Premiers pas avec Rurple

Fiche professeur

▹ Piloter le robot

Les élèves commencent par prendre en main l'environnement.

- Le menu « Scène/Nouvelle... » permet de créer une nouvelle scène. ;
- Le menu « Scène/Nombre de billes... » permet de placer des billes dans la poche du robot.

Le robot **Rurple** obéit aux ordres élémentaires suivants :

- « avance » (touche A) : le robot avance d'une case devant lui ;
- « gauche » (touche G) : le robot tourne sur lui-même de 90° vers la gauche ;
- « depose » (touche D) : le robot dépose une bille sur la case où il se trouve ;
- « prends » (touche P) : le robot prends une bille sur la case où il se trouve.

▹ Programmer le robot

Après cette rapide prise en main le professeur expose aux élèves l'objectif de l'activité : faire en sorte que le robot suive un comportement précis préalablement défini. Ce comportement peut par exemple être : « Le robot fait le tour de la scène en déposant une bille au quatre coins. »

Après avoir piloté le robot avec les touches du clavier, l'élève rédige dans la fenêtre de gauche les ordres qu'il a utilisé. Il peut s'appuyer pour cela sur un petit document fourni par le professeur qui décrit le rôle des diverses instructions et les règles de syntaxes à respecter. De façon alternative, le professeur peut aussi fournir un exemple de programme que l'élève étudie et exécute pour comprendre de lui-même la sémantique et la syntaxe des instructions dont il a besoin. L'élève doit expérimenter ces deux approches car elles sont complémentaires dans la pratique quotidienne du programmeur.

L'élève vient d'écrire ainsi son premier programme. Il passe ainsi du rôle d'exécutant à celui de programmeur. Plutôt que d'accomplir lui même la tâche il décrit dans un texte la suite d'instructions à exécuter pour réaliser cette tâche. Programmer consiste à faire faire.

▹ Mettre au point un programme

Il convient de laisser l'élève se débrouiller seul. Le professeur encourage simplement l'élève à étudier la documentation, à expérimenter et à analyser les messages d'erreurs. Le professeur valorise aux yeux de l'élève cette phase de mise au point qui fait partie intégrante du travail d'écriture d'un programme.

Le robot ne commet aucune erreur. Si les choses ne se passent pas comme prévues cela signifie que l'erreur provient du texte. On dit qu'il contient un « bug ». Il est rare que l'on réussisse à écrire un programme sans bug du premier coup.

Si un passage du texte est incompréhensible pour le robot, celui-ci s'arrête en affichant un message d'erreur (en anglais) dans la zone d'affichage en bas à droite ou bien dans une boîte de dialogue. Le texte ne respecte pas les rèlges de grammaire du langage : une *erreur de syntaxe* a été commise, par exemple des parenthèses

ont été oubliées ou un mot inconnu du robot a été utilisé. Pas de panique, il faut alors tenter de déchiffrer le message d'erreur et examiner attentivement le texte pour trouver l'origine du problème. On corrige alors le texte, on réinitialise la scène et on relance le programme.

Le texte peut aussi être syntaxiquement correct et donc intelligible par le robot. Toutefois le comportement du robot n'est pas tout à fait ou bien pas du tout celui attendu. Là encore pas de panique. Le texte dit autre chose que ce que l'on a en tête. Il faut l'étudier pour découvrir ce qui ne va pas.

▹ Documenter un programme

Lorsque le programme fonctionne l'élève l'enregistre dans un fichier afin de le conserver et pouvoir ainsi le retrouver en cas de besoin. Pour le retrouver facilement plus tard parmi de multiples autres fichiers que l'élève va produire au cours de l'année, il est utile qu'il réfléchisse dès maintenant à une méthode de classement et à un système de nommage.

D'autre part, pour se souvenir plus tard de ce que fait exactement le programme il est très commode d'ajouter au texte quelques lignes explicatives. Par exemple :

```
"""Scène : Salle carrée 9x9. Le robot a quatre billes en poche.
           Il est en bas à gauche et regarde vers l'Est.
   Action : Le robot dépose ses quatre billes aux quatre coins de la salle.
"""
```

L'encadrement du texte par trois guillemets double (") permet de signaler à l'interpréteur qu'il ne s'agit pas d'instructions qu'il doit analyser et exécuter mais de *commentaires* qu'il doit ignorer. Si l'éditeur utilisé connait la syntaxe du langage **Python** le texte en commentaires apparaît dans une couleur différente.

En **Python** *les textes entre trois guillemets double en début de fichier ou en début de définition de fonctions ne sont pas totalement ignorés de l'interpréteur. Il s'agit de* docstring *c'est-à-dire de chaînes de caractères qui documentent le fichier ou la fonction et qui peuvent être lues et manipulées par des outils de documentation.*

Les vrais commentaires débutent par un #.

▹ Utiliser des procédures

Lorsque l'on écrit des programmes on est souvent amené à répéter plusieurs fois une même séquence d'instructions. C'est fastidieux, surtout que les erreurs qui ne manquent pas de se produire doivent être corrigées à chaque endroit.

Pour nous faciliter la vie, le langage dispose d'une syntaxe qui permet de regrouper une séquence et de la désigner par un mot unique.

Le professeur peut par exemple proposer à l'élève d'étudier et exécuter le programme ci-contre.
La séquence qui définit le mot **diagonale** est légèrement décalée sur la droite. On dit que le texte est « indenté ». On utilise la touche TAB du clavier pour indenter les instructions. Cette indentation est indispensable, c'est ce qui permet au robot d'identifier clairement où commencent et où s'arrêtent les instructions qui définissent le mot **diagonale**. La ligne vide n'est pas obligatoire pour l'interpréteur mais elle rend la séparation plus claire pour nous.
Dans ce programme, le mot **diagonale** est appelé une *procédure*.

```
def diagonale():
    depose()
    avance()
    gauche()
    avance()
    gauche()
    gauche()
    gauche()

diagonale()
diagonale()
diagonale()
diagonale()
```

Les procédures permettent de mieux structurer le programme ce qui facilite sa compréhension. En évitant d'écrire plusieurs fois la même chose, les procédures facilitent ausi la mise au point du programme. Enfin, le choix de noms significatifs pour les procédures contribue à la documentation du programme.

Le professeur propose à l'élève de reprendre son programme initial afin d'en créer une version plus courte qui utilise une ou plusieurs procédures.

▹ Lire l'état de l'environnement

Le professeur demande à l'élève d'expérimenter les instructions ci-dessous.

```
print("Je suis capable de percevoir mon environnement")
print("Je regarde vers le nord :", face_au_nord())
print("J'ai au moins une bille dans ma poche :", bille_en_poche())
print("Il y a au moins une bille au sol :", bille_au_sol())
print("Il y a un mur devant :", mur_devant())
print("Il y a un mur à droite :", mur_a_droite())
print("Il y a un mur à gauche :", mur_a_gauche())
```

Les *procédures* mur_devant(), bille_au_sol(), etc, ont toutes la particularité de *fournir une information* qui est soit True soit False. Le robot perçoit son environnement et sait à tout moment s'il y a un mur devant lui, s'il y a une bille au sol, etc.

À chaque instant l'environnement est dans un état bien précis : le robot est sur certaine case, il regarde dans une certaine direction, il dispose d'un certain nombre de billes dans sa poche, etc. Le langage dispose d'instructions qui permettent de changer l'état de cet environnement. L'élève découvre maintenant que le langage dispose également d'instructions permettant de lire l'état de l'environnement. Il va alors être possible d'adapter les instructions à exécuter en fonction de l'état dans lequel se trouve l'environnement. Il va être possible d'écrire des programmes un peu plus « intelligents ».

Remarque : L'existence de variables (absente dans cette activité) introduit un état interne qui peut être modifié et lu par la machine. La manipulation de variables va considérablement enrichir les possibilités d'expression du langage.

▹ Utiliser des boucles

Le professeur soumet un nouveau programme à l'élève. L'exécution pas à pas va lui permettre de comprendre l'effet du mot-clé « **while** ».

Il est alors invité à modifier une dernière fois son programme initial pour y intégrer cette nouvelle syntaxe.

Cette nouvelle syntaxe doit permettre à l'élève de créer un programme qui fonctionne correctement pour *n'importe quelle scène rectangulaire,* c'est-à-dire de dimensions quelconques.

```
while not bille_au_sol():
    avance()
while bille_au_sol():
    prends()
avance()
while bille_en_poche():
    depose()
avance()
```