SIMULATION D'OBJETS TECHNIQUES : de la programmation PROCÉDURALE à la programmation DÉCLARATIVE

Christian ORANGE, Jacky COURTAIS

Un des principaux intérêts pédagogiques de Logo est qu'il permet la définition de procédures qui peuvent venir s'ajouter aux primitives du langage et l'enrichir (macro-primitives). On peut créer de cette façon de nombreux micro-mondes spécialisés.

II est ainsi possible de fournir aux élèves, sous forme de macro-primitives, les fonctions et les commandes de base d'un objet technique qu'ils pourront analyser et simuler. Le logiciel Ecluse (valise I.P.T.) est un micro-monde de ce type (voir, en particulier l'article de M. Dupont, *bulletin E.P.I.* n°47). Nous proposons en annexes le cahier des charges et la liste de macro-primitives permettant la simulation d'un monte-charge simple (représenté en annexe 1).

II ne s'agit pas ici de décrire une utilisation pédagogique de ces macro-primitives mais de montrer comment elles permettent des programmations très différentes du monte-charge, allant de la

programmation séquentielle pure à la programmation déclarative par règles de production. C'est aussi une illustration simple des caractéristiques de ces divers types de programmation.

Convenons que le monte-charge (voir schéma de l'annexe 1) doit aller chercher, à partir de sa position de départ, la caisse à l'étage inférieur et la monter à l'étage supérieur.

1) PREMIÈRE APPROCHE : PROGRAMMATION PUREMENT SÉQUENTIELLE

Cette programmation n'utilise que la séquence et la répétition (REPETE) qui n'est qu'une séquence "ramassée". Elle nécessite la connaissance de la distance entre les étages et du déplacement nécessaire pour le chargement et le déchargement de la caisse. Ces

distances peuvent être trouvées par tâtonnements en déplaçant les objets pas à pas grâce à l'utilisation en mode direct des macro-primitives.

Voici le résultat d'une telle analyse du problème (nous supposons l'affichage du dessin fait, grâce à DESSIN, pour ne considérer que le fonctionnement).

Le travail du monte-charge est décomposé en quatre actions successives : DESCENDRE, CHARGER, MONTER, DECHARGER.

Donc :

```
POUR TRAVAIL
DESCENDRE
CHARGER MONTER
DECHARGER
FIN
```

avec :

```
POUR DESCENDRE
REPETE 38 [DESCENDS]
FIN
POUR CHARGER
REPETE 9 [CAISSE-D]
FIN
```

```
POUR MONTER
REPETE 38 [MONTE]
FIN
POUR DECHARGER
REPETE 9 [CAISSE-G]
FIN
```

Malgré sa structuration, cette façon de programmer le monte-charge est bien complètement séquentielle. Elle n'utilise aucun des prédicats : le monte-charge ne "sait" où il en est que par référence à tout ce qu'il a déjà fait. Son fonctionnement est donc totalement rigide et un arrêt forcé de ce programme (par un CNT C) empêche toute reprise au point où il en est.

2) DEUXIÈME APPROCHE : PROGRAMMATION UTILISANT LA SÉQUENCE ET LA CONDITION

Pour rendre cette machine un peu plus souple il est intéressant de faire intervenir dans sa programmation les prédicats de position donnés comme macro-primitives (voir cahier des charges). Ces prédicats permettent, pendant le fonctionnement, une prise d'information sur l'"extérieur". Ils correspondent aux interrogations des capteurs de la machine réelle.

On garde la même subdivision du travail en quatre actions successives (séquence) mais chacune de ces actions est programmée d'une manière différente de celle exposée ci-dessus.

La procédure TRAVAIL donc ne change pas mais

```
POUR DESCENDRE
SI BAS? [STOP]
DESCENDS
DESCENDRE
FIN
POUR CHARGER
SI CHARGE? [STOP]
CAISSE-D
CHARGER
FIN
```

```
POUR MONTER
SI HAUT? [STOP]
MONTE
MONTER
FIN
POUR RECHARGER
SI DECHARGE? [STOP]
CAISSE-G
RECHARGER
FIN
```

Remarque : on utilise ici la récursivité du langage Logo ; il s'agit uniquement de récursivité terminale qui joue le même rôle que le TANT QUE (ou WHILE) d'autres langages.

Cette utilisation des prédicats, qui prennent leurs informations à l'"extérieur", rend la machine plus souple. Ainsi, si un arrêt du programme au milieu de la montée, par exemple, ne permet pas la relance de la procédure TRAVAIL (car celle-ci contient une séquence qui assujettit son fonctionnement à tout ce qui s'est déjà passé), il est possible alors d'utiliser la procédure MONTER qui fera arriver le plateau en haut exactement, ce qui n'aurait pas été le cas avec la procédure MONTER purement séquentielle de la première approche.

La comparaison de ces deux premières façons de programmer, très habituelles l'une comme l'autre en Logo, montre que la rigidité du système vient à chaque fois des séquences. II serait intéressant de les supprimer complètement : cela revient à faire de la programmation déclarative.

3) TROISIÈME APPROCHE : PROGRAMMATION DÉCLARATIVE

Il faut pour cela changer de méthode d'analyse du fonctionnement : au lieu de chercher à le décomposer en succession d'actions (séquence) pour arriver à un algorithme, nous devons essayer de définir ce que doit être à chaque instant la conduite du système, en fonction des états des

prédicats (relation état/action). On peut pour cela construire le tableau suivant où sont indiquées, pour chaque commande, les conditions de son déclenchement exprimées en fonction des prédicats.

Fonctions / Commandes	CHARGE?	DECHARGE?	HAUT?	BAS?	ATTENTE?
MONTE	V	(F)	F	V ou F	V ou F
DESCENDS	(F)	V	V ou F	F	V
CAISSE-D	F	V ou F	(F)	V	V
CAISSE-G	V ou F	F	V	(F)	V ou F

V, F signifient VRAI, FAUX. Les V et F entre parenthèses correspondent à des informations redondantes.

À partir de ce tableau il est alors possible d'écrire les règles (le conduite du système. Ces règles sont mises dans un ordre quelconque, sans chercher une chronologie : il ne s'agit pas ici de décrire la suite des actions à faire (séquence) mais de donner un ensemble de règles valables à chaque instant et déterminant l'action à effectuer.

Écrivons ces règles en Logo et mettons-les dans une procédure REGLES. Nous avons gardé l'ordre du tableau (qui n'est pas chronologique), mais toute autre succession conviendrait.

```
POUR REGLES
SI ET CHARGE? NON HAUT? [MONTE]
SI ET DECHARGE? ET NON BAS? ATTENTE? [DESCENDS]
SI ET NON CHARGE? ET BAS? ATTENTE? [CAISSE-D]
SI ET NON DECHARGE? HAUT? [CAISSE-G]
FIN
```

NB : l'utilisation des "ET" préfixés peut gêner ceux qui n'y sont pas habitués ; cette écriture correspond en français, à un "ET" entre chacune des conditions d'une règle.

Puis fabriquons un petit moteur d'inférence (il est très simple ici en raison des caractéristiques particulières des règles qui régissent de tels objets techniques)

```
POUR MOTEUR
REGLES
MOTEUR
FIN
```

Il suffit alors de lancer MOTEUR pour que le monte-charge se mette au travail. Si rien ne prévoit de renouveler les caisses au niveau

inférieur, il s'arrêtera de lui-même (le monte-charge, non la procédure MOTEUR).

Pour bien se convaincre de l'indépendance d'un système ainsi programmé vis à vis de la chronologie on peut arrêter le fonctionnement à tout moment (CNT C) et relancer MOTEUR : le monte-charge reprendra le travail sans aucun problème. Si nous profitons de l'arrêt pour changer la caisse de place et la remettre en bas, le monte-charge s'adaptera parfaitement.

4) CONCLUSION

Il nous semble que la comparaison de ces différentes façons de programmer un même objet technique, à partir des mêmes outils de base, présente quelques intérêts :

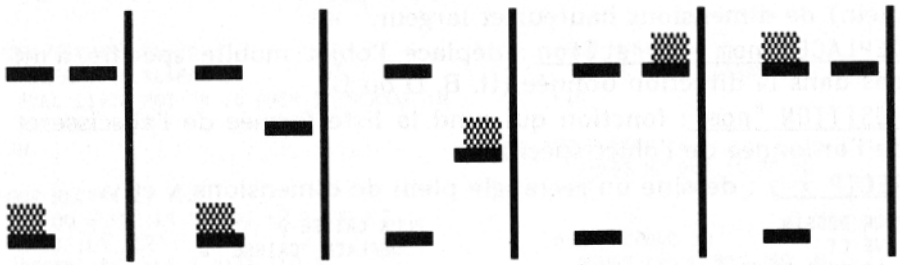
- elle illustre bien certains avantages de la programmation déclarative sur la programmation séquentielle (procédurale).
- elle montre deux approches possibles de l'étude d'un objet technique : par description algorithmique de la suite de ses actions (programmation 1 et 2) ou par définition des états qu'il peut prendre (programmation 3). Ces deux approches, utilisables pour beaucoup d'autres objets techniques, peuvent chacune donner lieu à des activités avec des élèves.

C. ORANGE.
Professeur à l'École Normale de la Manche
50200 Coutances

J. COURTAIS,
Coordinateur départemental des
activités informatiques (Manche)

ANNEXE 1 MONTE-CHARGE

Cahier des charges de macro-primitives de simulation



COMMANDES

DESSIN dessine à l'écran le monte-charge dans sa position initiale (plateau mobile vide, une charge attend sur un plateau fixe).

MONTE fait monter le plateau mobile d'un pas. S'il est chargé, fait également monter la caisse.

DESCENDS fait descendre le plateau mobile d'un pas. S'il est chargé, fait également descendre la caisse.

CAISSE-D fait bouger la caisse d'un pas vers la droite.

CAISSE-G fait bouger la caisse d'un pas vers la gauche.

FONCTIONS

CHARGE? fonction booléenne (=prédicat) ; rend VRAI si la caisse est en place sur le plateau mobile, FAUX sinon.

DECHARGE? prédicat ; rend VRAI si la caisse est en place sur un des deux plateaux fixes.

HAUT? prédicat ; rend VRAI si le plateau mobile est en haut, FAUX sinon.

BAS? prédicat ; rend VRAI si le plateau mobile est en bas, FAUX sinon.

ATTENTE? prédicat ; rend VRAI si la caisse est en bas, FAUX sinon.

ANNEXE 2 Codage en logo des **macro-primitives** du **monte-charge**

NB : Ce codage utilise les macro-primitives suivantes (annexe 3).

`OBJ "nom hauteur largeur` : définit et dessine à l'endroit de la tortue, et de couleur de la tortue, un objet mobile (rectangle plein) de dimensions hauteur et largeur.

`DEPLACE "nom "direction` : déplace l'objet mobile spécifié d'un pas dans la direction donnée (H, B, D ou G).

`POSITION "nom` : fonction qui rend la liste formée de l'abscisse et de l'ordonnée de l'objet spécifié.

`RECTP x y` : dessine un rectangle plein de dimensions x et y.

```
POUR DESSIN
   VE CT
   LC FPOS C1 0]
   BC RECTP 50 3
   LC FPOS [-17 0]
   BC RECTP 1 7
   LC FPOS [-17 38]
   BC RECTP 1 7
   LC FPOS C-8 38]
   BC FCC 0
   OBJ "PLATEAU 1 7
   LC FPOS [-17 21]
   FCC 1 BC
   OBJ "CAISSE 7 7
FIN
```

```
POUR CAISSE-D
   DEPLACE "CAISSE "D
FIN

POUR CAISSE-G
   DEPLACE "CAISSE "G
FIN
```

```
POUR DESCENDS
   SI CHARGE? [DEPLACE "PLATEAU "B DEPLACE "CAISSE "BI [DEPLACE
      "PLATEAU "B]
FIN
POUR MONTE
   SI CHARGE? [DEPLACE "CAISSE "H DEPLACE "PLATEAU "H] [DEPLACE
      "PLATEAU "H]
FIN
POUR CHARGE?
   SI PLG? PREM POSITION "CAISSE -9 [RENDS VRAI] [RENDS FAUX]
FIN
POUR DECHARGE?
```

```
   SI PLP? PREM POSITION "CAISSE -16 [RENDS VRAI] [RENDS FAUX]
FIN
POUR HAUT?
   SI PLG? DER POSITION "PLATEAU 37 [RENDS VRAI] [RENDS FAUX]
FIN
POUR BAS?
   SI PLP? DER POSITION "PLATEAU 1 [RENDS VRAI] [RENDS FAUX]
FIN
POUR ATTENTE?
   SI PLP? DER POSITION "CAISSE 3 [RENDS VRAI] [RENDS FAUX]
FIN
```

ANNEXE 3 : Codage des macro-primitives "mobiles"

```
POUR OBJ :N :L1 :L2
   DONNE MOT "*0 :N MP POS MP LISTE :L1 :L2 MP CC C]
   DONNE "*CAP CAP
   FCAP 0
   RECTP :L1 :L2 FCAP :*CAP
FIN
POUR DEPLACE :N :D
   *DEPART :N *CARAC :N
   EXEC LISTE MOT "* :D PREM SP *CARAC :N
   *RETOUR :N
FIN
POUR POSITION :O
   RENDS PREM *CARAC :0
FIN
POUR RECTP :L1 :L2
   SI OU PLP? :L1 1 PLP? L2 1 [STOP]
   RECT :L1 :L2
   RECTP DIFF :L1 1 DIFF :L2 1
FIN
POUR RECT :L1 :L2
   REPETE 2 [AV :L1 TD 90 AV :L2 TD 90]
FIN
POUR *G :L TD 90
   AV DER :L TD 180
   *ZOU *INV :L
   DONNE MOT "*0 :N *MODIF *CARAC :N [-1 0]
```

```
FIN
POUR *H :L *ZOU :L
   DONNE MOT "*0 :N *MODIF *CARAC :N [0 -1]
FIN

POUR *B :L
   AV PREM :L TD 90
   AV DER :L TD 90 *ZOU :L
   DONNE MOT "*0 :N *MODIF
FIN
POUR *D :L AV PREM :L TD 90
   *ZOU *INV :L
   DONNE MOT "*0 :N *MODIF *CARAC :N [1 0 ]
FIN
POUR *INV :L
   RENDS MP DER :L SD :L
FIN
POUR *CARAC :N
   RENDS CHOSE MOT "*0 :N
FIN
POUR *DEPART :N :L
   DONNE "*CAP CAP
   DONNE "*CC CC
   DONNE "*POS POS
   DONNE "*BC? BC? LC
   FCC DER :L
   FPOS PREM :L
   FCAP 0
FIN
POUR *RETOUR :N
   FPOS :*POS
   FCAP :*CAP
   FCC :*CC
   SI :*BC? [BC]
FIN
POUR *ZOU :L BC
   AV SOMME PREM :L 1 TD 90
   AV DER :L
   CARAC :N [0 -1]     TD 90
```

```
   AV SOMME PREM :L 1 TD 90
   FCC DIFF -1 CF AV DER :L
LC FIN
POUR *MODIF :L :M
   RENDS MP LISTE SOMME PREM :M PREM PREM :L SOMME DER :M DER
      PREM :L SP :L
FIN
```