QWIXX STRATEGIES USING SIMULATION AND MCMC METHODS

A Thesis
presented to
the Faculty of California Polytechnic State University,
San Luis Obispo

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in Statistics

by
Joshua Blank
June 2024
COMMITTEE MEMBERSHIP

TITLE: Qwixx Strategies Using Simulation and MCMC Methods

AUTHOR: Joshua Blank

DATE SUBMITTED: June 2024

COMMITTEE CHAIR: Kevin James Ross, Ph.D
Professor of Statistics

COMMITTEE MEMBER: Matthew Carlton, Ph.D
Professor of Statistics

COMMITTEE MEMBER: Allison Theobold, Ph.D
Professor of Statistics
ABSTRACT

Qwixx Strategies Using Simulation and MCMC Methods

Joshua Blank

This study explores optimal strategies for maximizing scores and winning in the popular dice game Qwixx, analyzing both single and multiplayer gameplay scenarios. Through extensive simulations, various strategies were tested and compared, including a score-based approach that uses a formula tuned by MCMC random walks, and race-to-lock approaches which use absorbing Markov chain qualities of individual score sheet rows to find ways to lock rows as quickly as possible. Results indicate that employing a score-based strategy, considering gap, count, position, skip, and likelihood scores, significantly improves performance in single player games, while move restrictions based on specific dice roll sums in the race-to-lock strategy were found to enhance winning and scoring points in multiplayer games. While the results do not achieve the optimal scores attained by prior informal work, the study provides valuable insights into decision-making processes and gameplay optimization for Qwixx enthusiasts, offering practical guidance for players seeking to enhance their performance and strategic prowess in the game. It also serves as a lesson for how to approach optimization problems in the future.

Keywords: Qwixx, Simulation, Absorbing Markov Chain, Random Walk
TABLE OF CONTENTS

LIST OF TABLES........................................................................................................... vii
LIST OF FIGURES....................................................................................................... ix
CHAPTER
1. INTRODUCTION ........................................................................................................ 1
   1.1 Qwixx Rules .......................................................................................................... 1
   1.2 Problem ................................................................................................................ 3
   1.3 Approach and Goals ............................................................................................. 4
2. LITERATURE REVIEW .............................................................................................. 6
3. METHODOLOGY ......................................................................................................... 9
   3.1 Code Overview ...................................................................................................... 9
   3.2 No Strategy .......................................................................................................... 10
   3.3 Score-Based Strategy .......................................................................................... 10
      3.3.1 Strategy Components .................................................................................. 10
      3.3.2 Tuning and Exploration ............................................................................. 14
   3.4 Race-to-Lock Strategy ......................................................................................... 15
   3.5 Multiplayer Game Strategies .............................................................................. 24
4. RESULTS .................................................................................................................... 25
   4.1 No Strategy .......................................................................................................... 25
   4.2 Score-Based Strategy .......................................................................................... 26
      4.2.1 Tuned Gap, Count, and Position Scores Only ............................................. 26
      4.2.2 Tuned Gap, Count, Position, and Skip Scores Only .................................... 27
      4.2.3 Tuned Gap, Count, Position, Skip, and Likelihood Scores ....................... 29
      4.2.4 Attempts for Improvement and Tuning ....................................................... 31
   4.3 Race-to-Lock Strategy ......................................................................................... 33
      4.3.1 No Move Restrictions .................................................................................. 33
      4.3.2 Move Restrictions (1 Sum) ......................................................................... 34
      4.3.3 Move Restrictions (2 Sums) ....................................................................... 35
      4.3.4 Move Restrictions (3 Sums) ....................................................................... 37
   4.4 Multiplayer Games ............................................................................................... 38
      4.4.1 Score-Based vs. Score-Based ................................................................. 38
      4.4.2 Race-to-Lock vs. Race-to-Lock ............................................................... 39
      4.4.3 Score-Based vs. Race-to-Lock ................................................................. 40
5. CONCLUSION ........................................................................................................... 43
   5.1 Summary ............................................................................................................. 43
   5.2 Informing Gameplay ............................................................................................ 44
6. DISCUSSION .............................................................................................................. 46
BIBLIOGRAPHY ............................................................................................................. 49
APPENDICES
   A. Helper Functions .................................................................................................... 50
   B. Code for Strategy 0 (No Strategy) Function ..................................................... 51
   C. Code for Strategy 1 (Score-Based Strategy) Function ........................................ 55
   E. Code for Single Player Game Simulation Function ........................................... 69
F. Code for Multiplayer Game Simulation Function ...........................................73
G. Code for Single Player Game Simulation Results ..........................................77
H. Code for 2 Player Game Simulation Results ..................................................81
I. Code for Tuning Parameters ........................................................................84
J. Code for Race-to-Lock Strategy Transition Probabilities and Means .............86
## LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Summary statistics of scores from 10,000 simulations using “No Strategy”</td>
<td>25</td>
</tr>
<tr>
<td>2. Summary statistics of scores from 10,000 simulations using the score-based strategy applying only tuned gap, count, and position scores</td>
<td>27</td>
</tr>
<tr>
<td>3. Summary statistics of scores from 10,000 simulations using the score-based strategy applying tuned gap, count, position and skip scores only</td>
<td>28</td>
</tr>
<tr>
<td>4. Summary statistics of scores from 10,000 simulations using the score-based strategy applying tuned gap, count, position, skip, and likelihood scores</td>
<td>30</td>
</tr>
<tr>
<td>5. Top 5 skip scores from 2 different runs of the MCMC algorithm</td>
<td>33</td>
</tr>
<tr>
<td>6. Summary statistics of scores from 10,000 simulations using the race-to-lock strategy with no move restrictions</td>
<td>34</td>
</tr>
<tr>
<td>7. Summary statistics of scores from 10,000 simulations using the race-to-lock strategy with move restrictions based on the sum of 2 dice rolls</td>
<td>35</td>
</tr>
<tr>
<td>8. Summary statistics of scores from 10,000 simulations using the race-to-lock strategy with move restrictions based on 2 different sums of 2 of the dice from 3 dice rolls</td>
<td>36</td>
</tr>
<tr>
<td>9. Summary statistics of scores from 10,000 simulations using the race-to-lock strategy with move restrictions based on 3 different sums of 2 of the dice from 3 dice rolls</td>
<td>37</td>
</tr>
<tr>
<td>10. Summary statistics of scores from 10,000 simulations of 2 player games where both players use the optimal score-based strategy</td>
<td>39</td>
</tr>
</tbody>
</table>
11. Summary statistics of scores from 10,000 simulations of 2 player games where both players use the optimal race-to-lock strategy ........................................40

12. Summary statistics of scores from 10,000 simulations of 2 player games where the optimal score-based strategy competes against the optimal race-to-lock strategy ........................................................................................................41
### LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Qwixx Scorecard and Possible Dice Roll</td>
<td>2</td>
</tr>
<tr>
<td>2.</td>
<td>Race-to-lock absorbing Markov chain with no move restrictions</td>
<td>17</td>
</tr>
<tr>
<td>3.</td>
<td>Race-to-lock absorbing Markov chain including the mean number of rolls</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>required to reach the absorbing state based on the sum of 2 dice rolls</td>
<td></td>
</tr>
<tr>
<td>4.</td>
<td>Race-to-lock absorbing Markov chain including the mean number of rolls</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>required to reach the absorbing state and move restrictions based on the sum of</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 dice rolls</td>
<td></td>
</tr>
<tr>
<td>5.</td>
<td>Race-to-lock absorbing Markov chain including the mean number of rolls</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>required to reach the absorbing state based on 2 sums of 2 dice from 3 dice rolls</td>
<td></td>
</tr>
<tr>
<td>6.</td>
<td>Race-to-lock absorbing Markov chain including the mean number of rolls</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td>required to reach the absorbing state and move restrictions based on 2 sums of</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 dice from 3 dice rolls</td>
<td></td>
</tr>
<tr>
<td>7.</td>
<td>Race-to-lock absorbing Markov chain including the mean number of rolls</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td>required to reach the absorbing state based on 3 sums of 2 dice from 3 dice rolls</td>
<td></td>
</tr>
<tr>
<td>8.</td>
<td>Race-to-lock absorbing Markov chain including the mean number of rolls</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>required to reach the absorbing state and move restrictions based on 3 sums of</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 dice from 3 dice rolls</td>
<td></td>
</tr>
<tr>
<td>9.</td>
<td>Histogram of scores from 10,000 simulations using “No Strategy”</td>
<td>26</td>
</tr>
<tr>
<td>10.</td>
<td>Histogram of scores from 10,000 simulations using the score-based strategy</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td>applying only tuned gap, count, and position scores</td>
<td></td>
</tr>
</tbody>
</table>
11. Histogram of scores from 10,000 simulations using the score-based strategy 
   applying tuned gap, count, position, and skip scores only ..............................................28
12. Example decision made by strategy without likelihood scores ..............................................29
13. Histogram of scores from 10,000 simulations using the score-based strategy 
   applying tuned gap, count, position, skip, and likelihood scores ......................................30
14. Example decision made by strategy with likelihood scores ..............................................31
15. Histogram of scores from 10,000 simulations using the race-to-lock strategy 
   with no restrictions ........................................................................................................................................34
16. Histogram of scores from 10,000 simulations using the race-to-lock strategy 
   with move restrictions based on the sum of 2 dice rolls .........................................................35
17. Histogram of scores from 10,000 simulations using the race-to-lock strategy 
   with move restrictions based on 2 different sums of 2 of the dice from 3 dice 
   rolls.........................................................................................................................................................36
18. Histogram of scores from 10,000 simulations using the race-to-lock strategy 
   with move restrictions based on 2 different sums of 2 of the dice from 3 dice 
   rolls.........................................................................................................................................................38
19. Histogram of scores from 10,000 simulations of 1 player from 2 player games 
   where both players use the optimal score-based strategy .........................................................39
20. Histogram of scores from 10,000 simulations of 1 player from 2 player games 
   where both players use the optimal race-to-lock strategy .........................................................40
21. Density ridge plot of scores from 10,000 simulations of 2 player games where 
   the optimal score-based strategy competes against the optimal race-to-lock 
   strategy ...............................................................................................................................................41
22. Scatterplot of scores from 10,000 simulations of 2 player games where the both players use the optimal race-to-lock strategy.
1.1 Qwixx Rules

Qwixx is a single or multiplayer dice game in which players attempt to score as many points as possible, or outscore their opponents, by crossing off spaces on their score sheet. Each player’s score sheet consists of 4 different colored rows (red, yellow, green, blue), each containing 11 numbered spaces. In the red and yellow rows, the spaces are ordered from 2 to 12, while the green and blue rows have spaces ordered backwards from 12 to 2.

At each player’s turn, 6 dice are rolled (2 white, 1 red, 1 yellow, 1 green, 1 blue), and the player has the option to make 2 moves. For the first move, the sum of the 2 white dice can be used to cross off 1 space on any of the 4 rows. For the second move, the sum of 1 white dice and 1 colored dice can be used to cross off 1 space in the row matching the colored dice. In multiplayer games, players take turns rolling the dice. When it is not a player’s turn, they still have the option to cross off any space with the sum of the 2 white dice but cannot make a second move. The number of points earned for each row can be determined by the equation \(x(x + 1)/2\), where \(x\) is the number of spaces crossed off.

However, spaces may only be crossed off from left to right in each row. Additionally, players may choose, or be forced to skip 1 or both of their moves. If a player skips both their moves during their turn, they will incur a penalty which will subtract 5 points from their final score. Players can also lock specific rows, earning an extra marked off space and preventing all players from making moves on the locked row. This occurs when a
player has crossed off at least 5 spaces in a given row, in addition to the rightmost space (12 on red and yellow or 2 on green and blue). Finally, the game ends either when a player has committed 4 penalties, or 2 rows have been locked. An example of the game score card and possible dice roll are displayed in Figure 1.

Figure 1 Qwixx Scorecard and Possible Dice Roll
1.2 Problem

Although Qwixx may seem trivial at first glance, finding an optimal solution to maximize scores is a very complex task due to the sheer number of states that a score sheet can be in, as well as the number of dice rolls that can occur at each turn. These complexities are further multiplied in multiplayer games with the addition of multiple score sheets and a change in the focus of the game from maximizing scores to simply outscoring opponents before the game ends.

Additionally, very little work has been done to find and compare optimal strategies. In 2016, Reddit user Bmhowe34 posted their work on 2 different Qwixx strategies to the subreddit r/boardgames, 1 of which they claim to be the optimal solution for maximizing single player game scores. Unfortunately, this work has not been formally verified or published, and the methods used to find the optimal strategy require programming knowledge that I do not have. Initially, the plan for this report was to replicate the single player optimal strategy by creating a lookup table to find the best move at each board state using dynamic programming. I would then use this table to discern more general properties of the optimal strategy and find other strategies that could compete with it in multiplayer games. However, I found that this was infeasible as I do not have access to the necessary computational power.
1.3 Approach and Goals

I propose a new framework to approximate the optimal single player strategy through game simulation, Markov Chain Monte Carlo (MCMC) methods, and building off some of the ideas proposed by Bmhowe34. Rather than solving for the move that will result in the highest average final score at every possible board state and dice roll combination, I took a more formulaic approach that inputs any board state and dice roll combination, and outputs the best move option for maximizing the final score. Then, I looked to make improvements to this formula using simulated game scores and other data (i.e., number of rolls, penalties, locks) to compare strategies and MCMC methods to tune parameters. I thought that this would give strong insight into what aspects of the game should be prioritized when making decisions.

I also explored and aimed to improve a strategy proposed by Bmhowe34 which treats rows on the score sheet as individual Markov chains and aims to lock these rows as fast as possible. Although the goal of this strategy does not directly align with the overall single player game objective to maximize the final score, the ability to lock rows indicates that several spaces have been crossed off which is correlated with higher game scores. Therefore, I believe that this strategy is relevant to explore.

This goal to lock rows is also an important element of the final game aspect that I examined which is the comparison of strategies in multiplayer games, as locking rows limits the number of possible moves for opponents. Even though I did not go as in depth, I took a similar approach to the strategies in single player games as I retuned strategies to maximize scores and wins with opponents involved, giving insight into how a player should change their playstyle under certain conditions.
Overall, by using formulas and sets of rules to find strategies, my goal is not just to approximate optimal strategies for single and multiplayer games, but rather to find strategies that can more easily be translated to a specific playstyle when playing real life games. Even though the current claimed optimal strategy found by Bmhowe34 is impressive if true, finding the optimal move from a 2 trillion row lookup table at each turn is impractical and boring.
At this point, no formal work has been done to find an optimal strategy, nor to compare different kinds of strategies for single or multiplayer games. The only extensive work performed was by an anonymous Reddit user under the alias Bmhowe34. In 2016, they made 2 separate posts to the subreddit r/boardgames in which they discussed and linked to their work on analyzing different Qwixx strategies.

The focus of the first post was to find a strategy that would work well in multiplayer games, leading them to derive what they call the “race-to-lock” strategy, appropriately named as it aims to lock the rows as fast as possible. The primary method used to derive this strategy was absorbing Markov chains. Markov chains are stochastic processes that contain a finite number of states such that the probability of being in a particular state is dependent upon the previous state. Absorbing Markov chains are unique in that there exists a state in the chain that can be reached from all other states and once entered, cannot be left. Qwixx can be thought of as a Markov chain with the absorbing state being the end of the game. Rather than treating the entire game as a Markov chain, the “race-to-lock” strategy simplifies things by treating each row as its own Markov chain, and the absorbing state as any board state that contains 5 or more spaces crossed off, meaning it can be locked. This reduces the chain down to 26 possible states, 1 transient state which is the start of the game before any moves have been made, 1 absorbing state which is any state in which the row is lockable, and 24 other transient states represented by the space of the rightmost cross and the number of spaces crossed off in the row. To approximate the probability of moving from state to state, Bmhowe34 used the sum of 2 independent
dice rolls. After approximating these transition probabilities, matrix algebra can be applied to solve the mean number of rolls it will require to reach the absorbing state from all other board states.  

After solving for these values, Bmhowe34 observed that it was possible to move to a state with a higher mean number of rolls required to reach lockability which is most likely inefficient. To counter this, further restrictions were enforced to ensure that each transition resulted in a lower mean number of rolls required to reach lockability. This adjustment was shown to be successful as it reduced the expected number of rolls required from 37.1 to 26.2. Over 200,000 single player simulations, this strategy achieved an expected score of 74.98 points. In the same post, Bmhowe34 briefly discussed the creation of an alternative strategy which is focused more on maximizing points by both minimizing the gap between crossed out spaces and weighing in the cost of penalties. This strategy is reported to achieve an average score of around 93 points, however, verification of this score is not possible as the specific details behind the strategy are not described in depth.

The focus of the second post was on the derivation of the true optimal single player strategy for score maximization. Bmhowe34 begins the analysis by giving a little background on the game and the work required to solve the optimal strategy. It can be shown that the total number of states that the game can be found in is 73,881,680. Furthermore, the number of states that the dice can be in at each turn is 27,216, meaning that we need to find the move to maximize the expected final score for 2,010,763,802,880 different states. In a single multi-threaded C program, Bmhowe34 used dynamic programming to solve for the most optimal move by starting with game-ending states and working backwards. Since the necessary computations are incredibly time consuming,
especially when running on a single-core computer, they decided to rent time on the Amazon Web Services EC2 platform. This was able to decrease the computing time from around 530 hours (about 3 weeks) to only 16 hours. These computations found that the optimal single player strategy could achieve an average score of 115.48 points, 54% higher than the average score found using the race-to-lock strategy. Interestingly, when simulated against each other, the race-to-lock strategy beat the single player optimal strategy nearly 80% of the time, revealing that being able to score as many points as possible in single player games does not translate to wins in multiplayer games.
Chapter 3

METHODOLOGY

3.1 Code Overview

All code for game simulation, MCMC processes, data visualization, etc. was written using the R programming language. In a single R markdown document, separate functions were produced to simulate single player games, multiplayer games, and each general strategy type. All of this code can be found in either the Appendices or in a GitHub repository linked in the Bibliography. When simulating games, the functions took in as input the strategy type and some other parameters to further specify how the chosen strategy would work. These ideas will become clearer after the discussion of the different strategies. The output was the game scores, the number of rolls performed, and the board state when the game ends, including the number of locks and penalties taken. Each strategy function took in as input information about the current board state and the 45 possible moves that could be taken and output what it determined to be the best valid move possible.

3.2 No Strategy

Before attempting to code a good strategy for either single or multiplayer games, I was interested in seeing the results if one were to randomly choose a valid move at every turn, essentially taking no strategy. Relative to the other strategies, this code was simple to produce as it involved inputting the board state and possible moves from the dice rolls,
determining which moves did not break the game rules, then randomly outputting one of these moves.

3.3 Score-Based Strategy

3.3.1 Strategy Components

The first attempt that I made to approximate the single player optimal strategy was what I will be referring to as the “score-based” strategy as it produces numerical scores for the 45 different move options at each turn using a formula, then outputs the move that attains the highest score. The goal here was to produce a formula that could be easily added to and adjusted for improvement. To start building this formula, I first thought of aspects of the game that I, and others believed would be beneficial for producing higher scores, leading 3 elements to come to mind.

First, it is important to reduce the number of gaps between marked spaces in each row. For example, if a 3 has been crossed off in the red row, crossing off an 8 for the next move is probably not beneficial as we are losing 4 potential spaces that we could cross off to gain more points. This will be called the gap priority.

Second, it is important to add marks to rows where we have already crossed off spaces because each additional mark increases the number of points in that row by the new number of spaces that are marked off. For example, crossing off a space to a row that already has 2 spaces crossed off increases the row’s score by 3 points, while if the row had 6 spaces crossed off the score would increase by 7 points. This will be called the count priority.
Third, it is important to ensure that we are crossing off spaces in all 4 rows. If we were to only focus on the first 2 priorities, it is possible for games to end with many spaces not having yet been crossed off as 2 of the rows would become the focus of the scoring system once spaces start to get crossed off within them. To account for this, we should also prioritize crossing off spaces found on the left side of the board to help ensure that all rows receive attention. This will be called the position priority.

Both the count and position priorities receive scores on a scale from 0 to 10 for both sub-move within the overall move as there are 11 spaces on the board. For example, placing a cross on space 2 within the red row would result in a count score of 0 as it is the first space crossed off in the row and a position score of 10 as it is the left-most space in the row, while placing a cross on space 12 within the red row after all other spaces have been crossed off would result in a count score of 10 as it would be the eleventh space crossed off in the row and a position score of 0 as it is the right-most space in the row. The gap priority, however, receives scores on a scale from 0 to 9 for each sub-move, as that is the largest number of spaces away a crossed off space can be from another in the same row. Under this system, each of the 45 move options receives 3 scores from each sub-move for a total of 6 scores which are then added up to achieve an overall score. As previously stated, the move with the highest score is then chosen. This system can be modeled by the following equation:
\[ Y_i = W_G(X_{G1i} + X_{G2i}) + W_C(X_{C1i} + X_{C2i}) + W_P(X_{P1i} + X_{P2i}) \]

- \( Y_i \): the total score for move decision \( i \) where \( i = 1, 2, \ldots, 45 \)
- \( W_j \): the weight for score type \( j \) where \( j = G \) (Gap), \( C \) (Count), \( P \) (Position)
- \( X_{Gki} \): the gap score for sub-move \( k \) within move decision \( i \)
- \( X_{Cki} \): the count score for sub-move \( k \) within move decision \( i \)
- \( X_{Pki} \): the position score for sub-move \( k \) within move decision \( i \)

In the case of a tie, an argument in the strategy functionality could be used to choose the order of the tie breaker between 3 scores (sum of the gap scores, sum of the count scores, sum of the position scores). However, the order of these tie breakers was found to be negligible in changing the overall mean game score, so this feature was not a key focus in the analysis. Instead, separate weights were applied to the 3 score types before they were added up to see whether certain priorities were more important than others.

One major flaw with this formula so far is that all sub-moves that are skips are given a score of 0, meaning that skips only really occur if they are required. Although this may seem like a good thing as we want to avoid skips to avoid penalties, there are some instances in which skips may be beneficial. For example, if our formula determines that our best move requires us to create an 8-space gap between marked spaces in a specific row, we will be removing several spaces that have the potential to help us accumulate more points later in the game. Alternatively, we could skip this move and hope for better rolls in subsequent turns. To model this in the formula, each of the 3 scores for each sub-move involving a skip would be replaced with a single “skip score.” This skip score can then be tuned using the same tuning process of the gap, count, and position score weights.
A specific scenario where one can directly see the strategy in its current state making the wrong decision would be in the case where a penalty is chosen to be taken instead of crossing off at least 1 space, specifically after 3 penalties have already been committed. Of course, choosing to cross off a space, no matter how bad of a decision it may be, would be more beneficial to maximizing the score than ending the game. To account for this, the strategy has been specifically coded so that a fourth penalty is only taken if necessary. Building off this idea, generally, it may be beneficial to factor in the number of penalties that have been committed into whether a sub-move should be skipped, or a space should be crossed off. Later in the game when more penalties have been taken and there are less spaces left to fill on the score sheet, trying to skip less may be more appropriate. To account for this, a tunable “skip adjustment” parameter is included in the formula which decreases the skip score by a specified amount, depending on how many penalties have been committed.

So far, the described decision-making formula factors in elements related to the board state including the number of penalties committed, but it does not consider the likelihood of being able to make a specific move at a given state, something entirely dependent on the dice roll outcome. When calculating the sum of 2 independent dice rolls, there is only 1 combination that achieves a 12, while there are 6 different combinations that achieve a 7. Therefore, capitalizing on opportunities to make moves from rarer dice rolls is an element that the formula should consider. To account for this, each sub-move receives a “likelihood score” on a scale from 0 to 5 in which rolling a 2 would receive a score of 5 while a rolling 7 would receive a score of 0. This value is then multiplied by the sum of the count and gap scores of the same sub-move to ensure that we take advantage of unlikely rolls when they are most needed and skip them when they are useless. This
product is then added to the total move score. The weight of this score in the overall formula, like the gap, count, and position scores, is tunable. This updated system can be modeled by the following equation:

\[ Y_i = W_G(X_{G1i} + X_{G2i}) + W_C(X_{C1i} + X_{C2i}) + W_P(X_{P1i} + X_{P2i}) + W_L(X_{L1i} + X_{L2i}) \]

- \( Y_i \): the total score for move decision \( i \) where \( i = 1, 2, ..., 45 \)
- \( W_j \): the weight for score type \( j \) where \( j = G \) (Gap), \( C \) (Count), \( P \) (Position), \( L \) (Likelihood)
- \( X_{Gki} \): the gap score for sub-move \( k \) within move decision \( i \)
- \( X_{Cki} \): the count score for sub-move \( k \) within move decision \( i \)
- \( X_{Pki} \): the position score for sub-move \( k \) within move decision \( i \)
- \( X_{Lki} \): the likelihood score for sub-move \( k \) within move decision \( i \)

### 3.3.2 Tuning and Exploration

A combination of intuition and Markov Chain Monte Carlo (MCMC) methods were used to tune the different parameters and weights in the formula. Based on the level of importance that I believed a specific value should have in determining whether a move should be made, a specific value was chosen for the starting state, then an MCMC random walk was used to explore other options. This random walk would propose a new value for the parameter of interest, then run several game simulations and calculate the mean score for the newly proposed state. The probability that the algorithm would move from the current to the proposed state was determined by the new mean divided by the current mean. For example, if the proposed mean was 60 and the current mean was 80,
then the algorithm would move from the current to the proposed state with probability 0.75. Values greater than 1 are guaranteed to change states. The state that the value was in most frequently is then determined to be the optimal value for the parameter of interest. These parameters and weights were tuned in the order they were introduced in the prior paragraphs.

To search for other potential ways to improve the strategy, a process was set up to run simulations and keep track of the decisions made at each state that occurred. However, instead of always choosing the best move option according to the formula, the simulation would choose a move option based on the top $n$ scores, where $n$ is the number of potential moves that could be picked. The choices would be weighted so that the moves with the highest scores would still be most likely to be chosen. If a move in a simulation were repeated, then it would make the same choice as before. The goal of this was to see whether varying our move choices could produce a higher mean score than if we were to just choose the best move according to the formula every time, then try to decipher how this new strategy improved the scores based on looking through the stored simulation data. Unfortunately, none of the simulations from this set up improved the mean scores, so this approach was deemed unsuccessful.

### 3.4 Race-to-Lock Strategies

An entirely different approach to creating a strategy that performs well in Qwixx is to treat the game as an absorbing Markov chain where the transient states are different board states, and the absorbing state is the end of the game after 2 rows have been locked. Then, we can calculate how long it would take to end the game, and how we can reduce this
time to finish as fast as possible. Although the goal of this strategy has changed from maximizing the score to ending the game quickly, this strategy should still produce strong scores as being able to lock the rows ensures the accumulation of points. Also, in multiplayer games, ending the game quickly prevents opponents from being able to score. Of course, due to the vast number of board states, it would be favorable to attempt to reduce the problem to something simpler.

One way to do this would be to treat each row as its own absorbing Markov chain, an idea proposed and explored by Bmhowe34, and visualized in Figure 2. In this setup, there are 26 total states in the Markov chain. The 25 transient states are row states in which it is still possible for the row to be locked, 1 of which is an empty row, and the other 24 identifiable by the combination of the rightmost crossed off space and the number of spaces crossed off in the row. The absorbing state is any row state in which the row can be locked. This can be implemented by writing code to identify which state in the Markov chain a row is in, then identifying if it can transition to a new state closer to the absorbing state based on the roll outcome. In the case where there are multiple moves to choose from, the state change that produces the smallest gap will be chosen, like how the gap scoring system works in the score-based strategy.
However, there may be instances where transitioning from 1 row state to another may not be optimal for locking a row as quickly as possible. For example, even though crossing off space 7 for the first move in a row is technically valid since it is still possible to lock the row, it is not very strategic as there now exists only 1 move pattern to lock the row (crossing off every space after the 7) which is not very likely to achieve. Instead skipping here is more optimal. To identify which valid transitions are optimal, a transition matrix needs to be produced to determine the probability of moving to each state from another. Then, matrix algebra can be applied to calculate the mean number of moves required to reach the absorbing state. Given these means, we can further restrict move options so that
a state can only transition to another if it decreases the mean number of moves required to reach the absorbing state.

Before we can implement this process, a criterion is required to determine the probability of transitioning from state to state. In Qwixx, the sum of 2 dice out of 3 independent dice rolls (2 white and 1 colored) can be used to mark off a space in any given row, allowing for 3 options (white 1 + white 2, white 1 + colored, white 2 + colored). However, these options decrease to 2 if the sum of the 2 white dice is used in a different row, and they further decrease to 1 if the sum of 1 white die and 1 colored die are used. Therefore, we are given 3 different criteria to choose and make move restrictions from. Since the processes were all similar, I performed the analyses to produce 3 different strategies to compare and identify which restrictions led to the highest mean game scores. It is important to note that after identifying which transitions increased the number of moves required to reach lockability, the mean number of moves required were recalculated using the new transition matrices, and these often resulted in certain moves increasing the means which previously did not. Therefore, this process was iterative until only transitions that decreased the mean number of moves required were left. This was then chosen for the optimal strategy under the specified probability criterion.

This process is visualized in the 6 figures below. The gray boxes in Figure 3 state the mean number of rolls required at each row state to reach lockability, assuming the probability of transitioning from one state to another is determined by the sum of 2 independent dice rolls. However, we may not want to move from the starting state (no spaces crossed off in the row) to state 7 in column 1 (crossing off space 7 for the first move in a row) since we are increasing the mean number of rolls required to reach the lockable state from 37.1 to 46.2. After making the move restrictions to only allow for
moves that decrease the mean number of rolls, we obtain Figure 4 where the green arrows are valid moves, and the red arrows are restricted moves. Figures 5 and 7 are like Figure 3 in that they allow for all possible moves, however, the means in the gray boxes at each state are determined by the 2 other criteria previously discussed. Additionally, Figures 6 and 8 are like Figure 4 in that they restrict to only allow moves that reduce the means.

![Figure 4](image)

Figure 3 Race-to-lock absorbing Markov chain including the mean number of rolls required to reach the absorbing state based on the sum of 2 dice rolls
Figure 4 Race-to-lock absorbing Markov chain including the mean number of rolls required to reach the absorbing state and move restrictions based on the sum of 2 dice rolls.
Figure 5 Race-to-lock absorbing Markov chain including the mean number of rolls required to reach the absorbing state based on 2 sums of 2 dice from 3 dice rolls.
Figure 6 Race-to-lock absorbing Markov chain including the mean number of rolls required to reach the absorbing state and move options based on 2 sums of 2 dice from 3 dice rolls.
Figure 7 Race-to-lock absorbing Markov chain including the mean number of rolls required to reach the absorbing state based on 3 sums of 2 dice from 3 dice rolls.
3.5 Multiplayer Game Strategies

Due to lack of time, the same exploration and tuning processes performed on the strategies during single player games were not performed on multiplayer games. Instead, the versions of the strategies that were most optimal for single player games were competed against each other to measure potential changes in their score outcomes. This included both the score-based and race-to-lock strategies in games that included 2 to 5 players.
Chapter 4
RESULTS

4.1 No Strategy

As expected, randomly selecting a valid move to make after each dice roll did not turn out to be a good strategy for maximizing scores in single player games. In fact, the strategy turned out to be worse than I had expected as it had a mean score of 0.4 and a median score of -1 after 10,000 game simulations, 99% of which ended with 4 penalties. This mean estimate, and all subsequent simulated means, have a margin of error of approximately 1 point. This indicates that when playing Qwixx with no strategy, the number of points accumulated by crossing spaces off the board should roughly be cancelled out by the number of points lost due to penalties, leading me to wonder whether the game’s penalty system was chosen to achieve this by its creators. One quality that was good about this strategy is that it was far more consistent than any other that was tested as the simulated scores produced a standard deviation of around 8.4 while no other strategy had a standard deviation below 20.

Table 1 Summary statistics of scores from 10,000 simulations using “No Strategy”

<table>
<thead>
<tr>
<th>Mean</th>
<th>Median</th>
<th>Std. Dev.</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.4</td>
<td>-1.0</td>
<td>8.4</td>
<td>-17.0</td>
<td>85.0</td>
</tr>
</tbody>
</table>
4.2 Score-Based Strategy

4.2.1 Tuned Gap, Count, and Position Scores Only

Using only tuned gap, count, and position scores was a big improvement as both the mean and median scores increased to 46.1 and 43 respectively. To simplify the scoring system, I scaled each of the 3 score types from 0 to 10, meaning I only had to adjust the gap scores as they were initially scored on a scale from 0 to 9. The scores were maximized when the 3 different score types were weighted equally, which was found as I started by weighing them equally, and all other simulations with different weights produced lower mean and median game scores.
Table 2 Summary statistics of scores from 10,000 simulations using the score-based strategy applying only tuned gap, count, and position scores

<table>
<thead>
<tr>
<th>Mean</th>
<th>Median</th>
<th>Std. Dev.</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>46.1</td>
<td>43.0</td>
<td>23.8</td>
<td>-9.0</td>
<td>135.0</td>
</tr>
</tbody>
</table>

Figure 10 Histogram of scores from 10,000 simulations using the score-based strategy applying only tuned gap, count, and position scores

4.2.2 Tuned Gap, Count, Position, and Skip Scores Only

After tuning the skip score, the mean and median scores nearly doubled to around 87.3 and 86 respectively. The scores were maximized when the skip score was chosen to be 5, which makes sense since gap, count, and position scores are all on scales from 0 to 10 and have a midpoint of 5. Since adding a skip score can make it possible for the strategy to choose to take a fourth penalty when other moves are available, the code was adjusted accordingly to prevent this from happening. The same equation used in the previous section can be used to calculate the scores, but now skip scores replace the gap, count,
and position scores for sub-moves involving skips. Figure 12 shows an example decision made by this strategy. Later, we will see how this decision changes as we improve our formula.

Table 3 Summary statistics of scores from 10,000 simulations using the score-based strategy applying tuned gap, count, position and skip scores only

<table>
<thead>
<tr>
<th></th>
<th>Mean</th>
<th>Median</th>
<th>Std. Dev.</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>87.3</td>
<td>86.0</td>
<td>26.7</td>
<td>2.0</td>
<td>185.0</td>
</tr>
</tbody>
</table>

Figure 11 Histogram of scores from 10,000 simulations using the score-based strategy applying tuned gap, count, position, and skip scores only
4.2.3 Tuned Gap, Count, Position, Skip, and Likelihood Scores

After adding and tuning the likelihood score in the formula, the mean and median scores both increased slightly from before to around 90.4 and 89 respectively. These scores were maximized when each sub-move’s likelihood score was divided by around 300. Figure 14 shows the same game state as in Figure 12; however, the blue 3 is chosen to be crossed off instead of the green 5. This is because it is less likely to roll a 3, and the new formula considers this and takes advantage of the rarer roll.
Table 4  Summary statistics of scores from 10,000 simulations using the score-based strategy applying tuned gap, count, position, skip, and likelihood scores

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>Median</td>
<td>Std. Dev.</td>
<td>Minimum</td>
<td>Maximum</td>
<td></td>
</tr>
<tr>
<td>90.4</td>
<td>89.0</td>
<td>26.1</td>
<td>0.0</td>
<td>196</td>
<td></td>
</tr>
</tbody>
</table>

Figure 13  Histogram of scores from 10,000 simulations using the score-based strategy applying tuned gap, count, position, skip, and likelihood scores
4.2.4 Attempts for Improvement and Tuning

Several other unsuccessful attempts were made to increase mean scores by adjusting the formula, the most notable of which was the skip adjustment parameter. After multiple attempts with different values, it was evident that decreasing the skip score-based on the number of penalties does not improve the mean scores, and often worsened the strategy. Furthermore, using MCMC random walks to tune parameters was deemed unsuccessful. Due to the large amount of variation in game scores, the random walks also varied heavily in their output and would frequently choose poor values. Although they sometimes pointed me in the right direction to find an optimal value, the only way to get
them to function effectively was to increase the number of simulations performed at each step before the mean score was calculated. Unfortunately, that drastically increased the runtime of the simulations. Instead, it was a lot easier and more useful to tune parameters by manually choosing values that I believed could be the best, then running many simulations to get mean score estimates for each. These could then be used to either determine the best parameter value or point me in the right direction to eventually find better values.

These drawbacks are best seen in the algorithm used to tune the skip score. Here, the starting state was chosen to be 5 as the skip score can be any value between 0 and 10. At each state, 100 games are simulated and the mean score from these games is calculated. Then, a new state is proposed which is either lower or higher than the current state value by 0.1. 100 games are simulated using the proposed state value and the mean score from these games is then calculated. The probability of transitioning to the proposed state is determined by the minimum value between the proposed mean divided by the current mean or 1. 1000 different states are visited, and the state value visited the most is deemed the best. Unfortunately, as stated before, the algorithm was both computationally expensive and produced inconsistent results. Table 5 below shows differences in the output of 2 different runs from this algorithm, each of which took over 10 minutes. To make this algorithm more effective, more simulations need to be performed at each state, and/or more states need to be visited, however, this will result in even longer and less manageable runtimes. Alternatively, we can just run 10,000 simulations on a few different possible skip scores, providing us with accurate mean game score estimates for each. This simpler process quickly guided me to determine that a skip score of 5 is the best for maximizing game scores.
Table 5 Top 5 skip scores from 2 different runs of the MCMC algorithm

<table>
<thead>
<tr>
<th>Skip Score Rank</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run 1</td>
<td>4.3</td>
<td>4.2</td>
<td>5.7</td>
<td>5.8</td>
<td>4.1</td>
</tr>
<tr>
<td>Run 2</td>
<td>4.6</td>
<td>4.9</td>
<td>6.1</td>
<td>5.9</td>
<td>4.7</td>
</tr>
</tbody>
</table>

4.3 Race-to-Lock Strategy

4.3.1 No Move Restrictions

Only allowing for moves that ensure that the rows can still be locked is also an improvement from taking no strategy as the mean and median scores increased to around 54.7 and 53 respectively. However, this is still a worse approach than the score-based strategy, most likely because the strategy is getting into states where it's difficult to make moves, leading to more skips and penalties being taken. This can be seen as only around 25% of games ended after 2 rows were locked and around 33% of games ended without a single row being locked, a result that is not ideal for a strategy designed to lock rows.
Table 6 Summary statistics of scores from 10,000 simulations using the race-to-lock strategy with no move restrictions

<table>
<thead>
<tr>
<th></th>
<th>Mean</th>
<th>Median</th>
<th>Std. Dev.</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>54.7</td>
<td>53.0</td>
<td>22.8</td>
<td>-13.0</td>
<td>160.0</td>
</tr>
</tbody>
</table>

Figure 15 Histogram of scores from 10,000 simulations using the race-to-lock strategy with no restrictions

4.3.2 Move Restrictions (1 Sum)

After making restrictions to only allow for moves that reduce the number of rolls taken when 2 independent dice rolls are summed, the mean and median scores increased to around 73.8 and 77 respectively. Now, the percentage of games that ended after 2 rows were locked increased to around 37% and the percentage of games that ended without a single row being locked decreased to around 29%, an improvement from the previous strategy which can partially explain the score increases.
Table 7 Summary statistics of scores from 10,000 simulations using the race-to-lock strategy with move restrictions based on the sum of 2 dice rolls

<table>
<thead>
<tr>
<th>Mean</th>
<th>Median</th>
<th>Std. Dev.</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>73.8</td>
<td>77.0</td>
<td>28.6</td>
<td>-20.0</td>
<td>163.0</td>
</tr>
</tbody>
</table>

Figure 16 Histogram of scores from 10,000 simulations using the race-to-lock strategy with move restrictions based on the sum of 2 dice rolls

4.3.3 Move Restrictions (2 Sums)

After making restrictions to only allow for moves that reduce the number of rolls needed to lock a row when we assume that we have 2 different 2-dice sums from 3 independent dice rolls to choose from, the mean and median scores increased to around 76.6 and 78 respectively. Again, part of this improvement can be explained by the fact that the percentage of games that ended after 2 rows were locked increased to around 40% and the percentage of games that ended without a single row being locked decreased to
around 24%. This indicates that making decisions based on 2 of the possible sums from 3 independent rolls better resembles most in game decision making processes.

Table 8 Summary statistics of scores from 10,000 simulations using the race-to-lock strategy with move restrictions based on 2 different sums of 2 of the dice from 3 dice rolls

<table>
<thead>
<tr>
<th>Mean</th>
<th>Median</th>
<th>Std. Dev.</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>76.6</td>
<td>78.0</td>
<td>25.3</td>
<td>-12.0</td>
<td>167.0</td>
</tr>
</tbody>
</table>

Figure 17 Histogram of scores from 10,000 simulations using the race-to-lock strategy with move restrictions based on 2 different sums of 2 of the dice from 3 dice rolls
4.3.4 Move Restrictions (3 Sums)
After making restrictions to only allow for moves that reduce the number of rolls needed to lock a row when we assume that we have all 3 different 2-dice sums from 3 independent dice rolls to choose from, the mean and median scores decreased slightly to around 74.8 and 75 respectively. Interestingly, the percentage of games that ended after 2 rows were locked increased and the percentage of games that ended without a single row being locked stayed the same as the previous strategy at around 40% and 24% respectively, indicating that some other factor is driving the scores down since its locking ability is the same.

Table 9 Summary statistics of scores from 10,000 simulations using the race-to-lock strategy with move restrictions based on 3 different sums of 2 of the dice from 3 dice rolls

<table>
<thead>
<tr>
<th>Mean</th>
<th>Median</th>
<th>Std. Dev.</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>74.8</td>
<td>75.0</td>
<td>24.3</td>
<td>-12.0</td>
<td>157.0</td>
</tr>
</tbody>
</table>
4.4 Multiplayer Games

4.4.1 Score-Based vs. Score-Based

As expected, when 2 players using the optimal score-based strategy face each other, the game is even as both win around 49% of the time with the remaining 2% being the result of ties. Also, the mean and median scores decrease from the scores seen in single player games to around 83 and 84 respectively. This most likely occurs because the probability of a player locking 2 rows or committing 4 penalties earlier in the game increases since there are now 2 players.
Table 10 Summary statistics of scores from 10,000 simulations of 2 player games where both players use the optimal score-based strategy

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Mean</th>
<th>Median</th>
<th>Std. Dev.</th>
<th>Min.</th>
<th>Max.</th>
<th>Win %</th>
</tr>
</thead>
<tbody>
<tr>
<td>S.B.</td>
<td>83.4</td>
<td>84.0</td>
<td>24.3</td>
<td>-3.0</td>
<td>194</td>
<td>49.9%</td>
</tr>
<tr>
<td>S.B.</td>
<td>83.0</td>
<td>83.0</td>
<td>24.2</td>
<td>2.0</td>
<td>171.0</td>
<td>48.7%</td>
</tr>
</tbody>
</table>

Figure 19 Histogram of scores from 10,000 simulations of 1 player from 2 player games where both players use the optimal score-based strategy

4.4.2 Race-to-Lock vs. Race-to-Lock

Like the results from the score-based strategy, when 2 players using the optimal race-to-finish strategy face each other, the game is even as both win around 49% of the time with the remaining 2% being the result of ties. Again, the mean and median scores both decrease from the scores seen in single player games to around 70, most likely due to the increase in the number of players.
Table 11 Summary statistics of scores from 10,000 simulations of 2 player games where both players use the optimal race-to-lock strategy

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Mean</th>
<th>Median</th>
<th>Std. Dev.</th>
<th>Min.</th>
<th>Max.</th>
<th>Win %</th>
</tr>
</thead>
<tbody>
<tr>
<td>R.T.L.</td>
<td>69.7</td>
<td>70.0</td>
<td>21.5</td>
<td>-10.0</td>
<td>154.0</td>
<td>48.8%</td>
</tr>
<tr>
<td>R.T.L.</td>
<td>69.6</td>
<td>70.0</td>
<td>21.6</td>
<td>-11.0</td>
<td>150.0</td>
<td>49.6%</td>
</tr>
</tbody>
</table>

Figure 20 Histogram of scores from 10,000 simulations of 1 player from 2 player games where both players use the optimal race-to-lock strategy

4.4.3 Score-Based vs. Race-to-Lock

Interestingly, when the optimal score-based strategy competes against the optimal race-to-lock strategy, the race-to-lock strategy wins around 53% of the time while the optimal score-based strategy only wins around 45% of the time. This is despite the race-to-lock strategy being a worse strategy for optimizing scores in single player games. This was somewhat expected since previous work showed that a version of the race-to-lock strategy won most games when playing against the claimed optimal strategy for single
player score maximization. This most likely occurs since the race-to-lock strategy has a lower mean number of rolls per game at around 18, while the score-based strategy has a mean number of rolls per game at around 22. Therefore, the race-to-lock strategy ends the game before the score-based strategy adds finishing points to its score.

Table 12 Summary statistics of scores from 10,000 simulations of 2 player games where the optimal score-based strategy competes against the optimal race-to-lock strategy

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Mean</th>
<th>Median</th>
<th>Std. Dev.</th>
<th>Min.</th>
<th>Max.</th>
<th>Win %</th>
</tr>
</thead>
<tbody>
<tr>
<td>S.B.</td>
<td>72.8</td>
<td>73.0</td>
<td>24.2</td>
<td>-2.0</td>
<td>156.0</td>
<td>45.2%</td>
</tr>
<tr>
<td>R.T.L.</td>
<td>76.1</td>
<td>77.0</td>
<td>21.3</td>
<td>-13.0</td>
<td>159.0</td>
<td>53.4%</td>
</tr>
</tbody>
</table>

Figure 21 Density ridge plot of scores from 10,000 simulations of 2 player games where the optimal score-based strategy competes against the optimal race-to-lock strategy
Figure 22 Scatterplot of scores from 10,000 simulations of 2 player games where the both players use the optimal race-to-lock strategy
Chapter 5
CONCLUSION

5.1 Summary

Approximating optimal strategies and testing others for both single and multiplayer Qwixx games not only revealed what kinds of moves exist in good strategies and how those vary between game styles, but also how effective the process to approximate these strategies is.

For single player games, the closest approximation achieved to maximize game scores involved considering equally weighted gap, count, and position scores on a scale from 1 to 10 each, skip scores equal to 5 that remain consistent throughout the game, and likelihood scores with a weight 1/300 relative to the weight of the other scores. This produced simulations that achieved mean and median scores around 90 and 89 respectively. Using MCMC random walk methods to tune the weight of the scores in the equations was not found to be super useful, but tuning was still possible by manually testing out values and using results from many simulations for each.

When compared to previous work, this strategy and the process used to identify it appear to be poor because the true optimal strategy achieved mean scores around 25 points higher and a previous approximation was claimed to achieve mean scores somewhere between 90 and 95. However, it is difficult to conclusively compare results as the previous work was never officially verified.

Out of the 4 potential versions of the race-to-finish strategy, the third option that assumes that we can only make moves from 2 different 2-dice sums from 3 independent dice rolls
was found to be the most effective in single player games as it achieved mean and median scores around 76.6 and 78 respectively. This was also found to be the most effective strategy for multiplayer games between the 2 main strategies tested as it won around 53% of games against the most optimal score-based strategy.

5.2 Informing Gameplay

The results from both the single and multiplayer game simulations give us some insights into how players should approach both game styles, and which types of moves should be prioritized in varying situations.

In terms of single player game score maximization, minimizing gaps between marked off spaces, marking off spaces in rows that contains many marks, and utilizing spaces from all 4 rows on the score sheet was all confirmed to be important. More notably, none of the qualities were deemed more important than the others, so players should give them equal amounts of attention in their decision-making process. Next, if only faced with sub-moves that appear to be poor decisions, deciding whether to skip can be done without intense amounts of effort by calculating the gap, count, and position scores for what appears to be the best available move. If the sum of these together is not approximately greater than 15, then skipping may be the appropriate decision. This skipping policy should not change unless 3 penalties have already been taken. Last, if there are more than 2 available sub-moves that all look equally beneficial, choosing the sub-moves that mark off spaces furthest from the middle of the score sheet (furthest from space 7) may be a good way to narrow the decision as these sub-moves are produced from rolls that are less likely to occur again later on in the game.
When making decisions in multiplayer games, restricting moves to only those permitted by the race-to-lock strategy that assumes that we can only make moves from 2 different 2-dice sums from 3 independent dice rolls is the best course of action, as this should achieve a decently high game score early on in the game and allows one to lock rows before opponents have the chance to accumulate points. This requires little thought as the visualization for this strategy can simply be used as a cheat sheet.
Although the process used for approximating an optimal strategy for single player score maximization did find ways to make meaningful improvements in mean game scores, the approach taken can still be improved. The goal with this work was to create a near-optimal strategy that could be translated to an easy-to-adopt playstyle. The reason the chosen approach was taken was because I did not have the computational power to replicate and verify the work performed by Bmhowe34. Although the claimed optimal single player strategy found by Bmhowe34 is not in an easy-to-adopt playstyle in its current form, attaining the data on every optimal move would be incredibly helpful. In my current approach, there is some guessing involved to identify which kinds of moves will help make improvements. Sometimes these are useful as seen by the success of adding the likelihood score while other times these are not as seen by the skip score adjustment. Generally, a better approach would have been to find the true optimal strategy, then use it to inform an implementable strategy and playstyle.

In addition to the claimed optimal strategy, there were a few other processes and results from Bmhowe34’s work that would have been helpful to have had more information on. First, when applying the move restrictions to the race-to-lock strategy to only allow for moves that reduce the mean number of rolls to reach the absorbing state, an algorithm was implemented that achieved different results from the process I took. It was also unclear how these restrictions were implemented into a specific, playable strategy. Furthermore, an alternative strategy to approximate the optimal solution was described in their first post which they claimed achieved an average score of around 93 points,
however, they were again unclear about the rules of this strategy preventing me from being able to implement it and tests its validity. It would have been interesting to see if I could combine my ideas with the ideas from this strategy and find an even better approximation.

Finally, the approach I took still ended up running into some computational issues. The MCMC random walks used to tune parameters for the score-based strategy were not efficient in their ability to find weights and values that maximized the mean. Since the variances of the game simulations for each strategy were so large, many simulations were required to find meaningful differences in the means between strategies that used slightly different rules. For example, most “good” score-based strategies had standard deviations around 25 points, meaning that 10,000 simulations would be required to achieve mean estimates with an error of around 1 point. Unfortunately, the code that I wrote takes over a minute to run that many simulations for most score-based strategies, and this would be required at each newly visited state in the walk. Therefore, the runtime for the random walks would need to be several thousand minutes. Although the same tuning process was not performed on the race-to-lock strategies, the runtime for those simulations was even longer, discouraging the potential use of random walks to find new sets of move restriction that may have further increased mean scores.

One pattern I observed from the simulation results was that as mean and median scores increased, the standard deviations generally increased as well. However, there was a point where the standard deviation appeared to reach its maximum as further increases in the mean either did not change the standard deviation or decreased it slightly. In the cases where there was a decrease, histograms revealed the development of a right skew as scores close to and below 0 began to disappear, leading me to believe that the true
optimal strategy is probably able to adapt to unlucky rolls and still find ways to accumulate some points in bad games. Therefore, I believe that one element that could be added to the score-based formula to increase mean scores may be the current game score at each move, relative to the number of current rolls and penalties.

Another possible way these issues could be solved would be to have a general formula, then change to a new formula depending on how unlucky the start of the game is. Essentially, this is an ensemble strategy that applies different formulas depending on the luck of the rolls. To tune a strategy designed for poor game starts, a randomly generated poor board state could be produced at the start of the game, then the same process used to tune the initial score-based strategy could be applied until these scores are maximized. The results from this may also be used to inform new ways to better tune the original general formula.
BIBLIOGRAPHY


APPENDICES

Appendix A. Helper Functions

# calculates score given the number of marks in row
row_score <- function(row) {
  if (row == 0) return(0)
  return(row + row_score(row - 1))
}

# converts green/blue row positions to red/yellow
rpc <- function(row_position) {
  return(row_position - (10 - (12 - row_position) * 2))
}

# calculates gap score for strategy 1
gsc <- function(score) {
  return(score - (11 - (10 - score) * 2))
}

# calculates position score for strategy 1
psc <- function(score) {
  return(score - (10 - (10 - score) * 2))
}

# calculates gap score for strategy 2
gsc2 <- function(score) {
  return(score - (5 - (6 - score) * 2))
}
Appendix B. Code for Strategy 0 (No Strategy) Function

```r
strategy0 <- function (row_r, row_y, row_g, row_b, decisions, turn = TRUE) {
  skip_score <- 1
  gap_scores1 <- rep(NA, 45)
  gap_scores2 <- rep(NA, 45)
  position_scores1 <- rep(NA, 45)
  position_scores2 <- rep(NA, 45)
  count_scores1 <- rep(NA, 45)
  count_scores2 <- rep(NA, 45)

  if (!turn) {
    decisions$roll2 = rep(NA, 45)
    decisions$roll_color2 = rep('skip', 45)
  }

  for (i in 1:45) {
    if (decisions$roll_color1[i] == 'skip') {
      gap_scores1[i] <- skip_score
      position_scores1[i] <- skip_score
      count_scores1[i] <- skip_score
    } else if (decisions$roll_color1[i] == 'red') {
      gap_scores1[i] <- gsc(decisions$roll1[i] - row_r[1] - 1)
      position_scores1[i] <- psc(decisions$roll1[i] - 2)
      count_scores1[i] <- row_r[2]
    } else if (decisions$roll_color1[i] == 'yellow') {
      gap_scores1[i] <- gsc(decisions$roll1[i] - row_y[1] - 1)
      position_scores1[i] <- psc(decisions$roll1[i] - 2)
      count_scores1[i] <- row_y[2]
    } else if (decisions$roll_color1[i] == 'green') {
      gap_scores1[i] <- gsc(row_g[1] - decisions$roll1[i] - 1)
      position_scores1[i] <- psc(rpc(decisions$roll1[i]) - 2)
      count_scores1[i] <- row_g[2]
    } else if (decisions$roll_color1[i] == 'blue') {
      gap_scores1[i] <- gsc(row_b[1] - decisions$roll1[i] - 1)
      position_scores1[i] <- psc(rpc(decisions$roll1[i]) - 2)
      count_scores1[i] <- row_b[2]
    }
  }
}
```

if (decisions$roll_color2[i] == 'skip') {
    gap_scores2[i] <- skip_score * 9/10
    position_scores2[i] <- skip_score
    count_scores2[i] <- skip_score
}
else if (decisions$roll_color2[i] == 'red') {
    if (decisions$roll_color1[i] == 'red') {
        gap_scores2[i] <- gsc(decisions$roll2[i] - decisions$roll1[i] - 1)
        position_scores2[i] <- psc(decisions$roll2[i] - 2)
        count_scores2[i] <- row_r[2] + 1
    }
    else {
        gap_scores2[i] <- gsc(decisions$roll2[i] - row_r[1] - 1)
        position_scores2[i] <- psc(decisions$roll2[i] - 2)
        count_scores2[i] <- row_r[2]
    }
}
else if (decisions$roll_color2[i] == 'yellow') {
    if (decisions$roll_color1[i] == 'yellow') {
        gap_scores2[i] <- gsc(decisions$roll2[i] - decisions$roll1[i] - 1)
        position_scores2[i] <- psc(decisions$roll2[i] - 2)
        count_scores2[i] <- row_y[2] + 1
    }
    else {
        gap_scores2[i] <- gsc(decisions$roll2[i] - row_y[1] - 1)
        position_scores2[i] <- psc(decisions$roll2[i] - 2)
        count_scores2[i] <- row_y[2]
    }
}
else if (decisions$roll_color2[i] == 'green') {
    if (decisions$roll_color1[i] == 'green') {
        gap_scores2[i] <- gsc(decisions$roll1[i] - decisions$roll2[i] - 1)
        position_scores2[i] <- psc(rpc(decisions$roll2[i]) - 2)
        count_scores2[i] <- row_g[2] + 1
    }
    else {
        gap_scores2[i] <- gsc(row_g[1] - decisions$roll2[i] - 1)
        position_scores2[i] <- psc(rpc(decisions$roll2[i]) - 2)
        count_scores2[i] <- row_g[2]
    }
}
} 
else if (decisions$roll_color2[i] == 'blue') {
    if (decisions$roll_color1[i] == 'blue') {
        gap_scores2[i] <- gsc(decisions$roll1[i] - decisions$roll2[i] - 1)
        position_scores2[i] <- psc(rpc(decisions$roll2[i]) - 2)
        count_scores2[i] <- row_b[2] + 1
    }
    else {
        gap_scores2[i] <- gsc(row_b[1] - decisions$roll2[i] - 1)
        position_scores2[i] <- psc(rpc(decisions$roll2[i]) - 2)
        count_scores2[i] <- row_b[2]
    }
}

if (decisions$roll_color1[i] != 'skip') {
    if (!is.na(gap_scores1[i]) & (gap_scores1[i] < 0 | gap_scores1[i] >= 10))
        gap_scores1[i] <- NA
    if (!is.na(position_scores1[i]) & ((position_scores1[i] < 0 | position_scores1[i] >= 11) |
        (position_scores1[i] == 0 & count_scores1[i] < 5)))
        position_scores1[i] <- NA
    if (!is.na(count_scores1[i]) & (count_scores1[i] < 0 | count_scores1[i] >= 11))
        count_scores1[i] <- NA
}

if (decisions$roll_color2[i] != 'skip') {
    if (!is.na(gap_scores2[i]) & (gap_scores2[i] < 0 | gap_scores2[i] >= 10))
        gap_scores2[i] <- NA
    if (!is.na(position_scores2[i]) & ((position_scores2[i] < 0 | position_scores2[i] >= 11) |
        (position_scores2[i] == 0 & count_scores2[i] < 5)))
        position_scores2[i] <- NA
    if (!is.na(count_scores2[i]) & (count_scores2[i] < 0 | count_scores2[i] >= 11))
        count_scores2[i] <- NA
}

# calculating scores
gap_scores <- gap_scores1 + gap_scores2
position_scores <- position_scores1 + position_scores2
count_scores <- count_scores1 + count_scores2
total_scores <- gap_scores + position_scores + count_scores
# adjusting worst and best scores
total_scores[!is.na(total_scores)] <- 1
total_scores[is.na(total_scores)] <- 0

# finding score weights
weights <- total_scores / sum(total_scores)

# finding random eligible score index
score_index <- sample(1:45, 1, prob=weights)

return(decisions[score_index,])
Appendix C. Code for Strategy 1 (Score-Based Strategy) Function

# Priorities:
# 1) Gap: smallest number of spaces between proposed move(s) and last chosen space
# 2) Position: furthest left move(s)
# 3) Count: row(s) with the most spaces crossed off
# Skip score: Determines how many points a skip receives for its gap, position, and count scores
# Skip adjustment: Determines how much the skip score decreases for each penalty taken
# Weight: Determines how much the probability score is weighted

strategy1 <- function (row_r, row_y, row_g, row_b, decisions, priority_order = '123',
                      skip_score = 5, skip_adjustment = 0, weight = 1/300, penalty,
                      turn = TRUE) {
    skip_score <- skip_score + skip_adjustment*penalty
    gap_scores1 <- rep(NA, 45)
    gap_scores2 <- rep(NA, 45)
    position_scores1 <- rep(NA, 45)
    position_scores2 <- rep(NA, 45)
    count_scores1 <- rep(NA, 45)
    count_scores2 <- rep(NA, 45)
    prob_scores1 <- rep(0, 45)
    prob_scores2 <- rep(0, 45)

    if (!turn) {
        decisions$roll2 = rep(NA, 45)
        decisions$roll_color2 = rep('skip', 45)
    }

    for (i in 1:45) {
        if (decisions$roll_color1[i] == 'skip') {
            gap_scores1[i] <- skip_score * 9/10
            position_scores1[i] <- skip_score
            count_scores1[i] <- skip_score
        } else if (decisions$roll_color1[i] == 'red') {
            gap_scores1[i] <- gsc(decisions$roll1[i] - row_r[1] - 1)
            position_scores1[i] <- psc(decisions$roll1[i] - 2)
            count_scores1[i] <- row_r[2]
        } else if (decisions$roll_color1[i] == 'yellow') {
            gap_scores1[i] <- gsc(decisions$roll1[i] - row_y[1] - 1)
        }
    }

75
position_scores1[i] <- psc(decisions$roll1[i] - 2)
count_scores1[i] <- row_y[2]
}
else if (decisions$roll_color1[i] == 'green') {
  gap_scores1[i] <- gsc(row_g[1] - decisions$roll1[i] - 1)
  position_scores1[i] <- psc(rpc(decisions$roll1[i]) - 2)
  count_scores1[i] <- row_g[2]
}
else if (decisions$roll_color1[i] == 'blue') {
  gap_scores1[i] <- gsc(row_b[1] - decisions$roll1[i] - 1)
  position_scores1[i] <- psc(rpc(decisions$roll1[i]) - 2)
  count_scores1[i] <- row_b[2]
}

if (decisions$roll_color2[i] == 'skip') {
  gap_scores2[i] <- skip_score * 9/10
  position_scores2[i] <- skip_score
  count_scores2[i] <- skip_score
}
else if (decisions$roll_color2[i] == 'red') {
  if (decisions$roll_color1[i] == 'red') {
    gap_scores2[i] <- gsc(decisions$roll2[i] - decisions$roll1[i] - 1)
    position_scores2[i] <- psc(decisions$roll2[i] - 2)
    count_scores2[i] <- row_r[2] + 1
  }
  else {
    gap_scores2[i] <- gsc(decisions$roll2[i] - row_r[1] - 1)
    position_scores2[i] <- psc(decisions$roll2[i] - 2)
    count_scores2[i] <- row_r[2]
  }
}
else if (decisions$roll_color2[i] == 'yellow') {
  if (decisions$roll_color1[i] == 'yellow') {
    gap_scores2[i] <- gsc(decisions$roll2[i] - decisions$roll1[i] - 1)
    position_scores2[i] <- psc(decisions$roll2[i] - 2)
    count_scores2[i] <- row_y[2] + 1
  }
  else {
    gap_scores2[i] <- gsc(decisions$roll2[i] - row_y[1] - 1)
    position_scores2[i] <- psc(decisions$roll2[i] - 2)
    count_scores2[i] <- row_y[2]
  }
}
else if (decisions$roll_color2[i] == 'green') {
    if (decisions$roll_color1[i] == 'green') {
        gap_scores2[i] <- gsc(decisions$roll1[i] - decisions$roll2[i] - 1)
        position_scores2[i] <- psc(rpc(decisions$roll2[i]) - 2)
        count_scores2[i] <- row_g[2] + 1
    } else {
        gap_scores2[i] <- gsc(row_g[1] - decisions$roll2[i] - 1)
        position_scores2[i] <- psc(rpc(decisions$roll2[i]) - 2)
        count_scores2[i] <- row_g[2]
    }
} else if (decisions$roll_color2[i] == 'blue') {
    if (decisions$roll_color1[i] == 'blue') {
        gap_scores2[i] <- gsc(decisions$roll1[i] - decisions$roll2[i] - 1)
        position_scores2[i] <- psc(rpc(decisions$roll2[i]) - 2)
        count_scores2[i] <- row_b[2] + 1
    } else {
        gap_scores2[i] <- gsc(row_b[1] - decisions$roll2[i] - 1)
        position_scores2[i] <- psc(rpc(decisions$roll2[i]) - 2)
        count_scores2[i] <- row_b[2]
    }
}

if (decisions$roll_color1[i] != 'skip') {
    if (!is.na(gap_scores1[i]) & (gap_scores1[i] < 0 | gap_scores1[i] >= 10))
        gap_scores1[i] <- NA
    if (!is.na(position_scores1[i]) &
        ((position_scores1[i] < 0 | position_scores1[i] >= 11) |
        (position_scores1[i] == 0 & count_scores1[i] < 5))
        position_scores1[i] <- NA
    if (!is.na(count_scores1[i]) & (count_scores1[i] < 0 | count_scores1[i] >= 11))
        count_scores1[i] <- NA
}

if (decisions$roll_color2[i] != 'skip') {
    if (!is.na(gap_scores2[i]) & (gap_scores2[i] < 0 | gap_scores2[i] >= 10))
        gap_scores2[i] <- NA
    if (!is.na(position_scores2[i]) &
        ((position_scores2[i] < 0 | position_scores2[i] >= 11) |}
(position_scores2[i] == 0 & count_scores2[i] < 5))
position_scores2[i] <- NA
if (!is.na(count_scores2[i]) & (count_scores2[i] < 0 | count_scores2[i] >= 11))
count_scores2[i] <- NA
}

# calculate probability scores
if (decisions$roll_color1[i] != 'skip')
  prob_scores1[i] <- (abs(position_scores1[i] - 5)) * (gap_scores1[i] + count_scores1[i])
if (decisions$roll_color2[i] != 'skip')
  prob_scores2[i] <- (abs(position_scores2[i] - 5)) * (gap_scores2[i] + count_scores2[i])

# only skip if necessary with 3 penalties
if (decisions$roll_color1[i] == 'skip' & decisions$roll_color2[i] == 'skip' & penalty == 3) {
  gap_scores1[i] <- 0
  gap_scores2[i] <- 0
  position_scores1[i] <- 0
  position_scores2[i] <- 0
  count_scores1[i] <- 0
  count_scores2[i] <- 0
}

# calculating scores
gap_scores <- (gap_scores1 + gap_scores2) * 10/9
position_scores <- (position_scores1 + position_scores2)
count_scores <- (count_scores1 + count_scores2)
prob_scores <- (prob_scores1 + prob_scores2) * weight

total_scores <- gap_scores + position_scores + count_scores + prob_scores

best_score_index <- which(total_scores == max(na.omit(total_scores)))

# priority options
if (priority_order == '123') {
  if (length(best_score_index) > 1) {
    gap_scores[-best_score_index] <- NA
    best_score_index <- which(gap_scores == max(na.omit(gap_scores)))
  }
}
best_score_index <- which(position_scores == max(na.omit(position_scores)))
if (length(best_score_index) > 1) {
  count_scores[-best_score_index] <- NA
  best_score_index <- which(count_scores == max(na.omit(count_scores)))
  if (length(best_score_index) > 1) best_score_index <- sample(best_score_index, 1)
}
}
}

else if (priority_order == '132') {
  if (length(best_score_index) > 1) {
    gap_scores[-best_score_index] <- NA
    best_score_index <- which(gap_scores == max(na.omit(gap_scores)))
    if (length(best_score_index) > 1) {
      count_scores[-best_score_index] <- NA
      best_score_index <- which(count_scores == max(na.omit(count_scores)))
      if (length(best_score_index) > 1) best_score_index <- sample(best_score_index, 1)
    }
  }
}
}

else if (priority_order == '213') {
  if (length(best_score_index) > 1) {
    position_scores[-best_score_index] <- NA
    best_score_index <- which(position_scores == max(na.omit(position_scores)))
    if (length(best_score_index) > 1) {
      gap_scores[-best_score_index] <- NA
      best_score_index <- which(gap_scores == max(na.omit(gap_scores)))
      if (length(best_score_index) > 1) best_score_index <- sample(best_score_index, 1)
    }
  }
}
}
else if (priority_order == '231') {
    if (length(best_score_index) > 1) {
        position_scores[-best_score_index] <- NA
        best_score_index <- which(position_scores == max(na.omit(position_scores)))
    }
}

else if (priority_order == '312') {
    if (length(best_score_index) > 1) {
        count_scores[-best_score_index] <- NA
        best_score_index <- which(count_scores == max(na.omit(count_scores)))
    }
}

else if (priority_order == '321') {
    if (length(best_score_index) > 1) {
        gap_scores[-best_score_index] <- NA
        best_score_index <- which(gap_scores == max(na.omit(gap_scores)))
    }
}
if (length(best_score_index) > 1) best_score_index <- sample(best_score_index, 1)
}
}
}

return(decisions[best_score_index,])
Appendix D. Code for Strategy 2 (Race-to-Lock Strategy) Function

# Move Options:
# 1) Move to any valid state
# 2) Only move to states that decrease number of rolls needed to lock a row based on 1 sum
# 3) Only move to states that decrease number of rolls needed to lock a row based on 2 sums
# 4) Only move to states that decrease number of rolls needed to lock a row based on 3 sums

strategy2 <- function (row_r, row_y, row_g, row_b, decisions, move_option = 3, turn = TRUE) {
  states <- list(A = c(1, 0), B = c(2, 1), C = c(3, 1), D = c(4, 1), E = c(5, 1),
                 F = c(6, 1), G = c(7, 1), H = c(3, 2), I = c(4, 2), J = c(5, 2),
                 K = c(6, 2), L = c(7, 2), M = c(8, 2), N = c(4, 3), O = c(5, 3),
                 P = c(6, 3), Q = c(7, 3), R = c(8, 3), S = c(9, 3), T = c(5, 4),
                 U = c(6, 4), V = c(7, 4), W = c(8, 4), X = c(9, 4), Y = c(10, 4))

  if (move_option == 1) {
    moves <- list(A = c('B', 'C', 'D', 'E', 'F', 'G'),
                  B = c('H', 'T', 'J', 'K', 'L', 'M'),
                  C = c('T', 'J', 'K', 'L', 'M'),
                  D = c('J', 'K', 'L', 'M'),
                  E = c('K', 'L', 'M'),
                  F = c('L', 'M'),
                  G = c('M'),
                  H = c('N', 'O', 'P', 'Q', 'R', 'S'),
                  I = c('P', 'Q', 'R', 'S'),
                  J = c('Q', 'R', 'S'),
                  K = c('R', 'S'),
                  L = c('S'),
                  M = c('S'),
                  N = c('U', 'V', 'W', 'X', 'Y'),
                  O = c('V', 'W', 'X', 'Y'),
                  P = c('V', 'W', 'X', 'Y'),
                  Q = c('W', 'X', 'Y'),
                  R = c('X', 'Y'),
                  S = c('Y'),
                  T = c('Z'),
                  U = c('Z'),
                  V = c('Z'),
                  W = c('Z'),
                  X = c('Z'),
                  Y = c('Z'),
                  Z = c('Z'))
  }
  else if (move_option == 2) {
    moves <- list(A = c('B', 'C', 'D'),
                  B = c('H', 'T', 'J'),
                  C = c('T', 'J', 'K'),
                  D = c('J', 'K'),
                  E = c('K'),
                  F = c('L'),
                  G = c('M'),
                  H = c('N', 'O', 'P'),
                  I = c('O', 'P', 'Q'),
                  J = c('P', 'Q'),
                  K = c('Q'),
                  L = c('R'),
                  M = c('S'),
                  N = c('T', 'U', 'V', 'W', 'X', 'Y'),
                  O = c('U', 'V', 'W', 'X', 'Y'),
                  P = c('V', 'W', 'X', 'Y'),
                  Q = c('W', 'X', 'Y'),
                  R = c('X', 'Y'),
                  S = c('Y'),
                  T = c('Z'),
                  U = c('Z'),
                  V = c('Z'),
                  W = c('Z'),
                  X = c('Z'),
                  Y = c('Z'),
                  Z = c('Z'))
  }
}

62
else if (move_option == 3) {
    moves <- list(A = c('B', 'C', 'D', 'E'), B = c('H', 'I', 'J'), C = c('I', 'J', 'K'),
                  D = c('J', 'K'), E = c('K'), F = c('L'), G = c('M'),
                  H = c('N', 'O', 'P'), I = c('O', 'P', 'Q'), J = c('P', 'Q'),
                  K = c('Q'), L = c('R'), M = c('S'),
                  N = c('T', 'U', 'V', 'W'), O = c('U', 'V', 'W'), P = c('V', 'W'),
                  Q = c('W', 'X'), R = c('X'), S = c('Y'), T = c('Z'), U = c('Z'),
                  V = c('Z'), W = c('Z'), X = c('Z'), Y = c('Z'), Z = c('Z'))
}
else if (move_option == 4) {
    moves <- list(A = c('B', 'C', 'D', 'E'), B = c('H', 'I', 'J', 'K'),
                  C = c('I', 'J', 'K'), D = c('J', 'K'), E = c('K'), F = c('L'), G = c('M'),
                  H = c('N', 'O', 'P', 'Q'), I = c('O', 'P', 'Q'), J = c('P', 'Q'),
                  K = c('Q'), L = c('R'), M = c('S'),
                  N = c('T', 'U', 'V', 'W'), O = c('U', 'V', 'W'), P = c('V', 'W', 'X'),
                  Q = c('W', 'X'), R = c('X'), S = c('Y'), T = c('Z'), U = c('Z'),
                  V = c('Z'), W = c('Z'), X = c('Z'), Y = c('Z'), Z = c('Z'))
}

scores1 <- rep(NA, 45)
scores2 <- rep(NA, 45)
scores <- rep(NA, 45)

positions1 <- rep(NA, 45)
positions2 <- rep(NA, 45)
counts1 <- rep(NA, 45)
counts2 <- rep(NA, 45)

if (!turn) {
    decisions$roll2 = rep(NA, 45)
    decisions$roll_color2 = rep('skip', 45)
}

state_r <- 'Z'
state_y <- 'Z'
state_g <- 'Z'
state_b <- 'Z'

for (i in 1:25) {
    if (sum(row_r == states[[i]]) == 2) state_r <- names(states)[i]
    if (sum(row_y == states[[i]]) == 2) state_y <- names(states)[i]
if (sum(c(rpc(row_g[1]), row_g[2]) == states[[i]]) == 2) state_g <- names(states)[i]
if (sum(c(rpc(row_b[1]), row_b[2]) == states[[i]]) == 2) state_b <- names(states)[i]
}

for (i in 1:45) {
  if (decisions$roll_color1[i] == 'skip') {
    scores1[i] <- 0
  }
  else if (decisions$roll_color1[i] == 'red') {
    state1 <- 'Z'
    for (j in 1:25) {
      if (sum(c(decisions$roll1[i], row_r[2] + 1) == states[[j]]) == 2)
        state1 <- names(states)[j]
    }
    if (state1 %in% moves[[state_r]]) {
      scores1[i] <- gsc2(decisions$roll1[i] - row_r[1])
      positions1[i] <- decisions$roll1[i]
      counts1[i] <- row_r[2] + 1
    }
  }
  else if (decisions$roll_color1[i] == 'yellow') {
    state1 <- 'Z'
    for (j in 1:25) {
      if (sum(c(decisions$roll1[i], row_y[2] + 1) == states[[j]]) == 2)
        state1 <- names(states)[j]
    }
    if (state1 %in% moves[[state_y]]) {
      scores1[i] <- gsc2(decisions$roll1[i] - row_y[1])
      positions1[i] <- decisions$roll1[i]
      counts1[i] <- row_y[2] + 1
    }
  }
  else if (decisions$roll_color1[i] == 'green') {
    state1 <- 'Z'
    for (j in 1:25) {
      if (sum(c(rpc(decisions$roll1[i]), row_g[2] + 1) == states[[j]]) == 2)
        state1 <- names(states)[j]
    }
    if (state1 %in% moves[[state_g]]) {
      scores1[i] <- gsc2(row_g[1] - decisions$roll1[i])
      positions1[i] <- rpc(decisions$roll1[i])
      counts1[i] <- row_g[2] + 1
    }
  }
}
else if (decisions$roll_color1[i] == 'blue') {
    state1 <- 'Z'
    for (j in 1:25) {
        if (sum(c(rpc(decisions$roll1[i]), row_b[2] + 1) == states[[j]]) == 2)
            state1 <- names(states)[j]
    }
    if (state1 %in% moves[[state_b]]) {
        scores1[i] <- gsc2(row_b[1] - decisions$roll1[i])
        positions1[i] <- rpc(decisions$roll1[i])
        counts1[i] <- row_b[2] + 1
    }
}

if (decisions$roll_color2[i] == 'skip') {
    scores2[i] <- 0
}
else if (decisions$roll_color2[i] == 'red') {
    state2 <- 'Z'
    if (decisions$roll_color1[i] == 'red') {
        for (j in 1:25) {
            if (sum(c(decisions$roll2[i], row_r[2] + 2) == states[[j]]) == 2)
                state2 <- names(states)[j]
        }
        if (state2 %in% moves[[state1]]) {
            scores2[i] <- gsc2(decisions$roll2[i] - decisions$roll1[i])
            positions2[i] <- decisions$roll2[i]
            counts2[i] <- row_r[2] + 2
        }
    }
    else {
        for (j in 1:25) {
            if (sum(c(decisions$roll2[i], row_r[2] + 1) == states[[j]]) == 2)
                state2 <- names(states)[j]
        }
        if (state2 %in% moves[[state_r]]) {
            scores2[i] <- gsc2(decisions$roll2[i] - row_r[1])
            positions2[i] <- decisions$roll2[i]
            counts2[i] <- row_r[2] + 1
        }
    }
}

65
}{
}else if (decisions$roll_color2[i] == 'yellow') {
  state2 <- 'Z'

  if (decisions$roll_color1[i] == 'yellow') {
    for (j in 1:25) {
      if (sum(c(decisions$roll2[i], row_y[2] + 2) == states[j]) == 2)
        state2 <- names(states)[j]
    }
    if (state2 %in% moves[[state1]]) {
      scores2[i] <- gsc2(decisions$roll2[i] - decisions$roll1[i])
      positions2[i] <- decisions$roll2[i]
      counts2[i] <- row_y[2] + 2
    }
  }
  else {
    for (j in 1:25) {
      if (sum(c(decisions$roll2[i], row_y[2] + 1) == states[j]) == 2)
        state2 <- names(states)[j]
    }
    if (state2 %in% moves[[state_y]]) {
      scores2[i] <- gsc2(decisions$roll2[i] - row_y[1])
      positions2[i] <- decisions$roll2[i]
      counts2[i] <- row_y[2] + 1
    }
  }
}
else if (decisions$roll_color2[i] == 'green') {
  state2 <- 'Z'

  if (decisions$roll_color1[i] == 'green') {
    for (j in 1:25) {
      if (sum(c(rpc(decisions$roll2[i]), row_g[2] + 2) == states[j]) == 2)
        state2 <- names(states)[j]
    }
    if (state2 %in% moves[[state1]]) {
      scores2[i] <- gsc2(decisions$roll1[i] - decisions$roll2[i])
      positions2[i] <- rpc(decisions$roll2[i])
      counts2[i] <- row_g[2] + 2
    }
  }
  else {
    for (j in 1:25) {
      if (sum(c(rpc(decisions$roll2[i]), row_g[2] + 1) == states[j]) == 2)
state2 <- names(states)[j]
}
if (state2 %in% moves[[state_g]]) {
  scores2[i] <- gsc2(row_g[1] - decisions$roll2[i])
  positions2[i] <- rpc(decisions$roll2[i])
  counts2[i] <- row_g[2] + 1
}
else if (decisions$roll_color2[i] == 'blue') {
  state2 <- 'Z'
  if(decisions$roll_color1[i] == 'blue') {
    for (j in 1:25) {
      if (sum(c(rpc(decisions$roll2[i]), row_b[2] + 2) == states[j]) == 2)
        state2 <- names(states)[j]
    }
    if (state2 %in% moves[[state1]]) {
      scores2[i] <- gsc2(decisions$roll1[i] - decisions$roll2[i])
      positions2[i] <- rpc(decisions$roll2[i])
      counts2[i] <- row_b[2] + 2
    }
    else {
      for (j in 1:25) {
        if (sum(c(rpc(decisions$roll2[i]), row_b[2] + 1) == states[j]) == 2)
          state2 <- names(states)[j]
      }
      if (state2 %in% moves[[state_b]]) {
        scores2[i] <- gsc2(row_b[1] - decisions$roll2[i])
        positions2[i] <- rpc(decisions$roll2[i])
        counts2[i] <- row_b[2] + 1
      }
    }
  }
}
else if (decisions$roll_color2[i] != 'skip') {
  if (!is.na(scores1[i]) &
      ((scores1[i] <= 0 | scores1[i] >= 7) | (positions1[i] == 12 & counts1[i] < 6)))
    scores1[i] <- NA
}
if (decisions$roll_color2[i] != 'skip') {
  if (!is.na(scores2[i]) &
((scores2[i] <= 0 | scores2[i] >= 7) | (positions2[i] == 12 & counts2[i] < 6)))
scores2[i] <- NA
}
scores[i] <- scores1[i] + scores2[i]
#
# calculating scores
best_score_index <- which(scores == max(na.omit(scores)))
if (length(best_score_index) > 1) best_score_index <- sample(best_score_index, 1)
return(decisions[best_score_index,])
}
Appendix E. Code for Single Player Game Simulation Function

```r
single_player <- function(strategy, priority_order = '123', skip_score = 5.5,
                         skip_adjustment = 0, weight = 1/200, move_option = 3, table = NULL) {
  penalty <- 0
  lock <- 0
  locks <- rep(FALSE, 4)
  row_r <- c(1, 0)
  row_y <- c(1, 0)
  row_g <- c(13, 0)
  row_b <- c(13, 0)

  iter <- 0
  while (penalty < 4 & lock < 2) {
    iter <- iter + 1

    # dice roll
    die_w1 <- sample(1:6, 1)
    die_w2 <- sample(1:6, 1)
    die_r <- sample(1:6, 1)
    die_y <- sample(1:6, 1)
    die_g <- sample(1:6, 1)
    die_b <- sample(1:6, 1)

    # roll 1
    roll1 <- die_w1 + die_w2

    # roll 2 options
    roll_r1 <- die_w1 + die_r
    roll_r2 <- die_w2 + die_r
    roll_y1 <- die_w1 + die_y
    roll_y2 <- die_w2 + die_y
    roll_g1 <- die_w1 + die_g
    roll_g2 <- die_w2 + die_g
    roll_b1 <- die_w1 + die_b
    roll_b2 <- die_w2 + die_b

    # decisions
    roll1s <- c(NA, rep(roll1, 4), rep(NA, 8), rep(roll1, 32))
    roll_colors1 <- c('skip', 'red', 'yellow', 'green', 'blue', rep('skip', 8),
                      rep('red', 8), rep('yellow', 8), rep('green', 8), rep('blue', 8))
    roll2s <- c(rep(NA, 5),
```
rep(c(roll_r1, roll_r2, roll_y1, roll_y2, roll_g1, roll_g2, roll_b1, roll_b2), 5))
roll_colors2 <- c(rep('skip', 5),
   rep(c(rep('red', 2), rep('yellow', 2),
   rep('green', 2), rep('blue', 2)), 5))
decisions <- data.frame(roll1 = roll1s, roll_color1 = roll_colors1,
   roll2 = roll2s, roll_color2 = roll_colors2)
for (i in 1:4) {
   if (i == 1) color = 'red'
   else if (i == 2) color = 'yellow'
   else if (i == 3) color = 'green'
   else if (i == 4) color = 'blue'
   if (locks[i]) {
      for (j in 1:45) {
         if (decisions$roll_color1[j] == color) {
            decisions$roll1[j] <- NA
            decisions$roll_color1[j] <- 'skip'
         }
         if (decisions$roll_color2[j] == color) {
            decisions$roll2[j] <- NA
            decisions$roll_color2[j] <- 'skip'
         }
      }
   }
}

# strategy
if (strategy == 0) {
   decision <- strategy0(row_r, row_y, row_g, row_b, decisions)
}
else if (strategy == 1) {
   decision <- strategy1(row_r, row_y, row_g, row_b, decisions, priority_order,
      skip_score, skip_adjustment, weight, penalty)
}
else if (strategy == 2) {
   decision <- strategy2(row_r, row_y, row_g, row_b, decisions, move_option)
}
else if(strategy == 3) {
   state <- c(row_r, row_y, row_g, row_b, paste(min(die_w1, die_w2),
      sum(die_w1, die_w2)),
      die_r, die_y, die_g, die_b, penalty)

output <- strategy3(row_r, row_y, row_g, row_b, decisions, skip_score, table, state)
decision <- output[[1]]
table <- output[[2]]
}
roll1 <- decision[[1]]
roll_color1 <- decision[[2]]
roll2 <- decision[[3]]
roll_color2 <- decision[[4]]

# roll 1 decision
if (roll_color1 == 'red') row_r <- c(roll1, row_r[2] + 1)
else if (roll_color1 == 'yellow') row_y <- c(roll1, row_y[2] + 1)
else if (roll_color1 == 'green') row_g <- c(roll1, row_g[2] + 1)
else if (roll_color1 == 'blue') row_b <- c(roll1, row_b[2] + 1)

# roll 2 decision
if (roll_color2 == 'red') row_r <- c(roll2, row_r[2] + 1)
else if (roll_color2 == 'yellow') row_y <- c(roll2, row_y[2] + 1)
else if (roll_color2 == 'green') row_g <- c(roll2, row_g[2] + 1)
else if (roll_color2 == 'blue') row_b <- c(roll2, row_b[2] + 1)

# penalty
if (roll_color1 == 'skip' & roll_color2 == 'skip') penalty <- penalty + 1

# lock
  locks[1] <- TRUE
  lock <- lock + 1
}
  locks[2] <- TRUE
  lock <- lock + 1
}
  locks[3] <- TRUE
  lock <- lock + 1
  row_g[2] <- row_g[2] + 1
}
  locks[4] <- TRUE
lock <- lock + 1

# calculate score
score <- row_score(row_r[2]) + row_score(row_y[2]) + row_score(row_g[2]) + row_score(row_b[2]) - penalty*5

return(list(score, iter, row_r, row_y, row_g, row_b, lock, penalty, table))
}
Appendix F. Code for Multiplayer Game Simulation Function

multiplayer <- function(num_players, strategies, priority_orders, skip_scores, skip_adjustments, weights, move_options) {
  turn <- sample(1:num_players, 1)
  penalties <- rep(0, num_players)
  lock <- 0
  locks <- rep(FALSE, 4)
  row_r <- vector("list", length = num_players)
  row_y <- vector("list", length = num_players)
  row_g <- vector("list", length = num_players)
  row_b <- vector("list", length = num_players)

  for (i in 1:num_players) {
    row_r[[i]] <- c(1, 0)
    row_y[[i]] <- c(1, 0)
    row_g[[i]] <- c(13, 0)
    row_b[[i]] <- c(13, 0)
  }

  iter <- 0
  while (max(penalties) < 4 & lock < 2) {
    iter <- iter + 1

    # dice roll
    die_w1 <- sample(1:6, 1)
    die_w2 <- sample(1:6, 1)
    die_r <- sample(1:6, 1)
    die_y <- sample(1:6, 1)
    die_g <- sample(1:6, 1)
    die_b <- sample(1:6, 1)

    # roll 1
    roll1 <- die_w1 + die_w2

    # roll 2 options
    roll_r1 <- die_w1 + die_r
    roll_r2 <- die_w2 + die_r
    roll_y1 <- die_w1 + die_y
    roll_y2 <- die_w2 + die_y
    roll_g1 <- die_w1 + die_g

roll_g2 <- die_w2 + die_g
roll_b1 <- die_w1 + die_b
roll_b2 <- die_w2 + die_b

# decisions
roll1s <- c(NA, rep(roll1, 4), rep(NA, 8), rep(roll1, 32))
roll_colors1 <- c('skip', 'red', 'yellow', 'green', 'blue', rep('skip', 8),
rep('red', 8), rep('yellow', 8), rep('green', 8), rep('blue', 8))
roll2s <- c(rep(NA, 5),
rep(c(roll_r1, roll_r2, roll_y1, roll_y2,
    roll_g1, roll_g2, roll_b1, roll_b2), 5))
roll_colors2 <- c(rep('skip', 5),
rep(c(rep('red', 2), rep('yellow', 2),
    rep('green', 2), rep('blue', 2)), 5))
decisions <- data.frame(roll1 = roll1s, roll_color1 = roll_colors1,
    roll2 = roll2s, roll_color2 = roll_colors2)
for (i in 1:4) {
    if (i == 1) color = 'red'
    else if (i == 2) color = 'yellow'
    else if (i == 3) color = 'green'
    else if (i == 4) color = 'blue'

    if (locks[i]) {
        for (j in 1:45) {
            if (decisions$roll_color1[j] == color) {
                decisions$roll1[j] <- NA
                decisions$roll_color1[j] <- 'skip'
            }
            if (decisions$roll_color2[j] == color) {
                decisions$roll2[j] <- NA
                decisions$roll_color2[j] <- 'skip'
            }
        }
    }
}
locks_temp <- rep(FALSE, 4)
for (i in 1:num_players) {
    # strategies
    if (strategies[i] == 0) {
        decision <- strategy0(row_r[i], row_y[i], row_g[i], row_b[i], decisions,
            (i == turn))
    }
else if (strategies[i] == 1) {
    decision <- strategy1(row_r[i], row_y[i], row_g[i], row_b[i], decisions, priority_orders[i], skip_scores[i], skip_adjustments[i], weights[i], penalties[i], (i == turn))
}
else if (strategies[i] == 2) {
    decision <- strategy2(row_r[i], row_y[i], row_g[i], row_b[i], decisions, move_options[i], (i == turn))
}
roll1 <- decision[1]
roll_color1 <- decision[2]
roll2 <- decision[3]
roll_color2 <- decision[4]

# roll 1 decision
if (roll_color1 == 'red') row_r[i] <- c(roll1, row_r[i][2] + 1)
else if (roll_color1 == 'yellow') row_y[i] <- c(roll1, row_y[i][2] + 1)
else if (roll_color1 == 'green') row_g[i] <- c(roll1, row_g[i][2] + 1)
else if (roll_color1 == 'blue') row_b[i] <- c(roll1, row_b[i][2] + 1)

# roll 2 decision
if (roll_color2 == 'red') row_r[i] <- c(roll2, row_r[i][2] + 1)
else if (roll_color2 == 'yellow') row_y[i] <- c(roll2, row_y[i][2] + 1)
else if (roll_color2 == 'green') row_g[i] <- c(roll2, row_g[i][2] + 1)
else if (roll_color2 == 'blue') row_b[i] <- c(roll2, row_b[i][2] + 1)

# penalty
if (i == turn & roll_color1 == 'skip' & roll_color2 == 'skip')
    penalties[i] <- penalties[i] + 1

# lock
if (!locks[1] & row_r[i][1] == 12 & row_r[i][2] > 5 & lock < 2) {
    locks_temp[1] <- TRUE
    row_r[i][2] <- row_r[i][2] + 1
}
    locks_temp[2] <- TRUE
    row_y[i][2] <- row_y[i][2] + 1
}
if (!locks[3] & row_g[i][1] == 2 & row_g[i][2] > 5 & lock < 2) {
    locks_temp[3] <- TRUE
}
row_g[[i]][2] <- row_g[[i]][2] + 1
}
if (!locks[4] & row_b[[i]][1] == 2 & row_b[[i]][2] > 5 & lock < 2) {
  locks_temp[4] <- TRUE
  row_b[[i]][2] <- row_b[[i]][2] + 1
}
}
}

for (i in 1:4) {
  if (!locks[i] & locks_temp[i]) {
    locks[i] <- TRUE
    lock <- lock + 1
  }
}

if (turn == num_players) turn <- 1
else turn <- turn + 1
}

# calculate scores and winner
scores <- rep(NA, num_players)
for (i in 1:num_players) {
  scores[i] <- row_score(row_r[[i]][2]) + row_score(row_y[[i]][2]) +
              row_score(row_g[[i]][2]) + row_score(row_b[[i]][2]) - penalties[i]*5
}
winner <- which(scores == max(scores))

if (length(winner) > 1) return(list('Tie', scores, iter))
return(list(paste('Player', winner), scores, iter))
}
Appendix G. Code for Single Player Game Simulation Results

# Strategy 0
n <- 10000
scores <- c()
num_rolls <- c()
row_r1 <- c()
row_r2 <- c()
row_y1 <- c()
row_y2 <- c()
row_g1 <- c()
row_g2 <- c()
row_b1 <- c()
row_b2 <- c()
locks <- c()
penalties <- c()

for (i in 1:n) {
  output <- single_player(0)
  scores <- append(scores, output[[1]])
  num_rolls <- append(num_rolls, output[[2]])
  row_r1 <- append(row_r1, output[[3]][1])
  row_r2 <- append(row_r2, output[[3]][2])
  row_y1 <- append(row_y1, output[[4]][1])
  row_y2 <- append(row_y2, output[[4]][2])
  row_g1 <- append(row_g1, output[[5]][1])
  row_g2 <- append(row_g2, output[[5]][2])
  row_b1 <- append(row_b1, output[[6]][1])
  row_b2 <- append(row_b2, output[[6]][2])
  locks <- append(locks, output[[7]])
  penalties <- append(penalties, output[[8]])
}

results <- data.frame(score = scores, num_rolls = num_rolls, red_position = row_r1, red_x = row_r2, yellow_position = row_y1, yellow_x = row_y2, green_position = row_g1, green_x = row_b2, blue_position = row_b1, blue_x = row_b2, locks = locks, penalties = penalties) %>%
tibble()
# Strategy 1
n <- 10000
scores <- c()
num_rolls <- c()
row_r1 <- c()
row_r2 <- c()
row_y1 <- c()
row_y2 <- c()
row_g1 <- c()
row_g2 <- c()
row_b1 <- c()
row_b2 <- c()
locks <- c()
penalties <- c()

for (i in 1:n) {
  output <- single_player(1, skip_score = 5, weight = 1/300)
  scores <- append(scores, output[[1]])
  num_rolls <- append(num_rolls, output[[2]])
  row_r1 <- append(row_r1, output[[3]][1])
  row_r2 <- append(row_r2, output[[3]][2])
  row_y1 <- append(row_y1, output[[4]][1])
  row_y2 <- append(row_y2, output[[4]][2])
  row_g1 <- append(row_g1, output[[5]][1])
  row_g2 <- append(row_g2, output[[5]][2])
  row_b1 <- append(row_b1, output[[6]][1])
  row_b2 <- append(row_b2, output[[6]][2])
  locks <- append(locks, output[[7]])
  penalties <- append(penalties, output[[8]])
}

results <- data.frame(score = scores, num_rolls = num_rolls,
                       red_position = row_r1, red_x = row_r2,
                       yellow_position = row_y1, yellow_x = row_y2,
                       green_position = row_g1, green_x = row_b2,
                       blue_position = row_b1, blue_x = row_b2,
                       locks = locks, penalties = penalties) %>%
  tibble()

# Strategy 2
n <- 10000
scores <- c()
num_rolls <- c()
row_r1 <- c()
row_r2 <- c()
row_y1 <- c()
row_y2 <- c()
row_g1 <- c()
row_g2 <- c()
row_b1 <- c()
row_b2 <- c()
locks <- c()
penalties <- c()

for (i in 1:n) {
  output <- single_player(2, move_option = 3)
scores <- append(scores, output[[1]])
  num_rolls <- append(num_rolls, output[[2]])
  row_r1 <- append(row_r1, output[[3]][1])
  row_r2 <- append(row_r2, output[[3]][2])
  row_y1 <- append(row_y1, output[[4]][1])
  row_y2 <- append(row_y2, output[[4]][2])
  row_g1 <- append(row_g1, output[[5]][1])
  row_g2 <- append(row_g2, output[[5]][2])
  row_b1 <- append(row_b1, output[[6]][1])
  row_b2 <- append(row_b2, output[[6]][2])
  locks <- append(locks, output[[7]])
  penalties <- append(penalties, output[[8]])
}

results <- data.frame(score = scores, num_rolls = num_rolls,
  red_position = row_r1, red_x = row_r2,
  yellow_position = row_y1, yellow_x = row_y2,
  green_position = row_g1, green_x = row_b2,
  blue_position = row_b1, blue_x = row_b2,
  locks = locks, penalties = penalties) %>%
tibble()

# Scores
summary(scores)
ggplot(results) +
  geom_histogram(aes(x = score), fill = 'red', alpha = 1, binwidth = 3) +
  labs(title = '', x = 'Score', y = '') +
  theme_bw()
# Number of Rolls

summary(num_rolls)
ggplot(results) +
  geom_histogram(aes(x = num_rolls), fill = 'black', alpha = 1, bins = 15) +
  labs(title = '', x = 'Rolls', y = '') +
  theme_bw()

# Other

cor(scores, num_rolls)
table(results$locks, results$penalties)
plot(scores, num_rolls, pch = 16, main = '',
      xlab = 'Scores', ylab = 'Rolls')
Appendix H. Code for 2 Player Game Simulation Results

# Strategy 1 vs. Strategy 1
n <- 10000
scores <- data.frame(winner = rep(NA, n), p1_score = rep(NA, n), p2_score = rep(NA, n),
        num_rolls = rep(NA, n))
for (i in 1:n) {
    score <- multiplayer(2, c(1, 1), c('123', '123'), c(5, 5), c(0, 0), c(1/300, 1/300),
            c(NA, NA))
    scores$winner[i] <- score[[1]]
    scores$p1_score[i] <- score[[2]][1]
    scores$p2_score[i] <- score[[2]][2]
    scores$num_rolls[i] <- score[[3]]
}

# Strategy 2 vs. Strategy 2
n <- 10000
scores <- data.frame(winner = rep(NA, n), p1_score = rep(NA, n), p2_score = rep(NA, n),
        num_rolls = rep(NA, n))
for (i in 1:n) {
    score <- multiplayer(2, c(2, 2), c(NA, NA), c(NA, NA), c(NA, NA), c(NA, NA), c(3, 3))
    scores$winner[i] <- score[[1]]
    scores$p1_score[i] <- score[[2]][1]
    scores$p2_score[i] <- score[[2]][2]
    scores$num_rolls[i] <- score[[3]]
}

# Winner Distribution
table(scores$winner)

# Player Scores
summary(scores$p1_score)
summary(scores$p2_score)
ggplot(scores) +
    geom_histogram(aes(x = p1_score), fill = 'red', alpha = 1, binwidth = 6) +
    labs(title = '', x = 'Score', y = '') +
    theme_bw()
ggplot(scores, aes(x = p1_score, y = p2_score, color = winner)) +
  geom_point() +
  labs(title = '', x = 'Player 1 Score (S.B.)', y = 'Player 2 Score (S.B.)',
       color = "Winner") +
  scale_color_manual(values=c("red", "blue", "gold")) +
  theme_bw()

# Number of Rolls
summary(scores$num_rolls)

ggplot(scores) +
  geom_histogram(aes(x = num_rolls), fill = 'black', alpha = 1, bins = 18) +
  labs(title = '', x = 'Rolls', y = '') +
  theme_bw()

# Strategy 1 vs. Strategy 2
n <- 10000
scores <- data.frame(winner = rep(NA, n), p1_score = rep(NA, n), p2_score = rep(NA, n),
                     num_rolls = rep(NA, n))

for (i in 1:n) {
  score <- multiplayer(2, c(1, 2), c('123', NA), c(5, NA), c(0, NA), c(1/300, NA), c(NA, 3))
  scores$winner[i] <- score[[1]]
  scores$p1_score[i] <- score[[2]][1]
  scores$p2_score[i] <- score[[2]][2]
  scores$num_rolls[i] <- score[[3]]
}

# Wide to Long Data
scores2 <- gather(scores[,2:3], player, score, p1_score:p2_score, factor_key=TRUE)
scores2$player <- ifelse(scores2$player == 'p1_score', 'Player 1 Score (S.B.)','Player 2 Score (R.T.L.)')

# Winner Distribution
table(scores$winner)

# Player Scores
summary(scores$p1_score)
summary(scores$p2_score)

ggplot(scores2, aes(x = score, y = player)) +
  geom_density_ridges(aes(fill = player), bandwidth = 3) +
  ...
labs(title = ", x = 'Score', y = ") +
scale_fill_manual(values = c('red', 'blue')) +
guides(fill="none") +
theme_bw()

ggplot(scores, aes(x = p1_score, y = p2_score, color = winner)) +
  geom_point() +
  labs(title = ", x = 'Player 1 Score (S.B.)', y = 'Player 2 Score (R.T.L)',
       color = "Winner") +
  scale_color_manual(values=c("red", "blue", "gold")) +
  theme_bw()

# Number of Rolls
summary(scores$num_rolls)

ggplot(scores) +
  geom_histogram(aes(x = num_rolls), fill = 'black', alpha = 1, bins = 18) +
  labs(title = ", x = 'Rolls', y = ") +
  theme_bw()
Appendix I. Code for Tuning Parameters

```r
# Tuning Skip Score
n <- 1000
magnitude <- 0.1
skips <- rep(NA, n)
mean_scores <- rep(NA, n)

reps <- 100
scores <- rep(NA, reps)
current <- 5

for (rep in 1:reps) {
  output <- single_player(1, skip_score = current, weight = 0)
  scores[rep] <- output[[1]]
}

mean_scores[1] <- mean(scores)
skips[1] <- current

for (i in 2:n) {
  scores <- rep(NA, reps)
direction <- sample(c(-1, 1), 1, prob = c(0.5, 0.5))
if (current + magnitude*direction > 0 | current + magnitude*direction < 10)
  proposed <- current + magnitude*direction
else proposed <- current - magnitude*direction

for (rep in 1:reps) {
  output <- single_player(1, skip_score = proposed, weight = 0)
  scores[rep] <- output[[1]]
}

mean_scores[i] <- mean(scores)
prob <- max(0, min(1, mean_scores[i] / mean_scores[i-1]))
change <- sample(c(TRUE, FALSE), 1, prob = c(prob, 1-prob))

if (change) skips[i] <- current <- proposed
else skips[i] <- current
}

skips %>% round(1) %>% table() %>% sort(decreasing=TRUE) %>% head(10)
```
# Tuning Likelihood Score Weight
n <- 1000
magnitude <- 10
weights <- rep(NA, n)
mean_scores <- rep(NA, n)
reps <- 100
scores <- rep(NA, reps)
current <- 300

for (rep in 1:reps) {
  output <- single_player(1, weight = 1/current)
  scores[rep] <- output[[1]]
}
mean_scores[1] <- mean(scores)
weights[1] <- current

for (i in 2:n) {
  scores <- rep(NA, reps)
  direction <- sample(c(-1, 1), 1, prob = c(0.5, 0.5))
  if (current + magnitude*direction > 0) proposed <- current + magnitude*direction
  else proposed <- current - magnitude*direction
  for (rep in 1:reps) {
    output <- single_player(1, weight = 1/proposed)
    scores[rep] <- output[[1]]
  }
  mean_scores[i] <- mean(scores)
  prob <- max(0, min(1, mean_scores[i] / mean_scores[i-1]))
  change <- sample(c(TRUE, FALSE), 1, prob = c(prob, 1-prob))
  if (change) weights[i] <- current - proposed
  else weights[i] <- current
}
weights %>% table() %>% sort(decreasing=TRUE) %>% head(10)
Appendix J. Code for Race-to-Lock Strategy Transition Probabilities and Means

```r
poss <- data.frame(die1 = rep(c(rep(1, 6), rep(2, 6), rep(3, 6),
       rep(4, 6), rep(5, 6), rep(6, 6)),
       6),
    die2 = rep(c(rep(c(1, 2, 3, 4, 5, 6), 6)), 6),
    die3 = c(rep(1, 36), rep(2, 36), rep(3, 36),
       rep(4, 36), rep(5, 36), rep(6, 36)))
poss$sum1 <- poss$die1 + poss$die2
poss$sum2 <- poss$die1 + poss$die3
poss$sum3 <- poss$die2 + poss$die3

# 1 Sum

# No Restrictions
prob1 <- c(# 0 to 1
  1,
  mean(poss$sum1 == 2),
  mean(poss$sum1 == 3),
  mean(poss$sum1 == 4),
  mean(poss$sum1 == 5),
  mean(poss$sum1 == 6),
  mean(poss$sum1 == 7),
  rep(0, 19),
  # 1 to 2
  0,
  1,
  rep(0, 5),
  mean(poss$sum1 == 3),
  mean(poss$sum1 == 4),
  mean(poss$sum1 == 5),
  mean(poss$sum1 == 6),
  mean(poss$sum1 == 7),
  mean(poss$sum1 == 8),
  rep(0, 13),
  rep(0, 2),
  1,
  rep(0, 5),
  mean(poss$sum1 == 4),
  mean(poss$sum1 == 5),
  mean(poss$sum1 == 6),
```

mean(poss$sum1 == 7),
mean(poss$sum1 == 8),
rep(0, 13),
rep(0, 3),
1,
rep(0, 5),
mean(poss$sum1 == 5),
mean(poss$sum1 == 6),
mean(poss$sum1 == 7),
mean(poss$sum1 == 8),
rep(0, 13),
rep(0, 4),
1,
rep(0, 5),
mean(poss$sum1 == 6),
mean(poss$sum1 == 7),
mean(poss$sum1 == 8),
rep(0, 13),
rep(0, 5),
1,
rep(0, 5),
mean(poss$sum1 == 7),
mean(poss$sum1 == 8),
rep(0, 13),
rep(0, 6),
1,
rep(0, 5),
mean(poss$sum1 == 8),
rep(0, 13),
# 2 to 3
rep(0, 7),
1,
rep(0, 5),
mean(poss$sum1 == 4),
mean(poss$sum1 == 5),
mean(poss$sum1 == 6),
mean(poss$sum1 == 7),
mean(poss$sum1 == 8),
mean(poss$sum1 == 9),
rep(0, 7),
rep(0, 8),
1,
\text{rep}(0, 5),
\text{mean(poss$sum1 == 5)},
\text{mean(poss$sum1 == 6)},
\text{mean(poss$sum1 == 7)},
\text{mean(poss$sum1 == 8)},
\text{mean(poss$sum1 == 9)},
\text{rep}(0, 7),
\text{rep}(0, 9),
1,
\text{rep}(0, 5),
\text{mean(poss$sum1 == 6)},
\text{mean(poss$sum1 == 7)},
\text{mean(poss$sum1 == 8)},
\text{mean(poss$sum1 == 9)},
\text{rep}(0, 7),
\text{rep}(0, 10),
1,
\text{rep}(0, 5),
\text{mean(poss$sum1 == 7)},
\text{mean(poss$sum1 == 8)},
\text{mean(poss$sum1 == 9)},
\text{rep}(0, 7),
\text{rep}(0, 11),
1,
\text{rep}(0, 5),
\text{mean(poss$sum1 == 8)},
\text{mean(poss$sum1 == 9)},
\text{rep}(0, 7),
\text{rep}(0, 12),
1,
\text{rep}(0, 5),
\text{mean(poss$sum1 == 9)},
\text{rep}(0, 7),
\# 3 to 4
\text{rep}(0, 13),
1,
\text{rep}(0, 5),
\text{mean(poss$sum1 == 5)},
\text{mean(poss$sum1 == 6)},
\text{mean(poss$sum1 == 7)},
\text{mean(poss$sum1 == 8)},
\text{mean(poss$sum1 == 9)},
mean(poss$sum1 == 10), 0,
rep(0, 14),
1,
rep(0, 5),
mean(poss$sum1 == 6), mean(poss$sum1 == 7), mean(poss$sum1 == 8), mean(poss$sum1 == 9), mean(poss$sum1 == 10), 0,
rep(0, 15),
1,
rep(0, 5),
mean(poss$sum1 == 7), mean(poss$sum1 == 8), mean(poss$sum1 == 9), mean(poss$sum1 == 10), 0,
rep(0, 16),
1,
rep(0, 5),
mean(poss$sum1 == 8), mean(poss$sum1 == 9), mean(poss$sum1 == 10), 0,
rep(0, 17),
1,
rep(0, 5),
mean(poss$sum1 == 9), mean(poss$sum1 == 10), 0,
rep(0, 18),
1,
rep(0, 5),
mean(poss$sum1 == 10), 0,
# 4 to 5
rep(0, 19), 1,
rep(0, 5),
mean(poss$sum1 %in% 6:11),
rep(0, 20),
1,
rep(0, 4),
mean(poss$sum1 %in% 7:11),
rep(0, 21),
1,
rep(0, 3),
mean(poss$sum1 %in% 8:11),
rep(0, 22),
1,
rep(0, 2),
mean(poss$sum1 %in% 9:11),
rep(0, 23),
1,
0,
mean(poss$sum1 %in% 10:11),
rep(0, 24),
1,
mean(poss$sum1 == 11),
rep(0, 25),
1)

# Restrictions
prob1 <- c(# 0 to 1
1,
mean(poss$sum1 == 2),
mean(poss$sum1 == 3),
mean(poss$sum1 == 4),
0,
0,
0,
rep(0, 19),
# 1 to 2
0,
1,
rep(0, 5),
mean(poss$sum1 == 3),
mean(poss$sum1 == 4),
mean(poss$sum1 == 5),
0,
0,
0,
rep(0, 13),
rep(0, 2),
1,
rep(0, 5),
mean(poss$sum1 == 4),
mean(poss$sum1 == 5),
mean(poss$sum1 == 6),
0,
0,
rep(0, 13),
rep(0, 3),
1,
rep(0, 5),
mean(poss$sum1 == 5),
mean(poss$sum1 == 6),
0,
0,
rep(0, 13),
rep(0, 4),
1,
rep(0, 5),
mean(poss$sum1 == 6),
0,
0,
rep(0, 13),
rep(0, 5),
1,
rep(0, 5),
mean(poss$sum1 == 7),
0,
rep(0, 13),
rep(0, 6),
1,
rep(0, 5),
mean(poss$sum1 == 8),
rep(0, 13),
# 2 to 3
rep(0, 7),
1,
rep(0, 5),
mean(poss$sum1 == 4),
mean(poss$sum1 == 5),
mean(poss$sum1 == 6),
0,
0,
0,
rep(0, 7),
rep(0, 8),
1,
rep(0, 5),
mean(poss$sum1 == 5),
mean(poss$sum1 == 6),
mean(poss$sum1 == 7),
0,
0,
rep(0, 7),
rep(0, 9),
1,
rep(0, 5),
mean(poss$sum1 == 6),
mean(poss$sum1 == 7),
0,
0,
rep(0, 7),
rep(0, 10),
1,
rep(0, 5),
mean(poss$sum1 == 7),
0,
0,
rep(0, 7),
rep(0, 11),
1,
rep(0, 5),
mean(poss$sum1 == 8),
0,
rep(0, 7),
rep(0, 12),
1,
rep(0, 5),
mean(poss$sum1 == 9),
rep(0, 7),
# 3 to 4
rep(0, 13),
1,
rep(0, 5),
mean(poss$sum1 == 5),
mean(poss$sum1 == 6),
mean(poss$sum1 == 7),
mean(poss$sum1 == 8),
0,
0,
0,
rep(0, 14),
1,
rep(0, 5),
mean(poss$sum1 == 6),
mean(poss$sum1 == 7),
mean(poss$sum1 == 8),
0,
0,
0,
rep(0, 15),
1,
rep(0, 5),
mean(poss$sum1 == 7),
mean(poss$sum1 == 8),
0,
0,
0,
rep(0, 16),
1,
rep(0, 5),
mean(poss$sum1 == 8),
mean(poss$sum1 == 9),
0,
0,
rep(0, 17),
1,
rep(0, 5),
mean(poss$sum1 == 9),
0,
0,
rep(0, 18),
1,
rep(0, 5),
mean(poss$sum1 == 10),
0,
# 4 to 5
rep(0, 19),
1,
rep(0, 5),
mean(poss$sum1 %in% 6:11),
rep(0, 20),
1,
rep(0, 4),
mean(poss$sum1 %in% 7:11),
rep(0, 21),
1,
rep(0, 3),
mean(poss$sum1 %in% 8:11),
rep(0, 22),
1,
rep(0, 2),
mean(poss$sum1 %in% 9:11),
rep(0, 23),
1,
0,
mean(poss$sum1 %in% 10:11),
rep(0, 24),
1,
mean(poss$sum1 == 11),
rep(0, 25),
1)

# 2 Sums

# No Restrictions
prob2 <- c(# 0 to 1
1,
mean(poss$sum1 == 2 | poss$sum2 == 2),
mean((poss$sum1 == 3 | poss$sum2 == 3) &
    (poss$sum1 >= 3 & poss$sum2 >= 3)),
mean((poss$sum1 == 4 | poss$sum2 == 4) &
    (poss$sum1 >= 4 & poss$sum2 >= 4)),
mean((poss$sum1 == 5 | poss$sum2 == 5) &
    (poss$sum1 >= 5 & poss$sum2 >= 5)),
mean((poss$sum1 == 6 | poss$sum2 == 6) &
    (poss$sum1 >= 6 & poss$sum2 >= 6)),
(poss$sum1 >= 6 & poss$sum2 >= 6)),
mean((poss$sum1 == 7 | poss$sum2 == 7) &
(poss$sum1 >= 7 & poss$sum2 >= 7)),
rep(0, 19),
# 1 to 2
0,
1,
rep(0, 5),
mean(poss$sum1 == 3 | poss$sum2 == 3),
mean((poss$sum1 == 4 | poss$sum2 == 4) &
 !(poss$sum1 %in% 3) & !(poss$sum2 %in% 3)),
mean((poss$sum1 == 5 | poss$sum2 == 5) &
 !(poss$sum1 %in% 3:4) & !(poss$sum2 %in% 3:4)),
mean((poss$sum1 == 6 | poss$sum2 == 6) &
 !(poss$sum1 %in% 3:5) & !(poss$sum2 %in% 3:5)),
mean((poss$sum1 == 7 | poss$sum2 == 7) &
 !(poss$sum1 %in% 3:6) & !(poss$sum2 %in% 3:6)),
mean((poss$sum1 == 8 | poss$sum2 == 8) &
 !(poss$sum1 %in% 3:7) & !(poss$sum2 %in% 3:7)),
rep(0, 13),
rep(0, 2),
1,
rep(0, 5),
mean(poss$sum1 == 4 | poss$sum2 == 4),
mean((poss$sum1 == 5 | poss$sum2 == 5) &
 !(poss$sum1 %in% 4) & !(poss$sum2 %in% 4)),
mean((poss$sum1 == 6 | poss$sum2 == 6) &
 !(poss$sum1 %in% 4:5) & !(poss$sum2 %in% 4:5)),
mean((poss$sum1 == 7 | poss$sum2 == 7) &
 !(poss$sum1 %in% 4:6) & !(poss$sum2 %in% 4:6)),
mean((poss$sum1 == 8 | poss$sum2 == 8) &
 !(poss$sum1 %in% 4:7) & !(poss$sum2 %in% 4:7)),
rep(0, 13),
rep(0, 3),
1,
rep(0, 5),
mean(poss$sum1 == 5 | poss$sum2 == 5),
mean((poss$sum1 == 6 | poss$sum2 == 6) &
 !(poss$sum1 %in% 5) & !(poss$sum2 %in% 5)),
mean((poss$sum1 == 7 | poss$sum2 == 7) &
 !(poss$sum1 %in% 5:6) & !(poss$sum2 %in% 5:6)),
mean((poss$sum1 == 8 | poss$sum2 == 8) &
 !(poss$sum1 %in% 5:7) & !(poss$sum2 %in% 5:7)),
rep(0, 13),
rep(0, 3),
1,
(!(poss$sum1 %in% 5:7) & !(poss$sum2 %in% 5:7)),
rep(0, 13),
rep(0, 4),
1,
rep(0, 5),
mean(poss$sum1 == 6 | poss$sum2 == 6),
mean((poss$sum1 == 7 | poss$sum2 == 7) &
  (!!(poss$sum1 %in% 6) & !(poss$sum2 %in% 6))),
mean((poss$sum1 == 8 | poss$sum2 == 8) &
  (!!(poss$sum1 %in% 6:7) & !(poss$sum2 %in% 6:7))),
rep(0, 13),
rep(0, 5),
1,
rep(0, 5),
mean(poss$sum1 == 7 | poss$sum2 == 7),
mean((poss$sum1 == 8 | poss$sum2 == 8) &
  (!!(poss$sum1 %in% 7) & !(poss$sum2 %in% 7))),
rep(0, 13),
rep(0, 6),
1,
rep(0, 5),
mean(poss$sum1 == 8 | poss$sum2 == 8),
rep(0, 13),
# 2 to 3
rep(0, 7),
1,
rep(0, 5),
mean(poss$sum1 == 4 | poss$sum2 == 4),
mean(poss$sum1 == 5 | poss$sum2 == 5) &
  (!!(poss$sum1 %in% 4) & !(poss$sum2 %in% 4))),
mean((poss$sum1 == 6 | poss$sum2 == 6) &
  (!!(poss$sum1 %in% 4:5) & !(poss$sum2 %in% 4:5))),
mean((poss$sum1 == 7 | poss$sum2 == 7) &
  (!!(poss$sum1 %in% 4:6) & !(poss$sum2 %in% 4:6))),
mean((poss$sum1 == 8 | poss$sum2 == 8) &
  (!!(poss$sum1 %in% 4:7) & !(poss$sum2 %in% 4:7))),
mean((poss$sum1 == 9 | poss$sum2 == 9) &
  (!!(poss$sum1 %in% 4:8) & !(poss$sum2 %in% 4:8))),
rep(0, 7),
rep(0, 8),
1,
rep(0, 5),
mean(poss$sum1 == 5 | poss$sum2 == 5),
mean((poss$sum1 == 6 | poss$sum2 == 6) &
    (!!(poss$sum1 %in% 5) & !(poss$sum2 %in% 5))),
mean((poss$sum1 == 7 | poss$sum2 == 7) &
    (!!(poss$sum1 %in% 5:6) & !(poss$sum2 %in% 5:6))),
mean((poss$sum1 == 8 | poss$sum2 == 8) &
    (!!(poss$sum1 %in% 5:7) & !(poss$sum2 %in% 5:7))),
mean((poss$sum1 == 9 | poss$sum2 == 9) &
    (!!(poss$sum1 %in% 5:8) & !(poss$sum2 %in% 5:8))),
rep(0, 7),
rep(0, 9),
1,
rep(0, 5),
mean(poss$sum1 == 6 | poss$sum2 == 6),
mean((poss$sum1 == 7 | poss$sum2 == 7) &
    (!!(poss$sum1 %in% 6) & !(poss$sum2 %in% 6))),
mean((poss$sum1 == 8 | poss$sum2 == 8) &
    (!!(poss$sum1 %in% 6:7) & !(poss$sum2 %in% 6:7))),
mean((poss$sum1 == 9 | poss$sum2 == 9) &
    (!!(poss$sum1 %in% 6:8) & !(poss$sum2 %in% 6:8))),
rep(0, 7),
rep(0, 10),
1,
rep(0, 5),
mean(poss$sum1 == 7 | poss$sum2 == 7),
mean((poss$sum1 == 8 | poss$sum2 == 8) &
    (!!(poss$sum1 %in% 7) & !(poss$sum2 %in% 7))),
mean((poss$sum1 == 9 | poss$sum2 == 9) &
    (!!(poss$sum1 %in% 7:8) & !(poss$sum2 %in% 7:8))),
rep(0, 7),
rep(0, 11),
1,
rep(0, 5),
mean(poss$sum1 == 8 | poss$sum2 == 8),
mean((poss$sum1 == 9 | poss$sum2 == 9) &
    (!!(poss$sum1 %in% 8) & !(poss$sum2 %in% 8))),
rep(0, 7),
rep(0, 12),
1,
rep(0, 5),
mean(poss$sum1 == 9 | poss$sum2 == 9),
rep(0, 7),

97
# 3 to 4
rep(0, 13),
1,
rep(0, 5),
mean(poss$sum1 == 5 | poss$sum2 == 5),
mean((poss$sum1 == 6 | poss$sum2 == 6) &
  !(poss$sum1 %in% 5) & !(poss$sum2 %in% 5)),
mean((poss$sum1 == 7 | poss$sum2 == 7) &
  !(poss$sum1 %in% 5:6) & !(poss$sum2 %in% 5:6)),
mean((poss$sum1 == 8 | poss$sum2 == 8) &
  !(poss$sum1 %in% 5:7) & !(poss$sum2 %in% 5:7))),
mean((poss$sum1 == 9 | poss$sum2 == 9) &
  !(poss$sum1 %in% 5:8) & !(poss$sum2 %in% 5:8)),
mean((poss$sum1 == 10 | poss$sum2 == 10) &
  !(poss$sum1 %in% 5:9) & !(poss$sum2 %in% 5:9)),
0,
rep(0, 14),
1,
rep(0, 5),
mean(poss$sum1 == 6 | poss$sum2 == 6),
mean((poss$sum1 == 7 | poss$sum2 == 7) &
  !(poss$sum1 %in% 6) & !(poss$sum2 %in% 6)),
mean((poss$sum1 == 8 | poss$sum2 == 8) &
  !(poss$sum1 %in% 6:7) & !(poss$sum2 %in% 6:7))),
mean((poss$sum1 == 9 | poss$sum2 == 9) &
  !(poss$sum1 %in% 6:8) & !(poss$sum2 %in% 6:8)),
mean((poss$sum1 == 10 | poss$sum2 == 10) &
  !(poss$sum1 %in% 6:9) & !(poss$sum2 %in% 6:9))),
0,
rep(0, 15),
1,
rep(0, 5),
mean(poss$sum1 == 7 | poss$sum2 == 7),
mean((poss$sum1 == 8 | poss$sum2 == 8) &
  !(poss$sum1 %in% 7) & !(poss$sum2 %in% 7)),
mean((poss$sum1 == 9 | poss$sum2 == 9) &
  !(poss$sum1 %in% 7:8) & !(poss$sum2 %in% 7:8)),
mean((poss$sum1 == 10 | poss$sum2 == 10) &
  !(poss$sum1 %in% 7:9) & !(poss$sum2 %in% 7:9)),
0,
rep(0, 16),
1,
rep(0, 5),
mean(poss$sum1 == 8 | poss$sum2 == 8),
mean((poss$sum1 == 9 | poss$sum2 == 9) &
    (!((poss$sum1 %in% 8) & !((poss$sum2 %in% 8)))))
mean((poss$sum1 == 10 | poss$sum2 == 10) &
    (!((poss$sum1 %in% 9) & !((poss$sum2 %in% 9))))),
0,
rep(0, 18),
1,
rep(0, 5),
mean(poss$sum1 == 9 | poss$sum2 == 9),
mean((poss$sum1 == 10 | poss$sum2 == 10) &
    (!((poss$sum1 %in% 9) & !((poss$sum2 %in% 9))))),
0,
# 4 to 5
rep(0, 19),
1,
rep(0, 5),
mean(poss$sum1 %in% 6:11 | poss$sum2 %in% 6:11),
rep(0, 20),
1,
rep(0, 4),
mean(poss$sum1 %in% 7:11 | poss$sum2 %in% 7:11),
rep(0, 21),
1,
rep(0, 3),
mean(poss$sum1 %in% 8:11 | poss$sum2 %in% 8:11),
rep(0, 22),
1,
rep(0, 2),
mean(poss$sum1 %in% 9:11 | poss$sum2 %in% 9:11),
rep(0, 23),
1,
0,
mean(poss$sum1 %in% 10:11 | poss$sum2 %in% 10:11),
rep(0, 24),
1,
mean(poss$sum1 == 11 | poss$sum2 == 11),
rep(0, 25),
1)

# Restrictions
prob2 <- c(# 0 to 1
1,
mean(poss$sum1 == 2 | poss$sum2 == 2),
mean((poss$sum1 == 3 | poss$sum2 == 3) &
    (poss$sum1 >= 3 & poss$sum2 >= 3)),
mean((poss$sum1 == 4 | poss$sum2 == 4) &
    (poss$sum1 >= 4 & poss$sum2 >= 4)),
mean((poss$sum1 == 5 | poss$sum2 == 5) &
    (poss$sum1 >= 5 & poss$sum2 >= 5)),
0,
0,
rep(0, 19),
# 1 to 2
0,
1,
rep(0, 5),
mean(poss$sum1 == 3 | poss$sum2 == 3),
mean((poss$sum1 == 4 | poss$sum2 == 4) &
    (!((poss$sum1 %in% 3) & !((poss$sum2 %in% 3))),
mean((poss$sum1 == 5 | poss$sum2 == 5) &
    (!((poss$sum1 %in% 3:4) & !((poss$sum2 %in% 3:4))),
0,
0,
0,
rep(0, 13),
rep(0, 2),
1,
rep(0, 5),
mean(poss$sum1 == 4 | poss$sum2 == 4),
mean((poss$sum1 == 5 | poss$sum2 == 5) &
    (!((poss$sum1 %in% 4) & !((poss$sum2 %in% 4))),
mean((poss$sum1 == 6 | poss$sum2 == 6) &
    (!((poss$sum1 %in% 4:5) & !((poss$sum2 %in% 4:5))),
0,
0,
rep(0, 13),
rep(0, 3),
1, 
rep(0, 5), 
mean(poss$sum1 == 5 | poss$sum2 == 5),
mean((poss$sum1 == 6 | poss$sum2 == 6) & 
    (!!(poss$sum1 %in% 5) & !(poss$sum2 %in% 5))),
0, 
0, 
rep(0, 13), 
rep(0, 4), 
1, 
rep(0, 5), 
mean(poss$sum1 == 6 | poss$sum2 == 6),
0, 
0, 
rep(0, 13), 
rep(0, 5), 
1, 
rep(0, 5), 
mean(poss$sum1 == 7 | poss$sum2 == 7),
0, 
rep(0, 13), 
rep(0, 6), 
1, 
rep(0, 5), 
mean(poss$sum1 == 8 | poss$sum2 == 8),
rep(0, 13), 
# 2 to 3 
rep(0, 7), 
1, 
rep(0, 5), 
mean(poss$sum1 == 4 | poss$sum2 == 4),
mean((poss$sum1 == 5 | poss$sum2 == 5) & 
    (!!(poss$sum1 %in% 4) & !(poss$sum2 %in% 4))),
mean((poss$sum1 == 6 | poss$sum2 == 6) & 
    (!!(poss$sum1 %in% 4:5) & !(poss$sum2 %in% 4:5))),
0, 
0, 
rep(0, 7), 
rep(0, 8), 
1, 
rep(0, 5),
\begin{verbatim}
mean(poss$sum1 == 5 | poss$sum2 == 5),
mean((poss$sum1 == 6 | poss$sum2 == 6) & 
    !(poss$sum1 %in% 5) & !(poss$sum2 %in% 5)),
mean((poss$sum1 == 7 | poss$sum2 == 7) & 
    !(poss$sum1 %in% 5:6) & !(poss$sum2 %in% 5:6)),
0,
0,
rep(0, 7),
rep(0, 9),
1,
rep(0, 5),
mean(poss$sum1 == 6 | poss$sum2 == 6),
mean((poss$sum1 == 7 | poss$sum2 == 7) & 
    !(poss$sum1 %in% 6) & !(poss$sum2 %in% 6)),
0,
0,
rep(0, 7),
rep(0, 10),
1,
rep(0, 5),
mean(poss$sum1 == 7 | poss$sum2 == 7),
0,
0,
rep(0, 7),
rep(0, 11),
1,
rep(0, 5),
mean(poss$sum1 == 8 | poss$sum2 == 8),
0,
rep(0, 7),
rep(0, 12),
1,
rep(0, 5),
mean(poss$sum1 == 9 | poss$sum2 == 9),
rep(0, 7),
# 3 to 4
rep(0, 13),
1,
rep(0, 5),
mean(poss$sum1 == 5 | poss$sum2 == 5),
mean((poss$sum1 == 6 | poss$sum2 == 6) & 
    !(poss$sum1 %in% 5) & !(poss$sum2 %in% 5)),
\end{verbatim}
mean((poss$sum1 == 7 | poss$sum2 == 7) &
   !(poss$sum1 %in% 5:6) & !(poss$sum2 %in% 5:6)),
mean((poss$sum1 == 8 | poss$sum2 == 8) &
   !(poss$sum1 %in% 5:7) & !(poss$sum2 %in% 5:7)),
0,
0,
0,
rep(0, 14),
1,
rep(0, 5),
mean(poss$sum1 == 6 | poss$sum2 == 6),
mean((poss$sum1 == 7 | poss$sum2 == 7) &
   !(poss$sum1 %in% 6) & !(poss$sum2 %in% 6)),
mean((poss$sum1 == 8 | poss$sum2 == 8) &
   !(poss$sum1 %in% 6:7) & !(poss$sum2 %in% 6:7)),
0,
0,
0,
rep(0, 15),
1,
rep(0, 5),
mean(poss$sum1 == 7 | poss$sum2 == 7),
mean((poss$sum1 == 8 | poss$sum2 == 8) &
   !(poss$sum1 %in% 7) & !(poss$sum2 %in% 7)),
0,
0,
0,
rep(0, 16),
1,
rep(0, 5),
mean(poss$sum1 == 8 | poss$sum2 == 8),
mean((poss$sum1 == 9 | poss$sum2 == 9) &
   !(poss$sum1 %in% 8) & !(poss$sum2 %in% 8)),
0,
0,
rep(0, 17),
1,
rep(0, 5),
mean(poss$sum1 == 9 | poss$sum2 == 9),
0,
0,
rep(0, 18),
0,
0,
1, rep(0, 5), mean(poss$sum1 == 10 | poss$sum2 == 10), 0,
# 4 to 5 rep(0, 19), 1, rep(0, 5), mean(poss$sum1 %in% 6:11 | poss$sum2 %in% 6:11), rep(0, 20), 1, rep(0, 4), mean(poss$sum1 %in% 7:11 | poss$sum2 %in% 7:11), rep(0, 21), 1, rep(0, 3), mean(poss$sum1 %in% 8:11 | poss$sum2 %in% 8:11), rep(0, 22), 1, rep(0, 2), mean(poss$sum1 %in% 9:11 | poss$sum2 %in% 9:11), rep(0, 23), 0, mean(poss$sum1 %in% 10:11 | poss$sum2 %in% 10:11), rep(0, 24), 1, mean(poss$sum1 == 11 | poss$sum2 == 11), rep(0, 25), 1)

# 3 Sums

# No Restrictions

prob3 <- c(# 0 to 1
1, mean(poss$sum1 == 2 | poss$sum2 == 2 | poss$sum3 == 2), mean((poss$sum1 == 3 | poss$sum2 == 3 | poss$sum3 == 3) & (poss$sum1 >= 3 & poss$sum2 >= 3 & poss$sum3 >= 3)), mean((poss$sum1 == 4 | poss$sum2 == 4 | poss$sum3 == 4) & (poss$sum1 >= 4 & poss$sum2 >= 4 & poss$sum3 >= 4)), mean((poss$sum1 == 5 | poss$sum2 == 5 | poss$sum3 == 5) &
(poss$sum1 >= 5 & poss$sum2 >= 5 & poss$sum3 >= 5)),
mean((poss$sum1 == 6 | poss$sum2 == 6 | poss$sum3 == 6) &
(poss$sum1 >= 6 & poss$sum2 >= 6 & poss$sum3 >= 6)),
mean((poss$sum1 == 7 | poss$sum2 == 7 | poss$sum3 == 7) &
(poss$sum1 >= 7 & poss$sum2 >= 7 & poss$sum3 >= 7)),
rep(0, 19),
# 1 to 2
0,
1,
rep(0, 5),
mean(poss$sum1 == 3 | poss$sum2 == 3 | poss$sum3 == 3),
mean((poss$sum1 == 4 | poss$sum2 == 4 | poss$sum3 == 4) &
(!poss$sum1 %in% 3) & !(poss$sum2 %in% 3) & !(poss$sum3 %in% 3)),
mean((poss$sum1 == 5 | poss$sum2 == 5 | poss$sum3 == 5) &
(!poss$sum1 %in% 3:4) & !(poss$sum2 %in% 3:4) & !(poss$sum3 %in% 3:4)),
mean((poss$sum1 == 6 | poss$sum2 == 6 | poss$sum3 == 6) &
(!poss$sum1 %in% 3:5) & !(poss$sum2 %in% 3:5) & !(poss$sum3 %in% 3:5)),
mean((poss$sum1 == 7 | poss$sum2 == 7 | poss$sum3 == 7) &
(!poss$sum1 %in% 3:6) & !(poss$sum2 %in% 3:6) & !(poss$sum3 %in% 3:6)),
mean((poss$sum1 == 8 | poss$sum2 == 8 | poss$sum3 == 8) &
(!poss$sum1 %in% 3:7) & !(poss$sum2 %in% 3:7) & !(poss$sum3 %in% 3:7)),
rep(0, 13),
rep(0, 2),
1,
rep(0, 5),
mean(poss$sum1 == 4 | poss$sum2 == 4 | poss$sum3 == 4),
mean((poss$sum1 == 5 | poss$sum2 == 5 | poss$sum3 == 5) &
(!poss$sum1 %in% 4) & !(poss$sum2 %in% 4) & !(poss$sum3 %in% 4)),
mean((poss$sum1 == 6 | poss$sum2 == 6 | poss$sum3 == 6) &
(!poss$sum1 %in% 4:5) & !(poss$sum2 %in% 4:5) & !(poss$sum3 %in% 4:5)),
mean((poss$sum1 == 7 | poss$sum2 == 7 | poss$sum3 == 7) &
(!poss$sum1 %in% 4:6) & !(poss$sum2 %in% 4:6) & !(poss$sum3 %in% 4:6)),
mean((poss$sum1 == 8 | poss$sum2 == 8 | poss$sum3 == 8) &
(!poss$sum1 %in% 4:7) & !(poss$sum2 %in% 4:7) & !(poss$sum3 %in% 4:7)),
rep(0, 13),
rep(0, 3),
1,
rep(0, 5),
mean(poss$sum1 == 5 | poss$sum2 == 5 | poss$sum3 == 5),
mean((poss$sum1 == 6 | poss$sum2 == 6 | poss$sum3 == 6) &
   (! (poss$sum1 %in% 5) & ! (poss$sum2 %in% 5) & ! (poss$sum3 %in% 5))),
mean((poss$sum1 == 7 | poss$sum2 == 7 | poss$sum3 == 7) &
   (! (poss$sum1 %in% 5:6) & ! (poss$sum2 %in% 5:6) & ! (poss$sum3 %in% 5:6))),
mean((poss$sum1 == 8 | poss$sum2 == 8 | poss$sum3 == 8) &
   (! (poss$sum1 %in% 5:7) & ! (poss$sum2 %in% 5:7) & ! (poss$sum3 %in% 5:7))),
rep(0, 13),
rep(0, 4),
1,
rep(0, 5),
mean(poss$sum1 == 6 | poss$sum2 == 6 | poss$sum3 == 6),
mean((poss$sum1 == 7 | poss$sum2 == 7 | poss$sum3 == 7) &
   (! (poss$sum1 %in% 6) & ! (poss$sum2 %in% 6) & ! (poss$sum3 %in% 6))),
mean((poss$sum1 == 8 | poss$sum2 == 8 | poss$sum3 == 8) &
   (! (poss$sum1 %in% 6:7) & ! (poss$sum2 %in% 6:7) & ! (poss$sum3 %in% 6:7))),
rep(0, 13),
rep(0, 5),
1,
rep(0, 5),
mean(poss$sum1 == 7 | poss$sum2 == 7 | poss$sum3 == 7),
mean((poss$sum1 == 8 | poss$sum2 == 8 | poss$sum3 == 8) &
   (! (poss$sum1 %in% 7) & ! (poss$sum2 %in% 7) & ! (poss$sum3 %in% 7))),
rep(0, 13),
rep(0, 6),
1,
rep(0, 5),
mean(poss$sum1 == 8 | poss$sum2 == 8 | poss$sum3 == 8),
rep(0, 13),
# 2 to 3
rep(0, 7),
1,
rep(0, 5),
mean(poss$sum1 == 4 | poss$sum2 == 4 | poss$sum3 == 4),
mean((poss$sum1 == 5 | poss$sum2 == 5 | poss$sum3 == 5) &
   (! (poss$sum1 %in% 4) & ! (poss$sum2 %in% 4) & ! (poss$sum3 %in% 4))),
mean((poss$sum1 == 6 | poss$sum2 == 6 | poss$sum3 == 6) & 
  (!!(poss$sum1 %in% 4:5) & !(poss$sum2 %in% 4:5) & !(poss$sum3 %in% 
  4:5))),
mean((poss$sum1 == 7 | poss$sum2 == 7 | poss$sum3 == 7) & 
  (!!(poss$sum1 %in% 4:6) & !(poss$sum2 %in% 4:6) & !(poss$sum3 %in% 
  4:6))),
mean((poss$sum1 == 8 | poss$sum2 == 8 | poss$sum3 == 8) & 
  (!!(poss$sum1 %in% 4:7) & !(poss$sum2 %in% 4:7) & !(poss$sum3 %in% 
  4:7))),
mean((poss$sum1 == 9 | poss$sum2 == 9 | poss$sum3 == 9) & 
  (!!(poss$sum1 %in% 4:8) & !(poss$sum2 %in% 4:8) & !(poss$sum3 %in% 
  4:8))),
rep(0, 7),
rep(0, 8),
1,
rep(0, 5),
mean(poss$sum1 == 5 | poss$sum2 == 5 | poss$sum3 == 5),
mean((poss$sum1 == 6 | poss$sum2 == 6 | poss$sum3 == 6) & 
  (!!(poss$sum1 %in% 5) & !(poss$sum2 %in% 5) & !(poss$sum3 %in% 5))),
mean((poss$sum1 == 7 | poss$sum2 == 7 | poss$sum3 == 7) & 
  (!!(poss$sum1 %in% 5:6) & !(poss$sum2 %in% 5:6) & !(poss$sum3 %in% 
   5:6))),
mean((poss$sum1 == 8 | poss$sum2 == 8 | poss$sum3 == 8) & 
  (!!(poss$sum1 %in% 5:7) & !(poss$sum2 %in% 5:7) & !(poss$sum3 %in% 
  5:7))),
mean((poss$sum1 == 9 | poss$sum2 == 9 | poss$sum3 == 9) & 
  (!!(poss$sum1 %in% 5:8) & !(poss$sum2 %in% 5:8) & !(poss$sum3 %in% 
  5:8))),
rep(0, 7),
rep(0, 9),
1,
rep(0, 5),
mean(poss$sum1 == 6 | poss$sum2 == 6 | poss$sum3 == 6),
mean((poss$sum1 == 7 | poss$sum2 == 7 | poss$sum3 == 7) & 
  (!!(poss$sum1 %in% 6) & !(poss$sum2 %in% 6) & !(poss$sum3 %in% 6))),
mean((poss$sum1 == 8 | poss$sum2 == 8 | poss$sum3 == 8) & 
  (!!(poss$sum1 %in% 6:7) & !(poss$sum2 %in% 6:7) & !(poss$sum3 %in% 
  6:7))),
mean((poss$sum1 == 9 | poss$sum2 == 9 | poss$sum3 == 9) & 
  (!!(poss$sum1 %in% 6:8) & !(poss$sum2 %in% 6:8) & !(poss$sum3 %in% 
  6:8))),
rep(0, 7),
rep(0, 10),
1,
rep(0, 5),
mean(poss$sum1 == 7 | poss$sum2 == 7 | poss$sum3 == 7),
mean((poss$sum1 == 8 | poss$sum2 == 8 | poss$sum3 == 8) &
    (!(poss$sum1 %in% 7) & !(poss$sum2 %in% 7) & !(poss$sum3 %in% 7))),
mean((poss$sum1 == 9 | poss$sum2 == 9 | poss$sum3 == 9) &
    (!(poss$sum1 %in% 7:8) & !(poss$sum2 %in% 7:8) & !(poss$sum3 %in% 7:8))),
rep(0, 7),
rep(0, 11),
1,
rep(0, 5),
mean(poss$sum1 == 8 | poss$sum2 == 8 | poss$sum3 == 8),
mean((poss$sum1 == 9 | poss$sum2 == 9 | poss$sum3 == 9) &
    (!(poss$sum1 %in% 8) & !(poss$sum2 %in% 8) & !(poss$sum3 %in% 8))),
rep(0, 7),
rep(0, 12),
1,
rep(0, 5),
mean(poss$sum1 == 9 | poss$sum2 == 9 | poss$sum3 == 9),
rep(0, 7),
# 3 to 4
rep(0, 13),
1,
rep(0, 5),
mean(poss$sum1 == 5 | poss$sum2 == 5 | poss$sum3 == 5),
mean((poss$sum1 == 6 | poss$sum2 == 6 | poss$sum3 == 6) &
    (!(poss$sum1 %in% 5) & !(poss$sum2 %in% 5) & !(poss$sum3 %in% 5))),
mean((poss$sum1 == 7 | poss$sum2 == 7 | poss$sum3 == 7) &
    (!(poss$sum1 %in% 5:6) & !(poss$sum2 %in% 5:6) & !(poss$sum3 %in% 5:6))),
mean((poss$sum1 == 8 | poss$sum2 == 8 | poss$sum3 == 8) &
    (!(poss$sum1 %in% 5:7) & !(poss$sum2 %in% 5:7) & !(poss$sum3 %in% 5:7))),
mean((poss$sum1 == 9 | poss$sum2 == 9 | poss$sum3 == 9) &
    (!(poss$sum1 %in% 5:8) & !(poss$sum2 %in% 5:8) & !(poss$sum3 %in% 5:8))),
mean((poss$sum1 == 10 | poss$sum2 == 10 | poss$sum3 == 10) &
    (!(poss$sum1 %in% 5:9) & !(poss$sum2 %in% 5:9) & !(poss$sum3 %in% 5:9))),
0,
\[
\begin{align*}
\text{rep}(0, 14), & \\
1, & \\
\text{rep}(0, 5), & \\
\text{mean}(\text{poss}$\text{sum}1 == 6 \mid \text{poss}$\text{sum}2 == 6 \mid \text{poss}$\text{sum}3 == 6), & \\
\text{mean}(\text{poss}$\text{sum}1 == 7 \mid \text{poss}$\text{sum}2 == 7 \mid \text{poss}$\text{sum}3 == 7) & \& \\
(!\text{poss}$\text{sum}1 \text{ in} 6) \& !\text{poss}$\text{sum}2 \text{ in} 6) \& !\text{poss}$\text{sum}3 \text{ in} 6)), & \\
\text{mean}(\text{poss}$\text{sum}1 == 8 \mid \text{poss}$\text{sum}2 == 8 \mid \text{poss}$\text{sum}3 == 8) & \& \\
(!\text{poss}$\text{sum}1 \text{ in} 6:7) \& !\text{poss}$\text{sum}2 \text{ in} 6:7) \& !\text{poss}$\text{sum}3 \text{ in} 6:7)), & \\
\text{mean}(\text{poss}$\text{sum}1 == 9 \mid \text{poss}$\text{sum}2 == 9 \mid \text{poss}$\text{sum}3 == 9) & \& \\
(!\text{poss}$\text{sum}1 \text{ in} 6:8) \& !\text{poss}$\text{sum}2 \text{ in} 6:8) \& !\text{poss}$\text{sum}3 \text{ in} 6:8)), & \\
\text{mean}(\text{poss}$\text{sum}1 == 10 \mid \text{poss}$\text{sum}2 == 10 \mid \text{poss}$\text{sum}3 == 10) & \& \\
(!\text{poss}$\text{sum}1 \text{ in} 6:9) \& !\text{poss}$\text{sum}2 \text{ in} 6:9) \& !\text{poss}$\text{sum}3 \text{ in} 6:9)), & \\
0, & \\
\text{rep}(0, 15), & \\
1, & \\
\text{rep}(0, 5), & \\
\text{mean}(\text{poss}$\text{sum}1 == 7 \mid \text{poss}$\text{sum}2 == 7 \mid \text{poss}$\text{sum}3 == 7), & \\
\text{mean}(\text{poss}$\text{sum}1 == 8 \mid \text{poss}$\text{sum}2 == 8 \mid \text{poss}$\text{sum}3 == 8) & \& \\
(!\text{poss}$\text{sum}1 \text{ in} 7) \& !\text{poss}$\text{sum}2 \text{ in} 7) \& !\text{poss}$\text{sum}3 \text{ in} 7)), & \\
\text{mean}(\text{poss}$\text{sum}1 == 9 \mid \text{poss}$\text{sum}2 == 9 \mid \text{poss}$\text{sum}3 == 9) & \& \\
(!\text{poss}$\text{sum}1 \text{ in} 7:8) \& !\text{poss}$\text{sum}2 \text{ in} 7:8) \& !\text{poss}$\text{sum}3 \text{ in} 7:8)), & \\
\text{mean}(\text{poss}$\text{sum}1 == 10 \mid \text{poss}$\text{sum}2 == 10 \mid \text{poss}$\text{sum}3 == 10) & \& \\
(!\text{poss}$\text{sum}1 \text{ in} 7:9) \& !\text{poss}$\text{sum}2 \text{ in} 7:9) \& !\text{poss}$\text{sum}3 \text{ in} 7:9)), & \\
0, & \\
\text{rep}(0, 16), & \\
1, & \\
\text{rep}(0, 5), & \\
\text{mean}(\text{poss}$\text{sum}1 == 8 \mid \text{poss}$\text{sum}2 == 8 \mid \text{poss}$\text{sum}3 == 8), & \\
\text{mean}(\text{poss}$\text{sum}1 == 9 \mid \text{poss}$\text{sum}2 == 9 \mid \text{poss}$\text{sum}3 == 9) & \& \\
(!\text{poss}$\text{sum}1 \text{ in} 8) \& !\text{poss}$\text{sum}2 \text{ in} 8) \& !\text{poss}$\text{sum}3 \text{ in} 8)), & \\
\text{mean}(\text{poss}$\text{sum}1 == 10 \mid \text{poss}$\text{sum}2 == 10 \mid \text{poss}$\text{sum}3 == 10) & \& \\
(!\text{poss}$\text{sum}1 \text{ in} 8:9) \& !\text{poss}$\text{sum}2 \text{ in} 8:9) \& !\text{poss}$\text{sum}3 \text{ in} 8:9)), & \\
0, & \\
\text{rep}(0, 17), & \\
1, & \\
\text{rep}(0, 5), & \\
\end{align*}
\]
mean(poss$sum1 == 9 | poss$sum2 == 9 | poss$sum3 == 9),
mean((poss$sum1 == 10 | poss$sum2 == 10 | poss$sum3 == 10) &
   (!!(poss$sum1 %in% 9) & !(poss$sum2 %in% 9) & !(poss$sum3 %in% 9))),
0,
rep(0, 18),
1,
rep(0, 5),
mean(poss$sum1 == 10 | poss$sum2 == 10 | poss$sum3 == 10),
0,
# 4 to 5
rep(0, 19),
1,
rep(0, 5),
mean(poss$sum1 %in% 6:11 | poss$sum2 %in% 6:11 | poss$sum3 %in% 6:11),
rep(0, 20),
1,
rep(0, 4),
mean(poss$sum1 %in% 7:11 | poss$sum2 %in% 7:11 | poss$sum3 %in% 7:11),
rep(0, 21),
1,
rep(0, 3),
mean(poss$sum1 %in% 8:11 | poss$sum2 %in% 8:11 | poss$sum3 %in% 8:11),
rep(0, 22),
1,
rep(0, 2),
mean(poss$sum1 %in% 9:11 | poss$sum2 %in% 9:11 | poss$sum3 %in% 9:11),
rep(0, 23),
1,
0,
mean(poss$sum1 %in% 10:11 | poss$sum2 %in% 10:11 | poss$sum3 %in% 10:11),
rep(0, 24),
1,
mean(poss$sum1 == 11 | poss$sum2 == 11 | poss$sum3 == 11),
rep(0, 25),
1)

# Restrictions
prob3 <- c(0 to 1
1,
  mean(poss$sum1 == 2 | poss$sum2 == 2 | poss$sum3 == 2),
  mean((poss$sum1 == 3 | poss$sum2 == 3 | poss$sum3 == 3) &
(poss$sum1 >= 3 & poss$sum2 >= 3 & poss$sum3 >= 3)),
mean((poss$sum1 == 4 | poss$sum2 == 4 | poss$sum3 == 4) &
(poss$sum1 >= 4 & poss$sum2 >= 4 & poss$sum3 >= 4)),
mean((poss$sum1 == 5 | poss$sum2 == 5 | poss$sum3 == 5) &
(poss$sum1 >= 5 & poss$sum2 >= 5 & poss$sum3 >= 5)),
0,
0,
rep(0, 19),
# 1 to 2
0,
1,
rep(0, 5),
mean(poss$sum1 == 3 | poss$sum2 == 3 | poss$sum3 == 3),
mean((poss$sum1 == 4 | poss$sum2 == 4 | poss$sum3 == 4) &
(!poss$sum1 %in% 3) & !(poss$sum2 %in% 3) & !(poss$sum3 %in% 3)),
mean((poss$sum1 == 5 | poss$sum2 == 5 | poss$sum3 == 5) &
(!poss$sum1 %in% 3:4) & !(poss$sum2 %in% 3:4) & !(poss$sum3 %in% 3:4)),
mean((poss$sum1 == 6 | poss$sum2 == 6 | poss$sum3 == 6) &
(!poss$sum1 %in% 3:5) & !(poss$sum2 %in% 3:5) & !(poss$sum3 %in% 3:5)),
0,
0,
rep(0, 13),
rep(0, 2),
1,
rep(0, 5),
mean(poss$sum1 == 4 | poss$sum2 == 4 | poss$sum3 == 4),
mean((poss$sum1 == 5 | poss$sum2 == 5 | poss$sum3 == 5) &
(!poss$sum1 %in% 4) & !(poss$sum2 %in% 4) & !(poss$sum3 %in% 4)),
mean((poss$sum1 == 6 | poss$sum2 == 6 | poss$sum3 == 6) &
(!poss$sum1 %in% 4:5) & !(poss$sum2 %in% 4:5) & !(poss$sum3 %in% 4:5)),
0,
0,
rep(0, 13),
rep(0, 3),
1,
rep(0, 5),
mean(poss$sum1 == 5 | poss$sum2 == 5 | poss$sum3 == 5),
mean((poss$sum1 == 6 | poss$sum2 == 6 | poss$sum3 == 6) &
(!poss$sum1 %in% 5) & !(poss$sum2 %in% 5) & !(poss$sum3 %in% 5)),

111
0,
0,
rep(0, 13),
rep(0, 4),
1,
rep(0, 5),
mean(poss$sum1 == 6 | poss$sum2 == 6 | poss$sum3 == 6),
0,
0,
rep(0, 13),
rep(0, 5),
1,
rep(0, 5),
mean(poss$sum1 == 7 | poss$sum2 == 7 | poss$sum3 == 7),
0,
rep(0, 13),
rep(0, 6),
1,
rep(0, 5),
mean(poss$sum1 == 8 | poss$sum2 == 8 | poss$sum3 == 8),
rep(0, 13),
# 2 to 3
rep(0, 7),
1,
rep(0, 5),
mean(poss$sum1 == 4 | poss$sum2 == 4 | poss$sum3 == 4),
mean((poss$sum1 == 5 | poss$sum2 == 5 | poss$sum3 == 5) &
    !(poss$sum1 %in% 4) & !(poss$sum2 %in% 4) & !(poss$sum3 %in% 4)),
mean((poss$sum1 == 6 | poss$sum2 == 6 | poss$sum3 == 6) &
    !(poss$sum1 %in% 4:5) & !(poss$sum2 %in% 4:5) & !(poss$sum3 %in% 4:5)),
mean((poss$sum1 == 7 | poss$sum2 == 7 | poss$sum3 == 7) &
    !(poss$sum1 %in% 4:6) & !(poss$sum2 %in% 4:6) & !(poss$sum3 %in% 4:6)),
0,
0,
rep(0, 7),
rep(0, 8),
1,
rep(0, 5),
mean(poss$sum1 == 5 | poss$sum2 == 5 | poss$sum3 == 5),
mean((poss$sum1 == 6 | poss$sum2 == 6 | poss$sum3 == 6) &
    !(poss$sum1 %in% 4:5) & !(poss$sum2 %in% 4:5) & !(poss$sum3 %in% 4:6)),
(!(poss$sum1 %in% 5) & !(poss$sum2 %in% 5) & !(poss$sum3 %in% 5)),
mean((poss$sum1 == 7 | poss$sum2 == 7 | poss$sum3 == 7) &
  (!(poss$sum1 %in% 5:6) & !(poss$sum2 %in% 5:6) & !(poss$sum3 %in% 5:6))),
0,
0,
rep(0, 7),
rep(0, 9),
1,
rep(0, 5),
mean(poss$sum1 == 6 | poss$sum2 == 6 | poss$sum3 == 6),
mean((poss$sum1 == 7 | poss$sum2 == 7 | poss$sum3 == 7) &
  (!(poss$sum1 %in% 6) & !(poss$sum2 %in% 6) & !(poss$sum3 %in% 6))),
0,
0,
rep(0, 7),
rep(0, 10),
1,
rep(0, 5),
mean(poss$sum1 == 7 | poss$sum2 == 7 | poss$sum3 == 7),
0,
0,
rep(0, 7),
rep(0, 11),
1,
rep(0, 5),
mean(poss$sum1 == 8 | poss$sum2 == 8 | poss$sum3 == 8),
0,
rep(0, 7),
rep(0, 12),
1,
rep(0, 5),
mean(poss$sum1 == 9 | poss$sum2 == 9 | poss$sum3 == 9),
rep(0, 7),
# 3 to 4
rep(0, 13),
1,
rep(0, 5),
mean(poss$sum1 == 5 | poss$sum2 == 5 | poss$sum3 == 5),
mean((poss$sum1 == 6 | poss$sum2 == 6 | poss$sum3 == 6) &
  !(poss$sum1 %in% 5) & !(poss$sum2 %in% 5) & !(poss$sum3 %in% 5))),
mean((poss$sum1 == 7 | poss$sum2 == 7 | poss$sum3 == 7) &
  !(poss$sum1 %in% 5) & !(poss$sum2 %in% 5) & !(poss$sum3 %in% 5)),
(!\text{poss$\text{sum1} \in 5:6 \& \text{poss$\text{sum2} \in 5:6 \& \text{poss$\text{sum3} \in 5:6})},
\text{mean}((\text{poss$\text{sum1} == 8 \& \text{poss$\text{sum2} == 8 \& \text{poss$\text{sum3} == 8}) \&
  \text{(!poss$\text{sum1 \in 5:7 \& \text{poss$\text{sum2 \in 5:7 \& \text{poss$\text{sum3 \in 5:7}})}}}),
0, 0, 0, \text{rep}(0, 14), 1, \text{rep}(0, 5),
\text{mean}(\text{poss$\text{sum1} == 6 \& \text{poss$\text{sum2} == 6 \& \text{poss$\text{sum3} == 6))},
\text{mean}((\text{poss$\text{sum1} == 7 \& \text{poss$\text{sum2} == 7 \& \text{poss$\text{sum3} == 7}) \&
  \text{(!poss$\text{sum1 \in 6 \& \text{poss$\text{sum2 \in 6 \& \text{poss$\text{sum3 \in 6}})}}}),
\text{mean}((\text{poss$\text{sum1} == 8 \& \text{poss$\text{sum2} == 8 \& \text{poss$\text{sum3} == 8}) \&
  \text{(!poss$\text{sum1 \in 7:8 \& \text{poss$\text{sum2 \in 7:8 \& \text{poss$\text{sum3 \in 7:8}})}}}),
0, 0, 0, \text{rep}(0, 15), 1, \text{rep}(0, 5),
\text{mean}(\text{poss$\text{sum1} == 7 \& \text{poss$\text{sum2} == 7 \& \text{poss$\text{sum3} == 7))},
\text{mean}((\text{poss$\text{sum1} == 8 \& \text{poss$\text{sum2} == 8 \& \text{poss$\text{sum3} == 8}) \&
  \text{(!poss$\text{sum1 \in 7 \& \text{poss$\text{sum2 \in 7 \& \text{poss$\text{sum3 \in 7}})}}}),
\text{mean}((\text{poss$\text{sum1} == 9 \& \text{poss$\text{sum2} == 9 \& \text{poss$\text{sum3} == 9}) \&
  \text{(!poss$\text{sum1 \in 8 \& \text{poss$\text{sum2 \in 8 \& \text{poss$\text{sum3 \in 8}})}}}),
0, 0, 0, \text{rep}(0, 16), 1, \text{rep}(0, 5),
\text{mean}(\text{poss$\text{sum1} == 8 \& \text{poss$\text{sum2} == 8 \& \text{poss$\text{sum3} == 8))},
\text{mean}((\text{poss$\text{sum1} == 9 \& \text{poss$\text{sum2} == 9 \& \text{poss$\text{sum3} == 9}) \&
  \text{(!poss$\text{sum1 \in 8 \& \text{poss$\text{sum2 \in 8 \& \text{poss$\text{sum3 \in 8}})}}),\text{114}
mean(poss$sum1 == 9 | poss$sum2 == 9 | poss$sum3 == 9), 0,
0,
rep(0, 18), 1,
rep(0, 5),
mean(poss$sum1 == 10 | poss$sum2 == 10 | poss$sum3 == 10), 0,
# 4 to 5
rep(0, 19), 1,
rep(0, 5),
mean(poss$sum1 %in% 6:11 | poss$sum2 %in% 6:11 | poss$sum3 %in% 6:11), rep(0, 20), 1,
rep(0, 4),
mean(poss$sum1 %in% 7:11 | poss$sum2 %in% 7:11 | poss$sum3 %in% 7:11), rep(0, 21), 1,
rep(0, 3),
mean(poss$sum1 %in% 8:11 | poss$sum2 %in% 8:11 | poss$sum3 %in% 8:11), rep(0, 22), 1,
rep(0, 2),
mean(poss$sum1 %in% 9:11 | poss$sum2 %in% 9:11 | poss$sum3 %in% 9:11), rep(0, 23), 1,
0,
mean(poss$sum1 %in% 10:11 | poss$sum2 %in% 10:11 | poss$sum3 %in% 10:11),
rep(0, 24), 1,
mean(poss$sum1 == 11 | poss$sum2 == 11 | poss$sum3 == 11),
rep(0, 25), 1)

# Transition Matrices and Mean Turns/Rolls
P1 <- t(matrix(prob1, 26, 26))
P2 <- t(matrix(prob2, 26, 26))
P3 <- t(matrix(prob3, 26, 26))

P1[1, 1] <- 1 - sum(P1[26:1])
P2[1, 1] <- 1 - sum(P2[1:26])
P3[1, 1] <- 1 - sum(P3[1,2:26])
for (i in 2:25) {
    P1[i, i] <- 1 - sum(P1[i,c(1:(i-1), (i+1):26)])
P2[i, i] <- 1 - sum(P2[i,c(1:(i-1), (i+1):26)])
P3[i, i] <- 1 - sum(P3[i,c(1:(i-1), (i+1):26)])
}
solve(diag(25) - P1[1:25, 1:25], rep(1, 25))
solve(diag(25) - P2[1:25, 1:25], rep(1, 25))
solve(diag(25) - P3[1:25, 1:25], rep(1, 25))