Pendant cette séance, nous allons modifier la représentation graphique de la grue et du programme.

Exercice 1. État final et arrêt automatique

(a) Nous allons modéliser un «exercice» dans notre réplique de Cargo-Bot. Pour cela, ajoutez une nouvelle classe **`Exercise`** à **`cargobot.py`**. Un exercice sera représenté par:

- un état initial, nommé **`initial_box_placement`**;
- un état cible, nommé **`target_box_placement`**;
- un programme à exécuter, nommé **`program`**.

Déclarez la classe et ajoutez la méthode **`__init__`** qui accepte les valeurs pour chacun de ces champs et initialisez les trois champs en conséquence.

(b) Dans **`cargobot.py`**, ajoutez cette définition (copiable depuis la page Moodle), qui crée un exercice à quatre colonnes, la première étant initialement vide. Le but du programme est de déplacer toutes les colonnes d'un cran vers la gauche.

```
GoLeft = Exercise(
    initial_box_placement = [
        [],
        [X, X, X],
        [O, O, O],
        [M, M, M]
    ],
    target_box_placement = [
        [X, X, X],
        [O, O, O],
        [M, M, M],
        []
    ],
    program = Program(
        P1 = [IF_NO_BOX, GO_RIGHT, GO_DOWN_AND_UP, IF_BOX, GO_LEFT, GO_TO_P1]
    )
)
```

(c) L'idée est maintenant de lancer le programme avec un objet de type **`Exercise`**, qui servira à connaître la position de départ, la position voulue et le programme à exécuter. Changez donc votre fichier **`gui.py`** pour stocker, dans une variable **`exercise`**, vers le haut du fichier, l'exercice sur lequel la grue va opérer. Initialisez cette variable à l'exercice **`GoLeft`**. Changez ensuite l'initialisation des variables **`crane`** et **`program`** de manière à utiliser les informations véhiculées par l'exercice en question: la grue doit être initialisée avec la position de départ spécifié dans l'exercice, et le programme à exécuter de même. Quant à la position cible des caisses stockée par l'exercice, nous la traiterons plus tard.

À ce stade, votre code devrait faire exécuter à la grue les instructions fournies par l'exercice stocké dans la variable **`exercise`**. Ce code, cependant, est une boucle infinie et va finir par casser la grue en la faisant aller trop à droite, car nous ne stoppons pas encore automatiquement l'exécution lorsque les caisses se trouvent dans la position désirée.

(d) Ajoutez à la classe **`Crane`** une méthode qui dira si oui ou non les caisses sont actuellement dans un état donné, tel que passé en paramètre. Appelez cette méthode **`is_in_state`**. Donnez-lui le bon type de retour, le bon type pour son paramètre, et implémentez-la.

Indice: pour vérifier si deux listes **`list1`** et **`list2`** contiennent les mêmes éléments dans le même ordre, vous pouvez simplement tester si **`list1 == list2`** plutôt que d'écrire une boucle.

(e) Dans **`execute_program`**, avant d'exécuter une instruction, vérifiez avec **`is_in_state`** si l'état cible des caisses est atteint. Si c'est le cas, arrêtez l'exécution du programme.

Votre grue devrait maintenant s'arrêter toute seule après avoir déplacé les trois colonnes de caisses. (Si vous voulez que votre grue se déplace plus rapidement, changez la durée de pause dans la méthode **`pause`**.)

Exercice 2. Représentation graphique de la grue

Nous allons changer la représentation textuelle de la grue par une représentation graphique. Pour ce faire, nous allons déclarer et implémenter une sous-classe de **`Canvas`**, une classe de Tkinter qui peut «dessiner» des lignes, rectangles, etc. à l'écran.

(a) Créez un nouveau fichier **`widgets.py`** et copiez-collez-y le code de base de **`CraneView`**, disponible sur la page Moodle du cours.

Observez un peu ce code avec ces indications.

- La classe **`CraneView`** est une sous-classe de **`Canvas`**, qui représente un élément graphique que l'on peut ajouter à une fenêtre de Tkinter. C'est donc un élément graphique personnalisé que nous définissons.
- Sa méthode **`__init__`** demande une référence sur la grue dont il faut représenter l'état.
- La classe est précédée de quelques définitions de constantes de type **`int`**.
- La méthode spéciale **`redraw`** est appelée automatiquement par Tkinter chaque fois que cet élément graphique doit s'afficher.
- Le corps de cette méthode utilise des méthodes sur **`self`** pour dessiner des choses à l'écran. Elle dessine d'abord une ligne en haut, puis la grue, puis (le cas échéant) la caisse que la grue tient, et enfin toutes les caisses dans les colonnes en bas. Pour chaque commande de dessin, des coordonnées sont calculées en utilisant les constantes définies plus haut et l'état de la grue. Les coordonnées sont toujours exprimées depuis le point supérieur gauche: l'axe X va donc vers la droite et l'axe Y descend.
- Le code pour dessin d'une caisse en particulier, utilisé plusieurs fois, a été factorisé dans la méthode annexe **`draw_box`**.

(b) Dans **`gui.py`**, changez la défintion de la variable **`crane_view`** pour qu'elle soit maintenant un **`CraneView`** et non plus un **`Text`** de Tkinter. Pour cela, importez **`CraneView`** du module **`widgets`**. Ensuite:

- changez l'initialisation du champ en créant un nouveau **`CraneView`** avec les bons paramètres;
- changez l'implémentation de la méthode **`update_view`** pour appeler successivement les méthodes **`update`** et **`redraw`** sur **`crane_view`**. Appeler **`redraw`** dit à Tkinter que cet élément graphique a besoin de se redessiner car son état a changé.

Vous devriez maintenant voir **`CraneView`** en action si vous faites tourner votre code.

(c) Ajoutez du code tout au début de la méthode **`draw_box`** dans **`CraneView`** pour faire en sorte que les caisses de types différents (**`X`**, **`O`**, **`M`**) aient des couleurs de fond différentes.

Exercice 3. Représentation graphique du programme

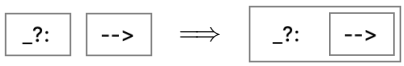
(a) Dans **`widgets.py`** réez une nouvelle classe **`ProgramView`**, qui nous servira à afficher les quatre sous-programmes et leurs instructions. Copiez-collez-y le code de base de cette classe, disponible sur la page Moodle du cours. Lisez-le (la structure est similaire à celle de **`CraneView`**) et comprenez-en l'essentiel.

(b) Modifiez votre code dans **`gui.py`** pour utiliser **`ProgramView`** à la place du **`Text`** comme type pour le champ **`program_view`**. Pour mettre à jour ce que **`program_view`** fait, utilisez la méthode **`redraw`** avec les bons arguments (quand aucun sous-programme ne s'exécute, vous pouvez passer **`None`** et **`-1`** pour les arguments **`current_subprogram`** et **`current_instruction_index`**, respectivement).

Faites également en sorte que, lorsque l'exécution s'arrête, aucune instruction ne reste mise en évidence.

(c) Modifiez **`ProgramView`** pour faire en sorte que la couleur de fond utilisée pour montrer l'instruction en cours d'exécution soit différente selon que l'exécution est (i) un mouvement de grue; (ii) un saut à un des sous-programmes; ou (iii) une condition.

(d) Faites en sorte que l'affichage d'une instruction de type condition englobe l'instruction de droite sur laquelle l'instruction agit, selon ce schéma:



Pour tester tout ceci avec des situations plus complexes, ajoutez à **`cargobot.py`** les définitions disponibles sur la page Moodle et essayez d'assigner d'autres exercices à la variable **`exercise`** de **`gui.py`**.