Crymland Simulation Analysis

Gerardo Palacios

11/26/2020

DSC 430: Python Programming

DePaul University

YouTube Link:
https://youtu.be/VMPusza_YZc

Student honor statement
I have not given or received any unauthorized assistance on this assignment.

Contents

Crymland Simulation Analysis

The Crymland Simulation was a challenging exercise involving all the skills and techniques learned throughout the quarter into a single project. The following report contains six different sections that help describe the simulation's architecture and visualize the results.

Top-Down Structure Chart

A few assumptions were made during the design of the simulation.

1) If a Lieutenant is arrested, the remaining team members are no longer in play.
2) After a successful bribe, the Detective continues to work for Mr. Bigg without additional pay.

The appendix contains the top-down structure chart for the Crymland Analysis. It displays the necessary components for the simulation to run. Specifically, displaying class objects and their accompanying methods. As one can see from the chart, the simulation is mainly composed of two classes, Simulated Data and Crymland. The Simulated Data class acts as a data handler that retains all the weekly information created by the Crymland class. The Crymland class uses composition by defining its attributes as two other classes, Academy and Syndicate. Which, in turn, encapsulate the warring factions and their foot soldiers.

Describing Classes and Integration

Simulated Data.
The simulated data class acts as the data handler for the simulation analysis. It is composed of eight methods and nine attributes that help manage and view the data it encapsulates. All the data is maintained within the class's attributes under a panda's data frame. This method was chosen to facilitate computations and multi-column data.

Crymland.
The Crymland class acts as the simulated city of Crymland. It uses composition when it is first initialized. Specifically, it is used in line #632 - #633. Here, the Syndicate and Academy classes are defined and initiated as attributes. Crymland is composed of two methods and two attributes. This class acts as a container for the simulation workflow and records the weekly results to the Simulated Data class.

Academy
The Academy is composed of four methods and two attributes. The Academy class acts as the container for the dedicated detectives. This class helps manage, assign, and maintain the detectives and their associated data using composition (Line #471) and defining the Detective class as part of its initializing attributes.

Detectives.
The Detective class acts as the Academy's foot soldiers meant to investigate potential heists. It is composed of five methods and six attributes. This class's attributes and methods help manage the data for the experience, risk, recovered, and any bribed assets of the associated Detective. This architecture helps organize all the data in the proper branches since multiple detectives work under the Academy.

Syndicate.

The Syndicate class encapsulates the entire simulated criminal enterprise for Mr. Biggs. This class is composed of three methods and three attributes. It acts as the root branch for the criminal enterprise and maintains all actors and their associated data organized within each other. The Syndicate class also uses composition (Line # 427) when first initialized by defining Mr. Bigg as a Lieutenant. The Lieutenant class, in turn, initializes seven thieves. Thus, creating the first branch of the syndicate.

Thief and Lieutenant.
The Thief and Lieutenant classes act as the ranking actors working under the syndicate. The Thief class comprises three methods and two attributes, and the Lieutenant class extends it by four additional methods and overwriting two. The Lieutenant class inherits the attributes and methods from the Thief class. This allows the Thief class to be extended and provide additional methods (Line # 228 and #297). In this case, it is extended into the Lieutenant class. The Lieutenant class would still have to collect profits and provide some to their (if any) ranking officer; since the Thief class already does something similar, the Lieutenant class overwrites the method when it is extended. Additionally, the Lieutenant class uses composition to define multiple thieves that report up (Line # 322). This way, The Lieutenant class, and Thief class are inextricably linked throughout each potential branch and maintains data organized within each initialized object

Six, Ten, and Twenty-Sided Die.
The six-sided die is the parent class that simulates classic rolling dice. It is composed of two attributes and two methods. It leverages the Python random library to create a pseudorandom number. The ten and twenty-sided die are composed of the same attributes and methods, except for the defined values in the class's attributes. This is also an example of inheritance since the ten and twenty-sided die extends the six-sided die but increases the number of sides available. These classes were used to calculate the random probabilities for different simulation sections, such as increasing the discovery rate for a corrupted detective or increasing the Detective's experience after a successful investigation.

Integration.
Either inheritance or composition integrated all the classes. This architecture allowed for the sharing of data between classes since all classes had access to each other throughout the simulation. This also maintained data integrity by continually maintaining the appropriate branches (no matter how infinitely large) together.

Describing Future Extensions

The architecture of the simulation allows for the simulation to be extended in the future. Reasons to extend the simulation could include running a variant of the simulation where the simulation workflow may want to be compared with a previous one. Three ways a developer could extend the simulation can be:

1) Create a new class by inheriting the Crymland class and creating an overwriting workflow in the simulation method.
2) Extend the Lieutenant Class to add a new layer into the criminal hierarchy of Mr. Bigg's syndicate
3) Extend the Simulated Class to add a new data column or new statistic to be tracked within the data handler.

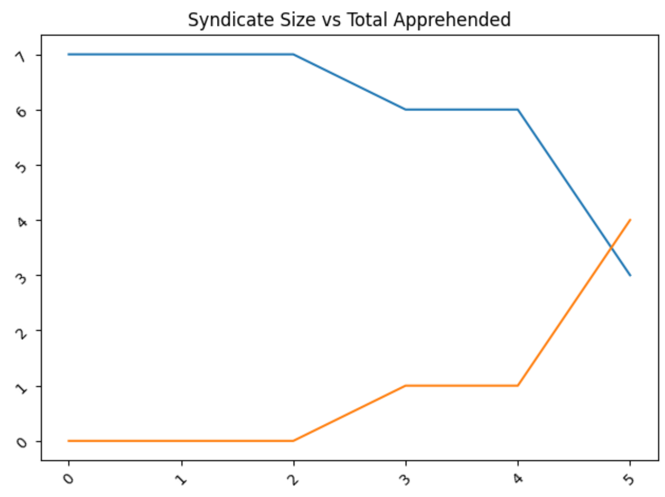
Each of these extensions can tweak the simulation without completely redesigning the central architecture.

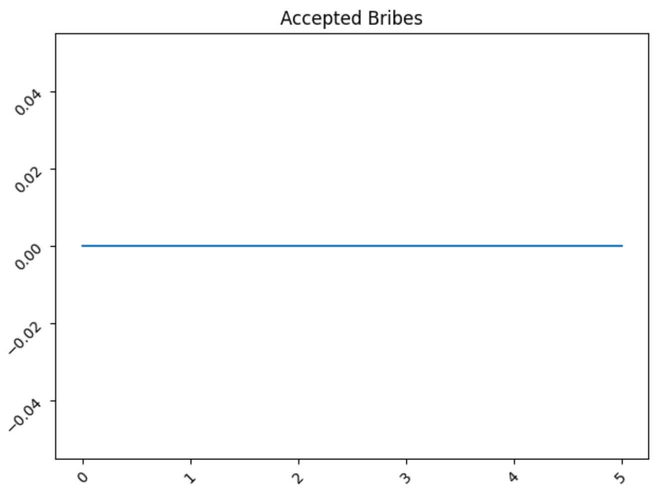
Data Analysis

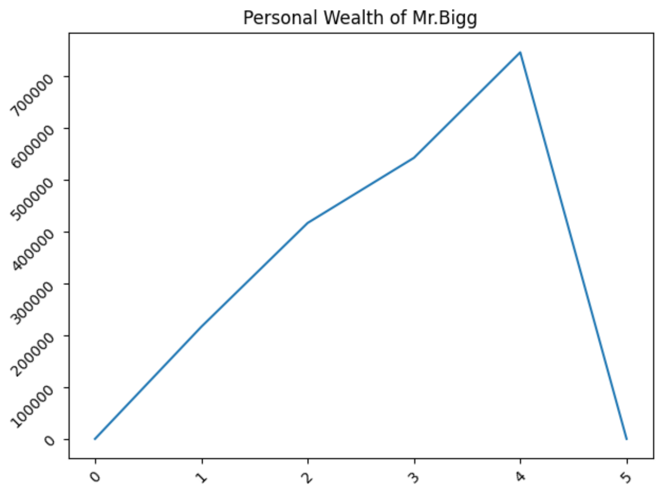
The results (iteration 1), using the simulation's default parameters, are in the plots below. The initial results show that Mr. Bigg accumulated a maximum personal wealth of over $700k until usurped by the Academy and losing everything at the end of five weeks. No bribes were successful, and no thieves were promoted or recruited.

To the right, Syndicate size vs. Total apprehended shows a perfect inverse relationship with each other. This behavior is understandable since the criminal enterprise did not grow. The syndicate would decrease at the same rate as the number of jailed actors increases. The personal wealth of Mr. Bigg was increasing at a seemingly constant rate before his wealth was taken. According to the initial charts, the default parameters favor the Academy as the simulation was over within 5 weeks instead of reaching the end of 500 weeks.

The following section *Simulations* displays the data after running the simulation several times. Overall, the pattern persists that the default parameters are in favor of the Academy. The longest Mr. Bigg was able to maintain control was seven weeks with a range of accumulated wealth between 100k and 1.3M. Mr. Bigg was always apprehended in the end.







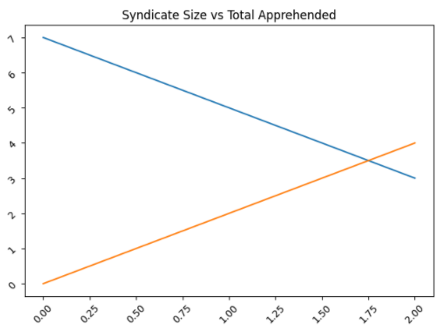
Simulations

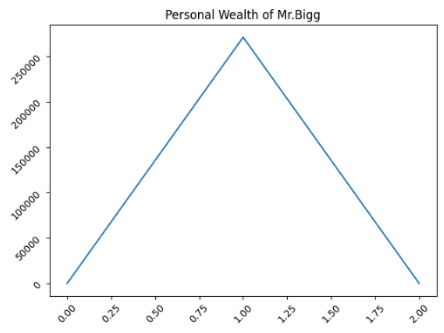
Weeks	Actors	Jailed	Wealth	Bribes
Iteration 1				
0	7	0	0	0
1	7	0	216,000	0
2	7	0	416,000	0
3	6	1	541,500	0
4	6	1	744,750	0
5	3	4	0	0
Iteration 2				
0	7	0	0	0
1	5	2	106,500	0
2	2	5	0	0
Iteration 3				
0	7	0	0	0
1	6	1	227,500	0
2	5	2	566,000	0
3	3	4	0	0
Iteration 4				
0	7	0	0	0
1	6	1	411,500	0
2	5	2	512,250	0
3	5	2	642,250	0
4	3	4	0	0
Iteration 5				
0	7	0	0	0
1	7	0	210,500	0
2	7	0	415,250	0
3	6	1	664,750	0
4	5	2	908,750	0
5	5	2	1,044,000	0
6	5	2	1,344,000	0
7	3	4	0	0
Iteration 6				
0	7	0	0	0
1	7	0	365,000	0
2	7	0	504,750	0
3	7	0	711,500	0
4	6	1	835,500	0
5	6	1	1,052,500	0
6	5	2	1,338,000	0
7	3	4	0	0

Scenario Design

Almost Always Arrested

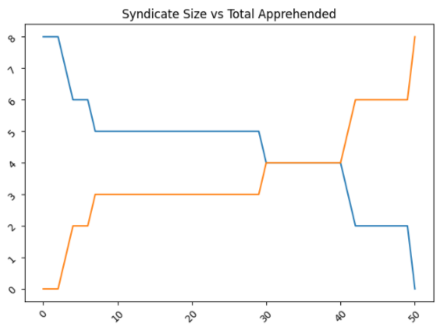
Designing a scenario where Mr. Bigg gets arrested was simple because the default parameters were already in favor of the Academy. In that case, a few things can be changed to further tip to scale towards the Academy. This design increases the number of detectives by two. This would mean more detectives increase the chances to apprehend, interrogate, and capture Mr. Biggs. Other ways they could have been changed to achieve the same result would have been to decrease the total number of jailed thieves that conspire against their supervisor or removing the experience cap on the detectives.

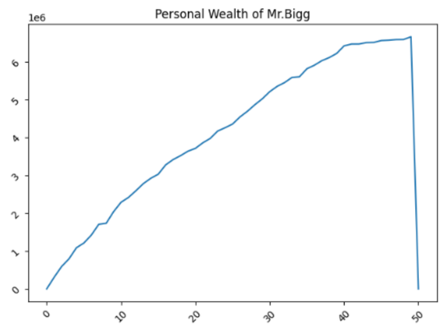


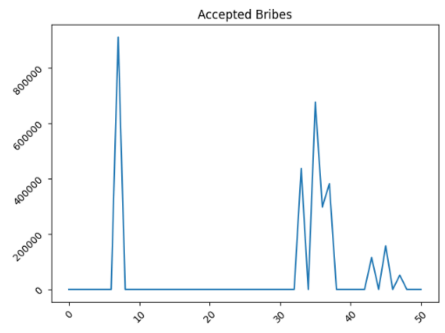


Almost Always Goes Free

Designing this scenario proved to be more difficult because the simulation could go on forever due to the potential of an infinitely increasing syndicate branch. Thus, I change nearly all the parameters in a way that would tilt in favor of the syndicate. In this case, the total number of thieves in jail required to indict their boss decrease the number of detectives, increase the initial bribe rate, and the Detective's initial solve rate and cap.







Appendix

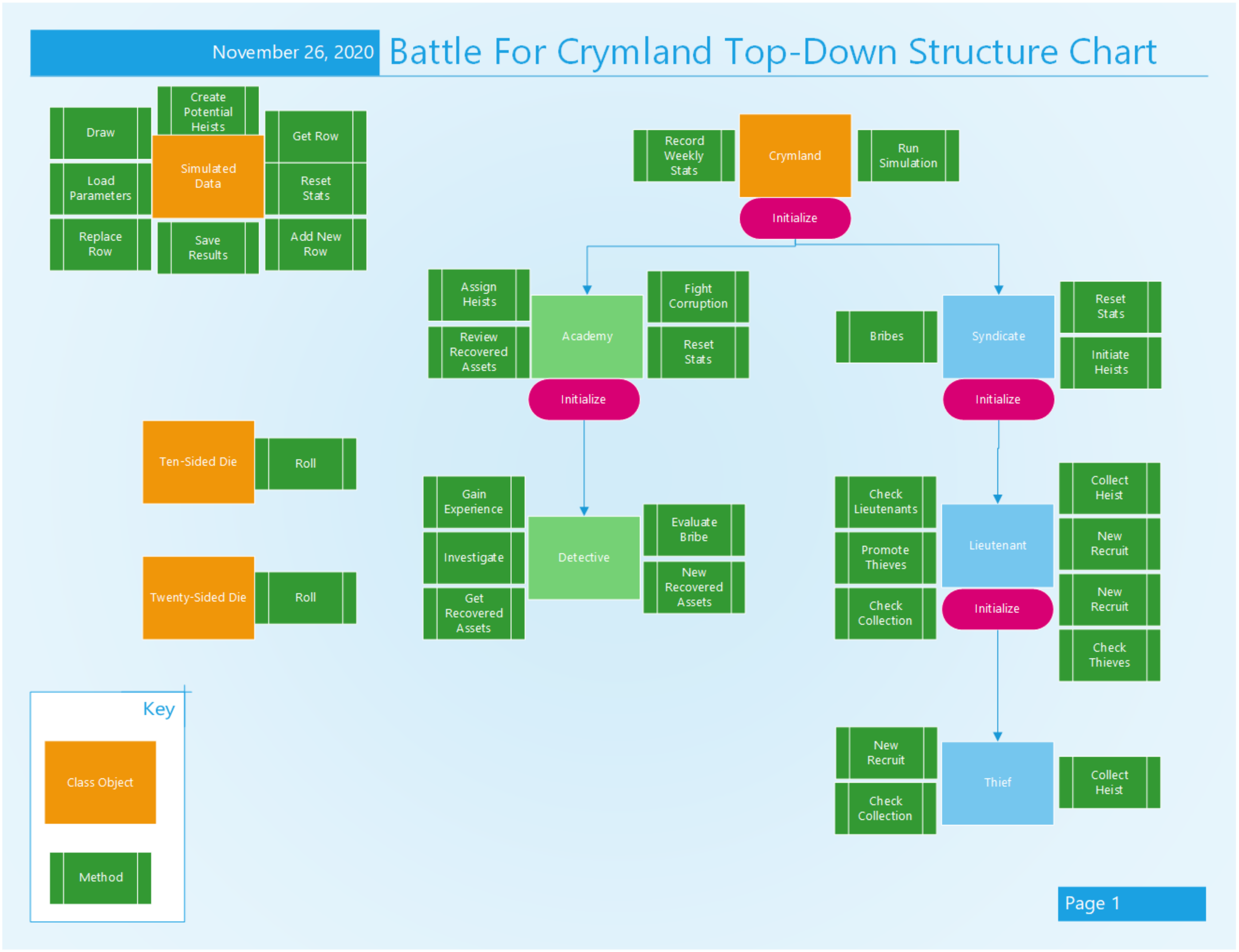
Parameter File Data Scenarios

Weeks	Thieves	Heist	Promotion	Jailed	Detectives	Solve	Cap	First	after	Risk	Bribe
500	7	1000	1000000	3	3	0.25	0.75	1000000	1000000	0.05	0.1
500	7	1000	1000000	3	5	0.25	0.75	1000000	1000000	0.05	0.1
500	7	1000	1000000	7	2	0.10	0.10	1000000	0	0.01	0.8

Simulation Code:

```
# Sample parameters1.csv file to upload data from.

#
weeks,num_thieves,heist_coefficient,promotion_wealth,jailed_thieves,num_detec
tives,solve_init,solve_cap,seizes_first,seizes_thereafter,initial_risk,initia
l_bribe
# 500,7,1000,1000000,3,3,.25,0.75,1000000,1000000,0.05,0.1

# Generate a twenty-sided and ten-sided die
twenty_sided = TwentySidedDie()
ten_sided = TenSidedDie()

scenarios = ['parameters1.csv', 'parameters2.csv', 'parameters3.csv']

for i in range(len(scenarios)):
    # Create Data handler
    simulated_data = SimulatedData()

    # load data parameters
    simulated_data.load_parameters(scenarios[i])

    # Create Crymland
    crymland = Crymland()

    # Run simulation
    crymland.run_sim(simulated_data.parameters['weeks'])

    # Plot results
    simulated_data.plot_time_series('Syndicate Size vs Total Apprehended',
['actors', 'jailed'])
    simulated_data.plot_time_series('Personal Wealth of Mr.Bigg', 'wealth')
    simulated_data.plot_time_series('Accepted Bribes', 'bribes')

    # Save results
    simulated_data.save_results('simulation_results{}.csv'.format(i))
```

Top-Down Structure Chart


November 26, 2020
Battle For Crymland Top-Down Structure Chart
Draw
Create Potential Heists
Get Row
Load Parameters
Simulated Data
Reset Stats
Replace Row
Save Results
Add New Row
Record Weekly Stats
Crymland
Run Simulation
Initialize
Assign Heists
Academy
Fight Corruption
Review Recovered Assets
Reset Stats
Initialize
Bribes
Syndicate
Reset Stats
Initiate Heists
Initialize
Ten-Sided Die
Roll
Gain Experience
Investigate
Get Recovered Assets
Detective
Evaluate Bribe
New Recovered Assets
Check Lieutenants
Promote Thieves
Check Collection
Lieutenant
Initialize
Collect Heist
New Recruit
New Recruit
Check Thieves
Twenty-Sided Die
Roll
Key
Class Object
Method
New Recruit
Check Collection
Thief
Collect Heist
Page 1