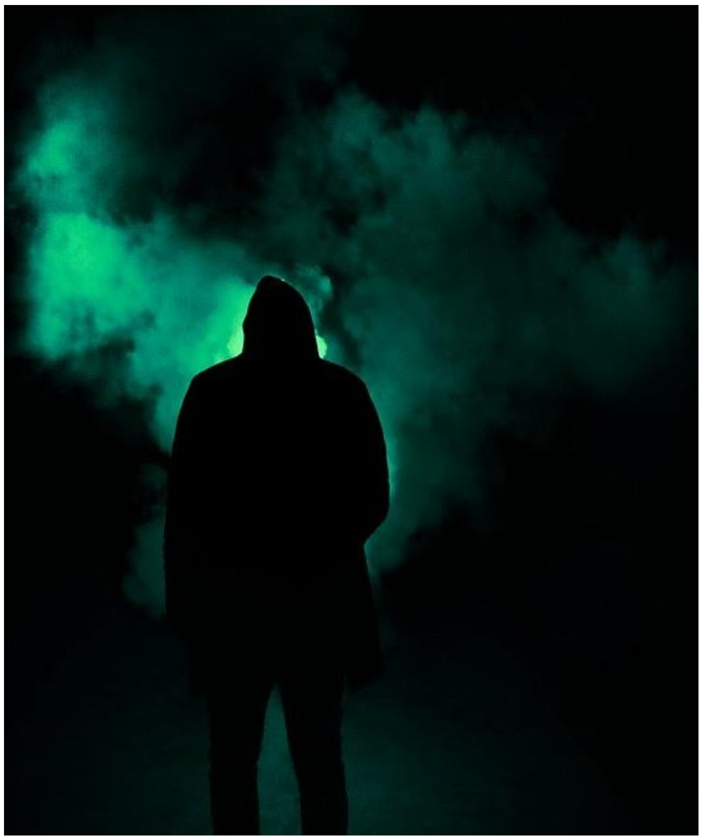


CRYMLAND

Written by Patrick Keener

HONOR STATEMENT:
I have not given or received any unauthorized assistance on this assignment.

Assignment:
DSC 430 Python Programming, Week 10 Assignment 2

YouTube Video Link:

https://youtu.be/nd2D1IPLTgA

Contents

Structure Chart

The structure chart and description of the functions of the Crymland simulation.

Chart

Figure 1 Structure Chart of Crymland()

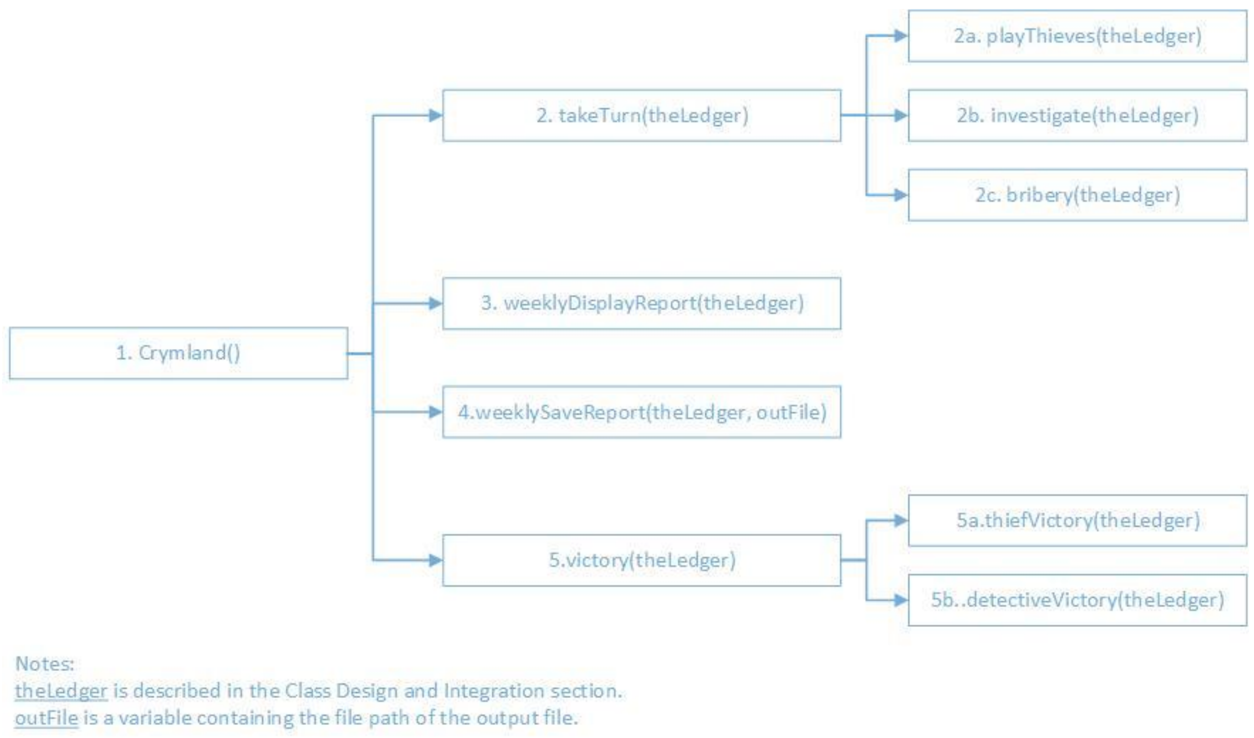

1. Crymland()
2. takeTurn(theLedger)
2a. playThieves(theLedger)
2b. investigate(theLedger)
2c. bribery(theLedger)
3. weeklyDisplayReport(theLedger)
4.weeklySaveReport(theLedger, outFile)
5.victory(theLedger)
5a.thiefVictory(theLedger)
5b..detectiveVictory(theLedger)
Notes:
theLedger is described in the Class Design and Integration section.
outFile is a variable containing the file path of the output file.

Figure 1 describes the structure of the functions used in the Crymland() simulation.

Function Descriptions:

1. Crymland() – This is the executable function that controls the overall program
2. takeTurn() – This function advances the simulation by a turn by first calling on the advanceWeek() method from theLedger and then executing its three sub-functions, in order:
    a. playThieves() – This function executes the actions belonging to the thief. First it conducts heists, then checks for promotion, and then finally promotes eligible thieves at the end of the turn[1].
    b. investigation() – This function executes the actions of the detectives. It first determines the number of investigations (the minimum of the number of thieves or detectives), then randomly assigns detectives to heists. It will then use the investigate() method to determine the outcome of the investigation (accounting for bribery and detective experience)
    c. bribery() – The bribery() function checks each detective to determine whether they are already bribed. If so, an investigation is conducted to determine whether they are caught by internal affairs using the IAinvestigation() method. If not, a check is

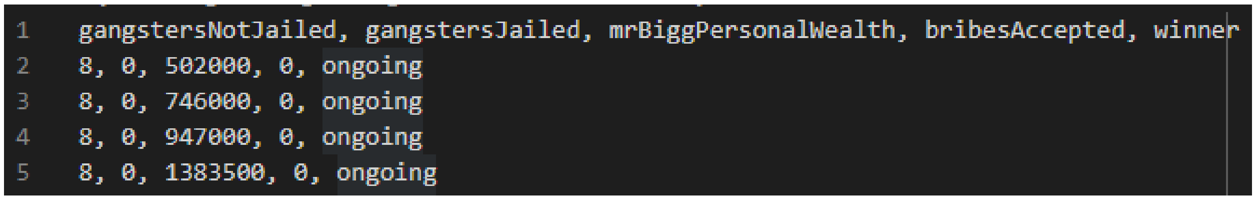
[1] Promotion is done separately from checking for promotion to avoid a situation where thieves are continually being added to the same list that is being checked which, while it will not create an infinite loop, may result in many unnecessary calculations, especially in the end-game

conducted to determine whether Mr. Bigg will attempt to bribe them and if so, a bribe attempt is made using the bribe() method.

3. weeklyDisplayReport() – When the displayWeeklyReport variable is set to True this function will display various stats from throughout the week such as the total number of thieves and lieutenants, lieutenants promoted, loot stolen, bribed detectives, traitors uncovered, Mr. Bigg's total witness count, Mr. Bigg's cut this week, and Mr. Bigg's total stash size.

4. weeklySaveReport – Each week certain stats will be saved to a file for later analysis. The stats are: number of gangsters not in jail (active thieves + active lieutenants + Mr. Bigg), the number of gangsters in jail (jailed thieves + jailed lieutenants), Mr. Bigg's personal wealth, the total $ value of bribes accepted, and who the winner of the simulation is (it is set to 'ongoing' until it's determined).

Figure 2 Example of the Output File

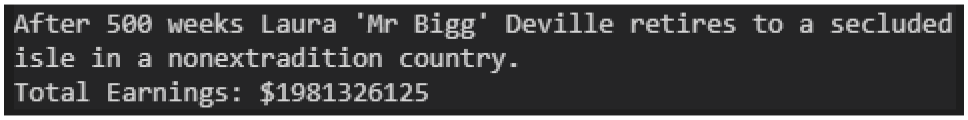


```
gangstersNotJailed, gangstersJailed, mrBiggPersonalWealth, bribesAccepted, winner
8, 0, 502000, 0, ongoing
8, 0, 746000, 0, ongoing
8, 0, 947000, 0, ongoing
8, 0, 1383500, 0, ongoing
```

5. victory() – This final function looks at whomever has won and displays a message on the screen about the outcome. This is done by examining the object "theLedger", where victory status is recorded. It then selects the appropriate message based on the victor.

    a. thiefVictory() – This is the outcome if The Don (Mr. Bigg) is not caught by the end of the simulation.

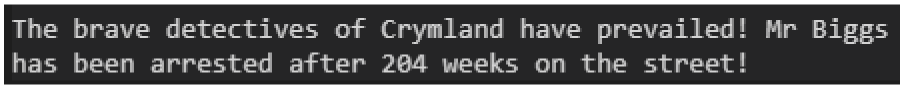
Figure 3 Mr. Bigg Victory



```
After 500 weeks Laura 'Mr Bigg' Deville retires to a secluded
isle in a nonextradition country.
Total Earnings: $1981326125
```

    b. detectiveVictory() – This is the victory if the detectives catch Mr. Bigg.

Figure 4 Detective Victory



```
The brave detectives of Crymland have prevailed! Mr Biggs
has been arrested after 204 weeks on the street!
```

Assumptions

There are many assumptions made from a 'rules of the game' perspective. For example, we assume a constant growth rate of thieves, that each thief is promoted after they reach a certain amount of gold (there is no favoritism and the threshold is constant), and that each newly promoted lieutenant recruits exactly 7 additional thieves.

It is also assumed that every thief becomes a witness, and only 3 witnesses are required to bring down their boss. We also assume that when a boss is taken down, their corresponding thieves don't defect or retire but instead just report to the next higher-level Lieutenant (which may allow a Lieutenant to have more than 7 thieves associated- we assume that this does not degrade their performance).

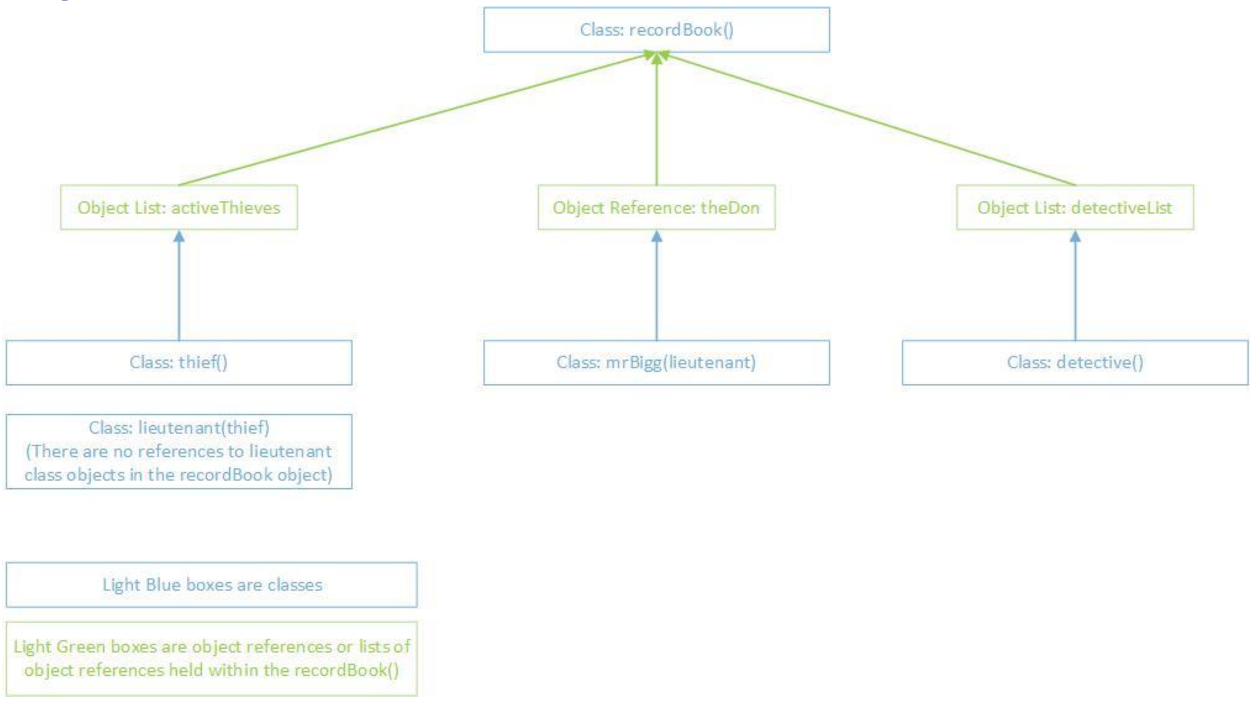
There exists an assumption that each detective's bribe likelihood is dictated by the constants defined (10k, 100k, 1m) and those are the same across all detectives, as well as an assumption of the likelihood that a bribed detective will be investigated at a certain rate and likelihood of being caught.

From a logic perspective, the assumption that arrested thief, lieutenant, and detective objects will not be needed again and thus they are safe to delete.

Class Design and Integration

The classes are designed to be interrelated. The thief class is the superclass for its line, with the lieutenant class as its subclass (inheritance). The lieutenant class has its own subclass which is the mrBigg subclass (inheritance). The detective class has no super or sub classes. Finally, the recordBook class contains references to the thief, mrBigg, and lieutenant classes (composition). All of the classes will also modify various tracking attributes of the recordBook class.

Integration of Classes


Class: recordBook()
Object List: activeThieves
Object Reference: theDon
Object List: detectiveList
Class: thief()
Class: mrBigg(lieutenant)
Class: detective()
Class: lieutenant(thief) (There are no references to lieutenant class objects in the recordBook object)
Light Blue boxes are classes
Light Green boxes are object references or lists of object references held within the recordBook()

Class: recordBook()

The recordBook() class is stored as a variable, which spans all functions and classes, and serves two purposes:

1. The class holds all of the settings imported from the input file through the variables (attributes) held in it. These attributes are used throughout the program.
2. To hold various lists and trackers internal to the program that facilitate communication between functions and allow for automatic updating of various object attributes throughout the program.

The primary manner that the other classes interact with the recordBook() is through the lists of active objects that correspond to the thieves and detectives that haven't been arrested. A function cycles through these two lists and activates the functions that drive the simulation each turn.

This class also resets various lists and trackers at the start of each week through the advanceWeek() method.

Class: thief()

The thief() class is the superclass over the lieutenant() and mrBigg() classes. This class registers the recordBook() class which it uses to add itself to the activeThieves list during initialization, as well is import the various settings required for proper execution from the input file. Each thief can conduct heists, add to its bank account, check itself for whether it's time to be promoted, and when a thief is promoted the method is within the thief class.

It should be noted that the thief() and lieutenant() classes each have an attribute called the "boss", which references the object that created it (ie a lieutenant) which allows it to pass up its earnings as well as register which specific lieutenant it is testifying against.

Class: lieutenant(thief)

As a subclass of the thief() class, the lieutenant() class inherits the method to add to its bank account. Upon initialization, the class will create 7 thieves that will register themselves with the recordBook().

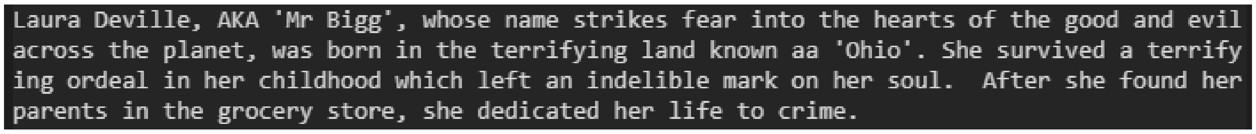
It can also check for whether it is arrested, and if so it removes itself from various lists in the recordbook(). It also contains the "boss" attribute that allows them to pass up portions of their earnings as well as testify against specific higher-level bosses. Ultimately, this mechanism will allow 3 lieutenants to testify against Mr. Bigg.

The lieutenant() class is also able to promote thieves. This is done in a class outside of the thief() class because promotions happens by creating a new lieutenant() object and then deleting the original thief object. It should be noted that a portion of this process involves calling methods on the thief to remove itself from all lists before the object is set for garbage collection.

Class: mrBigg(lieutenant)

The mrBigg class is a subclass of the lieutenant class. It does not have a boss (since this is the highest boss). It can check whether it is arrested, reset its own earnings, add to its bank account, and interact with the detective class through bribery methods. It also has a backstory that is told at the start of the simulation, with a customizable name and pronouns.

Figure 5 Mr. Bigg's backstory which is displayed at the start of each simulation



```
Laura Deville, AKA 'Mr Bigg', whose name strikes fear into the hearts of the good and evil
across the planet, was born in the terrifying land known aa 'Ohio'. She survived a terrify
ing ordeal in her childhood which left an indelible mark on her soul.  After she found her
parents in the grocery store, she dedicated her life to crime.
```

Class: detective()

The detective class is neither a sub- nor super-class. The detective has methods to determine when "it's on the take" (has been bribed), whether their corruption has been discovered, the investigation process, and the method to investigate heists.

It should be noted that the detective, when bribed, has its probability of solving a crime set to 0.

Results Analysis

Outcome Analysis

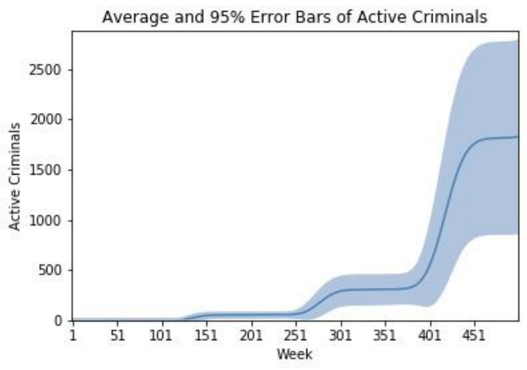
The simulation naturally lends itself to a Monte Carlo simulation, and as such it was run 5,000 times using a scenario selected specifically because it prevented either side from completely dominating the results.

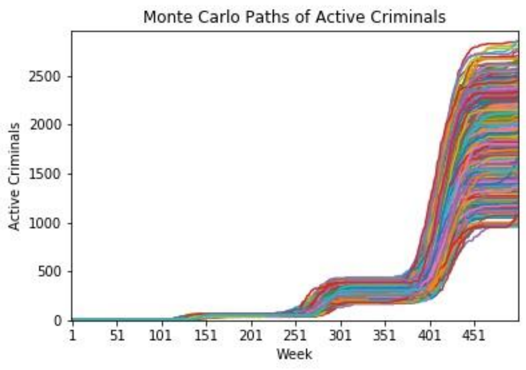
Two graphs are seen below. The first graph has the average result in dark blue along with a region representing the 95% confidence interval. The second graph plots all 5,000 paths taken in the scenario.

The scenario had three key differences from the default scenario:

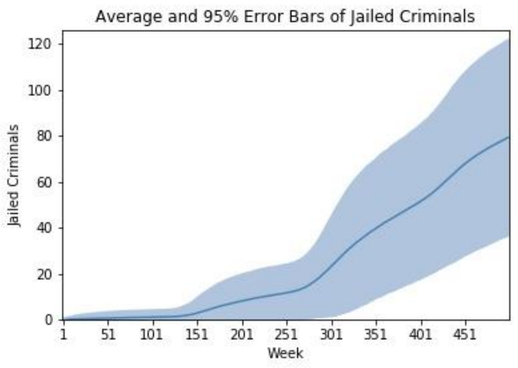
1. The required loot to be promoted to lieutenant was set to $10m
2. The number of detectives was set to 100
3. The detective solve probability was set to 1.5%

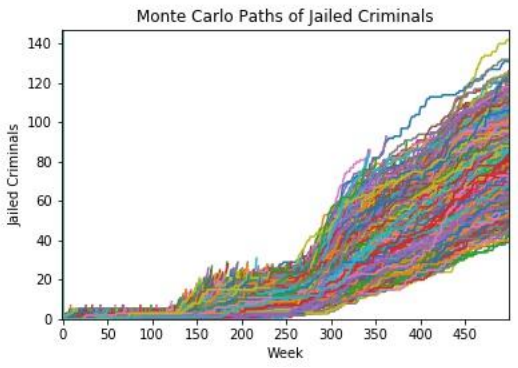
The primary advantage is that it inhibited growth of the thief population and prevented runaway growth within the 500-period time horizon.



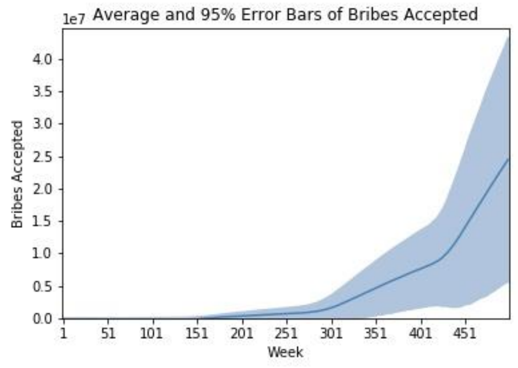


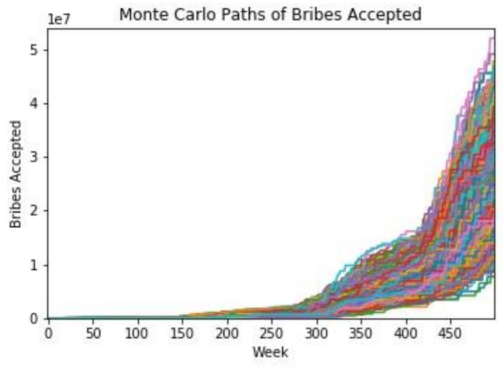
The first chart is for active criminals. It is interesting how there seem to be jumps in the number of thieves, then stabilization, then jumps, then stabilization, and that this was fairly consistent across all visualizations. It was also interesting that the final jump (between week 400 and 450) led to such dramatic variance- between 500 and 2500 thieves were created in this period.



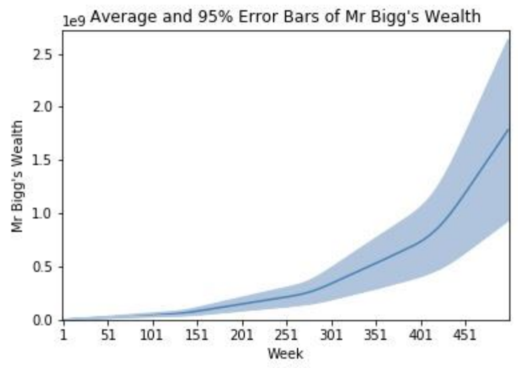


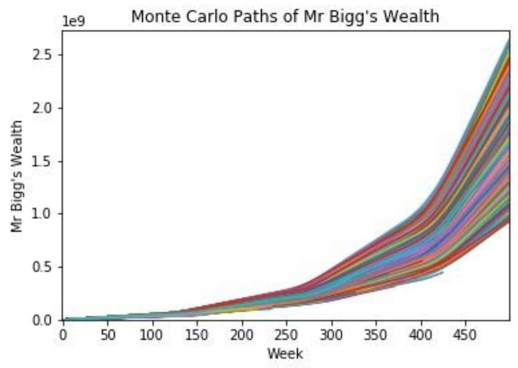
The next chart is for jailed criminals. This also has inflection points; however, it becomes much more stable after week 250- likely because the full number of detectives (100) are employed around week 250.





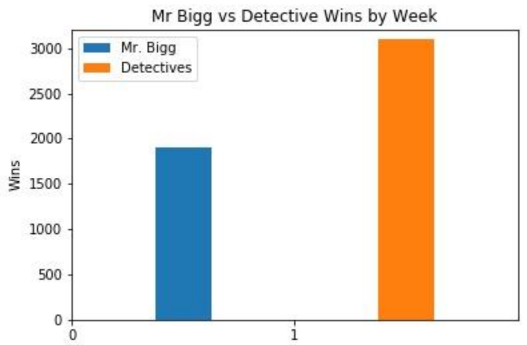
The bribes accepted graph follows the same pattern as active criminals of having clear inflection points. These points signify an exponential rise, and make sense in the context of an increasing number of thieves (and therefore employed detectives- remember that there are 100 active detectives and not all can be employed at once until there are at least 100 thieves).

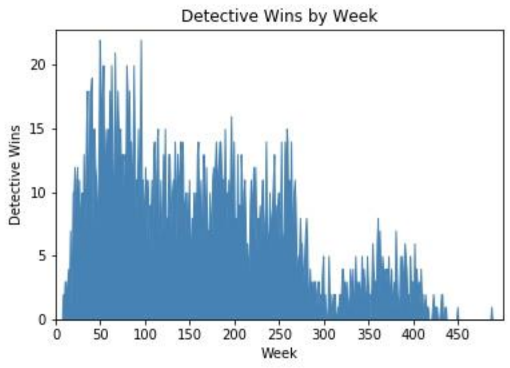




Finally, Mr. Bigg's wealth grows exponentially but mirrors the number of thieves pretty closely. This is expected as they are the direct drivers of his wealth.

Arrest and Wealth Accumulation Analysis





Wealth accumulation over multiple (5,000) scenarios is described in the previous section as the final piece. The key point is that it mirrors the number of thieves, however it should also be noted that in 500 weeks Mr. Bigg accumulates around $2.5billion. It might be worth looking into a life of crime.

Mr. Bigg wins approximately 38% of the time, while the detectives win the other 62%. Looking at the wins by week, we see an inverse pattern to the number of thieves, which makes sense. Past week 450 there is only one or two detective wins. It should be noted that around week 300 there is a large dip that actually recovers significantly between week 325 and 425. This is still conducted over 5,000 scenarios, so these results are of interest. Further investigation would be needed to explain it- potentially adding more simulations because the data becomes sparse at this point.

Scenario Design and Analysis

[Scenarios in which detectives almost always arrest mr bigg and one where mr bigg almost always goes free]

Crymland Extension

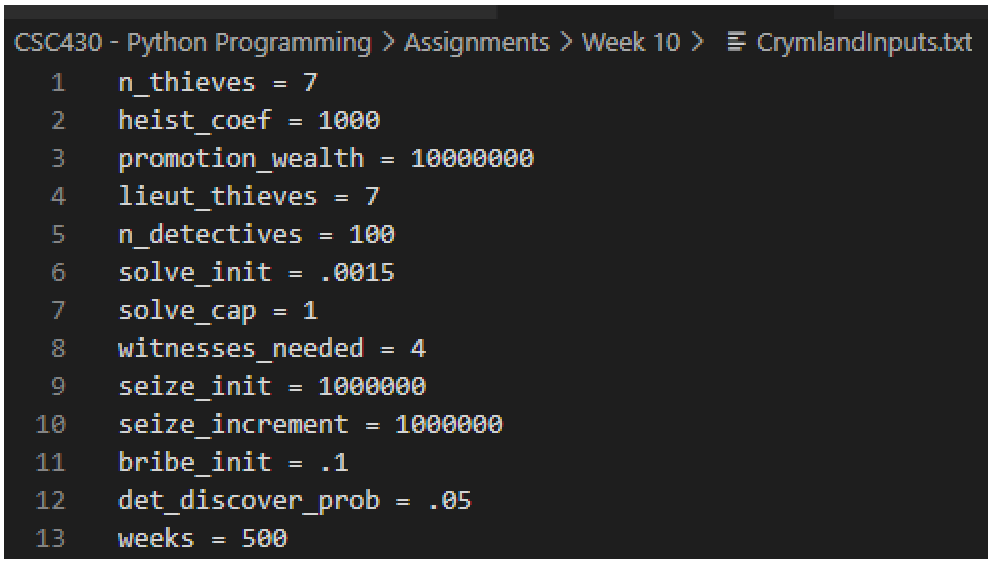
- Allow detective #s to grow with the thieves
- Allow users to manipulate the settings
- Detectives only investigate thieves that steal a certain amount of loot.

Appendix A – Inputs

[Placed in an input file. By default, the input file's name should be 'Crymlandinputs.txt']

```
n_thieves = 7
heist_coef = 1000
promotion_wealth = 10000000
lieut_thieves = 7
n_detectives = 100
solve_init = .0015
solve_cap = 1
witnesses_needed = 4
```

seize_init = 1000000
seize_increment = 1000000
bribe_init = .1
det_discover_prob = .05
weeks = 500


CSC430 - Python Programming > Assignments > Week 10 > CrymlandInputs.txt

```
n_thieves = 7
heist_coef = 1000
promotion_wealth = 10000000
lieut_thieves = 7
n_detectives = 100
solve_init = .0015
solve_cap = 1
witnesses_needed = 4
seize_init = 1000000
seize_increment = 1000000
bribe_init = .1
det_discover_prob = .05
weeks = 500
```

Appendix B – Crymland Analysis Code

```
# Written by Patrick Keener on 11/26/2020
# Video Link:  https://youtu.be/nd2D1IPLTgA
# Honor Statement:  "I have not given or received any unauthorized assistence
#                     on this assignment"
#
# DSC 430: Python Programming
# Assignment 1002: Crymland Analysis

import os.path
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import random


# def rename_cols():
    # for file in os.listdir('MC_Run_1'):
```

```
    #     filePath = os.path.join('MC_Run_1', file)
    #     df = pd.read_csv(filePath)

#         newCols = {'gangstersNotJailed': 'gangstersNotJailed',
#         ' gangstersJailed': 'gangstersJailed',
#         ' mrBiggPersonalWealth': 'mrBiggPersonalWealth',
#         ' bribesAccepted': 'bribesAccepted',
#         ' winner':'winner'}

#         df = df.rename(newCols, axis = 'columns')

#         df.to_csv(filePath)

#     print('Done')

# rename_cols()



def saveAggregates(filePath, df):
    """ saves files to filePath as CSV"""
    df.to_csv(filePath, index = False)

def aggregateResults(inputFile, outputFile):
    """ Aggregate the results into files containing only like-outputs so analysis
     may be conducted
     """

    df_notJailed = list()
    df_Jailed = list()
    df_biggWealth = list()
    df_bribes = list()
    df_winner = list()
    df_working = list()

    for file in os.listdir(inputFile):
        filePath = os.path.join(inputFile, file)
        df_working = pd.read_csv(filePath)

        # Break file into list of lists that will eventually be converted
        # into a dataframe.  Lists used first so new objects aren't created
        # from every merge.
        df_notJailed.append(df_working['gangstersNotJailed'].tolist())
        df_Jailed.append(df_working['gangstersJailed'].tolist())
        df_biggWealth.append(df_working['mrBiggPersonalWealth'].tolist())
        df_bribes.append(df_working['bribesAccepted'].tolist())
        df_winner.append(df_working['winner'].tolist())



    # Convert to data frames after aggregation and save to file
    filePath = os.path.join(outputFile, 'notJailed.csv')
    saveAggregates(filePath, pd.DataFrame(df_notJailed))

    filePath = os.path.join(outputFile, 'Jailed.csv')
    saveAggregates(filePath, pd.DataFrame(df_Jailed))

    filePath = os.path.join(outputFile, 'biggWealth.csv')
    saveAggregates(filePath, pd.DataFrame(df_biggWealth))

    filePath = os.path.join(outputFile, 'bribes.csv')
    saveAggregates(filePath, pd.DataFrame(df_bribes))

    filePath = os.path.join(outputFile, 'winner.csv')
    saveAggregates(filePath, pd.DataFrame(df_winner))

    return


def singleLine(subject, mean, stdev, inputFile):
    """ creates a plot with a single line and error bars """
    CI = 2.78 # Corresponds to 95% confidence interval

    # plot initial lines
    plt.plot(mean, color = 'steelblue')
    plt.plot(mean + stdev * CI, color = 'lightsteelblue')
    plt.plot(mean - stdev * CI, color = 'lightsteelblue')

    # add error shading
    plt.fill_between(range(0,500),mean + stdev * CI, mean, color = 'lightsteelblu
e')
    plt.fill_between(range(0,500),mean - stdev * CI, mean, color = 'lightsteelblu
e')

    # find max x value ignoring NaN
    maxX = np.nanmax(mean + stdev * CI)*1.0375

    # adjust axis to better fit the chart
    plt.axis([0,500, 0, maxX])
    plt.xticks(np.arange(1, 500, 50))
```

```python
    # add axes lables
    plt.title('Average and 95% Error Bars of ' + subject)
    plt.ylabel(subject)
    plt.xlabel('Week')

    # save chart to file
    outfile = os.path.join(inputFile, subject.strip() + '_Avg_95.jpeg')
    plt.savefig(outfile)

    # show & flush the chart
    plt.show()


def multipleLines(df, subject, inputFile):
    """ creates a plot with multiple lines """

    # Cycle through each simulation and add line to chart
    for sim in range(len(df)):
        plt.plot(df.iloc[sim])

    # find max x value ignoring NaN
    maxX = np.nanmax(df)*1.0375

    # adjust axis to better fit the chart
    plt.axis([0,500, 0, maxX])
    plt.xticks(np.arange(1, 500, 50))

    # add axes lables
    plt.title('Monte Carlo Paths of ' + subject)
    plt.ylabel(subject)
    plt.xlabel('Week')

    # save chart to file
    outfile = os.path.join(inputFile, subject.strip() + '_MC.jpeg')
    plt.savefig(outfile)

    # show & flush the chart
    plt.show()


def getStats(df):
    """ Calculate some summary statistics """

    mean = df.mean(axis=0, skipna=True)
    stdev = df.std(axis=0, skipna=True)
    return mean, stdev


def visualize(inputFile, fileName, subject):
    """ visualizes the data """

    df = openDF(inputFile, fileName)

    multipleLines(df, subject, inputFile)
    mean, stdev = getStats(df)
    singleLine(subject, mean, stdev, inputFile)


def winChart(detWinVector, inputFile):
    """ Creates a chart of detective wins over time """
    x = np.arange(0, 500, 1)
    plt.fill_between(x,detWinVector,color = 'steelblue')

    # find max x value ignoring NaN
    maxX = np.nanmax(detWinVector)*1.0375

    # adjust axis to better fit the chart
    plt.axis([0,500, 0, maxX])
    plt.xticks(np.arange(0, 500, 50))

    # add axes lables
    plt.title('Detective Wins by Week')
    plt.ylabel('Detective Wins')
    plt.xlabel('Week')

    # save chart to file
    outfile = os.path.join(inputFile, 'DetectiveWins.jpeg')
    plt.savefig(outfile)

    # show & flush the chart
    plt.show()


def winComparison(biggWins, detWins, inputFile):
    """ Creates a bar chart comparing absolute number of mr bigg and detective wi
ns """

    winList = [biggWins, detWins]
    x = np.arange(1,3)
    width = 0.25

    plt.bar(.5, biggWins, width, label='Mr. Bigg')
    plt.bar(1.5, detWins, width, label='Detectives')

    # find max x value ignoring NaN
    maxX = max(biggWins, detWins)*1.0375

    # adjust axis to better fit the chart
    plt.axis([0,2, 0, maxX])
    plt.xticks(np.arange(0, 2, 1))

    # add axes lables
    plt.title('Mr Bigg vs Detective Wins by Week')
    plt.ylabel('Wins')

    # add legend
    plt.legend()

    # save chart to file
    outfile = os.path.join(inputFile, 'winComparison.jpeg')
    plt.savefig(outfile)

    # show & flush the chart
    plt.show()



def getWinStats(df):
    """ get win stats """

    detWins = abs(int(np.nansum(df[df == 1])))
    biggWins = abs(int(np.nansum(df[df == -1])))

    p_Det = detWins/(detWins+biggWins)
    p_MrBigg = 1-p_Det

    detWinVector = df.sum()
    detWinVector[-1] = biggWins + detWinVector[-1] # adjust for Mr Bigg wins

    return round(p_MrBigg, 4), biggWins, round(p_Det, 4), detWins, detWinVector


def visualizeWins(inputFile, fileName):
    """ This functions visualizes the winner """

    df = openDF(inputFile, fileName)

    # replace outcomes with digits for counting
    df.replace({' detectives': 1, ' mrBigg': -1, ' ongoing': 0}, inplace = True)

    p_MrBigg, biggWins, p_Det, detWins, detWinVector = getWinStats(df)

    winChart(detWinVector, inputFile) # Generate time series chart of wins
    winComparison(biggWins, detWins, inputFile) # Generate comparison of wins

# Generates a tuple of random numbers between 0 and 1 to be used for random color
s
# for the graphs
def genRandomColor(): return (random.random(), random.random(), random.random())


def openDF(inputFile, fileName):
    """ opens file as  a csv """

    filePath = os.path.join(inputFile, fileName)
    df = pd.read_csv(filePath)
    return df

def visualizeResults(outputFile):
    """ Visualization for the results """

    visualize(outputFile, 'notJailed.csv', 'Active Criminals')
    visualize(outputFile, 'Jailed.csv', 'Jailed Criminals')
    visualize(outputFile, 'biggWealth.csv', "Mr Bigg's Wealth")
    visualize(outputFile, 'bribes.csv', 'Bribes Accepted')
    visualizeWins(outputFile, 'winner.csv')


def analysis():
    inputFile = 'MC_Run_1\\'
    # inputFile = 'outputs\\'
    outputFile = 'aggregatedDS\\'

    # aggregateResults(inputFile, outputFile)

    visualizeResults(outputFile)
    print('Fin.')



analysis()
```