Réalisé Par :

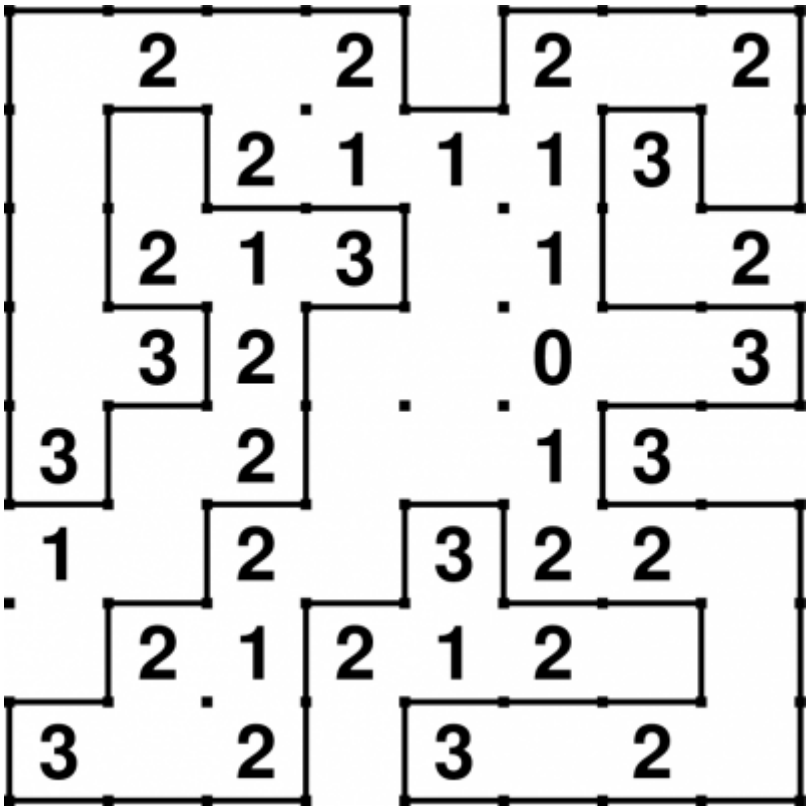
Yassine EL JIDE

Berat DERIN

Groupe du TP :

TP G-H

Projet de Programmation : Le jeu Slitherlink



Année Scolaire : 2020/2021

Les objectifs du programme :

Dans ce projet de fin de semestre, on nous a demandé de créer un programme graphique utilisant FLTK permettant de jouer au jeu SlitherLink (expliqué dans le manuel) et de plus rajouter une option de solveur ou générateur automatique des solutions avec le BackTracking.

Manuel d'utilisateur :

1. Description du jeu :

SlitherLink (aussi connu comme Clôtures et Loop de Loop) est un casse-tête logique avec des règles simples et des solutions complexes. ... Vous devez tracer des lignes entre les points pour former une boucle unique sans croiser ni faire de branche. Les nombres indiquent combien de lignes entourent chaque case. Les règles sont assez simples :

- 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.

2. Guide d'utilisation :

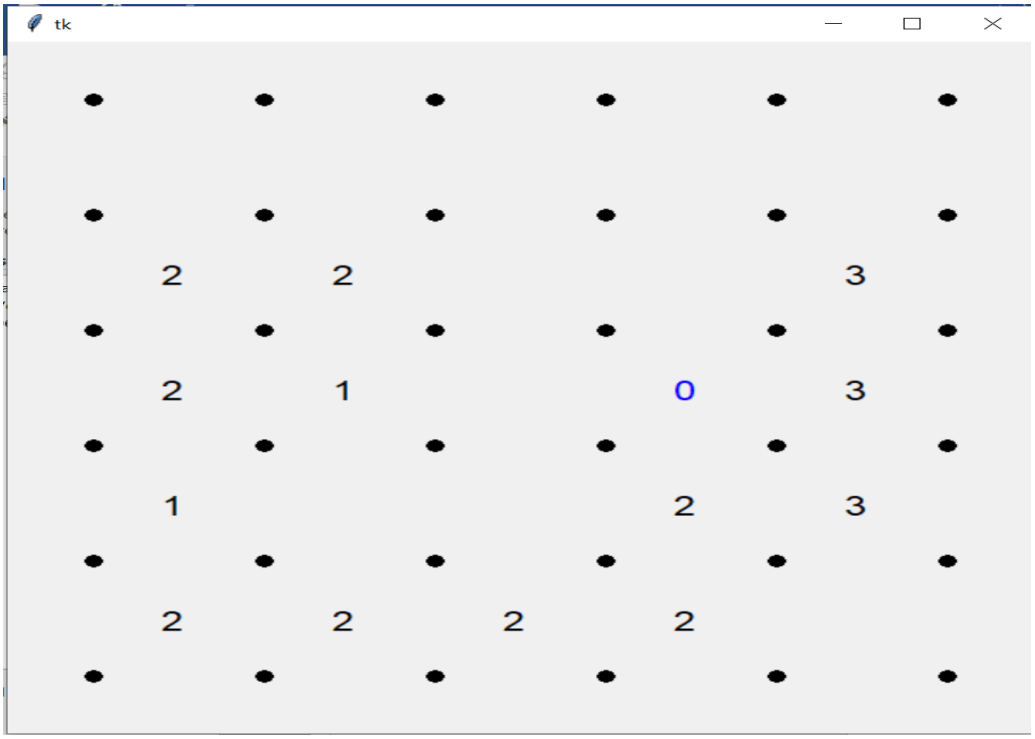
Pour démarrer le jeu il faut exécuter le fichier slitherlink.py en utilisant la **commande python slitherlink.py** dans le terminal**.**

Une fenêtre graphique va s'afficher avec le texte :

« **Tapez sur une touche pour commencer !»**.

Il y aura ensuite un menu avec toutes les grilles disponibles et le bouton **quittez** en dessous de la fenêtre.

Après avoir choisir une grille, on aura une fenêtre de ce type par exemple :



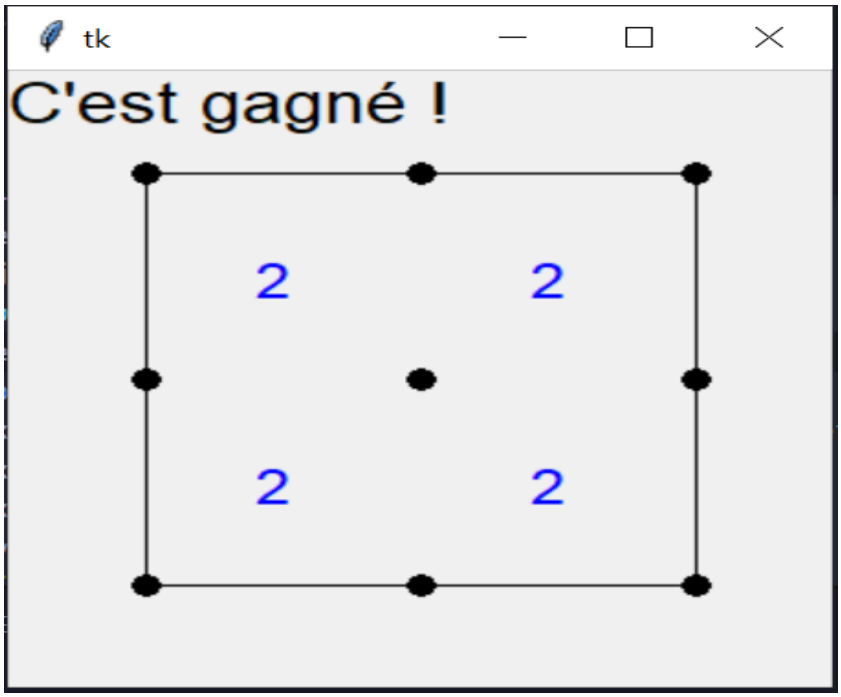
Pour créer un segment entre deux points, il suffit de faire **un clic gauche précisément entre les deux points** et de ne pas s'éloigner des points sinon ça ne fonctionnera pas.

Pour préciser qu'un point est interdit, il suffit de faire **un clic droit avec le même principe,** précisément entre les deux points.

Pour enlever un segment déjà tracé il suffit de **faire un autre clic gauche pour l'enlever**.

La couleur des indices change en fonction de la validité des nombres segments tracés autour d'une case, si la couleur est **bleue** ça veut dire que la case est satisfaite, sinon elle est en **rouge** ; cela c'est rend le jeu plus « user friendly ».

Si le joueur arrive à vérifier les conditions du jeu, un message de victoire est affiché en haut gauche de l'écran. Voici un exemple :



Pour résoudre la grille automatiquement, il suffit de **cliquer sur n'importe quelle touche du clavier**.

Démarches de programmation :

Dans l'énoncé du sujet on a reçu quelques aides concernant les démarches et les étapes à suivre pour créer le jeu.

Le code est réparti d'une façon correspondante à celle mentionnée dans l'énoncé du sujet : un fichier pour les fonctions qui s'occupent de la victoire et la manipulation des segments et des listes pour la tâche 1 et 2, un fichier graphique pour la partie graphique du jeu et enfin un fichier solveur responsable de la tâche 4 de la résolution du jeu.

Concernant les fonctions de la tâche 1 et 2 :

- Les fonctions « **est_trace, est_interdit, est_vierge** » nous permettent de détecter si le segment passé en paramètre vérifie la condition ou pas (tracer/ interdit/ vierge) et renvoie une valeur booléenne.
- La fonction **seg_adjacent** nous renvoie une liste des segments adjacents à un sommet passé en paramètre.
- Les fonctions « **tracer_segment, interdire_segment, effacer_segments** » nous permettent de modifier la valeur du segment dans le dictionnaire état en 1 pour tracer, -1 pour interdire et le supprimer pour effacer.
- Les fonctions « **segments_traces, segments_interdits, segments_vierges** » nous renvoie une liste soit une liste des segments interdits ou tracés ou vierges adjacents au sommet passé en paramètre et on s'aide avec la fonction créée **seg_adjacents** pour nous

renvoyer les segments puis on les vérifie avec les fonctions du début.

- La fonction **statut_case** prend une case en paramètre avec le dictionnaire état et le tableau des indices et vérifie si la valeur contenue dans la case correspond aux nombres de segments qui sont tracés et nous renvoie la valeur 0 si c'est valide, -1 si on a dépassé le chiffre de la case sinon 1 au cas où on a encore besoin de tracer d'autres segments.
- La fonction **cases_satisfait** vérifie que toutes les cases du tableau indices sont satisfaites et renvoie True si c'est le cas.
- Les fonctions « **premier_indice, deuxieme_indice** » vérifient les conditions de la victoire si les segments tracés forment une boucle fermée et si chaque case contenant un chiffre k possède k côtés tracés.
- La fonction **longueur_boucle** vérifie si un segment appartient à une boucle, si le segment n'appartient pas à une boucle la fonction renvoie None sinon elle renvoie la longueur de la boucle.
- La fonction **affichage** affiche dans la console à chaque fois si les conditions sont vérifiées ou pas à l'aide des fonctions premier_indice et deuxieme_indice.
- La fonction **lire_fichier** qui crée la grille de jeu indices, pour cela elle lit un fichier et parcourt chaque ligne de ce dernier, si le caractère est un chiffre compris entre 0 et 3 ou est un tiret du bas, alors on l'ajoute dans indice avec le tiret transformé en None.

Concernant les fonctions de la tâche 3 :

- La fonction **trop_proche** est une fonction qui nous permet d'avoir une marge d'erreur lors d'un clic.
- La fonction **menu** permet d'afficher le menu du jeu et nous renvoie le fichier sur lequel l'utilisateur a cliqué.
- La fonction **ready** et **fin** sont des fonctions graphiques qui permettent d'afficher le message du départ du jeu et celui de la victoire.
- Les fonctions « **print_points, print_indices, print_etat** » sont les fonctions principales responsables de l'update du graphique et la mise en place des données stockées pour les points, on les affichent avec une double boucle en prenant en compte une marge qu'on définit comme variable globale et pour l'état qui contient les données sur les segments, on trace des lignes suivantes une formule mathématique pour trouver les coordonnées exactes et enfin pour les indices on les affichent aussi en prenant en compte la marge (avec l'aide reçu dans l'énoncé pour régler ce problème pour ce type de jeu).

Concernant les fonctions de la tâche 4 :

Pour la tâche 4 la fonction principale est la fonction solveur qui est une fonction récursive où on a suivi les étapes données dans l'énoncé mais cela ne fonctionne pas correctement pour les grilles 1 et 2 sinon ça fonctionne pour les petites grilles.

En ce qui concerne « slitherlink.py », c'est le fichier principal du jeu où l'on exécute toutes les fonctions d'une manière organisée avec la détection des clics que nous fournit la bibliothèque FLTK.

Répartition du travail et problèmes rencontrés :

Nous avons réussi à reproduire un clone du jeu Slitherlink.

En suivant les étapes données dans l'énoncé qui nous ont beaucoup aidé lors de la programmation du jeu.

Cependant, nous trouvons notre programme satisfaisant même si nous pourrions sûrement l'améliorer, notamment pour le solveur où nous avons eu un résultat satisfaisant mais nous n'avons pas réussi pas à régler le problème pour résoudre les grilles de plus grandes tailles telle que la grille 1 et 2 donnée dans l'énoncé.

Concernant notre travail en binôme, nous avons réussi à bien communiquer et travailler à deux à l'aide de Discord mais aussi sur zoom via nos partages d'écran. Cela a été en effet une bonne expérience et surtout très amusant de programmer à deux un clone d'un jeu tel que SlitherLink.