

Python para Modelagem Baseada em Agentes

Lista Exercicios II – Classes

Furtado, Bernardo Alves

June 29, 2020

Menu do dia

Class Exercício II – Transações: passo-a-passo

Contas I

1. Objetivo: criar agentes e interações. Agentes vão às compras e ganham satisfação. Lojas, com capacidade limitada, recebem os clientes. Vendem satisfação. Análise da interação (simples) na medida em que se aumentam clientes.
2. Na prática: (i) Criar três classes, (ii) Criar listas de agentes, (iii) Fazer a interação dos agentes, (iv) Generalizar/incluir variabilidade e contar/calcular resultados.

Contas II

3. Três classes distintas: Conta, Loja, Agente
4. Conta é simples. Inicia-se com `self.saldo = 0` e um número de registro `self.registration = i`
5. Conta também deverá ter dois métodos:
 - 5.1 Um que recebe recursos `quantia` e acrescenta ao balanço:
`self.saldo += quantia`
 - 5.2 E outro que retira recursos (`quantia`) do balanço (e retorna a `quantia` retirada) `self.saldo -= quantia` TODO
6. Tanto as classes Agente, quanto Loja terão uma Conta
7. E precisarão de um número de registro `i`

Contas III

8. Portanto, o primeiro método `__init__` de `Agent` e `Loja` deverá constar como uma instância da class `Conta`

- ▶ Como precisa de um parâmetro, o registro, fica `self.conta = Conta(i)`
- ▶ Notem que, para acessar o método depósito da `Conta`, será necessário algo como:
- ▶ `bob.conta.deposito(quantia)`
- ▶ Do mesmo modo, na `Loja`
- ▶ `loja.conta.deposito(quantia)`

Testem, testem, testem...

Contas IV

9. A class Loja deverá ter ainda no seu `__init__`, três variáveis: `capacidade`, `fun (satisfacao)`, `custo`.
10. Todas três podem ser geradas, no momento de criação da instância, por meio do `random.randrange(início, fim)`

```
random.randint(inicio, fim)
```

Contas V

11. Crie métodos, de forma que:

- 11.1 Sempre que ocorrer uma visita de um cliente, diminua a capacidade por 1: `self.capacidade -= 1`
- 11.2 Se a capacidade chegar a zero, retorne `False`. Senão, retorne `True`
- 11.3 Antes de permitir a visita e a compra, **cheque** se a capacidade permite

if condition:

Contas VI

`fun=experiencia`

12. O **Agente** tem pelo menos um método para acumular a `fun`, satisfação que recebe ao visitar cada **Loja**
13. Adicionalmente, pode-se ter um método que verifica se os recursos do **Agente** são suficientes para bancar o `Loja.custo`

Interações I

- ▶ Outro módulo, `import` as classes e contém algumas funções:
- ▶ Uma para criar os agentes e as lojas, a partir de um número `n` e controla o id específico `i`
- ▶ Criam-se listas
- ▶ Faz-se um `loop` utilizando `n`, `i`, `append` às listas, retorna
- ▶ Outra função, acessa as contas dos agentes e deposita recursos, por exemplo:
- ▶ `a[i].conta.deposito(random.randrange(10, 50))`
- ▶ Por fim, talvez a função mais difícil, a interação entre agentes e lojas.
- ▶ Por exemplo, para todos os agentes, escolhe-se uma loja aleatória: `random.choice(listalojas)`

Interações II

- ▶ Então, verifica-se a capacidade da loja, os recursos do agente, faz-se a visita, computa-se o gasto e adquire-se satisfação!

- ▶ Por exemplo:

loja

agente1

valor do custo -- reto

- ▶ `s1.conta.deposito(a[i].conta.retirada(s1.custo))`
`a[i].get_fun(s1.fun)`

- ▶ Por fim, escreva uma função que tire a média da satisfação do agente, do custo das lojas e das contas.

Generalização I

- ▶ Simplesmente, crie uma rotina que **testa números de agentes e de lojas, roda a interação e verificam-se os resultados:**
- ▶ **média fun, média saldo, média custo**
- ▶ Salve em um arquivo.
- ▶ Possivelmente, depois, será possível realizar plots(gráficos)
- ▶ Um pouco mais detalhado:
 - ▶ Crie uma lista com vários números de **Agentes**
 - ▶ Claro, com alguma relação óbvia. Por exemplo, aumentando linearmente de tamanho. Ou exponencialmente

Generalização II

- ▶ Crie uma lista com vários números de Lojas
- ▶ Em um `for loop`, chame a função principal do módulo interações (que recebe com parâmetros número de agentes e lojas).