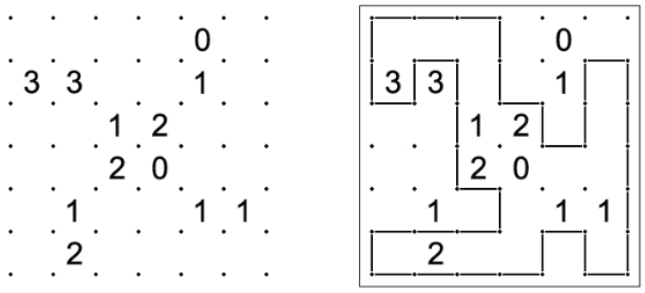
SLITHERLINK

L'objectif de ce projet est d'implémenter un petit jeu de type "puzzle" avec interface graphique en python 3 et utilisant le module fltk et un algorithme de recherche automatique de solutions.



https://en.wikipedia.org/wiki/Slitherlink

Introduction

Objectifs:

Objectif principal : Création du programme (graphique) permettant de jouer au jeu.

Deuxième objectif : programmation d'un solveur (= générateur automatique de solutions)

Démarche logique / mathématiques

4 Tâches :

➔ Structures de données (chargement de la grille au format .txt, représentation de l'état de jeu et fonctions d'accès)

➔ Condition de victoire (chaque indice est satisfait et l'ensemble des segments tracés forment une boucle fermée + affichage de la condition de victoire)

➔ Interface graphique (avec fltk, détection des clics)

➔ Recherche de solutions

Le programme:

Démarche programmation :

1. Fonctions d'accès

Pour la programmation des fonctions d'accès, nous avons juste appliqué l'algorithme indiqué dans le sujet du projet et rajouter certaines fonctions intermédiaires.

Ajout d'une fonction `segments_adjacents(etat, sommet)` qui retourne la liste des 4 segments adjacents à un sommet. Cette fonction sera appelée dans segments_traces, segments_interdits et segments_vierges ; fonctions qui vérifieront si les segments adjacents à un sommet sont tracés, interdits ou vierges.

Ajout de la fonction `verifier_coord(segment)` qui ordonne les segments en commençant par la plus petite coordonnée.

Ajout d'une fonction `segments_qui_composent _une_case(etat, case)` qui renvoie la liste des 4 segments qui composent une case et qui sera appelé dans statut_case.

Ajout d'une fonction `statut_case(indices, etat, case)` qui vérifie le statut d'une case, c'est à dire qu'elle vérifie si la case possède le bon nombre de tracé, s'il en manque ou s'il y en a trop. Elle appelle alors segments_qui_composent_une_case puis affichage_un_indice en fonction de ce statut pour que le nombre de segment soit affiché de la bonne couleur. Enfin, elle renvoie le nombre de segment où le tracé y est respecté ainsi que booleen_rouge, utilisés lors de son appel dans la fonction indices_satisfaits.

2. Conditions de victoire

Pour la tâche 2, nous avons également suivi l'algorithme présenté en créant la fonction `indices_satisfaits()` sans paramètre qui vérifie que chaque indice est satisfait (première partie du programme). Elle renvoie alors True ou False en fonction de si tous les indices sont satisfaits, suivi de booleen_rouge récupéré dans la fonction statut_case (cette dernière sert dans le cas du solveur par exemple).

La fonction `longueur_boucle(etat, segment)` qui vérifie si le segment appartient à une boucle et sa longueur (ou renvoie None s'il n'appartient pas à une boucle).

3. Chargement de la grille

Création d'une fonction charger_grille() qui va permettre de charger la grille de son choix au format txt. La fonction va lire un fichier puis va vérifier en premier lieu que la grille existe, puis va vérifier si la grille est conforme, c'est-à-dire pas de caractère interdit (la grille ne doit contenir que les symboles '0', '1', '2', '3' qui correspond aux indices, et '_' qui correspond à l'indice None, et va enfin vérifier qu'il n'y a pas d'erreur de format, donc des lignes de longueurs différentes. Si une erreur est trouvée, le programme va informer le type d'erreur rencontré à l'utilisateur et va lui redemander une nouvelle grille.

Une fois toutes ces vérifications effectuées, on va pouvoir afficher la grille.

4. Interface graphique

D’autres fonctions graphiques vont être implémentées :

`coords_sommets()` va afficher tous les sommets.

`init_affichage_indices(indices)` est appelée en début de jeu. Elle va afficher les indices en noir en appelant affichage_un_indice, sauf pour les indices en 0 qui sont initialisées en bleu.

`affichage_un_indice(idx, idy, color)` va afficher l’indice selon la couleur entrée en paramètre ainsi que son emplacement calculé grâce à idx et idy. Elle est appelée par init_affichage_indices et statut_case présentées précédemment.

5. Détection des clics

La fonction `obtenir_segment(evt_x, evt_y)` va vérifier si l’on est en train de cliquer sur un segment ou non. Pour cela on a défini une certaine marge (initialisé à ecart = 20) où lorsque l’on clique sur une zone rectangulaire définie autour du segment (= zone de clic), va vérifier que l’on a bien cliqué sur le segment. La fonction va alors retourner le segment correspondant.

evt_x et evt_y correspondent aux coordonnées du clic.

Pour calculer les coordonnées du sommet nous avons utilisé les formules d’explications et les formules fournies dans le sujet. Pour calculer l’indice du sommet on va faire idx = evt_x - taille_marge) / taille_case, et nous ferons la même chose pour idy avec evt_y. Les valeurs ainsi obtenues vont être arrondis.

Puis nous allons calculer sommet_x et sommet_y de la façon suivante :

sommet_x = idx * taille_case + taille_marge (resp. pour sommet_y)

Ensuite nous allons calculer les écarts pour vérifier que le segment se trouve bien du rectangle prédéfini (invisible). Si le clic a bien été effectué à l’intérieur du rectangle (donc verticalement et horizontalement) alors on retournera les coordonnées du segment. Sinon segment vaut None.

6. Dessiner le segment une fois que l’on a cliqué dessus

Nous allons d’abord récupérer le tag du segment avec la fonction `obtenir_tag(segment)`.

Puis on pourra dessiner le segment avec la fonction `dessiner(segment, type_trace)`

On va d’abord récupérer les coordonnées des extrémités du segment, donc 2 tuples.

Calculer les coordonnées du premier et du deuxième sommet (à l’aide de la formule fournie).

Si type_trace vaut 1, on va tracer le segment en noire avec la fonction ligne de fltk. Sinon on trace une croix au centre. On calcule le milieu et on trace la croix en rouge. Lors de l’utilisation de la fonction ligne, on précise le tag permettant utilisé plus tard pour effacer le trait si nécessaire avec la fonction effacer_segment.

Pour ce qui est de l'affichage, ce sera dans le programme principal.

7. Recherche de solutions

Pour la fonction `solveur(etat, sommet, lbl)` vérifiant si le jeu admet une solution, on suit différentes étapes de l'algorithme présenté :

- dans le cas où le sommet entré en paramètre possède 2 segments tracés : la fonction condition_indices (vérifiant que tous les indices sont satisfaits) est égale à True, elle renvoie bien True, et dans le cas contraire : False.

- dans le cas où sommet adjacent à plus de deux segments tracés dans etat cela crée un branchement, il sera donc impossible d'avoir une boucle fermé : elle retourne donc False.

- sinon par élimination, dans le cas où le sommet n'est adjacent à aucun ou à un seul segment : On boucle pour chaque segment adjacent au sommet mais qui ne soit pas déjà tracé.

On vérifie alors qu'il n'enfreint pas les indices en vérifiant qu'il n'existe pas d'indice dépassant le nombre de segments autorisés avec booleen_rouge récupéré lors de l'appel de indices_satisfaits(). Si ce dernier révèle un indice dans le rouge, on efface ce segment et on essaie sur les autres. S'il n'en possède pas, on peut passer à la suite.

On rappelle alors récursivement l'algorithme sur l'autre extrémité du segment rajouté. Si le résultat de la fonction récursive est True, on renvoie alors True. Dans le cas contraire, on efface alors ce segment.

- enfin, dans le cas où aucun segment partant de sommet ne permet de continuer, on renvoie False, il n'existe pas de solution partant de ce sommet.

8. Programme principal

On récupère le type d'évènement (clic gauche ou clic droit) ainsi que les coordonnées de l'endroit cliqué.

Par la suite, on vérifie que le clic se situe bien dans la zone de jeu et dans la zone de clic d'un segment (vérifié lors de l'appel de la fonction obtenir_segment pour récupérer les coordonnées des points du segment correspondant).

Dans le cas où il s'agit d'un clic gauche et que le segment est vierge, on va le tracer. S'il est déjà tracé on va l'effacer. Sinon on efface le tracé précédent et le remplace par un segment.

De même, s'il s'agit d'un clic droit et que le segment est vierge on va tracer une croix. Si le segment est déjà interdit, donc que la croix est déjà dessinée, on va l'effacer. Sinon on efface ce qui est déjà tracer et on met la croix.

On fait ensuite appelle au fonction indices_satisfaits et longueur_boucle pour vérifier les deux conditions de victoire. Si ces dernières sont remplies, on affiche on message de félicitation et on attend un clic pour sortir de la fenêtre. Dans l’autre cas, le jeu continue.

Manuel utilisateur:

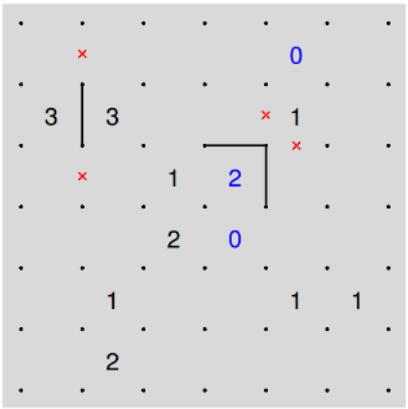
Slitherlink est un jeu de type casse-tête, se jouant sur une grille de nombres et d'inconnues. Le but du jeu est de tracer les côtés des cases en respectant les règles suivantes :

• Un numéro dans une case indique le nombre de côtés de la case qui doivent être tracés : ni plus, ni moins.

• Une case vide est une absence d'information, le joueur pourra tracer autant de côtés qu'il le souhaite. L'ensemble des côtés tracés doit former une unique boucle fermée.

Clic gauche : Tracer ou effacer un segment déjà existant

Clic droit : segment interdit (représenté par une croix rouge)



Informations complémentaires:

Problèmes rencontrés : La partie la plus difficile à programmer fut le solveur. Il n'était pas évident de le programmer même si on a compris la démarche et la logique derrière l'algorithme.

Une autre partie un peu difficile fut la détection des clics et les parties d'affichage.

Déroulé / ressentis personnel :

Chacune de nous a participé activement au projet. Nous avons répartis le travail de manière assez équitable que ce soit pour le programme ou le compte-rendu.

Estimation : (programme : Sarah 60%, Laure 40% - rapport Sarah 40%, Laure 60%)

Ce projet était particulièrement difficile mais très intéressant et enrichissant.