16 – Programmation évènementielle : principe et application

Introduction

Programmation évènementielle ? Il y aurait donc plusieurs types de programmation? Plusieurs façons de programmer ?

Et bien oui, il en existe un grand nombre, on appel ça des **paradigmes de programmation**.

Un paradigme de programmation est une manière dont **les solutions aux problèmes doivent être formulées** dans un langage de programmation. C'est à dire que les solutions aux problèmes ont été trouvées, on choisi maintenant une façon, une philosophie de programmation pour transcrire efficacement ces solutions en lignes de code.
Ce choix revient au développeur ou à l'équipe de développement en fonction des caractéristiques de ses solutions.

Il existe beaucoup de paradigme de programmation :

- Programmation procédurale
- Programmation orientée objet
- Programmation descriptive
- …

Et donc celui qui nous intéresse plus particulièrement, la programmation évènementielle.

I- Le principe de la programmation évènementielle

a- Kesako et à quoi ça sert ?

Elle **s'oppose à la programmation procédurale** qui elle consiste à l'exécution successive d'instructions (en fonction des différents embranchements) écrites par le programmeur. La programmation évènementielle est définie par ses **réactions aux différents évènements** qui interviennent pendant l'exécution du programme. Les programmes écrient avec un paradigme de programmation procédurale **agissent** alors que ceux écrient avec un paradigme de programmation évènementielle **réagissent**.

Cette programmation est beaucoup utilisée depuis que les ordinateurs sont devenus assez puissants pour afficher des interfaces graphiques. Mais on peut tout à fait faire de l'évènementielle sans interface graphique. Par exemple, on peut écrire un petit programme qui reçoit un évènement venant de l'horloge interne toutes les 5 minutes. A la captation de cet évènement, le programme nettoie la RAM de notre ordinateur. On a donc un programme qui réagit à un évènement mais ne possède pas d'interface graphique.

Ce qui nous intéresse le plus ici, c'est d'observer les réactions aux évènements et donc nous allons créer des interfaces graphiques

b- Les évènements et leur nature

Un évènement est une action qui est reconnu par le programme. Ces actions peuvent être générées par l'utilisateur ou par le programme lui-même.

Différents types d'évènement :

Les évènements systèmes :

- **Le Timer**, on peut créer un timer dans notre programme qui génère un événement à intervalle de temps régulier ou à une certaine heure par exemple.
- Les messages venant d'autres programmes.

Les événements utilisateurs :

- **Mouse events** : la direction, si elle est en mouvement, si ou plusieurs boutons sont cliqués, si un bouton est maintenu pressé, si il n'est pas pressé, si la molette est touché et dans quelle direction, …
- **Keyboard events** : quelles touches sont pressées, des combinaisons de touches, des touches restées enfoncées, …
- **Touchscreen events :** pression forte ou légère, combien de doigts, le mouvement de ceux-ci, …
- **Windows Event :** création/destruction, ouverture/fermeture, sélection/dé-sélection, déplacement, changement de taille …

- **Device events :** présent notamment sur les Smartphones avec l'accéléromètre. Détection du mouvement, des rotations, si on secoue le téléphone, …

Ces événements sont détectés et gérés dans ce qu'on appelle une boucle d'évènement.

c- La boucle d'évènement

C'est une boucle « infini » qui va détecter les évènements, leur type et ainsi les traiter. On définit la plupart du temps une combinaison de touche, ou un bouton, qui quitte la boucle d'événement. Cette interruption de la boucle est souvent synonyme d'arrêt du programme (bouton croix et alt+F4).

C'est donc une boucle qui **attend activement**, c'est-à-dire qu'elle vérifie de façon répétée si une condition est vrai, cette condition c'est : « est-ce qu'un événement se produit ». Ce attente s'oppose à l'attente passive, qui elle ne vérifie pas de façon répétée cette condition et donc n'utilise pas de ressource processeur, qui peut ainsi être utilisée par un autre processus.

Dès qu'un évènement est détecté par la boucle, en fonction du type d'événement, cela va déclencher une action, appeler une fonction, …

d- Cas particuliers

Qu'est ce qu'on fait quand on attend un événement et qu'il n'arrive jamais ?

On peut mettre un timeout, c'est-à-dire qu'au bout d'un certain temps, on va arrêter d'attendre et continuer la suite du programme.

Qu'est-ce qu'on fait quand deux évènements arrivent en même temps ?

1ère solutions : On ne gère pas les évènements concurrents, un seul est traités, souvent celui qui arrive en premier dans la boucle d'évènements.

2ème solutions : On établie une liste de priorité entre les évènements et on les traite l'un après l'autre.

3ème solutions : On traite les évènements en parallèle. C'est à dire qu'on va créer un processus pour chaque fonction de traitement d'un événement, celles-ci vont s'exécuter en même temps. Les notions de processus et de parallélisme ne sont pas au programme mais c'est bien de savoir qu'elles existent.

II- En pratique

Programmer des interfaces graphiques réactives au événements, il existe un tas de bibliothèque de haut-niveau tel quel Qt Software, Windows API, Node.js, …
Elle permette notamment de dessiner les interfaces graphiquement, pas besoin de placer en ligne de code les différents éléments, en risquant de se tromper d'un pixel et que les éléments se recouvrent, ou que les proportions soient mauvaises,…

De notre côté, on va observer cette programmation évènementielle à travers des petites interfaces graphiques en python et sur scratch.

a- Programmer en python

Il existe une bibliothèque en python qui permet de créer des interfaces graphiques et d'utiliser les évènements, il s'agit de *Tkinter*. Cette librairie est assez bas niveau (même si la boucle d'événement est cachée) et est un peu compliquée à utiliser sans l'aide d'internet ou de la doc. Pour illustrer mes propos nous allons donc utiliser une autre bibliothèque de python, *Turtle*. Elle utilise la libraire Tkinter donc les méthodes liées aux *events* ne diffèrent pas tant que ça, mais cette bibliothèque est beaucoup plus simple et rapide à utiliser.

Comment utiliser *Turtle* ?

Dans un premier temps on importe la bibliothèque :

```
from turtle import *
```

Ensuite on appelle on fonction qui va demander au programme de prendre en compte les évènements, sans cette fonction, ils ne seront pas pris en compte.

```
listen()
```

Ensuite nous allons définir les évènements que le programme doit traiter :

```
type_de_l_evenement(action, autres arguments)
```

Quelques types d'évènement avec *Turtle* :

```
//clique sur la turtle, btn -> quel bouton ?
onclick(fun, btn=1, add=None)
//relachement du click, btn -> quel bouton ?
onrelease(fun, btn=1, add=None)
//clique sur l'écran
onscreenclick(fun, btn=1, add=None)
//drag and drop
ondrag(fun, btn=1, add=None)
//touche clavier
onkey(fun, key)
//timer qui appel fun après t millisecondes
ontimer(fun, t=0)
```

Pour finir, on lance la boucle d'événement :

```
mainloop()
```

Si avec *Tkinter* on voulait gérer l'évènement : « appuyer sur la flèche du haut » :

```
fenetre = Tk.Tk()
fenetre.bind("<Up>", fonction)
```

turtle.py *turtle_tkinter.py*

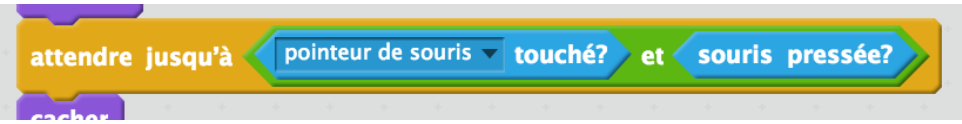
b- Programmer sur Scratch

Scratch permet très facilement de créer des interactions entre le programme et l'utilisateur mais aussi entre les différents éléments de celui-ci.

Déplacer un lutin en appuyant sur une touche (keyboard Event):

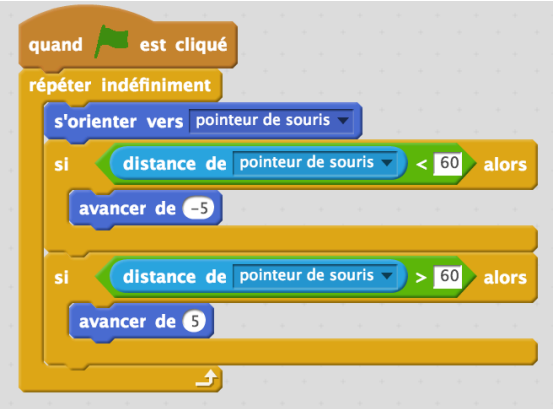
Test répété de la condition, le « répéter indéfiniment » correspond à la boucle d'événement, on se situe dans une attente active.

Cliques souris pour appuyer sur un lutin faisant office de bouton (MouseClick Event):



Attente passive de l'événement, on ne poursuit pas ce script tant qu'on ne clique pas sur le bouton

Actions au mouvement de souris (MouseMove Event) :



Le lutin s'oriente toujours vers la souris et avance ou recule.

Evènements liés au chronomètre (Timer Event) :

Le script reprendra lorsque le chronomètre aura dépassé la valeur choisie aléatoirement (entre 3 et 10 secondes).

Messages envoyés entre lutins :

Les messages permettent de déclencher des scripts dans un autre lutin.

Scratch permet de faire du parallélisme, plusieurs script peuvent s'exécuter en même temps au sein du même lutin. Plusieurs lutins peuvent aussi exécuter plusieurs script en même temps.

Jeu : https://scratch.mit.edu/projects/183975030/
Ouch : https://scratch.mit.edu/projects/190951119/
Follow me : https://scratch.mit.edu/projects/190521092/
Somme entiers impairs : https://scratch.mit.edu/projects/189919529/
Résolution : https://scratch.mit.edu/projects/190005834/
PGCD : https://scratch.mit.edu/projects/190016570/

Conclusion

Très facile des créer des activités autour de la programmation avec des élèves de collège grâce à scratch. En revanche, dès qu'on arrive au lycée et qu'on veut quitter scratch pour s'orienter vers de la programmation en python cela devient plus compliqué. En effet la bibliothèque de base (*Tkinter*) est un peu compliquée à manipuler, elle requiert un peu d'expérience et de travail.