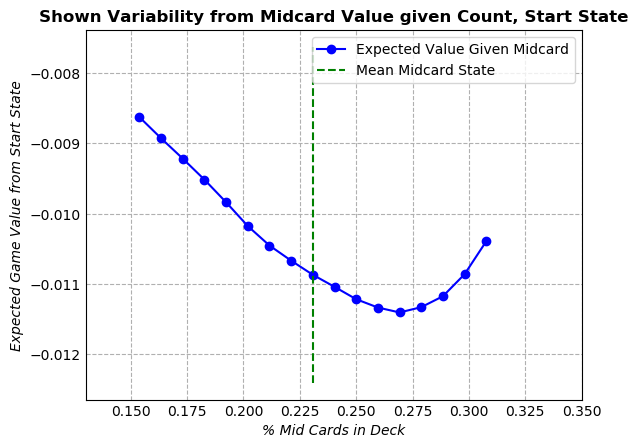
In addition to variable expected utility for different card states, the optimum policy for playing the game also changes with card count. The plot below shows the percentage of actions in the optimum policy that are similar to the optimum policy for a count of 0. As the count increases or decreases, the optimum policy continually changes. If a player wated to really make use of a card counting strategy in the casino, they would have to adjust their playing policy based on the count state of the deck.



Mid Card Analysis:

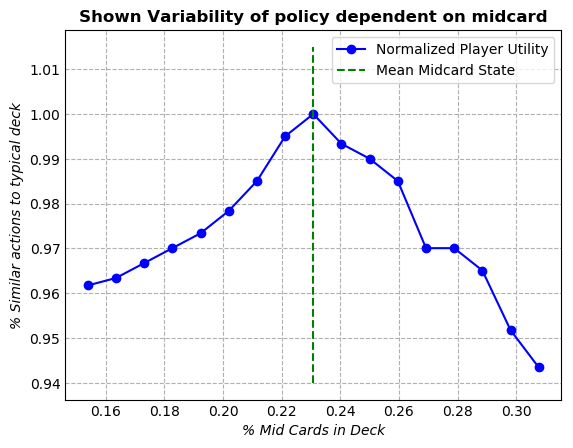
While card count is a useful metric to understand relative high-card and low-card probability, it does not give any information regarding the concentration of 7, 8, 9 (mid-cards) in the deck. For the previous value iteration experiment, it was assumed in the MDP that there will be 12 mid-cards for every 52 cards in the deck. However, in an actual game, this will not always be the case. For this experiment, the number of mid-cards in the deck was changed and the value iteration was run (the count was kept at an arbitrary value of 0).

The expected utility from the start state of a hand was found to change with respect to the number of mid-cards in the deck. However, this amount is fairly small (only about .2% of the bet). Additionally, the expected value seems to change favorably as values of mid-card move away from the mean (the mean is almost at a local minima).



The one possible flaw for this reasoning is that this mid-card analysis was conducted over only one count value. To be complete, the analysis could be conducted over many different count values. If expected utility was not found to vary significantly over different mid-card values for all different values of count, we could conclude that card count alone is a sufficient identifier to estimate the expected utility. However, due to lack of resources and time (and generally wanting to move into more interesting analysis) this aspect wasn’t completed.

In addition to slight expected value change, mid-card changes also induced slight variability in the policy as well. This was deemed to be insignificant, as there was no more than a 6% difference in policy actions between mid-card states, which shouldn’t contribute to much change in our expected value.



Accurate Blackjack MDP:

Purpose: Created a more representative MDP for an actual casino blackjack game in order to test the optimum policies learned in the previous value iteration MDP. In this model, exact card probabilities are calculated and probabilities of drawing cards without replacement is considered.

State: (playerState, dealerState, CardsRemaining, CardCount) PlayerState and dealerState are represented the same as in the approximate MDP. The new variable ‘CardsRemaining’, is a tuple of integers that represents the number of each card remaining in the deck. The ‘CardCount’ variable is no longer a constant, and is changed to reflect the current deck state. This value is rounded to the nearest integer value.

Actions: Actions are the same as the previously simplified MDP. No new actions are added.

Transitions: Transition states were calculated based on the ‘CardsRemaining’. The probability of each card being drawn is [CardsRemaining[i]/sum(CardsRemaining) for i in len(CardsRemaining)]. In the case where multiple cards were drawn as the result of an action.

Reward: The rewards are the same as those previously stated in the simplified problem.

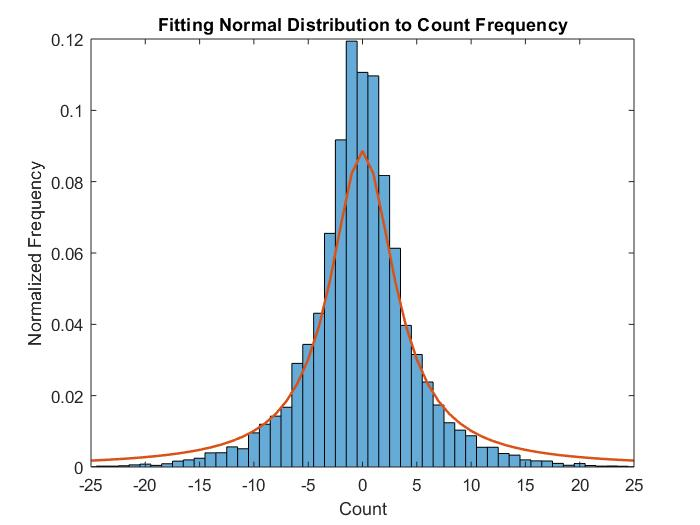
Simulation:

A Monte Carlo Model Fixed simulation algorithm was run using the succProbReward function of the accurate Blackjack MDP. The optimum policy created and saved from value iteration was used as the policy in the simulation. Using this method, we can test the optimum policy and observe the exact value.

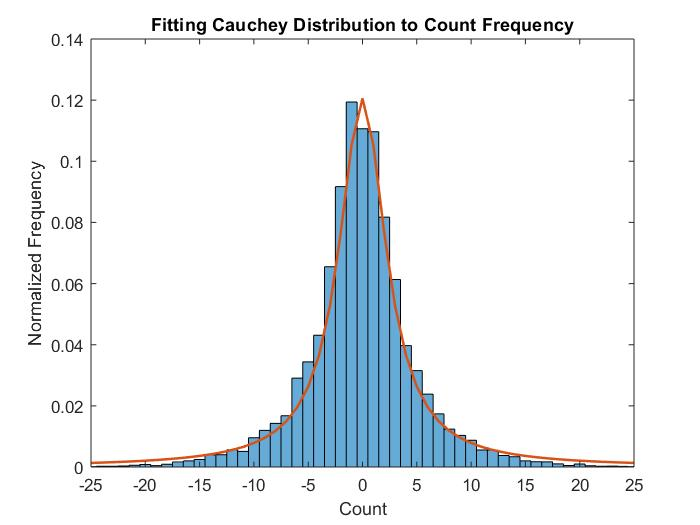
Probablistic Analysis of Expected Value using Simplified MDP Results:

P(Count) – (PDF of Count

A set of true card count values was found by simulating 45,000 games of blackjack. A normal distribution and a Cauchey distribution (adjusted normal distribution) were used along with a least squares method to find the best fit for the data. The results of the fitting can be seen below.



\te



It was determined that the Cauchey Distribution was a better representation of the true count distribution data. Thus, we use this continuous PDF to represent the data.

Unfortunately, P(Count) != PDF(Count) for a continuous distribution function such as a Cauchey distribution. However, we can discretize the Cauchey distribution at each count value with each count value being one unit apart. This will create a discrete PDF function that will sum to one, and can be used to integrate to find the expected utility.

Π\_bet –

This is a policy function that plots initial bet amount against the true count of the game. Five different betting policies were created and tested.

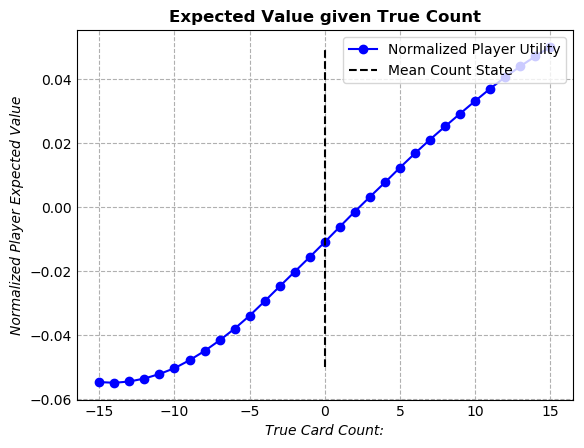
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Policy | | | | |
| # | Policy vector | If count < 0 | If 0<count<2 | If count>2 |
| 1 | [.5, 1, 5] | .5 unit | 1 unit | 5 units |
| 2 | [0, 1, 5] | 0 | 1 unit | 5 units |
| 3 | [0, 0, 5] | 0 | 0 | 5 units |
| 4 | [1, 1, 1] | 1 unit | 1 unit | 1 unit |
| 5 | [.5, 1, 2.5] | .5 unit | 1 unit | 2.5 units |

Plot of policies

In general, policies were created to improve player expected value. Thus, for higher values of count where the player has a larger chance of winning, the bet is raised.

E[Reward | Count] –

This expected value function is the starting utility of the simplified MDP for different count values.



By solvi\\

ng equation one, we can get an estimate of the expected value for each betting policy.

|  |  |
| --- | --- |
| Policy | Estimated Utility from Simplified MDP |
| [.5, 1, 5] |  |
| [0, 1, 5] |  |
| [0, 0, 5] |  |
| [1, 1, 1] |  |
| [.5, 1, 2.5] |  |
| [.5, 1, 5] |  |

The plot of the integrand is the deaggregation of the utility. These plots can be seen below. These plots show which values of count are contributing most to the expected value. If values on the deaggregation plot are negative, they are associated with negative utility, and vice versa. The goal here is to maximize the positive area on this plot and minimize the negative area.

Deaggregation plots.

Simulation Test:

* Simulate many hands of blackjack given Monte Carlo Analysis’

For each of the five betting policies, a simulation algorithm was run playing approximately 50,000 games of Blackjack. The average expected value of each hand was calculated and compared to the expected utility found via the probabilistic analysis.

|  |  |  |  |
| --- | --- | --- | --- |
| Policy | Estimated Utility from Simplified MDP | Simulation Utility using learned optimal policy | Residual |
| [.5, 1, 5] | .0251 | .0315 | .006 |
| [0, 1, 5] | .0322 | .00091 | .0231 |
| [0, 0, 5] | .0338 | .0628 | .0390 |
| [1, 1, 1] | -.0089 | -.0020 | .0069 |
| [.5, 1, 2.5] | .0082 | .0102 | .002 |

The expected value residuals were slightly higher than expected. However, the expected values from the simulation are both higher and lower than the expected values from the simplified MDP, showing that the simplified values aren’t biased.

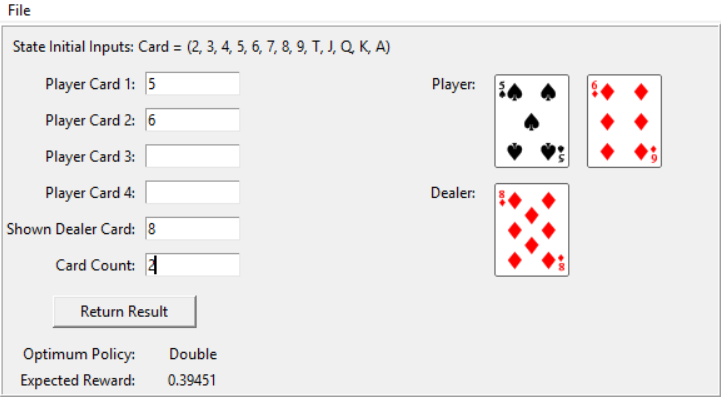
Thus, it seems likely that the source of error between the two values is due to a lack in number of trials. This is especially true with the betting policies that bet high amounts on counts that are greater than 2. Only about 25% of hands are played with a count greater than two, and betting a large number on a smaller amount of hands introduces variation into the expected value. Overall, I believe that the simplified values are good.

Result Summary and Error Analysis:

Final Product:

A Blackjack card counting program application was made to display the optimum policy learned during the value iteration process. For any player training about how to play blackjack more efficiently, this is an excellent way to do so. Inputs to the GUI are player card values, dealer card values, and the current card count at the time. The player card and dealer card fields can be left blank if the user wants to know the expected reward associated with a certain true count at the start of the round.

The application works by using a database of the optimum policy and expected values generated during the value iteration process. This application is basically an easier way to parse through the database and receive information based on your current state. This is a great way to train yourself for any blackjack situation, and a great learning tool for those looking to improve their game and win some money.



Disclaimer:

This project was not made with the intent to encourage gambling. While counting cards is completely legal, it is often looked down upon by casinos due to the edge that it can give the player. Please gamble at your own risk!

References:

Jensen, Kamron, "The Expected Value of an Advantage Blackjack player" (2014). All Graduate Plan B and other Reports. 524

Professional Blackjack by Stanford Wong, page 31, 1994 ed.

\documentclass[a4paper, 11pt]{article}

\usepackage{comment} % enables the use of multi-line comments (\ifx \fi)

\usepackage{lipsum} %This package just generates Lorem Ipsum filler text.

\usepackage{fullpage} % changes the margin

\usepackage{graphicx}

\usepackage[utf8]{inputenc}

\usepackage[english]{babel}

\title{UploadingImages}

\author{team+learn }

\date{\today}

\begin{document}

\tableofcontents

\newpage

\section{Introduction}

Blackjack is marketed as a simple game, one that casino novices should pick up with ease. The premise is simple: draw cards until your total card value is as close to 21 as possible and you beat the dealer’s hand. Past analysis on the game has found that blackjack has the best odds of any game in the casino, with only a 0.5\% house edge when played properly.\\

\\

While casino novices often decide to risk their money on the blackjack table, they often don’t play with the optimum policy. Blackjack is a game with many states (over 80,000 for just a 26-card deck), and knowing how to maneuver through all of them to give yourself the best chance to win can be confusing. \\

\\

Some companies try to capitalize on this by selling ‘Basic Strategy Cards’, which claim to give the player information on how to approach betting at every state. However, these cards don’t account for differences in state such as the number of cards left in the deck and the card count at certain points. These additional factors can be leveraged to give you an edge against the casino. I believe that I can make a program that can improve the basic strategy of the starting player, by providing information on how to bet, and what their odds are in every situation on the board.

\section{Project Goal}

The goal of this project is to evaluate card counting as a strategy and decide how useful it is in gain advantage over the casino. Doing this will first involve deriving an optimal policy by which to play card counting blackjack, and the testing this policy through simulation. The utility of different betting policies will also be analyzed, and a recommendations will be made to the player on how to succeed in the game of Blackjack. A basic tool will be developed to display the optimum strategy to the player through an easy to use interface.

\section{Basic Overview}

Blackjack is a card game, where the player (agent) plays against an adversarial dealer with a set policy. The goal of the player is simple: get a hand value as close as possible to 21 without going over while also beating the dealer. The player can keep drawing cards until they are satisfied with their hand. The dealer must keep drawing cards until the value of his hand is between 17 and 21. \\

\\

Card counting is a strategy that can employed by the player that gives them an advantage over the dealer. In a normal game, the casino will have anywhere between a 0.5 - 1 percent edge over the player. However, while using a card counting strategy, you can gain information about the relative probabilities of cards left in the deck. This all comes from knowing the value of 'True Card Count', which is the number of low cards (2, 3, 4, 5, 6) that have been drawn from the deck minus the number of high cards (10, J, Q, K, A) all divided by the total number of decks (52 cards) left to be drawn. The number of mid-card (7, 8, 9) that are still in the deck are not accounted for in this metric.

\subsection{Baseline and Oracle}

\textit{Baseline:}\\

The baseline for this project is to have a start state utility of -.01 for the optimum move policy and betting policy. This represents the approximately 1\% house edge built into a normal game of blackjack without card counting. This should be the value that we get if we bet one unit at all times and played with the optimum policy for all count values. \\

\\

\textit{Oracle:}\\

The oracle for this challenge lies with an individual that claims to have won over \$100,000 dollars a year playing advantage blackjack. One team out of Seattle even claim that they won over 3.5 million in just a couple of years. While these total value numbers are amazing, it doesn't give us a good estimate of the expected value of the game, since we don't know how much was invested. However, there is documentation of card counting providing a 1.2\% advantage to the player, with a standard deviation of 3.5 (very high). This 1.2\% advantage will act as our baseline for the experiment.

\section{Models}

\subsection{Simplified MDP}

\textit{Purpose}:\\

A simplified MDP model was created to produce an optimum policy and expected values for the game of Blackjack. This MDP operates under the assumption that the probability of drawing a low card, mid-card, or high-card is completely dependent on the true card count. Thus, for this MDP, the transition probability for drawing a certain card doesn’t change, regardless of what previous cards have been drawn. Despite the simplifications in this MDP, and it's differences from how Blackjack is actually played, it should provide us with useful data to start with.\\

\\

\textit{States: (PlayerState, DealerState, CardCount)}\\

PlayerState and DealerState are set up in the following format: \textbf{State = [XX]\*[keys]}. [XX] represents the maximum legal value of either the player or dealers hand at the moment. If the player holds and ace, the value of the ace is 11 unless it takes the total value over 21. Then it is made 1. [keys] represents letters added onto the string that provide information about possible next moves. If ‘D’ or ‘S’ are in the key portion of the state, then this means that a double or split move is legal on the next turn. If ‘A’ is in keys, this mean the player has an ace in their hand with value 11. When the ace has been incorporated into the value with value 1, then the ‘A’ key is deleted. \\

\\

\textit{Actions: (Stay, Draw, Double, Begin)}\\

\textit{Stay} means that the player will stop drawing cards. Now the dealer will draw from the deck until the dealer value is greater than 17. \textit{Draw} means that the player will take another card. If this card puts the player over a value of 21, the player loses. \textit{Double} means that the player doubles their bet and draws one more card. On the next more, the player must stay. \textit{Begin} is always the first move in a new hand. Here, the dealer draws one card and the player two cards. \\

\\

\textit{Transitions}:\\

The transition probabilities for the simplified MDP are directly related to the card-count and mid-card value. These transition probabilities do not change throughout the MDP. If more than one card is drawn per action, then the transition probability would involve multiplying individual card probabilities:

$$P(Card 1) \* P(Card 2) … \* P(Card N)$$

\\

\textit{Rewards}:\\

Rewards are only earned at the end of the MDP. Typically, the player either earns or loses one betting unit depending on whether they win or lose. All ties result in a push. Some edge cases includes ‘BLACKJACK’, where the player gets a score of 21 after the \textit{Begin} action the first turn and the dealer does not. In this case, the player receives 1.5 times their bet value. Additionally, if the player chooses to \textit{Double} on a hand, the reward (whether win or lose) will be doubled.\\

\\

\textit{Initialization of the Class}:\\

There are two inputs needed from the user in order to initialize the simplified MDP class. First of all, you need to input a true count value. This count value must be an integer, and is taken as a constant for this entire MDP, meaning it will not change in any of the successor states. The second value is the mid-card value. The mid-card value is defined as the number of mid-cards (7, 8, 9 cards) per 52 cards remaining in the deck. In a game where no cards are drawn yet with some number of 52 card decks, the number of mid-cards would be 12. However, once a blackjack game begins, this number is variable. The mid-card value affects the probability of drawing any card, and is a necessary to calculate card probabilities as count doesn’t provide information enough information for this.

\subsection{Full Blackjack MDP}

\textit{Purpose}:\\

Created a more representative MDP for an actual casino blackjack game in order to test the optimum policies learned in the previous value iteration MDP. In this model, exact card probabilities are calculated and probabilities of drawing cards without replacement is considered. \\

\\

\textit{State: (playerState, dealerState, CardsRemaining, CardCount)}\\

\textit{PlayerState} and \textit{dealerState} are represented the same as in the approximate MDP. The new variable \textit{CardsRemaining}, is a tuple of integers that represents the number of each card remaining in the deck. The \textit{CardCount} variable is no longer a constant, and is changed to reflect the current deck state. This value is rounded to the nearest integer value. \\

\\

\textit{Actions:} Same as simplified MDP.\\

\\

\textit{Transitions:}\\

Transition states were calculated based on the ‘CardsRemaining’. The probability of each card being drawn is:

$$ P(Card\_i) = CardsRemaining\_i/\sum\_{n=1}^{i}CardsRemaining $$

In the case where multiple cards were drawn as the result of an action, probabilities will be multiplied similar to the procedure shown for the simplified MDP.\\

\\

\textit{Reward:} Same as simplified MDP.

\section{Value Iteration}

\subsection{Algorithm}

Value iteration was used to solve this MDP and produce an expected utility and optimal policy. Two different analysis experiments were performed using the data collected from this algorithm. First, a true card count analysis was run, analyzing how different count values give different utility values and optimum policies. Second, a mid-card analysis was run to observe how sensitive utility and policy values are to a change in the number of mid cards in the deck. Both of these methods will be further described below.

\subsection{Card Count Analysis}

The MDP was run for 31 different values of true card count (-15 to 15 incremented by 1). The optimum policy and expected utility was saved for each MDP solution. The expected value for the starting state of each MDP trial was extracted and plotted. The starting state's expected value is most relevant because this is the state at which players place their bets. As the true card count increases, the expected value of a hand increases approximately linearly. An important feature is the point where the expected utility changes from negative to positive, which is the point where the player now has the edge over the casino. This is the point in time where the player should want to bet big, since they should win more money than they lose. When the count approaches the large positive or negative numbers, the count seems to approach an asymptotic maximum or minimum value.

\begin{figure}[!ht]

\centering

\includegraphics[width=9cm]{VCompCount}

\caption{E[Utility$|$Count]}

\label{fig: CountUtility}

\end{figure}

\\Literature states that the casino typically has a house edge between 0.5\% and 1\%. From our expected value based on card count, we see that a typical game (card count of 0) has an expected house edge of just over \%1. However, we do not take in some additional rare edge cases in our program for complexity reasons (such as doubling or insurance), so it would make sense that this expected value is slightly lower. \\

\\

In addition to variable expected utility for different card states, the optimum policy for playing the game also changes with card count. The plot below shows the percentage of actions in the optimum policy that are similar to the optimum policy for a count of 0. As the count increases or decreases, the optimum policy continually changes. If a player wanted to really make use of a card counting strategy in the casino, they would have to adjust their playing policy based on the count state of the deck.

\begin{figure}[h]

\centering

\includegraphics[width=9cm]{PolicyCompCount}

\caption{\% Difference in Policy Actions over Different Count}

\label{fig: CountPolicy}

\end{figure}

\subsection{Mid-Card Analysis}

While card count is a useful metric to understand relative high-card and low-card probability, it does not give any information regarding the concentration of 7, 8, 9 (mid-cards) in the deck. For the previous value iteration experiment, it was assumed in the MDP that there will be 12 mid-cards for every 52 cards in the deck. However, in an actual game, this will not always be the case. For this experiment, the number of mid-cards in the deck was changed and the value iteration was run (the count was kept at an arbitrary value of 0). \\

\\

The expected utility from the start state of a hand was found to change with respect to the number of mid-cards in the deck. However, this amount is fairly small (only about .2\% of the bet). Additionally, the expected value seems to change favorably as values of mid-card move away from the mean (the mean is almost at a local minima).

\begin{figure}[h]

\centering

\includegraphics[width=9cm]{PiCompCount}

\caption{E[Utility$|$Mid-Card, Count=0]}

\label{fig: MidCardUtility}

\end{figure}

The one possible flaw for this reasoning is that this mid-card analysis was conducted over only one count value. To be complete, the analysis could be conducted over many different count values. If expected utility was not found to vary significantly over different mid-card values for all different values of count, we could conclude that card count alone is a sufficient identifier to estimate the expected utility. However, due to lack of resources and time (and generally wanting to move into more interesting analysis) this aspect wasn’t completed.

\begin{figure}[h]

\centering

\includegraphics[width=9cm]{PiCompMid}

\caption{Policy Actions Comparison for Mid-Cards}

\label{fig: MidCardPolicy}

\end{figure}

In addition to slight expected value change, mid-card changes also induced slight variability in the policy as well. This was deemed to be insignificant, as there was no more than a 6\% difference in policy actions between mid-card states, which shouldn’t contribute to much change in our expected value.\\

\\

Overall, we can tentatively say that our simplified MDP provided a solid optimum policy for which to base our card counting strategy. While differing mid-card values may slightly affect the actions and expected value of this policy, these deviations will be small and won't end up hurting our expected value very much.

\section{Betting Policies and Simulation}

A Monte Carlo Model Fixed simulation algorithm was run using the succProbReward function of the accurate Blackjack MDP. The optimum policy created and saved from value iteration was used as the policy in the simulation. Using this method, we can test the optimum policy based on count as well as different betting policies to maximize the expected return.

\subsection{Prediction of Expected Value of Different Betting Policies}

Probability theory was used to get an estimate of expected value using the following equation:

$$E[Reward|\pi\_{bet}] = \int\pi\_{bet}\*E[Reward|Count]\*P(Count)dCount$$

\\

\textit{P(Count):}\\

A set of true card count values was found by simulating 45,000 games of blackjack. A normal distribution and a Cauchey distribution (adjusted normal distribution) were used along with a least squares method to find the best fit for the data. It was determined that the Cauchey PDF distribution was the best fit due to it's much lower $R^2$ value.

\begin{figure}[h]

\centering

\includegraphics[width=8cm]{CauchyFit}

\caption{Cauchy PDF Fit}

\label{fig: Cauchy Fit}

\end{figure}\\

Unfortunately, P(Count) $\neq$ PDF(Count) for a continuous distribution function. However, we can discretize the Cauchey distribution at each count value with each value one unit apart. This will create a discrete PDF function that will sum to one, and can be used to integrate to find the expected utility. \\

\\

\textit{E[Reward$|$Count]}: Derived earlier; see Figure 1.\\

\\

\textit{$\pi\_{bet}$}: \\

This is a policy function that plots initial bet amount against the true count of the game. Five different betting policies were created and tested.

\begin{figure}[h]

\centering

\includegraphics[width=12cm]{Poly}

\end{figure}\\

In general, policies were created to improve player expected value. Thus, for higher values of count where the player has a larger chance of winning, the bet is raised.\\

\\

When these three components were put together, they can be multiplied and integrated to marginalize out the true count and find the expected value of the deck of a given bet.

\subsection{Simulation Test}:

For each of the five betting policies, a simulation algorithm was run playing approximately 50,000 games of Blackjack. The average expected value of each hand was calculated and compared to the expected utility found via the probabilistic analysis done in Section 6.1. The summary of all values can be seen below.

\begin{figure}[h]

\centering

\includegraphics[width=15cm]{Comparison}

\end{figure}\\

\subsection{Results and Error Analysis}

The expected value residuals seen on the previous plot were slightly higher than expected for trials two and three. However, the expected values from the simulation are both higher and lower than the expected values from the simplified MDP, showing that one approach isn't skewed in one direction.\\

\\

Thus, it seems likely that the source of error between the two values is due to a lack in number of trials. This is especially true with the betting policies that bet high amounts on counts that are greater than 2. Only about 25\% of hands are played with a count greater than two. Statistically, betting a large number on a smaller sample will create higher variation in the expected value. In order to get better simulation data, more trials would have to be conducted. Unfortunately, due to lack of time and computing power, more simulation trials were not able to be run. Each 50,000 trial simulation takes around 1 hour. If I decide to keep working on this project in the future, this is definitely where I would start.\\

\\

To address an earlier point, it doesn't seem like the variation in mid-card value (accounted for in the simulation) is significantly affecting the expected utility since in general, simulation values are higher than the utility estimated from the simplified MDP (which does not account for mid-card variation). Thus, I believe we can conclude that mid-card values don't significantly affect the optimum policy derived by focusing on card count alone.

\section{Conclusion}

Both the simulation and the probability approach show that betting schemes that are skewed towards positive values are optimal for maximizing your net reward in blackjack. The larger the bets you place when you have the advantage (true card count 3 or above), the larger your expected returns will be. However, casino's have gotten wise to the idea of card counting, and are always on the lookout for players that alter their bet over a wide range based on the count. Though counting cards is in no way illegal, the casino is well within their right to ask you to leave. Due to this, I recommend that you use the [.5, 1, 2.5] betting strategy. This still will give you a positive expected utility, but will keep you generally under the radar.

\section{Moving Forward, Final Product}

\begin{figure}[h]

\centering

\includegraphics[width=12cm]{Gooey}

\caption{Application Training Tool Interface}

\label{fig: Application Interface}

\end{figure}

A Blackjack card counting program application was made to display the optimum policy learned during the value iteration process. For any player training about how to play blackjack more efficiently, this is an excellent way to do so. Inputs to the GUI are player card values, dealer card values, and the current card count at the time. The player card and dealer card fields can be left blank if the user wants to know the expected reward associated with a certain true count at the start of the round. The application works by using a database of the optimum policy and expected values generated during the value iteration process. This application is basically an easier way to parse through the database and receive information based on your current state. This is a great way to train yourself for any blackjack situation, and a great learning tool for those looking to improve their game and win some money. The variable betting strategy has not been implemented into the GUI application yet, but could be at a later date.\\

\\

\newpage

\section{Appendix}

\begin{figure}[h]

\centering

\includegraphics[width=16cm]{Deagg}

\caption{All Betting Policies and Deaggregation or Expected Value Charts}

\label{fig: Application Interface}

\end{figure}

%-------------------------------------------

\begin{thebibliography}{9}

\bibitem{Jensen} Jensen, Kamron, \emph{The Expected Value of an Advantage Blackjack player}. All Graduate Plan B and other Reports. 524

\bibitem{Wong} Wong, Stanford 1994. \emph{Professional Blackjack}, page 31, 1994 ed.

\end{thebibliography}

\end{document}

%------------------------------------------

I approached minimizing the state space in the following way. For each state, 3 separate values were stored in a tuple:

\begin{enumerate}

\item Player’s Hand (string with numbers, letters used as code for what exactly is in hand)

\item Dealer’s Hand (string with numbers, letters used as code for what exactly is in hand)

\item Cards Remaining the deck (tuple with index corresponding to number of cards for certain card)

\end{enumerate}

Both the Player’s Hand and the Dealer’s Hand have a specific code that tells the program exactly what is in each hand. The code is as follows:

\begin{itemize}

\item First 1 or 2 characters: Value (numerical representation of value that the dealer or player holds)

\item Letter “D” in characters following Value: Double Down allowed on next action

\item Letter “S” in characters following Value: Splitting allowed on next action (duplicate cards in hand)

\item Letter “A” in characters following Value: One ace in hand who’s value is taken at 11.

\end{itemize}

By creating a code to represent the different states while not having to store exact card values or the order in which cards are drawn, I hoped to minimize the state space. For example, if the player draws the 4 and a 5 in the initial draw, this would be represented by ‘9D’ state. The ‘9D’ state could also represent a draw of 3 and 6, 2 and 7, etc. This minimizes the state space since you do not have to keep track of the individual values of cards drawn.

\section\*{Actions}

For the current code, 4 actions can be taken:

\begin{enumerate}

\item Begin: Player and dealer each draw 2 cards

\item Stay: Player stays, dealer draws until value is over 17

\item Draw: Player draws one card

\item Double: Player draws one card, dealer draws until value is over 17

\end{enumerate}

To limit the tree width, some actions were eliminated for certain states that reflect simple blackjack strategy. For example, you aren’t allowed to stay if your value is less than 11, as there is not chance of losing if you take another card. Additionally, you are only allowed to double down on your bet with certain numbers, mainly you cannot double down on hand value greater than 12 unless you have an ace in your hand. Lastly, you are not allowed to draw another card if you have an 17 or higher with no ace.

\section\*{Value Iteration}

Currently, I have implemented a value iteration algorithm similar to that used in earlier class assignments. The goal of this is to find the optimum policy and expected value associated with a certain deck of cards. As currently oriented, this value iteration on the MDP has full information about the state of the problem, including the total number of the cards in the deck and the dealer and player cards. An initial deck state can be inputted into the program to see how the value changes in regards to playing with different deck states, sizes, and counts.\\

\\

The run time for a starting deck state of 13 cards is about 10 seconds for the value iteration process, which seems reasonable to me. However, when the number of cards in the deck increases, the state space of the MDP increases exponentially, causing very long run-times for the program. My goal for the next assignment is to increase the efficiency of my code with larger decks. \\

\\

Playing a game with a start state of 26 cards (each card type having an equal number in the deck) nets an expected value of 0.02 for each unit that you spend. Unfortunately, I believe that this analysis is wrong, since the odds should be against you for a normal game of blackjack. I need to go back and check my succProbReward function, to ensure that all different actions taken in this section give pay out the proper reward.

\section\*{Q-Learning}

Additionally, reinforcement learning will be implemented, since value iteration isn’t totally realistic showing of the game. In a real game of blackjack, the player will have knowledge of the count (the state), but no knowledge of the exact probabilities of the reward associated with each action. While the count gives the player a general knowledge of whether a high or low card is drawn, it does not tell the player exactly what cards are left in the deck. \\

\\

This portion of the algorithm has not been implemented yet.

\section\*{Q-Learning Features}:

For the current code, 4 actions can be taken:

\begin{enumerate}

\item Current Card Count (true count, way to represent deck state as float)

\item Dealer’s Top Card (only card visible to the player)

\item Player Card Value (total value of players cards)

\item Ace Counter (more aces in players hand provides more flexibility)

\end{enumerate}

Hopefully, these features are sufficient to explore the state space and provide a policy that is very close to optimum.

\section\*{Next Submission}

For the next submission, I plan on working on finishing my algorithm, and completing Q-learning as well.