ICC: Programmation — Miniprojet 1

GC/MX, 15 novembre 2019

Nous allons développer une petite application graphique qui permettra à l'utilisateur de piloter et de programmer (un tout petit peu) une grue, selon le modèle du jeu Cargo-Bot (http://twolivesleft.com/CargoBot/). Nous allons donc programmer quelque chose qui pourra lui-même être programmé…

Exercice 1. Modélisation de la grue et de la position des caisses

(a) Créez un nouveau dossier dans votre workspace, **`miniprojet`**. C'est dans ce dossier qu'on mettra les différents fichiers du miniprojet, et vous pourrez le réutiliser tout au long des changements que nous ferons lors des séances futures. Créez-y un nouveau fichier **`cargobot.py`**, qui contiendra les structures de données nécessaires — pour le moment, ce sera uniquement la classe **`Crane`** pour modéliser la grue. Cette classe modélisera l'état complet de la grue à un moment du jeu. Selon la démo faite en cours, on doit donc savoir (1) le nombre de colonnes qu'il y a dans le jeu (c'est-à-dire le nombre d'emplacements où l'on pourra empiler des caisses); (2) la position actuelle de la grue; (3) si oui ou non la grue est en train de transporter une caisse; (4) le nombre de caisses déjà empilées dans chacune des colonnes. Nous allons faire les choix de modélisation suivants:

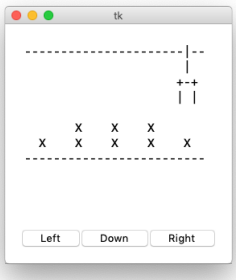
(1) Le nombre de colonnes sera un attribut de notre classe **`Crane`**, **`num_columns`**, de type **`int`**. Cet attribut ne sera pas censée changer: durant un exercice, le nombre de colonne reste constant.

(2) La position de la grue sera aussi un champ de type **`int`**, nommé **`position`**. Si l'on veut que la grue puisse bouger, il ne sera évidemment pas constant, mais évoluera selon les mouvements faits. Par convention, on dira que ses valeurs vont de 0 (la position la plus à gauche) à **`num_columns - 1`**, la position la plus à droite.

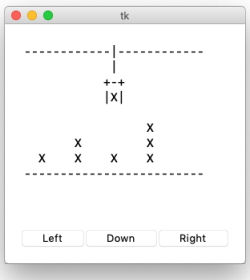
(3) Pour indiquer si la grue porte actuellement une caisse ou non, on utilisera un champ **`has_box`** de type **`bool`**.

(4) Finalement, pour représenter combien de caisses se trouvent où, on utilisa un champ **`box_placements`** qui sera une liste de **`int`** de taille **`num_columns`**, et où la case numéro **`i`** indiquera combien de caisses se trouvent dans la **`i`**$^{\text{e}}$ colonne.

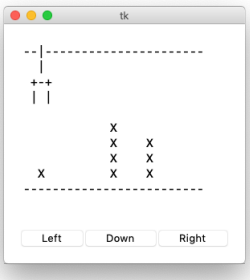
Les exemples suivants montrent comment trois situations sont représentées selon ce modèle:




```
num_columns = 5
position = 4
has_box = false
box_placements =
[1, 2, 2, 2, 1]
```




```
num_columns = 5
position = 2
has_box = true
box_placements =
[1, 2, 1, 3, 0]
```




```
num_columns = 5
position = 0
has_box = false
box_placements =
[1, 0, 4, 3, 0]
```

Ajoutez la méthode standard **`__init__`** qui prend un seul argument nommé **`initial_box_placements`** de type **`List[int]`**, qui indique la position de départ des caisses. Initialisez les champs **`num_columns`** et **`box_placements`** à partir de cet argument. Pour **`position`** et **`has_box`**, initilisez-les pour représenter la position de la grue tout à gauche, sans tenir de caisse.

(b) Depuis la page Moodle du cours, copiez-collez le code complémentaire de l'exercice 1 (b) dans votre classe **`Crane`**, qui donne à la grue une implémentation pour **`__repr__`** comme utilisé ci-dessus. Lisez et comprenez (dans les grandes lignes) ce que ce code fait.

(c) Dans le dossier du miniprojet, créez un nouveau fichier **`terminal.py`**, qui va nous permettre de jouer avec la grue via une interface terminal. (Plus tard, nous ajouterons une interface graphique.) Dans ce fichier, importez (avec un **`from-import`**, pas en faisant copier-coller) la classe **`Crane`** définie plus haut. Créez une nouvelle grue avec une position initiale qui correspond à l'exemple de droite ci-dessus et affichez-le sur le terminal avec **`print`**. La représentation textuelle devrait correspondre.

Exercice 2. Opérations sur la grue

Nous allons maintenant ajouter trois méthodes dans **`Crane`** pour simuler un déplacement à gauche, un déplacement à droite, et enfin un déplacement «vers le bas puis remontée».

(a) Ajoutez une méthode **`go_right`** à votre grue. Dans le corps de la méthode, faites les modifications nécessaires au(x) champ(s) affecté(s) par un tel déplacement selon la modélisation décrite à l'exercice 1.

(b) Testez votre nouvelle méthode depuis le fichier **`terminal.py`**, en affichant la grue avec **`print`** avant et après un déplacement à droite avec **`go_right`**.

(c) On veut empêcher que le code ne déplace la grue trop à droite. Pour ce faire, on va faire volontairement crasher le code, et ce en créant nous-mêmes une erreur d'exécution.

Modifiez votre méthode **`go_right`** pour vérifier, avant d'effectivement bouger la grue, si elle n'est pas déjà tout à droite. Si c'est le cas, «levez une exception» avec ce code:

```
raise RuntimeError("bad move")
```

Testez cette modification.

(d) De la même manière, ajoutez, implémentez et testez la méthode **`go_left`**. Faites en sorte que le programme crashe si l'on déplace la grue trop à gauche.

(e) Finalement, ajoutez la méthode **`go_down_and_up`**. Elle agit comme suit:

- si la grue porte actuellement une caisse, elle la dépose dans la colonne dans laquelle elle se trouve;
- sinon (la grue ne porte pas de caisse):
  - soit le nombre de caisses déposées dans la colonne où la grue se trouve est supérieur à zéro et la grue prend une caisse de cette colonne et la tient;
  - soit il n'y a pas de caisse dans cette colonne, et rien de spécial ne se passe.

Testez votre méthode de la même manière que précédemment.

(f) Dans **`terminal.py`**, effacez votre code de test et faites en sorte que l'on puisse jouer avec votre grue dans le terminal. Pour ce faire, téléchargez le fichier complémentaire **`utils.py`** de la page Moodle du cours dans votre dossier de miniprojet, Ensuite, votre code peut procéder ainsi:

(1) Créez une grue dans un état initial. Effacez tout le terminal (avec la fonction **`clear_terminal`** définie dans le module **`utils`**), puis affichez votre grue avec **`print`**.

(2) Dans une boucle sans fin: récupérez le caractère tapé dans le terminal avec la méthode **`input_char`** du module **`utils`**. Puis:

- si c'est **`"a"`**, déplacez la grue à gauche, effacez le terminal et réaffichez-la;
- si c'est **`"d"`**, déplacez la grue à droite, effacez le terminal et réaffichez-la;
- si c'est **`" "`** (espace), faites descendre et remonter la grue, effacez le terminal et réaffichez-la;
- si c'est **`"q"`**, sortez de la boucle dans fin avec un **`break`**.

Vous devriez ainsi pouvoir jouer dans le terminal avec la grue.

Exercice 3. Interface graphique

Nous allons maintenant créer une fenêtre avec trois boutons pour piloter la grue, comme dans les exemples montrés plus haut.

(a) Créez un nouveau fichier **`gui.py`**, qui va contenir le code de l'interface graphique tout en réutilisant les structures de données définies dans **`cargobot.py`**, à l'instar de ce que nous avons fait dans **`terminal.py`**. Vous pouvez utiliser en haut de votre fichier pour importer ce dont vous aurez besoin:

```
from tkinter import * # type: ignore
from tkinter.ttk import * # type: ignore
from cargobot import Crane
```

Rappel: la documentation de Tkinter est ici: https://anzeljg.github.io/rin2/book2/2405/docs/tkinter/index.html.

(b) Avec l'aide des slides du cours et la documentation de Tkinter, créez une nouvelle fenêtre principale (avec l'expression `Tk()`). Dans celle-ci, ajoutez une zone de texte centrale (elle contiendra la représentation textuelle de la grue) et trois boutons: Left, Down et Right.

- La zone de texte centrale doit avoir une hauteur de 12 lignes de texte et une largeur de 28 caractères. Elle doit être redimensionnée en largeur et en hauteur avec la fenêtre.
- Les trois boutons doivent se trouver en dessous et prendre chacun un tiers de la largeur de la zone de texte, et être redimensionnés en largeur uniquement. Pour l'instant, ils n'ont pas d'effet.

Utilisez des appels de type `grid(...)`, `rowconfigure(...)` et `columnconfigure(...)` sur les bons éléments pour régler l'affichage.

Les images de la première page montrent à quoi votre interface doit ressembler. N'oubliez pas l'appel `mainloop()` tout à la fin, à faire sur l'objet retourné par `Tk()`.

(c) Initialisez un objet de type `Crane` avec une situation initiale donnée. Ajoutez une fonction `update_view` qui efface tout le contenu de la zone de texte centrale et qui le remplace par la représentation actuelle de la grue. Pour ce faire, faites les appels de méthodes respectifs suivants, toujours sur l'objet qui représente votre zone de texte: `delete(1.0, END)` et `insert(1.0, repr(crane))`.

Appelez votre fonction juste avant l'appel `mainloop()`. Votre zone de texte doit maintenant afficher la représentation initiale de votre grue.

(d) Faites en sorte qu'un clic sur le bouton Left déplace votre grue à gauche puis mettre à jour l'affichage avec `update_view`. Faites en sortes que les deux autres boutons fassent aussi ce pour quoi ils ont prévus.

Vous devriez maintenant pouvoir jouer avec la grue dans la fenêtre.