Auteurs : NGUYEN Tien Thanh, NGO Truong Son,
PHAM Thanh Nguyen, Guokuang DENG

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

DECOUPAGE FONCTIONNEL

A. LISTE DES FONCTIONS ESSENTIELLES :

1. GUI AVEC TKINTER :

- **Tkinter (Tk interface)** est un module intégré à la bibliothèque standard de Python, permettant de créer des interfaces graphiques. Grâce à ce module, on peut créer aisément des fenêtres, des widgets tel que les boutons, les labels, les sous-titres, etc
- Les widgets et les fonctions que l'on utilise dans le code :
  - **Button(), Label(), Frame()** permettent de créer un interface du jeu.
  - **bind()**: permet de lier une fonction *callback* à un événement sur le widget. Dans le jeu, on utilise cette fonction pour mettre à jour la liste de données, supporter des joueurs à marquer un chemin, ouvrir d'autres zones d'affichage.
  - **Canvas()** permet de dessiner le labyrinthe et de tracer le chemin entre 2 points marqués sur la zone d'affichage.

2. CHRONOMETRE :

- **after() :** un attribute de tkinter qui permet de faire appel une fonction call-back après un quantité du temps, donc, aide à lancer un chronomètre en jouant
- **time :** un bibliothèque avec des fonctions utiles comme Time() pour recevoir des informations de temps actuelle.

3. LIRE ET EXPORTER DES DONNEES :

- **csv :** un bibliothèque aide à lire des fichiers csv où toutes les informations relatives stokées dans la labyrinthe.
- **panda :** une autre bibliothèque qui peut gérer des fichiers csv plus complexes, donc permet de stocker les informations relatives à la progression du joueur, telles que les scores plus élevés, le nombre de fois où il a joué, etc.

4. L'ALGORITHME : TROUVER LE PLUS COURT CHEMIN

- Il permet de trouver le chemin le plus efficace entre deux points. Il peut être utilisé pour résoudre des problèmes dans divers domaines, tels que trouver le chemin en évitant des pièges donnés à la main ou par la fonction de random
- Il peut être appliqué à différents types de graphes, tels que les graphes non orientés et orientés, pondérés ou non pondérés.
- Il existe plusieurs algorithmes efficaces pour trouver le plus court chemin, tels que Dijkstra, BFS,...

5. AMELIORER DES EXPERIENCES DES JOUEURS :

- **Pygame :** une bibliothèque multiplateforme qui facilite le développement du jeu avec langage de programmation et on l'utilise pour jouer de la musique. Le module très utile pour le faire est **mixer().**
- **Pillow :** une bibliothèque nécessaire de gérer les images pour améliorer la qualité de notre jeu.

6. D'AUTRES FONCTIONS / BIBLIOTHEQUES :

- **Random :** une bibliothèque avec un nombreuse de fonction qui sont utiles pour générer des nombres aléatoires. Une fois importée, elle permet créer la matrice où des éléments aléatoires sont placés
- **Math** : un module standard qui fournit des fonctions mathématiques avancées. Il permet de réaliser des calculs mathématiques plus complexes que ceux possibles avec les fonctions mathématiques de base.

B. LISTE DE FONCTIONS PREVOIR :

class **Fenetre_jeu**():

def **init**(self, arg*): *'''initialisation des variables'''*
Entrée: <***self***> attributs et méthodes de *Fenere_jeu,* <***arg****> *:* autres informations
Retour : None

def **creer_widgets**(self): *'''créer le premier interface avec Tkinter'''*
Entrée: <***self***> attributs et méthodes de *Fenere_jeu*
Retour : None

def **ouvrir_fenetre_Canvas**(self, *event*): *'''créer un TopLevel pour dessiner un canvas du jeu'''*
Entrée: <***self***> attributs et méthodes de *Fenere_jeu,* <***event***> des événement
Retour : None

def **lire_fichier**(self, *level*): *'''lire le fichier csv qui contient des informations du niveau level'''*
Entrée: <***self***> attributs et méthodes de *Fenere_jeu,* <***level*** *: string*> level pour jouer
Retour : <**data** : list 2D> l'information du labyrinthe

def **dessine_labyrinthe**(self, data): *'''créer un Canvas pour dessiner une zone de labyrinthe'''*
Entrée: <***self***> attributs et méthodes de *Fenere_jeu,* <***data*** *: liste 2D*> le labyrinthe
Retour : None

def **run_timer**(self): *'''être utilisée avec after() pour faire le chronomètre'''*
Entrée: <***self***> attributs et méthodes de *Fenere_jeu*
Retour : None

def **afficher_résultat**(self): *'''affichier le chemin plus court sur le Canvas'''*
Entrée: <***self***> attributs et méthodes de *Fenere_jeu*
Retour : None

def **sauvergarder**(self): *'''sauvergarder les scores du joueur'''*
Entrée: <***self***> attributs et méthodes de *Fenere_jeu*
Retour : None

def **creer_matrice_aléatoire**(self, taille): *'''créer une matrice aléatoire du labyrinthe'''*
Entrée: <***self***> attributs et méthodes de *Fenere_jeu,* <***taille*** *: tuple*> width, height du labyrithe
Retour : <**map** : list 2D> un labyrinthe en créant aléatoirement

def **calcul_chemin_plus_court**(self, map): *'''trouver le plus court chemin du labyrinthe entré'''*
Entrée: <***self***> attributs et méthodes de *Fenere_jeu,* <***map*** *: liste 2D*> le labyrinthe
Retour : <**chemin** : liste de tuple> et <**distance** :float> : le plus court chemin et sa distance

def **quitter**(self): *'''quitter le jeux'''*
Entrée: <***self***> attributs et méthodes de *Fenere_jeu*
Retour : None

def **jouer_musique**(self, song): *'''jouer la musique, créer des effets sonores '''*
Entrée: <***self***> attributs et méthodes de *Fenere_jeu,* <***song*** *: string> nom de la musique*
Retour : <***id***> l'ID de la chanson

def **pause_musique**(self, id): *'''faire une pause de la musique'''*
Entrée: <***self***> attributs et méthodes de *Fenere_jeu,* <***id***> *: id de la musique*
Retour : None