Sciences Manuelles du Numérique *Cargo-bot*

Auteur : Benjamin.Wack@imag.fr
Université Grenoble Alpes, IREM-Grenoble

Préambule

Lorsqu'on a trouvé une solution à un problème de Cargo-bot, on a écrit un *algorithme* pour le résoudre : une méthode qui marche à tous les coups pour arriver au résultat voulu. La grue n'a alors plus qu'à *exécuter* cet algorithme, c'est-à-dire faire exactement ce qu'on lui dit. En revanche, elle n'a pas besoin de « comprendre » ce qu'elle fait pour le faire correctement, ni de savoir à quoi va ressembler le résultat final.

Le cadre dans lequel on écrit ces algorithmes est relativement contraint :
— très peu d'instructions élémentaires (aussi appelées *primitives*) ;
— la place allouée à l'écriture des programmes est restreinte
ce qui force à rechercher une certaine concision dans l'écriture des algorithmes.

De plus, plusieurs solutions distinctes sont parfois possibles ; se pose alors la question des critères pour les comparer. Le format des problèmes tend à valoriser les solutions courtes, mais on pourra aussi discuter de la lisibilité des programmes produits.

Un ou plusieurs programmes

Une particularité dans Cargo-Bot, c'est qu'on écrit non pas un mais quatre programmes, qui peuvent ensuite s'*appeler* les uns les autres. Cela permet de regrouper dans un programme une suite d'instructions particulière (par exemple qui déplace un gobelet d'un cran vers la droite), puis d'appeler ce programme plusieurs fois (pour déplacer plusieurs gobelets). On crée ainsi une sorte de boucle.

Un programme conçu de cette façon sera d'autant plus facile à comprendre et à modifier que chaque petit programme est prévu pour résoudre une tâche particulière bien définie.

Avec un peu d'habitude, on peut même s'appuyer sur ce mécanisme pour construire ses programmes, par ce qu'on appelle l'*analyse descendante*. On découpe le problème à résoudre en sous-problèmes, et on cherche à écrire un morceau de programme résolvant chacun des sous-problèmes. Les sous-programmes les plus souvent utilisés sont nommés pour pouvoir être appelés plusieurs fois ; on peut même utiliser leur nom dans un autre programme avant de les avoir réellement écrits !

Récursivité

Si on veut déplacer toute une pile de gobelets vers la droite, au lieu de répéter plusieurs fois l'appel à un programme, il est plus économique d'écrire un programme qui signifie « Déplace un gobelet vers la droite et recommence ». De plus ce programme fonctionnera quelle que soit la taille de la pile à déplacer. C'est possible si on écrit un programme qui s'appelle lui-même.

En toute généralité, il est possible de remplacer complètement la structure de boucle algorithmique par ce mécanisme de programme auto-appelant. Cela donne des langages de programmation assez surprenants, dits *fonctionnels*, dans lequel l'écriture d'un programme ressemble à la définition d'une suite récurrente en mathématiques.

Au premier abord, devant un tel programme, on se demande souvent s'il va s'arrêter, puisqu'il se relance lui-même. Même au sein de la communauté scientifique informatique, il a fallu un certain nombre de travaux avant que tout le monde soit convaincu que cette manière de procéder était correcte. En réalité, il est très fréquent d'obtenir un programme infini si on ne prend pas suffisamment de précautions, mais c'est déjà le cas avec les boucles.

Dans Cargo-Bot, la situation est un peu biaisée : en effet, la grue s'arrête dès qu'on arrive dans une configuration gagnante. Il y a donc une « boucle d'exécution » implicite, qui teste systématiquement si le but est atteint. C'est également le cas lorsqu'on fait de la *programmation événementielle* comme en Scratch, où beaucoup de conditions (appui sur une touche, collision entre objets...) sont implicitement testées à intervalles réguliers.

Des primitives

Au fur et à mesure des problèmes, l'ensemble des primitives proposées évolue : manipulation de la grue, appels de sous-programmes, instructions conditionnelles...

De manière générale, lorsqu'on programme une machine, le jeu d'instructions dont on dispose correspond aux actions qu'est capable de réaliser physiquement la machine. Cependant, afin de faciliter la tâche du programmeur, ces instructions sont représentées par des *symboles*, suffisamment clairs pour être compris par un humain avec un peu d'apprentissage, mais choisis rigoureusement pour que la machine puisse ensuite les traduire en actions sans ambiguïté. Tout symbole inexistant dans le langage, ou toute utilisation inadaptée d'un symbole, conduira à un refus de la machine de fonctionner.

Cette contrainte dans l'écriture des programmes semble parfois pénible lorsqu'on apprend à programmer, mais elle est indispensable : le programmeur a besoin de savoir comment la machine va se comporter, ce qui ne serait pas possible si on lui laissait la moindre liberté d'interprétation.

Parmi ces symboles, on pourra distinguer ceux qui correspondent à des actions élémentaires (déplacement de la grue), et d'autre part ceux qui permettent de moduler ou de combiner entre elles d'autres instructions. Ainsi, un programme formé d'une étiquette conditionnelle seule n'a pas de sens, elle doit obligatoirement être associée à une instruction.

De plus, on combine également des instructions en les écrivant l'une à la suite de l'autre dans l'ordre où elles doivent s'exécuter ; ce type d'enchaînement est souvent implicitement représenté par un alignement vertical ou horizontal dans les langages de programmation, par économie d'écriture.

Plus on veut écrire des algorithmes complexes dans Cargo-Bot, plus on introduit des instructions proches de celles des langages de programmation.

Notion d'état et de mémoire

Dans Cargo-Bot, l'objet sur lequel agit le programme est la position des gobelets à un moment donné. Plus généralement, on ne devrait pas dire qu'un programme « fait quelque chose »

quand on l'exécute, mais qu'il « transforme quelque chose ». En général, c'est sur la mémoire d'un ordinateur qu'il agit, mais pas seulement : les programmes peuvent aussi commander des périphériques, des actionneurs électroniques, des mécanismes robotisés...

Est-il vraiment différent de manipuler des gobelets ou des nombres dans une mémoire ? Pour l'écriture du programme, pas tellement : l'important reste de déterminer une suite d'instructions qui nous permettra de passer d'un état initial à un état final via un certain nombre de transitions élémentaires.

Mais si on pense plutôt au type de calculs que pourront effectuer nos programmes, alors la différence est claire : dans une mémoire d'ordinateur, il est toujours possible de dupliquer et d'effacer des informations, alors que bien sûr le nombre de gobelets présents sur le terrain de jeu ne peut pas changer au cours des manipulations. Pour autant, il reste possible de représenter des opérations simples (par exemple une addition sous la forme de deux piles de gobelets qu'on rassemblera en une seule), mais cela reste fastidieux et limité.

À propos de langages de programmation

Il existe des milliers de langages de programmation, chacun avec ses spécificités. Selon le programme qu'on souhaite écrire, il pourra être facile de l'écrire dans certains langages, et plus difficile dans d'autres. Cependant, il est intéressant de noter deux caractéristiques importantes :

— Tous les langages de programmation permettent d'écrire tous les algorithmes possibles. Parfois il sera très compliqué, voire infaisable en pratique, de traduire un programme d'un langage dans un autre, mais en théorie c'est toujours possible.

— La plupart des langages offrent les mêmes instructions pour combiner les instructions entre elles : séquence, instruction conditionnelle, boucle.

Ainsi, une fois qu'on connaît un ou deux langages de programmation, il devient nettement plus facile d'en apprendre d'autres, car on retrouve facilement des structures communes.

En revanche, pour vraiment utiliser correctement un langage donné, il faut bien connaître la partie qui lui est spécifique : les bibliothèques disponibles, les structures de données représentables, les facilités de manipulation de ces structures de données... et cela peut demander un apprentissage conséquent.

Remerciements Cargo-Bot est un jeu de Rui Vianna, adapté en Javascript par Joe Tessler.

Merci au groupe Informatique Sans Ordinateur de l'IREM de Clermont-Ferrand pour les échanges fructueux et leurs apports à l'élaboration de cette activité.