PHSX815_Project1:
Monopoly and Unfair Dice

Kurt Hamblin

March 1, 2021

1 Introduction

The game of Monopoly, both simultaneously loved and hated by people everywhere, is rather interesting from a mathematical point of view. The simplicity of the game, based solely on dice rolls with a few simple rules, allows it to be studied easier than a more complex game (e.g. chess). While there have been many mathematical looks into Monopoly, the effect that biased (i.e. "unfair") dice has on the game outcomes has not been extensively investigated. We aim to fill this gap, and investigate the differences, if any, that biased dice make.

This paper is organized as follows: Sec. 2 provides an overview of the code used to simulate the game of Monopoly, and in Sec. 3 we present and our results, analysis, and our conclusions.

2 Overview of Code and Experimental Procedure

For this experiment, we must simulate an entire game of Monopoly. To be exact, we must simulate enough games of Monopoly to arrive at the probability distribution for all the tiles on the board. For every game simulated, we track the positions of the player for every "turn" they play, following all the rules of the game.

2.1 Overall Code Structure

We create a Monopoly Class, to be used in conjunction with the Random class used previously throughout the course. This Monopoly class is self contained, and includes all the methods and class/instance attributes needed to simulate an entire game. To simulate a game, a Monopoly object is created, and then the method `play_games()` is called. it allows the number of games to be played to be specified, as well as the number of turns per game. The method returns a 2D array of shape (Nturns, 40), where each row in the 2D array is the outcome for a given turn, and every column corresponds to a tile on the board. By simulating multiple games at once, the resulting output approaches the true probability distribution for the game (after many turns and as $Ngames \to \infty$).

2.2 Dice Rolling

To simulate dice rolling, we add a method to the Random class, `Random.roll_die(Nsides, weights)`, that returns an integer from 1 to Nsides. We include the ability to roll a biased die with the inclusion of "weights," which allow the user to specify the exact probability of each side of the die being rolled. The algorithm divides the space from 0 to 1 into slices corresponding to each side of the die (either by splitting 0 to 1 evenly, or using the weights as the boundaries for the 'slices'), and then generates a random float from 0 to 1, and returns the side that this random float corresponds to.

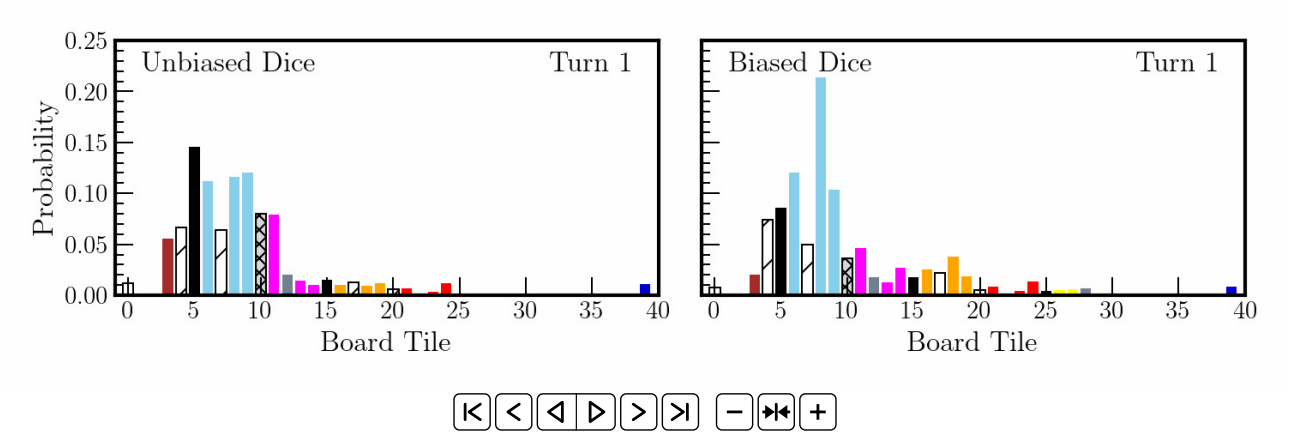

Figure 1: Probability distributions for simulated monopoly games using unbiased dice (left) and biased dice (right). We note that tile 30 always has a probability of 0 because it is the 'Go to Jail' tile and the player never ends a turn there. Property tiles are color coded, jail is marked with the crossed-grey bar, utility tiles are grey, and all other tiles (chance, chest, tax, etc.) are clear with lines through them. **View in `Adobe Acrobat Reader` to see animation.**

2.3 Game Nuances

There are a few nuances to the game of Monopoly that make the simulation more difficult than it otherwise may be. Namely, this includes the 'Chance' and 'Community Chest' decks, of which (10) and (2) cards, respectively, can move the player around the board. We choose to fully incorporate these cards, and use the `roll_die()` method to generate a random integer in order to determine which card to draw when the player lands on a 'Chance' or 'Community Chest' tile. Each time, we remove the card from the deck, and when the deck is emptied, we re-initialize it. Note that there is no need to 'shuffle' the deck, since we choose which card to draw randomly.

Additionally, the game's 'Jail' complicates the simulation. In a real game of Monopoly, when a player is sent to jail, they can choose (starting the turn after they were sent to jail) to pay a fine to get out of jail immediately, or stay in jail for 3 turns, or until they roll doubles. To simplify the simulation, we assume that the player always chooses to pay the fine immediately.

2.4 Experimental Simulation

In order to arrive at the 'true' probability distribution for fair dice, we simulate 10^7 games using fair dice. We then simulate 10^4 games using biased dice, with weights drawn randomly from a normal distribution. Optimally, the 'true' distribution would be compiled by running for more games than this, but comparison in the distribution between 10^6 games and 10^7 games showed little difference.

3 Results and Conclusions

In Figure (1), we present animations for the tile probability distributions for unbiased and biased dice (Note: view in Adobe Acrobat Reader in order to view animations properly). Qualitatively, these animations are quite interesting, as they allow a side-by-side comparison for every turn. In the beginning, the two distributions visibly differ, but by turn ~10, the begin to appear quite similar, where they both begin to flatten out. Logically, it makes sense that they look similar after many turns, since player positions should become much more spread out across the board.

While the qualitative comparison provided by the animations in Figure (1) is interesting and useful, we must also quantitatively compare the two distributions. To do so, we perform a chi-squared test, where we aim to test whether the experimental results (from using biased dice) come from the same distribution as the "known" distribution for fair dice. Our hypotheses are:

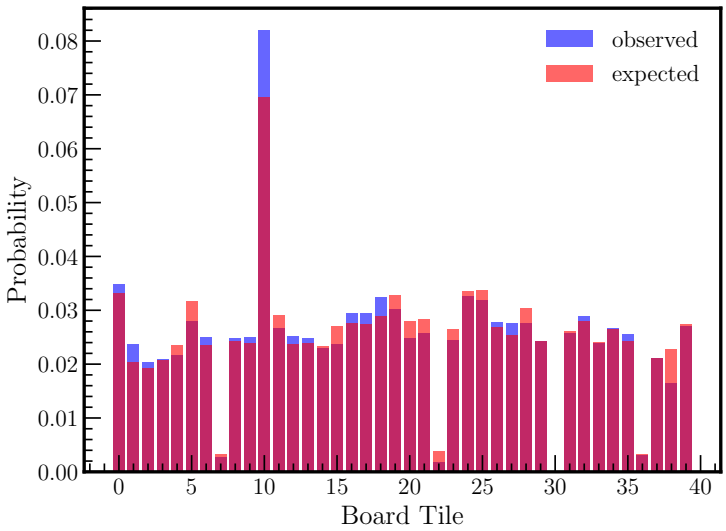

Figure 2: Probability distributions for the final turn. Expected distribution (unbiased dice) shown in red, and the experimentally observed distribution (with biased dice) is shown in blue.

Hypothesis $\mathbf{H_0}$: Fair dice were used to play the experimental games

Hypothesis $\mathbf{H_1}$: Unfair dice were used to play the experimental games

We adopt a significance level of $\alpha = 0.05$, and with 39 dof (39 rather than 40 since tile 30 is never counted) we will reject hypothesis H_0 if $\chi^2 > 54.57$.

We use `scipy` to perform the test, and we use the probability distributions from the last turn, shown in Figure (2) (50 here) for each sample. With this test, we find: $\chi^2 = 99.31 > 54.57$, thus we reject the null hypothesis. The player knows now that the dice used in the experiment are biased. From Figure (2) it is evident that the biased die had a notably higher chance of landing the player in jail, which makes sense since two biased dice will be more likely to roll doubles.