1 Introduction

This project looks to analyze the game *Chutes and Ladders*. While the game has traditionally been played exclusively with chutes and ladders, this project expands the historic game to include other aids and obstacles. Since the classic game has seen years of play, producers know how to structure the game rules to ensure the game finishes in a timely manor. With these new aids and obstacles a first question that surfaces is "How long does the game take?" Games like *Monopoly* and *Risk* are notorious for outlasting the patience of their players. To avoid this stigma, this project will look to find an optimal configuration to ensure the game finishes in 50-100 rounds. In addition to finding this optimal configuration, producers might seek to find other rules that either make the game more or less difficult to reach different audiences and improve the longevity of the game. This paper will attempt to find a feasible range of configurations to the aids and obstacles that make the game more and less difficult. Finally, this paper will address the complexity of the program that will simulate *Chutes and Ladders*. In doing so, producers can understand the limitations this simulation places on future experimentation.

2 Approach

2.1 Players

To simplify the construction of the players, three classes were built to manage the player functionality. *PlayerHolder* was a class that simply contained the set of players in the game. This remained constant throughout the simulation, so the playerHolder object was declared static. Within the playerHolder class, an Array List[1] containing objects of type *Player* were stored. Since all access to the players occurred in a sequential process, an Array List was used to simplify access to all players. The *Player* class contained all information the standard player in *Chutes and Ladders* would be privy to. This included a treasure total, a token to move around the board, as well as knowledge of their last roll and the space their token is currently on. Due to the complexity of the object, a player's token was structured as its own class. **Figure 1** summarizes the class structure of the player in this simulation.

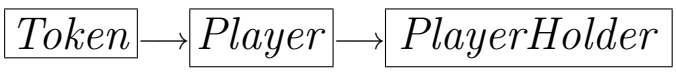

Figure 1: Summary of the Player structure

2.2 Board

The *Board* class has a two dimensional Array List to represent the board. The inner Array Lists were composed of objects of type *Space*. At the beginning of the game, the board is constructed through a standard normal distribution$^{(3,7)}$. Each space has a probability of being one of the six possible spaces. A sample is taken from the standard normal and depending upon the quantile of each space, that type of space is constructed and added to the board. Since the board is constant throughout the game and accessible to all players, the board object was declared static.

The *Space* class was an abstract class that was instantiated through six child classes. These classes include *Empty*, *Hold*, *HoldQ*, *PriorityHold*, *TreasurePotA*, and *TreasurePotB*. *Empty* represents the space that has no actions associated with it. *Hold* represents the least stringent of the obstacles. If a player lands on a Hold space, they must roll the same value as their last roll to leave. Upon landing on this space, the player's number, token, and die roll are all added to a Linked List[4]. If the player rolls the same value these values are removed from the list. A Linked List was used for quick removal of items from the list. *HoldQ* is an obstacle that is similar to the *Hold* space but has queue operations. Instead of every player attempting to leave the space every round, only the player that is at the front of the queue may attempt to leave. A Linked List was used for fast removal of the Token. *PriorityHold* is the final obstacle space. Containing a *PriorityQueue*[5] constructed using a comparator object[2], the players are added to the queue with a priority of their last die roll. Only the player with the highest priority may attempt to leave the queue on any given round. The hold spaces also had a multiplier associated with the space[1]. Once a player left their space, the multiplier was multiplied by the players roll and the player moved that many spaces. This multiplier could have been negative or positive.

The treasure pot classes, *TreasurePotA* and *TreasurePotB*, both held treasure totals that were dispersed to the players as their tokens landed on each space. *TreasurePotA* had a set number of treasure to output with a total number of removals limit. *TreasurePotB* had a treasure total. Players remove their last roll value from the treasure pot until the pot was empty. The values of multipliers and treasure totals were specified in a configuration file that was read in through a *Reader* class. The *Space* class contained a reader object from which the subclasses read the specific values necessary for their construction. **Figure 2** summarizes the board construction.

[1]In this version a multiplier was not fixed. It was randomly selected from a range of numbers at each retrieval of the multiplier.

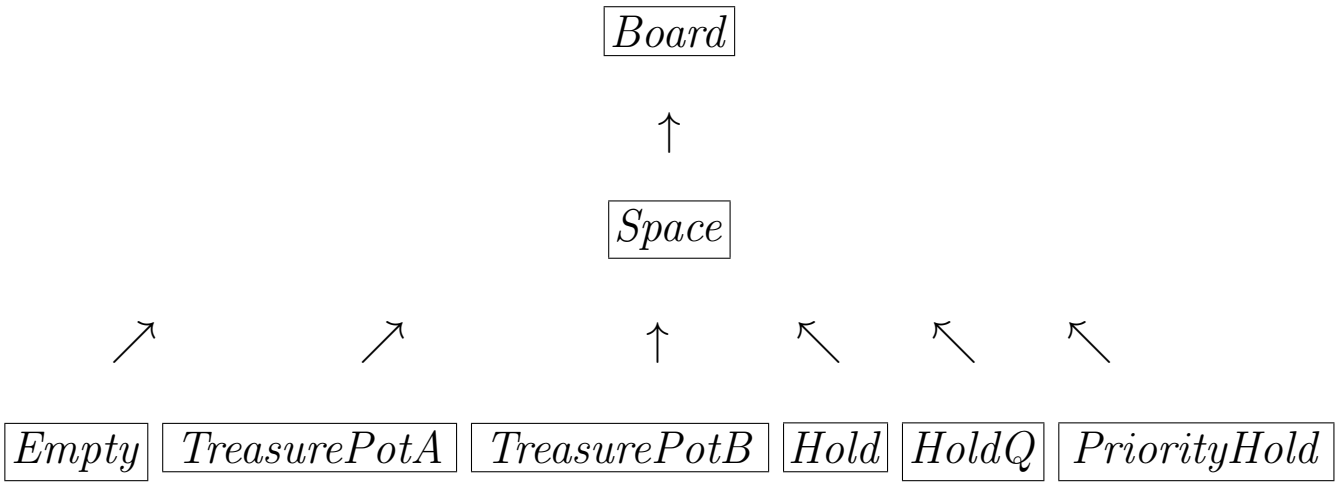

Figure 2: Summary of the Board structure

2.3 Other

In addition to these classes, three other types of classes were used to complete the simulation of the game. The *Dice* class held a die that played the roll of the die in the game. In traditional game play, every player shares the same die. For this reason, a die was declared static for this simulation. For experimental purposes, three classes were used to export data to *CSV* files. These classes were *ExperimentController*, *MyResults*, and *OutputData*. Running experiments in experiment controller, a *MyResults* object was created and then written to a file through an *OutputData* object. All experimentation was controlled by these three classes. A more in depth discussion of the experimentation can be found in Section 3. Finally, a *ChutesAndLadders* class controlled the game run from the terminal. An interactive game can be completed by calling this class through the terminal. Player summaries, board summaries, and round by round progressions are available in this class's main method.

3 Methods

As outlined in the Introduction, this project looks find the optimal configuration of the game board such that the game can terminate between 50 and 100 rounds of play. To analyze this, a simulation experiment was conducted. The three input variables were board size, die number, and number of players. To ensure that no interaction effects were comprised in the experimentation process, a general exploration of the parameter space was devised. Board size varied from 5-50 increasing by 5. Die number was either 6, 8, or 10. Number of players varied from 2-10 increasing by 1. Every possible combination of parameters was tested five times. In total, one experiment consisted of 1215 simulations. From each simulation, the winner, the round total, and the time to complete the simulation was recorded.

To explore how to make the game more or less difficult, this experiment was replicated across four different configuration files. This information will allow for inference regarding how board

configuration altered the game. This experimental design will provide a broad covering of the parameter space in which multiple regression models can predict round totals.

Two additional experiments were devised to test the complexity of this simulation. Holding all other input variables constant, board size was varied from 5-100 by 1 in an attempt to understand the complexity of board size. The same experiment was completed by varying the number of player from 5-100 by 1. These experiments will yield information that is valuable to large scale simulations.

4 Data and Analysis

4.1 Optimal Configuration

To find an optimal configuration of the game board, a broad experiment was run across four different configuration files. Before this experiment was completed, however, data from the complexity experiments were used to visualize the relationship between number of players and board size and the number of rounds required to finish the game. This visualization can be found in **Figure 3**.

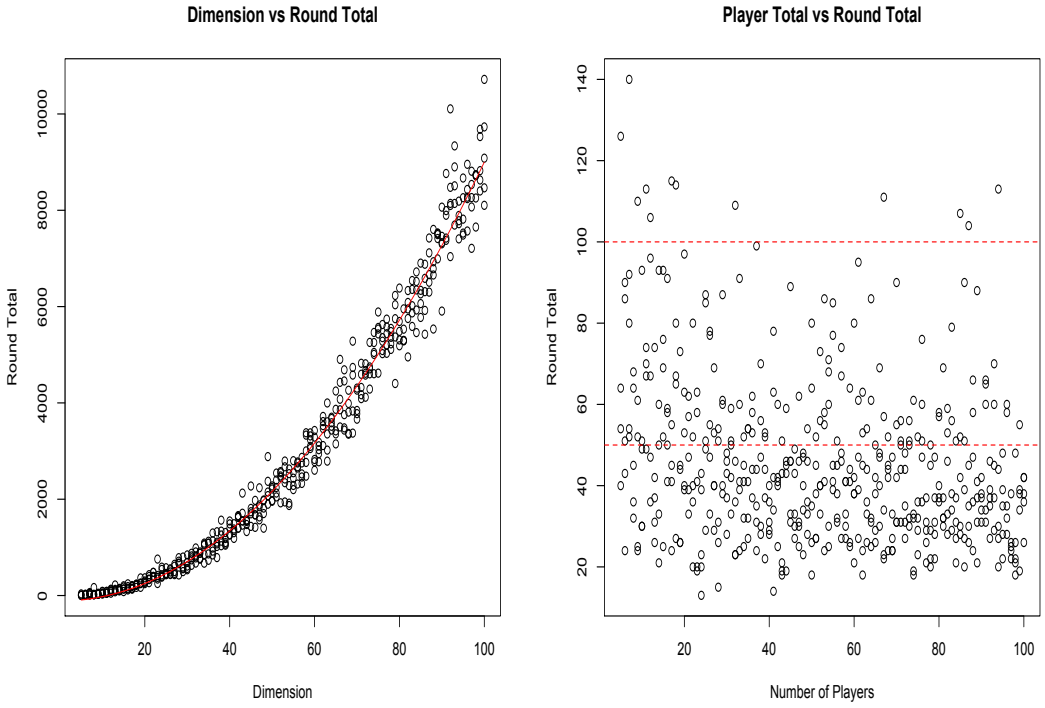

Figure 3: The visualization of board dimension and number of player's effect on round total.

As seen in the figure, we see that the number of players has little effect on the round total in a given game. The region between the dashed red lines represent the "optimal" region. It appears that no number of players will ensure that the game will finish in this optimal region. The board

dimension graph shows a strong correlation between the board dimension and the round total. Note however, that the round total scale ranges from 0 - 10,000. To better visualize the optimal region, a subset plot can be seen in **Figure 4**.

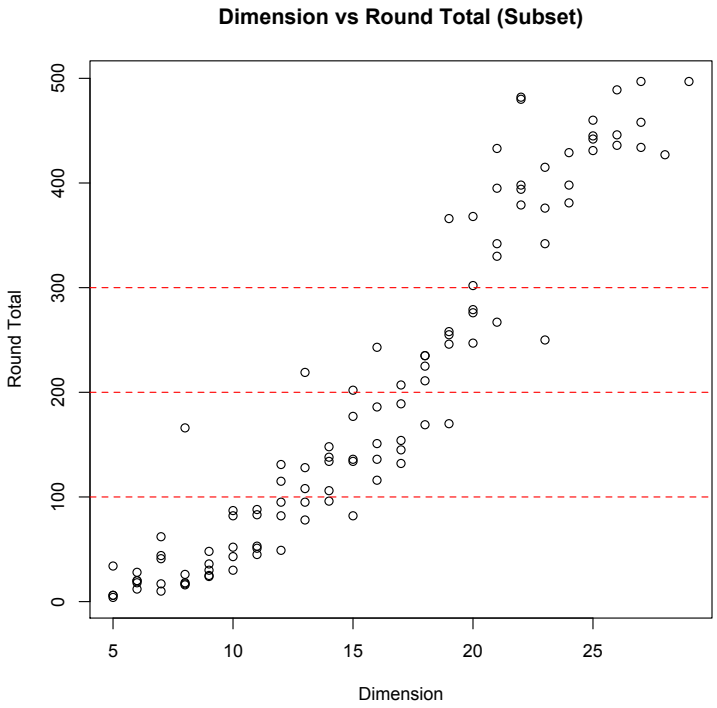

Figure 4: A more granular visualization of Figure 3

The dashed red lines in Figure 4 represent the 100, 200, and 300 round totals. It appears that every board dimension of 15 or greater averages round totals greater than 100. Thus we conjecture that board size cannot surpass the 15 dimension barrier.

Now that a baseline has been established for experimentation, an analysis of the broad experimentation will follow. Recall that the broad experiment included variation of the die number, the board dimension, and the number of players simultaneously. Multiple regression models were constructed to best understand the relationship between the predictor variables and the round totals. No interaction terms were included in the model building process for simplistic interpretation of the regression parameters as well as the independent nature of the input variables.

An initial experiment was completed using the default configuration file. This configuration file is summarized in **Table 1**.

HoldQ	10	2	5
Hold	10	-1	-3
treasurePotA	10	5	50
PriorityHold	10	-2	2
treasurePotB	10	60	

Table 1: Default configuration file

A multiple regression model was built to explain the round total as a function of board dimension, number of players, and the die number. As conjectured above, and supported by a Box-Cox analysis, there is a quadratic trend in the round total data. To correct for this trend, the data was transformed using a square root transform. After making appropriate adjustments, the best model was found to be $\sqrt{roundTotal} = 0.0607 + 0.96 * dimension - 0.31 * playerTotal$. All terms in this model were found statistically significant on the $p = .05$ level with $R^2 = 0.9628$. Notice that die number was not a predictor included in this model. The die number appeared to have no effect on the round total variable. Furthermore, notice that there is a positive coefficient on the board total while a negative coefficient on the number of players term. This backs our intuition. As the board grows in size, the game takes longer to complete. In contrast, as more players are added to the game, the game finishes faster. On the absolute scale, the board size has a greater affect than the number of players. Thus producers should consider focusing on board size if they wish to ensure that their game finished promptly. Residual plots for this model can be found in the Appendix section.

This same procedure was completed for three other configuration files. Those configuration files are summarized in **Table 2**.

HoldQ	10	5	10
Hold	10	-3	-6
treasurePotA	10	5	50
PriorityHold	10	1	5
treasurePotB	10	60	

HoldQ	10	-3	-1
Hold	10	-5	-6
treasurePotA	10	5	50
PriorityHold	10	-5	-1
treasurePotB	10	60	

HoldQ	10	5	10
Hold	10	5	3
treasurePotA	10	5	50
PriorityHold	10	-10	10
treasurePotB	10	60	

Table 2: A summary of the remaining configuration files.

The model for the second configuration file was $\sqrt{roundTotal} = 2.57 + 0.71 * dimension - 0.32 * playerTotal - 0.13 * dieNumber$. All terms in this model were significant on the $p = .05$ level with $R^2 = 0.953$. Residual plots for this model can be found in the Appendix. Notice that this model is very similar to the first model, with some minor changes. First, die number is now included in the model. While still holding an low absolute effect, the model indicated that the large the die number the shorter the game will last. Second, notice how the board dimension coefficient decreased. Having altered the hold parameters, this may have had an affect on this coefficient. Finally, notice how the player number coefficient held the same coefficient regardless of the parameter change. By slightly altering these parameters, changes in the relationship between the input variables and the round total of the game began to change.

The third model was found to be $\sqrt{roundTotal} = 2.15 + 0.48 * dimension - 0.10 * playerTotal - 0.10 * dieNumber$. All terms were significant on the $p = 0.5$ level with $R^2 = 0.76$. This was the weakest of the models. In this experiment, 924 games concluded with no winners. Thus the produces should stay away from this aggressive of a board configuration. The model suggests that very little of the game outcome was explained by the input variables. As a producer wishes to have control, or at least understanding, of how the game should function overtime, the producers should not use this board configuration.

The fourth model was $\sqrt{roundTotal} = 1.43 + 0.58 * dimension - 0.23 * playerTotal$. All terms were significant on the $p = .05$ level with $R^2 = 0.9566$. This model suggest that for this configuration file that board size has less of an effect that the other feasible configuration files. If the produces were looking to release a game in which the player's board was larger than the default, using this configuration file would allow for similar playing time, while changing that parameter of the game.

These models offer insight into how this new version of *Chutes and Ladders* should be structured. By fixing either the number of players, or the board size, an easy computation can be used to solve for the optimal configuration. While this project only inspected four configuration files, by repeating this process on other configuration files, an optimal board configuration can be obtained in little time. This procedure offers a vital tool to game developers.

4.2 Winners

Through experimentation, the winning player was recorded for a board of altering dimension on the default configuration file. From these results, the winning totals were recorded. These results are summarized in **Figure 5**

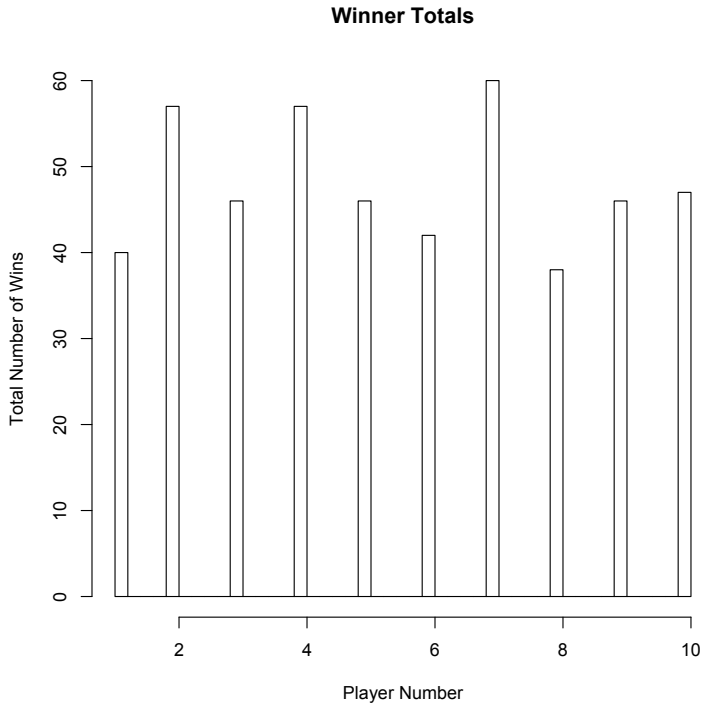

Figure 5: A histogram of wins by player number.

It appears that there is little correlation between the player number and total number of wins. In games like *Monopoly*, player position has a large effect on the game. It appears that there is no correlation in this simulation of *Chutes and Ladders*. A multiple Chi-Squared test was complete to compare these winning totals. The proportion table can be found below in **Table 3**.

Player	1	2	3	4	5	6	7	8	9	10
Winning %	8.4	11.9	9.6	11.9	9.6	8.8	12.5	7.9	9.6	9.8

Table 3: The winning percentage of each player.

A Chi-Squared test was completed to see if there was a difference in these proportions. There was not enough evidence to reject the hypothesis that these proportions were the same. While not evidence for this hypothesis, in this simulation there appears to be no player order effect.

4.3 Complexity

As discussed in section 3, complexity experiments were completed for both the board size and the total number of players. Through completing 495 simulations, the board complexity chart can be found in **Figure 6**.

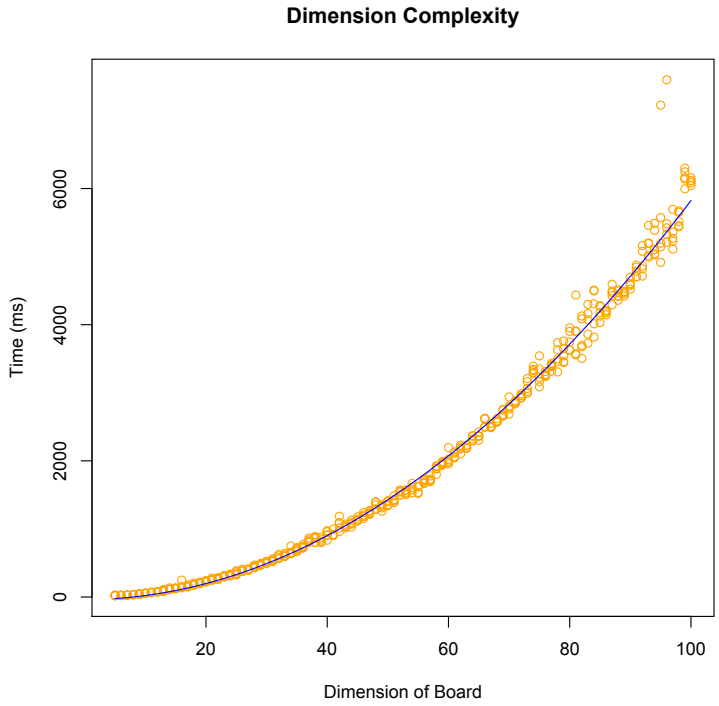

Figure 6: Dimension Complexity $O(n^2)$

A quadratic model was fit to the data and is represented by the blue line in the figure. With $R^2 = 0.978$, the variation in the change in time was explained almost perfectly by board size. As expected, board dimension has complexity $O(n^2)$. With each increase of a board dimension from n to $n+1$, $2n+1$ new spaces are added to the board. This construction of the board and the increased playing time contributes to this quadratic trend. Thus for large scale simulations, board sizes over 1000 should be avoided.

Through a similar experiment, the number of players impact on run time was completed. 495 simulations varying the number of players from 5-100 was completed. The results can be seen in **Figure 7**.

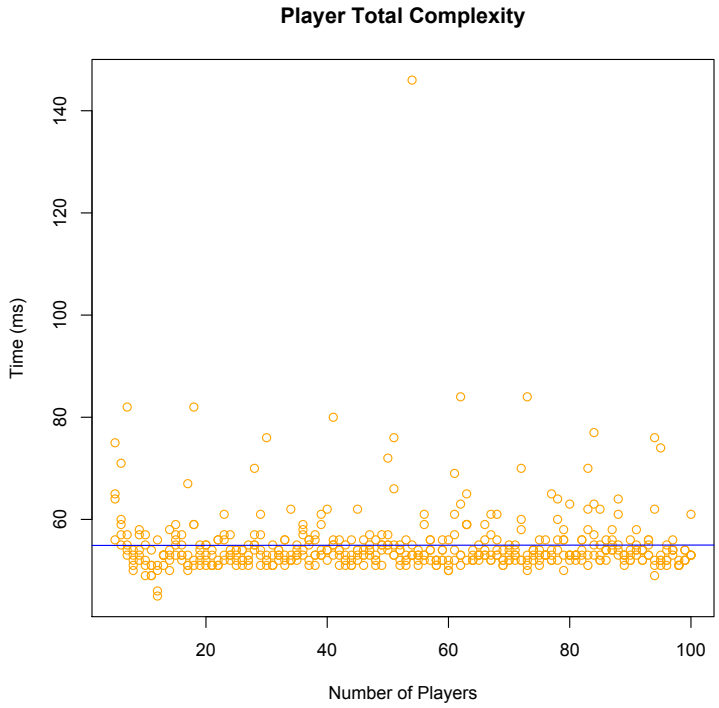

Figure 7: A graph of run time as a function of the number of players.

A linear model was fit to the data and is represented by the blue line shown above. With $R^2 = 9.131 \times 10^{-6}$, this suggests that no variation in the run-time can be explained by the number of players playing the can. From this and the data seen in the figure, we conjecture that the complexity is $O(1)$. That is, it does not depend on the number of players playing the game. This result offers insight for the type of simulations to be completed in the future. While board size should be kept under 1000, player number is free to grow arbitrarily large.

5 Conclusion

This project simulated several combinations of board size, number of players, die number as well as several different board configurations. The complexity of this simulation was found to be $O(n^2)$ for board size and $O(1)$ for number of players. Also it was conjectured that no player order effect occurs in this version of *Chutes and Ladders*. From the multiple regression models, a baseline parameter space was studied. For three configuration files, games ended consistently and a strong model was built to predict the round total of the game. Producers of this game can use these models to find the correct dimensions and suggested number of players. More importantly, perhaps, this

project builds a guideline to test new board configurations to ensure that the game ends in the desired time.

6 References

1. "ArrayList (Java Platform SE 8)." *LinkedList (Java Platform SE 8).* `https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html` N.p., n.d. Web. 19 Feb. 2016.

2. "Comparator (Java Platform SE 8)." *Comparator (Java Platform SE 8).* `https://docs.oracle.com/javase/8/docs/api/java/util/Comparator.html` N.p., n.d. Web. 17 Feb. 2016.

3. Degroot, Morris and Mark Schervish. *Probability and Statistics.* Boston, MA: Addison Wesley, 2002. Print.

4. "LinkedList (Java Platform SE 8)." *LinkedList (Java Platform SE 8).* `https://docs.oracle.com/javase/7/docs/api/java/util/LinkedList.html` N.p., n.d. Web. 10 Feb. 2016.

5. "PriorityQueue (Java Platform SE 8)." *PriorityQueue (Java Platform SE 8).* `https://docs.oracle.com/javase/8/docs/api/java/util/PriorityQueue.html` N.p., n.d. Web. 17 Feb. 2016.

6. "Project Description 1." `http://cs.lafayette.edu/~liew/courses/cs150/` N.p., n.d. Web. 17 Feb. 2016.

7. "Random (Java Platform SE 8)." *Random (Java Platform SE 8).* `https://docs.oracle.com/javase/8/docs/api/java/util/Random.html` N.p., n.d. Web. 12 Feb. 2016.

8. Weiss, Mark Allen. *Data Structures and Problem Solving Using Java.* Reading, MA: Addison Wesley, 1998. Print.

7 Appendix

Residual plots for the models found in section 4.1 can be found below in **Figure 8**. The top left plot represents model 1, the top right represents model 2, and so on.

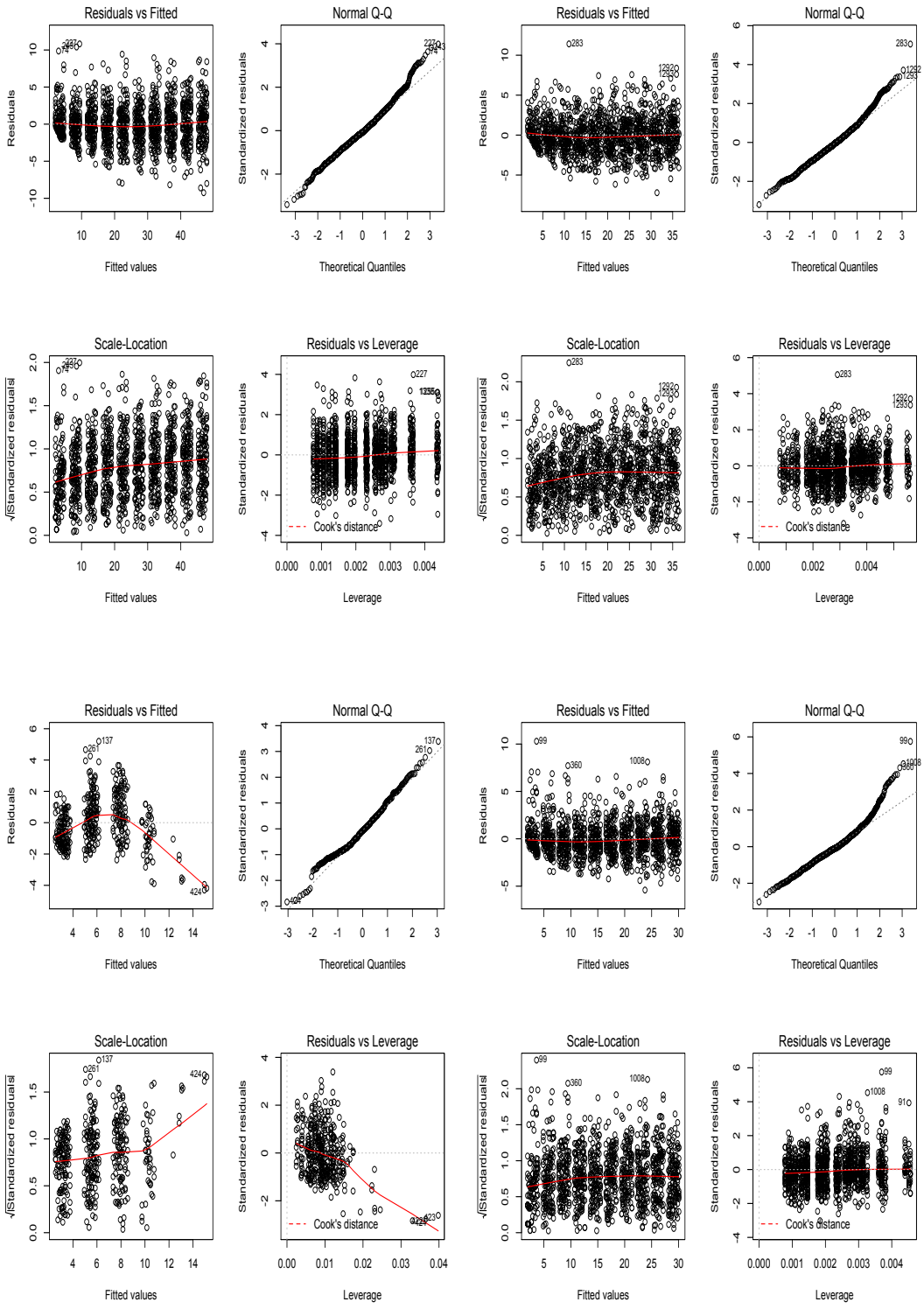

Figure 8: The residual plots for the multiple regression models found in section 4.1