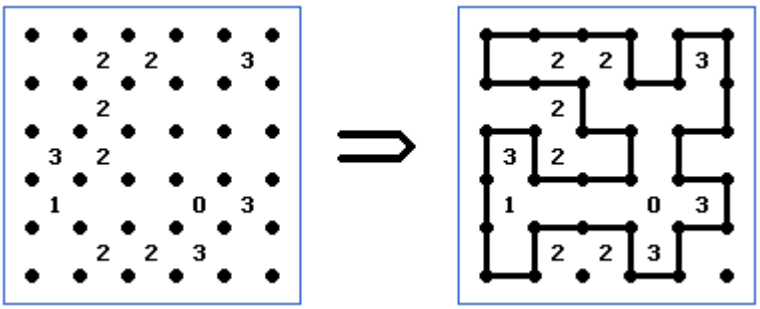
Projet SlitherLink



Introduction

Durant notre deuxième semestre de L1 en Math/Info l'UGE nous avons dû faire un, en Algorithme et Programmation 2 (AP2), slitherlink ainsi qu'un solveur de celui-ci.

Un slitherlink est un jeu consistant d'une grille et de nombre constituant les indices de nombres de segments que doit entourer la case de cette indice, si le joueur réussit à faire une boucle en vérifiant chaque indice la partie se termine et le joueur gagne.

Objectifs:

L'objectif principal de ce projet est de créer un Slitherlink fonctionnel vérifiant donc les conditions de victoires.

Nous avons donc dû créer une interface graphique permettant d'afficher la grille et les segments que l'utilisateur créera ainsi que les indices de la grille. Il pourra aussi marquer sur la grille les segments qu'il pense interdit, c'est-à-dire, qui ne respectent pas les indices des cases.

Outre la création d'un jeu Slitherlink, on nous a aussi demandé de mettre en place un solveur automatique de celui-ci permettant donc de gagner une partie sans rien faire mais aussi de s'adapter à n'importe quelle grille l'utilisateur importera.

Manuel utilisateur / Mode d'emploi:

Afin d'utiliser ce programme, il faut ouvrir l'invite de commande et aller dans le répertoire où est placé le programme et les grilles.

Après avoir fait cela il suffit d'écrire : "*python3 slitherlink.py*", l'interface vous demandera de choisir votre grille (votre fichier de grille doit être appelé tel que : "*grilleN.txt*" avec N un chiffre entre 1 et 9.

Après le choix d'une grille l'utilisateur peut choisir entre utiliser le solveur automatique de la grille ou s' il veut faire une partie sur la grille.

Si il choisit de jouer manuellement, il peut, grâce, au clic gauche de la souris placer un segments entre deux points, si il souhaite le supprimer il lui suffira de cliquer sur le segment de nouveau et si il veut interdire un segment, qui sera marqué en rouge, il faudra qu'il utilise le clic droit de la souris.

Lorsque la partie est terminée, il aura le choix de revenir au menu de choix des grilles ou bien de fermer le programme.

Le programme:

Démarche programmation :

Lors de la programmation de ce projet, nous avons d'abord fait en sorte que le programme gère qu'elles sont les segments qui sont considérés comme "*tracé*", "*interdit*" ou "*vierge*", pour cela nous avons mis en place un dictionnaire permettant de stocker chaque segments ainsi que leurs propriétés. Ces propriétés peuvent être vérifiées grâce aux fonctions "*est_trace*" , "*est_interdit*" et "*est_vierge*".

Il fallait ensuite faire en sorte d'ajouter les coordonnées du segment dans le dictionnaire stockant les états de chaque segment. On a donc mis en place des fonctions permettant de considérer deux points de la grille comme étant tracée, interdite ou vierge les fonctions étant "*tracer_segment*", "*interdire_segement*" et "*effacer_segment*".

Une fonction de tri a aussi été mise en place permettant ainsi de considérer, par exemple, le segment ((0,0),(0,1)) et ((0,1),(0,0)) comme étant le même segment.

Les conditions de victoires ont ensuite été mises en place, une fonction nommée “*statut_case*” va permettre au programme de vérifier si chaque indice de la grille est vérifié ou non, c’est-à-dire, si elle possède le bon nombre de segment autour de la case si celle-ci possède un chiffre entre 1 et 3.

Il y a ensuite la fonction “*longueur_boucle*” vérifiant si la boucle se ferme.

Et la fonction “*victoire1*” qui va vérifier chaque case de la grille ainsi que les indices de celles-ci en faisant appelle à la fonction “*statut_case*” et renvoyant True si l’indice est vérifié et False sinon.

Après cela, nous avons choisi de mettre en place l’interface graphique, à l’aide de la fonction “*interface*” qui prend la longueur de la liste des indices afin de déterminer la taille de la grille qui va être affichée et si en parcourant la liste il tombe sur un indice qui est un chiffre en 1 et 3 qui va être afficher dans l’interface et qui va aussi afficher si le nombre de segments autour de celui-ci est bon (affiché en bleu) ou non (en rouge si la case en possède trop, en noir si elle n’en a pas assez ou pas du tout). Cette fonction permet aussi d’afficher les segments qui sont dans le dictionnaire “*etat*” regroupant les segments tracés et interdits, les segments tracés sont reprensentés en noir et les segments interdits sont représentés en rouge.

Une fonction permettant de mettre à jour l’interface nommée “*maj*” va gérer tout ce qui vient de l’utilisateur, il va détécter à l’aide de la bibliothèque “*fltk.py*”, il permettra donc lorsque qu’il détecte que si l’utilisateur utilise son clic gauche entre deux points de coordonnées de la grille ajouté les coordonnées des deux points dans le dictionnaire “*etat*” et les considérer comme des segments grâce à la fonction “*tracer_segment*” tout en vérifiant au préalable que le segment était bien vierge avant le clic de l’utilisateur. Le même procédé est utilisé pour le clic droit qui va interdire un segment. Si l’utilisateur souhaite supprimer un segment tracé ou interdit il peut simplement cliquer dessus que ce soit avec un clic droit ou clic gauche.

Ces fonctions vont être utilisées dans la fonction “*jeu*” permettant de jouer au jeu manuellement qui s'arrêtera, si elle est appelée, lorsque le joueur gagne la partie.

Afin d’importer les grilles de l’utilisateur, la fonction “*creer_indices*” va permettre de lire le contenu d’un fichier texte qui représente la grille et ses indices, ce fichier texte va être converti en une liste de liste, appelée matrice comportant tous les éléments du fichier importé.

D’autres interfaces graphiques ont aussi été créées, telle qu’une interface proposant à l’utilisateur de choisir la grille qu’il veut, pouvant donner le choix entre 9 grilles qui

est géré par la fonction “*choix_grille*” et des interfaces qui permettent à l’utilisateur de choisir entre faire une partie avec le solveur ou manuellement ainsi qu’une interface demandant à l’utilisateur s’il souhaite rejouer ou non.

Pour finir nous avons mis en place le solveur, la fonction “*solveur*” est la fonction principale du solveur, elle va permettre de parcourir la grille en essayant toutes les combinaisons jusqu’à trouver une solution, si une de ses combinaisons n’est pas bonne alors il efface son segment le plus récent et va essayer autre chose, tout ça est effectué de façon récursive et en faisant appelle aux fonctions “*tracer_segment*”, “*segment_trace”* et “*segment_vierge*” ces deux fonctions permettant au solveur de regarder si les segments adjacents à la position du solveur est tracés ou vierges.

La fonction “*case_adj*” est aussi utilisée dans la fonction principale du solveur, qui va permettre à celui-ci de vérifier si les cases adjacentes à la position du solveur possède un indice et si il est bien vérifié.

Pour finir, nous avons aussi la fonction “*start*” qui permet au solveur de toujours commencer à partir d’une case qui possède un indice, pour cela le programme va parcourir la grille et si il trouve un indice entre 1 et 3 il posera son point de départ sur celui-ci, si la grille ne possède pas d’indice égal à 3 alors il cherchera une case d’indice 2 si il n’en trouve pas alors il cherchera une case d’indice 1 sinon le programme s'arrête et le joueur gagne par défaut.

Toutes ces fonctions vont être utilisées dans une boucle, qui s'arrêtera lorsque l’utilisateur aura gagné sa partie et qu’il ne souhaite pas rejouer ou s’il ferme la fenêtre.

Le temps que le solveur prend à résoudre la grille sera aussi affiché si l’utilisateur l’utilise.

Informations complémentaires:

Lors de l’importation graphique du solveur, nous avons remarqué que pendant que celui-ci cherchait à résoudre la grille l’interface graphique du Slitherlink commence à clignoter ce qui peut être très dérangeant lors de l’utilisation du programme.

Déroulé / ressentis personnel:

Lors de ce projet nous avons décidés de nous répartir les tâches, l'un s'occupait de l'implantation des fonctions permettant de vérifier l'état de chaque segment, des conditions de victoires ainsi que du solveur tandis que l'autre s'occupait de la partie graphique du programme et de l'implantation des fonctions sous la forme graphique, notamment l'implantation du solveur sous forme graphique affichant l'avancement de celui-ci lorsque l'utilisateur l'utilise.

Globalement, ce projet a été plutôt amusant à réaliser, il nous a montré l'importance d'une bonne répartition des tâches afin de pouvoir le finir plus ou moins rapidement et ainsi s'occuper des problèmes ainsi que l'importance d'une bonne communication dans le groupe, ces deux atout nous permettant ainsi de s'entraider lorsqu'une personne ne comprend pas comment s'y prendre ou tombe sur un problème lors de la programmation d'une fonction.

Cette entraide a été très utile lors de la réalisation du solveur qui était une des partie les plus dure du projet.