

Rapport Mega Machine à Caoua

Paul Ecoffet

Mathieu Seurin

Vendredi 28 Novembre 2014

Nous allons ici détailler les fonctions utilisées dans notre programme (signature et axiomes) ainsi que faire l'analyse de leur complexité en notation **O**. Après chaque méthode, nous indiquerons le fichier correspondant aux tests que nous avons effectués, ainsi que les noms des tests.

Fonctions de Machine, mode fonctionnement

order : #TODO

1. Signature : $(\text{Monnaie}, \text{Commande}) \Rightarrow (\text{Boisson} \cup \text{Error} \cup \emptyset \times \text{Monnaie})$
2. Axiomes : $\forall \text{Commande} \in \{\text{Drink}\}$, Ensemble de tous les drinks possibles tel que $\forall \text{type} \in \text{Commande.stock}, \text{Commande.stock}[\text{type}] \neq \text{Machine.Stock}[\text{type}] \forall \text{Monnaie} \in \text{Coins}$
tel que $\text{Monnaie.compute_surplus}(\text{Machine.MaxCashInput}) \neq \text{Error}$ et $\text{Monnaie.value} > \text{Commande.price}$
3. Complexité : ON DOIT FAIRE LES AUTRES AVANT
4. Test : *test_machine.py*
 - test_order_simple()
 - test_order_complex()
 - test_order_fail_not_enough_cash()
 - test_order_fail_not_drink()
 - test_order_fail_no_stock()
 - test_order_cant_get_maxcash()

Fonctions de Machine, mode Maintenance

reset :

1. Signature : $\emptyset \Rightarrow \emptyset$
2. Axiomes :

$$\text{machine.reset}() \Rightarrow \begin{cases} \text{machine.}_\text{stocks}[\text{type}] = 0 & \forall \text{type} \in \text{machine.StocksType} \\ \text{machine.}_\text{coins}[\text{type}] = 0 & \forall \text{type} \in \text{machine.CoinsType} \\ \text{machine.}_\text{cash} = 0 \end{cases}$$

edit_prices :

1. Signature : $(\text{dictionnaire_prix}) \Rightarrow \emptyset \cup \text{Error}$
2. Axiomes :
 - $\forall \text{stock} \in \{\text{'thé'}, \text{'café'}, \text{'lait'}, \text{'chocolat'}\},$
 $\forall \text{prix} \geq 0$
 $\text{edit_prices}(\text{stock}=\text{prix}) \Rightarrow \text{machine.}_\text{stock_prices}[\text{stock}] = \text{prix}$
 - Si $\text{stock} = \text{'sucre'} \forall 0 \leq \text{prix}_i \leq \text{prix}_{i+1}, i \in [0, 3]$ $\text{edit_prices}(\text{sucre}=[\text{prix}_i])$
 $\text{machine.}_\text{stock_prices}[\text{stock}] = [(\text{prix}_i)] \forall i \in [0, 3]$
3. Complexité : $O(n)$ avec n le nombre de produits payant
4. Test : *test_machine.py*
 - test_edit_prices

edit_stocks :

1. Signature : $(\text{dictionnaire_stocks}) \Rightarrow \emptyset \cup \text{Error}$
2. Axiomes :
 - $\forall \text{stock} \in \{\text{'thé'}, \text{'café'}, \text{'lait'}, \text{'chocolat'}, \text{'sucre'}\},$
 $\forall \text{machine.quantite}[\text{stock}] < \text{quantite} \leq \text{machine.quantite_max}[\text{stock}]$
 $\text{machine.edit_stock}(\text{stock}=\text{quantite}) \Rightarrow \text{machine.quantite}[\text{stock}] = \text{quantite}$
 - $\forall \text{stock} \in \{\text{'thé'}, \text{'café'}, \text{'lait'}, \text{'chocolat'}, \text{'sucre'}\},$
 $\forall \text{quantite} \leq \text{machine.quantite}[\text{stock}]$ ou $\text{quantite} > \text{machine.quantite_max}[\text{stock}]$
 $\text{machine.edit_stock}(\text{stock}, \text{quantite}) \Rightarrow \text{machine.quantite}[\text{stock}] = \text{machine.quantite}[\text{stock}]$
3. Complexité : $O(n)$ avec n le nombre de stocks différents
4. Test : *test_machine.py*
 - test_edit_stocks

refill__stocks :

1. Signature : $\emptyset \Rightarrow \emptyset$
2. Axiomes :
 - $\forall \text{ stock} \in \text{Machine.StocksType},$
 $\text{machine.refill_stock}() \Rightarrow \text{machine.quantite}[\text{stock}]$
 $= \text{machine.quantite_max}[\text{stock}]$
3. Complexité : $O(n)$ avec n le nombre de stocks différents
4. Test : *test_machine.py*
 - test_edit_prices

edit__coins :

1. Signature : $\text{coins} \Rightarrow \emptyset$
2. Axiomes :
 - $\forall \text{ pieces} \in \text{machine.CoinsType} \text{ et } \text{pieces} \in \text{coins},$
 $\forall 0 \leq \text{coins}[\text{pieces}] \leq \text{machine._max_coins}[\text{pieces}]$
 $\text{edit_coins}[\text{coins}] \Rightarrow \text{machine._coins}[\text{pieces}] = \text{coins}[\text{pieces}], \forall \text{pieces}$
3. Complexité : $O(n)$ avec n le nombre de types de pièces différentes gérées par la machine
4. Test : *test_machine.py*
 - test_edit_prices

refill__coins :

1. Signature : $\emptyset \Rightarrow \emptyset$
2. Axiomes :
 - $\text{machine.refill_coins}() \Rightarrow \forall \text{ valeur} \in \text{Machine.CoinsType},$
 $\text{machine.coins}[\text{valeur}] = \text{machine.max_coins}[\text{valeur}]$
3. Complexité : $O(n)$ avec n le nombre de types de pièces différents
4. Test : *test_machine.py*
 - test_edit_prices

remove__stocks :

1. Signature : $\text{stock_dict} \Rightarrow \emptyset$
2. Axiomes :

- $\forall A = (\text{stock_type}, \text{value})_i, i \in \mathbb{N}, \text{stock_type}_i \in \text{Machine.StocksType}$
 $\text{machine.remove}(A) \Rightarrow \forall \text{stock_type}, \text{value} \in A, \text{machine._stocks}[\text{stock_type}]$
 $= \text{machine._stocks}[\text{stock_type}] - \text{value}$
- 2. Complexité : $O(n)$ avec n le nombre de types de stocks gérés par la machine
- 3. Test : *test_machine.py*
 - test_remove_stocks

add_to_cash :

1. Signature: $\text{Coins} \Rightarrow \emptyset$
2. Axiomes :
 - $\forall \text{coins} \in \text{Coins},$
 $\text{machine.add_to_cash}(\text{coins}) \Rightarrow \forall \text{type}, \text{quantite} \in \text{coins},$
 $\text{machine.cash}[\text{type}] = \text{machine.cash}[\text{type}] + \text{quantite}$

Fonctions de Coins

Coins hérite de collections.Counter.

compute_surplus

1. Signature: $\text{value} \Rightarrow \text{change} \in \text{Coins} \cup \text{NoChangePossibleException}$
2. Axiomes:
 - $\forall \text{coins in Coins}, \text{coins.value} \geq \text{value},$

$$\text{coins.compute_surplus}(x) \Rightarrow \begin{cases} (\text{coins} - \text{change}).\text{value} = \text{value} & \text{si possible} \\ \text{NoChangePossibleException} & \text{si impossible} \end{cases}$$
 - $\forall \text{coins in Coins}, \text{coins.value} \leq \text{value},$
 $\text{coins.compute_surplus}(x) \Rightarrow \text{NoChangePossibleException}$
3. Complexité: $O(2^n)$, n le nombre de pièces dans coins.
4. Tests: *test_coins.py*
 - test_compute_surplus

compute_change

1. Signature : $\text{change_value} \Rightarrow \text{Coins}$
2. Axiomes :
 - $\forall \text{ coins in Coins, coins.value} \geq \text{value},$
$$\text{coins.compute_change}(x) \Rightarrow \begin{cases} (\text{change}).\text{value} = \text{change_value} & \text{si possible} \\ \text{NoChangePossibleException} & \text{si impossible par division} \end{cases}$$
 - $\forall \text{ change, change} > \text{coins.value},$
 $\text{coins.self_compute}(\text{change}) \Rightarrow \text{NoChangePossibleException}$
3. Complexité: $O(n)$ avec n le nombre de types de pièces dans coins.
4. Tests: *test_coins.py*
 - test_compute_change
 - test_compute_change_not_enough_cash
 - test_compute_change_impossible

value

1. Signature : $\emptyset \Rightarrow \mathbb{N}$
2. Axiomes :
 - $\forall \text{ coins in Coins, coins} = (\text{valeur}, \text{quantite})_{n \in \mathbb{N}},$
$$\text{coins.value} = \sum_{i=1}^n \text{valeur}_i \times \text{quantite}_i$$
3. Complexité: $O(n)$ avec n le nombre de types de pièces dans coins.
4. Tests: *test_coins.py*
 - test_coins_value

Méthodes de Drink

price

1. Signature : $\emptyset \Rightarrow \mathbb{N}$
2. Axiomes :
 - $\forall \text{ stock_i} \in \text{drink.stocks}, \forall \text{ price_i} \in \text{drink.stock_prices}, i \in \{1, \dots, n\}$

$$\text{drink.price} = \sum_{i=1}^n \text{stock}_i \times \text{price}_i$$

3. Complexité : $O(n)$ avec n le nombre de types de stock différents dans drink.
4. Tests: *test_drink.py*
 - test_drink_price

has_beverage

1. Signature: $\emptyset \Rightarrow T, F$
2. Axiomes :
 - \forall drink avec $\exists x \in \{\text{'chocolate'}, \text{'tea'}, \text{'coffee'}\}, x \in \text{drink.stock}, \text{drink.has_beverage} \Rightarrow \text{T}$
 - \forall drink avec $\forall x \in \{\text{'chocolate'}, \text{'tea'}, \text{'coffee'}\}, x \notin \text{drink.stock}, \text{drink.has_beverage} \Rightarrow \text{F}$
3. Complexité : $O(1)$
4. Tests: *test_drink.py*
 - test_drink_has_beverage