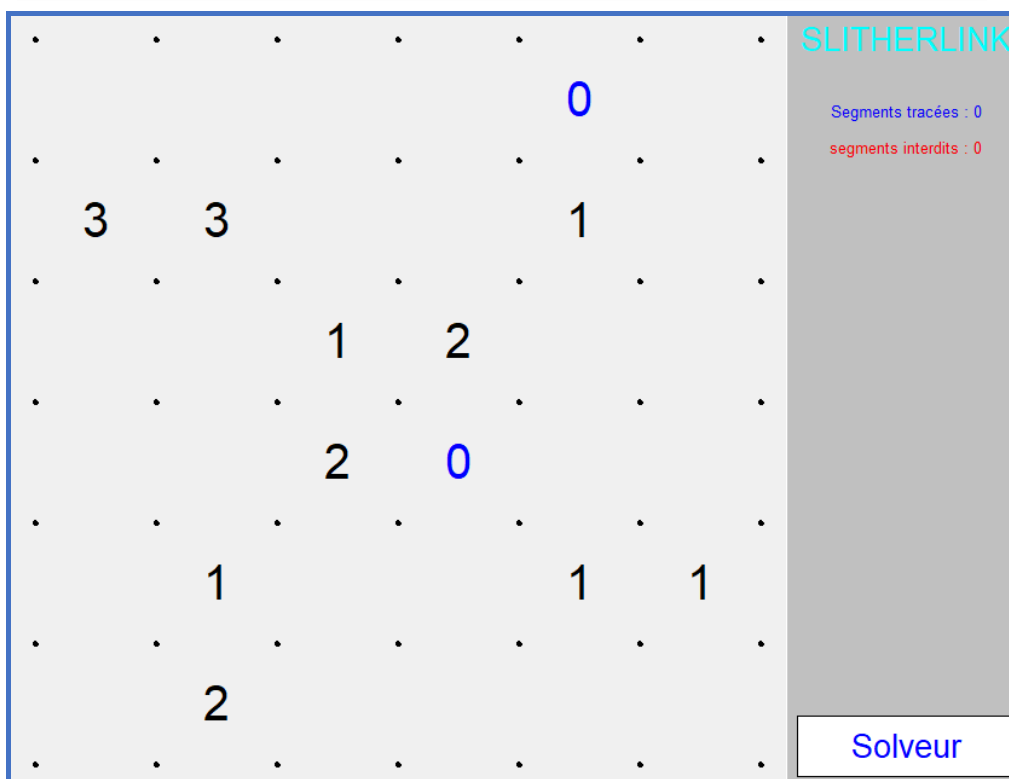


Slitherlink

RAPPORT DE PROJET

КЪРЪОКЛ ДЕ РЪОЈЕЛ



The image shows a screenshot of a Slitherlink puzzle interface. The puzzle is an 8x8 grid of dots. Numbers are placed in some of the cells, indicating how many segments of the loop are adjacent to that cell. The numbers are: (1,1)=3, (1,2)=3, (1,4)=1, (2,3)=1, (2,4)=2, (3,3)=2, (3,4)=0, (4,2)=1, (4,5)=1, (4,6)=1, (5,3)=2. Some numbers are in blue (0, 0, 1, 1). The interface includes a sidebar on the right with the title 'SLITHERLINK', a status bar showing 'Segments tracées : 0' and 'segments interdits : 0', and a 'Solveur' button at the bottom.

Row \ Col	1	2	3	4	5	6	7	8
1	3	3			1			
2			1	2				
3			2	0				
4			1		1	1		
5			2					
6								
7								
8								

Ako Seer Harley MENSAH-ASSIAKOLEY TD : A

Anass OUSAID TD : A

Introduction

Slitherlink est un célèbre jeu de logique, se jouant sur une grille de nombres et d'inconnues. Nous avons recréé ce célèbre jeu de logique en nous basant sur les compétences acquises au cours de cette année en langage Python.

I. Présentation du Projet

Le but est de tracer les côtés des cases afin de former une boucle fermée qui respecte les règles suivantes :

- **Chaque case contient un indice qui indique le nombre de segments exacts qui doivent être tracés autour de la case.**
- **Une case sans indice peut contenir autant de côté qu'on souhaite.**
- **L'ensemble des cotés doivent former une unique boucle fermée.**

La solution est bien constituée d'une unique boucle fermée, et on voit que chaque indice de case est respecté : autour des cases contenant un 0 aucun côté n'est tracé, un seul côté pour les cases contenant un 1, et ainsi de suite.

II. Guide d'utilisation

Le programme donne la possibilité au joueur en début de partie de débiter une nouvelle partie ou de restaurer la dernière partie sauvegardée. S'il choisit la première option, le programme lui propose des grilles de taille différente parmi lesquelles il devra choisir.

N.B : Tous ces choix se feront par un clic gauche de sa souris.

Le Jeu se joue à l'aide de la souris et de quelques boutons du clavier :

- Pour tracer un segment reliant deux points, un clic gauche à l'endroit souhaité est requis. Ainsi on observera l'apparition d'une ligne reliant les deux points.
- Pour interdire un segment, un clic droit à l'endroit souhaité est requis. Pour signaler au joueur que ce segment est interdit, on y dessine une croix rouge.
- Un clic gauche ou droit sur un segment tracé ou interdit à pour effacer de le rendre vierge, c'est-à-dire de le supprimer.
- Pour lancer le Solveur graphique, le joueur doit effectuer un clic gauche sur le bouton "Solveur" situé dans le coin droit en bas de la fenêtre de jeu.
Pour faire tourner le solveur, il doit effectuer des clics répétés sur la touche 'Espace' de son clavier.
Pour arrêter le solveur, il devra cliquer sur la touche 'Esc' ou 'Echap' de son clavier, puis sur la touche 'Entrée' ou 'Return' pour revenir à ce qu'il faisait.
- Si la partie est terminée, on donne le choix au joueur de pouvoir quitter le jeu ou de retourner au menu principal s'il veut rejouer.
- En quittant une partie non terminée, elle est automatiquement sauvegardée.

III. Structure du code

Nous avons initialisé un dictionnaire 'JEU' qui se chargera de contenir toutes les données nécessaires au jeu.

A. Fonctions de base

- **est_trace(etat, segment)** : fonction qui renvoie **True** si la clé 'segment' vaut 1 dans le dictionnaire 'etat' ou **False** sinon.
- **est_interdit(etat, segment)** : fonction qui renvoie **True** si la clé 'segment' vaut -1 dans le dictionnaire 'etat' ou **False** sinon.

- **est_vierge(etat, segment)** : fonction qui renvoie **True** si la clé 'segment' n'est pas dans le dictionnaire 'etat' ou **False** sinon.
- **tracer_segment(etat, segment)** : fonction modifiant "etat" afin de représenter le fait que segment est maintenant tracé, c'est-à-dire que "segment" devienne une clé de "etat" avec 1 comme valeur.
- **interdire_segment(etat, segment)** : fonction modifiant "etat" afin de représenter le fait que segment est maintenant interdit, c'est-à-dire que "segment" devienne une clé de "etat" avec -1 comme valeur.
- **effacer_segment(etat, segment)** : fonction modifiant "etat" afin de représenter le fait que segment est maintenant vierge, c'est-à-dire que "segment" sera supprimé de "etat" s'il s'y trouve.
- **segments_traces(etat, sommet)** : fonction qui renvoie la liste des segments tracés adjacents à "sommet" dans "etat", c'est-à-dire la liste des clés de "etat" dont "sommet" est un des éléments et la valeur dans "etat" est 1.
- **segments_interdits(etat, sommet)** : fonction renvoyant la liste des segments interdits adjacents à "sommet" dans "etat", c'est-à-dire la liste des clés de "etat" dont "sommet" est un des éléments et la valeur dans "etat" est -1.
- **segment_dans_grille(indices, segment)** : fonction qui renvoie un booléen qui indique si le segment est bien dans la grille "indices".
- **segments_vierges(indices, etat, sommet)** : fonction renvoyant la liste des segments vierge adjacents à "sommet" dans "etat", c'est-à-dire la liste des tuples (segment) qui ne sont pas dans etat et dont "sommet" est un des éléments.
- **statut_case(indices, etat, case)** : fonction qui renvoie **None** si la case ne porte pas d'indice, ou **0** si l'indice est satisfait, ou **1** s'il est encore possible de satisfaire l'indice, ou **-1** s'il n'est plus possible de satisfaire l'indice.

B. Conditions de victoire

Pour définir les conditions de victoire du joueur on a défini ces fonctions :

- **indices_satisfaits(indices, etat)** qui va renvoyer **True** si tous les indices de la grille sont satisfait et **None** sinon.
- **construction_boucle(etat, depart, precedent, courant, nb_seg=1)** qui est une fonction récursive prenant le dictionnaire "etat", trois tuples "depart", "precedent", "courant", et un compteur "nb_seg" en paramètre et renvoyant: **None** si le tracé des segments ne forme pas une boucle et "nb_seg" si il forme bien une boucle, "nb_seg" serait alors dans ce cas le nombre de segments constituant cette boucle.
- **longueur_boucle(etat, segment)** prenant le dictionnaire "etat" et un tuple de tuples "segment" et renvoyant **None** si le segment n'appartient pas à une boucle, et la longueur de la boucle à laquelle il appartient sinon.
- **nb_segment_trace(etat)** qui va renvoyer le nombre de segment tracé dans la grille, et aussi un segment qui sera utiliser comme un segment de depart pour la fonction '**longueur_boucle**'
- **fin_partie(etat)** qui va appeler toutes les fonctions d'avant, et après il va détecter si le nombre de segment parcouru par '**longueur_boucle**' est égale au nombre de segment de la grille renvoyé par '**nb_segment_trace**' pour renvoyer **True**, sinon il renvoie **False**.

C. Interface Graphique

On a créé plusieurs fonctions qui nous ont aidés à implanter les menus, la sauvegarde et la restauration du jeu:

- **sauvegarde_grille(jeu, grillegoc)** : Fonction qui sauvegarde la liste des indices du jeu dans le fichier grillegoc.

- **restauration_grille(jeu, grilledoc)** : Fonction restaure la liste des indices du jeu contenue dans le fichier grilledoc et de celle-ci, initialise la hauteur et la largeur de la grille du jeu.
- **sauvegarde_etat(jeu, etatdoc)** : Fonction qui sauvegarde le dictionnaire 'etat' des segments tracés ou interdits du jeu dans le fichier etatdoc.
- **restauration_etat(jeu, etatdoc)** : Fonction qui restaure le dictionnaire 'etat' du jeu contenu dans le fichier etatdoc.
- **nb_seg_trac_interd(etat)** : Fonction comptant le nombre des clés ayant -1 comme valeur (segments interdits) et le nombre des clés ayant 1 comme valeur (segments tracés) dans le dictionnaire 'etat'.
- **menu_choice(jeu)** : Fonction gérant le menu de choix de la grille de jeu.
- **menu_start(jeu, grilledoc="grilledoc.txt", etatdoc="etatdoc.txt")** : Fonction gérant le menu de démarrage du jeu donnant le choix à l'utilisateur de débiter une nouvelle partie ou de restaurer la partie sauvegardée.
- **menu_right(jeu)** : Fonction affichant l'avancée du jeu.
- **menu_final(jeu)** : Fonction affichant le menu final lors de la fin de la partie quand la grille est résolue.

N.B : Ces fonctions se situent dans le fichier python menu qui accompagnera tout le travail.

---> Et après nous avons créé les fonctions qui détectent l'endroit des clics et savent à quel endroit tracer le segment ou l'interdire ou encore le rendre vierge:

- **coord_som(indice, jeu)** : Fonction prenant un entier indice et un dictionnaire "jeu" et renvoyant une coordonnée correspondante dans la grille.
- **index_som(coord, jeu)** : Fonction prenant un entier "coord" et un dictionnaire "jeu" et renvoyant un float selon la taille des cases et la marge de la grille.
- **distance(sommet1, sommet2)** : Fonction prenant des tuples sommet1 et sommet2 représentant des points puis calculant la distance entre ces deux points avant de le retourner.
- **proximite(x, y, jeu)** : Fonction prenant en paramètre deux entiers x, y (symbolisant les coordonnées d'un clic), le dictionnaire jeu puis vérifie si le clic a été fait aux alentours d'un sommet de la grille. Et si oui, renvoie une liste dont le premier élément est ce sommet et le second la liste des sommets potentiels correspondant au segment recherché.
- **clic_left(x, y, jeu)** : Fonction prenant en paramètre deux entiers x, y (symbolisant les coordonnées d'un clic), le dictionnaire jeu et se chargeant de modifier la clé 'etat' de 'jeu' en y ajoutant ou en retirant un segment selon le vouloir de l'utilisateur si son clic est aux environs du segment.
- **clic_right(x, y, jeu)** : Fonction prenant en paramètre deux entiers x, y (symbolisant les coordonnées d'un clic), le dictionnaire jeu et se chargeant de modifier la clé 'etat' de 'jeu' en y ajoutant ou en retirant un segment selon le vouloir de l'utilisateur si son clic est aux environs du segment.
- **dessine_pts_grille(jeu)** : Fonction prenant en paramètre le dictionnaire jeu et dessinant les points de la grille sur la fenêtre.
- **dessine_indices(jeu)** : Fonction prenant en paramètre le dictionnaire jeu et dessinant les indices dans leur case respectif avec une couleur spécifique selon le statut de la case dans l'avancée du jeu sur la fenêtre. cette couleur est :
 - Noire si la case a encore la possibilité d'être satisfaite en traçant des segments autour de la case.
 - Bleu si elle est satisfaite.
 - Rouge si s'il n'est plus possible de satisfaire l'indice parce que trop de segments sont déjà tracés ou interdits autour de la case.
- **dessine_segments(jeu)** : Fonction prenant le dictionnaire jeu et dessinant les segments tracés et interdits contenus dans le dictionnaire 'etat' de 'jeu'.
- **dessine_grille(jeu)** : Fonction gérant l'affichage de tous les éléments de la grille de jeu.

D. Recherche de solutions

Cette partie du code est dédié au solveur.

- **exces_indice(indices, etat):** vérifie qu'il n'y a pas un excès de segments dans la grille ("pour toutes les cases de la grille")
- **cherche_sommets(indices)** : La fonction qui renvoie les sommets de depart de l'algorithme et si il n'y a pas d'indices dans la grille il renvoie "None".
- **cherche_solution(indices, etat, sommet):** Fonction qui implémente l'algorithme du solveur en commençant d'un sommet et à chaque fois dessiner des segments jusqu'à arriver à une boucle unique fermer et il renvoie un booléen qui indique si on peut arriver à faire une boucle fermer en satisfaisant tous les cases.
- **si_grille_valide(indices, etat):** Fonction qui renvoie un booléen qui indique si la grille est valide ou pas en utilisant les fonctions d'avant.
- **dessine_seg_solveur(solveur)** : Fonction qui reçoit un dictionnaire qui ressemble à celui de la variable "etat" qui définit le jeu mais dans ce cas il définit celui du solveur, et en fonction de la valeur associer à chaque segment il trace soit un segment tracé soit un segment interdit dans l'interface graphique.
- **draw_indice_solveur(jeu)** : Fonction qui reçoit un dictionnaire "jeu" contenant toutes les informations du jeu inclus la liste des indices, et en fonction de chaque indice de la grille il détecte s'il est satisfait pour le tracé en Bleu, et s'il n'est pas encore satisfait pour le tracé en Noir, et en rouge s'il n'est plus possible de le satisfaire.
- **dessine_solveur(solveur):** Fonction qui reçoit le dictionnaire etat du "solveur" et qui appelle toutes les fonctions nécessaire pour tracé les segments du solveur et les indices de la grille dans l'interface graphique.

E. Tâches complémentaires effectuées

- Nous avons implanté un **solveur graphique** afin qu'ils puissent afficher les étapes en traçant et effaçant à chaque fois les segments dans la fenêtre de jeu au fur et à mesure de la recherche et à son arrêt, nous pouvons à nouveau compléter ce que nous avons déjà fait dans la grille avant de choisir le solveur.
- Nous avons codé également la **sauvegarde** du jeu à chaque fois que le joueur clique sur "quitter", et aussi la **restauration** du jeu afin que le joueur puisse restaurer le jeu de la partie précédente.
- Nous avons aussi essayé d'afficher le nombre de segments tracés et interdits.

IV. Avancée du Projet

Nous avons terminé toutes les tâches qui nous ont été imposées et avons essayer quelques tâches supplémentaires.

- **Pourcentage de Réalisation**
Les Fonctions de base ainsi que les conditions de victoire ont été réalisées conjointement.
L'Interface Graphique a été réalisé par Ako Seer et Le solveur par Anass.
- **Bugs rencontrés** : C'était beaucoup de difficultés, parfois nous avons des bugs dont nous ne pouvons même pas identifier les causes dans le code, mais grâce à certaines techniques de débogage comme l'utilisation des « print », nous avons finalement pu les localiser.
 - Nous avons rencontré des soucis pour associer l'interface graphique à nos fonctions et permettre la détection des clics de correspondre au segment souhaité par l'utilisateur.
 - Nous avons également eu une légère difficulté à connecter le solveur à l'interface graphique, et nous avons pu le faire à la fin.

Conclusion

Grâce à ce projet, nous avons pu mettre en pratique toutes connaissances acquises durant cette première année de Licence Mathématiques Informatique.

Ce projet nous a permis d'obtenir beaucoup de nouvelles choses qui vont nous aidez dans notre vie professionnelle :

- Améliorer notre compétence en langage python.
- Essayer de chercher l'information tous seul par internet.
- Arriver à localiser le bug dans un code long.
- Résoudre des bugs compliqués.
- Savoir connecter le mode graphique et le mode console.
- Implémenter des solveurs rapides et efficaces.
- Écrire un code organiser et efficace.

Table des matières

Introduction	2
I. Présentation du Projet	2
II. Guide d'utilisation.....	2
III. Structure du code.....	2
A. Fonctions de base.....	2
B. Conditions de victoire	3
C. Interface Graphique	3
D. Recherche de solutions.....	4
E. Tâches complémentaires effectuées.....	5
IV. Avancée du Projet	5
Conclusion	6