UNIVERSITÉ GRENOBLE ALPES

UFR IM²AG

DÉPARTEMENT LICENCE SCIENCES ET TECHNOLOGIE

LICENCE SCIENCES & TECHNOLOGIES, 1re ANNÉE

UE INF101, INF104

—

PROJET : EXPLORATION DE LABYRINTHE AVEC TURTLE

Responsables d'UE : Carole Adam et François Puitg
carole.adam@univ-grenoble-alpes.fr francois.puitg@univ-grenoble-alpes.fr

2022-2023
rendu : semaine du 12 décembre 2022

On s'intéresse ici à programmer la tortue du module `turtle` pour explorer des labyrinthes. Nous allons procéder en plusieurs étapes :

1. Lecture d'une configuration de labyrinthe dans un fichier (fonction fournie) et affichage graphique de ce labyrinthe
2. Guidage manuel de la tortue dans le labyrinthe
3. Exploration automatique de labyrinthes simples
4. Bonus facile : ajouter différents types d'objets dans le labyrinthe
5. Bonus : exploration automatique de labyrinthes plus complexes (boucles, sorties multiples...)
6. Bonus : course entre plusieurs tortues dans le même labyrinthe

Première partie

Exportation et affichage de labyrinthes

1 Travail préparatoire

Récupérer l'archive `projet-laby.zip`, la décompresser pour extraire son contenu.

- Le fichier fourni `lireLaby.py` contient une fonction `labyFromFile(fn)`. Cette fonction prend en paramètre le nom du fichier contenant la description du labyrinthe, et renvoie les éléments suivants :
  1. le labyrinthe lu : une liste à 2 dimensions ;
  2. son entrée : liste des 2 coordonnées ligne et colonne ;
  3. sa sortie : idem.
- Quelques fichiers décrivant des labyrinthes sont fournis également.

```
# Lire un labyrinthe dans un fichier et le renvoyer au format liste
# Read a maze from a file and return it as a list
def labyFromFile(fn):
        f = open(fn)
        laby = []
        indline = 0
        for fileline in f:
                labyline = []
                inditem = 0
                for item in fileline:
                        # empty cell / case vide
                        if item == ".":
                                labyline.append(0)
                        # wall / mur
                        elif item == "#":
                                labyline.append(1)
                        # entrance / entree
                        elif item == "x":
                                labyline.append(0)
                                mazeIn = [indline,inditem]
                        # exit / sortie
                        elif item == "X":
                                labyline.append(0)
                                mazeOut = [indline,inditem]
                        # discard "\n" char at the end of each line
                        inditem += 1
                laby.append(labyline)
                indline += 1
        f.close()
        return laby, mazeIn, mazeOut
```

1. Écrire un programme principal qui demande à l'utilisateur le nom d'un fichier, puis utilise la fonction `labyFromFile` pour y lire le labyrinthe correspondant. La fonction affiche de manière lisible la liste représentant le labyrinthe (sous forme d'une matrice), ainsi que les coordonnées de l'entrée et de la sortie.
2. Tester votre programme avec les labyrinthes fournis pour vérifier que vous comprenez bien le format de description des labyrinthes, et des variables renvoyées par la fonction de lecture.
3. **Important :** On choisit de conserver toutes les informations utiles au jeu dans un dictionnaire *dicoJeu*. Pour commencer, initialiser ce dictionnaire à partir d'un

fichier en y stockant plusieurs clés associées à la liste labyrinthe, les coordonnées de l'entrée (liste de 2 coordonnées), et les coordonnées de la sortie (liste de 2 coordonnées). Ce dictionnaire sera passé en paramètre de toutes les fonctions qui auront ainsi accès à toutes les informations nécessaires pour fonctionner. On pourra le compléter au fur et à mesure des besoins avec de nouvelles informations.

2 Affichage de labyrinthe

1. Écrire une fonction `afficheTextuel` qui reçoit un labyrinthe, et qui l'affiche textuellement dans la console, en représentant les murs par un caractère dièse `#`, les passages par un espace, l'entrée par un `x` et la sortie par un `o`. Attention : il ne faut pas modifier la liste !

2. Écrire une fonction `afficheGraphique` qui reçoit un labyrinthe, et qui l'affiche graphiquement dans la fenêtre `turtle`, les murs en couleur pleine (de votre choix), et les passages restent blancs. Afficher l'entrée et la sortie d'une autre couleur pour les distinguer.

3. S'assurer qu'on peut paramétrer facilement l'épaisseur des murs du labyrinthe, et la coordonnée du coin supérieur gauche, sans devoir modifier le code d'affichage.

3 Positionnement de la tortue

Les coordonnées de la tortue en pixels peuvent être récupérées grâce aux fonctions `xcor()` et `ycor()` mais nous allons devoir les convertir pour obtenir les coordonnées de la cellule du labyrinthe correspondante. Inversement, pour positionner la tortue dans une cellule, il faudra être capable de trouver les coordonnées en pixels du centre de cette cellule. On s'intéresse ici à écrire et tester ces fonctions auxiliaires de conversion. Les fonctions devront aussi recevoir en paramètre le dictionnaire contenant toutes les informations utiles (en particulier ici la taille des cellules et la coordonnée du coin supérieur gauche).

1. Écrire une fonction `pixel2cell` qui assure la conversion des coordonnées x,y de la position de la tortue à l'écran (en pixels), vers les coordonnées (ligne et colonne) de la cellule correspondante du labyrinthe.

2. Écrire une fonction `testClic(x,y,dicoJeu)` qui reçoit les coordonnées d'un clic, les convertit, et affiche la ligne et la colonne correspondante du labyrinthe (ou un message d'erreur si clic hors du labyrinthe).

3. Pour tester, on utilisera la commande `turtle.onscreenclick(testClic)` pour associer cette fonction au clic, puis on lancera `turtle.mainloop()`. Cliquer à divers endroits et vérifier les coordonnées affichées.

4. Écrire aussi la fonction `cell2pixel(i,j)` qui réalise la conversion inverse (coordonnées du centre de la cellule i-j). La tester.

4 Cases spéciales

1. Écrire une fonction `typeCellule` qui reçoit les coordonnées (ligne et colonne) d'une case du labyrinthe, et qui renvoie son type : entrée, sortie, passage, mur.
2. Modifier cette fonction pour qu'elle puisse aussi indiquer si une case « passage » est une impasse, un passage standard, ou un carrefour. Indice : on pourra compter les voisines de cette case par type (mur ou pas).
3. Bonus : lors de l'affichage du labyrinthe, afficher les carrefours d'une autre couleur pour les matérialiser.

Deuxième partie
Navigation guidée

5 Travail préparatoire

- Récupérer le fichier `navigation.py` dans l'archive. Le lancer et observer le comportement de la tortue quand on appuie sur les flèches.

```
import turtle

def gauche():
        print("gauche ; left")
def droite():
        print("droite ; right")
def bas():
        print("bas ; down")
def haut():
        print("haut ; up")

# key bindings
turtle.onkeypress(gauche,"Left")
turtle.onkeypress(droite,"Right")
turtle.onkeypress(haut,"Up")
turtle.onkeypress(bas,"Down")
turtle.listen()

# start loop
turtle.goto(0,0)
turtle.mainloop()
```

- Le fichier contient des fonctions à compléter, qui sont associées avec les clés (les touches du clavier) des flèches. Comprendre le code fourni.

6 Navigation guidée

1. Compléter le code des fonctions de déplacement fournies (gauche, droite, haut, bas). Tester le déplacement de la tortue dans la fenêtre.

2. Empêcher maintenant la tortue de passer à travers les murs ou de sortir du labyrinthe (notamment par l'entrée). Si l'utilisateur essaye de diriger la sortie vers un mur, elle devient rouge, ne bouge pas, et un message d'erreur s'affiche dans la console.

3. Détecter quand la tortue est sur une case spéciale, et changer sa couleur en conséquence : une couleur pour une impasse, une autre couleur pour un carrefour.

4. Détecter quand la tortue trouve la sortie : changer sa couleur en vert et afficher un message de victoire.

5. Mémoriser le chemin suivi par la tortue, sous la forme de la liste des commandes données par le joueur.

6. Écrire une fonction `suivreChemin(li)` qui reçoit une liste de commandes (chaînes de caractères formatées : `g`,`d`,`h`,`b`) et qui déplace la tortue en suivant ces commandes. Si un mouvement demandé est impossible, la fonction s'arrête en affichant un message d'erreur. Si tout le chemin est suivi avec succès, la fonction s'arrête en affichant un message de réussite.

7. Écrire une fonction `inverserChemin(li)` qui reçoit un chemin (même format que précédemment) et qui le suit en sens inverse.

8. Tester ces fonctions en mémorisant le chemin suivi par le joueur pour sortir du labyrinthe ; puis grâce aux 2 fonctions ci-dessus, rejouer le chemin à l'envers pour revenir au départ, et à l'endroit pour revenir à la sortie.

Troisième partie
Navigation automatique dans un labyrinthe simple

On considère ici des labyrinthes simples, arborescents, ne contenant aucune boucle, et ayant une seule sortie. Le but de cette partie est d'explorer automatiquement le labyrinthe pour trouver le chemin entre entrée et sortie.

1. Écrire une fonction `explorer` qui fait explorer la tortue automatiquement à la recherche de la sortie. La tortue se déplace seule, fait demi-tour dans les impasses,

et explorer le labyrinthe jusqu'à trouver la sortie. Il faut mémoriser le chemin déjà parcouru, ainsi quand une branche est explorée, la tortue doit être capable de revenir au carrefour précédent pour explorer les autres branches.

2. Modifier la fonction pour renvoyer un chemin entre entrée et sortie sous la forme d'une liste des déplacements à faire (gauche, droite, haut, bas).

3. Bonus : modifier le chemin renvoyer pour en supprimer les détours pour explorer des impasses, et obtenir ainsi le plus court chemin vers la sortie.

4. Tester le chemin généré ainsi, en utilisant la fonction précédente `suivreChemin` pour conduire la tortue de l'entrée à la sortie du labyrinthe.

Quatrième partie
Extensions

Une fois que votre tortue est capable d'explorer automatiquement les labyrinthes simples (sans boucle, avec une seule sortie), vous pouvez choisir une ou plusieurs extensions à ajouter à votre projet, parmi celles proposées ci-dessous, ou de votre invention. Le but n'est pas de réaliser toutes les extensions ! Choisissez ce qui correspond le mieux à vos capacités et vos envies.

7 Améliorer l'interface

Vous pouvez améliorer l'interface en y ajoutant un menu, des boutons, des options, des informations, etc, par exemple :

- Ajouter un bouton pour recommencer l'exploration du même labyrinthe (sans devoir le recharger depuis le fichier).

- Ajouter des boutons pour lancer l'exploration automatique ou manuelle

- Ajouter un bouton pour charger un nouveau labyrinthe (en effaçant le précédent)

- Afficher les messages dans la fenêtre graphique plutôt que la console.

- Afficher diverses statistiques intéressantes : temps d'exploration, nombre de pas avant de trouver la sortie, nombre de demi-tours, nombre d'impasses et de carrefours dans le labyrinthe chargé, etc

8 Génération de labyrinthes

- Permettre à l'utilisateur de dessiner un labyrinthe en cliquant dans l'interface `turtle`, et enregistrer ce labyrinthe dans une liste, puis dans un fichier.
- Vérifier qu'en chargeant le fichier avec la fonction fournie, on retrouve bien l'affichage du labyrinthe ainsi créé.

9 Donjons and turtles

On pourra transformer ce labyrinthe en jeu, avec des objets à collecter, des ennemis à éviter…

- Ajouter des objets dans le labyrinthe : il faudra prévoir d'autres types de case, détecter quand la tortue passe sur un objet, et lui ajouter des points selon les objets collectés.
- Proposer à l'utilisateur de poser lui-même les objets à certains endroits (par exemple dans une impasse) en cliquant dans le labyrinthe.
- Ajouter des ennemis (im)mobiles que la tortue doit éviter sur son chemin vers la sortie.
- Détecter les collisions avec les ennemis, interrompre la partie, afficher un message de défaite, et proposer de recommencer.
- On pourra limiter le nombre d'essais pour finir chaque niveau / labyrinthe. Certains objets pourraient offrir un bonus (invincibilité provisoire, vie supplémentaire…).
- En cas de victoire, charger le niveau suivant (un nouveau labyrinthe).

10 Turtle racing

Ici on veut mettre plusieurs tortues dans le même labyrinthe et les faire faire la course jusqu'à la sortie. On fournit une fonction `createTurtle` qui crée une deuxième tortue.

- Option 1 : créer une tortue sur la sortie (qui doit trouver l'entrée), et une autre sur l'entrée (qui doit trouver la sortie). Les faire démarrer en même temps, et déclarer la gagnante dès que l'une est arrivée, ou bien comparer les temps de parcours. (On considère que les tortues peuvent se croiser sur la même case.)
- Option 2 : créer plusieurs tortues dotées d'algorithmes d'exploration différents, les lancer en même temps depuis l'entrée du labyrinthe, et comparer leurs temps de parcours.

11 Labyrinthe à plusieurs sorties

On considère maintenant des labyrinthes pouvant avoir plusieurs sorties.

- Trouver la liste des chemins conduisant à une sortie, par une exploration exhaustive du labyrinthe. Renvoyer tous les chemins.
- A partir de la liste des chemins renvoyée par la fonction d'exploration, trouver le chemin le plus court, le chemin le plus « droit » (ayant le moins de changements de direction), le chemin le plus long…
- Si vous avez ajouté des objets dans le labyrinthe, calculer le chemin qui rapporte le plus de points.

12 Exploration de labyrinthes avec boucle

On attaque ici une partie (beaucoup) plus complexe. Pour l'instant on n'a considéré que des labyrinthes sans boucles, on va maintenant relâcher cette contrainte. A vous de trouver un algorithme d'exploration qui permet de trouver la sortie même dans un labyrinthe qui boucle.

1. Pour commencer, on considère un labyrinthe qui peut comporter des boucles simples, c'est-à-dire qu'un chemin peut se refermer sur lui-même (équivalent à une impasse). Indice : la tortue doit « marquer » les cellules par un gradient pour se rappeler par où elle est déjà passée et dans quel sens. Si elle revient sur ses pas, le chemin qui boucle est considéré comme une impasse (rappel : pour l'instant il ne peut pas y avoir de carrefour dans la boucle).
2. Le labyrinthe peut maintenant boucler d'un chemin vers un autre. La tortue peut alors avoir besoin de re-parcourir un chemin déjà exploré pour trouver des carrefours à finir d'explorer. Attention c'est beaucoup plus difficile, l'algorithme d'exploration peut être assez complexe. **Non obligatoire !**