Sylvain TRAN / Hakim AOUDIA

TP a-b

Slitherlink

Sommaire:

-Introduction
-Principe du Jeu
-Manuel utilisateur
-Résolution grille
-Étapes de programmation
-Problèmes rencontrés
-Tâches complémentaires

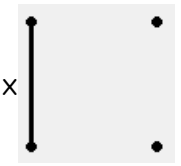
•*Introduction* :

Pour la fin du semestre 2, nous devions réaliser le projet suivant : le jeu Slitherlink. C'est un jeu graphique nécessitant une souris et un clavier pour utiliser l'option du solveur automatique. Le jeu a été réalisé en binômes et en python à l'aide de la bibliothèque fltk.

•Principe du jeu :

Le but du jeu est simple, on commence avec une grille carré/rectangulaire comportant soit des cases vides soit des cases avec des chiffres allant de 0 à 3.

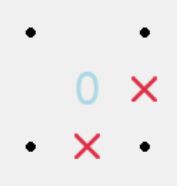
Quand on clique avec le clique gauche de la souris entre deux points, un segment apparaît.



Le but est de satisfaire toutes les cases numérotées avec des segments. Chaque case numérotée doit avoir le même nombre de segments adjacents que son chiffre. (Ex: Une case 2 doit avoir 2 segments adjacents)
C'est en partant de ce principe que le joueur doit satisfaire toutes les cases numérotées et réaliser une boucle pour gagner.
Il peut utiliser autant de coups qu'il souhaite et n'a pas de temps limite pour gagner.

•Manuel utilisateur :

Le joueur peut utiliser le clique gauche et le clique droit pour jouer seulement.
-Lorsque l'indice d'une case est satisfait, cette case apparaît en bleu.
-Si la case n'est pas encore satisfaite elle apparait en **noir**
-Si le nombre de segments adjacents à la case dépasse son indice alors la case apparaît en rouge.



Le joueur peut s'aider du clique droit de la souris qui fait apparaître une croix à l'endroit cliqué. Cela permet au joueur de se rappeler à quels endroits il n'est pas possible d'avoir un segment.

•Résolution de la grille :

Si le joueur est en difficulté et qu'il veut voir la solution de la grille, il peut utiliser un solveur automatique de la grille en appuyant sur **P**. Ce solveur fonctionne récursivement c'est-à-dire qu'il part d'un sommet puis va tracer un segment entre ce sommet et un nouveau sommet adjacent.

Si le segment tracé fait dépasser le nombre de segments autorisés par une case contenant un indice, alors il revient en arrière pour tracer un autre segment. Et tout cela jusqu'à ce que le solveur ait résolu la grille.

•Étapes de programmation :

Concernant la répartition des tâches. Pour ce projet, il y avait 4 tâches principales à réaliser. Nous avons commencé par programmer les fonctions de la tâche 1 à deux.
Pareillement pour la tâche 2. Pour l'interface graphique, l'un s'en est occupé tandis que l'autre s'est occupé de la gestion des clics. Et enfin, nous nous sommes mis à deux pour la programmation de l'algorithme de résolution de la tâche 4 l'un commençant par programmer les bases de la fonction et l'autre s'est chargé de finir cette fonction.
Il n'y a pas vraiment eu de répartition entre nous. A chaque fois, on faisait des appels sur discords pour travailler ensemble et lorsque l'un de nous avait des bugs/problèmes avec une fonction, on essayait de le corriger à deux.

•Problèmes rencontrés :

La grande difficulté de ce projet a été la réalisation de l'algorithme de résolution.

Pour la tâche 4, le problème était que le solveur prenait beaucoup de temps à résoudre une grille. Le problème était que lorsque le solveur traçait un nouveau segment il ne vérifiait pas s' il dépassait l'indice d'une case.

C'est-à-dire qu'il traçait des segments et vérifiait si les indices de toutes les cases étaient satisfaits seulement lorsqu'une boucle était tracée. Pour résoudre ce problème, on a rajouté la condition verif_IndicesExces() qui permet de savoir si le nouveau segment tracé fait dépasser l'indice d'une case.
Et si c'est le cas, alors ce segment est retiré et un autre segment est testé.

•Tâches complémentaires :

Nous avons choisi d'implémenter 2 tâches optionnelles. Le solveur graphique permettant de voir comment l'algorithme de résolution procède pour trouver la solution.
Et on a implémenté un système de score et de temps et aussi à chaque fois que le joueur finit une grille, son score en nombre de coups et son temps de jeu est écrit dans un fichier texte.

•Concernant la résolution graphique, avant de lancer le solveur avec la touche **P** le joueur peut s' il le souhaite activer le **mode graphique** permettant de voir comment le solveur procède pour trouver la solution en appuyant sur **G**

A tout moment pendant la résolution de la grille par le solveur, le joueur peut désactiver le mode graphique en appuyant sur la touche **D**.
Pendant l'affichage, le joueur peut ralentir le programme ou l'accélérer en appuyant respectivement sur **A** ou **Z.**
A pour ralentir l'affichage e**t Z** pour l'accélérer.

Pour réaliser cette tâche, nous avons créé deux fonctions ModeGraphique() et SpeedAffichage(). Chaque fonction gère les touches permettant de gérer les parties qui leur sont associées. Ensuite, dans la fonction algo_backtracking() on regarde si une touche est cliquée. Si le mode graphique est activé, alors on dessine les segments.

•Concernant le système de score et de temps, c'est très simple à mettre en place.

Nous avons utilisé une liste pour stocker le score, et lorsque le joueur clique sur un segment, on incrémente de 1 seulement le premier élément de la liste.

Pour le choix du mode graphique, la vitesse d'affichage du mode graphique et le système de score, on a utilisé des listes bien qu'il y ait un seul élément car les variables globales booléen ne nous permettait pas de changer la valeur de la variable dans une fonction. Avec une liste on peut très bien la modifier dans une fonction.

Et concernant le temps, lorsque la partie commence, on sauvegarde le temps actuel dans une variable et quand la partie se finit, on fait la différence entre le temps de début de la partie et le temps actuel, ce qui donne le temps mis par le joueur pour finir la grille.