Jean-Marie Guyader
jean-marie.guyader@isen-ouest.yncrea.fr
Bureau 330

Année 2020-2021

Projet CENT 2 – *Intelligence artificielle*

Notion étudiée

L'intelligence artificielle est l'ensemble des théories et des techniques mises en œuvre en vue de réaliser des machines capables de simuler l'intelligence.

Objectif

L'objectif de ce projet est de développer une intelligence artificielle permettant de trouver un chemin menant à la sortie d'un labyrinthe. On proposera un programme de visualisation en temps réel de la résolution du labyrinthe (Figure 1) en utilisant le module Python `turtle`.

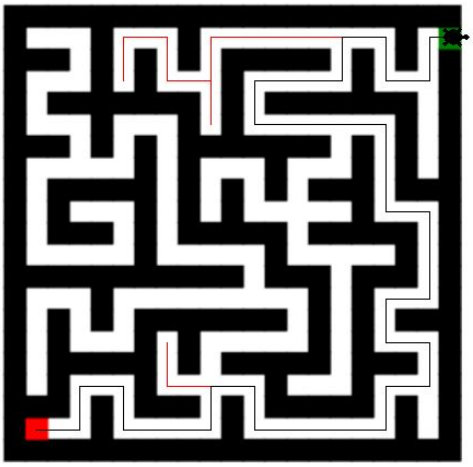

Figure 1 : exemple de labyrinthe. L'entrée du labyrinthe est représentée par la case rouge et sa sortie par la case verte. Le trait fin noir et rouge représente un parcours proposé par l'intelligence artificielle.

1. Le module `turtle`

Le module Python `turtle` est un module graphique permettant, entre autres fonctionnalités, de charger des images, de tracer des lignes, et de faire évoluer des curseurs. Dans cette partie, on fournit un condensé de la documentation complète du module `turtle`, disponible à l'adresse suivante : https://docs.python.org/2/library/turtle.html.

1.1. Importation du module

Le module `turtle` s'importe simplement en utilisant la commande : `import turtle`. Comme pour les autres modules Python, les différentes fonctions du module `turtle` s'appellent en faisant précéder leur intitulé par le mot-clé `turtle`.

1.2. Configuration de la fenêtre graphique

Le module `turtle` permet de dessiner des éléments graphiques à l'intérieur d'une fenêtre. Cette dernière est paramétrable. En particulier, on peut définir sa taille en utilisant la fonction `setup(...)`, son image de fond avec la fonction `bgpic(...)`, ou encore définir son titre avec la fonction `title(...)`.

Exemple :

```
1 import turtle
2
3 turtle.setup(900, 900)
4 turtle.bgpic('backgroundImage.png')
5 turtle.title('Titre de la fenêtre')
```

1.3. Dessin

Pou dessiner, le module `turtle` utilise un curseur qu'il est possible de lever, de poser et de déplacer. Deux stratégies principales sont possibles :

- déplacer le curseur vers une position précise. Par exemple :

```
1 turtle.down()
2 turtle.goto(10, 20) # x, y
3 turtle.up()
```

- déplacer le curseur en lui donnant une série de directions à suivre. Par exemple :

```
1 turtle.down()
2 turtle.forward(100) # 100 pixels
3 turtle.left(90) # 90 degrés
4 turtle.backward(50) # 50 pixels
5 turtle.right(45) # 45 degrés
6 turtle.up()
```

1.4. Choix de la couleur

La couleur du tracé peut être modifiée en utilisant une commande du type :

```
1 turtle.pencolor('red')
```

Le choix des couleurs se fait parmi les couleurs de base : `yellow`, `green`, etc.

1.5. Pointeurs

Différents types de pointeurs existent :

- `classic` : curseur classique en forme de petite flèche
- `arrow` : curseur en forme de flèche épaisse
- `circle` : curseur circulaire
- `square` : curseur carré
- `triangle` : curseur triangulaire
- `turtle` : curseur en forme de tortue
- `blank` : aucun curseur

Le type de curseur se modifie de la manière suivante :

```
1 turtle.shape('arrow')
```

1.6. Vitesse

La réalisation du tracé peut être fixée aux vitesses suivantes : `slowest`, `slow`, `normal`, `fast`, `fastest` :

```
1 turtle.speed('slowest')
```

1.7. Remplissage

Il peut parfois être nécessaire de remplir l'intérieur de la forme géométrique que l'on vient de tracer. À cet effet, on utilise la syntaxe suivante :

```
1 # Dessine un carré vert.
2 turtle.fillcolor('green')
3 turtle.begin_fill()
4 turtle.down()
5 for i in range(0, 4):
6   turtle.forward(10)
7   turtle.left(90)
8 turtle.up()
9 turtle.end_fill()
```

1.8. Fin de tracé et mise en pause

Pour laisser le dessin affiché tant que l'utilisateur n'a pas fermé la fenêtre ou cliqué à l'intérieur de celle-ci, il est possible d'utiliser la commande suivante :

```
1 turtle.exitonclick()
```

2. Cahier des charges

2.1. Partie 1 : recherche de la sortie d'un labyrinthe

Le but de cette partie est de développer une intelligence artificielle permettant de trouver et de représenter graphiquement le chemin (ou un des chemins possibles) menant de l'entrée d'un labyrinthe à sa sortie. Les labyrinthes considérés seront stockés sous la forme de fichiers texte (*.txt) de la forme suivante :

```
0000000
0102220
0202020
0202020
0202020
0222023
0000000
```

Les chiffres représentent respectivement :

- 0 : un mur
- 1 : case départ
- 2 : case libre de mouvement
- 3 : case sortie.

Fichiers fournis

Trois ensembles de fichiers labyrinthes sont fournis dans le cadre de ce projet. Chaque labyrinthe X est accompagné de deux fichiers :

- le fichier de description du labyrinthe : *MazeX.txt*
- une image au format .png constituant la représentation du labyrinthe sous forme d'image : *MazeX.png*

Contraintes à respecter

- Le fichier de description du labyrinthe sera chargé et stocké sous la forme d'une liste de listes (ou d'un tableau bi-dimensionnel d'entiers)
- Afin de trouver la sortie du labyrinthe, l'intelligence artificielle devra faire en sorte de vérifier si l'une des quatre cases qui entourent le curseur à un instant t donné (nord, sud, est, ouest) est accessible. Si tel est le cas, le curseur sera déplacé sur une des cases libres et on remplira la case précédente avec un `-1` afin de la définir comme déjà visitée.
- Afin que le curseur ne reste pas bloqué dans une impasse, on mémorisera les cases déjà visitées dans une liste `listPath`. Ainsi, si plus aucune case n'est accessible, on reviendra à une position antérieure jusqu'à en trouver une qui le soit.
- Afin de visualiser l'intelligence artificielle, il sera fait usage du module `turtle`. En particulier, on affichera l'image du labyrinthe à laquelle viendra s'ajouter le dessin du chemin parcouru par l'intelligence artificielle, ainsi que l'évolution du curseur au cours du temps. Dans les images .png considérées, chaque case du labyrinthe est représentée par un carré de 20×20 pixels.
- On demandera le nom du labyrinthe à charger au début du programme.
- On modifiera la couleur du trajet suivi par le curseur en cas de découverte d'une impasse (cf. Figure 1 : le curseur est de couleur noire en temps normal et devient rouge lorsqu'une impasse a été découverte).
- On suivra l'algorithme présenté à la Figure 2 pour faire évoluer la position du curseur au cours du temps.

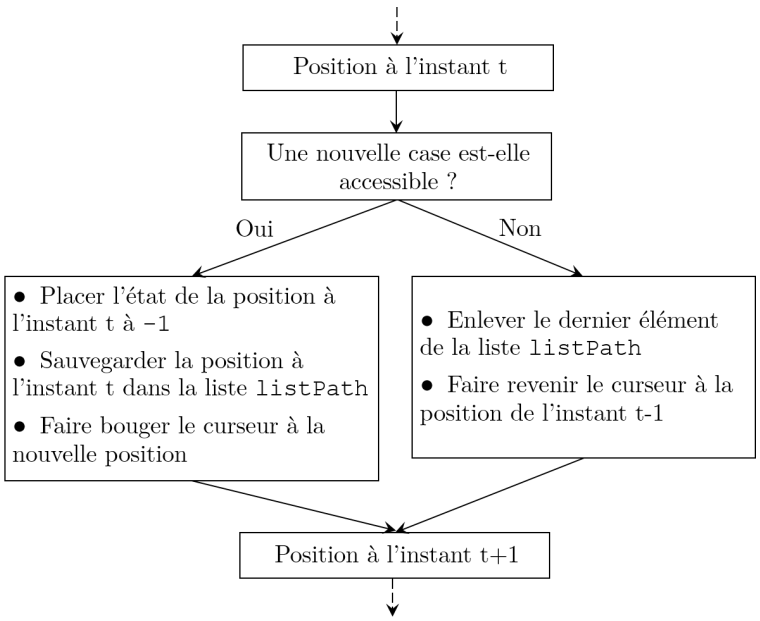

Figure 2 : algorithme proposé pour l'évolution du curseur au cours du temps.

2.2. Partie 2 : génération de la carte du labyrinthe à partir de son fichier texte uniquement

L'objectif de cette seconde partie est de mettre au point un programme Python qui puisse, à partir d'un fichier de description du labyrinthe (*MazeX.txt*), procéder :

- à l'affichage graphique de la grille du labyrinthe en utilisant le module `turtle`
- puis à la résolution du labyrinthe sur la fenêtre ainsi créée en utlisant l'algorithme développé à la partie 1.

2.3. Partie 3 : génération d'une carte de labyrinthe

Dans cette partie, on demande de développer un algorithme permettant de générer aléatoirement un fichier de description de labyrinthe (*MazeX.txt*) à partir d'une taille (e.g. 5 cases, 21 cases, etc.). Les fichiers de description générés seront carrés. On utilisera ensuite les codes élaborés à la partie 2 pour afficher l'image du labyrinthe ainsi constitué, puis les codes élaborés à la partie 1 pour procéder à la résolution du labyrinthe.

3. Rapport scientifique

Le projet devra déboucher sur la rédaction d'un *rapport scientifique* d'une dizaine de pages. Ce rapport comportera notamment une introduction, une description détaillée de la structure du programme et des solutions envisagées, un cahier de tests présentant le bon fonctionnement du programme et sa cohérence vis-à-vis du cahier des charges, et une conclusion.

Ce rapport, fourni au format PDF, ainsi que les codes développés au cours de ce projet seront transmis sous la forme d'une archive .zip à l'adresse email suivante : `jean-marie.guyader@isen-ouest.yncrea.fr`.