Mini-projet : Évasion du labyrinthe

Cours L2 MIASHS - Informatique

Introduction

L'objectif de ce projet est de créer un jeu simple où un joueur doit s'échapper d'un labyrinthe généré aléatoirement. Le joueur se déplace dans une grille où certaines cases sont des murs infranchissables, d'autres des chemins ouverts. L'objectif est d'atteindre la sortie du labyrinthe en évitant les obstacles. Ce projet vous permettra de manipuler les boucles, les conditions, et les matrices en Python tout en explorant des algorithmes simples de parcours.

Le projet est guidé pour que vous puissiez le compléter progressivement, avec des fonctionnalités de base et des extensions pour les plus motivés.

1 Initialisation et affichage du labyrinthe

1.1 Création de la matrice du labyrinthe

La première étape consiste à créer un labyrinthe de taille fixe, par exemple 10x10. On va le représenter par une matrice dont chaque case peut être soit un mur (représenté par le chiffre `1`) soit un chemin libre (représenté par `0`). On verra ensuite comment est représenté le joueur.

La position de départ du joueur sera dans un coin du labyrinthe (par exemple, en haut à gauche), il faut donc s'assurer que cette case est un chemin libre. Enfin la sortie sera placée aléatoirement dans un des trois autres coins. Dans la matrice, on représentera la sortie par un `2`.

- Écrivez une fonction `create_maze(size)` qui prend en paramètre la taille du labyrinthe (par exemple `size = 10`) et qui renvoie une matrice de cette taille remplie de murs (des `1`) et de chemins (des `0`), et qui comporte une sortie (`2`), selon les règles données précédemment, et en respectant une proportion donnée de chemins (par exemple, 70% de chemins sur l'ensemble des cases).
- Placez la sortie du labyrinthe dans une position aléatoire sur le bord opposé à celui du joueur.

Testez votre fonction en affichant la matrice générée à l'aide de `print`. Vous devriez obtenir quelque chose comme ça :

```
map = [[0, 1, 1, 1, 1, 0, 1, 1, 0, 2],
       [0, 0, 1, 1, 1, 1, 0, 0, 0, 1],
       [1, 0, 0, 1, 0, 0, 0, 1, 0, 1],
       [1, 1, 0, 0, 0, 1, 1, 0, 1, 1],
       [1, 1, 1, 0, 0, 1, 0, 1, 1, 1],
       [1, 0, 1, 1, 0, 0, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 0, 0, 1, 1, 1],
       [1, 1, 1, 0, 1, 1, 0, 0, 1, 1],
       [1, 1, 1, 1, 1, 0, 1, 0, 0, 1],
       [1, 1, 1, 1, 1, 1, 1, 1, 0, 0]]
```

1.2 Affichage

Une fois le labyrinthe généré, on peut se donner les moyens de l'afficher à l'écran de manière plus lisible. Une première solution est de représenter les murs par des `#` et les chemins par des espaces vides.

Définir un dictionnaire

```

dico = {0: ' ', 1: '#', 2: 'O'}
```

qui contient les images à afficher comme valeur associée à chacune des valeurs de la matrice de départ.

Et écrire ensuite une fonction `draw_maze(maze, dico)`.

Si vous testez votre fonction avec le labyrinthe de la question précédente avec `dico` comme dictionnaire, vous devriez obtenir :

```
# #### # # O
#       ## #
##  #   # #
##    #  ##
###  # # ##
 #  ##    #
##### # ##
###  ##  ##
##### #  #
########
```

2 Déplacement du joueur

2.1 Initialisation du joueur

Le but est désormais d'ajouter un petit personnage que l'on va représenter dans le code par un dictionnaire contenant trois entrées : `char`, `x`, et `y`. L'entrée `char` contient le caractère représentant le personnage (par exemple, `o`), tandis que `x` et `y` définissent sa position en abscisse et en ordonnée.

— Écrivez une fonction `create_perso(start)` qui prend en argument un couple d'entiers représentant la position de départ et renvoie un dictionnaire représentant le personnage. Par exemple, le résultat de `create_perso((0,0))` devrait être : `{"char": "o", "x": 0, "y": 0}`.

— En vous inspirant de votre fonction `draw_maze`, créez une nouvelle fonction `draw_maze_with_char(maze, dico, perso)` qui prend en paramètre une carte `maze`, un dictionnaire de caractères `dico`, et un personnage `perso`, et affiche la carte avec le personnage sur cette carte. Attention, cette fonction ne doit pas modifier les objets `maze`, `dico`, et `perso`.

Testez votre fonction `draw_maze_with_char` avec la même carte et le même dictionnaire donnés précédemment, ainsi que le personnage créé précédemment. Le résultat devrait être une version du labyrinthe avec le personnage représenté par le caractère `o` à sa position de départ.

```
o #### # # 0
#      ## #
##  #   # #
##    #  ##
###  # # ##
 #  ##    #
##### # ##
###  ##  ##
##### #  #
########
```

Déplacement

Le but est maintenant d'ajouter la possibilité de déplacer le personnage. Nous allons utiliser la fonction `input` que vous avez déjà utilisée, qui permet de récupérer ce que l'utilisateur écrit dans le terminal.

Rappel de l'utilisation de `input` :

```

d = input("Quel déplacement ?")
```

Ici, on affiche "Quel déplacement ?" puis on attend que l'utilisateur écrive quelque chose et appuie sur "entrée". La valeur entrée est ensuite stockée dans `d`.

Ce type de contrôle n'est pas ce qu'il y a de plus naturel, mais il pourra être amélioré dans la suite du TP.

Avec la fonction `input`, nous voulons récupérer une lettre parmi "z" (haut), "q" (gauche), "s" (bas), et "d" (droite). Si une autre lettre est entrée, nous afficherons un message d'erreur demandant de recommencer.

— Écrivez une fonction `update_p(letter, p)` qui prend en argument une lettre `letter` et un joueur `p` (représenté par un dictionnaire comme précédemment) et qui met à jour la position du joueur. Cette fonction ne renvoie rien (la fonction `input` n'est pas encore utilisée ici).
— Récupérez l'entrée de l'utilisateur avec `input`, puis mettez à jour la position du joueur avec `update_p`. Affichez ensuite de nouveau la carte avec le personnage à sa nouvelle position.
— Testez votre programme en entrant une des directions à l'exécution.

Remarque : Ici nous ne nous sommes pas encore préoccupés des murs, le personnage peut marcher par-dessus les murs ! La fonction `update_p` ne modifie pas la carte, en fait le joueur n'apparaît pas sur la carte, lors de l'affichage, la carte ne doit pas être modifiée.

Jouer à l'infini

Vous avez maintenant déplacé votre personnage une fois. Il est naturel de vouloir le déplacer autant de fois que nécessaire. Pour cela, vous aurez besoin d'une boucle `while`.

— Mettez le code nécessaire pour déplacer le personnage une fois (`input` puis mise à jour de la position, puis affichage de la carte) dans une boucle `while True:`. Cela répétera indéfiniment l'exécution du code contenu dans la boucle.
— Testez en essayant de déplacer votre personnage plusieurs fois. Vous devrez arrêter l'exécution en appuyant sur le bouton Stop (le carré proche du terminal).

Déplacement en respectant le labyrinthe

Le joueur peut maintenant se déplacer à l'aide des touches du clavier (`z` pour aller en haut, `q` pour aller à gauche, `s` pour aller en bas, et `d` pour aller à droite). :

Créez maintenant une fonction `update_p(maze, letter, p)` qui prend aussi en paramètre le labyrinthe et ne modifie les coordonnées du personnage que si c'est possible dans le labyrinthe, c'est à dire :

— si il n'y a pas de mur sur la case de destination,
— si le personnage ne sort pas des bords du labyrinthe.

3 Amélioration

Étape 4 : Atteindre la sortie

Le but du jeu est d'atteindre la sortie du labyrinthe. La sortie est représentée par un caractère comme `'S'`.

— Modifiez la boucle infinie pour vérifier si le joueur a atteint la sortie à chaque déplacement. Si le joueur atteint la sortie, affichez un message de victoire et terminez le jeu.

4 Améliorer les contrôles avec la bibliothèque curses

Cette partie est plus complexe et optionnelle, elle doit vous permettre de jouer plus confortablement en utilisant une bibliothèque pour gérer les entrées au lieu de juste la fonction input. Avant de commencer cette partie pensez à copier le fichier sur lequel vous travaillez pour pouvoir y revenir si vous n'arrivez pas à faire cette partie.

La première chose est d'ouvrir un terminal, cela se fait par exemple depuis Thonny, on y accède en cliquant sur "Outils" puis "Ouvrir la console du système". Pour installer la bibliothèque curses, il suffit d'écrire 'pip install curses' ou 'pip install windows-curses' si vous êtes sur Windows. Ensuite, il faudra toujours lancer votre programme depuis cette console (celle que vous utilisez habituellement avec Thonny ne fonctionnera pas) en écrivant `python nom_de_votre_programme.py`.

Passons maintenant à expliquer ce qui change dans votre programme et après ce que vous devez faire précisément sera détaillé dans les questions.

La première chose est de charger la bibliothèque avec 'import curses', ensuite on déclare l'écran sur lequel on affichera du texte avec `curses.initscr()` et on précise qu'on ne veut pas afficher les touches pressées avec `curses.noecho()`. Pour vous, ça revient à copier-coller le code suivant au début de votre code.

```
import curses
stdscr = curses.initscr()
curses.noecho()
```

`stdscr` est l'écran où on rattachera tous nos affichages. Il faut maintenant remplacer tous les `print` que l'on faisait par des affichages sur `stdscr`.

Ensuite, pour afficher quelque chose, on a désormais besoin de préciser la position où on l'affiche. Par exemple, pour afficher "x" à la position (1,2), on écrit :

```
stdscr.addstr(1, 2, "x")
stdscr.addstr(2, 3, "On peut aussi mettre des chaînes de caractères !")
```

La deuxième ligne est un exemple d'écriture de phrases au lieu d'un seul caractère. Mais il faut faire attention à ne pas écrire sur une zone où on a déjà écrit quelque chose, sinon on écrit par-dessus.

Maintenant, pour récupérer les déplacements, on utilise la fonction `getkey` de la façon suivante :

```
c = stdscr.getkey()
```

Enfin, il est nécessaire, entre deux déplacements, d'effacer l'écran avant d'afficher la carte et le personnage, puis de rafraîchir l'écran pour faire apparaître les changements.

Copiez-collez le code donné plus haut pour importer la bibliothèque et créer l'écran `stdscr`.

— Modifiez votre boucle while pour arriver au résultat suivant :

```
    while True:
        c = stdscr.getkey()
        stdscr.erase() # on efface l'écran
        update_p(c, p, m)
        draw_maze_with_char(m, p, d) # on affiche la carte et le joueur
        stdscr.refresh() # on met àjour l'écran
```

— Changez tous vos `print` pour des `addstr`. Par exemple, pour afficher la carte, vous pourrez parcourir la matrice et utiliser `stdscr.addstr(i, j, d[m[i][j]])` au lieu de `print(d[m[i][j]])`. Pour afficher les erreurs, vous pourrez les afficher en dessous de la carte en faisant par exemple :

```
    stdscr.addstr(len(m) + 2, 0, "erreur, vous n'avez pas indiqué un caractère dans {z
```

— Testez en exécutant et cherchez à résoudre les éventuels bugs. N'hésitez pas à demander de l'aide. N'oubliez pas de lancer depuis la console de votre système, à laquelle on accède dans Thonny depuis "Outils > Ouvrir la console du système..." puis vous lancez votre programme en écrivant `python nom_de_votre_programme.py` (sans oublier d'appuyer sur entrée).

5 Ajout d'objets à récolter

Pour aller plus loin dans le développement du jeu, nous allons introduire des objets que le joueur pourra ramasser dans le labyrinthe. Cela ajoutera de la complexité et de l'intérêt au jeu. Voici les différentes étapes à suivre pour implémenter cette fonctionnalité :

5.1 Création des objets à récolter

La première étape consiste à ajouter des objets à collecter dans le labyrinthe. Pour cela, nous allons créer une fonction `create_items(maze, num_items)` qui positionne un certain nombre d'objets (`num_items`) à des positions aléatoires du labyrinthe.

La fonction ne modifie pas le labyrinthe `maze`. Ell retourne une liste de tuples qui correspondent aux coordonnées des objets !

Il faut aussi s'assurer que :

— les objets sont placés uniquement dans des cases libres (marquées `0`).

— Les objets ne se trouvent ni sur la position de départ du joueur ni sur la sortie.

Pour cela vous aurez besoin de la fonction `random.randint`.

```
import random
x = random.randint(0, 10)
```

5.2 Affichage du labyrinthe avec les objets

Ensuite, nous allons modifier l'affichage du labyrinthe pour inclure les objets à collecter. Pou cela écrire une nouvelle fonction `draw_maze_with_char_and_items`

Consignes :

— Créez une fonction `draw_maze_with_char_and_items(maze, dico, items, perso)` qui prend en compte le labyrinthe, le dictionnaire des caractères, la liste des objets, et le personnage.

— Lors de l'affichage, si une case contient un objet, choisissez quel caractère le représente.

5.3 Ramassage des objets

Pour permettre au joueur de ramasser les objets, nous allons créer une fonction `collect\_item(perso, items)`.

Son effet est simplement de supprimer un objet de la liste des objets à ramasser.

5.4 Modification de la boucle de jeu

Il est maintenant nécessaire de modifier la boucle principale du jeu pour inclure la collecte des objets.

Il s'agit surtout d'appeler `collect\_item()` à chaque déplacement pour vérifier si le joueur ramasse un objet, et afficher le nombre d'objets collectés.

On peut aussi modifier la condition de fin du jeu pour qu'elle ne fonctionne que si on a rammasé tous les objets.

6 Fonctionnalités supplémentaires (optionnelles)

Pour rendre le jeu du labyrinthe encore plus captivant, voici une liste de nouvelles fonctionnalités que vous pouvez ajouter. Ces éléments permettront de diversifier les mécaniques de jeu, d'enrichir l'expérience du joueur, et d'augmenter la difficulté.

Le mieux est d'ailleurs de faire vos propres fonctionnalités. Vous pouvez vous inspirer de ce qui suit ou non.

6.1 Système de vies

Créez un système de vies pour le joueur. Le joueur commence avec un certain nombre de vies et en perd une lorsqu'il tombe dans un piège.

- **Pièges** : Ajoutez des cases spéciales représentant des pièges. Si le joueur tombe sur un piège, il perd une vie et retourne à la position de départ.
- **Affichage** : Affichez le nombre de vies restantes à chaque déplacement. Si le joueur n'a plus de vies, le jeu se termine.

6.2 Ennemi déplaçable

Ajoutez un ennemi qui se déplace aléatoirement dans le labyrinthe et qui poursuit le joueur.

- **Déplacement de l'ennemi** : Déplacez l'ennemi après chaque déplacement du joueur. L'ennemi suit une logique de poursuite aléatoire ou suit le joueur selon un algorithme.
- **Interaction avec le joueur** : Si l'ennemi atteint la même position que le joueur, le joueur perd une vie.

6.3 Portes et clés

Ajoutez des portes (par exemple, `'D'`) et des clés (par exemple, `'K'`) dans le labyrinthe. Pour ouvrir une porte, le joueur doit d'abord ramasser la clé.

- **Clés et portes** : Placez des clés et des portes aléatoirement dans le labyrinthe.
- **Interaction** : Le joueur doit collecter une clé pour pouvoir franchir une porte. Si le joueur tente de franchir une porte sans clé, il est bloqué.

6.4 Système de score

Ajoutez un système de score basé sur le nombre de déplacements, le temps mis pour s'échapper, ou le nombre d'objets ramassés.

- **Calcul du score** : Le score est déterminé par différents critères (moins de déplacements = meilleur score).
- **Affichage** : Affichez le score à la fin du jeu.

6.5 Labyrinthe changeant

Faites en sorte que certaines sections du labyrinthe changent de manière aléatoire pendant la partie.

- **Murs mouvants** : Certaines cases du labyrinthe changent de statut aléatoirement (mur à chemin, ou inversement) pendant que le joueur se déplace, rendant le jeu plus difficile.

6.6 Mode multi-niveaux

Ajoutez plusieurs niveaux avec une difficulté croissante.

— **Niveaux** : Chaque niveau est un labyrinthe de taille croissante, ou avec de plus en plus de pièges et d'ennemis.

— **Progression** : Le joueur passe au niveau suivant après avoir trouvé la sortie et atteint des objectifs spécifiques.

6.7 Niveaux aléatoires

Générez des niveaux aléatoirement.

6.8 Timer

Ajoutez une contrainte de temps pour chaque niveau.

— **Compte à rebours** : Le joueur dispose d'un temps ou d'un nombre de déplacements limité pour s'échapper. S'il échoue, il perd une vie ou recommence le niveau.

— **Affichage** : Affichez le temps restant à chaque déplacement.

6.9 Téléporteurs

Ajoutez des cases spéciales qui téléportent le joueur vers une autre partie du labyrinthe.

— **Création des téléporteurs** : Placez des téléporteurs (par exemple, `'T'`) aléatoirement dans le labyrinthe par paires.

— **Interaction** : Si le joueur atterrit sur un téléporteur, il est instantanément transporté vers la case correspondante.

10. Points de sauvegarde

Ajoutez des points de sauvegarde où le joueur peut sauvegarder sa progression dans le niveau.

— **Sauvegarde** : Si le joueur atteint un point de sauvegarde, il peut y revenir en cas de perte de vie.

— **Retour au point de sauvegarde** : Si le joueur tombe dans un piège, il est ramené à son dernier point de sauvegarde au lieu de recommencer depuis le début.

6.10 Zones de brouillard

Ajoutez des zones de brouillard qui cachent certaines parties du labyrinthe jusqu'à ce que le joueur s'en approche.

— **Brouillard de guerre** : Cachez les zones du labyrinthe qui sont loin du joueur, et révélez-les uniquement quand il s'en approche.

6.11 Pièges activables

Ajoutez des pièges que le joueur peut activer pour ralentir ou détourner l'ennemi.

- **Pose de pièges** : Le joueur peut poser des pièges à des endroits stratégiques pour piéger l'ennemi.
- **Effet des pièges** : Les pièges peuvent immobiliser ou ralentir les ennemis pendant un certain temps.

6.12 Mode furtif

Ajoutez un mode furtif pour que le joueur puisse éviter les ennemis en se déplaçant sans bruit.

- **Activation du mode furtif** : Le joueur peut activer le mode furtif pour se déplacer plus lentement mais sans attirer l'attention des ennemis.
- **Limitation** : Le mode furtif est limité par une barre d'énergie qui se vide progressivement.

6.13 Enigmes de portes

Ajoutez des portes qui ne peuvent être ouvertes qu'en résolvant une énigme.

- **Enigmes** : Lorsque le joueur atteint une porte, une énigme est proposée (par exemple, un calcul mathématique ou une question de logique).
- **Ouverture** : La porte ne s'ouvre que si l'énigme est résolue correctement.

Ces fonctionnalités supplémentaires offrent divers niveaux de difficulté et ajoutent une dimension plus stratégique au jeu, augmentant ainsi son intérêt et sa rejouabilité. Vous pouvez en implémenter certaines en fonction du niveau de complexité souhaité.