Python Project: A Casino implementation

Fabio Giralt

Toulouse School of Economics

January 29, 2017

1 Roulette and Craps Games

The games implementation does not require many explanation as the exact steps given in the slides were followed. The code should be self explanatory.

The function *r(n, seq)* returns a list of length n where each elements are drawn with replacement in *seq*[1]

I use quite extensively the built in function *zip* as it permits to have a sort of vectorized way to deal with lists thus making the code shorter and clearer as almost all of my classes or functions process lists.

Scaling the Craps wins

A Craps game has 11 possible different outcomes with o total of 6 different corresponding probabilities (2 and 12 have a probability of 1/36 each...). To avoid dominant strategies (always betting on one or a subset of possibilities), the amount given by the casino in case of a successful bet must be scaled. Every possible outcome should give the same expected gain of a given amount bet. If the share that goes back to the customers is set to be 90% in average, the rescaling should be as follows:

[1]I did not manage to install *numpy* (or *matplotlib*). I have somehow a problem with *pip*. Version 7 seems to be installed but I was not able to upgrade it to version 9.

$$\begin{cases} s_1 P(dices = 2) = 0.9 \\ s_2 P(dices = 3) = 0.9 \\ \qquad\qquad ... \\ s_{11} P(dices = 12) = 0.9 \end{cases}$$

Where the s's denote the rescaling for each outcomes and I do not take into account the minimum amount that can be bet: the casino wants to have 10%of the total bet by customers who have the possibility to make money.

The function *Scale* computes the 6 different rescaling parameters ($s_1 = s_{11}$ etc...). Running Craps games with random bets and random amounts yields the desired result: every time a simulation of 1000 games is ran, the total share (or the average share) of the customer is very close to 90%.

2 Casino Implementation

Some modifications

I have come up with a different implementation of the casino bar and how customers order their drinks. Customers can order in-between rounds, which means four times for one evening of three rounds (they can order before the first round and after the last). A customer can order zero, one or two times if she likes to (it is randomly determined with equal probability) and for each order she can buy one or two drinks (also random) and then tip the bartender between 0 and 20. Thus in average, customers are expected to order about four times during an evening. One can have from 0 to 16 drinks in one evening.

I also put a limit of 20 drinks served per bartender in one salve (an in-between round) such that the number of barmen actually matters in the casino set-up[2] as having only one barman is always better as only one wage is paid if there is no capacity limit. The function *PickCustomer* randomly picks "unlucky" customers who will not be served.

[2]Barmen complete each others: what matters is the total capacity. Of course some will get more tips than others.

Similarly, I put a restriction on the number of players at each tables. If there is more than 10 at one table, some customers (called "shy" customers) do not bet (which is equivalent to betting an amount equal to 0). If a table has more than 10 players, every players is a shy player with probability 1/3. Moreover, to make it even more important to have a fair number of tables, customers who bet 0 do not order drinks (they are unhappy).

Regarding the gift given to bachelor, I give it if their budget becomes lower than 20 (and its the first time) for it to be given more often.

Class implementation

I do not create any hierarchy between different classes. I use OOP as it helps make the main code shorter and clearer.

I have five classes: Casino, Customer, Bar, Roulette and Craps. Roulette and Craps classes are similar to the one defined in the first parts of the project except now the bet is randomly determined inside the function *SimulateGame*.

I try as much as I can to only deal with attributes in every class. Indeed, if a parameter does not change during an evening, it only has to be created at the beginning. Generating an object of the corresponding class create that attribute automatically and it becomes very clear in the main code what is being called if it is called (for example *casino.cash* for the cash flow of a Casino object called casino).

The class Casino does not have any functions and only have fixed parameters (for one evening). Its purpose is only to gather those parameters and keep the code clear. The classes Customer and Bar have attributes that can change across rounds(budget, amounts bet, expenditures in drinks etc...). Calling one of their functions doest not return anything but updates or overwrite on or many of their attributes such that in the code there is only a and no assignment to a storing variable for these functions.

The three external helping functions *AssignTypes*, *AssignWealth* and *PickCustomers* only appear in the Customer or Bar classes where they help in assigning elements to the attributes.[3]

[3]Unfortunately, it is not convenient to make attributes inheritance in Python (that I know of). I could have put all the parameters in the class Casino and make the others class inherit from it but I would

Evening Simulation

The method *SimulateEvening* takes all the parameters and creates the objects of class Casino, Customer and Bar also defining automatically their attributes. As customers can order drinks before starting to play, orders are generated and wealth (budget) updated before entering the "rounds" loop. A few storing variables are also initialized.

- For every round, customers choose a table (randomly) and an amount to bet (which depends on the table). In a way, tables are given ID numbers: the first loop is for the Roulette tables and the second for the Craps tables.
  - The object Roulette (or Craps) is generated with a minimum amount that can be bet. There is here and exception in the "rules" of the script: the minimum amount does not change in an evening but because it is generated randomly and the class Customer needs it to generate the amounts bet (because of the returning customers) it must come form the object customer.
  - Before dispatching the customers to their tables, they need to be tracked with their ID number (which goes for 0 tp 99 for 100 customers).
  - The games are simulated for each tables and the casino income and the customers gains are then stored.
- Once all the games were played, all the customers gains are recovered in same order as before dispatch. Their wealth is then updated and they can order more drinks. Finally, storing variables are incremented.

The function can return anything. The results of interest only have to be stored during the process. Also note that all the numbers returned are integers.

See *Simulation.py* for the simulation and the bonus answers.

---

have had to rewrite all the attributes in each subclasses with *super()*