Monopoly and Python

One student's foray into creating simulations

Anusika Nijher

Information Technology 12
Mr. Cabralda
April 10, 2018

INTRODUCTION

On December 8, 2016 Matt Parker released a video with Hannah Fry about the probabilities of hitting certain spots on a Monopoly board. Hannah used Markov Chains to calculate her results and Matt used a Python script. Their results matched perfectly despite using two different methods. However, Matt only ran his simulation through Python and did all his analysis in Excel. Using Matt's code as a basis, I created my own simulation and graphed my results using the matplotlib library. I also added a more complex simulation where the players could buy spots and be forced to pay rent. I tracked the amount of money a player had per game and could analyze the effect of a player's risky factor (how likely a player is to buy a spot, for example 20% risky vs 50% risky) in winning a game.

OBJECTIVES AND GOALS

1. Create a Player class and a Property class
    a. The Player class will include attributes such as position, amount of money, owned locations and risky factor
    b. The Property class will include attributes such as location on board, colour, price to rent and price to buy
2. There should be a dice rolling function
    a. If a player rolls doubles they get to roll again
    b. If they roll three doubles they go straight to jail
3. The simple simulation should include the following:
    a. Moving the player based off their roll
    b. Take actions if they hit a community chest or chance card
    c. Keep track of turns and games completed
    d. Only have one player moving around the board
4. Graphs based off simple simulation
    a. Graph the chance of hitting any one spot on the Monopoly board
    b. Create a second graph that shows that chance of hitting any colour on the Monopoly board
5. Create a complex simulation that includes the following:
    a. Have more than one player at a time
    b. Tracking how much money a player has
    c. Include the ability for players to buy spots
    d. Charge rent depending on if the spot is owned or not (charge different rent for railroads and utilities based on how many are owned)
6. Graphs based off complex simulation
    a. Show how the amount of money a player has changes over the amount of turns (include the player's risky factor)
    b. Create bar graph that shows the effect of turn order on amount of wins
    c. Create bar graph that show the effect of risky factor on amount of wins

PSEUDO CODE

The code I have attached has been thoroughly commented and is slightly over 700 lines long. To provide a general overview of the various functions it is easier to explain through pseudo code.

```
class Player(object):

    def __init__ (attributes): Set up all the attributes of player
including amount of money, locations owned and risky factor etc...
Each time I make a player I use this class

    def want(self, location): Generates a random number and
compares it to the players risk factor to see if they want to buy
that location

class Property(object):

    def __init__(attributes): set up all the attributes of the
property such as location on board and its colour etc… Also adds
itself to a master list of properties

def RollDice(player, jail):

    Choose two random integers between 1 and 6. If a player rolls
doubles they get to roll again. If they roll 3 doubles they have to
go to jail. If jail is active and the player is in jail, they must
roll doubles to get out otherwise they are stuck in jail.

def simple_simulation(games, numturns, jail):

    First add properties list, list of railroadd and list of
    utilities

    while completed games is less than specified amount of games:

        Generate player, roll dice, create community chest cards
        and chance cards

        while completed turns is less than specified turns:

            if dice roll says stay in jail and jail is active:

                stay in jail

            elif jail is active and player has been in jail for 3
            turns:

                player is out of jail
```

```
                elif dice roll sends player to jail:
                    go to jail

                else:
                    Move player however many spots the roll tells
                    them to
                    if that spot says "Go to Jail":
                        player goes to jail
                    elif: player is on community chest spot:
                        play community chest card and move player
                        accordingly
                    elif: player is on chance spot:
                        play chance card and move player
                        accordingly
            After turn is completed increase the amount of hits on the
            spot they ended up at
            Increase the amount of turns completed
        Once amount of specified turns are completed, increase amount
        of games completed
    return list of properties to be used for analyzation
def calc_time(games, numturns):
    runs a simple_simulation with jail active to see how many turns
are spent at each spot. Converts the returned list into percentages
and graphs them using matplotlib

def calc_hit(game, numturns):
    Runs a simple_simulation with jail inactive to see the chance
    of hitting any one spot. Converts the returned list into
    percentages and graphs them using matplotlib
def label_bars(bars):
    Adds the value of each bar in a bar graph on top of its
    respective bar
```

```
def compare_hits_time(games, numturns):
    Runs the simple_simulation twice, once with jail active and
    once with jail inactive and graphs the two runs side by side
def complicated_simulation(games, numturns, players):
    First add properties list, list of railroadd and list of
    utilities, jail is active always,
    while completed games is less than specified amount of games:
        Generate player, roll dice, create community chest cards
        and chance cards
        while completed turns is less than specified turns:
            for player in list of players:
                Roll dice
                if dice roll says stay in jail:
                    stay in jail
                elif  player has been in jail for 3 turns:
                    player is out of jail, the amount of money
                    they have decreases by 50
                elif dice roll sends player to jail:
                    go to jail
                else:
                    Move player however many spots the roll
                    tells them to
                    if that spot says "Go to Jail":
                        player goes to jail
                    elif: player is on community chest spot:
                        play community chest card and move
                        player accordingly, now includes
                        cards that change the amount of money
                        a player has
                    elif: player is on chance spot:
                        play chance card and move player
                        accordingly,now includes cards that
```

change the amount of money a player has

increase the amount of hits on the spot they ended up at

if the spot the player landed on is not owned and can be bought:

check if they want to buy it

if they do:

Decrease the amount of money they have by the cost of the property

Change the status of the property to owned

Add the property to list of properties owned by player

elif: the spot the landed on is owned and it is not owned by the active player:

if the property is a utility:

Cycle through list of players until you find the owner of the utility and see how many unitites they own

if the opponent only owns one:

Active player's amount of money decreases by 4 times their roll

Opponent gains that amount

elif the opponent owns two:

Active player's amount of money decreases by 10 times their roll

Opponent gains that amount

elif the property is a railroad:

Cycle through list of players until you find the owner of the utility and see how many unitites they own

if the opponent owns one railroad:

Active player's amount of money decreases by 25

Opponent gains that amount

elif the opponent owns two railroad:

Active player's amount of money decreases by 50

Opponent gains that amount

elif the opponent owns three railroad:

Active player's amount of money decreases by 100

Opponent gains that amount

elif the opponent owns four railroad:

Active player's amount of money decreases by 200

Opponent gains that amount

else:

Charge active player price of current locations rent

Cycle through list of players, find the owner of the property and have them gain that same amount

if player is on Go:

Increase the amount of money they have by 200

Add however much money the player has at the end of their turn to a master list, increase the amount of turns they've completed by one

Once all players have completed one turn increase amount of completed turns by one

At the end of each game, calculate winner by using values of property (cost to rent property) and the amount of money they have at the end of the game, increase winner's amount of total wins by one

Append the master list of money tracking to a larger list that contains all the money tracking for each game for

```
            each player

    return list of properties, and list of players

def calc_monies(games, numturns, list_of_players):

    Runs a complicated_simulation with specified number of games,
    turns and players, plots a line graph for each player that
    shows how much money they have (on average) at each turn using
    matplotlib

def compare_win(games, numturns, list_of_players):

    Runs a complicated_simulation with specified number of games,
    turns and players, makes a bar graph of win percentage for each
    player, each player has the same risky percentage in this
    function, returns a bar graph made with matplotlib

def compare_win_risky(games, numturns, list_of_players):

    Runs a complicated_simulation with specified number of games,
    turns and players, makes a bar graph of win percentage for each
    player, each player has a different risky percentage in this
    function, returns a bar graph made with matplotlib
```

GRAPHS AND ANALYSIS

1. Chance of Hitting Any Spot on the Monopoly Board

In Figure 1, there are 2 main things to note. Firstly, the spot that is mostly like to be hit is Jail at 6.31%. This makes sense as there are multiple ways to get to Jail (community chest and chance cards, hitting “Go To Jail”, or rolling doubles 3 times). Secondly, “Go To Jail” is at 0% because no player ends their turn at that spot. They end their turn at Jail.

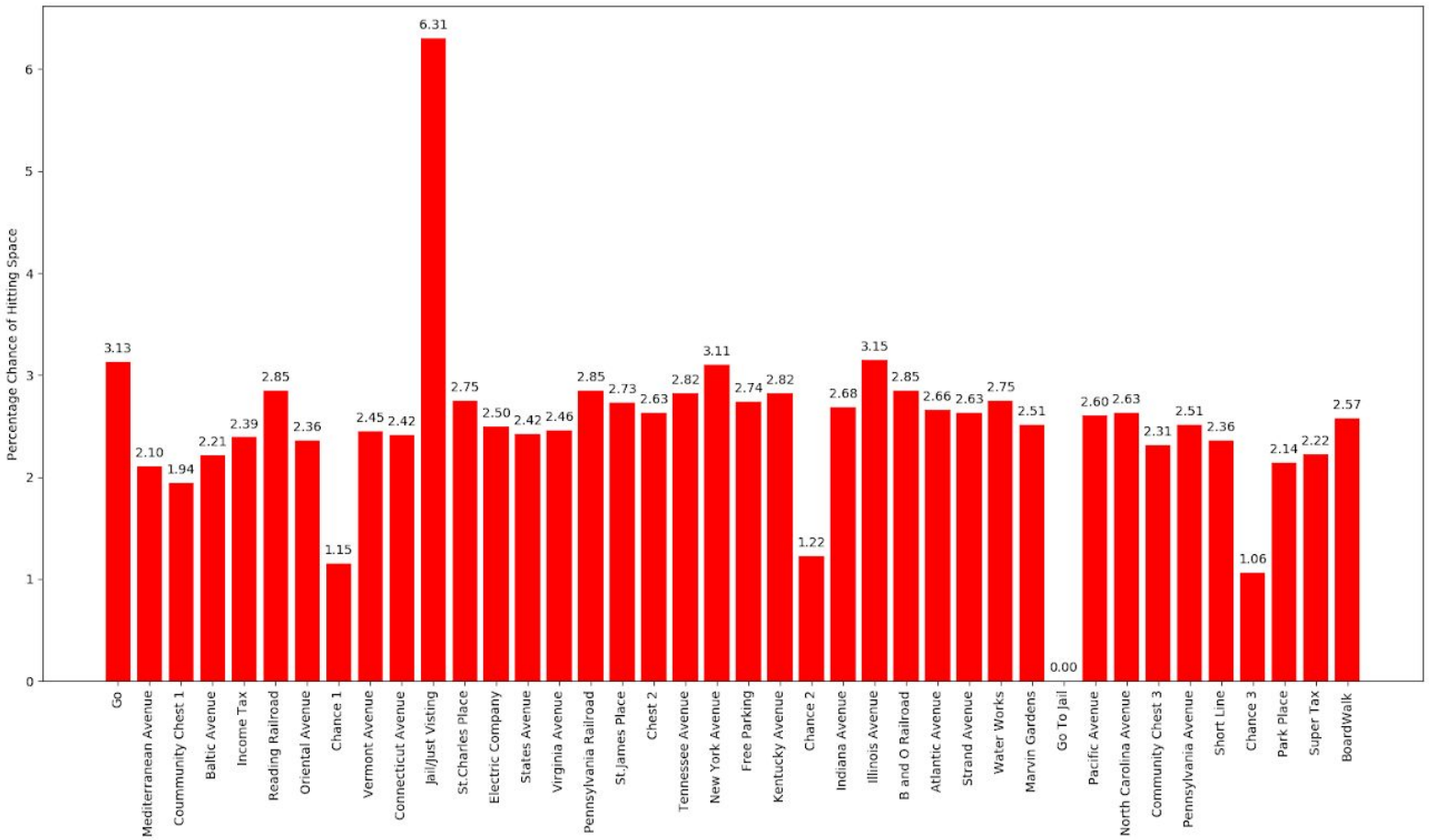

Figure 1: Percentage Chance of Hitting Any One Spot on the Monopoly Board (10,000 Games at 100 Games/ Turn)

2. Percentage of Turns Spent at Each Space

In Figure 2, the same general spikes appear as in Figure 1, however they have different values. Jail now has a 14.77% amount of turns spent. This is because a once a player is in jail, they can only leave if they roll doubles or if they have already spent three turns in jail. The drops in all the spots are to compensate for this large increase in amount of turns spent at Jail.

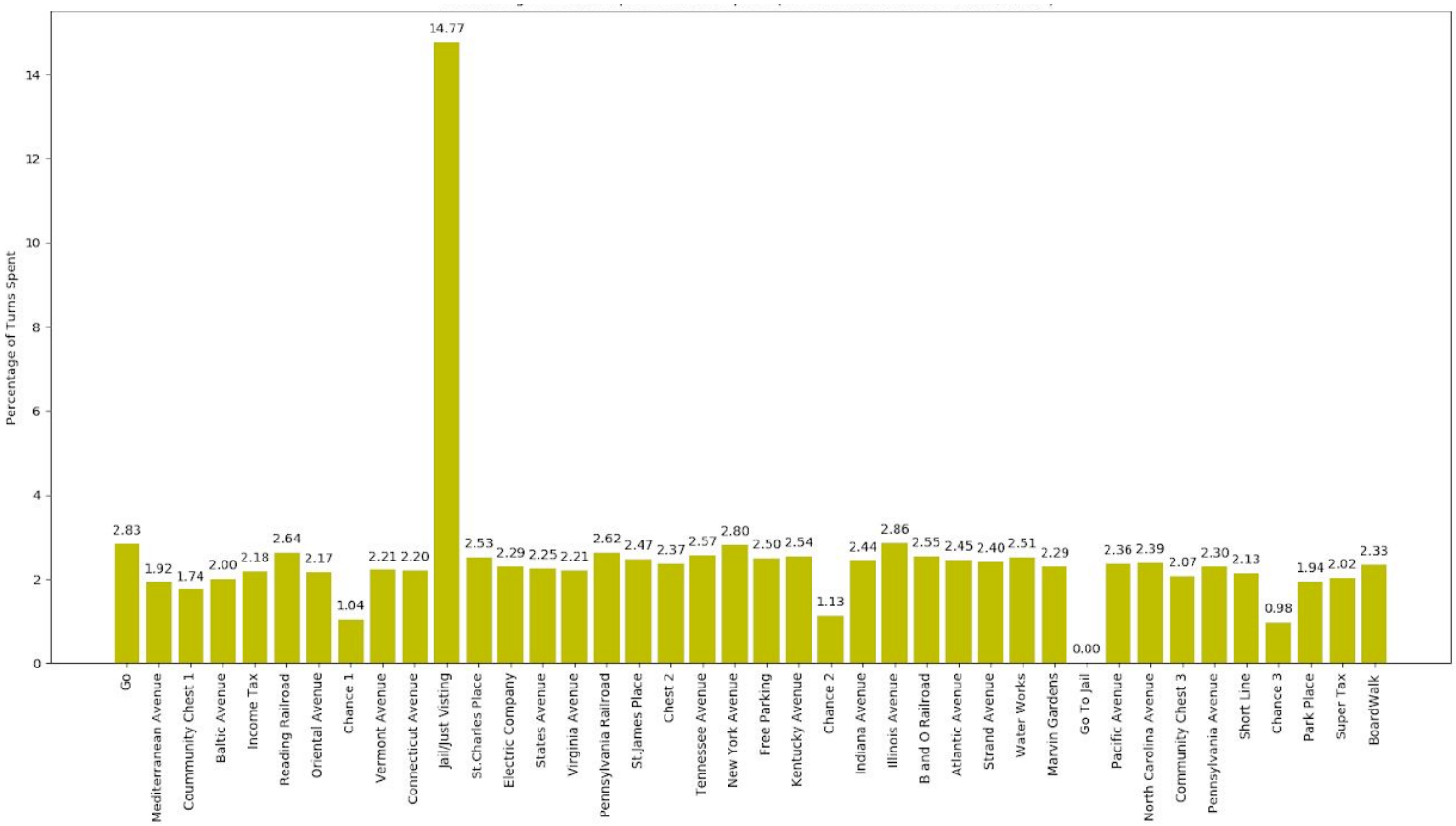

Figure 2: Percentage of Turns Spent at Each Space (10,000 Games at 100 Games/ Turn)

3. Comparison Between Percentage of Turns Spent and Percentage Change of Hitting Each Space

Figure 3 is a comparison between Figure 1 and Figure 2. It clearly shows that the biggest difference between *turns spent* and *chance of hitting* is Jail. In terms of challenging myself to use matplotlib in new ways, adding two bars/ location was surprisingly difficult and required setting the width to a variable that would be called repeatedly in different calculations. I also learned how to add a legend to my graphs.

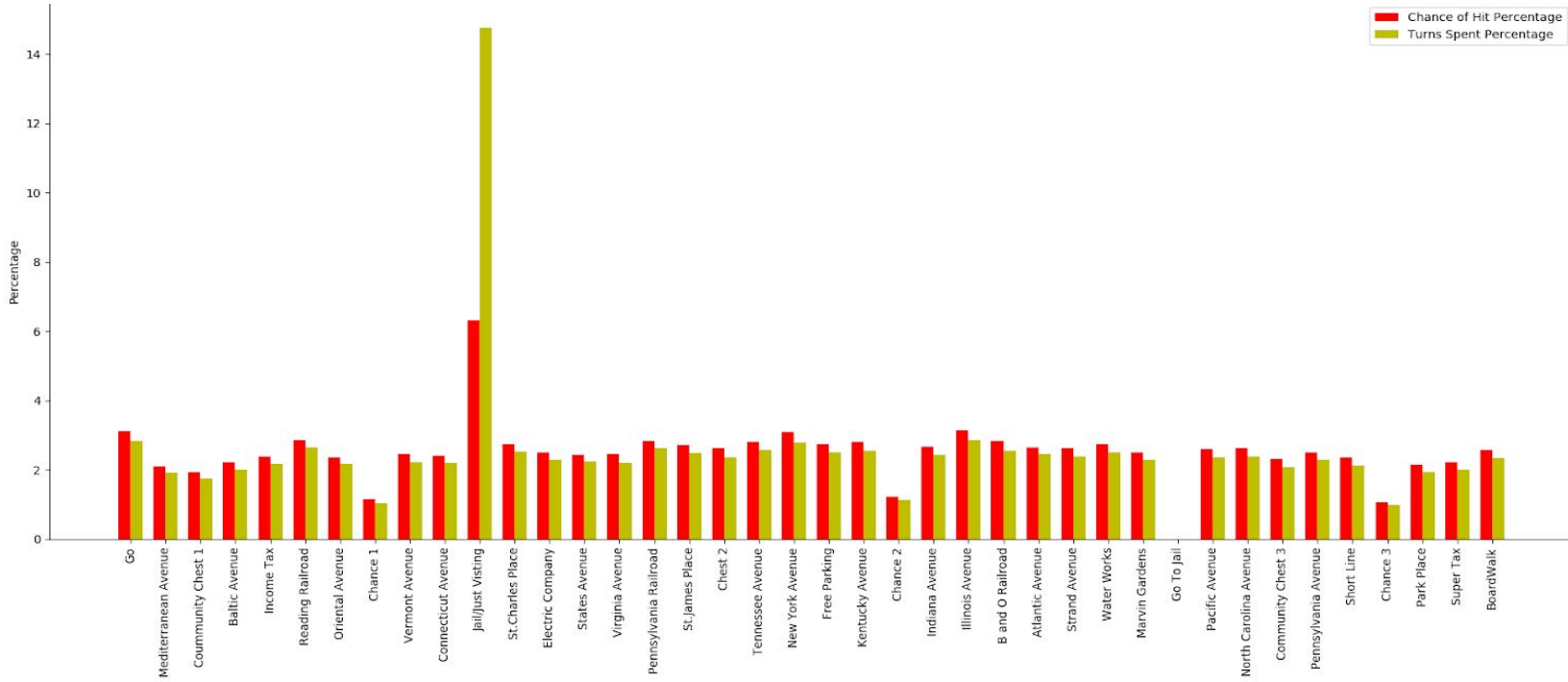

Figure 3: Comparison Between Percentage of Turns Spent and Percentage Change of Hitting Each Space (10,000 Games at 100 Games/ Turn)

4. What Is the Best Colour to Buy?

Figure 4 was created with jail inactive. This is because the main goal is to find which colour will be landed at most frequently. With jail active, I would have found at which colour most turns were spent at. ‘Null’ refers to the spots “Go to Jail”, “Jail” and “Go.” Although Railroads/Stations are the most likely to be hit out of buyable locations they are not the most profitable as their rent depends on how many other railroads are owned. That leaves Orange as being the best colour to buy and Red as a close second. This graph also required me to learn how to colour each bar individually.

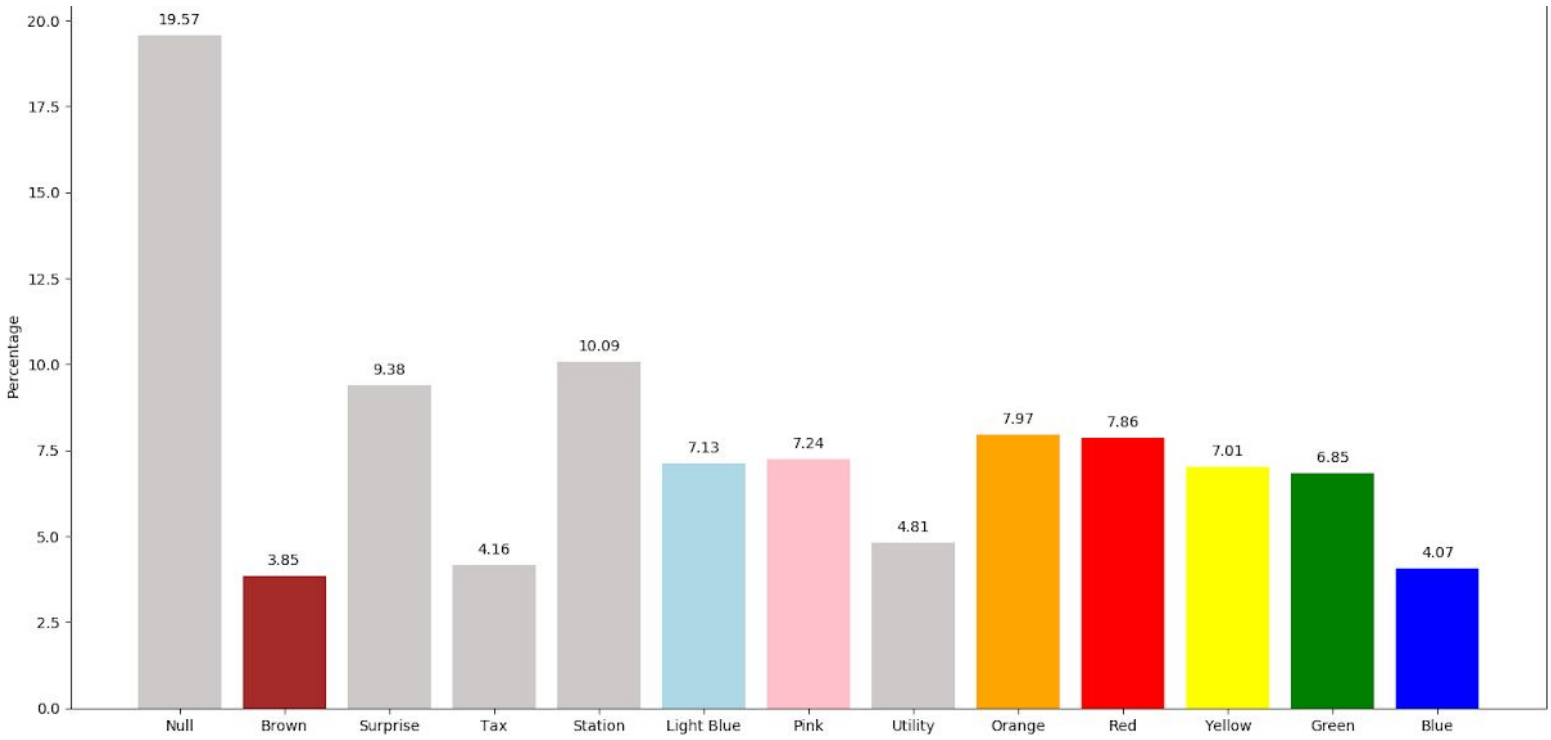

Figure 4: Percentage Change of Hitting Each Colour (50,000 Games at 30 Games/ Turn)

5. The Effects of Turn Order

In Figure 5, it is clear that turn order does not have a massive effect on the win percentage of a player. To try to keep the results as unbiased as possible, I gave all the players the same risk factor of 50%. However, the graph does show that going last does give the least win percentage. I believe that the reason why going second or third would give a higher win percentage is because the first player has a higher chance of hitting community chest and chance cards that send them to Jail, thus reducing the amount of turns they have to buy property.

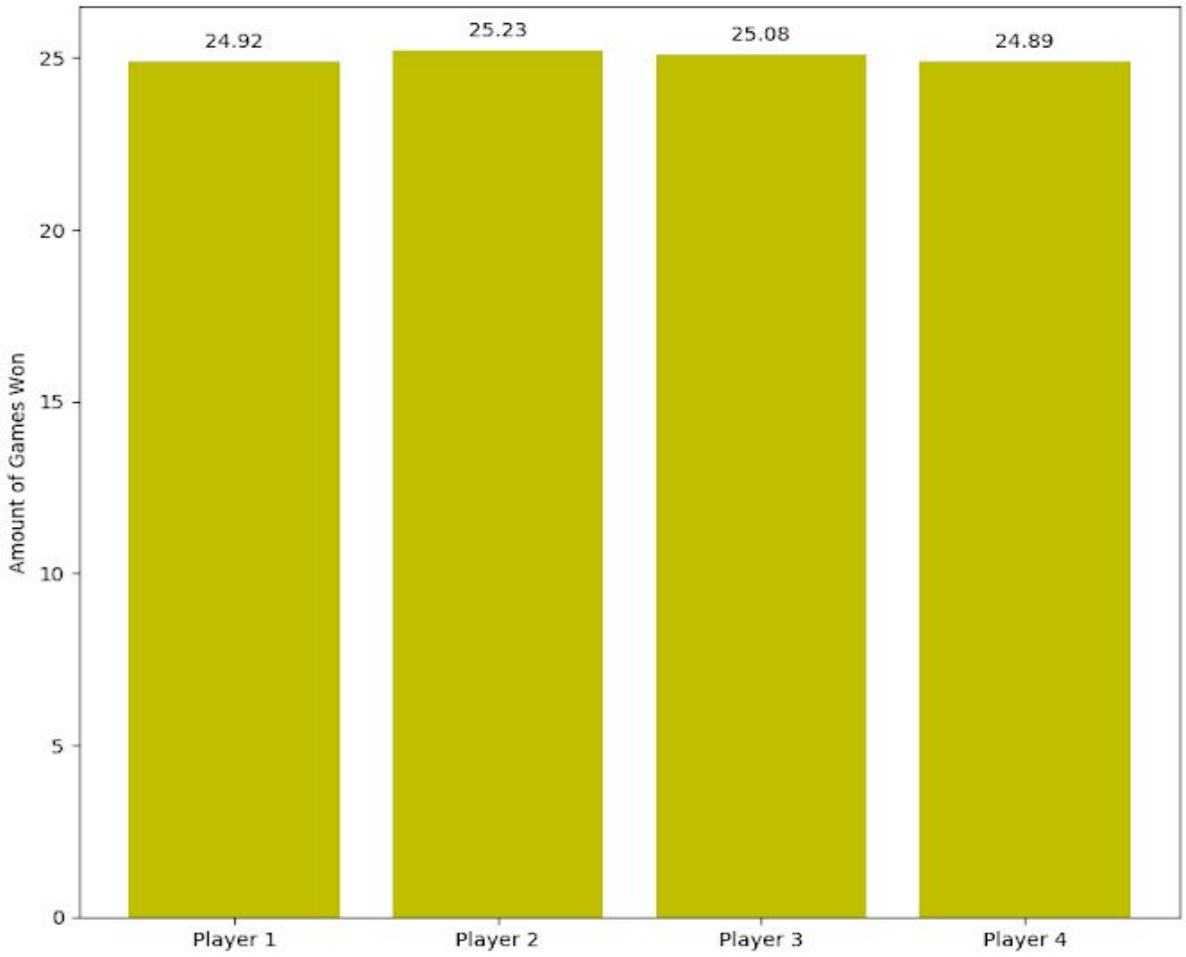

Figure 5: Win Percentage due to Turn Order (100,000 Games at 100 Turns/Game with 4 Players, Each Having Same Risk Factor)

6. The Effects of Different Risk Factors

In Figure 6, I decided to give each player a different risk factor and see how it affects their win percentage. I hypothesized that being on either extreme (100% risky or 20% risky) would result in winning less games. That turns out to be what the results say, although being 100% risky is better than being only 20% risky. The best position seems to be 70% risky, where the player has accumulated enough property value to be in the lead while still having money leftover to secure first place. However, like Figure 5, the percentages are still extremely close.

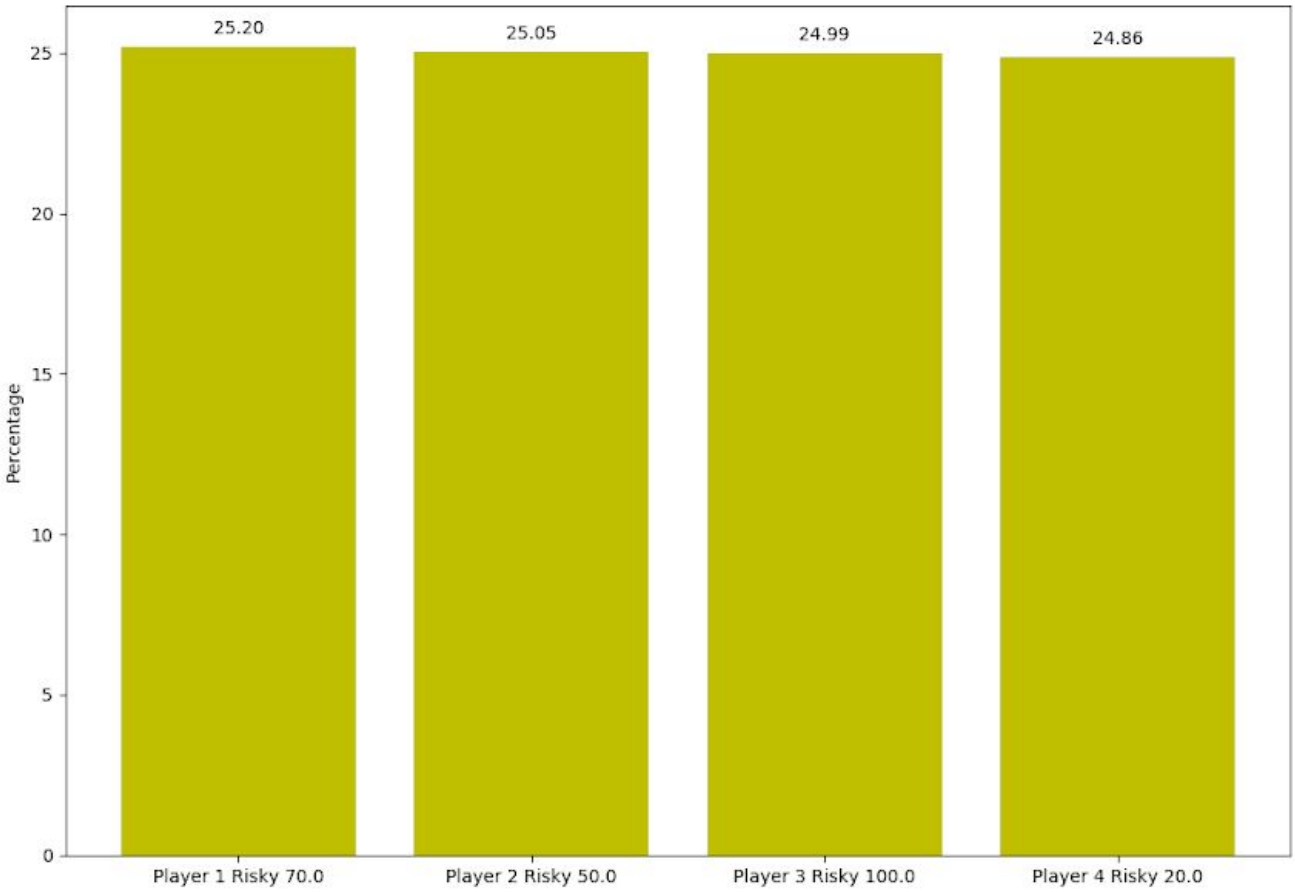

Figure 6: Win Percentage With Different Risk Factors (100,000 Games at 100 Games/Turn with 4 Players)

7. Graphing Amounts of Money

In Figure 7 we can see that averaged out all players gain money over 30 turns regardless of their risk factors. The most interesting area of the graph is between the first 1-2 turns. Player 1 is likely to have lost money as they have a 100% risk factor, therefore buying any property they can. Player 3, despite having a 0% risky factor ended up with the most money on average (probably due to chance and community chest cards). However, because winning takes property value into consideration they did not win many games.

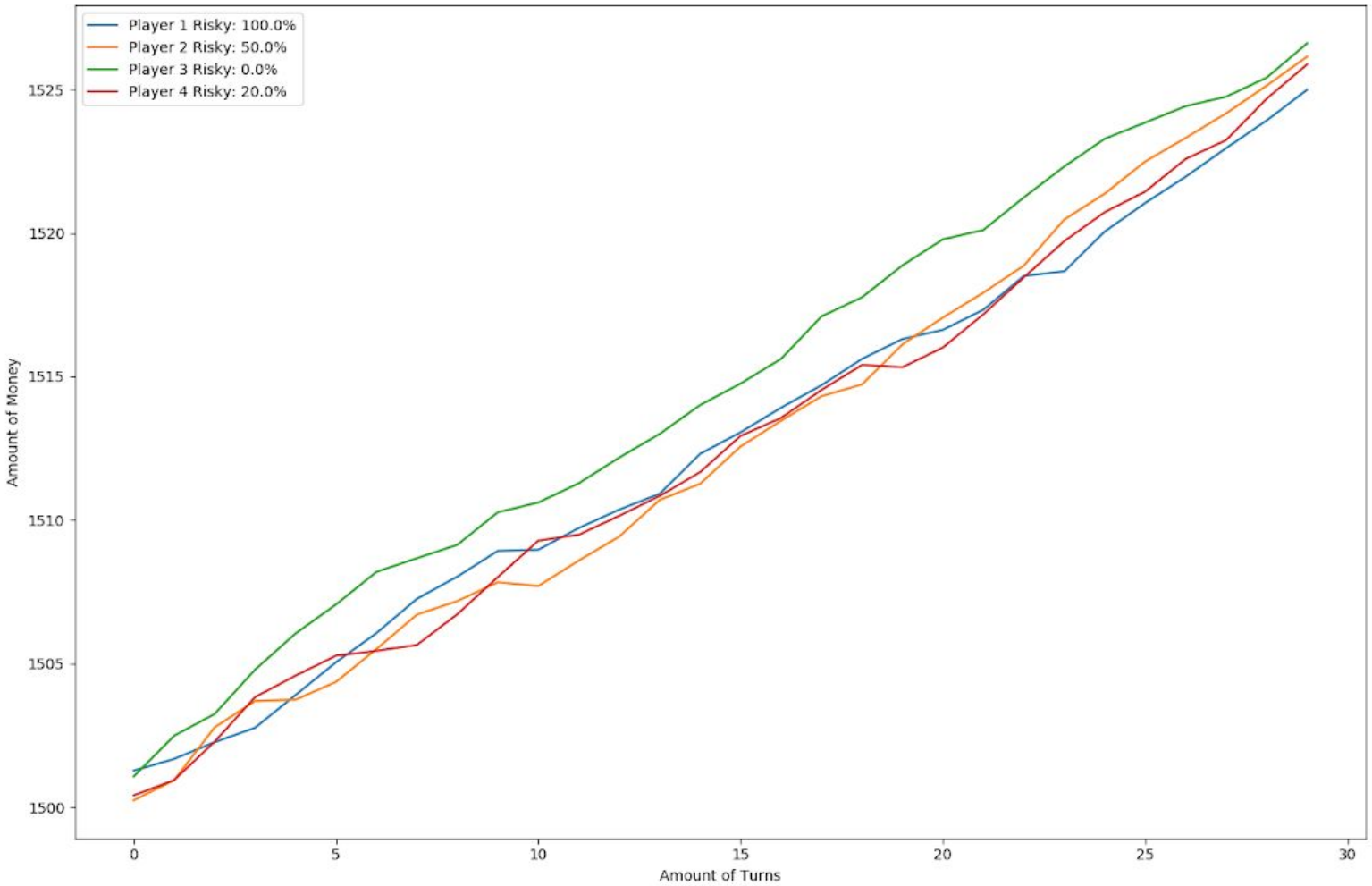

Figure 7: Change in Amount of Money Averaged Over 200,000 Games with 30 Turns/Game and 4 Players with Different Risk Factors

INTERESTING CHALLENGES

1. WHY DO MY PERCENTAGES SUM UP TO 200%?!

In my `compare_hits_time` function I run the simple simulation twice. Once with jail inactive and once again with jail active. To make sure my simulation was working correctly I summed up all the percentages from the first run and all the percentages from the second run. The sum from the first run (with jail inactive) was 100% which is perfect. But from the second run the percentage sum would be anywhere from 204% to 207%. After using print statements to better understand the loops in my simulation the problem became clear. When I first created a list of properties at the beginning of my simulation, each property had an attribute self.hits which would record how many times it had been landed on. When I ran the simulation the second time, I had not cleared the amount of hits so it continuously adding on top of the hits from the first simulation. This lines that fixed this problem were:

```
for place in properties_list:
    place.hits = 0
```

2. WHY IS PLAYER 2 WINNING 97% OF ALL GAMES?!

When writing my `compare_win` and `compare_win_risky` functions, the graphs that I got originally resulted in one player winning a majority of the the games. The main thing I have learned about programming is the "DOES THIS MAKE SENSE?" test. Now, it makes no sense that one player should win anywhere from 97%-99% of 100,000 games. So I went back to debugging using print statements and carefully analyzing 1-2 games at a time with around 15 turns per game. Again, the problem was related to running the complicated simulation multiple times. I was not clearing the resetting of money or locations owned by each player. Essentially once a player had bought a location they had bought it for the rest of the 999, 999 other games. This meant that one player would form a monopoly from the first game and just demolism the rest of the players in subsequent games. The lines that fixed this problem were.

```
for player in players:
    player.money = 1500
    player.monies = []
    player.locations = []
    player.total = 0
```

CONCLUSION

The main conclusion to draw is that there is a difference between the percentage of *turns spent* vs the percentage *chance of hitting* any one spot on the Monopoly board. This is because we must into consideration that a player must roll doubles or have spent three turns in jail to leave. Also, you should prioritize buying orange properties over all else. I can also draw the conclusion that buying property is good however you don't need to buy 100% of all the property you land on. Nonetheless until I add the ability to buy houses and hotels it is hard to be confident in my results from the complicated simulation.

FUTURE PLANS

Ways to further this simulation and make it even more realistic:

1. Add the ability for players to buy houses and hotels
2. If a player has negative money, they should have the option to mortgage their properties
3. Instead of using a risk factor, give players a savings factor (for example 20% would mean they are only willing to spend 80% of their money and keep 20% in savings)
4. Compare results of risk factor to new savings factor graphically

REFLECTION

This project was a fun and exciting way to challenge myself. Having slightly over 700 lines of code made debugging difficult but it also gave me a deeper logical understanding of programming loops. As someone planning to pursue computer engineering, this project gave me valuable input about whether or not I would enjoy this path.

Spoiler Alert: I think I would really enjoy working with computers as a part of my future job.