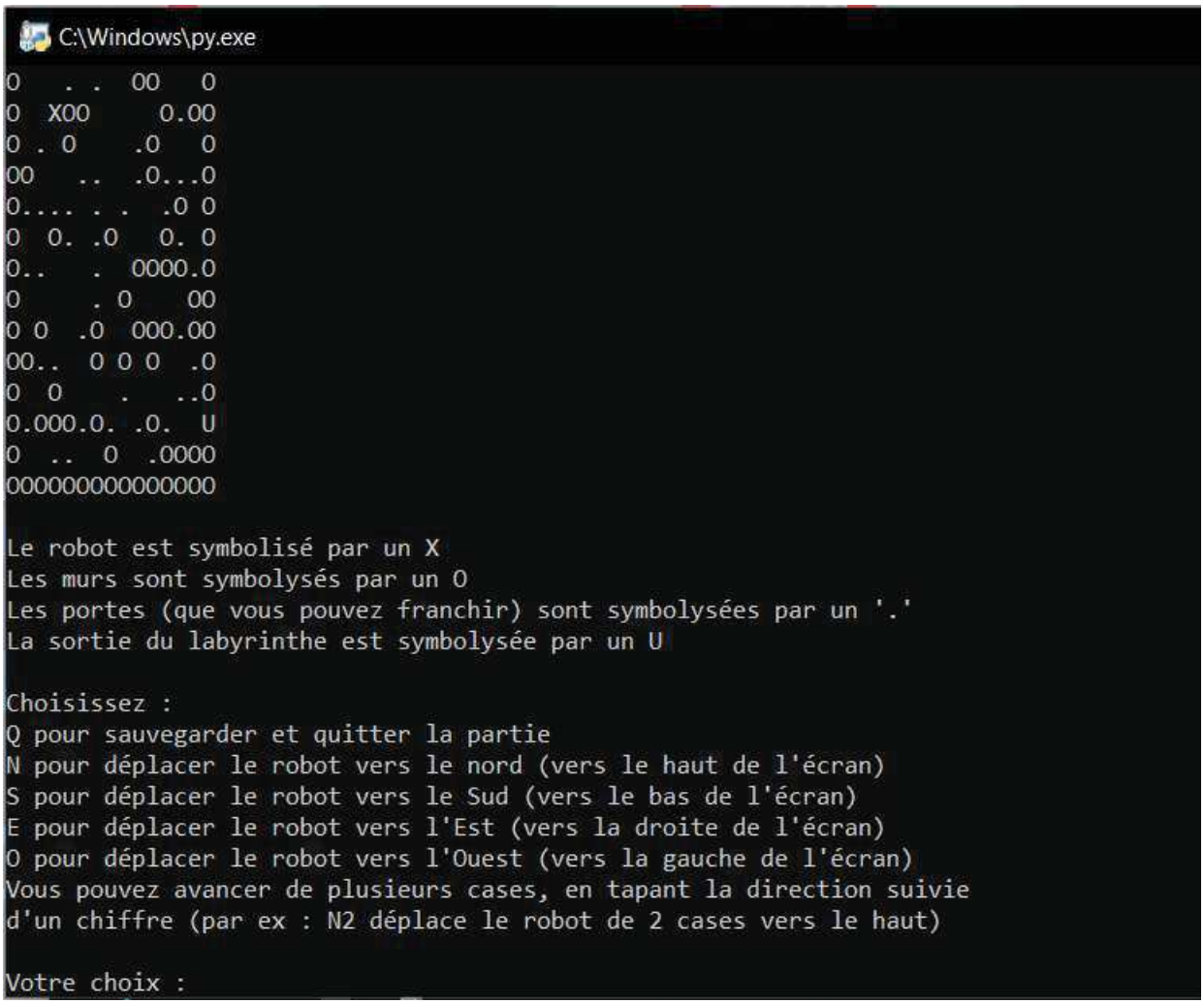
Fiche de Synthèse : Roboc *Juin 2019*

Développement :
C'est l'histoire d'un petit robot, qui s'appelle X, qui doit sortir du labyrinthe dans lequel il est perdu.
Il s'agit d'un petit jeu développé pour un mode console, les graphismes sont donc un peu rudimentaires.



Cadre :
Il s'agit d'un TP proposé dans le cadre du cours "Apprenez à programmer en Python ", par Vincent Le Goff.

Support :
Développé en Python 3 avec l'idle de Python.

Contraintes :
Le jeu doit permettre soit de continuer une partie en cours, soit de jouer avec un nouveau labyrinthe.
Les labyrinthes doivent être proposés en mode texte, c'est à dire en utilisant seulement les caractères " O " pour symboliser les murs, et " . " pour symbolier des portes, ceci afin de permettre à l'utilisateur de créer de nouvelles cartes à sa guise.

Amélioration incluse :
J'ai rajouté une possibilité : permettre à l'utilisateur de jouer avec un labyrinthe généré aléatoirement. Ainsi, le jeu reste intéressant (y compris pour le développeur, qui teste le jeu encore et encore pour s'assurer que tout fonctionne comme attendu).

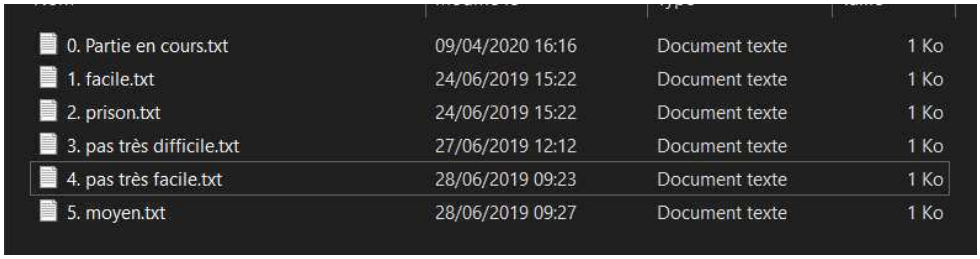
Description détaillée du développement

J’ai suivi les étapes suivantes pour le développement:
- la création des cartes, en mode texte
- développement de la classe Carte
- création de la classe Labyrinthe
- le code principal du jeu

1. La création des cartes

Une carte “ facile.txt ” et “ prison.txt ” nous étaient fournies à titre d’exemple, j’en ait rajouté 3.
Exemple, une carte “ pas très facile ” :

```
000000000000000
0.X0. 0. .00.00
00 .0 . .   0 0
0 ....0.  0.0 0
000. .    0 .00
0 .  0.... 0 .0
0000  0  .0 .00
0  000. .  ...0
0 ..   0   .0 0
0 .00 0   .0  0
00...0.0 000 00
0 .  .0.00 0. 0
000 00  0.    U
0 000 00 0. 000
000000000000000
```



X est donc le petit robot.
U représente la sortie du labyrinthe.
On ne peut pas franchir les murs O
J’ai ajouté des numéros aux noms de mes cartes, pour que la carte soit plus facile à sélectionner pour l’utilisateur.

2. Développement de la classe Carte

Pour manipuler, dans le développement, ces cartes, j’utilise une liste de listes.
Donc pour charger une carte existante, je lis la carte en format texte, j’ajoute une liste pour chaque ligne, et je place toutes ces listes/lignes dans une liste labyrinthe.

```
    def chargement(self) :
        """Récupère la carte choisie par l'utilisateur, et la transfère dans
une liste des lignes de la carte"""
        with open(self.path +'/'+ self.nom,'r') as fichier :
            self.labyrinthe = []
            while 1 :
                ligne = fichier.readline()
                if ligne == '' :
                    break
                else :
                    ligne = list(ligne)
                self.labyrinthe.append(ligne)
```

Dans cette classe, on trouve aussi les méthodes suivantes :
- __repr__(self) : qui est une redéfinition de la méthode spéciale __repr__ afin que la fonction print() affiche la carte
- enregistrement(self) : qui enregistre la carte “ en cours ” dans le fichier “ 0. Partie en cours “ : en effet, le labyrinthe est enregistré dans ce fichier à chaque mouvement du robot, pour pouvoir arrêter la partie à n’importe quel moment et la retrouver la fois suivante
- chargement_en_court(self) : méthode qui permet de démarrer la partie sur la carte “ 0. Partie en cours ”
- affiche_start(self) : méthode qui permet de charger l’une des cartes proposées
- et enfin la méthode la plus intéressante, la méthode generation_aleatoire(self), qui génère automatiquement un labyrinthe aléatoire.

Pour cette dernière méthode, je commence par générer avec random() 15 lignes aléatoires, puis ensuite je “ ferme ” les bords du labyrinthe avec des murs.
Histoire de ne pas risquer que le robot apparaisse trop près de la sortie (parfois le hasard ne fait pas bien les choses), je le positionne toujours au même endroit, vers la partie en haut à gauche du labyrinthe, et la sortie en bas à droite.

```
    def generation_aleatoire(self) :
        """Génère automatiquement des cartes aléatoires, sans garantir que le
labyrinthe sera 'faisable'"""
        elts_laby = [' ','O','.',' ']
        self.labyrinthe = []

        # création de 15 listes de symboles aléatoires
        for _ in range(15):
            line = []
            for _ in range(15):
                line.append(choice(elts_laby))
            self.labyrinthe.append(line)

        for ligne in self.labyrinthe :
            ligne.append('\n')

        # ensuite remplacement des symboles entourant le labyrinthe par
        # des 'o' pour qu'il soit "fermé"
        self.labyrinthe[0] = ['O']*15
        self.labyrinthe[0].append('\n')

        self.labyrinthe[14] = ['O']*15
        self.labyrinthe[14].append('\n')

        i = 0
        while i <= 14 :
            self.labyrinthe[i][14] = 'O'
            self.labyrinthe[i][0] = 'O'
            i += 1

        # et le robot + la sortie
        self.labyrinthe[12][14] = 'U'
        self.labyrinthe[2][3] = "X"
```

Enfin, c'est le constructeur de cette classe qui a la charge d'initialiser le jeu. En voici un extrait :

```
def __init__(self):
    """Constructeur de la classe, et initialisation du jeu"""

    self.path = os.path.abspath("cartes")
    liste_cartes = os.listdir(self.path)
    liste_cartes.sort()
    maxi = len(liste_cartes)-1

    for cartes in liste_cartes :
        print(cartes)

    choix_a = input("si vous voulez l'une de ces cartes, tapez 'C'  \n"\
                    "si vous préférez une carte générée aléatoirement, tapez 'A'\n")

    if choix_a == 'C' or choix_a =='c' :
        choix = input('Choisissez une des cartes ci-dessus, en tapant un nombre entre 0 et ' + str(maxi) + " : ")
        while 1 :
            try :
                choix = int(choix)
                assert choix >= 0 and choix <= maxi
                break

            except :
                choix = input("Veuillez taper un nombre entre 0 et " + str(maxi) + " : ")

        self.nom = liste_cartes[choix]
        print(self.nom)
        self.affiche_start()
        self.chargement()
        self.enregistrement()

    elif choix_a == 'A' or choix_a == 'a' :
```

3. Création de la classe Labyrinthe

C'est une classe fille, qui hérite de Carte(). La classe Labyrinthe gère les déplacement du robot du joueur dans le labyrinthe.
A chaque déplacement, la carte d'affiche de nouveau avec le nouvelle position du robot.
Dans le constructeur, on trouve donc celui de la classe mère Carte, ainsi qu'une variable pour le robot. C'est important cette variable, pour déplacer le robot il faut qu'on sache où il est est, et donc qu'on soit capable de le trouver.

```
    def __init__(self):
        """Constructeur de la classe, composé du constructeur de la classe
mère "Carte" et du robot"""
        Carte.__init__(self)

        self.robot = "X"
```

Ensuite, la classe contient " seulement " les méthodes qui correspondent au déplacement possibles du robot c'est à dire :
- depl_Nord(self, commande)
- depl_Sud(self, commande)
- depl_Est(self, commande)
- depl_Ouest(self, commande)

Pour chaque commande du joueur : on commence par identifier la position du robot dans le labyrinthe (position_robot), on contrôle si le joueur veut avancer d'une ou plusieurs cases (pas), on vérifie qu'il n'y a pas de murs sur le chemin que veut emprunter le robot, on regarde si le joueur a gagné en plaçant son robot sur la sortie, ou on place le robot à l'endroit désigné.

Par exemple, pour un déplacement vers le Nord, ça ressemble à ceci :

```
    def depl_Nord(self,commande) :
        """Déplacement du robot vers le Nord ; détermine si le robot va dans
un mur, gagne la sortie, ou fait seulement un déplacement de une ou plusieurs
cases vers le haut selon le choix de l'utilisateur"""

        for ligne in self.labyrinthe :
            if self.robot in ligne :
                num_ligne = self.labyrinthe.index(ligne)
                position_robot = self.labyrinthe[num_ligne].index(self.robot)

        # i pour une incrémentation, 'pas' c'est le nombre de cases de
        # déplacement du robot
        i,pas = 1,0

        if len(commande) == 1 :
            pas = 1
        else :
            pas = int(commande[1:])

        # je crée une liste contenant tout ce qu'il y a comme caractères sur
        # le chemin du robot, pour pouvoir vérifier ensuite s'il y a un
        # obstacle
        chemin = []
        while i <= pas and num_ligne - i >= 0 :
                chemin.append(self.labyrinthe[num_ligne - i][position_robot])
                i += 1


        if 'O' in chemin :
            print('Aïe un mur ! Déplacement impossible')

        elif num_ligne <  0 :
            print("Oups pas si vite, c'est trop loin !")
            print("Il faut mettre le robot SUR la sortie :)")

        elif self.labyrinthe[num_ligne - pas][position_robot] == 'U' :
            print("Wouahou ! Enfin libre !")
            self.labyrinthe[num_ligne - pas][position_robot] = self.robot
            self.labyrinthe[num_ligne][position_robot] = ' '
            self.enregistrement()
            print(self)
            print("C'est GAGNE !!!")
            sys.exit()

        else :
            self.labyrinthe[num_ligne - pas][position_robot] = self.robot
            self.labyrinthe[num_ligne][position_robot] = ' '
```

Les méthodes pour déplacer le robot dans les autres directions sont développées sur le même principe.

4. Le code principal du jeu

Maintenant que nous avons de jolies cartes, et un Labyrinthe qui les exploite pour déplacer le robot…..il ne reste plus qu'à utiliser tout ce travail :

```
import os

from carte import*
from labyrinthe import *

print('Bienvenue dans Roboc, le plus sympa des labyrinthes')
print('Voici les cartes proposées par défaut : ')

jeu = Labyrinthe()

print("Le robot est symbolisé par un X\n"\
      'Les murs sont symbolysés par un O\n'\
      "Les portes (que vous pouvez franchir) sont symbolysées par un '.'\n"\
      "La sortie du labyrinthe est symbolysée par un U\n")

commande = ''

while 1 :

    print('Choisissez :\n'\
      "Q pour sauvegarder et quitter la partie\n"\
      "N pour déplacer le robot vers le nord (vers le haut de l'écran)\n"\
      "S pour déplacer le robot vers le Sud (vers le bas de l'écran)\n"\
      "E pour déplacer le robot vers l'Est (vers la droite de l'écran)\n"\
      "O pour déplacer le robot vers l'Ouest (vers la gauche de l'écran)\n"\
      "Vous pouvez avancer de plusieurs cases, en tapant la direction suivie\n"\
      "d'un chiffre (par ex : N2 déplace le robot de 2 cases vers le haut)\n")

    commande = input('Votre choix : ')

    if commande == 'Q' or commande == 'q' :
        break

    elif commande[0] == 'N' or commande[0] == 'n' :
        jeu.chargement_en_court()
        jeu.depl_Nord(commande)
        jeu.enregistrement()
        print(jeu)

    elif commande[0] == 'S' or commande[0] == 's' :
        jeu.chargement_en_court()
        jeu.depl_Sud(commande)
        jeu.enregistrement()
        print(jeu)

    elif commande[0] == 'E' or commande[0] == 'e' :
```

Exemple du résultat d’une fin de partie exécutée avec l’idle de Python :

```
Votre choix : n1
Aïe un mur ! Déplacement impossible
OOOOOOOOOOOOOOO
O. . .OOO.  .OO
O .    O OO. .O
O OOOO  .  O. O
OO.   O O   O O
O . OOO O  O OO
O   OOO . .  OO
OO   OO  O .. O
O.OO..O.   .O O
O  . O.  OO O O
OO  .   OOO ..O
O OO ....O .O O
O  .O. ..O .X.U
O    .     O  O
OOOOOOOOOOOOOOO

Choisissez :
Q pour sauvegarder et quitter la partie
N pour déplacer le robot vers le nord (vers le haut de l'écran)
S pour déplacer le robot vers le Sud (vers le bas de l'écran)
E pour déplacer le robot vers l'Est (vers la droite de l'écran)
O pour déplacer le robot vers l'Ouest (vers la gauche de l'écran)
Vous pouvez avancer de plusieurs cases, en tapant la direction suivie
d'un chiffre (par ex : N2 déplace le robot de 2 cases vers le haut)

Votre choix : e2
Wouahou ! Enfin libre !
OOOOOOOOOOOOOOO
O. . .OOO.  .OO
O .    O OO. .O
O OOOO  .  O. O
OO.   O O   O O
O . OOO O  O OO
O   OOO . .  OO
OO   OO  O .. O
O.OO..O.   .O O
O  . O.  OO O O
OO  .   OOO ..O
O OO ....O .O O
O  .O. ..O . .X
O    .     O  O
OOOOOOOOOOOOOOO

C'est GAGNE !!!
```