Algorithmique au collège

Petite introduction à la programmation événementielle

IREM d'Aquitaine – Groupe Algorithmique

Les « paradigmes » de programmation

Un paradigme de programmation est un « style de programmation » qui précise de quelle façon le développeur (programmeur) doit mettre en œuvre sa solution (son programme).

Chaque paradigme de programmation peut être supporté par différents langages de programmation :

- Programmation séquentielle : Python, C, C++, Java, Pascal, Fortran...
- Programmation événementielle : Scratch, Visual Basic, Simula...
- Programmation orientée objet : Python, C++, Java, Smalltalk...
- Programmation concurrente : C++, Java, Go...
- Programmation fonctionnelle : Lisp, Caml, Ruby...
- Programmation logique : Prolog...
- etc.

La programmation séquentielle

Le paradigme de programmation séquentielle est le paradigme de programmation le plus « traditionnel » (parfois appelé impératif, ou procédural) :

Le programme s'exécute étape par étape, dans un ordre contrôlé par le programmeur :

- première instruction = début du programme
- enchaînement défini par les structures de contrôles utilisées :
  - enchaînement séquentiel (par défaut),
  - structures alternatives (du type si-alors-sinon)
  - structures répétitives (boucles pour et tant-que)
  - appel de sous-programmes (fonctions)
  - ...

La programmation séquentielle (un exemple)

Un programme Python affichant la liste des nombres premiers inférieurs à 100 :

```
# ce programme affiche la liste des nombres premiers inférieurs à 100
import math
for n in range(1,100):
    # initialisations
    racineDeN = int(math.sqrt(n))
    diviseur = 2
    # tant qu'on n'a pas trouvé de diviseur, on avance...
    while ((n % diviseur != 0) and (diviseur <= racineDeN)):
        diviseur = diviseur + 1
    # si diviseur est allé au-delà de racineDeN, N est premier
    if (diviseur > racineDeN):
        print(n)
```


La programmation événementielle

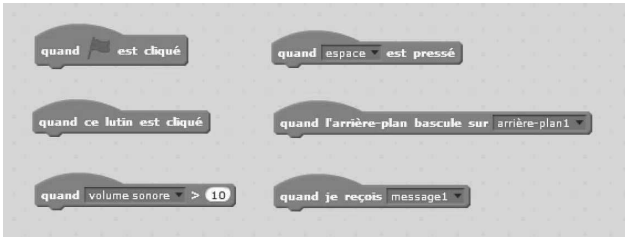
Le programme est constitué de « portions de code », qui seront exécutées en fonction de la survenue d'événements particuliers.

Exemples d'événements :

- lancement du programme,
- appui sur une touche clavier,
- action sur la souris : clic gauche, clic droit, double-clic, déplacement...
- message reçu (par une autre portion de code),
- ...

Le programmeur « se contente » d'écrire les portions de code associées aux événements. La gestion des événements (appel des portions associées) est entièrement automatisée...

Quelques événements Scratch



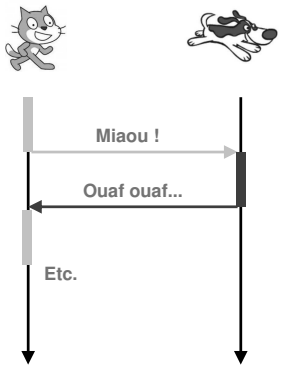
La portion de code associée est « emboîtée » sous l'événement correspondant...

Synchronisation par échange de messages

Outre les réponses aux événements déclenchés par l'utilisateur du programme (lancement, touches clavier, souris...), les lutins Scratch peuvent « réagir » (exécuter une portion de code) à la réception de messages... envoyés par d'autres lutins !

Cela va notamment permettre aux lutins de « synchroniser leurs actions »...

Illustration : petit dialogue entre chien et chat



Deux remarques :

- un lutin peut « continuer à travailler » après l'envoi d'un message...
- plusieurs lutins peuvent « se mettre à travailler » lors de la réception d'un même message, simultanément...

Merci de votre attention...