ICC: Programmation — Miniprojet 3

GC/MX, 29 novembre 2019

Pendant cette séance, nous allons modifier notre code pour permettre à la grue de gérer des caisses de différents types et de gérer des instructions conditionnelles.

Exercice 1. Modélisation des types de caisses

Lisez tout l'exercice avant de commencer à faire des modifications: votre code ne fonctionnera pas correctement avant que toutes les étapes de ces modifications aient été faites.

(a) Nous aurons trois types de caisses différents, représentés par trois caractères différents: le type **`X`**, le type **`O`** et le type **`M`**. Ajoutez, dans **`cargobot.py`**, trois constantes avec ces noms, dont les valeurs sont le string qui contient le seul caractère correspondant.

(b) Il nous faut aussi changer la façon dont la grue est modélisée pour prendre en compte ces différents types de caisses. Dans **`Crane`**, faites les changements suivants (votre code ne compilera pas avant de les avoir tous faits):

- Le champ **`has_box`** doit disparaître. En effet, la grue ne doit pas simplement savoir si elle tient ou non une caisse, mais elle doit aussi connaître le type de cette caisse. Ajoutez donc à la place un champ nommé **`held_box`**. Par convention, on dira qu'il sera **`None`** lorsque la grue ne tiendra rien, et qu'il aura une des valeurs **`X`**, **`O`** ou **`M`** lorsque la grue tiendra une caisse du type correspondant. Son type est donc presque **`str`** – c'est soit la valeur **`None`**, soit une valeur de type **`str`**. On le déclare dans ce cas comme **`Optional[str]`**, en ajoutant **`from typing import Optional`** en haut du fichier.
- Le champ **`box_placements`** doit aussi être modifié dans son interprétation. Il ne s'agit plus de savoir combien de caisses on a dans chaque colonne, mais bien la liste précise des caisses empilées dans chaque colonne. Ce champ doit stocker plus d'informations: plus précisément, une **`List[List[str]]`** — une liste de listes: chaque liste intérieure représente une colonne (une série de caisses, donc dans le cas général, une séquence de **`X`**, **`O`** ou **`M`**), et la liste de listes représente l'ensemble des colonnes.
- Pour bien initialiser la position des caisses, il vous faudra aussi changer le type de l'argument **`initial_box_placements`** du constructeur pour qu'il soit, lui aussi, **`List[List[str]]`**.
- Comme vous avez changé le constructeur, vous ne pouvez plus initialiser la grue de la même manière dans **`gui.py`**. À la place de la ligne

```
crane = Crane([1, 0, 4, 3, 0])
```

  Faites ceci:

```
crane = Crane([
    [X],          # column 0
    [],           # column 1, empty
    [O, X, O, X], # column 2
    [M, M, M],    # column 3
    []            # column 4, empty
])
```

  Le nombre de caisses par colonne est toujours le même que dans l'exemple de la semaine passée, mais on a varié les types de caisse. Notez qu'on utilise directement les valeurs **`X`**, **`O`** ou **`M`** telles qu'importées depuis le module **`cargobot`**.
- Il y a toujours du code à modifier dans **`Crane`**. Faites les modifications nécessaires suite au changement de modélisation de l'état de la grue:
  - l'initialisation des champs dans le constructeur;
  - la méthode **`go_down_and_up`** (réfléchissez à ce que fait le code avec l'ancienne modélisation de la grue et faites-lui faire l'équivalent avec la nouvelle modélisation). *Indice:* l'appel de méthode **`pop()`** sur une liste supprime le dernier élément de la liste et vous le retourne;
  - la méthode **`__repr__`** — vous ne l'avez pas écrite, mais devez être capable de faire les modifications nécessaires dans le calcul de la variable **`max_height`** et dans la boucle qui la suit immédiatement;
  - attention aux deux endroits dans **`__repr__`** où le code insérait le string **`"X"`** pour signaler la présence d'une caisse: maintenant, vous devez utiliser ce qui correspond au type de caisse en question.

Tout devrait à nouveau fonctionner, et vous devriez maintenant pouvoir faire tourner votre code en voyant les trois types de caisses à l'écran.

Exercice 2. Rappel: Dictionnaires

Ajoutez du code à la fin de la méthode **`__init__`** de **`Crane`** pour que la grue affiche sur la console des informations sur les caisses avec lesquelles elle a été initialisées. Pour notre exemple, nous voulons qu'elle affiche ceci:

```
Crane ready with 8 boxes.
- 3 boxes of type X
- 2 boxes of type O
- 3 boxes of type M
```

On vous demande explicitement que votre code marche même si de nouveaux types de caisses sont utilisés dans le futur: avec d'autres valeurs et/ou avec un nombre différent de valeurs. Pour ce faire, utilisez un **`Dict[str, int]`**, qui reliera le **`str`** représentant un type de caisse au nombre de fois que cette caisse est présente initialement dans la grue, et comptez les caisses en itérant à travers les colonnes.

Exercice 3. Ajout d'instructions de conditions

Nous voulons maintenant permettre à la grue, en étendant sa liste d'instructions, d'ajouter des instructions-conditions. Nous aurons deux conditions: (1) la grue tient une caisse (**`IF_BOX`**), (2) la grue ne tient pas de caisse (**`IF_NO_BOX`**). L'idée est qu'une instruction donnée (par exemple **`GO_DOWN_AND_UP`**), si elle est précédée de l'instruction-condition **`IF_BOX`**, ne sera exécutée que si la grue tient une caisse. Ceci est conceptuellement la même chose que ce qui vous a été démontré pendant le cours sur le vrai Cargo-Bot.

(a) Dans **`cargobot.py`** rajoutez les deux valeurs possibles pour nos instructions-conditions: **`IF_BOX`** et **`IF_NO_BOX`**. Donnez-leur aussi une représentation textuelle de 3 caractères, par exemple **`"*?:"`** et **`"_?:"`**, respectivement.

(b) Pour implémenter le comportement de ces instructions, nous devons modifier la fonction **`execute_program`**.

- Ajoutez les deux blocs de code **`if`** ou **`elif`** nécessaires au traitement des deux nouvelles instructions.
- Pour ces deux cas, évaluez la condition (vous aurez besoin de pouvoir demander à la grue si elle tient ou non une caisse actuellement: pour ceci, accéder au champ **`held_box`** de la grue) et stockez cette valeur de type **`bool`** dans une variable locale qui sera accessible lors de la prochaine itération.
- Insérez ensuite en haut du code de votre boucle, avant de faire la série de **`if-elif`**, du code qui va vérifier cette valeur **`bool`** et sauter l'exécution de l'instruction en cours si la condition est **`False`**.

Testez votre code en initialisant votre grue avec cette configuration:

```
crane = Crane([
    [],        # column 0
    [X, M, O], # column 1
    [O],       # column 2
    [],        # column 3
    [M, X]     # column 4
])
```

Et ce programme, qui est censé aller chercher toutes les caisses à droite et les empiler dans la première colonne:

```
program = Program(
    P1=[GO_TO_P2, GO_TO_P2, GO_TO_P2, GO_TO_P2, GO_TO_P2, GO_TO_P2],
    P2=[GO_TO_P3, GO_DOWN_AND_UP],
    P3=[GO_RIGHT, GO_DOWN_AND_UP, IF_NO_BOX, GO_TO_P3, GO_LEFT]
)
```

(c) Ajoutez les instructions-conditions **`IF_BOX_X`**, **`IF_BOX_O`** et **`IF_BOX_M`** et implémentez-en le comportement correspondant pour faire en sorte de tester spécifiquement si la grue contient une caisse de type **`X`**, **`O`** ou **`M`**, respectivement.

(d) *(Si vous avez le temps…)* Trouvez comment c'est possible que le code du programme ci-dessus, qui contient une seule instruction **`GO_LEFT`**, fasse le bon nombre de mouvements à gauche pour ramener une caisse qui se trouve sur n'importe quelle colonne vers la première.