Projet Python 2019-2020

TD n°7 : Taquin

Objectif. - Réaliser un taquin pleinement jouable, sans bug.

Il existe de nombreux taquin en ligne, par exemple taquin.net (toutefois il contiennent souvent des bugs ou sont peu ergonomiques).

Vous devez bien connaître ce qui est expliqué dans le tutoriel du jeu Memory, en particulier

- la séparation logique du jeu et vue
- la communication du canevas avec les deux grilles de données, cf. le codage du jeu

Pour le code de démarrage, on se base sur le corrigé donné au TP n°2 (fichier `etape0.zip`). Le déplacement d'une tuile sera **non progressif** (c'est *beaucoup* plus facile à coder). Pour déplacer une tuile, il faut qu'une de ses voisines soit vide. Dans ce cas, un clic sur la tuile la déplacera dans la case vide.

On va privilégier une nette *séparation* entre la logique du jeu et la vue. L'état du jeu sera enregistré dans un tableau 2D de taille $n \times n$, nommé `board` (en pratique $n = 4$ mais vous *devez* traiter le cas général où n est quelconque, important pour le debug). Chaque case de `board` représentera à chaque instant du jeu la valeur entre 1 et $n^2 - 1$ figurant sur le plateau dessiné sur le canevas. La case vide aura la valeur `0`.

Point capital : tout changement dans le jeu doit avoir sa traduction dans `board`.

On dispose, cf. fichier joint `vue0.py`, d'une fonction qui dessine dans un canevas Tkinter un plateau de jeu. Tester que ça marche.

Le déplacement d'une tuile sera provoqué par un clic sur la tuile que l'on veut bouger (il n'y a jamais qu'un seul déplacement possible). La tuile à déplacer sera supposée être directement voisine de la case vide.

La partie modèle

Vous utiliserez le fichier modèle `modele0.py` (faire une sauvegarde). Ce fichier est complètement indépendant de Tkinter. Il travaille sur le modèle `board`.

Tester la fonction d'affichage `print_board`.

Coder les fonctions suivantes et les TESTER avec la fonction d'affichage disponible dans le fichier :

- Une fonction `voisins(n, i, j)` qui renvoie la liste de toutes les cases voisines (nord, sud, est ou ouest) de la case en ligne `i` et colonne `j` d'un plateau carré de côté `n`.
- Une fonction `caseVide(board, i, j)` qui renvoie la position de la case vide si elle est voisine de la position `(i,j)` et `None` sinon.
- Une fonction `swap(board, i, j)` qui échange dans `board` les valeurs entre la valeur en la position `(i,j)` et la case vide si elle est voisine de la position `(i,j)`.
- Une fonction `win(board)` qui détecte si la grille est convenablement triée (et donc que le jeu est gagné).
- Pour déboguer (si nécessaire), une fonction `find_empty(board)` qui détermine la position ligne x colonne de la case vide du jeu.

La partie vue

Inutile d'attaquer cette partie si la partie modèle n'est pas terminée. Vous repartirez du fichier `etape1.zip`. Pour simplifier, on n'utilisera pas d'id pour gérer le déplacement des tuiles, mais **on effacera tous les items et on redessinera**. Ecrire les fonction suivantes dans `vue1.py` :

- Une fonction `pos2lineCol(n, x,y)` qui à partir d'une position `(x, y)` du plateau sur le canevas (donc en pixels) renvoie les indices ligne et colonne `i` et `j` dans la grille `board` de côté `n`.
- Une fonction `clic` qui permette de jouer (bien connaître les événements de la souris).
- Tester, en particulier l'annonce de la victoire.

Améliorations

- Placer un label qui annonce la victoire
- Séparer en deux fichiers, l'un pour la partie modèle, l'autre pour la partie vue (qui importera des fonctionnalités de la partie modèle)
- Modifier la fonction de mélange pour que puzzle soit résoluble ; pour cela partir d'une grille correcte, mélanger un nombre assez grand de fois (disons $2n^2$) case par case avec la fonction `swap`.
- Modifier l'ergonomie du jeu pour que le joueur puisse déplacer plusieurs tuiles (jusqu'à $n-1$) d'une même colonne ou d'une même ligne, comme on fait avec un jeu réel.
- Au lieu de tout effacer, ne déplacer que la tuile qui doit bouger (utiliser la méthode `move`).
- Implémenter un compteur de coups.
- Implémenter un déplacement progressif des tuiles.
- Implémenter un solver.