Team SWAAB Final Report

Overview of Our Game

All of the cards from Ace, 2, …, **max_rank** are taken from the deck and distributed randomly amongst two players so that they have the same number of cards in their hands but not necessarily the same number of each rank, e.g.

max_rank = 4

Player 1 hand: A, A, A, 2, 2, 3, 3, 4

Player 2 hand: A, 2, 2, 3, 3, 4, 4, 4

Players take turns playing one of their cards to the middle, adding on to the cards played before. The values of the middle pile are accumulated. The sequence goes on till none of the players can add a card to the middle without exceeding **max_sum**. At this point the sequence is discarded and the last player to play a card in the sequence starts the next sequence. The player able to play all of their cards first wins.

The default setup for this game is max_rank = 4 and max_sum = 11.

Since the Progress Report

Our aim for this project, as mentioned in the original project statement as well as the Plans section of the progress report is to find the most optimal strategy with the highest probability of winning the game.

Before our progress report, we had brainstormed two categories of items: *strategies* for our game, to help answer questions of how a player might be most successful while playing our game, and *hyperparameters* for our game, which could change the relative success of the strategies and otherwise influence the game's unfolding. The items in these categories were what we would analyze throughout this semester, to explore how they interplay and how they would affect players' best strategies. In the work covered up to the progress report, we focused on analyzing three player strategies: always playing the *maximum value card* without exceeding max_sum, always playing the *minimum value card* in their hand, and selecting a *random card* from their hand.

In the work after the report, we intended to focus on the other half of the items, the *hyperparameters* of the game. However, another strategy that we were initially interested in looking at that didn't make it into the work prior to the progress report (and was encouraged to explore in the report's feedback) was a *smart* strategy. We will go into further detail of what this strategy entails in the "Smart Strategy" section.

Since then, we coded in this strategy in order to run simulations of the game comparing the smart strategy against the other strategies, similar to the work shown in the progress report.

Beforehand, we had only analyzed how different strategies affected the win rates of Players 1 and 2. Additionally, because this is a two-player game, a variable under consideration by default was *whether the player moves first or second* (i.e. player 1 or 2). In the work since the progress report, we focused on two other hyperparameters of the game, *max_rank* and *max_sum*. For each, we chose one value lower than that of the default setup and two values higher, i.e.

max_rank = 3, **4**, 5, 6
max_sum = 10, **11**, 13, 16

to analyze how the players' win rates change with the parameters. And accordingly with the recommendation in the progress report feedback, we compared them to those in the default setup (the results in the progress report) to see if the distribution of player victories is similar to those in alternative game setups.

Methods

The first thing that we did was add the new items into our game simulator: coding the smart strategy and parameterizing the max_rank and max_sum. Similar to the work up to the progress report, after coding up the additional strategies, we simulated numerous games with code, then output the victories by each player using X strategy. We repeated this for every combination of our selected max_rank and max_sum values.

Code

We coded a simulation for the game in Python. Our game requires several components to make a working simulation. Below is a list of all the classes and documentation for their functions. Those highlighted in yellow were added since the Progress Report.

class Suit - enum for each suit (Clubs, Diamonds, Hearts, Spade).

class Card - holds a specific card's rank and suit.

class Deck - holds an instance of a deck with a given max rank, rank order, and a shuffle boolean.

class Player - holds an instance of a player with a given player number, strategy, and the current cards in the player's hand. The Player class keeps track of the cards a player currently has. A player's functions include playing a card under a certain strategy, receiving cards, and checking if they can still play cards.

class Round - holds an instance of a round of the game with a given max_sum and two players. The Round class keeps track of what cards have been played and whose turn it is and whether or not a round has ended. The Round class is also in charge of

simulating a round by having the two players take turns playing cards. At each step of a round, the Round class prints the current sum, which player plays what card, and all the cards that have been played. At the end of a round, the Round class returns the player who will go first the next round and prints the current hand of each player.

class Game - holds an instance of a game and accepts the hyperparameters of the game; max rank, max sum, strategy for players one and two, and a custom deck. The Game class initializes two players and a deck and simulates a game by creating instances of the Round class until a player wins. The Game class keeps track of whether or not the game is over and who goes first in the next round. At the end of a game, the Game class returns the winner of the game and the number of rounds.

class Node - represents a Node object in our game tree. It keeps track of both the players' hands, cards played, and the possible winners at each node based on the tree below.

class GameTree - used to simulate all the paths that the game can take. We have separate functions for simulating one round (i.e. cards played till max_sum is reached) and for simulating the complete game (expensive at higher values of the hyperparameters). We use these trees to make optimal moves and also to check if a game is deterministic, i.e. if a player is bound to win given the initial configuration/hands.

simulation.py - holds functions to simulate hundreds of games for each permutation of one strategy vs. another strategy and plots the results in the figure below. We will be updating this file with more functions to analyze the game.

Data & Analysis

In order to more intuitively analyze the data and draw more accurate conclusions, we input the exported the results from our simulations into a spreadsheet to create data visualizations. This process was much more complex and difficult than when it was done for the progress report, because now we were working with multivariate data and 16 separate tables. There was a lot of discussion around what data visualizations to produce, because although static multivariate visualizations exist (e.g. star plot), for our intents, it would be most useful to produce an interactive visualization. Unfortunately, none of us was proficient enough in any visualization tools or languages to do so. Accordingly, we have chosen static charts to display, most of which are made up of aggregated data, that best visualize notable findings from our analysis.

The Smart Strategy

Based on the setup of our game, both players can deduce what cards opponents have based on the cards in their own hand. They can also keep track of their opponent's hand at every step based on what cards are played. In the smart strategy, the player takes into account their opponent's current hand and aims to win each *round* of the game (cards being played up to max_sum) using this information. This is because winning a round gives the player the

advantage to start the next round, thus allowing them to play cards twice in a row—when the object of the game is to be the first to get rid of your cards, this is helpful.

The smart strategy assumes that if we attempt to win every round, we will win the game. While this is true, it rules out any type of sacrificial play where one might lose a round to win the game later. In the code for this strategy, we create the tree until the end of the round and play the card that has the maximum number of paths where one wins. Here we don't look at the proportion but the absolute number, as we want to keep our chances open. This strategy assumes optimality from the opponent, i.e. that they will always play their best card. Therefore, if we ever encounter a situation where the opponent is bound to win, we do not care which card we play. We can replace this with the "max" strategy to keep our chances open if the opponent makes a mistake.

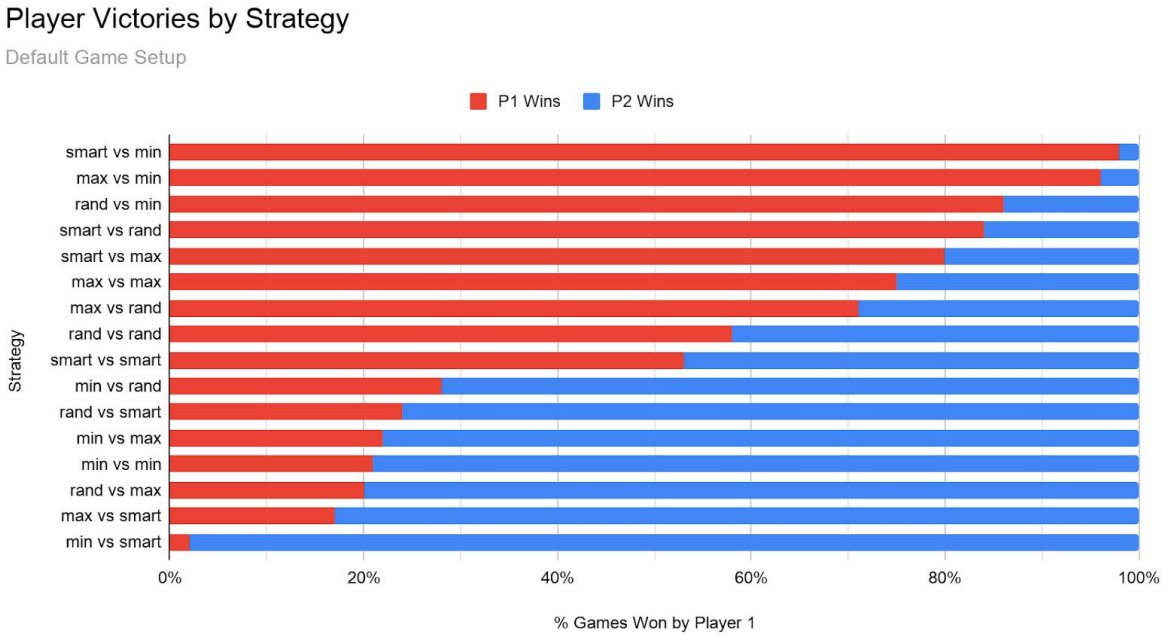

Fig. 1: A stacked bar chart comparing how Player 1 using strategy X fares against Player 2 using strategy Y. Sorted by Player 1 wins descending / Player 2 wins ascending.

Default Game Setup: max_rank = 4, max_sum = 11

Strategy Matchup	P1 wins	P2 wins	Strategy	Avg P1 Wins	Avg P2 Wins
smart vs smart	53	47	Smart	**78.75**	**76**
smart vs rand	84	16			
smart vs max	80	20			
smart vs min	98	2			
rand vs smart	24	76	Rand	47	39.75
rand vs rand	58	42			
rand vs max	20	80			

rand vs min	86	14			
max vs smart	17	83	Max	64.75	50.75
max vs rand	71	29			
max vs max	75	25			
max vs min	96	4			
min vs smart	2	98	Min	18.25	24.75
min vs rand	28	72			
min vs max	22	78			
min vs min	21	79			

Fig. 2: A table showing the number of wins for each strategy matchup in the default game setup (max_rank = 4, max_sum = 11) and the average percentage of Player 1 and Player 2 wins for each strategy. The data on the left side was used to create Fig. 1.

After running simulations with the smart strategy, we found that the strategy is actually more successful than any other previous strategies we investigated. *Fig. 1* displays the win percentages of each strategy against another, and we see that the smart strategy wins a majority of the time against a different strategy. The smart strategy wins a majority despite being the first or the second player. Furthermore, *Fig. 2* displays the numeric win rate values of each strategy against another when we have our default parameters, max_rank = 4 and max_sum = 11. On the right hand side of the figure, we calculated the average number of wins per strategy for Players 1 and 2. As seen by the averages, the smart strategy has the highest average number of wins whether being used by Player 1 (with an average of 78.75 wins) or Player 2 (with an average of 76 wins).

The interesting point to make is, according to *Fig. 1*, the smart strategy still wins despite being used by Player 2, who takes the second turn. This detail displays a different pattern than we are used to seeing (i.e. from the Progress Report). If we look at the right side of *Fig. 2*, we see that, typically, the player who goes first wins more often. Despite that fact, we still see that when Player 2 uses the smart strategy, they may still win a majority of the time. Therefore, our data conclude that the smart strategy is the best to choose when playing second.

Furthermore, we can conclude that the smart strategy is oftentimes the most optimal strategy to use, especially when playing second. Although the strategy may not always be optimal in every scenario, based on our simulations, it will be the best choice for most of the game scenarios. We will expand on the particular game scenarios' optimal strategy choice in the "Conclusion" section.

Hyperparameters: Interplay Between max_sum and max_rank

Our initial plan was to determine how the values for the max_sum and max_rank influenced the win distribution. Accordingly, we wanted to use one fixed value of max_rank to observe how

changing the max_sum influenced the win distribution and one fixed value of max_sum to observe how changing the max_rank influenced the win distribution, i.e. only changing one variable at a time.

Before doing any of the work to analyze max_rank and max_sum, we had two assumptions about the characteristics of our data. The first was that the hyperparameters acted *independently* of one another to influence the outcome of the game. Therefore, all of the charts with fixed max_sum and changing max_rank would have a similar shape, as would all the charts with fixed max_rank and changing max_sum.

In a similar fashion, in one chart where the max_sum was fixed and the max_rank was varied, the lines being close together would indicate that the win rates were similar no matter the max_rank, and vice versa.

The second was that the data would trend consistently as the values for the hyperparameters changed, i.e. we could say that “as max_rank increased, Player X won more of the time” or “as max_sum decreased, strategy X worked better”.

However, we were surprised to find that *neither* of these assumptions held true.

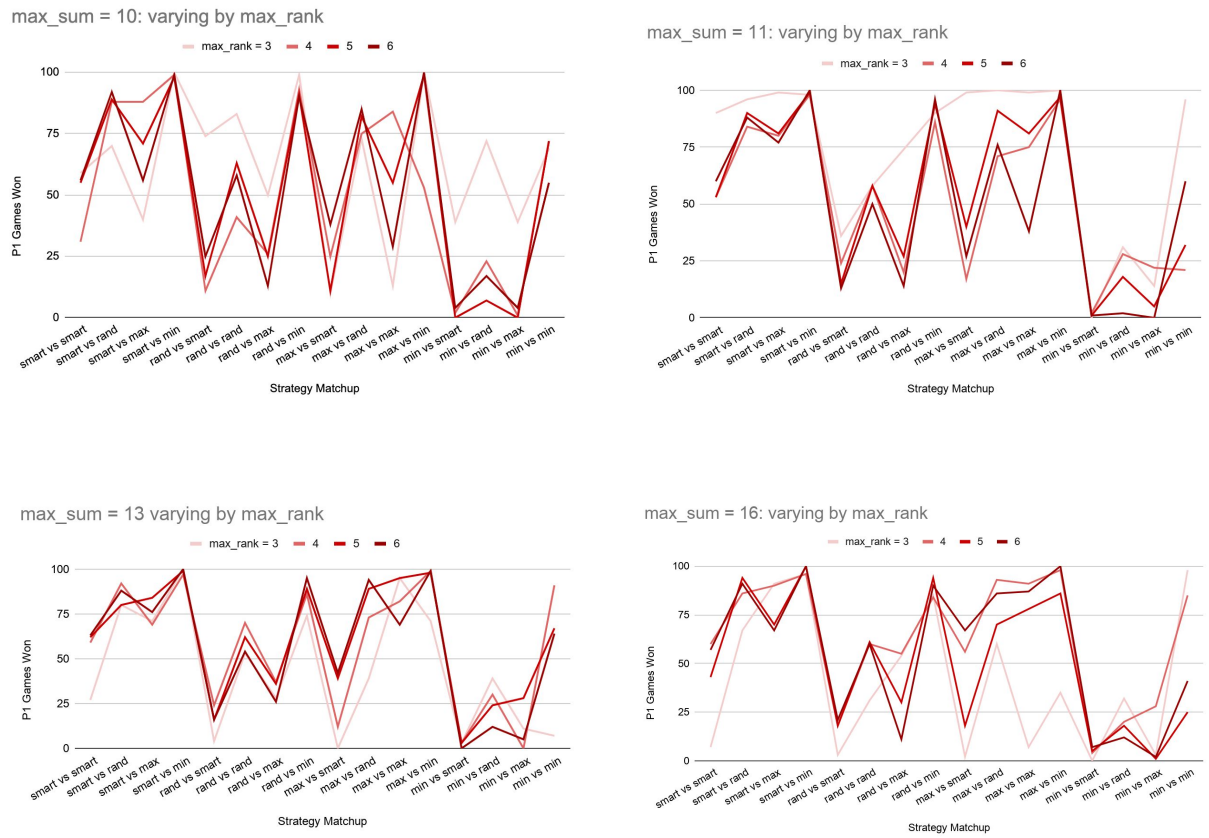

Fig. 3: The four charts displaying a fixed max_sum and varying max_rank. While there is some visual similarity in the general trend shapes between the four charts, a closer look at each *of the trendlines indicates some*

significant differences between each chart. All four charts were included to point out that no one *chart seems to meanly represent the trends of all four.*

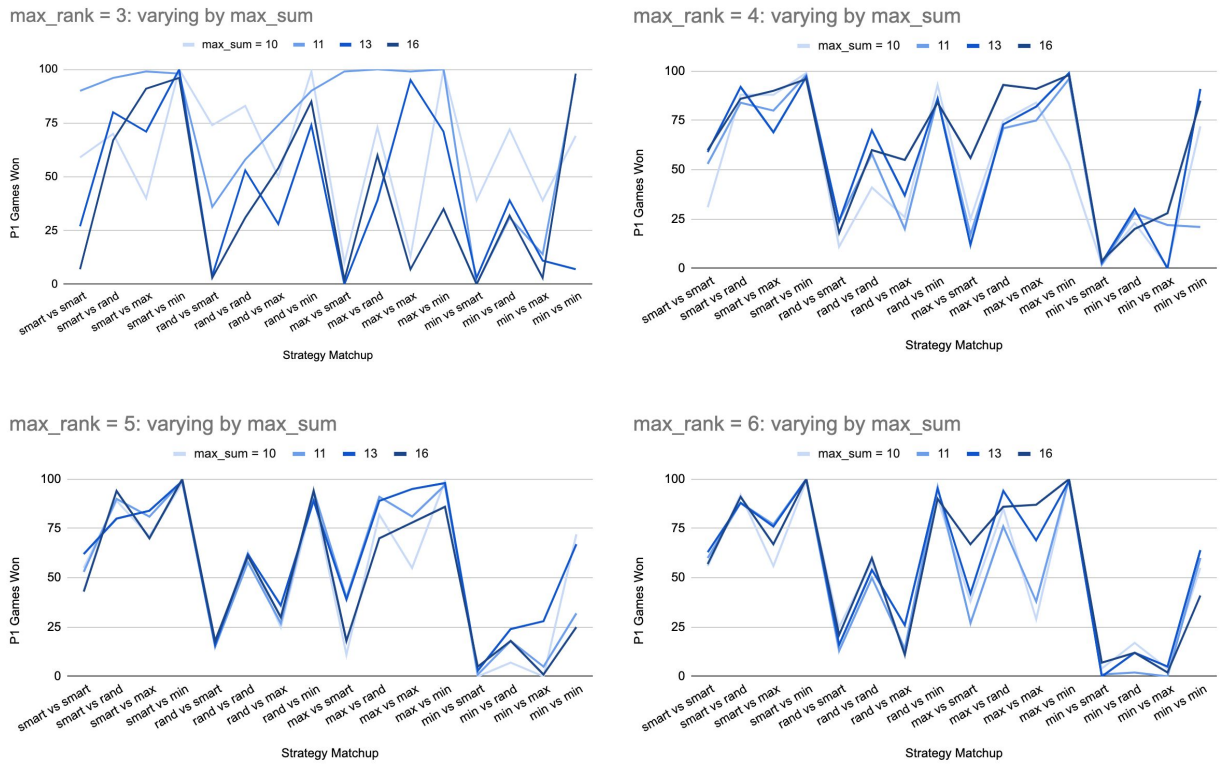

Fig. 4: The four charts displaying a fixed max_rank and varying max_sum. This can be seen as the "inverse" charts of those in Fig. 2, i.e. each blue chart is formed by taking all of the same-colored lines from the four charts and placing them onto a single chart.

What conclusions *could* be drawn?

We created data graphs to display win distributions between Player 1 and Player 2 depending on the max_sum and the max_rank used in the games. In red (*Fig. 3*), we have graphs that display the outcomes of each strategy playing against each other with a fixed max_sum and varying max_rank. On the other hand, in blue (*Fig. 4)* we display the outcomes with a fixed max_rank and varying max_sum. We can analyze varying max_sum and varying max_rank by looking at the two sets of different colored graphs.

First, we look at the fixed max_sum and the varying max_rank (*Fig. 3*). In each of the four red graphs, we increment the fixed max_sum and look at the outcomes with varying max_ranks 3, 4, 5, 6 (denoted by the darkness of the various red trendlines). When the fixed max_sum was 10, we see the four lines are not aligned, meaning there was a lot of variation in the win distribution when it came to the varying max_rank. Therefore we can conclude that when playing with a max_sum = 10, the max_rank can heavily influence the outcome of the games depending on the strategies used. We can also conclude that sticking with a single strategy may be more difficult, because the winning strategy will depend on the max_rank as well. When the

fixed max_sum was 11 or 13, we see there is a lot of overlap in the lines. Therefore, the varying max_rank had a lesser effect on the outcomes, meaning a single winning strategy may be generalized when the max_sum is either 11 or 13. Finally, when looking at a fixed max_sum of 16, we also see a lot of variation between the four max_rank values. Therefore, when the max_sum is 16, it may be difficult to use a single strategy throughout all of the games, and may be better to change up the strategy depending on the max_rank.

Next, we can look at the graphs with a fixed max_rank and varying max_sum (*Fig. 4*). In general, we see a lot more variation between these graphs, relative to the red ones, meaning that changing the max_sum may have a greater effect on the outcome of the game rather than the rank. When the fixed max_rank is 3, we find very little overlap between the lines, meaning a change in the max_sum greatly differs the winning strategy in a strategy matchup. Therefore, it is difficult to conclude an optimal strategy when the max_rank = 3. Looking at a fixed max_rank = 4, we see a little less variation than max_rank = 3, but still little overlap. Next, we see that the trendlines in the graphs of fixed max_rank of 5 and 6 have more overlap, meaning the winning strategies in their respective matchups will be the same throughout a varying max_sum. When the max_rank is 5 or 6, we can use the same strategy despite a change in max_sum and still expect a similar number of wins.

In general, when the graph displays a lot of variation within a fixed parameter value (i.e. fixed max_rank = 3), the outcome of the game heavily depends on the max_rank hyperparameter, in addition to the strategies chosen by both players. When the graphs have little variation (e.g. fixed max_rank = 5), the varying parameter is not a strong influence on the outcome of the game, making the chosen strategies relatively more important. In the case of having a fixed max_rank = 5 (*Fig. 4*), the varying max_sum does not have much influence on the outcome of the game. Therefore, the strategies play a major role in determining the winner of the game, no matter the max_sum.

Altogether, not only do the max_rank and max_sum together influence what the optimal strategy is, they also impact the strength that the strategy holds over the outcome of the game.

Extra: The “Fairest” Game

As we began factoring in many of the hyperparameters that the players are stuck with before the game even begins, one question that we came up with was “Does our default game setup give a fair chance to either player to win? How can we minimize unfair advantages to either player based simply off of the construction of the game?” In other words, we want to find the values for the max_rank and max_sum that would minimize the differences between the win rates of the two players, thus minimizing the effect of whether the player moves first or second on the probability of winning.

There are two ways that we could analyze the data to answer this question. The first is sum up the total wins from every strategy matchup in one distinct setup of the game (i.e. set values for max_rank and max_sum) and calculate the ratio of total wins between Player 1 and Player 2.

The closer this value is to 1, the closer there is a 1:1 ratio of wins between the players, thus minimizing the effect of the player moving first or second.

The second way takes into account the outcomes of the particular strategies, as this is a weakness that the first method faced—*if one strategy is intuitively ineffective such that a player likely would not choose to use it, why give it an equal weight to the more preferred and effective strategies?* Rather than comparing the total wins between Player 1 and Player 2, we find the average win rate of each strategy for each player and compare it to the other player's average win rate using the same strategy. We then calculate the *average difference* between the players' win rates for each strategy.

For example, we could find that when Player 1 uses strategy X, they win an average of 60% of the time, regardless of what strategy their opponent uses. We would then find that when Player 2 uses the same strategy X, they win an average of 72% of the time, regardless of their opponent's strategy. The difference between the players' win rates for strategy X is 12%. We then would calculate the average of the differences for all the strategies. This would be the *average difference in strategy win rates* for this particular setup of the game.

After repeating this process for all 16 of our game setups, we would look for the setup with the *minimum* average difference. This would equate to the game setup in which the success of the strategies was the least influenced by whether the player moved first or second. By finding this, we would be able to find the "fairest" game setup for Player 1 and Player 2.

We decided to calculate both of these values and plot them. We expected to see a positive correlation between the two.

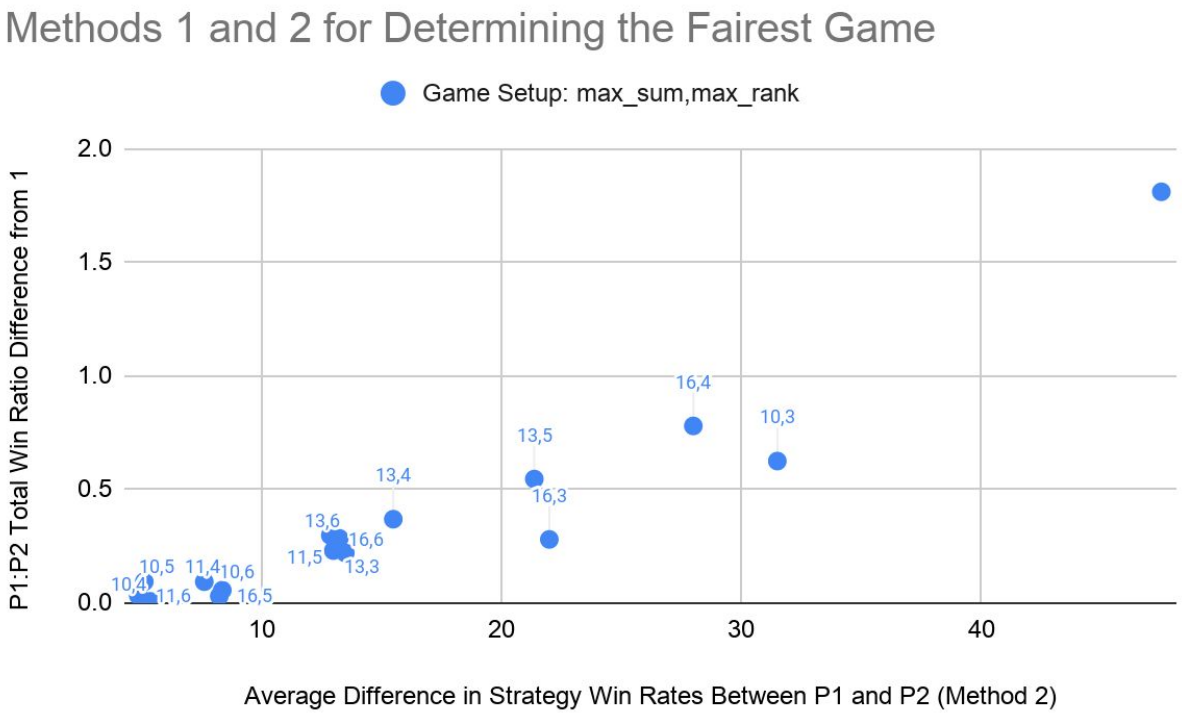

Fig. 5: Scatterplot of the values for the two methods to determine the fairest game. The "fairest" game setups are represented by the data points closest to (0,0).

In this scatterplot, we basically plotted the points with the value on each axis coordinating to their distance from the optimal values indicating a fair game—for method 1, 1; for method 2, 0. This means that the data points closest to the bottom-left corner, the origin, are the game setups in which whether the player moves first or second does not provide an advantage for the outcome of the game.

Our original game setup is of a max_sum,max_rank of 11,4. However, the points for 10,4, 10,5, and 11,6 all seem to be relatively "fairer" setups, particularly 10,4. Therefore, we should consider changing our default setup of the game from max_sum = 11, max_rank = 4 to max_sum = 10, max_rank = 4. This would give the two players a relatively equal chance of winning, independent of strategy, skill, or smarts.

Conclusion: What is the Optimal Way to Play?

Our intuition since the beginning of this project told us that out of our three original strategies, max, min, and rand, max would perform the best. Once we added in a "smart" strategy, we assumed that this would outperform the rest, and that if a (human) player were playing this game in real life, they would either use the max or the smart strategy, depending on how much mental effort they were willing to put into the game.

In order to determine the best strategy in each game setup, we took the average win rate of each strategy for Player 1 and Player 2, e.g. in a game setup for max_rank = 4 and max_sum = 11, we calculated the average Player 1 win rate for the smart vs smart, smart vs rand, smart vs max, and smart vs min cases. This would give us the average Player 1 win rate using the smart strategy in this particular game setup. These values can be seen on the right side of *Fig. 2*.

We repeated this for all four strategies, for both players, in each game setup. By finding the maximum value for each player, this gave us the optimal strategies in the particular game setup for Player 1 and Player 2. Because this was the average of all the matchups, this means that if you are unaware of your opponent's strategy, choosing this strategy is your best bet.

Decision Map of Ideal Strategy

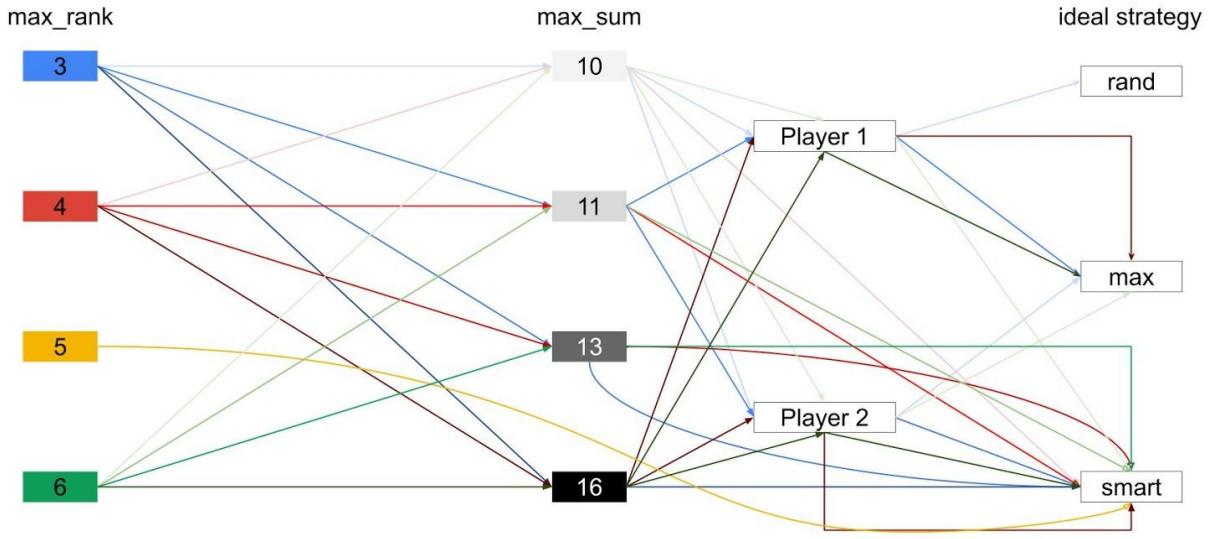

Fig. 6: A decision map indicating the strategy with the best probability of winning given the three parameters of the game: max_rank, max_sum, and whether moving first or second (Player 1 or 2). The hues of the lines indicate a max_rank, the shades indicate a max_sum, and the arrowheads indicate whether for Player 1, Player 2, or both.

Above is a decision map indicating the optimal strategy for a player given the parameters that go into the game: the max_rank, max_sum, and whether they are moving first or second. While ideally we hoped to be able to draw conclusions as to *why* these strategies are the best under certain conditions, our limited cases do not illustrate a predictable outcome nor a predictable best strategy. We will come back to this point in our "Possible Future Work" section.

Somewhat according to our original assumptions/intuition, the smart strategy fared the best in most game setups; the max strategy was occasionally better. This is the simple answer to our overarching question of: "What is the optimal strategy that gives the highest probability of winning the game?"

One anomaly we found was that the random card strategy fared the best for Player 1 when the max_rank = 3, max_sum = 10. Unfortunately, because we were not able to justify why the max

or smart strategies worked the best under particular conditions, we similarly are unable to do so for the rand strategy in this circumstance. One possible explanation is that this was by chance, say if 100 trials was not enough to centralize the outcomes of the strategies. A more probable explanation, however, likely has to do with the fact that the max_rank and max_sum values were relatively low. This is supported by the inconsistency between trendlines in the first charts of *Fig. 3* and *Fig. 4*.

Considering Game Theory

Like with determining the fairest game, there is a second method to determining the optimal strategy. Let us incorporate some basic game theory, i.e. by making assumptions on the strategy that the opponent will use, and choosing a strategy accordingly. As mentioned before, we assume that real players would likely choose to use either the max strategy or the smart strategy. Therefore, the ideal strategy for a player will likely depend on this choice by the opponent. We will include this variable in our decision map.

Decision Map of Ideal Strategy Considering Opponent's Strategy

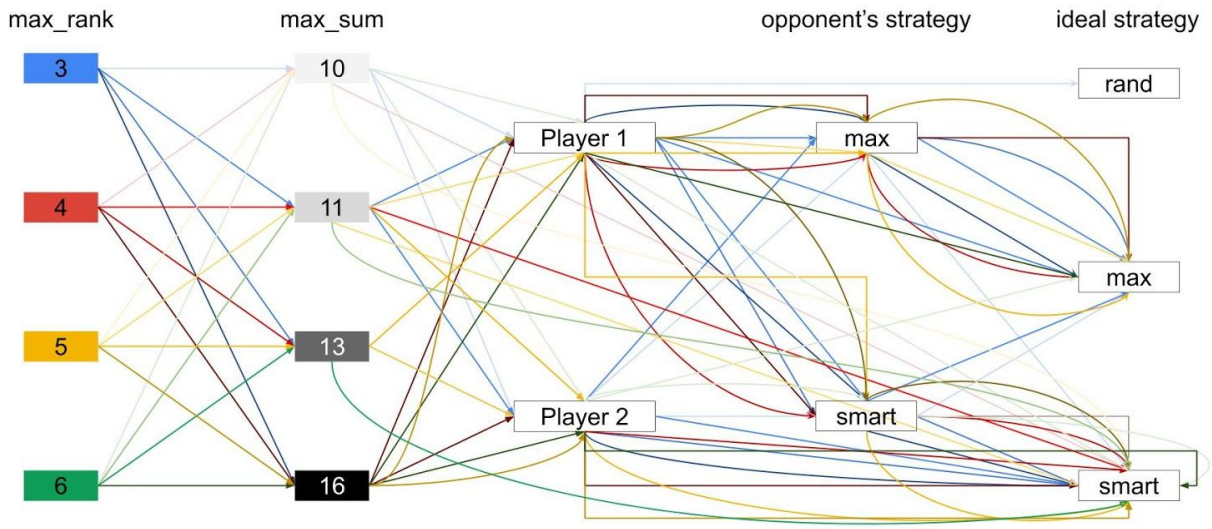

Fig. 7: A decision map indicating the strategy with the best probability of winning given the four parameters of the game: max_rank, max_sum, whether moving first or second (Player 1 or 2), and the opponent's strategy (max or smart). The hues of the lines indicate a max_rank, the shades indicate a max_sum, and the arrowheads indicate whether for Player 1, Player 2, or both.

Where the lines were drawn on the decision map were decided by comparing in one game case, all of the matchups where the opponent used either the max or smart strategy, e.g. the cases where Player 1 uses smart strategy is smart vs smart, smart vs rand, smart vs max, smart vs min. The ideal strategy, then, was the one that formed the matchup that minimized the opponent's probability of winning.

Did considering the opponent's strategy make a difference? Actually, yes. When taking into account the opponent's chosen strategy, the strategy with the highest probability of winning was not always the ideal strategy indicated in *Fig. 6*. In these cases, the lines are drawn from the *opponent's strategy* column to the *ideal strategy* column.

There are some cases where the opponent's strategy didn't make a difference, so the ideal strategy indicated in *Fig. 6* is also the ideal strategy in *Fig. 7*. In these cases, the lines are drawn directly from the *Player* column, or even the *max_sum* column if moving first or second did not influence the ideal strategy, to the *ideal strategy* column.

By the sheer number of arrows going into "smart" as the ideal strategy, we can tell that the smart strategy still dominates as the optimal strategy overall. However, there are still many instances where the max strategy fares better.

So the more complex answer to "What is the optimal strategy?" is—it depends. For the most part, the smart strategy fares the best. There are still cases where the max strategy is the optimal choice. And there is even the lone anomaly where the rand strategy works the best. One thing that can be said for certain is that the optimal strategy depends on the max_rank, max_sum, whether the player moves first or second, and the opponent's strategy.

Mathematical Conjecture

In the smart strategy, the player tries to win every round, i.e. to try to play the last card before the sum exceeds max_sum. Therefore we want to move the game such that the round has an odd number of cards played (called an odd sequence) if you started it and even (called an even sequence) otherwise. This can be modelled using generating functions as

$$x_1 + x_2 + \ldots + x_{max\_rank} = max\_sum$$

Here we want to use x_i to specify the values that can be made using the remaining cards of "rank i" in both the players' hands.

Let us take an example where the max_sum = 5 and player hands are

Player 1: [Hearts 1, Spades 1, Clubs 2, Clubs 3]
Player 2: [Diamonds 1, Spades 2, Hearts 2, Diamonds 2]

Therefore we have three cards with rank 1, four cards with rank 2, and one card with rank 3. Therefore the generating functions would be

A: $x_1 - 1 + x + x^2 + x^3$
B: $x_2 - 1 + x^2 + x^4 + x^6 + x^8$
C: $x_3 - 1 + x^3$

Therefore the number of solutions would be the coefficient of x^5 in A*B*C, i.e. $(1 + x + x^2 + x^3)(1 + x^2 + x^4 + x^6 + x^8)(1 + x^3)$.

A few things to keep in mind:

1. We have a product of, at most, max_rank polynomials.
2. Each polynomial has at most five terms, as each rank has at most four cards corresponding to four suits. The extra term arises as one can choose to play no card of that rank.
3. We make an assumption that the sum of all cards ≥ max_sum. Otherwise the round will conclude when all cards are played no matter what.
4. We also make an assumption that a subset of cards can correctly add up to max_sum but that doesn't have to be the case. This case can be analyzed using a slack variable.
5. We also assume that the players will take turns alternatively. This might not occur if a player has very high value / fewer cards.

The above special cases are only seen towards the endgame and therefore this analysis applies for most of the game and these cases can be handled with extra care.

As we are multiplying polynomials with at most five terms, we can use a naive polynomial multiplication algorithm to keep track of the powers adding up to max_sum and adding up their coefficients to calculate the number of sequences. As we know which polynomial corresponds to which rank, everytime we add up powers to max_sum we can count the number of cards in the sequence and determine if it's an odd or even sequence.

For example - if we multiply x^2 from A, 1 from B and x^3 from C, then we know that we used (2 + 0 + 1) = 3 cards.

The way to use the above analysis is to assume that when a card is played, how many sequences of the remaining sequences will end at you.

Case 1: You start the sequence

For every rank you have, you count the number of sequences that will end at you given you play one card of that rank. This is the same as counting the number of even sequences that add up to the (max_sum - rank). You choose to play the card/rank which has the highest number/proportion of such even sequences.

Case 2: Opponent starts the sequence

This is the same as the above case except you can replace max_sum by (max_sum - opponent's card rank).

There seems to be no closed form solution for this but it can be efficiently coded out with decent performance due to the small nature of the hyper-parameters. It will still be faster than simulating a tree for a round but will not give as much information. One can look at simulating the tree as looking at all the sequences and this can get expensive for higher values of hyperparameters. But building the tree has the added advantage of coming up with optimal moves assuming optimality from the opponent as well.

Possible Future Work

Our initial plan of attack for our analysis was to take a look at the *strategies* and the *hyperparameters* of the game and how they influenced the outcomes and win distributions. Additionally, as mentioned in our Progress Report, we decided to take a *data exploration* approach, allowing the data to guide our conclusions rather than coming up with what we *thought* were the optimal strategies and conditions for players, then allowing those assumptions to bias our findings.

We did a lot of analysis of our data, but were not able to come up with *explanations* for our results. Therefore, the brunt of the possible future work would consist of finding those explanations. This would likely require a deeper look into mathematical topics that could potentially be relevant to our game, as well as analysis of other card games (the unique limitations of playing cards could play a part into many card games).

Additionally, we analyzed only four strategies (or general playstyles), and two hyperparameters. While we assumed that there would be some general visible trend with the hyperparameters (as mentioned in the “Hyperparameters” section), there was not, so we were not able to generalize our findings for all possible setups of the game. Granted, we only analyzed four values of each hyperparameter, so there wasn’t enough data to extrapolate. Another avenue of future work, then, would be to analyze more (or even all possible) values of the max_rank and max_sum in order to come up with generalizable conclusions to answer our question of the “most optimal way to play the game” under any circumstance. Furthermore, we could explore more possible strategies, since there are many missed ways that people would think to play.

Lastly, while we partially focused on how the hyperparameters affect the game, in the “Fairest Game” section, we were able to find that the 10,4 game setup may be a more ideal default setup of the game. Therefore, if we were to choose to focus on the 10,4 setup, we could concentrate our efforts into analyzing this one version of the game. Also, the question of “What makes the fairest game?” has still gone unanswered. As to why the 10,4, 10,5, and 11,6 setups were more fair for the two players than the original 11,4 setup is still unknown. If we engaged in the future work mentioned in the previous paragraph, we may even find an even “fairer” game. Nevertheless, the interplay between the hyperparameters max_sum and max_rank and discovering the explanation behind their effects on 1) the optimal strategy, 2) the importance of the chosen strategy on the outcome of the game, and 3) the fairest game, is a direction for possible future work.