Writing #5: Final Project Report

Anirudh Ganesh & Otto Schenk
ganes099@umn.edu&schen345@umn.edu
12/17/2023

1 Abstract

This paper delves into the game of Blackjack, and how AI agents can be used to maximize profit. Existing research in the field has covered the Mathematics and Statistics that go into Blackjack, and how intelligent agents can be built to play the game optimally. This paper expands on that research by creating three Blackjack playing intelligent agents at varying levels of complexity, and comparing them to see which strategy is the most optimal. The first agent is a Probabilistic Random Agent that chooses actions based on the probability of busting (exceeding a hand total of 21) and places equal bets each round. The second agent is a Basic Strategy Simple Reflex Agent that uses existing Basic Strategy tables to make statistically informed decisions, and places randomly sized bets. The third agent is a Monte Carlo Card Counting Agent that performs a Monte Carlo search to build up a table of optimal decisions and counts cards as they come out to place optimally sized bets. Simulations were conducted by running 500 games of Blackjack for each agent, with each game consisting of 2 decks, and each player starting with a Bankroll of $500 at the start of each game. Wins, losses, bet sizes, and overall balance were tracked across the simulations. Analysis revealed that although all 3 agents seem to win a roughly equal number of games, the Monte Carlo Card Counting agent has a higher profit margin than the other two agents. This suggests that a combination of Monte Carlo Search and Card Counting (with a much bigger emphasis on the latter) seems to offer the highest profit in Blackjack.

2 Problem Description

The problem considered in this paper is the game Blackjack. Blackjack, also known as 21, is a popular non-deterministic card game primarily played in casinos. It's a card comparison game played between one or more players and a dealer, where each player competes against the dealer (and vice versa). The game is primarily played with one or more decks of 52 standard playing cards.

The main goal of blackjack is to beat the dealer by having a hand value closer to 21 than the dealer's hand, without exceeding 21. Face cards (Kings, Queens, and Jacks) are worth 10 points, cards 2 through 10 are worth their face value, and Aces can be worth either 1 or 11 points, depending on what benefits the hand the most without busting (exceeding 21).

Game play is broken up into the following stages:

1. **Betting**: At the beginning of each round, players place their bets. Theoretically, players can bet any amount, but in practice, most casinos place limits on the amount players can bet. Bets are placed before any cards are dealt, so in theory players place bets without any knowledge of how strong their hand (or the dealer's hand) will be.

2. **Dealing**: the dealer deals two cards to each player face-up while dealing themselves one card face-up and one card face-down (known as the hole card).

3. **Player's Turn**: On their turn, each player can choose to either hit or stand. If a player hits, they receive an additional card to increase the total value of their hand. If a player stands, they keep their current hand total without taking any more cards.

4. **Dealer's Turn**: After all players have completed their turns, the dealer reveals the hidden card and must hit until their hand's total value is 17 or higher. Once they reach 17 or higher without exceeding 21, they must stand.

5. **Winning/Losing Conditions**: If the player's hand total is closer to 21 than the dealer's without exceeding 21, or if the dealer busts (exceeds 21), the player wins an equal amount as the bet they placed. Conversely, if the dealer's hand total is closer to 21 than the player's without exceeding 21, or if the player busts, the dealer wins and the player loses their bet. If both the player and the dealer have the same total, it results in a tie, and the player's bet is returned.

Readers familiar with Blackjack may notice the lack of "Split" and "Double Down" rules in the problem description. Typically in Blackjack, a player has the option to "Split" when their hand is a pair (two of the same card), with the hand being split into two, and an equal bet being placed on each hand. The hands are then played individually, meaning the player can win one, both, or neither of the hands. Players also typically have the option to double down, wherein they double the initial bet after receiving the first two cards, then receive one more card and stand. Although these rules are normally included in games of Blackjack, they will not be present in this paper for simplicity's sake.

With the rules out of the way, the central problem this paper attempts to solve can be introduced. One idea for "solving" Blackjack would be trying to maximize win percentage, i.e the number of hands where the player wins or ties with the dealer. While devising agents to optimize win percentage could be useful, an even more useful approach combines win percentage with an additional factor – bet spread. The central goal of Blackjack, from the player's perspective, is to maximize profit, so building an agent that maximizes winnings makes perfect sense. By optimizing win percentage and bet spread (i.e the amount the player bets each turn), an agent can hopefully be created to generate high profit, effectively "solving" Blackjack.

2.1 Why this Problem is Worth Exploring

This problem is worth exploring in depth for both practical and scientific reasons. First, as previously mentioned, Blackjack is an extremely popular game, and millions around the world are very invested in finding ways to "beat the system" and make profit playing it. As a result, a successful Blackjack agent could potentially provide great insight into strategies that maximize profit. This paper also attempts to pit different Blackjack strategies against each other, which again is very valuable for Blackjack players trying to find optimal strategies. Although it's unrealistic to imagine actually taking an AI agent into a casino and using it to win Blackjack hands, studying its results and using them to modify one's game could help increase profit in Blackjack, which is a highly desired outcome by many gamblers.

Along with the practical benefits, studying this problem could also yield useful insight into the strength of different AI agents. This paper pits several AI agents against each other, from a probability based agent, to a simple reflex table based agent, to a complex Monte Carlo Tree Search based agent. These 3 strategies are quite different in terms of approach and complexity, and exploring which one performs best could provide a lot of insight into the nuances of different AI agents. Furthermore, devising a successful AI agent could yield insight into the exploitable parts of the game, which could pave the way for even more AI agents in games like Blackjack. One could imagine a more complicated game like Poker being solved using insight gained from a Blackjack AI agent. From a scientific perspective, exploring the intricacies of Blackjack and pitting different AI strategies against each other could yield a lot of insight into how Artificial Intelligence can best solve card games like Blackjack.

2.2 Additional Considerations

Before exploring the existing body of work on the topic, it is important to include some additional definitions and considerations that factor into this paper's core experiment. For one, as previously mentioned, this paper operates on a simplified version of Blackjack that only allows for hitting and standing. This decision was made mostly to simplify the agents – the main goal of the paper is to see how different agents using different strategies fare in terms of maximizing profit, so operating on a simplified version of Blackjack allows for simple comparison without worrying about all the nuances. Hitting and Standing are the core features of the game, and the only rules that have a **direct** effect on win percentage (Splitting and Doubling mostly affect profit), so keeping those rules in is a necessity.

Blackjack can be played with any number of decks in theory, but in practice most casinos use more than one deck, to make card counting (a topic which will be explored more in a later section) more difficult. For this paper's simulation, 2 decks will be used. Shuffling is another important consideration to make – a computer simulation theoretically allows the dealer to shuffle after every hand, but in practice casinos never do this (and it would also render card counting completely useless), so in the simulation, a deck will only be shuffled when it's first put into play.

The final considerations to make involve bet spread. For this version of Blackjack, no explicit "table minimum" (i.e. lowest possible bet) or "table maximum" (highest possible bet) will be enforced, because effectively maximizing profit should allow players to bet as much (or as little) as they wish on each hand. Furthermore, in some versions of Blackjack when the house gets Blackjack all players automatically lose, and when the player gets Blackjack the house pays 2 to 1 (i.e the player gets double their bet in profit), but in this version of Blackjack players are able to tie the house on Blackjack, and getting Blackjack themselves only results in the normal payout.

3 Background Sources

There is a pretty extensive body of work in existence that covers strategies used to solve Blackjack. Researchers have been studying the problem for decades, exploring how different statistical strategies and algorithms can give the player an edge. These sources range from simple to complex, with some researchers even exploring how advanced Deep Learning (a subset of Machine Learning) can be used to maximize profit. This paper doesn't delve into a territory as advanced as Machine Learning but does explore how probabilistic, simple-reflex and advanced Monte Carlo agents approach the problem of Blackjack. On that note, many sources provide extremely useful insight into the game of Blackjack, and how different algorithms can be used to solve it.

This paper will explore 12 different background sources, split up into several subsections. Some sources explore the mathematics behind Blackjack, some explore how AI can be applied to similar games like Poker, and some directly cover the methods this paper delves into.

3.1 Blackjack Sources (Mathematical/Statistical

The first source gives a general overview of Blackjack and introduces the mathematical component that makes it possible for an agent to make intelligent decisions. "Blackjack: The Math Behind the Cards"[1] first gives an overview of Blackjack, covering the basic rules and concepts. It then goes into Combinatorial analysis, where it discusses the probabilities of receiving different card pairs. The simplest analysis is performed on single-deck games, but the source also covers scenarios that include multiple decks. In essence, adding more decks introduces more variability in the card that comes up next, making it harder to predict whether hitting or staying is optimal. On that note, with regards to hitting and staying, best and worst-case scenarios are covered by the source. Visual aids, including tables, formulas, and charts, are used to illustrate the mathematics that go into optimal move selection in Blackjack. As mentioned previously, statistical analysis is key to building an AI Blackjack agent, since the only way to make an informed decision is to estimate the likelihood of a given card appearing when hitting/standing. As a result, this paper is key in gaining insight into the probabilistic foundation of Blackjack, and will certainly help with constructing an intelligent agent.

The second source discusses card counting, an integral concept in Optimal Blackjack strategy. The paper "Real Valued card counting Strategies for the Game of Blackjack"[11] discusses the concept of card counting, and how it can help players gain advantages. After going over the basics of Blackjack, the paper introduces card counting principles, which can be boiled down to assigning a positive, negative, or zero point value to each card value available. Low cards get positive point values and raise the value of the count, indicating a higher percentage of high cards remaining in the deck. Conversely, high cards lower the count, and make it less advantageous to bet high. The paper then discusses efficiency metrics used to evaluate card counting strategies, including Betting Correlation (how well the system detects player advantage) and Insurance Correlation (how well the system indicates whether an insurance bet should be taken or not). These metrics are very useful because they can be used to measure the success of the card-counting Monte Carlo agent. The researchers used a genetic algorithm to build a card-counting model based on multiple approaches (Hi-Lo, real-valued, etc.), and although Monte Carlo search will be used in this paper instead, it is still useful to examine how card-counting algorithms can be constructed. The results showed that using real-valued weights in card-counting strategies offers the highest advantage, so real-valued weights will most likely be used in this paper's card-counting model as well. Overall, this paper is very useful in introducing the concept of card counting and relating it to Artificial Intelligence agents.

The third source explains the concept of Basic Strategy in Blackjack[8]. Written in 2017, the source first introduces Blackjack and then discusses basic strategy using a table of hit, stand, double down, split, and surrender (in this paper's simulation only hitting and standing will be considered for simplicity's sake however). The paper explains the probability that goes into constructing the basic strategy, covering the mathematical formulas that include summations, conditional probability, and combinatorics. Although it isn't directly tied to AI, this source is incredibly useful in establishing the concept of a basic strategy that can be used in multiple agents. One of the planned agents for this paper is a simple reflex agent that uses a basic strategy table to make informed decisions, serving as the "medium complex" agent. In order to do this, the information must be available, which is where this source comes in.

3.2 Blackjack Sources (AI)

The first source comes from Jeffrey Popyack of Drexel University, and it covers a series of Blackjack-related assignments in an advanced AI course[7]. Although this source doesn't dive deep into the actual algorithms or code used for a Blackjack agent, it gives a good starting point of how to create an agent for a game like Blackjack. The first section discusses the mechanics of Blackjack and introduces some preliminary functions such as “prob(start, j)", which defines the probability that the system enters state j after the initial two cards are dealt to the player. After background is provided, the paper outlines some of the assignments used in the class. These assignments include stochastic analysis through Markov Modeling, Dealer strategy analysis using Transition matrices, and Player Policy creation through Markov decision processes. This source provides a great jumping-off point for further research because it introduces several techniques in an educational lens. Further research was done into techniques like Markov Modeling, but this source introduces those methods in the first place. It may seem odd to look at a source that simply outlines useful assignments for an Artificial Intelligence class, but it provides a great introduction to intelligently playing Blackjack and offers a useful jumping-off point.

The second source, "Intuitive Searching: An Approach to Search the Decision Policy of a Blackjack Agent"[6] discusses a novel Blackjack Agent that uses a concept known as "intuitive searching". The first topic covered is the aforementioned "Markov Model". The Markov model is a Nash Equilibrium strategy that makes low-risk decisions based on the potential of the current hand probabilities of training data. In contrast, the intuitive searching algorithm is an Exploitative strategy that tries to mimic a worst-case opponent for the given

Agent. After constructing the intuitive searching agent, the researchers ran it against the Markov Model and found that switching from the Markov Model to the intuitive agent increased the win rate for players and dealers. This provides compelling support for the intuitive agent and thus makes it an avenue worth looking into. The article did mention some issues that could go along with the intuitive search, including scalability, and increased complexity that comes with multiple decks and more complicated Blackjack rules like splitting and doubling. Even still, this source provides an interesting and useful technique for designing Blackjack agents.

The third source is an article that discusses simulating Blackjack with Python[13]. The article assumes the reader has a grasp of blackjack concepts, so it keeps the game description short and mostly covers the underlying code as well as the logistics of the simulation. Key components of the code include creating one or more decks of cards, calculating the value of hands, and simulating the dealer's and player's actions. Some important libraries used in this simulation code include pandas and numpy, which include functions that are integral for creating decks and calculating hand values. Whether the player hits or not is decided mostly on randomness in the provided code, but this can easily be changed to use an intelligent agent instead. The author also ran a simulation where the player won about 32 percent of the time, and the hope is to construct a more complex Monte Carlo agent that can beat that. This article is very useful for the simulation component of the final project – the intention is to compare the agents by running many simulations of Blackjack, and this source can be used to aid in writing the simulation. It will require some tweaking, but a great base is provided by the article.

The fourth source, "A Comparison between Cognitive and AI Models of Blackjack Strategy Learning"[9] attempts to distinguish popular player strategies from AI strategies, and conclude whether Blackjack can be effectively learned or not. After introducing the game, the paper discusses several popular Blackjack strategies, including Never Bust, Mimic the Dealer, and Basic Strategy. It then introduces CHREST, a cognitive architecture that enables the modeling of human processes of perception, learning, and memory. CHREST runs as a Java computer program, and it enables simulations/testing of cognitive models, such as the ones discussed earlier. Running tests using CHREST allowed researchers to discern how well a model can "learn" to play Blackjack simply by playing over and over again. One major conclusion was that learning blackjack by playing was very difficult – the AI models ended up far from optimal performance even after 10,000 hands of experience. Some of this can be chalked up to the stochastic nature of Blackjack (as noted by the researchers), but it still indicates the importance of teaching a model basic/complex strategy rather than just sending it out in the field to learn. After reviewing this source, the decision was made to not attempt a reinforcement learning style for this paper, since the researchers found even after tens of thousands of hands, models were still far from optimal. Instead, a basic strategy can be devised and integrated with card counting and Monte Carlo to hopefully get close to an optimal solution.

3.3 Monte Carlo Sources

The first source introduces the idea of Monte Carlo tree search as a framework to game AI[2]. The paper essentially formalizes the Monte Carlo method taught in class, discussing the concepts of Selection, Expansion, Simulation, and Backpropagation. At the time this paper came out, Monte Carlo Tree Search had only been established for two years and was not commonly used as an approach to creating agents for games. This paper discusses potential applications to Classic board games, including Chess and Checkers, Modern board games like Settlers of Catan, and even Video Games, like the Real-Time-Strategy game Spring. Even though this paper is quite old, it's still incredibly useful because it provides the Monte Carlo basis that can be further expanded on with other sources. Having a formal description of Monte Carlo Tree Search is a prerequisite for constructing a Monte Carlo agent and examining potential improvements/varied strategies, so this source provides a necessary foundation.

The second source builds off of the first, reviewing recent modifications and applications of Monte Carlo tree search in 2022[10]. After introducing Monte Carlo Tree Search formally, the paper discusses why the base variant isn't useful for complicated problems that involve combinatorial complexity, sparse rewards, or other kinds of inherent difficulty. The paper then sets about reviewing multiple modified versions of MCTS, including MCTS with Perfect Information, MCTS with Imperfect Information, MCTS with Machine Learning, MCTS with Evolutionary Methods, MCTS Applications Beyond Games, and Parallelization. The Basic Theory section doesn't provide a lot of different information from the previous MCTS source, but it does formalize a lot of the concepts and introduces the idea of a selection policy to guide agents. It discusses ways to improve the search using Transposition tables, History heuristics, and more, all of which can be used when writing MCTS for this project. The researchers concluded that the best-hybridized approaches included MCTS + Machine Learning, and MCTS + Domain Knowledge (i.e. exclusion of certain paths in the search space that are not viable due to possessed knowledge). While these conclusions are interesting, the most useful part of the source in the context of this paper is the modifications it makes to Vanilla MCTS. A hybridized reinforcement learning approach is unrealistic for the scope of this paper, but the Monte Carlo agent can definitely be made more efficient through methods like Transposition tables. Improving search efficiency will lead to more success for the complex agent, as well as faster simulation time (allowing for more simulations and thus more accurate results). As a result, this paper is very important for examining ways to build off of basic MCTS.

The third source is perhaps the most directly related to the approach used for the third agent in this paper, as it directly covers application of Monte Carlo Tree Search to Blackjack. The paper comes from 3 students at Stanford University, who explored whether or not MCTS performs better than the best practices for Blackjack[3]. The researchers defined a "human" agent as one who always hits until their hand total exceeds 17, then wrote two Monte Carlo Tree Search, one that doesn't keep track of history (i.e which cards have been played) and one that does. By comparing these 2 agents to the human agent, the researchers found that the Monte Carlo Tree Search outperforms the naive human agent, and lies closely in line with expert human Blackjack strategies. In their experiment, the existence of a card history didn't greatly affect the performance of their agent. Critically, however, the researchers operated on a game **without** betting, and thus only considered win percentage. This paper can expand on the work done by these researchers by integrating betting into the AI agents and seeking to maximize profit instead of just win percentage. Although card counting wasn't useful for the agent featured in this source, card counting can be used to optimize bet spread (i.e betting low when the count is low, betting high when the count is high), which should provide a distinct advantage over agents that don't factor in card history. This source is very useful for proving the value of Monte Carlo Tree Search in the context of Blackjack and offers plenty of room for extension via the integration of betting and varied bet spreads.

3.4 Approaches Used to Solve Similar Games

The first source discusses a game very similar to Blackjack, Poker[5]. This paper goes over probabilistic betting decisions and the use of simulation when playing poker. Although the source isn't directly about Blackjack, it is still interesting and useful to examine the uses of artificial intelligence in games with a hidden component. The hidden component of Blackjack is the dealer's face-down card, and the hidden component of poker is the other players' hands. The author mentions using "probability triples" or sets of three numbers, each representing the probabilities of taking one of three possible actions in poker (folding, calling, or raising). One can easily imagine integrating this idea of a probability triple to blackjack, where it would instead be a probability double of hit or stand. This approach accounts for the uncertainties of different game situations, allowing the AI to make more flexible and informed decisions. The researchers also introduce real-time simulations, used to compute expected win/loss from betting certain amounts. Again, one can imagine introducing real-time simulations to Blackjack in a similar way, examining how a bet in a given hand leads to some expected win or loss. By running many simulations of the probabilistic poker agent, the researchers were able to conclude that probability triples help unify several knowledge-based components and that a simulation-based betting strategy is superior to a static-evaluation-based alternative. Despite not being directly tied to Blackjack, this paper is very useful in providing 2 AI techniques that can be implemented into one or more of the agents.

The second source goes into the challenges of handling complex problems using game theory with imperfect information[4]. Blackjack isn't mentioned specifically, but it is notably a game played with imperfect information (players don't know the next card up or the dealer's face-down card), so examining how agents handle these games is still useful. The paper discusses how imperfect information doesn't greatly increase the cost of solving a game, but does greatly increase the complexity of the problem, which can make it harder to search and find an optimal move. To combat this, the researchers created a system called Gala, which can efficiently describe large complex games and find optimal strategies for games in extensive form. For this, researchers created their own custom programming language and found that algorithms using Gala ran exponentially faster than standard algorithms. Although the Gala system will not be present in this paper, the paper still provides many useful techniques that can be examined in further detail. For example, the paper discusses heuristics based on Nash Equilibrium that could certainly be integrated into a Monte Carlo Tree Search based agent to increase efficiency. It is also useful to understand the challenges of games with imperfect information because it encourages looking for ways to improve efficiency instead of just settling for a first-pass solution. Finally, the ways in which researchers ran simulations to test Gala could prove useful, as a large component of this paper will be running simulations to compare different Blackjack solving agents.

4 Problem Solving Approach

As previously mentioned, the core problem in Blackjack revolves around maximizing profit. This involves both maximizing win percentage and attempting to bet more on hands that win and less on hands that lose. This idea may seem unintuitive, as by design a player shouldn't be able to know when a hand will win or lose, but by card counting (as will be covered later in this section), an agent actually **can** probabilistically find hands that are more likely to win. The core goal of this paper is not only to "solve" Blackjack by achieving high profits in simulation but also to compare 3 AI agents at varying levels of complexity, to see which agent performs the best. With that in mind, the problem-solving approach involved creating 3 AI agents to play the game of Blackjack.

4.1 Agent 1: Probabilistic Random Agent

The first and simplest AI agent is based on basic card probability. In essence, once the agent is dealt its 2 cards and the player's turn begins, the agent acts based on 2 edge cases and a random number generator. If the score is greater than or equal to 10, the agent will always hit, since it is highly unlikely to win a hand standing at 10. Similarly, if the score is greater than or equal to 19, the agent will always stand, due to the extremely high likelihood of busting.

Once the edge cases are cleared, the core probabilistic element of the agent is introduced. Based on the agent's hand, a function calculates the number of cards in a 52-card deck that would result in a bust (score > 21). The probability of busting is the number of cards that result in a bust divided by the total number of cards in a deck. The formula can be expressed mathematically as $\frac{numNonBust}{52}$. Even for a multi-deck game, the probability will be the same, since the numerator and denominator will be scaled equally by the number of decks. Once the probability of busting is found, the agent generates a random number between 0 and 1 and hits when the number is greater than the probability of busting. For example, if an agent finds that 30% of cards remaining in the deck result in a bust (probability of 0.3), the agent hits when the random number it generates is greater than 0.3, around 70% of the time in the long run.

The logic behind the agent is quite simple – if drawing a new card is likely to result in a bust, the agent will prefer to stand, and vice versa. There are a few nuances that are important to note. When calculating the number of cards that result in a bust, the agent does not keep track of a history of cards it's already seen in play – it simply goes through all 52 possible cards, and tracks the amount of cards that result in a bust. Furthermore, the agent does **not** factor the dealer's hand into its logic at all. It's almost playing a "solo" game of Blackjack, where the only goal is to avoid busting while still achieving a high hand. As for bet-spread, the agent flat bets the same amount each round.

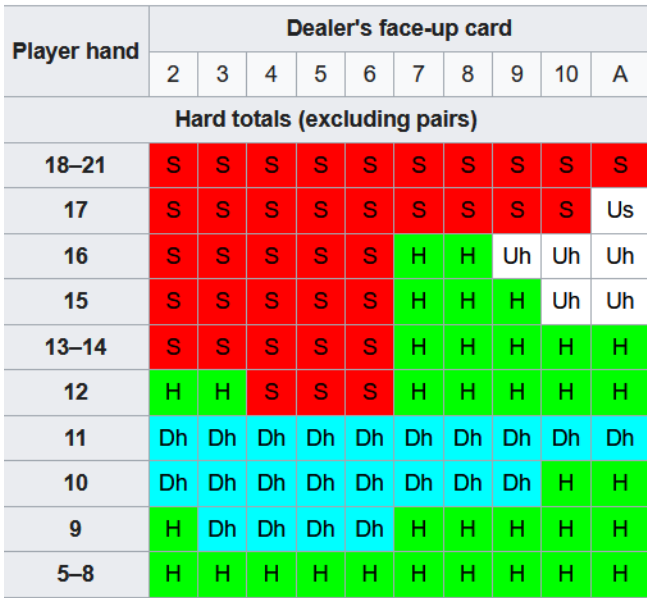
The lack of complexity present in the simple agent should result in poorer performance when compared to the other agents. The agent doesn't keep track of cards it has seen, doesn't factor in the dealer's position when making decisions, and can still make decisions that are not sensible (for example, the agent can hit on an 18 if the randomly generated number falls into the right range). As a result of flat-betting, the agent is also not expected to win much in the simulation – even with a consistent win percentage, the agent will only rack up relatively small amounts each round.

These decisions were made on purpose to have the simple agent mimic a casual Blackjack player – one unfamiliar with basic strategy, who bets the same amount each time and only hits when they think they're unlikely to bust. The agent should serve as somewhat of a control, as the other two agents (especially the Monte Carlo Tree Search agent) are expected to fare better.

4.2 Agent 2: Basic Strategy Simple Reflex Agent

A simple reflex agent is a basic AI agent that makes decisions on the basis of its current percepts (the input that an intelligent agent is perceiving at a given moment), ignoring the rest of its percept history. In the context of Blackjack, the agent's current percept is its own hand and the dealer's up card. Based on the percept, the agent makes decisions based on "basic strategy"

In essence, Blackjack basic strategy involves making decisions based on the value of your hand and the dealer's upcard to maximize your chances of winning. It's about making statistically optimal choices in each situation to increase the chances of winning. For instance, when a player's hand is low, and the dealer's upcard is high, it's statistically favorable to hit. Basic strategy charts exist to guide players on what moves to make based on every specific combination of cards they have versus the dealer's visible card. A basic strategy chart can be seen below:



Player hand	Dealer's face-up card									
	2	3	4	5	6	7	8	9	10	A
Hard totals (excluding pairs)										
18–21	S	S	S	S	S	S	S	S	S	S
17	S	S	S	S	S	S	S	S	S	Us
16	S	S	S	S	S	H	H	Uh	Uh	Uh
15	S	S	S	S	S	H	H	H	Uh	Uh
13–14	S	S	S	S	S	H	H	H	H	H
12	H	H	S	S	S	H	H	H	H	H
11	Dh	Dh	Dh	Dh	Dh	Dh	Dh	Dh	Dh	Dh
10	Dh	Dh	Dh	Dh	Dh	Dh	Dh	Dh	H	H
9	H	Dh	Dh	Dh	Dh	H	H	H	H	H
5–8	H	H	H	H	H	H	H	H	H	H

Figure 1. Basic Strategy Blackjack Table[12]

The table covers all possible combinations of player hand and dealer upcard and dictates whether the statistically optimal move is to hit or stand. In the chart, "Uh" means hit, and "Dh" means stand (they actually refer to "surrender", but that is not a rule in the modified version of Blackjack used in this project, so they are substituted with hit and stand). This table was programmed into the agent's memory (represented as a dictionary in the Python program), and on the agent's turn, it makes a decision by simply consulting the table, inputting its own hand and the dealer's upcard.

This agent is a bit more nuanced than the previous one, considering both the player and the dealer's hand. It also uses statistical reasoning to make decisions, consulting a chart that even the expert Blackjack players use. This being said, it doesn't keep track of a percept history or count cards, so it doesn't actually know which cards are more likely to show up when hitting. Additionally, the agent makes bets randomly from \$5 to \$30. Randomizing bet spread could end up backfiring if the agent loses high bet hands, but it should also allow the agent to capitalize on winning hands more than the previous agent.

4.3 Agent 3: Monte Carlo Card Counting Agent

The third agent harnesses the Monte Carlo Tree Search strategy and the concept of Card Counting to maximize profit. The fundamental concept of Monte Carlo is to simulate games from a given point and use the outcomes of those simulations to inform decisions. In this case, a simulation consisting of millions of hands is performed **before** the cards are even dealt, with the goal being to simulate many possible hands and see which decision (hit or stand) results in success most of the time for a given game state. The results of the simulation are distilled into a "decision matrix", which the agent can use directly to pick optimal decisions.

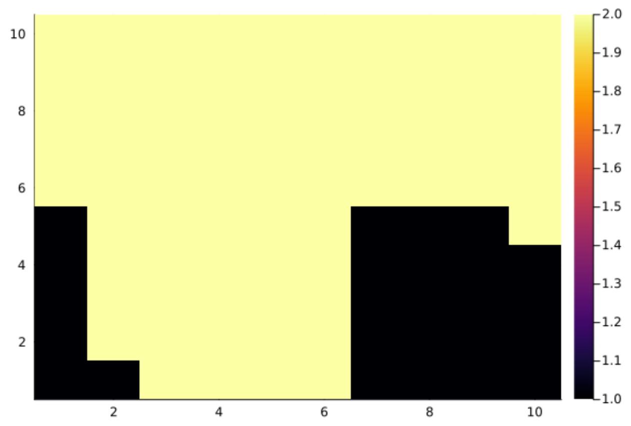
To construct the decision matrix that the agent uses, 10,000,000 hands of Blackjack are simulated, and the results of each runout are recorded and fed into the decision matrix. This approach ensures that each player's hand is tested against an incredibly large spectrum of potential dealer hands, from a low Ace through Ten. This in turn results in a strong decision matrix that should yield a high win percentage

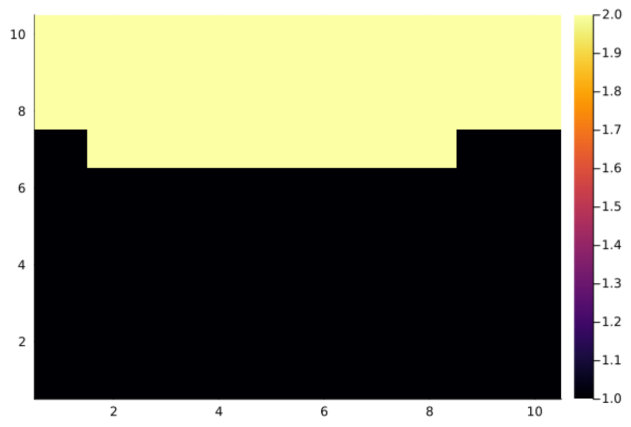
As the simulations unfold, the outcome of each iteration (win, loss, or draw) is recorded and associated with the given hand. Over time, patterns emerge, revealing which decisions lead to wins more often than losses for a given hand. This means as the simulations progress, the decision matrix gets filled, and the player begins to make smarter decisions. When the simulations begin, the player is just randomly choosing to hit or stand on its turn, but as the outcomes of iterations backpropagate, the player starts using the decision matrix to pick moves it thinks are optimal. Random decisions are still used occasionally to explore alternative paths and ensure the entire decision matrix is full by the time the simulation is over.

To perform the simulations described above, code was written in Python. First, a blackjack function was written to simulate a single round of Blackjack, simulating the cards being dealt, the player taking their turn, and the dealer taking their turn. The arguments for this function are q, qn, and π. q represents the Quality Matrix, which expresses the expected utility of taking a given action (i.e. hit, stand). qn represents the number of times the given game state (i.e. combination of player's hand and dealer's hand) has appeared in the simulations so far. π represents the Policy Matrix, which expresses the optimal decision for the given hand at that time.

As previously mentioned, the code simulates the runout of over 10,000,000 hands of Blackjack. "Status Updates" are triggered every millionth iteration to update the Quality and Policy Matrix. For a given iteration, each action is associated with a "reward", indicating whether the decision led to a win, loss, or draw. The average reward for a certain action in a certain game state directly influences the Quality and Policy matrices – if an action has a high average reward, it will have a higher expected utility in the Quality Matrix and will become the standard for the Policy Matrix. For example, if the game state where the player has two tens and the dealer has a six upcard has been run 100 times by the time a status update occurs, the average reward for "stand" will likely be very high, whereas the reward for "hit" will likely be very low (hitting is almost guaranteed to bust, standing at 20 is almost guaranteed to win). As a result, in the Policy Matrix, the state "two tens for the player and a six upcard for the dealer" will be associated with "stand", and the Quality Matrix will expect a high reward for standing in this position.

As the simulations unfold, the Quality Matrix converges toward the true expected rewards for each state-action pair under the optimal policy. Concurrently, the Policy Matrix evolves, with each state-action pair being nudged toward the action that maximizes the expected reward. In the end, this results in a Decision Matrix based on the Quality Matrix and Policy Matrix that picks the most optimal move for a given state-action pair. The decision matrices after the simulations can be seen below:





The top decision matrix represents game states with a usable ace (i.e. an ace that can be used as 1 or 11), and the bottom decision matrix represents game states with an unusable ace (i.e. an ace that must be used as 1 to avoid busting). The x-axis is the dealer's upcard, and the y-axis is the score of the player's combined hand. The y-axis starts at 0, but in this case, 0 represents the combined score of 12, since every game state with a combined hand score under 12 should automatically be hit. A value of white indicates the player should stand, while a value of black indicates the player should hit. These decision matrices are used by the agent on its turn to make decisions – similar to how the second agent uses a Basic Strategy table, this agent uses the results of the Monte Carlo Simulation. These decisions should be optimal, resulting in a high win percentage.

To optimize bet spread, this agent uses a strategy known as "Card Counting". Card counting in blackjack is a strategy used to track the relative proportions of high cards to low cards remaining in the deck. In essence, it involves keeping a mental tally of the cards that have been dealt to predict whether an upcoming card will be high or low. Players who card count typically have to perform quick mental maths, but for a computer, card counting is a much simpler task. The agent keeps a running count starting at 0, adding 1 to the count when a card with a value 2-6 is drawn, keeping the count the same when a card with a value 7-9 is drawn, and subtracting 1 when a card with a value 10-Ace is drawn.

After enough time, the count can be used to inform bet sizing. In essence, a positive count suggests more high-value cards are in the deck (since many low-value cards have already been spotted), which favors the player. Comparatively, a negative count suggests more low-value cards in the deck, which means the dealer is favored. Players should optimally adjust bet sizing by betting large when the count is high (i.e. the player edge is high), and betting small when the count is low. The third agent does exactly this, modifying its bet size for a given hand based on the running count.

To summarize, the third and most complex agent uses Monte Carlo Search to construct a decision matrix that informs its actions and uses card counting to place optimal bets. This should result in the highest profit out of all three agents, as the third agent should earn more when it wins, and lose less when it loses.

5 Design of the Experiments and Results

Once the agents were created and tested, an experiment was designed to effectively compare the three agents, and determine which is the most successful in generating profit. A few factors were key in maintaining a successful experiment. For one, the game conditions would have to be the same for all three agents, so in all trials, the agents had an equal bankroll to begin, the game was played with 2 decks, and a shuffle once when the deck was created policy was enforced. In each run, the agent starts with a bankroll of $500 and plays round after round of Blackjack until the 2 decks run out of cards or the player loses all their money. Additionally, each trial is run as a Player vs. Dealer game, rather than a Agent 1 vs. Agent 2 vs. Agent 3 vs. Dealer game. The latter would give a far less accurate result since there is a good chance one player could simply get lucky with the cards they were dealt on a given trial. With a Player vs. Dealer setup, however, the decks are only split between 2 players, resulting in less statistical deviation. Another important consideration is sample size – to reduce the effects of variance, each agent got 500 runs of a 2-deck game.

There are two main factors that can be used to measure an agent's success:

1. **Win Percentage**: Win percentage is defined simply as $\frac{numWins}{totalRounds}$. It gives a pretty good indication of how an agent performed by showing how their decision-making worked out. An agent with poor decision-making is more likely to lose a round, and conversely, a more intelligent

agent is more likely to win a round. Variance is definitely in play in a non-deterministic game like Blackjack, but over thousands of iterations, a good result can be derived.

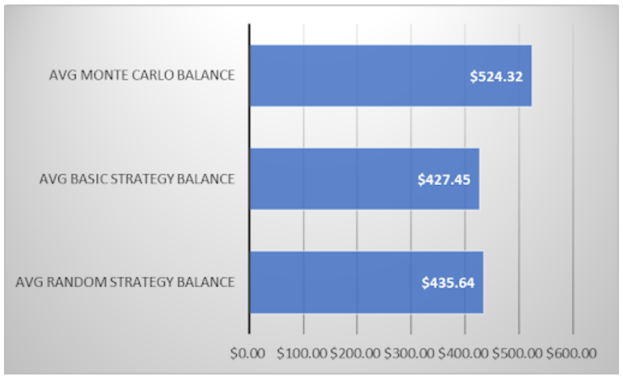
2. **Profit**: Profit is defined as the agent's endBalance - startBalance. All agents begin with the same bankroll, and as mentioned in the beginning the core goal of Blackjack is to win money, so profit is one of the biggest factors to consider. Rather than using profit from a single iteration, which is highly subject to luck, the average profit (or loss) of all 500 iterations can be considered.

Profit is certainly the more important metric, but the simulation measures both factors by keeping track of every single individual round that each agent plays. The wins, losses, and bets are all tracked, but the individual cards the player/dealer gets aren't – they don't provide much useful data since a given card is theoretically completely random. Once the simulations were run in Python, the data was collected in CSV format, and processed to create graphs.

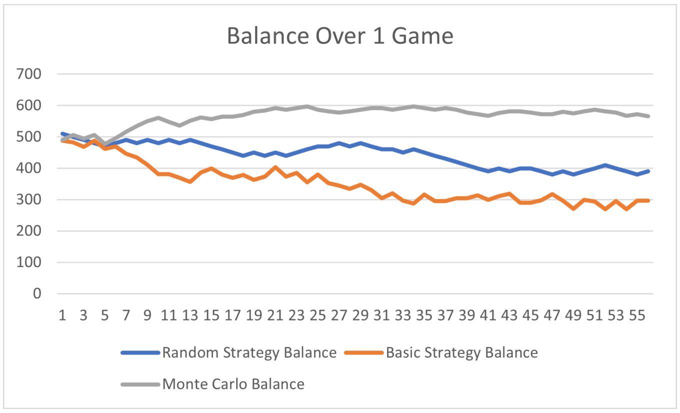
Integrating the agents into the simulation involved replacing the part of the Blackjack code that asked for player input on the player's turn with a call to the respective ACTION(s) function of each agent. The third agent's card counting strategy did require interfacing with the agent during the dealer's turn, so the agent could update its running count after viewing the dealer's cards. Additionally, the Decision Matrix created for the third agent was stored in a transposition table, so Monte Carlo didn't need to be run every iteration.

5.1 Graphs

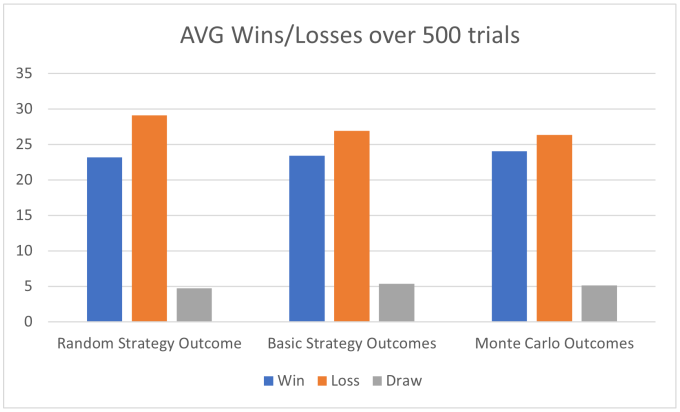
Based on the CSV files generated from the Python code, several graphs were created:



This first graph summarizes results from all 500 trials for the three agents. Average balance refers to the average ending balance each agent accumulated when the game was over (the beginning balance was $500 for all agents).



The second graph examines a single trial – that is, one 2-deck game of Blackjack. The metric tracked is Balance: the graph shows how the given agent's balance varied from the beginning of the game to the end. Although one trial isn't a large enough sample size to draw sweeping conclusions, looking at one game closely does provide insight into how an agent fares.



The final graph also summarizes results over the 500 trials, examining the win-to-loss-to-draw ratio. Along with profit, this is an important metric for analyzing how each agent fared, and the 500 trial sample size ensures the data is accurate.

6 Analysis of Results

With the graphs generated, an analysis of said graphs can be conducted to examine how each agent performed. The analysis section is broken up into several subsections – an analysis of the design of the experiment, an analysis of each individual graph, and an overall analysis.

6.1 Analysis of Experimental Design

Before analyzing the graphs themselves, it's important to analyze the experimental design and argue its merit. For one, the conditions present in each trial were controlled enough to receive an accurate result; the inclusion of multiple decks prevents card counting from being an impossible strategy to beat (if there were only one deck, the third agent could very easily predict what card was coming next), the starting bankroll is high enough to make losing all money unlikely (thus allowing all agents to get roughly equal playing time), and the Player vs. Dealer setup ensures the card distributions are roughly equal for each agent.

Additionally, the sample size is high enough to extract usable data. Assume on average each trial yields 18 rounds (this is assuming both the agent and dealer draw 3 cards total each round), then in total, each agent gets 9,000 hands of Blackjack. 9,000 hands of Blackjack is certainly high enough to demonstrate if a strategy is effective or not, and overall the experiment ran was controlled enough and extensive enough to yield data ripe for analysis.

6.2 Graph 1 Analysis

Graph 1 examines the average ending balance for the three agents over all 500 trials. The simplest agent averaged \$435.64, the basic strategy agent averaged \$427.45 and the complex agent averaged \$524.32. When compared head to head, each agent fared as such:

1. **Agent 1 vs. Agent 2 (Random vs. Basic Strategy)**: Agent 1 earned \$8.19 more than Agent 2 on average. In terms of percentage, Agent 1 earned 1.89% more than Agent 2. Both lost money overall

2. **Agent 1 vs. Agent 3 (Random vs. Monte Carlo)**: Agent 3 earned \$88.68 more than Agent 2 on average. In terms of percentage, Agent 1 earned 18.48% more than Agent 2. Agent 3 gained a profit overall, whereas Agent 2 lost money on average.

3. **Agent 2 vs. Agent 3 (Random vs. Basic Strategy)**: Agent 3 earned \$96.87 more than Agent 2 on average. In terms of percentage, Agent 1 earned 20.35% more than Agent 2. Agent 3 gained a profit overall, whereas Agent 2 lost money on average.

The first thing to note is that Agent 3 is the only agent that made a profit on average over the 500 trials. This is somewhat to be expected, as in reality the dealer almost always wins in the long run due to house edge. The only way to beat house edge is by counting cards, which is exactly what Agent 3 does. As a result, it makes sense that Agent 3 was able to use its slight edge over the house (around 1% mathematically) to make a profit, whereas Agents 1 and 2 were not. The relatively low profit from Agent 3 is most likely due to the Decision Matrix – it's possible there weren't enough Monte Carlo trials simulated to get the most optimal Decision Matrix possible, resulting in poorer decision-making that could have lost the agent some rounds.

One surprising outcome was the Random Strategy agent performed better than the Basic Strategy agent. Intuitively, this doesn't make much sense, as the Basic Strategy agent follows a table that is backed by comprehensive statistical analysis. However, the reason the Basic Strategy agent performed slightly worse is probably because of the randomized bet spread. The simplest agent bets the same amount each time, which means its final profit is essentially directly tied to its win-loss ratio. In contrast, the basic strategy agent bets randomly without any data to inform its betting decision, meaning it's certainly possible for the agent to end up betting small on winning hands, and betting large on losing hands. This was expected to balance out over the trials, but it appears to not have, resulting in a lower profit margin.

Overall, the results from Graph 1 indicate that the Monte Carlo agent was far in a way the most successful agent, earning a profit over (albeit not a large one) over the 500 trials.

6.3 Graph 2 Analysis

The second graph analyzes a single trial and tracks the balance for each agent. Analyzing the outcome isn't very useful (since one trial is far too low of a sample size), but analyzing the trends does provide some useful insight. For example, it seems that the agents started to separate from each other at around the 6-hand mark, upon which Agent 3 began to earn a profit, Agent 1 stayed relatively around its initial balance, and Agent 2 started to lose money. Many factors could have caused this, but it is likely that Agent 3 started to see a high count at around the 6-hand mark, encouraging higher bets that increased its profit. It's also likely that Agent 2 started placing high bets (due to its random betting strategy) on losing hands at that 6-hand mark since the biggest losses appear to occur around there. Agent 1 sees a bit of an earnings bump, but its lack of real strategy means most of the peaks and valleys are simply due to luck, and the relatively steady balance is likely due to the flat betting strategy.

Overall, this graph reveals interesting insights about Agents 2 and 3. The trial in question reveals a potential flaw in Agent 2's design – the randomized betting pattern can lead to bad results if high bets are placed on losing hands, and even with a statistically backed basic strategy, these losses can make it hard to break even. As for Agent 3, the general upward trend was a commonality in most trials and is again likely a result of the card-counting betting strategy. If the agent was flat betting or opted for a randomized bet strategy, it likely would have earned a far smaller profit, regardless of how many games it won.

6.4 Graph 3 Analysis

The third graph again analyzes all 500 trials, this time looking at the number of wins, losses, and draws each agent had on average. Agent 1 had 22 wins, 29 losses, and 5 draws per trial. Agent 2 had 23 wins, 27 losses, and 6 draws per trial. Agent 3 had 24 wins, 27 losses, and 5 draws per trial. These numbers are rounded so they are not perfectly accurate since the averages in the graphs all contain decimal values. The win percentages for Agents 1, 2, and 3, are as follows: 39.2%, 41.07%, 42.9%. When compared head to head, each agent fared as such:

1. **Agent 1 vs. Agent 2 (Random vs. Basic Strategy)**: Agent 2 won 1 more game than Agent 1 on average. Both lost more games than they won.

2. **Agent 1 vs. Agent 3 (Random vs. Monte Carlo)**: Agent 3 won 2 more games than Agent 1 on average. Both lost more games than they won.

3. **Agent 2 vs. Agent 3 (Random vs. Basic Strategy)**: Agent 2 won 1 more games than Agent 1 on average. Both lost more games than they won.

The most pressing thing to address is that the agents seemed to fare about as well in terms of wins, losses, and draws. Given the high sample size, this seems to indicate that the Random Agent that was expected to perform quite poorly actually seemed to be smarter than expected. This is most likely due to the probabilistic nature of the agent's decision-making process – the basis for its decisions is based on probability, specifically the probability that the upcoming card will result in a bust. Basic strategy (and Monte Carlo to some extent) are also based on probability, and although their methods are far more advanced, it seems as though the outcomes were similar for this experiment.

One important thing to note is that Basic Strategy works better for games with more decks, so it's possible Agent 2 would have fared better in a game with more than 2 decks. Card counting is much less effective in games with large decks, so conversely Agent 3 probably would have performed worse profit-wise. The results seem to indicate that the increased complexity of Basic Strategy and Monte Carlo search don't offer much of a benefit in the way of win percentage.

The second thing to note is that **none** of the agents had a positive win percentage. This does make sense when the house edge is considered, since even with advanced methods like Basic Strategy and Monte Carlo tree search, the house still retains an edge, meaning the win percentage should generally be below 50%. Agent 3's card counting does little in the way of increasing win percentage, but that's because it isn't really supposed to. As previously mentioned, the core principle of card counting is to bet more when the count is high (i.e. the player has an advantage) and bet less when the count is low (i.e. the dealer has an advantage). Even with counting cards, the agent doesn't know which card is up next, so the win percentage isn't changed. Rather, the varied bet spread allows for profitability in the long run, which is what was observed from Agent 3's performance in Graph 1.

Overall, this graph indicates that win percentage isn't greatly increased by basic strategy tables or Monte Carlo search when compared to a probabilistic agent and that the house edge still reigns supreme no matter the strategy.

6.5 Overall Analysis

Overall, the graphs outline several important conclusions. For one, the win percentage wasn't really tied to the complexity of the strategy in the simulation, which can be attributed both to the unexpected strength of the probabilistic agent and the dominance of the dealer advantage. Secondly, Agent 2's performance was likely hampered by its randomized betting strategy, as its win percentage was comparable to the other two agents, but it lost the most money overall. Finally, and arguably most importantly, Agent 3 is the only agent that came out with a profit and outperformed its competitors by a significant margin, illustrating the strength of card counting in Blackjack.

7 Conclusion and Future Work

Blackjack is a very popular non-deterministic card game that involves players competing against dealers and drawing cards to get as close to 21 as possible without exceeding it. The core problem of Blackjack is maximizing profit, which deals with both winning as many rounds as possible and placing optimally sized bets to yield the most profit from won rounds. The goal of this paper was to build three Blackjack playing agents and compare them to see which one generated the most profit. The three agents exhibit varying levels of complexity, from a simple Probability Based Random Agent to a Basic Strategy Simple Reflex agent, and finally a complex Monte Carlo Card Counting agent.

By running simulations that involved 500 trials of a 2 deck game of Blackjack for each agent, it was revealed that although the win percentages for the three agents were very similar, Agent 3 in a way performed the best in terms of making money. This suggests that the card counting approach employed by Agent 3 combined with the Decision Matrix constructed with Monte Carlo Search is the most effective method for maximizing earnings.

Practically speaking, it's unrealistic (if not impossible) for a real person to attempt a Monte Carlo search during a game of Blackjack, so the more useful finding is the clear benefit of card counting. For Blackjack players looking to beat the odds, picking up card counting and using it to alter bet sizes is a great idea. From the perspective of a scientist, the more useful finding is the limited difference in win percentages between the three methods – it suggests that despite all the complexity that goes into a strategy like Monte Carlo search, the actual percentage of wins seems to be around the same as a simple probabilistic approach. Monte Carlo is definitely still a useful tool and could be expanded on to be even more useful, but the findings indicate that in terms of pure wins, it doesn't seem to offer a huge advantage.

7.1 Future Work

There is plenty of room for future work on the topic, both to expand on the work of this paper and generally explore the topic of Blackjack AI. Two ideas include:

1. **Expanded Simulation**: Further research on the topic could simply expand on the simulation seen in this paper. Although 500 trials is a large sample size, an even larger sample size might reveal new trends. Furthermore, researchers could try exploring the results of adding more than 2 decks into the simulation (one could even imagine graphing the number of decks vs. profit for each agent and drawing conclusions from it). The simulation also operates on a simplified version of Blackjack, so there is plenty of room for expansion in the way of adding in rules like Split, Double, Surrender, and more.

2. **Monte Carlo Modifications**: This paper's findings indicated that Monte Carlo Tree Search did not yield a big difference in win percentage when compared to other intelligent strategies. Further research could serve to either reinforce this finding or discover a modification of Monte Carlo Search that does provide a meaningful advantage in terms of winning rounds of Blackjack. One of the sources we reviewed covered several potential modifications to MCTS (and we even used transposition tables to make our simulations more efficient), so further research could be done into how other modifications, like combining MCTS with machine learning or using parallelization to run multiple tree searches at once, affect its success in winning Blackjack games.

Both of these ideas could provide an interesting expansion to this project, and future work on the topic should generally focus on finding strategies to optimize profit in Blackjack, with specific emphasis on card-counting based strategies.

8 Team Member Contributions

8.1 Problem Description

Otto wrote the first subsection of the Problem Description section (i.e. the rules and nuances of Blackjack), while Anirudh wrote the other two sections.

8.2 Background Review

Anirudh and Otto both found 6 sources each and wrote the descriptions of the sources used in the paper.

8.3 Problem Solving Approach

Anirudh wrote the code and writeup section the first 2 agents, while Otto wrote the code and writeup section for the third agent.

8.4 Design of the Experiments and Results

Anirudh wrote the experimental design section in the writeup, while Otto coded up the actual simulations and assembled the graphs seen in the section.

8.5 Analysis

Anirudh analyzed Graphs 1 and 2, while Otto wrote the analysis for Graph 3.

8.6 Conclusion and Future Work

Anirudh wrote the conclusion section, and Otto researched and wrote the Future Work section.

8.7 Abstract

Anirudh wrote the abstract section.

References

[1] Blanchard. 2019. Blackjack: the math behind the cards. *Mathematics Senior Capstone Papers* 4 (June 2019). https://digitalcommons.latech.edu/mathematics-senior-capstone-papers/4

[2] Bakkes et al. Chaslot. 2008. Monte-carlo tree search: a new framework for game AI. *Proceedings of the Fourth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* (Oct. 2008), 216–217. https://dl.acm.org/doi/10.5555/3022539.3022579

[3] Mendoza Gianitsos, Isogawa. 2019. CS238 Project - Monte Carlo Blackjack. *Stanford University* (Dec. 2019). https://web.stanford.edu/class/aa228/reports/2019/final69.pdf

[4] Koller and Pfeffer. 1997. Representations and solutions for game-theoretic problems. *Artificial Intelligence* 94 (1997), 167–215. https://doi.org/10.1016/S0004-3702(97)00023-4

[5] Schaeffer et al. MBillings, Pena. 1999. Using probabilistic knowledge and simulation to play poker. *Proceedings of the sixteenth national conference on Artificial intelligence* (Jul 1999), 697–703. https://dl.acm.org/doi/10.5555/315149.315422

[6] Ge Pan, Xue. 2021. Intuitive Searching: An Approach to Search the Decision Policy of a Blackjack Agent. *Proceedings of Sixth International Congress on Information and Communication Technology* (Sept. 2021). https://doi-org.ezp1.lib.umn.edu/10.1007/978-981-16-2380-6_77

[7] Popyack. 2009. Blackjack-Playing Agents in an Advanced AI Course. *ACM SIGCSE Bulletin* 41 (Sept. 2009). https://doi.org/10.1145/1595496.1562944

[8] Chongwu Ruan. 2017. Blackjack and Probability. (July 2017). http://www.professorbray.net/Teaching/89s-MOU/2017-SummerTerm2/Papers/CWR_Paper1_Blackjack.pdf

[9] Schiller and Gobet. 2012. A Comparison between Cognitive and AI Models of Blackjack Strategy Learning. *KI 2012: Advances in Artificial Intelligence* (Sept. 2012), 143–155. https://doi.org/10.1007/978-3-642-33347-7_13

[10] Godlewski et al. Swiechoswki. 2022. Monte Carlo Tree Search: a review of recent modifications and applications. *University of Bahrai Scientific Journals* 56 (July 2022), 2497–2562. https://doi.org/10.1007/s10462-022-10228-y

[11] Iclanzan Vidami, Szilagyim. 2020. Real Valued Card Counting Strategies for the Game of Blackjack. *International Conference on Neural Information Processing* (Nov. 2020), 63–73. https://link.springer.com/chapter/10.1007/978-3-030-63833-7_6

[12] Wikipedia. 2023. (2023). https://en.wikipedia.org/wiki/Blackjack#Basic_strategy

[13] Yiu. 2019. Let's Play Blackjack (with Python). *Towards Data Science* (Sept. 2019). https://towardsdatascience.com/lets-play-blackjack-with-python-913ec66c732f