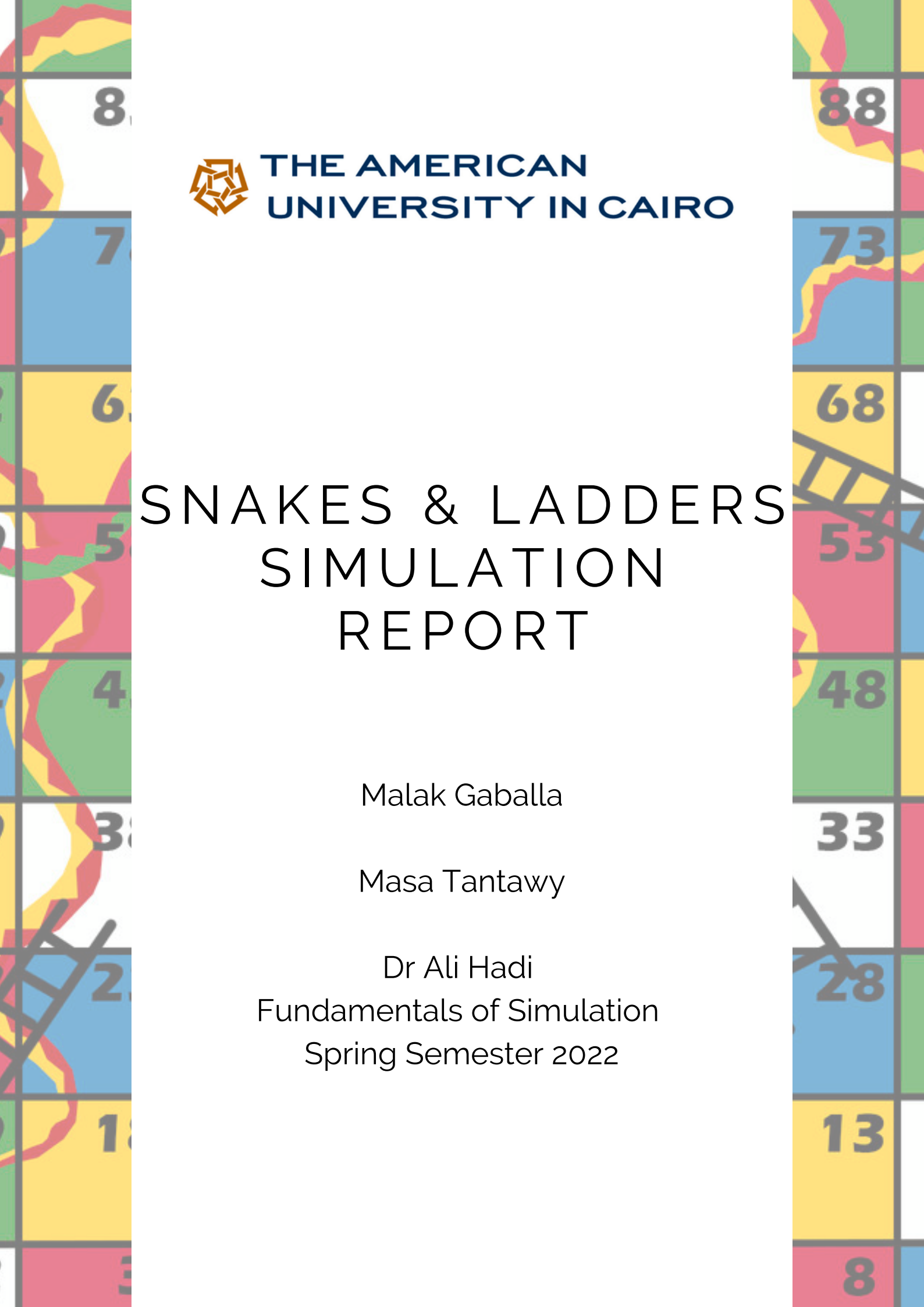


THE AMERICAN UNIVERSITY IN CAIRO

SNAKES & LADDERS SIMULATION REPORT

Malak Gaballa

Masa Tantawy

Dr Ali Hadi
Fundamentals of Simulation
Spring Semester 2022

TABLE OF CONTENTS

Introduction

Snakes and ladder is an ancient board game that originated in India. The central concept is that the game consists of a minimum of 2 players, all initially starting at zero in a numbered grid, and each gets a turn to roll the die. The number they get is the steps they have to advance. Encountering a snake on their stopping cell indicates they have to go back to the cell at the end of the snake whereas a ladder refers to the opposite. The objective of the game is to reach the final square before your opponents to win.

This project originated out of curiosity to understand the different scenarios to win the game. Some of the questions that were raised include:

- How many turns are we expected to play to win the game?
- Is it possible to win the game even though the player didn't land on a ladder throughout the whole game?
- Can we encounter snakes and still win the game? If so, how many snakes?
- Does the dimension of the board affect the number of turns for each player?
- Does the number of ladders and snakes existing in a board game affect the number of turns?

Hence, a code was written in R and Python, both almost identical, to analyze the game results by simulating playing the game a large number of times under different conditions. This report will explain in detail all the functions created in the code and analyze the results.

The Game Board

Before playing the game, we had to design the game board first. This process required the creation of several user-defined functions in order to grant the user the ability to create his/her own board game with the preferred dimensions. Both Python and R codes perform the same tasks, with differences in syntax, except for plotting the board which is only available in R.

The code begins with create_board function which has the dimension of the board as the input to allow boards of different sizes, such as 10X10 or 5X5. It returns a dataframe that is identical to a plan board grid, starting at 1 in the bottom left corner and ending at the dimension square in the last cell.

```
> create_board(10)
    V1 V2 V3 V4 V5 V6 V7 V8 V9 V10
1  100 99 98 97 96 95 94 93 92  91
2   81 82 83 84 85 86 87 88 89  90
3   80 79 78 77 76 75 74 73 72  71
4   61 62 63 64 65 66 67 68 69  70
5   60 59 58 57 56 55 54 53 52  51
6   41 42 43 44 45 46 47 48 49  50
7   40 39 38 37 36 35 34 33 32  31
8   21 22 23 24 25 26 27 28 29  30
9   20 19 18 17 16 15 14 13 12  11
10   1  2  3  4  5  6  7  8  9  10
```

```
> create_board(5)
   V1 V2 V3 V4 V5
1 21 22 23 24 25
2 20 19 18 17 16
3 11 12 13 14 15
4 10  9  8  7  6
5  1  2  3  4  5
```

The next function custom_sl creates random snakes and ladders if the dimension received is not 10, since 10 is the default board size so the snakes and ladders are predetermined. A random discrete uniform sample of size twice the dimension is selected from the numbers of the cells and then divided into an equal number of starts and ends. According to the starts and ends, the type is set as a snake or a ladder.

The sample is selected without replacement to ensure that no 2 snakes or ladders begin or end at the same place. The output of this function returns the list of ladders followed by a list of snakes as shown below.

```
> custom_sl(5)
[[1]]
  start end
1     7  20
2    12  24

[[2]]
  start end
3    15  14
4     6   1
5    18  10
```

```
> custom_sl(8)
[[1]]
  start end
1    16  55
2    19  47
6    43  44
7    39  60
8     2  10

[[2]]
  start end
3    46  12
4    22  17
5    15  13
```

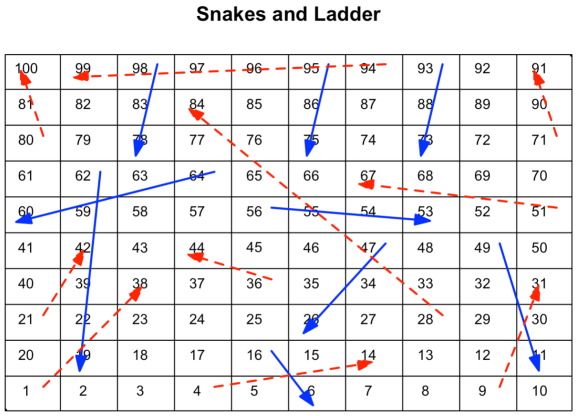
The function find_tile finds the coordinates or location of the tiles on the board. The inputs of this function are the tile that we want to locate and the dimension of the board because the location of the tile is heavily influenced by the board dimension chosen by the user. Inside this function, the function create_board is called to return the number-grid dataframe of the chosen dimension and locate the tile coordinates accordingly. The sole purpose of this function is to facilitate the process of locating and placing snakes and ladders on the board game.

```
> find_tile(45,10)
[[1]]
[1] 5

[[2]]
[1] 5
```

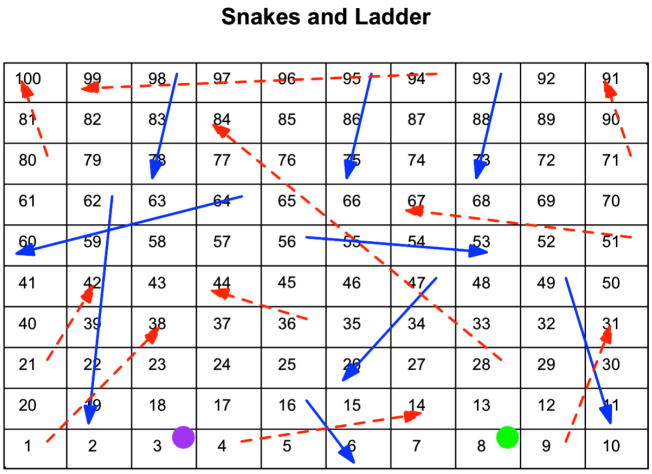
```
> find_tile(40,8)
[[1]]
[1] 5

[[2]]
[1] 8
```

In R, plot_board is a function with input parameters size, snakes and ladders as 2 dataframes, and the positions of players 1 and 2 set to zero by default. This function first plots the numbered grid based on the size without snakes and ladders and then plots the snakes as dashed red arrows and ladders in solid blue lines. Finally, the positions of player 1 and player 2 are plotted on the grid as purple and green circles respectively. The colors of the plots were all chosen to accommodate all forms of color blindness using an online color-blindness simulator. The first code snippet below displays the board with players at initial position zero while the second code snippet displays the board with the game in action such that player 1 is positioned at 3 and player 2 is positioned at 8.

```
> plot_board(10,ladders,snakes)
```




```
> plot_board(10,ladders,snakes,3,8)
```



Play the Game

All the previous functions were created to ease the creation of the main functions that simulate playing the game therefore they are part of 3 main functions: against_computer, single_game, and game_simulate. The first function permits the user to play against the computer a single game by pressing enter to roll their die on their turn. single_game and game_simulate simulate playing a single game, without the user rolling the die, and return the number of turns played by the winner and the number of snakes and ladders they have encountered.

The parameters of these three functions are almost identical except for game_simulate which needs the number of simulation runs N. The number of sides of the die is an input parameter that allows the user to freely create a die of their desire. Conspicuously, the user needs to specify the dimension of the board which is by default set to 10 and cannot exceed this number. The difficulty level is another common input parameter that accepts 3 different inputs ranging from 1-3. However, it is important to note that this input is valid only if the dimension of the board game is 10; otherwise, the number of snakes and ladders will be created at random using the custom_sl function explained earlier. If difficulty was chosen to be 1 then the board will contain the greatest number of ladders and lowest number of snakes. As you increase the difficulty, the number of snakes will increase while the number of ladders will decrease, making it harder and more challenging for the players.

The functions against_computer, single_game, and game_simulate all first check on the dimension entered to ensure that it is less than 10.

Next, snakes and ladders are created, either by custom_sl or according to difficulty level, based on the dimension. To start playing the game, variables for the positions, turns, snakes, and ladders encountered by each player are initially set to zero. Inside a while loop that checks on both players' positions being not in the last cell, a die is rolled for player 1 (which is the user in against_computer), via a discrete uniform sample from 1 to the chosen sides, then increments their turns and increases their position, and checks on snakes and ladders to update their position and their snakes or ladders accordingly. Before this process is repeated for player 2, we make sure that player 1 didn't win yet. This loop is repeated until one of the players reaches the last cell on the board, returning the winner's data back in a single-lined matrix.

Even though these functions are almost identical in terms of input parameters and outputs produced, single_game function plots the board by calling plot_board at the beginning and prints the turns and positions of each player throughout the game while game_simulate only displays the winner's data to speed up simulating playing the game N times when it is called inside play function explained in the following section. As for against_computer, the output is in the form of a game with the user playing against the computer by pressing on the enter button to roll the die and play his turn. Every time a turn is played, the board is plotted showing the positions and movements of each player which are also printed.

The Simulation

After creating the full mechanisms of the game, we created another function called play that simulates playing the game N times in order to facilitate the analysis which will be displayed in the next section.

The algorithm of this function is::

Step 1: The user enters the parameters simulation runs(N), sides of the die, dimension of the board, and difficulty level to the function.

Step 2: A matrix of size NX3 is created to store the number of turns, ladders, and snakes of the winner in each game.

Step 3: The function game_simulate is called N times. This function contains the following algorithm:

- 3.1: check the dimension specified by the user.
  - 3.1.a: If dimension is greater than 10, write "Please enter a number not greater than 10!" and break.
  - 3.1.b: If dimension is 10, check the difficulty level chosen and create the number of ladders and snakes accordingly (higher difficulty, more snakes and fewer ladders)
  - 3.1.c: If dimension is less than 10, use the custom_sl function to randomly create snakes and ladders.
- 3.2: Set initial player positions to zero and create three variables for each player's turn and how many snakes and ladders they landed on.
- 3.3: Create a while loop to have the players continue playing until either one of them wins and reaches the last cell. Inside the while loop is the following algorithm:

3.3.a: Die rolled for player 1 using a discrete uniform sample from 1 to the chosen number of sides of the die.

3.3.b: Player 1's number of turns is incremented.

3.3.c: Player 1's position increases (original position + number on the die).

3.3.d: Check if the player landed on a snake or ladder to update their position on the board. Also, the number of snakes or ladders they've faced will be incremented accordingly.

3.3.e: Check if the player won then the game ends, otherwise the same process is repeated for player 2.

3.4: A single-lined matrix containing the winner's data is returned containing the number of turns taken and the number of snakes and ladders encountered.

Step 4: The results of each game are stored in a row in the matrix.

Step 5: The column means of the matrix are printed to display the average number of turns, snakes, and ladders encountered by the winner in each of the N games.

Step 6: 3 histograms are plotted to display the distribution of each variable in the matrix in all the simulation runs.

In the next section, the simulation results yielded will be analyzed based on different user inputs whether it is the sides of the die, the dimensions of the board, or the difficulty level. By the end of the analysis, we will be able to answer the questions raised earlier.

Simulation Analysis

These are the hypotheses that we wish to test by analyzing and simulating the function play N times with different parameters.

- *Dimensions of the board:* the greater the dimension the harder it is to win the game (more turns, more time, etc..)
- *Difficulty levels (1,2,3):* as the difficulty level increases, the number of turns and snakes increase but the number of ladders decreases.
- *Sides of the die:* as the number of sides of the die increases, the number of turns should decrease

In all the following outputs, the number of simulation runs is set to 10,000. This number was chosen to optimize the accuracy of the results with the time taken to yield the output. Needless to say, as the number of simulations increases, the validity of the output increases. We also set the seed to ensure that our code could be replicated and yields the same output.

```
> N=10000
> set.seed(1949)
```

1. Board Dimensions

Hypothesis: *The greater the dimension the harder it is to win the game (more turns, more time, etc..).*

A standard 6-sided die (by default) is used in all the following simulations.

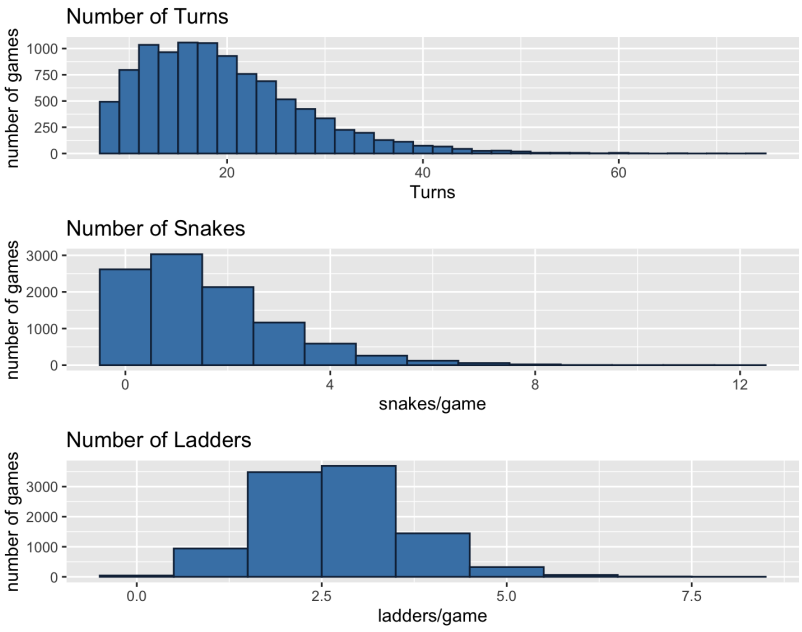
1.1: dimension = 10 (default difficulty =1)

> play(N,dimension=10)

Average of 10000 games

Turns Ladders Snakes

20.1425 2.6841 1.5834



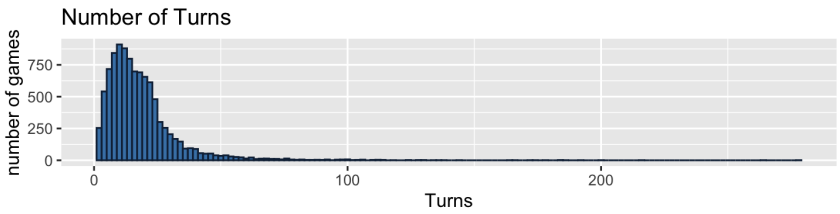
1.2: dimension = 9

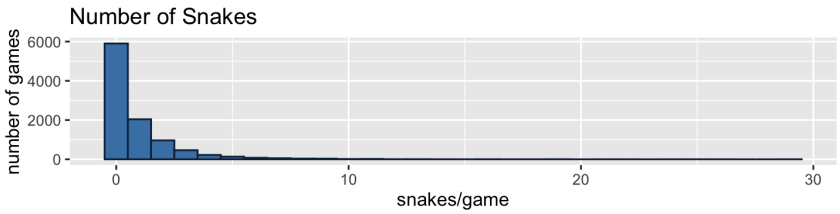
> play(N,dimension=9)

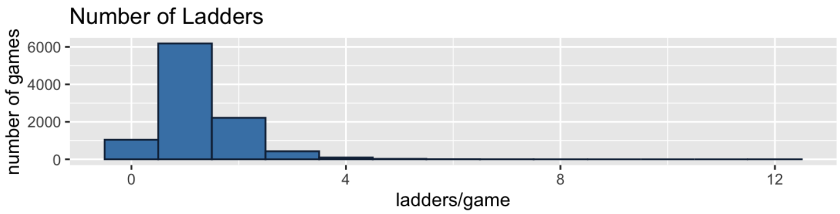
Average of 10000 games

Turns Ladders Snakes

18.9208 1.2474 0.9295







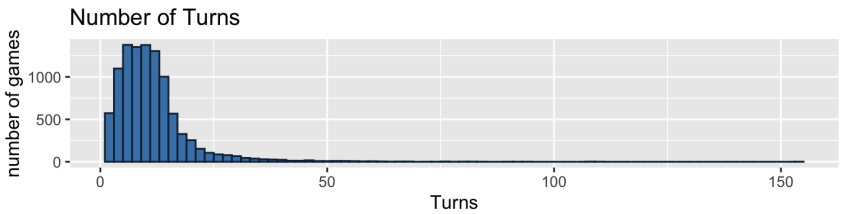
1.3: dimension = 7

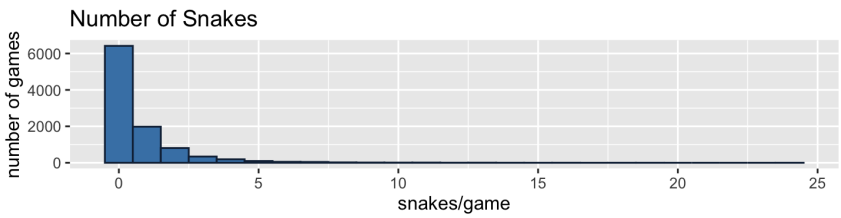
> play(N,dimension=7)

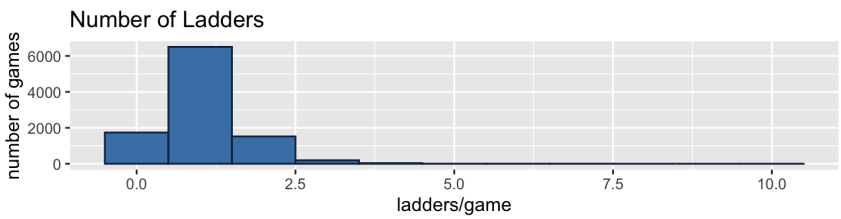
Average of 10000 games

Turns Ladders Snakes

11.8037 1.0338 0.7241







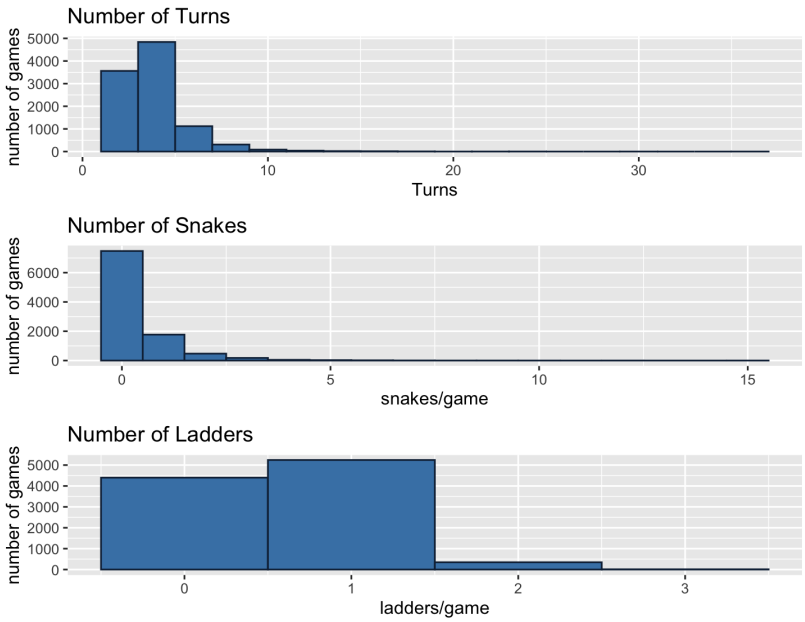
1.4: dimension = 4

> play(N,dimension=4)

Average of 10000 games

Turns Ladders Snakes

4.2368 0.5973 0.3785



According to the outputs shown above and the constructed table in the right, it can be concluded that the number of dimensions affect the number of turns taken and the ladders and snakes encountered. As the board dimension increases, the number of turns taken increases which is explainable since it takes more time to complete the game hence more turns. The figures also point out that the number of ladders encountered by the winner increase as the dimension increases, which is also justifiable because

Dimensions	Turns	Ladders	Snakes
4	4.2368	0.5973	0.3785
7	11.8037	1.0338	0.7241
9	18.9208	1.2474	0.9295
10	20.1425	2.6841	1.5834

the increase in dimension implies that more ladders are present on the board. Similarly, more snakes are encountered by the winner as the dimension increases.

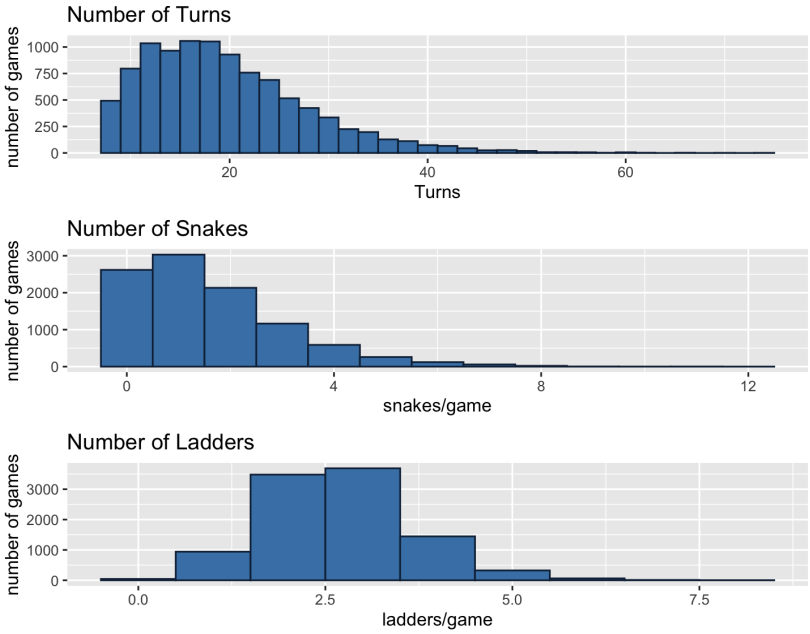
2. Difficulty Levels

Hypothesis: *As the difficulty level increases, the number of turns and snakes increase but the number of ladders decreases.*

The difficulty level is only available when the dimension of the board is 10. As in the previous sub-section, a standard 6-sided die (by default) is used in all the following simulations.

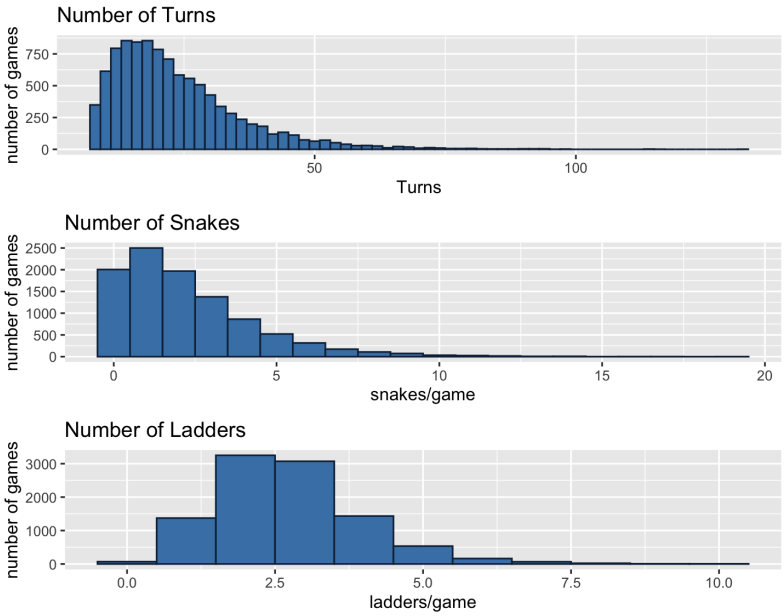
2.1: difficulty = 1

```
> play(N,difficulty=1)
Average of  10000 games
  Turns Ladders  Snakes
20.1425  2.6841  1.5834
```



2.2: difficulty = 2

```
> play(N,difficulty=2)
Average of  10000 games
  Turns Ladders  Snakes
24.1527  2.7254  2.2348
```



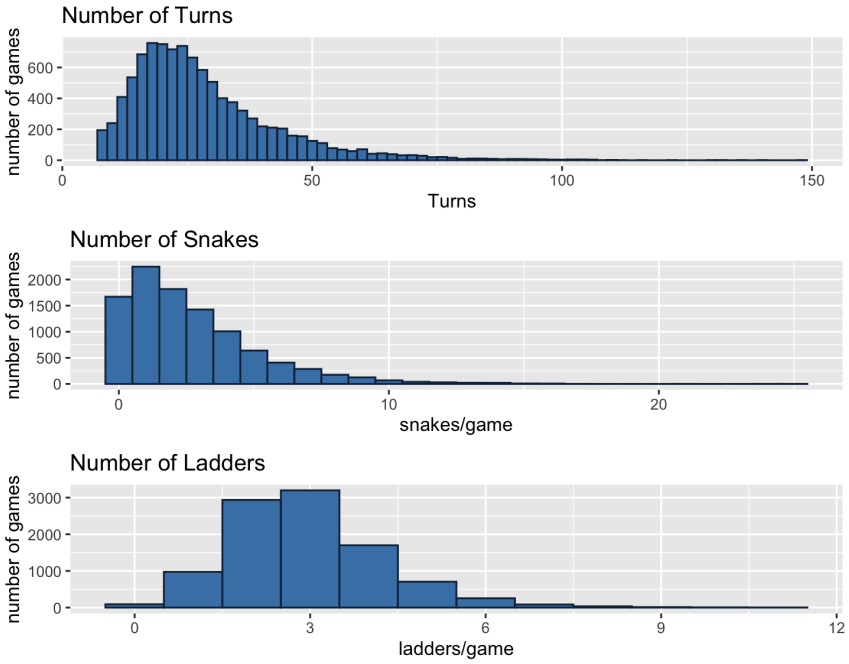
2.3: difficulty = 3

> play(N,difficulty=3)

Average of 10000 games

Turns Ladders Snakes

28.9012 2.9372 2.6738



After simulating the game based on the 3 difficulty levels offered, we concluded that the winner's data is greatly affected. On average, the number of turns increases by 4 as the difficulty increases. As the difficulty increases, the number of snakes increase and unexpectedly the number of ladders as well. Despite the decrease in the number of ladders in the board as the difficulty level increases, one interpretation for this unforeseen contradiction in the results is that the player encounters more snakes in higher difficulties hence being more like to slide down and start over, with the increased likelihood to land on a ladder.

Difficulty Level	Turns	Ladders	Snakes
1	20.1425	2.6841	1.5834
2	24.1527	2.7254	2.2348
3	28.9012	2.9372	2.6738

On average, we can say that the winner lands on 3 ladders regardless of the difficulty level. The number of snakes, however, changed from 1.6 to 2.67 when increasing the difficulty level from 1 to 3 which is almost double. This is expected since the number of snakes increases as the difficulty level increases, thus increasing the probability that a player would land on a snake.

3. Sides of the Die

Hypothesis: *As the number of sides of the die increases, the number of turns should decrease.*

In all the following simulations, the default parameters are used, which are board size fixed to 10 (dimension 10X10) and difficulty 1, to ensure that any changes in the plots are dependent only on the number of sides of the die used.

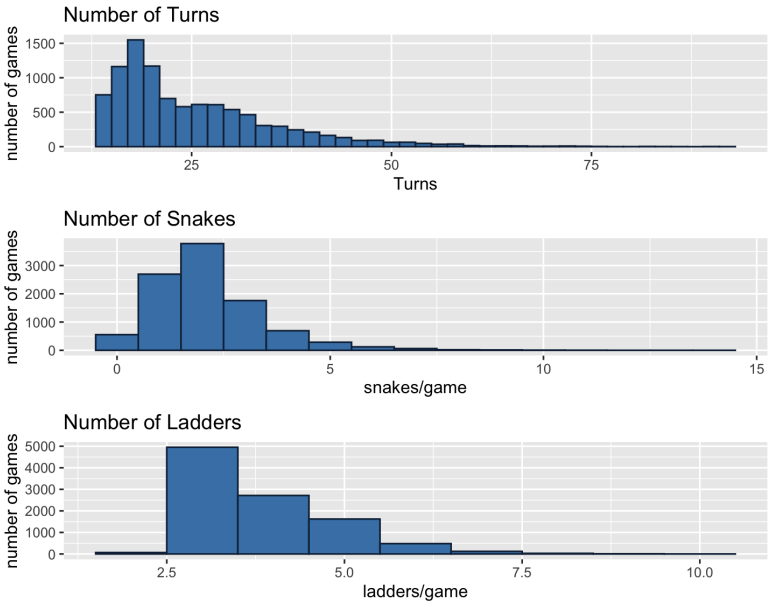
3.1: sides= 2

```
> play(N,sides=2)
```

Average of 10000 games

Turns Ladders Snakes

25.7714 3.8034 2.1357



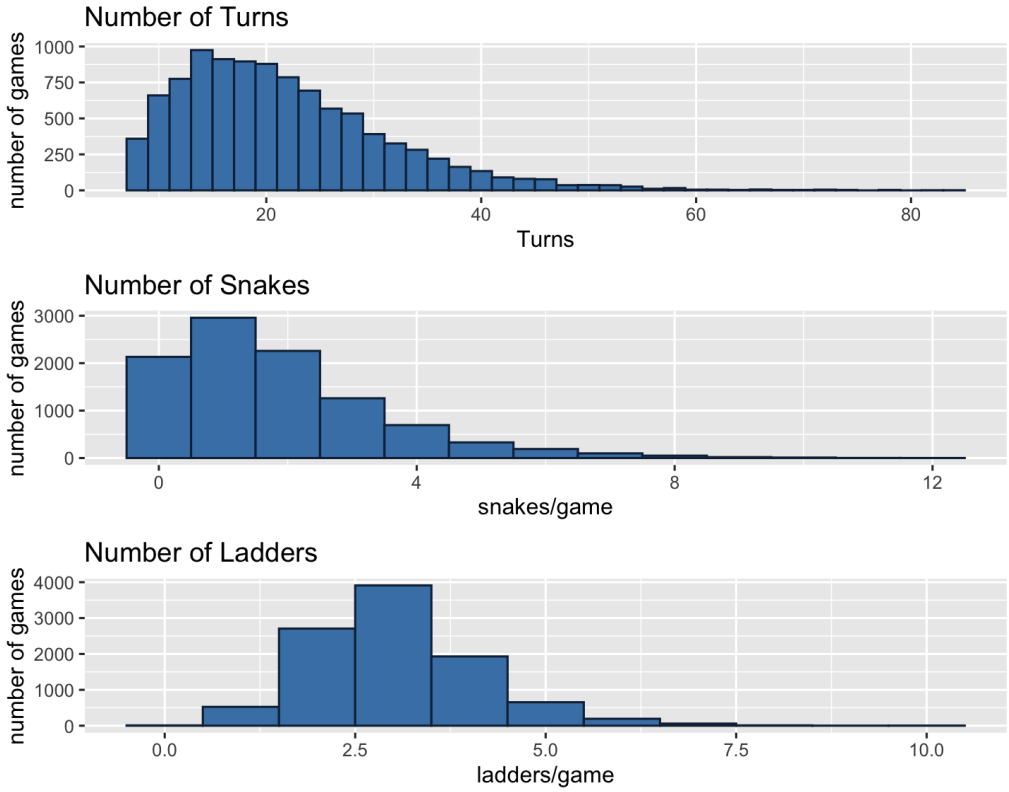
3.2: sides = 5

```
> play(N,sides=5)
```

Average of 10000 games

Turns Ladders Snakes

22.2356 3.0350 1.8205



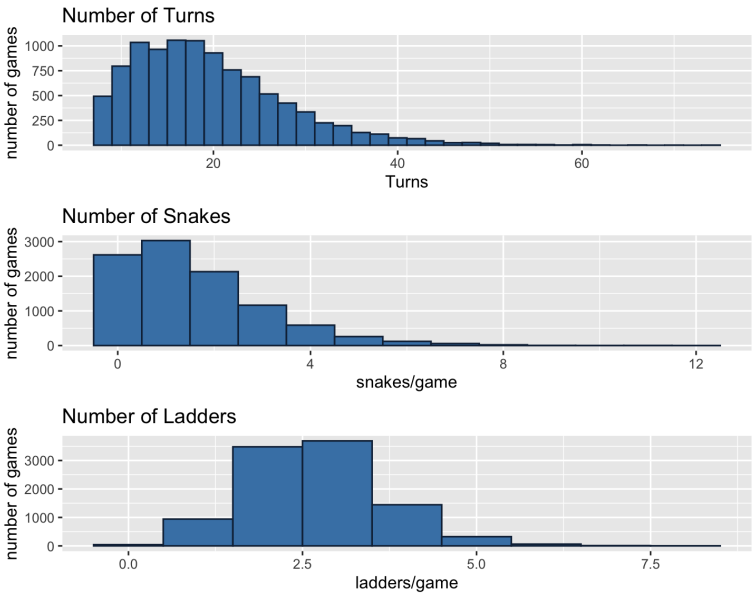
3.3: sides = 6 (default)

> play(N,sides=6)

Average of 10000 games

Turns Ladders Snakes

20.1425 2.6841 1.5834



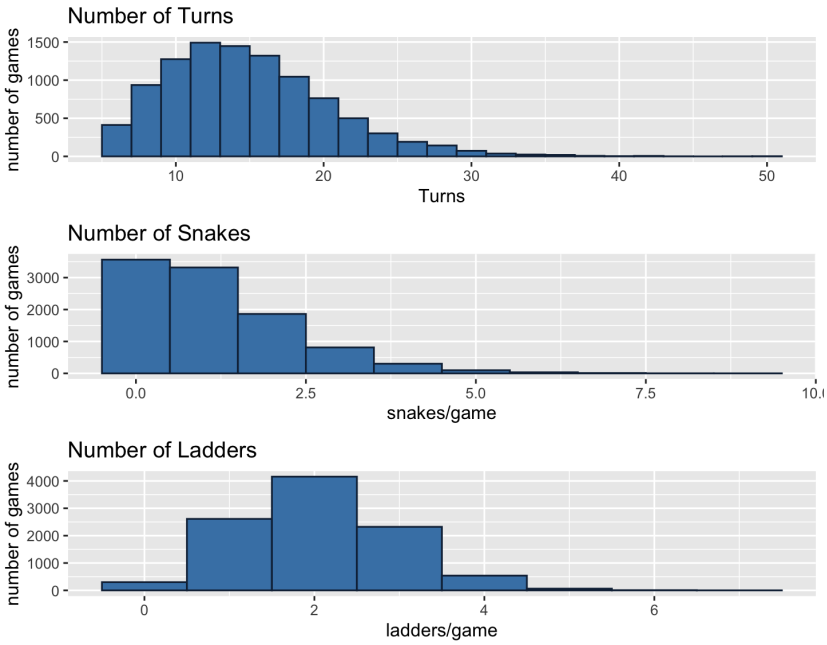
3.4: sides = 9

> play(N,sides=9)

Average of 10000 games

Turns Ladders Snakes

15.3960 2.0410 1.1475



Using the default 10x10 game board, we wanted to simulate the game using different numbers of sides for the die. We noticed the trend: as the number of sides increased, the number of turns significantly decreased from almost 26 to 15 or 16 turns per game. Also, the winner encountered on average around 4 ladders when the die had 2 sides then decreased to around half which is 2 ladders when the die had 9 sides. Likewise, the number of snakes decreased from 2 to 1 snake per game on average when the sides increased from 2 to 9. In other words, when the number of sides increases, the number of turns will decrease since the die contains larger numbers that could help the player win with a minimal number of turns.

Sides of the Die	Turns	Ladders	Snakes
2	25.7714	3.8034	2.1357
5	22.2356	3.0350	1.8205
6	20.1425	2.6841	1.5834
9	15.3960	2.0410	1.1475

Conclusion

In conclusion, the main aim of this project was to examine the different scenarios of playing snakes and ladders by simulating playing the game a large number of times and performing statistical analysis on the results of the winner. The code used was written in both R and Python, with the board plot only in the former language. The main functions used were against_computer, single_game, game_simulate, and play; they allowed the user to play a single game against the computer, simulate a computer-based single game, and run N simulations of the game relatively.

The algorithm of these functions depends on other user-defined functions, and they allow the user to choose the number of sides of the die, the size of the board, and the difficulty level for a board of size 10. First, in each game the size of the board and difficulty level are used to create snakes and ladders then a random discrete uniform sample is selected as the number on the die for the first player. Next, their position, turns, snakes and ladders are updated before checking if they have reached the last cell on the board. Otherwise, these steps are repeated for the second player and continue until a player wins. The board is plotted at the beginning and in against_computer, the positions of each player change every turn. In the function play, the number of turns, snakes and ladders of the winner in N games is recorded in a matrix and a histogram is plotted for each variable.

The results of analyzing the average number of turns taken to win the game, and the average numbers of snakes and ladders encountered by the winner in a game in 10,000 simulation runs indicate that they are

affected by the factors dimension of the board, difficulty level, and sides of the die. For the first factor, as dimension increases, turns, snakes and ladders increase, and likewise as difficulty levels increase. Lastly, as the number of sides of the die increase, all 3 variables decrease.