An analysis of TL Wimpout:
A probability study and an examination of game-playing strategies.

By:
Anthony T. Litsch III
TABLE OF CONTENTS

LIST OF FIGURES ................................................................. 3
ABSTRACT .............................................................................. 4
1. INTRODUCTION & BACKGROUND ........................................... 5
   1.1 Objective & Players ...................................................... 5-6
   1.2 The Dice ..................................................................... 5-6
   1.3 Scoring Points .......................................................... 6-8
   1.4 The Rules of Cosmic Wimpout ....................................... 8
2. TL WIMPOUT .................................................................. 9
   2.1 Background for TL Wimpout ....................................... 9
   2.2 Modifications to Cosmic Wimpout ............................... 9-10
   2.3 Research Problem Explained In More Detail .................. 10-11
3. Mathematical Background .................................................. 12
   3.1 Different Approaches to Solve TL Wimpout ................. 12
   3.2 Mathematical Theory .................................................. 12-13
   3.3 Markov Chains .......................................................... 13-16
   3.4 Branching Process .................................................... 16-17
   3.5 Random Walk ........................................................... 17-18
4. The Expected Points Method ............................................... 19
   4.1 Expected Points Method .............................................. 19
   4.2 Explanation of the Method ......................................... 19-20
   4.3 Critical Numbers for 1 Roll of the Dice ....................... 20-21
   4.4 Critical Numbers for 2 Rolls of the Dice ..................... 21-24
   4.5 Transition & Expected Points Matrices ....................... 24-27
   4.6 Conclusion ............................................................... 27-28
5. TL Wimpout Game Simulation ............................................. 29
   5.1 Why do we need Game Simulation? ............................. 29
   5.2 Algorithm for TL Game Simulation ............................. 29-30
      5.2.1 Generating a Roll of the Dice ............................. 30
      5.2.2 Generating a Turn ............................................. 30-31
      5.2.3 How the Game Ends ......................................... 31-32
      5.2.4 Risk Levels ...................................................... 32-33
   5.3 Game Simulation Results .......................................... 33-36
   5.4 Introduction to TL Wimpout Program Modifications ....... 36
      5.4.1 TL Wimpout Program Modification ...................... 37
      5.4.2 Game Strategies Available in TL Wimpout Game Sim.-- 37-39
      5.4.3 Simulation of Game Strategies ............................ 39-40
      5.4.4 Loop Structure ................................................. 40-41
6. Empirical Results ............................................................ 42
   6.1 Hypothesis Testing .................................................... 42-44
   6.2 TL Wimpout ............................................................. 44-47
7. Future Work .................................................................... 48-49
APPENDIX (Game Code) ....................................................... 50-86
Bibliography ................................................................. 87
BIOGRAPHICAL SKETCH ....................................................... 88
LIST OF FIGURES

FIGURE

1. Figure 1. Six sides of a die----------------------------------------------- 6
2. Figure 2. Six sides of fifth die----------------------------------------- 6
3. Figure 3. Flashes with associated points------------------------------- 7
4. Figure 4. Freight Trains with associated points------------------------- 7
5. Figure 5. Beginning of a turn---------------------------------------- 34
6. Figure 6. The “Wimpout” message------------------------------------- 35
7. Figure 7. End of game message----------------------------------------- 36
ABSTRACT

An analysis of 3-Dice Cosmic Wimpout: A probability study and an examination of game-playing strategies.

By:
Anthony T. Litsch III
May 2006
Advisor: Dr. Will Miles
Department: Mathematics and Computer Science

In this paper we explore a variant of the dice game Cosmic Wimpout. Our revision of the game consists of using three dice instead of five dice. Modifications were made to the new version of Cosmic Wimpout, called TL Wimpout, in order to narrow the problem of study. The focus is to mathematically find an ideal game-playing strategy that would optimize the likelihood of winning TL Wimpout. A computer simulation of the new Cosmic Wimpout has been created. Expected points was the method of choice in finding an ideal game playing strategy. We came close to finding the actual solution, however we were only able to find a reliable approximation to the three critical numbers which give an almost ideal game playing strategy.
CHAPTER 1
INTRODUCTION & BACKGROUND

This paper analyzes a modified version of the dice game called Cosmic Wimpout. Modifications were made to Cosmic Wimpout, resulting in a new game called TL Wimpout. The focus is to mathematically find an ideal game-playing strategy that would optimize the likelihood of winning TL Wimpout. A computer simulation of TL Wimpout has been created. We have begun the study of advanced probability theory. Stochastic processes, notably Markov Chains, seem applicable to the problem.

The paper is organized as follows. Chapter 1 discusses the original Cosmic Wimpout in depth. Chapter 2 discusses TL Wimpout in depth. Chapter 3 deals with the TL Wimpout game simulation and algorithm. Chapter 4 deals with the mathematical theories used in the analysis of TL Wimpout. Chapter 5 provides the goals for future research.

Since TL Wimpout is based on Cosmic Wimpout, a thorough description of Cosmic Wimpout seems appropriate. Such a description follows immediately.

1.1 Objective & Players

The goal when playing Cosmic Wimpout is to be the first player to obtain a prescribed number of points, typically 500 points. However before the game is started, the players can choose to set a different point level at which the game ends.

Cosmic Wimpout is a very accommodating game, in the sense that as many people who want to play may play. A minimum of two players are needed. For the purpose of this research, only two players will play TL Wimpout.

1.2 The Dice
Cosmic Wimpout is a dice game composed of five six sided dice. To play the
game, all that is required is a piece of paper, the rules, and the five dice. Four of the five
dice are identical. Each die contains two moons, three triangles, four lightning bolts, the
number 5, six stars, and the number 10. (See Figure 1)

![Figure 1. Six sides of a die][5]

Each picture representation on a given side of the cube can be thought of as just the
number of picture representations on the side of the cube. For example, four lightning
bolts can be viewed as 4. The only notable importance of picture representations on each
side of the dice, to the game Cosmic Wimpout, is the fact that the picture representations
make the game unique and the game dice unique. The fifth die is almost identical to its
four counterparts. One difference is the fifth cube, is black, while the other four are white.
A more important difference is the fifth cube does not have a three triangle picture
representation on one of its sides. In place of three triangles is the picture representation
of a Flaming Sun. (See Figure 2)

![Figure 2. Six sides of fifth die][5]

The flaming sun can be thought of as a wild side of the fifth cube. A flaming sun can be
considered by a player to be a 5, a 10, or count as nothing.

1.3 Scoring Points

There are a few different ways a player may score points. The game rules affect
scoring and will be discussed in the next section.
The simplest way to score points is to roll a 5, a 10, or a flaming sun. The other sides of a cube do not score by themselves. Rolling a 5 or 10 counts as the respective face value of the die. Recall a flaming sun may be counted as 0, 5, or 10.

Another way to score points is to roll three of a kind, i.e. three dice with the same side facing upwards. A three of a kind is referred to as a *flash*, and counts as ten times the number of which the three of a kind is made. (See Figure 3)

![Figure 3. Flashes with associated points](image)

Rolling five of a kind, is referred to as a *freight train*, and counts as one hundred times the number of which the five of a kind is made up from. (See Figure 4)

![Figure 4. Freight Trains with associated points](image)
One exception is worth noting. Rolling a freight train of six stars results in automatically winning the game. Rolling a freight train of the number 10 results in automatically losing the game. The last result is referred to as a *Supernova*. A final way to score points is to roll a pair, i.e. two dice with same side facing upwards, in conjunction with the flaming sun. Such a combination counts as a flash. It is worth noting pairs do not score nor does four of a kind. Rolling a pair of 5’s or a pair of 10’s, count as $5 + 5 = 10$ points, or $10 + 10 = 20$ points.

1.4 The Rules of Cosmic Wimpout

Cosmic Wimpout has three basic game rules.

1. **Get Into Game Rule.** A player must roll, score, and be able to keep 35 points in order to get “into” the game. The 35 points must be a result of a single turn. Until 35 points are obtained and kept in a single turn, all other points accumulated in any turn do not count in favor of a player. The points are simply lost.

2. **The Futless Rule.** This rule indicates that a player must continue rolling any non-scoring dice after a flash is rolled. For example, a player rolls all five dice. A flash of two moons is observed along with a pair of six stars. The flash cubes are set aside, and the pair of six stars, since they did not score, must be re-rolled.

3. **You May Not Want To But You Must Rule.** This rule indicates that a player must re-roll all five dice, if all five dice have scored in a single roll or a combination of rolls in a single turn.

If, at any time, a player does not score any points on a roll, the player has *Wimped Out*. Wimping Out causes a player to lose all points accumulated in that turn. In contrast, if a player has scored points in a roll and none of the three rules mentioned above apply, then the player has an option to stop and take his or her points, or to re-roll any non-scoring dice.
CHAPTER 2
TL WIMPOUT

To understand TL Wimpout, it is necessary to describe the changes to Cosmic Wimpout. Also the reasons for such changes will be discussed.

2.1 Background for TL Wimpout

The idea for TL Wimpout came from exploration of Cosmic Wimpout. TL represents the initials of the game’s founder, Tony Litsch. Cosmic Wimpout contains a wealth of problems worthy of mathematical exploration. However, using five six sided cubes results in 7,776 first roll combinations. Depending on the outcome of the first roll, there will be a large, yet finite, number of possible outcomes for the second roll. Clearly, trying to exhaust all first and second roll possible outcomes becomes quite tedious. TL Wimpout provides only 216 first roll outcomes, and far fewer second roll outcomes. It is the hope that by providing a solid foundation with three dice this research may be extended and applied to five dice.

2.2 Modifications to Cosmic Wimpout

1. Cosmic Wimpout is played using 5 dice. TL Wimpout is played with only 3 dice, none of which are the die containing the Flaming Sun.

2. Get Into Game Rule is eliminated altogether.

3. The You May Not Want to But You Must Rule is eliminated altogether. The new modification is you can stop at any time and take the points you have earned in the current turn, except for in the case of modification #4.

4. The Futless Rule which stipulates you must re-roll after a flash, is not kept in place.

5. There are no Freight Trains or Supernovas in the game, since five dice are no longer being used, it is not possible to roll either a freight train or a supernova. The highest amount of points in a single roll is a flash of 10’s.
6. Flashes, or 3 of a kind, count the same. No modifications to point schemes of flashes.

7. A modification to points is that pairs count as half of a flash. The revised point scheme is given below.

\[
\begin{align*}
2 & \times 2 = 10 \text{ points} \\
3 & \times 3 = 15 \text{ points} \\
4 & \times 4 = 20 \text{ points} \\
5 & \times 5 = 25 \text{ points} \\
6 & \times 6 = 30 \text{ points} \\
10 & \times 10 = 50 \text{ points}
\end{align*}
\]

In the simulation of TL Wimpout, discussed in Chapter 3, 7 7 is used in place of 10 10.

8. Fives and tens count as face value unless there is a pair or a flash of either.

9. Sides of two moons, three triangles, four lightning bolts, and six stars do not score by themselves, only in pairs and flashes.

10. The first player to reach the 250 point level wins the game.

11. Wimping Out is preserved in the new game version. At any time if no points are earned on a roll, all points accumulated up to that roll on the present turn are lost. This is called wimping out. Points earned from previous turns are not lost.

12. No other derivations or rules exist beyond those mentioned above.

### 2.3 Research Problem Explained In More Detail

The main focus of this research is to find the ideal game strategy a player should use in order to maximize his/her chance of winning TL Wimpout. In order to find the ideal solution, some questions must be answered. Questions include the following:

- What are the probabilities for a roll of the dice using 3 cubes, 2 cubes, or 1 cube?
- Which stochastic process or Markov Chain is best suited for the research? Why?
- Can we program the mathematical model into our game simulation? How will it be done?
- How can we use the probability of an event occurring in the future to
find the expected number of points in a future roll?
- How can mathematics be used to support expected number of points?
- Given any roll at any particular time in the game, what is the probability of getting any particular roll on the next roll of the dice?

To investigate these and future questions, it is necessary to research and become familiar with stochastic processes. In particular, Markov Chains and specific examples of Markov Chains such as branching processes and random walks will be used to answer such questions. After gaining a sound understanding of such mathematical models, the probabilities of a roll or a certain number of points given a particular roll will be able to be calculated.

Lastly a game simulation program needs to be created to implement the logic behind TL Wimpout. The logic will be based on the theoretical findings mentioned earlier, and will include modifications to Cosmic Wimpout discussed in Chapter 2. The simulation may be used to empirically validate the theoretical results or suggest particular strategies in the absence of theoretical results.

After finding probabilities, understanding and applying mathematical models, and implementing a game simulation, we hope to determine an ideal game strategy for TL Wimpout.
CHAPTER 3
MATHEMATICAL BACKGROUND

3.1 Different Approaches To Solve TL Wimpout

We begin defining the solution to TL Wimpout. We define a solution to the game to be a strategy which would maximize a player’s probability of winning the game. There are many different methods that might be used in order to solve TL Wimpout. In this section, we discuss a few of the options. One possible method is to minimize the number of turns in the game. We have also investigated Markov Chains, Branching Process, and Simple Random Walks as possible methods to solve for an ideal game strategy. However, we have decided to use a method which employs expected points gained to find an ideal game strategy. We will discuss this “expected points” method in detail later in this paper, but we first present some background on the previously mentioned methods: Markov Chains, Branching Process, and Simple Random Walks.

3.2 Mathematical Theory

In this section we present Markov Chains, Branching Process, Simple Random Walks. We begin by introducing a discrete stochastic process. A discrete stochastic process is a collection of random variables typically \( \{X_0, X_1, X_2, \ldots\} \), indexed by an ordered time parameter, all defined on the same probability space [1]. An example would be how many points a player has scored throughout a basketball season or the GPA of a Math student as he or she works throughout the year. In the basketball example, \( X_0 \) would be 0 (the player begins the season with 0 points). Then, \( X_1 = \) points after the first game, \( X_2 = \) points after the second game, and so on. Thus, the index designates the number of games played. A student’s GPA changes from graded assignment to graded assignment.
1. TL Wimpout can be thought of a stochastic process where random variables can be thought of as the number of points and the time parameter can be thought of as rolls of the dice. A very useful stochastic process is a process known as Markov chain. Later in the chapter we will present two specific applications of Markov chains: Branching Processes and Random Walks. However, first we discuss Markov chains in more detail.

### 3.3 Markov Chains

We begin with a formal definition of Markov chains. Let $S = (s_1, s_2, \ldots)$ be a set of states, and $(X_n : n = 0, 1, \ldots)$ be random variables such that $P(X_n \in S) = 1$, for all $n$. Suppose also that, for any $n$, times $m_1 < m_2 < \ldots < m_n$ and states $s_1, s_2, \ldots, s_n$, the probability $P(X_{m_n} = s_n | X_{m_1} = s_1, \ldots, X_{m_{n-1}} = s_{n-1}) = P(X_{m_n} = s_n | X_{m_{n-1}} = s_{n-1})$. This property is called the Markov property. A process with these characteristics is called a Markov chain [1]. We generally denote a Markov chain by $(X_n, S)$. A Markov chain makes use of conditional probabilities. We want to find the probability of going to state $j$, given we start in state $i$. The probability of reaching state $j$ at the next time step is dependent only on the current state of the process. Any previous path of states taken to get to the current state, $i$, is irrelevant with regards to finding the probability of reaching the next state, $i+1$. Since the Markov property is defined in terms of conditional probabilities, we now move our attention to a discussion of such probabilities.

The conditional probability of $A$, given that $B$ has occurred is defined as $P(A|B) = \frac{P(A \cap B)}{P(B)}$, provided $P(B) > 0$. Thus the probability of event $A$ happening is affected by the fact that event $B$ has already taken place, if event $A$ is dependent on event $B$ taking place [7]. The probability of both event $A$ and event $B$ occurring can be found by
multiplying both sides of our conditional probability equation by \( P(B) \), thus giving \( P(A \cap B) = P(B)P(A|B) \) [7].

Events A and B are dependent events, since the probability of event A occurring is conditioned upon event B already occurring. This definition is important to TL Wimpout since this is the manner in which conditional probabilities will be calculated. When finding the probability of moving from state i to state j, in TL Wimpout, we are finding the probability \( P(A \cap B) = P(B)P(A|B) \). The probability of event A occurring and event B occurring is denoted by \( P(A \cap B) \). We know \( P(B) \) and \( P(A|B) \), and thus can find \( P(A \cap B) \).

In terms of Markov processes, TL Wimpout seems to include embedded Markov chains. The inner chain occurs during a player’s turn. The state space of this chain would include the possible accumulated points within the turn. The outer chain deals with the overall status of the game. The state space of this chain would include the total number of points a player has at the end of each turn. Next we discuss the idea of transitioning between states.

The *transition probability*, \( p_{ij} \), is defined to be \( P(X_n = j \mid X_0 = i) \) [1]. The transitional probability \( p_{ij} \) is the conditional probability of moving from state i to state j. The exponent \( n \) denotes how many *steps*, or intermediate states, that must be passed through to get from state i to state j. A matrix containing all of the transitional probabilities is called a *transitional matrix*. The \((ij)^{th}\) entry of the matrix is \( p_{ij} \) [1]. Thus if there are \( n + 1 \) states, the transitional matrix \( p \) is given by:
All entries in the matrix $p$ are greater than zero and the sum of all the entries in each row equals 1 [1]. The row sum is equivalent to summing all the probabilities of a probability space. The sum must be 1 since all possible outcomes are accounted for. A matrix $P$ is called an *n-step matrix* when all entries are $p_{ij}^{[n]}$ [1]. This is the situation of given our current number of points, what is the probability we will gain $x$ number of points in $n$ rolls of the dice.

Now let’s discuss what it means to be a stationary distribution. Suppose $w = (w_i)$ where $w_i = P(W = i)$, and $w_i > 0$ and $\sum w_i = 1$. Then if $wP = w$, $w$ is said to be a *stationary* or an *equilibrium distribution* for $P$, where $P$ is the transition matrix [1]. The distribution is said to be stationary because it does not change. If $w$, the distribution of a state, is given, then the distribution for that state stays fixed at $w$. Consider the following sequence, $(p_{ij}^{(n)}) \rightarrow \pi_j$ as $n \rightarrow \infty$ and $\pi = (\pi_j)$ satisfies $\pi P = \pi$. The long term probability of being in state $j$ is equal to $\pi_j$ [1]. A sequence of $n$-step matrices, i.e. $(P^n)$, can be constructed. Finding the limit as the sequence approaches infinity gives the stationary distribution, denoted here by $\pi$. In this situation, being in state $j$ is a known probability, and is not dependent on the prior state. This remark is quite useful for TL Wimpout since
if we can find the stationary distribution for each roll of the dice, it will give a solid foundation for beginning to build an ideal game strategy.

Markov Chains are applicable to TL Wimpout since each roll of the dice is independent of everything except the current roll. The next roll is dependent upon the most recent roll since when ever a die scores it is put aside. For instance rolling three dice and scoring with two of them, leaves one die left. The next roll involves only one die, not three, since the previous roll used two dice, thus affecting the number of dice to be rolled in the subsequent roll.

Being able to calculate the probability of the next roll assists in finding the expected number of points in the next roll. Knowing the expected number of points in a roll assists in determining an ideal game strategy to employ when playing TL Wimpout which we will discuss shortly.

3.4 Branching Process

Consider the following situation, the passing of a family’s last name down from one generation to the next generation. The problem starts with one person, call him X. Since our society is patriarchal, a man’s last name is passed down to his son(s), and their son(s) and so on and so forth. The question becomes in the nth generation how many ‘sons’ have the last name. This is analogous to the question of how many sons, grandsons, great grandsons, great great grandsons, etc. X has. The nth generation is the number of descendents in that generation who traced their lineage back to X. The number of ancestors in the nth generation is denoted by $X_n$, where $X_0 = 1$, or X [1]. This is an example of a branching process.

A branching process might be applicable to TL Wimpout because it allows for
long term probabilities. Each generation may be thought of as a roll of the dice. Thus, instead of just finding the expected number of points on the next roll with a simple Markov chain, a branching process would allow finding the expected number of points of a turn given the first roll of the turn. Using Markov chains and a specific form of Markov chains, branching processes, together might offer the most useful method for determining an ideal game strategy for TL Wimpout, but further research will probably be needed.

3.5 Random Walk

We start off by defining a (simple) random walk. Then we discuss an application of a random walk, the famous Gambler’s Ruin problem along with a possible application to TL Wimpout.

Let \((X_n)\) be independent identically distributed random variables, or \(iidrv\), with \(P(X_n = 1) = p\) and \(P(X_n = -1) = q = 1-p\), and let \(S_0 = 0\), \(S_n = X_1 + \ldots + X_n\) for \(n \geq 1\). The values \((S_n)\) are said to form a simple random walk [1]. A random walk is simple because the steps along the walk, denoted by \(X_n\), can only accept values of +1 and -1 [1].

A famous problem that is modeled by random walks is the Gambler’s Ruin problem. The gambler’s ruin problem requires a modification to a simple random walk. Given \(c > 0\), let \(0 \leq a \leq c\), and suppose \(S_0 = a\) : we seek the probability that \(S_n = 0\) before \(S_n = c\). We are modeling a gambler with initial fortune \(a \geq 0\) playing an opponent whose initial fortune is \(c-a \geq 0\). At each stage in the game, the gambler either wins or loses some unit amount from the opponent with probabilities \(p\) and \(q\). The game ends when either the gambler or the opponent has a fortune of size 0. The focus is usually on the gambler reaching 0 first, hence the Gambler’s Ruin Problem, i.e. \(p_a = P(S_n = 0 \text{ before } S_n = c \mid S_0 = a) = P(\text{Ruin} \mid S_0 = a) [1].\)
A Random Walk might be applicable to TL Wimpout in the sense of a roll resulting in points or a roll resulting in a wimpout. The game strategy could be based on some modification to the Gambler’s Ruin problem, allowing an analysis of how long it takes to reach the win level, i.e. how many steps, but further research is required. We now discuss in more depth our chosen method of approach, expected points.
CHAPTER 4
THE EXPECTED POINTS METHOD

4.1 Expected Points Method

The expected points method was chosen due to time. This method did not require the additional time to research it further as Markov Chains, Branching Processes, and Simple Random Walks needed. Since time restraints are part of the research, it was appropriate to make use of the method we felt most knowledgeable about since this would let us begin the hands on research much sooner.

4.2 Explanation of the Method

In this chapter we explain how expected value calculations aid in the determination of a game-playing strategy. The expected value of a variable X is given by \( \sum xf(x) \), where x ranges over all possible outcomes and f(x) is probability of X=x [7]. The fundamental idea on which this method is based is as follows. If the expected value of points gained on the next roll is positive, then the player should roll again. This expected value is affected greatly by the number of points currently gained during the turn since those points are points which could be lost on a future roll of the dice. Thus, regardless of the probabilities of wimping out, there is some number of points at which the expected value of points to be gained will become non-positive. We call this crucial number of points a critical number of points. We illustrate the ideas of this method with an example. Suppose a player currently has x points and has one die with which he may continue his turn. If he rolls the die, he may wimp out, he may score five additional points, or he may score 10 additional points. Then the expected value of gained points if he were to roll the remaining die would be given by:
\[ E = (4/6)(-x) + (1/6)(5) + (1/6)(10). \]

In the above calculation, \( (4/6) = P(\text{wimping out}) \), \( (1/6) = P(\text{gaining 5 points}) \), and \( (1/6) = P(\text{gaining 10 points}) \). The strategy would indicate that the player should roll the die if \( E > 0 \). Thus, we solve the equation \( E = 0 \) for \( x \) to find the critical value associated with this situation. Critical values would also be needed for the situations involving two non-scoring dice and three non-scoring dice. Furthermore, as we look beyond one roll out, the expected value calculations change, as well as the corresponding critical numbers. If you use the above critical numbers and roll again when it is deemed suitable, the expected values when rolling two more times will differ from those just calculated. The same goes for rolling three more times, four more time, etc. If critical numbers could be found for rolling \( n \) more times, the true theoretical solution associated with this strategy would be the limits of these critical numbers as \( n \) approached infinity, provided the limits exist. In the following discussion, we find the critical numbers associated with rolling up to five more times. We draw some conclusions based on these results.

**4.3 Critical Numbers for 1 Roll of the Dice**

We first demonstrate how the expected points after one roll of the dice are found. In general, each possible point value is multiplied by its associated probability of occurring. The sum of these products is the expected points for that particular rolling of the dice. Below are the calculations for finding the critical numbers of one roll of the dice. The number for each dice situation indicates that if current earned points are less than the critical number it is best to roll again. Current earned points are greater than or equal to the critical number it is best to not roll again. The critical number is the number that results in each expected value equaling zero.
The above calculations indicate the critical numbers when rolling on more time are 3.75 when one die may be rolled, 25 when two dice may be rolled, and 153.125 when three dice may be rolled.

4.4 Critical Numbers for 2 Rolls of the Dice

To find the critical numbers associated with rolling the dice two times, we will start with our expected points equations and extend the points and probabilities to include a second roll. Below is the equation used for finding the critical numbers after two rolls of the dice starting with one die.

We find that the critical number of points after two rolls of the dice starting with one die gives a value of 11.2171. Next we turn our attention to the expected number of points after two rolls starting with two dice. The calculation is quite long. The calculation would be longer if the ‘risky’ situations were not removed beforehand. Whenever a player
is faced with less than three dice with which to roll, the situation is considered *risky*, based on the critical numbers found for one and two rolls of the dice. For example, if we roll two dice we have two possible first roll outcomes: score with one die or score with both dice. If we score with both dice we then have all three dice to roll with for our next, second, roll of the dice. However if we score with one die, we are left with one die with which to roll for our next roll of the dice. The likelihood of wimping out with one die or two dice is much greater than the probability of wimping out with three dice. For this reason, anytime we have only one die or two dice after a roll we choose to stop because the situation has become to risky. On the other hand, if we start and end a roll with all three dice, we will choose to roll again since this is the least risky situation of all three possible situations.
The critical number after two rolls starting with two dice gives an expected point value of 32.2845. The critical number for three dice is 148.363. It is tedious but we could continue to find the critical numbers after 3 rolls of the dice, and 4 rolls out to n rolls of
the dice. However there is a shorter procedure that we have discovered to find the sought after critical values of 3 rolls, 4 rolls, etc. The procedure makes use of the *transition* and *expected points* matrices which we now turn our attention towards.

### 4.5 Transition & Expected Points Matrices

Now that we have demonstrated how the expected points after a fixed number of rolls can be found, we now introduce the transition and expected points matrices, shown below.

**Transition Matrix**
- Probability of going from *n* to *m* dice.

<table>
<thead>
<tr>
<th>n</th>
<th>m</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>2/6</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
<td>8/36</td>
</tr>
<tr>
<td>84</td>
<td>72</td>
<td>36/36</td>
</tr>
</tbody>
</table>

**Expected Points Matrix**
- Average points scored going from *n* to *m* dice.

<table>
<thead>
<tr>
<th>n</th>
<th>m</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>7.5</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
<td>22.5</td>
</tr>
<tr>
<td>84</td>
<td>72</td>
<td>33</td>
</tr>
</tbody>
</table>

Each row represents the current number of dice to be rolled. Each column represents the number of dice which may be rolled during the next toss of the dice. For instance, column 1 deals with going from 1 die to 1 die, 2 dice to 1 die, and 3 dice to 1 die. The transition matrix is found by summing up all the possible ways of scoring that allow for transitioning between *n* and *m* dice. Observe the expected points matrix does not contain the same numbers as were found using the expected points equations shown earlier. The values in the expected points matrix are found under the assumption that the next roll does not result in a wimpout. Thus, the probabilities used to find the entries in the expected points matrix are conditioned on the event that a wimpout does not occur. The
expected values are then found via \( \sum (\text{number points})P(\text{points}|\text{no wimpout}) \) where the sum ranges over all possible point values which may be scored. However, we can use the transition matrix and the expected points matrix together to produce the same critical numbers as were found earlier by using the expected points equations, shown below.

\[
\begin{align*}
1 \text{ die situation} & \quad \frac{2}{6} \quad 3.75 \\
2 \text{ dice situation} & \quad \frac{12}{36} \quad 0 \quad 2 \quad 12 \quad 0 \quad 120 \quad 180 \quad 153.125 \\
3 \text{ dice situation} & \quad \frac{24}{216} \quad 0 \quad 0 \quad 36 \quad 72 \quad 108 \quad 153.125
\end{align*}
\]

The values in each equation now come strictly from the transition matrix and the expected points matrix. Consider the 1 die situation. There is a 2/6 probability of scoring with 1 die and the expected (average) points scored is 15/2. There is also a 4/6 probability of wimping out, which is the complement to the sum of all the probabilities in each row. Solving the expected points equation gives a result of 3.75. The same mentality is applied for the two and three dice situations which are shown above. Next we turn our attention to finding critical numbers associated with each dice situation after two rolls of the dice. These are shown below.
After two rolls we find the critical value to be 11.2171 for one die, 32.2845 for two dice, and 148.363 for three dice. It is important to observe that these critical values match the critical values obtained earlier in section 4.4. Using the transition and expected points matrices provides a sort of checks and balances approach to finding the critical values for each n rolls of the dice, where n = 1, 2, 3, …, n. Also important to note is the fact that the expected point equations condense the calculation significantly. For example the equation used in section 4.4 to calculate expected points after two rolls starting with two dice can be compared to the expected points equation in this section. The difference is about a page versus a line. The difference is even more remarkable in the three dice case.

Earlier we mentioned the notion of risky situations being anytime we have fewer than all three dice to roll with after a successful roll of the dice. In section 4.4 we indicated that we will stop if after a roll of the dice we have one or two dice left with which to roll in our next roll if we choose to continue our turn. We can employ the same mentality to our critical number equations. As we continue to extend the number of rolls
we are considering, we only expand the single case where we ended the previous roll with all three dice left. This is the case where we rolled either a flash or a pair with a five or ten. Perhaps the best example to illustrate this point is the critical number equation for three dice after five rolls, shown next. Only the 36/216 cases are expanded, which represent ending the previous roll with all three dice left.

### 4.6 Conclusion

We now conclude this section with a summary of the results we have found. We have demonstrated two different methods that can be used to find the critical numbers of points after n rolls of the dice. Both strategies have been shown to provide the same answer, however one method, the critical number equations, have been shown to be more concise and efficient. Using these we have expanded each dice situation out five rolls. The results are shown below in the table.

<table>
<thead>
<tr>
<th>Rolls</th>
<th>1 Die</th>
<th>2 Dice</th>
<th>3 Dice</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.75</td>
<td>25</td>
<td>153.125</td>
</tr>
<tr>
<td>2</td>
<td>11.2171</td>
<td>32.2845</td>
<td>148.363</td>
</tr>
<tr>
<td>3</td>
<td>12.096</td>
<td>33.0232</td>
<td>146.923</td>
</tr>
<tr>
<td>4</td>
<td>12.1928</td>
<td>33.0817</td>
<td>146.561</td>
</tr>
<tr>
<td>5</td>
<td>12.2008</td>
<td>33.0809</td>
<td>146.48</td>
</tr>
</tbody>
</table>
Strategy B, introduced in the previous chapter, makes use of the row 5 critical numbers. Since the row 5 numbers are closer to the true critical numbers for the game than are the row 1 numbers, strategy B is a better strategy than strategy A. In theory, the true critical numbers of the game would be the limiting cases of the critical numbers as the number of rolls, $n$, approached infinity. Thus, it may be expected that if we found row 6 numbers, those numbers would beat strategy B, row 7 numbers may be better and so forth. However this is likely not the case. However since each critical number falls in an interval of five points, we in practice can only obtain the low or high end of the five point intervals which contains the theoretical critical numbers. Thus, because of the discrete nature of point values inherent in the game, the ideal game strategy is 15, 35, and 150 for 1 die, 2 dice, and 3 dice. We now turn our attention to the TL Wimpout game simulation.
This chapter deals with the game simulation for TL Wimpout. We begin with the need for such a game simulation in our research.

5.1 Why Do We Need A Game Simulation?

A game simulation of TL Wimpout is necessary in our research to determine if our ideal game strategy actually works. While it is possible to play out each game by imposing a particular game strategy, it is not practical. By having a game simulation we may quickly test a particular game strategy many hundreds, even thousands of times, in a very short period of time. Thus a game simulation allows for many game strategies to be tested, allowing for a greater in deeper analysis of game strategies.

5.2 Algorithm for TL Wimpout Game Simulation

It is important to observe when playing TL Wimpout that there are only three possible situations. Three dice are rolled, two dice are rolled, or one die is rolled. For each situation there are 216 equally likely possible outcomes, 36 equally likely possible outcomes, or 6 equally likely possible outcomes respectively. For our game simulation we create three lists: one list for a three dice roll containing all 216 possible outcomes, one list for a two dice roll containing all 36 possible outcomes, and one list for a one die roll containing all 6 possible outcomes. The rolls are combinations of integers between 2 and 7 (inclusive). The numbers 2 through 6 correspond to the like sides of the game die while we use 7 to represent 10 on the actual die. This is done to simplify random number generation used within the program. For each outcome there is an associated number of points. Thus a list of points for each outcome is also made. Finally depending on the
current roll and how many dice score, there will be some non-scoring dice left. A list of dice left will be created with possible values 0, 1, or 2.

The end result consists of three arrays for three dice, three arrays for two dice, and three arrays for one die. Each dice roll combination has an array storing an exhaustive list of dice combinations, an array containing points for all roll outcomes, and an array containing dice left after for all roll outcomes.

We now have all the tools that TL Wimpout requires. First let’s look at how a roll is generated.

5.2.1 Generating a Roll of the Dice

Since each of the 216 possible rolls of three dice is equally likely, we simulate a roll of three dice by choosing a random integer between 0 and 215. An array index starts at 0 and not 1. Also the game begins with rolling three dice. Thus the random number generated corresponds to a slot in the list array. The roll combination found in that slot is the (random) roll of the dice. The random number also corresponds to a slot in the points array and dice left array. Thus a roll has been completed by generating a single random number. We have a random roll of the dice, and are given the number of points scored by the roll as well as how many dice may be re-rolled in the next roll.

5.2.2 Generating a Turn

A turn is simply linking a sequence of rolls together. Each new roll is dictated by the number of dice left from the previous roll. For instance if the number of dice left in the previous roll is 0, then all three dice scored, and the next roll must use all three dice. This means a random number needs to be generated between 0 and 215. If the number of dice left is 2, then one die scored in the previous roll, and next roll must use two dice.
This means a random number is generated between 0 and 35. If the number of dice left is 1, then two dice scored in the previous roll, and the next roll must use one die. This means a random number is generated between 0 and 5. Finally if the roll results in a Wimpout, the turn is over.

After a single roll, the points are found for that particular roll. Call those points the roll points. Roll points are always added to turn points. The player is then asked if he or she wishes to continue rolling to accumulate points. There are two exceptions to having the option to roll again: if a roll results in a wimpout or a flash is rolled. Currently the flash exception is not programmed into the game simulation. In this situation the turn automatically ends, points for the turn become zero, and total points remains the same. The next player’s turn then begins. Suppose the player has a successful roll of the dice, i.e. scored some points, and the player opts to stop. Turn points are added to total points and the turn is over for that player. Now suppose the player has a successful roll of the dice and opts to continue. Roll points are then added to turn points. The number of dice left from the current roll is used to generate a new roll. The points for the new roll are found. If the new points equals zero, then a wimpout has occurred and the turn is over with turn points equaling zero. If new points are anything but zero, then roll points are added to turn points and the player is faced with the option of stopping and keeping all points accumulated in the present turn or risking current earned points in the hope of gaining more points.

5.2.3 How the Game Ends

The process mentioned above is identical for player 1 and for player 2. The game ends when player 1 or player 2 has met or exceeded a certain point level, the win level,
and has more points than his or her opponent.

5.2.4 Risk Level

When a player plays TL Wimpout, he or she has a certain point at which he or she will stop if given the chance and take the present outcome of the game rather than continuing on in the game and risking what he or she has currently obtained. We define the point at which the player decides to keep his or her points as the risk level for that player. Of course, the risk level may vary from player to player and is an inherent trait in each player. A great example is gambling on a card game, such as blackjack. If a player bets on a hand and wins he or she can keep the money or try to double his or her money by betting it on the next hand. A particular player may chose to stop after 5 consecutive wins. The amount of money he or she had to lose, at the point when he or she chooses to stop playing would be considered his or her risk level. A risk level is a “point” or “level” at which the risk of losing all current obtained points or money becomes too much to risk and a player ends the turn or game.

A risk level can be programmed into the game for player 1 or player 2 or both. Also player 1 and player 2 may choose their risk levels. The way the risk level would affect game play is as follows. A turn cannot end until the risk level is met or exceeded. In this situation the probability of wimping out varies substantially depending on the chosen risk level. The higher the risk level, the more likely the turn will result in zero points gained. As an example assume player 1 has a risk level of 55 points. Suppose the first roll results in 25 points. Since 55 points have not been met or passed, player 1 will roll again. Now suppose the second roll results in 25 points. Since the sum of points from roll 1 and roll 2 results in 50 points, which is less than the risk level of 55 points, player 1
will roll again. Suppose that the third roll results in 0 points, or a wimpout. Thus the turn overall results in 0 points. Whether risk levels are programmed into the game or chosen by the players themselves, the example discussed above may occur quite frequently.

Finding the best risk level is the driving idea behind determining the ideal game strategy in which to play TL Wimpout. Risk levels can be viewed in two different manners. First a risk level can be held constant throughout the game. The probability of obtaining the risk level is the total number of possible turn combinations that result in reaching the risk level divided by the total number of possible turn combinations. In this case it is advisable to limit the number of rolls because of computational complexity. A second way of finding a risk level allows for dependence on the current status of the number of dice left. Consider the situation in which a player has \( x \) points using one scoring die. Should the player roll again? In this case, the player is deciding whether to risk the \( x \) points already earned. Thus, the question becomes, “What is the probability of obtaining \( y \) additional points by rolling the two non-scoring dice?” Using the answer to the previous question, we can compute the expected number of additional points using the two dice. Based on this expected value, an ideal game strategy becomes more precise.

Immediately following are some sample screen images from the simulation.

5.3 Game Simulation Results

Since the program will be used primarily as a tool, little attention was paid to the aesthetics of the user interface. In fact, eventually there will be no interactive play since the game strategy will dictated by future results of this research. The following screen image shows a user prompt. The current state of game is given within the prompt, and the user is asked whether or not he or she wishes to roll again. The background screen simply
keeps a log of the entire game.

Figure 5. Beginning of a turn

The following screen image depicts an alert box telling the player that the result of his or her turn was a wimpout. The background screen continues to maintain the status of the game.
Figure 6. The “Wimpout” message

The following screen image depicts an alert box indicating the conclusion of the game. The winner is identified and the final score is given. The background screen now contains a complete log of the game.
Figure 7. End of game message

5.4 Introduction to TL Wimpout Program Modifications

Sections 5.1-5.3 presented a discussion of the TL Wimpout program and how it simulated the game with the assistance of user input. In this chapter we begin by discussing the modifications that have been made to the TL Wimpout program to allow for simulation without any user assistance or input. Next we discuss the different types of strategies that the program is capable of playing. Finally we conclude this chapter with a brief presentation of how we can simulate a fixed number of games to be played using a chosen strategy to obtain empirical results which will be discussed in more depth in
Chapter 6.

5.4.1 TL Wimpout Program Modifications

If we recall the TL Wimpout program as described earlier in the chapter, which we will from now on refer to as the old program, required two users to make choices at each stage of the game simulation. For instance, the program would roll for a player but the player would have to then indicate to the program, by entering either 0 to stop or 1 to continue, what action the program should perform next. After each roll of the dice, each player would make a choice, either stop or continue his or her turn. The sole exception was when a wimpout occurred and automatically caused the turn to cease. The modified TL Wimpout program, which we will from now on refer to as the new program, does not require a player to input whether to continue or stop after each successful roll. The new program makes use of a strategy to determine at the end of each successful roll whether to continue rolling the dice or to stop and take all points that have been obtained in the turn. The same strategy can be used on behalf of each player, or each player may make use of different strategies. We now explain what we mean by strategy.

5.4.2 Game Strategies Available In TL Wimpout Game Simulation

A strategy is a course of action that is taken at each stage of the game. In the new program there are four strategies to pick from. The risk level strategy uses a fixed point value as the measuring stick after each successful roll. For instance if a player uses a risk level of 15 points, then the program will roll the dice, calculate the earned points, and then compare total earned points in the current turn to 15 points. If current earned points is less than 15 points, the program will opt to roll the dice again. If current earned points exceeds or equals 15 points, the program will choose to end the turn and add current
earned points to the overall total points for the player. When using the risk level strategy, the risk level is fixed regardless of the status of the game. The risk level strategy can be the same for both players or each player may play a different risk level. For instance player 1 could play with a risk level of 20 points and player 2 could play with a risk level of 35 points.

Another strategy that is available in the new program is called the random strategy. At the end of each roll of the dice, the program uses a random number generator to generate a random number between 0 and 1. If the random number generated is less than or equal to 0.5, the program will choose to end the turn and take current earned points and add them to the total overall points in the game for the player. However if the random number generated is greater than 0.5, the program will choose to continue the turn by rolling the dice again. The current earned points and the status of the game have no effect on whether to continue rolling or to stop.

The next strategy that can be used in the new program is called the expected points one strategy, or simply strategy A, because it makes use of expected points of the next roll in determining whether or not to continue rolling the dice. Strategy A is composed of the expected number of points after 1 roll of the dice. The calculations were introduced in section 3.5. For 1 die, 2 dice, and 3 dice left to continue a turn, the expected number of points after 1 roll are 3.75, 25, and 153.125. These numbers come from the first row of the table found in section 3.5.4. Strategy A is a better strategy than the risk level and random strategies as empirical results will demonstrate in the next chapter. Strategy A is very similar to how the risk strategy is employed. After each successful roll of the dice, the current earned points are calculated as well as finding the number of non-
scoring dice left. Based on the number of non-scoring dice, the number of current earned points is compared to the expected number of points using the non-scoring dice. If current earned points are less than the expected number of points, the program will continue the turn. If current earned points are greater than expected points, the program will end the turn and add current earned points to overall total points for the player.

The final strategy that can be used in the new program is called the expected points two strategy, or simply strategy B, because it, like strategy A, makes use of expected points of the next roll in determining whether or not to continue rolling the dice. However strategy B is composed of the expected number of points after 5 rolls of the dice. These numbers can be found in row 5 of the table in 3.5.4. For 1 die, 2 dice, and 3 dice, left to continue a turn, the expected number of points after 5 rolls are 12.2008, 33.0809, and 146.48. Strategy B is the best available strategy of the strategies we have to choose from since it is a better approximation of the three critical expected point numbers than the next best strategy, strategy A. Strategy B, like strategy A, is very similar to how the risk strategy is employed. After each successful roll of the dice, the current earned points are calculated as well as finding the number of non-scoring dice left. Based on the number of non-scoring dice, the number of current earned points is compared to the expected number of points using the non-scoring dice. If current earned points are less than the expected number of points, the program will continue the turn. If current earned points are greater than expected points, the program will end the turn and add current earned points to overall total points for the player.

5.4.3 Limitations of Game Strategies

Each of the strategies does have its own limitations. We will briefly describe the
limitation of each strategy. The risk level strategy never takes into account whether a player is winning big or losing big. The random strategy does not take into account the current earned points. For instance a player may have current earned points equaling 100 points, but the random number generated is greater than 0.5 causing the program to gamble the current 100 points instead of obtaining more points. A conservative player would choose to end his or her turn if they have the option after earning 100 points in a turn. However the random strategy is not rational and this is the major criticism of the random strategy. Finally strategy A makes use of expected or average points to be earned on the next roll of the dice. Strategy A is only one approximation of the critical numbers which would solve how to play TL Wimpout and optimize the likelihood of winning. A strategy that makes use of expected number of points after two rolls would be a better approximation and also a better strategy than strategy A. Strategy B is better than strategy A because it considers five rolls instead of one.

5.4.4 Loop Structure

We have covered how the new program plays on behalf of each player by using a chosen strategy. In order for the program to play a given strategy the player must first choose the strategy before hand. In the case of using the risk level strategy, the risk level must also be chosen before hand. Once the strategies are chosen and entered into the source code of the new program, the program will play the entire game until one player reaches the win level of 250 points. Another feature found in the new program is a loop structure that will have the program play the game a predetermined number of times using the chosen strategies and report the total number of games won by each player. This feature allows the game to be run 1,000 or 10,000 times in a matter of a sixty seconds or
less. Such efficiency allows for a great deal of empirical data to be collected, which we will discuss in Chapter 6.
Chapter 6 discusses the empirical results of the research gained by using the game simulation discussed in the last chapter. We start with a presentation of hypothesis testing and then use hypothesis testing to compare the different strategies. Results are then shown and a conclusion of the solution to TL Wimpout is then given. Now we introduce hypothesis testing.

6.1 Hypothesis Testing

We begin our discussion of hypothesis testing by considering a hypothesis test for one proportion, which will be defined more precisely shortly. A hypothesis test allows us to decide when to accept one method over another method. It allows for a comparison between the two methods [7]. A hypothesis test is composed of the null hypothesis, denoted by $H_0$, and the alternative hypothesis, denoted by $H_1$ [7]. We test the null against the alternative hypothesis. In order to perform such testing we will need to look at just one method, however this is not always the case. The result of the test containing the one method allows us to make a conclusion concerning both methods. This is where the comparison of the null and alternative hypotheses comes in. Before we continue, we issue a brief example concerning hypothesis testing. For example, we might want to test the number of jurors who vote to sentence a guilty person to life in prison against those who vote for the death penalty. We want to see if the same proportion of those who vote for life in prison is the same as those who vote for the death penalty. A proportion is represented by $Y/n$, where $Y$ represents the number of successes and $n$ the total number of
trials [7]. The overall fraction is the proportion of successes. Let’s say in our example we wish to look at the number of jurors who vote for the death penalty. The total number of jurors is \( n = 100 \), and number of jurors who vote for the death penalty is \( Y = 45 \). Our proportion is \( Y/n = 45/100 = 0.45 \). In our example we wish to test the null hypothesis \( H_0: p = p_0 \) against the alternative hypothesis \( H_1: p \neq p_0 \). Here \( p \) is the probability of jurors voting for the death penalty, and \( p_0 \) represents our hypothetical probability of success. It is important to note that \( p_0 \) is typically assigned a probability that is believed to be the actual probability. Since we are running the test we will choose \( p_0 \) to be 0.5, or rather that there is a 50% likelihood that a juror will vote for life in prison or the death penalty. The example gives us \( H_0: p = 0.5 \) and \( H_1: p \neq 0.5 \). We now present the formula for a hypothesis test of one proportion [7].

\[
\frac{Y}{n} - p_0 \quad p_0 \quad H_1 - p_0 \\
\frac{1}{n} \quad z \quad \alpha/2
\]

A hypothesis test for one proportion compares only one proportion, that of \( Y/n \) to the hypothetical probability \( p_0 \) [7]. In our formula above we have a \( z \) and a \( z_{\alpha/2} \). The \( z_{\alpha/2} \) represents a \( z \)-score. The \( z \)-score deals with the standard normal distribution and the value is looked up in a normal distribution table [7]. However we need an alpha in order to look the value up. The \( \alpha \) is called the significance level. A confidence level is one such that we are \( 100(1-\alpha)\% \) confident that our \( z \), in our formula above, is in the critical region. For a two-tailed test the critical region is the area under the probability density function that is found above \( z_{\alpha/2} \) and below \(-z_{\alpha/2} \) [7]. For the normal distribution, the probability density
function is the bell-shaped curve. If the absolute value of the computed $z$ value is greater than the critical $z$-score, then the null hypothesis is rejected. Now $\alpha / 2$ is different from just $\alpha$. We use $\alpha / 2$ because we wish to conduct a two-sided hypothesis test for one proportion. We would use just $\alpha$ if we wanted a one-sided hypothesis test for one proportion. A *one sided hypothesis test*, tests whether $p$ is greater than or less than $p_0$, but not both. A *two sided hypothesis test*, tests both [7].

We return to our example now that we have formally introduced all the pieces we need to determine whether the number of jurors who vote for life in prison or the death penalty has a 50% likelihood of occurring. We know $Y/n = 0.45$, $p_0 = 0.5$, $n = 100$, and we will choose $\alpha$ to be .05, giving a $z$-score of 1.96. Now all we must do to test $H_0: p = 0.5$ against $H_1: p \neq 0.5$ is to place all the values in the formula given earlier and see if our $z$ is greater than or less than our $z$-score of 1.96. The new formula with the values entered is shown next.

\[
\frac{Y - p_0}{\sqrt{\frac{p_0(1-p_0)}{n}}} = \frac{0.45 - 0.5}{\sqrt{\frac{0.5(1-0.5)}{100}}} = 1
\]

Since our $z$, which equaled 1, is not greater than our $z$-score, which is 1.96, we have failed to reject our null hypothesis. If our $z$ had been greater than 1.96 we would have rejected our null hypothesis. We cannot conclude then that there is a 50% likelihood that a juror will vote for life in prison or the death penalty, only that we do not have evidence to reject there is a 50% likelihood. Now we use hypothesis tests for one proportion to analyze the different strategies in the TL Wimpout game simulation.
6.2 TL Wimpout

Using the TL Wimpout program discussed in Chapter 5, we are now able to test various situations of interest using hypothesis tests for one proportion. One situation of interest is whether the simulation is fair. For example we tested to see whether the simulation was fair using strategy A versus strategy A. The null hypothesis was $H_0$: the game is fair or strategy A for player 1 had a 50% likelihood of winning the game. This gave us a $p_0$ of 0.5. The alternative hypothesis was $H_1$: the game is not fair or player 1 had either a greater or less than 50% likelihood of winning the game. Thus we used a two sided hypothesis test. The situation is depicted below in the formula.

\[
\frac{Y - 50}{\sqrt{5000}} = 1.96
\]

We ran the simulation 10000 times and found player 1 won $Y = 5014$ games. Thus we failed to reject our null hypothesis which was the game simulation was fair because our $z$ was less than our $z$-score of 1.96. We obtained the same result, failure to reject our null hypothesis, when we tested the random strategy versus the random strategy, and strategy B versus strategy B.

The next situation we tested was strategy A versus the random strategy. In this situation our null hypothesis was $H_0$: the strategies were equally likely of winning the game. Our alternative hypothesis was $H_1$: the good strategy had a greater than 50% likelihood of winning the game. Again we used a hypothetical testing probability of .5 and ran the simulation 10000 times. We found $Y = 9865$, or the number of times player 1 won using strategy A. The situation is depicted below in the formula.
We rejected our null hypothesis that the two strategies had the same likelihood of winning the game because our $z$ was greater than our $z$-score of 1.96. Instead we find strategy A is a better strategy than the random strategy. Next we look at strategy A versus strategy B.

Recall strategy A used the row 1 critical numbers found in the table in section 4.6, whereas strategy B used the row 5 critical numbers. We choose $p_0$ to be 0.5 and ran the simulation to find player 1 won $Y = 4633$ times with strategy A. In our hypothesis test, $y$ represents the number of wins using strategy A.

We reject our null hypothesis of the strategies being equal since our $z$ was greater than critical region or $z$-score of 1.96. Instead we find strategy B is a better strategy than strategy A. This hypothesis test confirms our claim in section 4.6 that row 5 would beat row 1. Next we compare the extremes of the intervals that contain the theoretical critical numbers discussed in section 4.6.

We will now test the strategy that uses the critical numbers 10, 30, and 145 versus the strategy that uses the critical numbers 15, 35, and 150. Our null hypothesis was $H_0$: the strategies were equally likely to win. Our alternative hypothesis was $H_1$: the strategies were not equal. Again we have $p_0 = 0.5$, and the simulation gave us $Y = 3345$ for player 1.
using the 10, 30, 145 strategy. The situation is shown below.

We reject our null hypothesis that the two strategies are equal since our z was much greater than our z-score of 1.96. Instead we find that the 15, 35, 150 strategy is the better strategy. This is the practical solution to TL Wimpout, since the game has a discrete point structure, the theoretical point values in section 4.6 cannot be obtained. However we are able to obtain 15, 35, and 150, thus making this the ideal game playing strategy to employ when playing TL Wimpout. We now close with a few other potential situations of interest.

We wanted to find which strategies canceled each other out, or rather had a 50% likelihood of winning the game as the competing strategy. We found fixed risk levels of 10 and 15, 20 and 25, 20 and 30, 25 and 30, as well as 60 and 65 are about the same. What is meant by this, is if two players, whom each choose one half of the pairs noted, would have the same likelihood of winning the game as the other player using the other half of the pair. Strategy A and B dominated the random strategies, and all fixed risk levels. These were the significant empirical findings using the TL Wimpout game simulation to test the various strategies. This concludes our research. We close this paper with chapter 7 and potential future work.
The research on the modification to Cosmic Wimpout, TL Wimpout, was close to a total success. An approximate ideal game playing strategy was found. If more time was available, the actual ideal game playing strategy would be the goal. In order to find the actual strategy we would have to continue our expected point equations until the critical numbers stabilized and met our tolerance level. We would have to determine what an acceptable tolerance level would be as well.

After we found the ideal game strategy consisting of the three actual critical numbers, we would try to generalize the formula. Once we have a generalized formula we can find approximately how many turns on average it would take to reach the win level, or any arbitrary win level. This was an earlier goal but we never got to the point where we could estimate the length of a game.

Further work could be done on the game simulation itself to make it more user friendly. For instance at the present time to change strategies and risk levels a user must make those choices by viewing the source code and manually making changes. A future version of the simulation would allow a user to decide what strategy to use, the risk level they want, etc. at the outset of the game without having to alter the source code. This is an aesthetic component and is not the most important aspect of the research to be continued in the future.

The final objective that might be tackled in future work is to build TL Wimpout back towards the original game Cosmic Wimpout and figure out how the expected point equations, transition and expected point matrices apply to Cosmic Wimpout. All the
questions posed for TL Wimpout would then be posed to Cosmic Wimpout, assuming no other work has previously been done on Cosmic Wimpout. This is a bold assumption, but perhaps there would be something new we could add with our research to the total research towards Cosmic Wimpout.
APPENDIX

Code for TL Wimpout modified game simulation can be found below.

```html
<HTML>
<HEAD><SCRIPT LANGUAGE=JavaScript>

//Prompt for risk level player has to enter
/*var risklevel;
 risklevel=prompt("Enter your risk level", ");
document.write(risklevel+"<Br>");
*/

//***************************************************************
//variable declarations and initializations
//***************************************************************

document.write("<br>");
var list;
var list1;
var list2;

list = new Array(216);
list1 = new Array(36);
list2 = new Array(6);

document.write("<br>");
var count;
var count1;
var count2;

count = 0;
count1 = 0;
count2 = 0;

var points;
points = new Array(216);

document.write("<br>");
var points1;
points1 = new Array(36);

var points2;
points2 = new Array(6);

var dice;
```

dice = new Array(216);

var dice1;
dice1 = new Array(36);

var dice2;
dice2 = new Array(6);

var die;
var die1;
var die2;

var match;

var rollagain = 2;
var rollagain2 = 2;

var turnover = 0;
var turnover2 = 0;

var diceleft = 0;
var diceleft2;

var gameover = 0;
var gameover2 = 0;

var totalpoints = 0;
var totalpoints2 = 0;

var roll;
var roll2;

var p1=0;
var p2=0;
var p1name = "Player one";
var p2name = "Player two";
var promptstring;

var player1points = 0;
var player2points = 0;

var winlevel = 100;
var p1wins = 0;
var p2wins = 0;

//*******************************************
//Each array tells how many points for each roll
//*******************************************
//load the three-dice points list
//*******************************************
points[0]=20;
points[1]=10;
points[2]=10;
points[3]=15;
points[4]=10;
points[5]=20;
points[6]=10;
points[7]=15;
points[8]=0;
points[9]=5;
points[10]=0;
points[11]=10;
points[12]=10;
points[13]=0;
points[14]=20;
points[15]=5;
points[16]=0;
points[17]=10;
points[18]=15;
points[19]=5;
points[20]=5;
points[21]=25;
points[22]=5;
points[23]=15;
points[24]=10;
points[25]=0;
points[26]=0;
points[27]=5;
points[28]=30;
points[29]=10;
points[30]=20;
points[31]=10;
points[32]=10;
points[33]=15;
points[34]=10;
points[35]=50;
points[36]=10;
points[37]=15;
points[38]=0;
points[39]=5;
points[40]=0;
points[41]=10;
points[42]=15;
points[43]=30;
points[44]=15;
points[45]=20;
points[46]=15;
points[47]=25;
points[48]=0;
points[49]=15;
points[50]=20;
points[51]=5;
points[52]=0;
points[53]=10;
points[54]=5;
points[55]=20;
points[56]=5;
points[57]=25;
points[58]=5;
points[59]=15;
points[60]=0;
points[61]=15;
points[62]=0;
points[63]=5;
points[64]=30;
points[65]=10;
points[66]=10;
points[67]=25;
points[68]=10;
points[69]=15;
points[70]=10;
points[71]=50;
points[72]=10;
points[73]=0;
points[74]=20;
points[75]=5;
points[76]=0;
points[77]=10;
points[78]=0;
points[79]=15;
points[80]=20;
points[81]=5;
points[82]=0;
points[83]=10;
points[84]=20;
points[85]=20;
points[86]=40;
points[87]=25;
points[88]=20;
points[89]=30;
points[90]=5;
points[91]=5;
points[92]=25;
points[93]=25;
points[94]=5;
points[95]=15;
points[96]=0;
points[97]=0;
points[98]=20;
points[99]=5;
points[100]=30;
points[101]=10;
points[102]=10;
points[103]=10;
points[104]=30;
points[105]=15;
points[106]=10;
points[107]=50;
points[108]=15;
points[109]=5;
points[110]=5;
points[111]=25;
points[112]=5;
points[113]=15;
points[114]=5;
points[115]=20;
points[116]=5;
points[117]=25;
points[118]=5;
points[119]=15;
points[120]=5;
points[121]=5;
points[122]=25;
points[123]=25;
points[124]=5;
points[125] = 15;
points[126] = 25;
points[127] = 25;
points[128] = 25;
points[129] = 50;
points[130] = 25;
points[131] = 35;
points[132] = 5;
points[133] = 5;
points[134] = 5;
points[135] = 25;
points[136] = 35;
points[137] = 15;
points[138] = 15;
points[139] = 15;
points[140] = 15;
points[141] = 35;
points[142] = 15;
points[143] = 55;
points[144] = 10;
points[145] = 0;
points[146] = 0;
points[147] = 5;
points[148] = 30;
points[149] = 10;
points[150] = 0;
points[151] = 15;
points[152] = 0;
points[153] = 5;
points[154] = 30;
points[155] = 10;
points[156] = 0;
points[157] = 0;
points[158] = 20;
points[159] = 5;
points[160] = 30;
points[161] = 10;
points[162] = 5;
points[163] = 5;
points[164] = 5;
points[165] = 25;
points[166] = 35;
points[167] = 15;
points[168] = 30;
points[169] = 30;
points[170]=30;
points[171]=35;
points[172]=60;
points[173]=40;
points[174]=10;
points[175]=10;
points[176]=10;
points[177]=15;
points[178]=40;
points[179]=50;
points[180]=20;
points[181]=10;
points[182]=10;
points[183]=15;
points[184]=10;
points[185]=50;
points[186]=10;
points[187]=25;
points[188]=10;
points[189]=15;
points[190]=10;
points[191]=50;
points[192]=10;
points[193]=10;
points[194]=30;
points[195]=15;
points[196]=10;
points[197]=50;
points[198]=15;
points[199]=15;
points[200]=15;
points[201]=35;
points[202]=15;
points[203]=55;
points[204]=10;
points[205]=10;
points[206]=10;
points[207]=15;
points[208]=40;
points[209]=50;
points[210]=50;
points[211]=50;
points[212]=50;
points[213]=55;
points[214]=50;
points[215]=100;

//**********************************************************
//load the two-dice points list
//**********************************************************
points1[0]=10;
points1[1]=0;
points1[2]=0;
points1[3]=5;
points1[4]=0;
points1[5]=10;
points1[6]=0;
points1[7]=15;
points1[8]=0;
points1[9]=5;
points1[10]=0;
points1[11]=10;
points1[12]=0;
points1[13]=0;
points1[14]=20;
points1[15]=5;
points1[16]=0;
points1[17]=10;
points1[18]=5;
points1[19]=5;
points1[20]=5;
points1[21]=25;
points1[22]=5;
points1[23]=15;
points1[24]=0;
points1[25]=0;
points1[26]=0;
points1[27]=5;
points1[28]=30;
points1[29]=10;
points1[30]=10;
points1[31]=10;
points1[32]=10;
points1[33]=15;
points1[34]=10;
points1[35]=50;

//**********************************************************
//load the one-dice points list
//**********************************************************
points2[0]=0;  
pods2[1]=0;  
pods2[2]=0;  
pods2[3]=5;  
pods2[4]=0;  
pods2[5]=10;  

//Each array tells how many dice are left after that roll of dice

//****************************************************************************
//number of dice left after a three-dice roll
// 4 => wimpout
//****************************************************************************
dice[0]=0;  
dice[1]=1;  
dice[2]=1;  
dice[3]=0;  
dice[4]=1;  
dice[5]=0;  
dice[6]=1;  
dice[7]=1;  
dice[8]=4;  
dice[9]=2;  
dice[10]=4;  
dice[11]=2;  
dice[12]=1;  
dice[13]=4;  
dice[14]=1;  
dice[15]=2;  
dice[16]=4;  
dice[17]=2;  
dice[18]=0;  
dice[19]=2;  
dice[20]=2;  
dice[21]=1;  
dice[22]=2;  
dice[23]=1;  
dice[24]=1;  
dice[25]=4;  
dice[26]=4;  
dice[27]=2;  
dice[28]=1;  
dice[29]=2;  
dice[30]=0;  
dice[31]=2;
dice[32]=2;
dice[33]=1;
dice[34]=2;
dice[35]=1;
dice[36]=1;
dice[37]=1;
dice[38]=4;
dice[39]=2;
dice[40]=4;
dice[41]=2;
dice[42]=1;
dice[43]=0;
dice[44]=1;
dice[45]=0;
dice[46]=1;
dice[47]=0;
dice[48]=4;
dice[49]=1;
dice[50]=1;
dice[51]=2;
dice[52]=4;
dice[53]=2;
dice[54]=2;
dice[55]=0;
dice[56]=2;
dice[57]=1;
dice[58]=2;
dice[59]=1;
dice[60]=4;
dice[61]=1;
dice[62]=4;
dice[63]=2;
dice[64]=1;
dice[65]=2;
dice[66]=2;
dice[67]=0;
dice[68]=2;
dice[69]=1;
dice[70]=2;
dice[71]=1;
dice[72]=1;
dice[73]=4;
dice[74]=1;
dice[75]=2;
dice[76]=4;
dice[77]=2;
dice[78]=4;
dice[79]=1;
dice[80]=1;
dice[81]=2;
dice[82]=4;
dice[83]=2;
dice[84]=1;
dice[85]=1;
dice[86]=0;
dice[87]=0;
dice[88]=1;
dice[89]=0;
dice[90]=2;
dice[91]=2;
dice[92]=0;
dice[93]=1;
dice[94]=2;
dice[95]=1;
dice[96]=4;
dice[97]=4;
dice[98]=1;
dice[99]=2;
dice[100]=1;
dice[101]=2;
dice[102]=2;
dice[103]=2;
dice[104]=0;
dice[105]=1;
dice[106]=2;
dice[107]=1;
dice[108]=0;
dice[109]=2;
dice[110]=2;
dice[111]=1;
dice[112]=2;
dice[113]=1;
dice[114]=2;
dice[115]=0;
dice[116]=2;
dice[117]=1;
dice[118]=2;
dice[119]=1;
dice[120]=2;
dice[121]=2;
dice[122]=0;
dice[123]=1;
dice[124]=2;
dice[125]=1;
dice[126]=1;
dice[127]=1;
dice[128]=1;
dice[129]=0;
dice[130]=1;
dice[131]=0;
dice[132]=2;
dice[133]=2;
dice[134]=2;
dice[135]=1;
dice[136]=0;
dice[137]=1;
dice[138]=1;
dice[139]=1;
dice[140]=1;
dice[141]=0;
dice[142]=1;
dice[143]=0;
dice[144]=1;
dice[145]=4;
dice[146]=4;
dice[147]=2;
dice[148]=1;
dice[149]=2;
dice[150]=4;
dice[151]=1;
dice[152]=4;
dice[153]=2;
dice[154]=1;
dice[155]=2;
dice[156]=4;
dice[157]=4;
dice[158]=1;
dice[159]=2;
dice[160]=1;
dice[161]=2;
dice[162]=2;
dice[163]=2;
dice[164]=2;
dice[165]=1;
dice[166]=0;
dice[167]=1;
dice[168]=1;
dice[169]=1;
dice[170]=1;
dice[171]=0;
dice[172]=0;
dice[173]=0;
dice[174]=2;
dice[175]=2;
dice[176]=2;
dice[177]=1;
dice[178]=0;
dice[179]=1;
dice[180]=0;
dice[181]=2;
dice[182]=2;
dice[183]=1;
dice[184]=2;
dice[185]=1;
dice[186]=2;
dice[187]=0;
dice[188]=2;
dice[189]=1;
dice[190]=2;
dice[191]=1;
dice[192]=2;
dice[193]=2;
dice[194]=0;
dice[195]=1;
dice[196]=2;
dice[197]=1;
dice[198]=1;
dice[199]=1;
dice[200]=1;
dice[201]=0;
dice[202]=1;
dice[203]=0;
dice[204]=2;
dice[205]=2;
dice[206]=2;
dice[207]=1;
dice[208]=0;
dice[209]=1;
dice[210]=1;
dice[211]=1;
dice[212]=1;
dice[213]=0;
dice[214]=1;
dice[215]=0;

/*********************************************
//number of dice left after a two-dice roll
// 4 => wimpout
/*********************************************
dice1[0]=0;
dice1[1]=4;
dice1[2]=4;
dice1[3]=1;
dice1[4]=4;
dice1[5]=1;
dice1[6]=4;
dice1[7]=0;
dice1[8]=4;
dice1[9]=1;
dice1[10]=4;
dice1[11]=1;
dice1[12]=4;
dice1[13]=4;
dice1[14]=0;
dice1[15]=1;
dice1[16]=4;
dice1[17]=1;
dice1[18]=1;
dice1[19]=1;
dice1[20]=1;
dice1[21]=0;
dice1[22]=1;
dice1[23]=0;
dice1[24]=4;
dice1[25]=4;
dice1[26]=4;
dice1[27]=1;
dice1[28]=0;
dice1[29]=1;
dice1[30]=1;
dice1[31]=1;
dice1[32]=1;
dice1[33]=0;
dice1[34]=1;
dice1[35]=0;
dice2[0]=4;
dice2[1]=4;
dice2[2]=4;
dice2[3]=0;
dice2[4]=4;
dice2[5]=0;

count = 0;
for(cube1 = 2; cube1 <= 7; cube1++){
    for(cube2 = 2; cube2 <= 7; cube2++){
        for(cube3 = 2; cube3 <= 7; cube3++){
            list[count] = cube1++ "+"+cube2++ "+"+cube3;
            //document.write("roll #"+count++ is "+list[count]+"<BR>");
            //generate the points for this roll
            //store the points in points[count]
            count++;
        }
    }
}

count = 0;
for(cube1 = 2; cube1 <= 7; cube1++){
    for(cube2 = 2; cube2 <= 7; cube2++){
        list1[count] = cube1++ "+"+cube2;
        //document.write("roll #"+count++ is "+list1[count]+"<BR>");
        count++;
    }
}

count = 0;

//number of dice left after a one-die roll
// 4  => wimpout
//*********************************************
for(cube1 = 2; cube1 <= 7; cube1++){
  list2[count] = cube1;
  //document.write("roll "+count+" is "+list2[count]+"<BR>");
  count++;
}

//*********************************
// now play the game
//*********************************

//Function to do the turn
//**************************************************************
function playa1(name, p1points)
{
  var roll;
  var player1points = 0;
  var firsttimethrough = 1;
  turnover = 0;
  diceleft = 0;
  player1points = 0;

  // loop to control the player's turn
  while (turnover == 0){
    if (diceleft == 0 & firsttimethrough == 1){
      rollagain = 1;
      firsttimethrough = 0;
    }
    if (rollagain ==1){
      if (diceleft == 2 && player1points < 25){
        roll=Math.floor(Math.random()*35);
        if (points1[roll] == 0){
          //document.write(name + " rolled "+ list1[roll]+".");
          //document.write(name + " wimped out.");
          player1points = 0;
          turnover = 1;
          //alert(name + " has wimped out!");
        }else{
          diceleft = dice1[roll];
          //document.write(name + " rolled "+ list1[roll]+".");
          //document.write(name + " has "+ diceleft+" dice left");
        }
      }else{
        //document.write(name + " has "+ points1[roll]+"");
      }
    }
  }
}
points"+".");

player1points = player1points + points1[roll];
//document.write(name + " has "+ player1points+"

points in this turn. ");

//alert(name + " has "+diceleft+" dice left,
"+player1points+" points in this turn.");
promptstring = name + " rolled "+ list1[roll]+".
"+name + " has "+diceleft+" dice left, "+player1points+" points in this turn."

if (diceleft == 1 & player1points < 11){
    roll=Math.floor(Math.random()*5);
    if (points2[roll] == 0){
        //document.write(name + " rolled +
list2[roll]+".");
        //document.write(name + " wimped out. ");
        player1points = 0;
        turnover = 1;
        //alert(name + " has wimped out!");
    }else{
        diceleft = dice2[roll];
        //document.write(name + " rolled +
dice2[roll]+" dice left"+".");
        //document.write(name + " has "+
points2[roll]+" points"+".");
        player1points = player1points +
points2[roll];
        //document.write(name + " has "+
player1points+" points in this turn. ");
        //alert("You have "+dice2[roll]+" dice left,
"+player1points+" points in this turn.");
promptstring = name + " rolled "+
list2[roll]+". "+name + " has "+diceleft+" dice left, "+player1points+" points in this turn."
    }
}else{
    if (diceleft == 0 & player1points < 154){
        roll=Math.floor(Math.random()*215);
        if (points[roll] == 0){
            //document.write(name + " rolled "+
rolled "+ list[roll]+"."");
        }else{
            //document.write(name + "

wimped out. ");

}
player1points = 0;
turnover = 1;

//alert(name + " has wimped out!");

else{
diceleft = dice[roll];

//document.write(name + "
rolled " + list[roll] + ").");

has " + dice[roll] +" dice left" + ".");

has " + points[roll] +" points" + ".");

player1points =

player1points + points[roll];

has " + player1points +" points in this turn.");

" + dice[roll] +" dice left, " + player1points +" points in this turn.");

promptstring = name + "
rolled " + list[roll] + ". " + name + " has " + diceleft +" dice left, " + player1points +" points in this turn."

}
}
}

}
}
}

else{

turnover = 1;

}

if(diceleft == 0 && player1points < 154){
turnover = 0; }

if(diceleft == 1 && player1points < 11){
turnover = 0; }

if(diceleft == 2 && player1points < 25){
turnover = 0; }

if(diceleft == 0 && player1points > 154){
turnover = 1; }

if(diceleft == 1 && player1points > 11){
turnover = 1; }

if(diceleft == 2 && player1points >= 25){
turnover = 1; }

}
```javascript
// end the turn loop
return (player1points);
}

function playa2(name, p2points)
{

var roll;
var player1points = 0;
var firsttimethrough = 1;
var randomnumber = (Math.random()*1);
turnover = 0;
diceleft = 0;
player1points = 0;

// loop to control the player's turn
while (turnover == 0){
    if (diceleft == 0 & firsttimethrough == 1){
        rollagain = 1;
        firsttimethrough = 0;
    }
    if (rollagain ==1){
        if (diceleft == 2){
            roll=Math.floor(Math.random()*35);
            if (points1[roll] == 0){
                //document.write(name + " rolled "+ list1[roll]+".");
                //document.write(name + " wimped out. ");
                player1points = 0;
                turnover = 1;
                //alert(name + " has wimped out!");
            }else{
                diceleft = dice1[roll];
                //document.write(name + " rolled "+ list1[roll]+".");
                //document.write(name + " has "+ diceleft+" dice left");
                //document.write(name + " has "+ points1[roll]+" points"+".");
                player1points  = player1points + points1[roll];
            }
        }
    }
}
```
points in this turn. ";)
//alert(name + " has "+diceleft+" dice left,
"+player1points+" points in this turn.");
promptstring = name + " rolled "+ list1[roll]+".
"+name + " has "+diceleft+" dice left, "+player1points+" points in this turn."}
} else{if (diceleft == 1){
roll=Math.floor(Math.random()*5);
if (points2[roll] == 0){
//document.write(name + " rolled "+
list2[roll]+".");
//document.write(name + " wimped out. ");
player1points = 0;
turnover = 1;
//alert(name + " has wimped out!");
} else{
diceleft = dice2[roll];
//document.write(name + " rolled "+
list2[roll]+".");
//document.write(name + " has "+
dice2[roll]+" dice left"+".");
//document.write(name + " has "+
points2[roll]+" points"+".");
player1points = player1points +
points2[roll];
//document.write(name + " has "+
player1points+" points in this turn. ");
//alert("You have "+dice2[roll]+" dice left,
"+player1points+" points in this turn.");
promptstring = name + " rolled "+
list2[roll]+". "+name + " has "+diceleft+" dice left, "+player1points+" points in this turn."}
} else{
/*This is where the problem is --->*/
if (diceleft == 0){
roll=Math.floor(Math.random()*215);
if (points[roll] == 0){
//document.write(name + "
rolled "+ list[roll]+".");
//document.write(name + "
wimped out. ");
player1points = 0;
turnover = 1;
alert(name + " has wimped out!");
}
}
}
}
else{
    turnover = 1;
}
if (turnover != 1){
    randomnumber = (Math.random()*1);
    //document.write("This is the random number "+randomnumber);
    if(randomnumber >=.5){
        rollagain = 1;
        turnover = 0;}
    if(randomnumber <.5){
        rollagain = 0;
        turnover = 1;
        //document.write(" turn should be over "+turnover);
    }
}
// end the turn loop
return (player1points);
function playa3(name, ppoints) {

    var roll;
    var player1points = 0;
    var risklevel = 50;
    var firsttimethrough = 1;
    var randomnumber = (Math.random()*1);
    turnover = 0;
    diceleft = 0;
    player1points = 0;

    // loop to control the player's turn
    while (turnover == 0){
        if (diceleft == 0 & firsttimethrough == 1){
            rollagain = 1;
            firsttimethrough = 0;
        }

        if (rollagain ==1){
            if (diceleft == 2){
                roll=Math.floor(Math.random()*35);
                if (points1[roll] == 0){
                    //document.write(name + " rolled "+ list1[roll]+".");
                    //document.write(name + " wimped out. ");
                    player1points = 0;
                    turnover = 1;
                    //alert(name + " has wimped out!");
                }else{
                    diceleft = dice1[roll];
                    //document.write(name + " rolled "+ list1[roll]+".");
                    //document.write(name + " has "+ diceleft+" dice left");
                    //document.write(name + " has "+ points1[roll]+" points");
                    player1points = player1points + points1[roll];
                    //document.write(name + " has "+ player1points+" points in this turn. ");
                    //alert(name + " has "+diceleft+" dice left,
                }
            }
        }
    }
}
"+player1points+" points in this turn.");

    promptstring = name + " rolled "+ list1[roll]+".
"+name + " has "+diceleft+" dice left, "+player1points+" points in this turn."}

} else if (diceleft == 1) {

    roll=Math.floor(Math.random()*5);

    if (points2[roll] == 0) {

        //document.write(name + " rolled "+
        list2[roll]+".");

        //document.write(name + " wimped out.");

        player1points = 0;

        turnover = 1;

        //alert(name + " has wimped out!");

    } else {

        diceleft = dice2[roll];

        //document.write(name + " rolled "+
        list2[roll]+".");

        //document.write(name + " has "+

        dice2[roll]+" dice left"+".");

        //document.write(name + " has "+

        points2[roll]+" points"+".");

        player1points = player1points +

        points2[roll];

        //document.write(name + " has "+

        player1points+" points in this turn.");

        //alert("You have "+dice2[roll]+" dice left,

        "+player1points+" points in this turn.");

        promptstring = name + " rolled "+

        list2[roll]+". "+name + " has "+diceleft+" dice left, "+player1points+" points in this turn."}

} else {

/*This is where the problem is --->*/

if (diceleft == 0) {

    roll=Math.floor(Math.random()*215);

    if (points[roll] == 0) {

        //document.write(name + "

        rolled "+ list[roll]+".");

        //document.write(name + "

        wimped out.");

    player1points = 0;

    turnover = 1;

    //alert(name + " has wimped

    out!");

}} else {

}
rolled "+ list[roll]+"."");

has "+ dice[roll]+" dice left"+".");

has "+ points[roll]+" points"+".");

player1points =
player1points + points[roll];

has "+ player1points+" points in this turn. ");

"+dice[roll]+" dice left, "+player1points+" points in this turn.");

rolled "+ list[roll]+."+name + " has "+diceleft+" dice left, "+player1points+" points in this turn.

} } 

} 

} } 

} 

} 

else{

    turnover = 1;

} 

} } 

if (turnover != 1){

    randomnumber = (Math.random()*1);

    //document.write("This is the random number "+randomnumber);

    if(player1points < risklevel){

        rollagain = 1;
        turnover = 0;
    } 

    if(player1points >= risklevel){

        rollagain = 0;
        turnover = 1;
        //document.write(" turn should be over "+turnover);
    } 

} } 

// end the turn loop
return (player1points);
function playa4(name, ppoints)
{

    var roll;
    var player1points = 0;
    var risklevel = 60;
    var firsttimethrough = 1;
    var randomnumber = (Math.random()*1);
    turnover = 0;
    diceleft = 0;
    player1points = 0;

    // loop to control the player's turn
    while (turnover == 0){
        if (diceleft == 0 & firsttimethrough == 1){
            rollagain = 1;
            firsttimethrough = 0;
        }

        if (rollagain ==1){
            if (diceleft == 2){
                roll=Math.floor(Math.random()*35);
                if (points1[roll] == 0){
                    //document.write(name + " rolled " + list1[roll]+".");
                    //document.write(name + " wimped out. ");
                    player1points = 0;
                    turnover = 1;
                    //alert(name + " has wimped out!");
                }else{
                    diceleft = dice1[roll];
                    //document.write(name + " rolled " + list1[roll]+".");
                    //document.write(name + " has "+ diceleft+" dice left");
                    player1points = player1points + points1[roll];
                    //document.write(name + " has "+ player1points+" points in this turn.");
                    //alert(name + " has "+diceleft+" dice left, "+player1points+" points in this turn.");
                    promptstring = name + " rolled " + list1[roll]+".
                    "+name + " has "+diceleft+" dice left, "+player1points+" points in this turn.";
                }
            }
        }
    }
}
```javascript
} else { if (diceleft == 1) {
    roll = Math.floor(Math.random() * 5);
    if (points2[roll] == 0) {
        //document.write(name + " rolled " +
        list2[roll] + ".");
        //document.write(name + " wimped out.");
        player1points = 0;
        turnover = 1;
        //alert(name + " has wimped out!");
    } else {
        diceleft = dice2[roll];
        //document.write(name + " rolled " +
        list2[roll] + ".");
        //document.write(name + " has "+
        dice2[roll] + " dice left" + "+.");
        //document.write(name + " has "+
        points2[roll] + " points" + "+.");
        //document.write(name + " has "+
        points2[roll];
        //document.write(name + " has "+
        player1points = player1points +
        points2[roll];
        //document.write(name + " has "+
        player1points + " points in this turn.");
        //alert("You have "+dice2[roll] + " dice left," +
        list2[roll] + ". " + name + " has "+diceleft + " dice left," +
        list2[roll] + ". " + name + " has "+dicelast + ".");
        promptstring = name + " rolled " +
    } else {
        /*This is where the problem is --->*/
        if (diceleft == 0) {
            roll = Math.floor(Math.random() * 215);
            if (points[roll] == 0) {
                //document.write(name + "
                rolled " + list[roll] + ".");
                //document.write(name + "
                wimped out.");
            } else {
                diceleft = dice[roll];
                //document.write(name + "
                rolled " + list[roll] + ".");
                //document.write(name + "
            }
        } else {
            /*This is where the problem is --->*/
```
function play1(name, ppoints) {
  play1points = ppoints;
  roll = Math.random();
  if (roll < 0.5) {
    if (play1points < risklevel) {
      play1points = play1points + points[roll];
      if (play1points >= risklevel) {
        return (play1points);
      }
    } else {
      turnover = 1;
    }
  }
  rollagain = 0;

  // end the turn loop
  return (play1points);
}

function play1(name, ppoints) {
  play1points = ppoints;
  roll = Math.random();
  if (roll < 0.5) {
    if (play1points < risklevel) {
      play1points = play1points + points[roll];
      if (play1points >= risklevel) {
        return (play1points);
      }
    } else {
      turnover = 1;
    }
  }
  rollagain = 0;

  // end the turn loop
  return (play1points);
}
var roll;
var player1points = 0;
var firsttimethrough = 1;
turnover = 0;
diceleft = 0;
player1points = 0;

// loop to control the player's turn
while (turnover == 0){
    if (diceleft == 0 & firsttimethrough == 1){
        rollagain = 1;
        firsttimethrough = 0;
    }
    if (rollagain ==1){
        if (diceleft == 2 && player1points < 33.0809){
            roll=Math.floor(Math.random()*35);
            if (points1[roll] == 0){
                //document.write(name + " rolled "+ list1[roll]+".");
                //document.write(name + " wimped out.");
                player1points = 0;
                turnover = 1;
                //alert(name + " has wimped out!");
            }else{
                diceleft = dice1[roll];
                //document.write(name + " rolled "+ list1[roll]+".");
                //document.write(name + " has "+ diceleft+" dice left");
                //document.write(name + " has "+ points1[roll]+" points");
                player1points = player1points + points1[roll];
                //document.write(name + " has "+ player1points+" points in this turn.");
                //alert(name + " has "+diceleft+" dice left,
                "+player1points+" points in this turn.");
                promptstring = name + " rolled "+ list1[roll]+".
                "+name + " has "+diceleft+" dice left, "+player1points+" points in this turn.";
            }
        }else{
            if (diceleft == 1 & player1points < 12.2008){
                roll=Math.floor(Math.random()*5);
                if (points2[roll] == 0){
                    //document.write(name + " rolled "+ list2[roll]+".");
                    //document.write(name + " wimped out.");
                }
            }
        }
    }
}
player1points = 0;
turnover = 1;
  //alert(name + " has wimped out!");

} else {
diceleft = dice2[roll];
  //document.write(name + " rolled "+
list2[roll]+".");
  //document.write(name + " has "+
dice2[roll]+" dice left"+".");
  //document.write(name + " has "+
points2[roll]+" points"+".");
  //document.write(name + " has "+
points2[roll];
  //document.write(name + " has "+
player1points+" points in this turn.");
  //alert("You have "+dice2[roll]+" dice left,
"+player1points+" points in this turn.");
  promptstring = name + " rolled "+
list2[roll]+". "+name + " has "+diceleft+" dice left,"+player1points+" points in this turn."
}

} else {
if (diceleft == 0 && player1points <
146.48){
  roll=Math.floor(Math.random()*215);
  if (points[roll] == 0){
    //document.write(name + "
rolled "+ list[roll]+".");
    //document.write(name + "
wimped out.");
  }
  } else {
    diceleft = dice[roll];
    //document.write(name + "
rolled "+ list[roll]+".");
    //document.write(name + "
has "+ dice[roll]+" dice left"+".");
    //document.write(name + "
has "+ points[roll]+" points"+".");
    player1points =
    //document.write(name + "
player1points + points[roll];
}

}
function playa6(name, ppoints)
{
    var roll;
    var player1points = 0;
    var firsttimethrough = 1;
    turnover = 0;
    diceleft = 0;
    player1points = 0;

    return (player1points);
}
// loop to control the player's turn
while (turnover == 0) {
    if (diceleft == 0 & firsttimethrough == 1) {
        rollagain = 1;
        firsttimethrough = 0;
    }
    if (rollagain == 1) {
        if (diceleft == 2 && player1points < 30) {
            roll = Math.floor(Math.random() * 35);
            if (points1[roll] == 0) {
                // document.write(name + " rolled " + list1[roll] + ".");
                // document.write(name + " wimped out.");
                player1points = 0;
                turnover = 1;
                // alert(name + " has wimped out!");
            } else {
                diceleft = dice1[roll];
                // document.write(name + " rolled " + list1[roll] + ".");
                // document.write(name + " has " + diceleft + " dice left");
                // document.write(name + " has " + points1[roll] + " points in this turn.");
                player1points = player1points + points1[roll];
                // document.write(name + " has " + player1points + " points in this turn.");
                // alert(name + " has " + diceleft + " dice left, " + player1points + " points in this turn.";
                promptstring = name + " rolled " + list1[roll] + ";
                // document.write(name + " has " + diceleft + " dice left, " + player1points + " points in this turn.";
            }
        } else {
            if (diceleft == 1 & player1points < 10) {
                roll = Math.floor(Math.random() * 5);
                if (points2[roll] == 0) {
                    // document.write(name + " rolled " + list2[roll] + ");
                    // document.write(name + " wimped out.");
                    player1points = 0;
                    turnover = 1;
                    // alert(name + " has wimped out!");
                } else {
                    diceleft = dice2[roll];
                    // document.write(name + " rolled " + list2[roll] + ");
                }
            }
        }
    }
}
dice2[roll] +" dice left"+".";
//document.write(name + " has "+
points2[roll] +" points"+".");
//document.write(name + " has "+
points2[roll];
player1points +" points in this turn.";
"+player1points+" points in this turn.");
alert("You have "+dice2[roll] +" dice left, 
promptstring = name + "+" rolled "+
list2[roll]+". "+name + " has "+diceleft+" dice left, "+player1points+" points in this turn." 
}
}

if (diceleft == 0 && player1points < 145) {

roll=Math.floor(Math.random()*215);  
if (points[roll] == 0) {
   //document.write(name + "
rolled "+ list[roll]+".");
   //document.write(name + "
wimped out.");
   player1points = 0;
turnover = 1;
   //alert(name + " has wimped out!");
}
else {
diceleft = dice[roll];
   //document.write(name + "
rolled "+ list[roll]+".");
   //document.write(name + "
has "+ dice[roll] +" dice left"+".");
   //document.write(name + "
has "+ points[roll]+" points"+".");
   player1points = player1points + points[roll];
   //document.write(name + "
has "+ player1points+" points in this turn.");
   //alert("You have "+dice[roll] +" dice left, "+player1points+" points in this turn.");
   promptstring = name + "
rolled "+ list[roll]+". "+name + " has "+diceleft+" dice left, "+player1points+" points in this turn."
}
}

}
else{
    turnover = 1;
}

if(diceleft == 0 &
 player1points < 145){
    turnover = 0;
}
if(diceleft == 1 &
 player1points < 10){
    turnover = 0;
}
if(diceleft == 2 &
 player1points < 30){
    turnover = 0;
}
if(diceleft == 0 &
 player1points >= 145){
    turnover = 1;
}
if(diceleft == 1 &
 player1points >= 10){
    turnover = 1;
}
if(diceleft == 2 &
 player1points >= 30){
    turnover = 1;
}

// end the turn loop
return (player1points);

function playa7(name, ppoints)
{
    var roll;
    var player1points = 0;
    var firsttimethrough = 1;
    turnover = 0;
    diceleft = 0;
    player1points = 0;

    // loop to control the player's turn
    while (turnover == 0){
        if (diceleft == 0 & firsttimethrough == 1){
            rollagain = 1;
            firsttimethrough = 0;
        }
        if (rollagain ==1){
            if (diceleft == 2 & player1points < 35){


82
```javascript
roll=Math.floor(Math.random()*35);
if (points1[roll] == 0) {
    //document.write(name + " rolled "+ list1[roll]+".");
    //document.write(name + " wimped out.");
    player1points = 0;
    turnover = 1;
    //alert(name + " has wimped out!");
} else {
    diceleft = dice1[roll];
    //document.write(name + " rolled "+ list1[roll]+".");
    //document.write(name + " has "+ diceleft+" dice left");
    //document.write(name + " has "+ points1[roll]+" points in this turn.");
    //alert(name + " has "+ diceleft+" dice left,
    "+player1points+" points in this turn.");
    promptstring = name + " rolled "+ list1[roll]+".
    promptstring = name + " has "+ diceleft+" dice left,"+player1points+" points in this turn.
}
} else {
    if (diceleft == 1 && player1points < 15) {
        roll=Math.floor(Math.random()*5);
        if (points2[roll] == 0) {
            //document.write(name + " rolled "+
            list2[roll]+".");
            //document.write(name + " wimped out.");
            player1points = 0;
            turnover = 1;
            //alert(name + " has wimped out!");
        } else {
            diceleft = dice2[roll];
            //document.write(name + " rolled "+
            list2[roll]+".");
            //document.write(name + " has "+
            dice2[roll]+" dice left"+".");
            //document.write(name + " has "+
            points2[roll]+" points"+".");
            player1points = player1points +
            points2[roll];
            //document.write(name + " has "+
            player1points+" points in this turn.");
            //alert("You have "+dice2[roll]+" dice left,
```
promptstring = name + " rolled " + list2[roll] + ". " + name + " has " + diceleft + " dice left, " + player1points + " points in this turn."

} else {
    if (diceleft == 0 && player1points < 150) {
        roll = Math.floor(Math.random() * 215);
        if (points[roll] == 0) {
            player1points = 0;
            turnover = 1;
            //alert(name + " has wimped out!");
        } else {
            diceleft = dice[roll];
            player1points = player1points + points[roll];
            //alert("You have " + dice[roll] + " dice left, " + player1points + " points in this turn.");
            promptstring = name + " rolled " + list[roll] + ". " + name + " has " + diceleft + " dice left, " + player1points + " points in this turn."
        }
    } else {
        turnover = 1;
    }
}
if(diceleft == 0 && player1points < 150){
    turnover = 0; }
if(diceleft == 1 && player1points < 15){
    turnover = 0; }
if(diceleft == 2 && player1points < 35){
    turnover = 0; }
if(diceleft == 0 && player1points >= 150){
    turnover = 1; }
if(diceleft == 1 && player1points >= 15){
    turnover = 1; }
if(diceleft == 2 && player1points >= 35){
    turnover = 1; }
}
// end the turn loop
return (player1points);

//************************************************************************************************
//calls function
//************************************************************************************************

gameover = 0;
winlevel = 250;
p1wins = 0;
p2wins = 0;
count = 0;
var numgames = 10000;
for(count = 1; count <= numgames; count++){
    p1=0;
    p2=0;
    gameover = 0;
    while (gameover == 0){
        p1points = p1;
        p2points = p2;
        document.write("<BR>");
        p1 = p1+playa6(p1name, p1points);
        //document.write(p1name +" has "+p1+" points"+".<BR>");
        document.write("<BR>");
        p2 = p2+playa7(p2name, p2points);
        //document.write(p2name +" has "+p2+" points"+".<BR>");

        if(p1>p2 && p1>winlevel){
            //document.write("You have won player 1","\n");
            document.write("CONGRATULATIONS Player 1, you have just won TL Wimpout !!!! Final Score: "+p1+" to "+p2);
gameover = 1;
p1wins = p1wins+1;

if(p2>p1 &amp;&amp; p2&gt;winlevel){
   //document.write("You have won player 2");
   document.write("CONGRATULATIONS Player 2, you have just
won TL Wimpout !!!! Final Score: "+p1+" to "+p2);
   gameover = 1;
   p2wins = p2wins+1;
}
}

document.write("<BR><BR>PLAYER 1 won "+p1wins+" games and PLAYER 2
won "+p2wins+" games");
alert("PLAYER 1 won "+p1wins+" games and PLAYER 2 won "+p2wins+" games");
						 Random Number Generator between 0 and 1, where does it go is the
problem?
						 var randomnumber=(Math.random()*1)
document.write("This is the random number"+randomnumber);
						 /*
						</SCRIPT>
						</HEAD>
						&lt;BODY&gt;
						</BODY&gt;
						&lt;/HTML&gt;
BIBLIOGRAPHY


BIOGRAPHICAL SKETCH

My full name is Anthony T. Litsch III. I was born on December 19th, 1984 in a Southampton hospital on the South Fork of Long Island, New York. I graduated from Spruce Creek High School in Port Orange, Florida in May of 2003 with an IB, International Baccalaureate, diploma. I started my undergraduate work at Stetson University in the fall of 2003. I chose the major of Mathematics in the spring of 2003 at the preview day for new students at Stetson. On that day, I made a choice to graduate from Stetson as quickly as possible. The target goal was three years or six semesters. In May of 2006 I will walk away from Stetson University with a Bachelor’s Degree of Arts & Science in Mathematics. In the fall of 2006 I will embark upon the next level, graduate school. My grandfather, Anthony T. Litsch, was a fine attorney. I will carry on the tradition and become an attorney myself. I will graduate from law school in May of 2009. Before the end of 2009, and before my 25th birthday, I will be sworn in as an attorney. I will never look back.