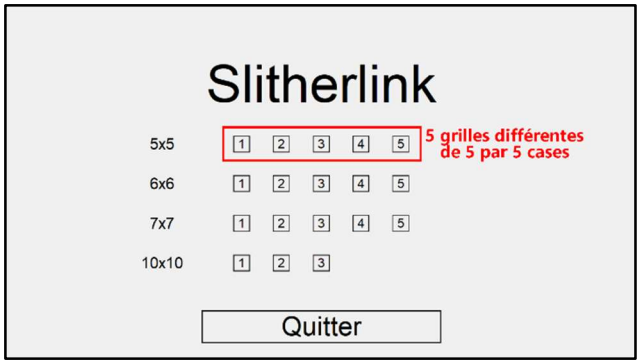
Slitherlink

❖ Guide Utilisateur ❖

▶ But du jeu

Le **Slitherlink** est un jeu de réflexion dans lequel le joueur, sur une **grille**, doit tracer des lignes entre des points pour former une **boucle unique** en **respectant des indices** (de 0 à 3) qui indiquent combien de lignes entourent une case.

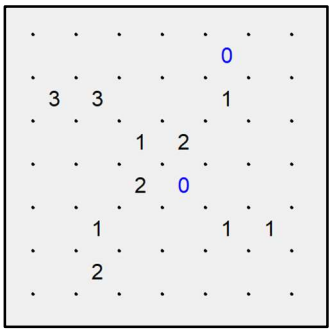
▶ Menu Principal



Au lancement du jeu, vous arrivez sur le menu principal qui vous propose plusieurs options sur lesquelles vous pouvez cliquer avec la souris :

- Un bouton **Quitter** qui fermera le jeu
- Des boutons avec des **chiffres de 1 à 5** qui correspondent aux 18 grilles de tailles différentes jouables

▶ Grille

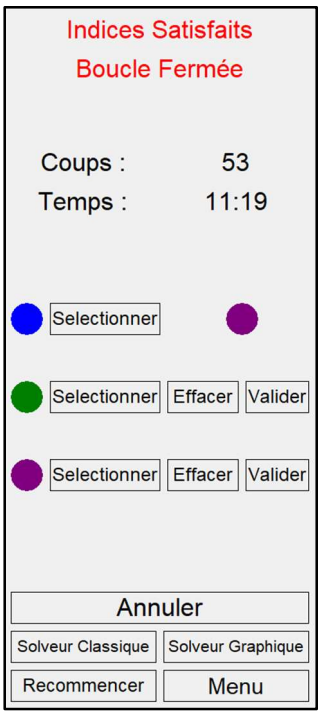


Dans la partie de gauche de l'interface se trouve la grille que vous allez devoir résoudre. Pour se faire, vous pouvez utiliser le **clic gauche** pour tracer ou effacer un **trait** et le **clic droit** pour mettre ou effacer une **croix** afin d'indiquer que vous êtes sûr qu'il n'y a pas de trait à cet endroit.

▶ Interface

Indices Satisfaits / Boucle Fermée : En haut de l'interface se trouve deux **conditions de victoire** qui doivent être remplies pour gagner. Celles-ci peuvent changer de couleur en fonction de l'état de la grille : **Bleu** si une condition est satisfaite, **Rouge** si elle ne l'est pas.

Coups / Temps : En dessous, on peut trouver une section dans laquelle on peut voir le **nombre de coups** que le joueur a effectué et le **temps écoulé depuis le début** de la partie.



Couleurs du Trait : Vers le milieu de l'interface, on peut trouver plusieurs boutons associé à une couleur permettant au joueur de faire des suppositions. En effet :

- Le bouton **Sélectionner** sert à **sélectionner une couleur**
- Le bouton **Effacer** sert à **effacer tous les traits** de la couleur sélectionnée
- Le bouton **Valider** sert à **remplacer la couleur** sélectionnée par du **Bleu**.

La couleur sélectionnée est affichée en haut à droite de la zone de sélection de couleur.

Annuler : Le bouton annuler sert à **annuler le dernier coup** du joueur.

Solveur Classique/Graphique : Le *Solveur Classique* permet d'**afficher directement la solution** de la grille. Le *Solveur Graphique* permet la même chose que le solveur classique mais permet de suivre les étapes de résolution de la grille.

Recommencer : Ce bouton permet de **recommencer une partie**. Ainsi la grille devient vide, le nombre de coups et le temps écoulé se réinitialisent.

Menu : Ce bouton permet de **revenir sur le menu**.

▶ Console

Lorsque le solveur (graphique ou non) trouve une solution, un message en console apparaît. Il indique le **temps, en secondes, de résolution** de la grille par le solveur.

❖ Etat d'Avancement ❖

La réalisation du projet a été assez efficace et nous avons réussis à finir toutes les tâches obligatoires :

▶ Tâche 1 : Structure de données

Le **chargement** de différentes **grilles en .txt** est possible sous la forme d'une **liste de listes d'indices**. Le **dictionnaire** `etat` qui représente l'**état des segments** fonctionne lui aussi, bien qu'il ait été modifié afin de permettre le tracé dans **différentes couleurs** (*voir améliorations*). Enfin, toutes les **fonctions d'accès** proposées par le sujet ont été implémentées dans le module **`fonctions_acces.py`**.

▶ Tâche 2 : Conditions de victoire

Les **conditions de victoire** sont correctement détectées à l'aide de :

- une fonction **`indices_satisfaits`** qui vérifie si les **indices sont satisfaits**
- des fonctions **`longueur_boucle`** et **`boucle_ferme`** qui vérifient si les segment forment bien une **unique boucle fermée**
- une fonction *victoire* qui vérifie si les **deux conditions précédentes sont réalisées**

L'**affichage des conditions de victoire**, proposé sur la console dans le sujet, est effectué directement sur l'**interface en jeu**.

▶ Tâche 3 : Interface graphique

La partie graphique fonctionne intégralement, la **détection de clic** est précise et ergonomique et l'**affichage** permet de savoir distinctement si les **indices sont satisfaits ou non**. De plus, certains ajouts ont été faits comme une **interface** ou bien la possibilité de **changer la couleur des traits** (*voir améliorations*).

▶ Tâche 4 : Recherche de solutions

La **recherche de solution** a été réalisée dans le module **`solveur.py`** et comprends toutes les fonctionnalités obligatoires :

- Le **choix du point de départ** est fait comme expliqué dans le sujet (un sommet d'un 3 est prioritaire…).
- La **fonction récursive** trouve la solution avec un algorithme proche de celui proposé dans le sujet.

De plus, certains ajouts ont été faits comme un **solveur graphique** ou certaines fonctionnalités qui **optimisent la recherche** de solution. Notre solveur peut résoudre les deux grilles proposées **quasiment instantanément** en non graphique et prend moins d'une minute pour la résolution graphique. Nous avons néanmoins choisi de **ne pas activer le solveur sur les grilles de 10x10** car la résolution est très longue même en non graphique.

❖ Améliorations ❖

▶ Annulation

Pour permettre au joueur d'**annuler les coups** qu'il a fait précédemment, à chaque fois qu'il fait quelque chose (tracer un trait, effacer une croix…), on ajoute dans une liste **`historique`** son action sous ce format :
[segment, action, couleur] où

- **Segment** représente le segment qui a été modifié
- **Action** représente comment le segment a été modifié (tracer, effacer, interdire)
- Si le segment a été effacé, **couleur** indique la couleur du trait ou de la croix qui a été effacé

Quand le joueur annule, on effectue l'**inverse de l'action du dernier élément** de la liste **`historique`**. (Par exemple, s'il a effacé un trait, on dessine un trait). Puis on **efface le dernier élément** de **`historique`**. Cela permet de pouvoir annuler **autant de coups que l'on veut.**

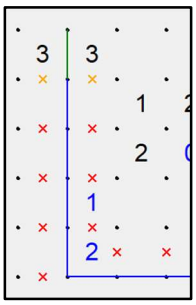
▶ Changement de couleur

Cette amélioration permet de **changer la couleur des traits et des croix** afin de **faire des suppositions**.
Pour cela, une variable **`couleur`** est modifiée lorsque l'on choisit une nouvelle couleur (Bleu → 1 / Vert → 2 / Violet → 3). Ce qui permet de **différencier les traits de différentes couleurs** dans le dictionnaire **`etat.`** En effet, chaque trait est associé à **`couleur`** et chaque croix est associée à **`-couleur`**.
Quand on **efface tous les traits** d'une couleur, on supprime toutes les clés de **`etat`** dont la valeur correspond au chiffre de la couleur.
Pour **valider nos suppositions**, on remplace les valeurs dans **`etat`** correspondant à la couleur choisie (2, 3, -2, -3) par 1 ou -1 correspondant à la couleur par défaut.

▶ Solveur graphique

Le **solveur graphique** permet de voir l'**évolution du solveur** dans sa recherche de la solution. Pour le réaliser, nous avons ajouté en paramètre de la fonction récursive du solveur un booléen **`graphique`** qui vaut **`True`** si l'affichage doit être effectué. Si ce booléen vaut **`True`**, on appelle la fonction d'affichage générale à la ligne 222 du module **`solveur.py`**, ce qui va permettre d'actualiser l'affichage à chaque appel de la fonction récursive.

▶ Solveur amélioré



Pour **améliorer les performances** de notre solveur, nous avons modifier le code de notre fonction récursive afin d'**éviter un maximum de traits inutiles**. Pour cela, à chaque fois que l'on va tracer un trait, on va aussi mettre des **croix sur les côtés du trait** car il est impossible de tracer un trait sinon cela ferait une boucle.
De plus, on va vérifier si les indices des cases à côté des croix sont encore possibles à satisfaire. Sinon la solution ne peut pas être bonne et on passe directement à la possibilité suivante.

▶ Autres améliorations

Interface : On a ajouté sur la droite de l'écran une interface qui **donne des informations sur la partie** et qui permet d'**actionner certaines fonctionnalités** comme le solveur ou le choix des couleurs. Pour cela, on a juste détecté des clics dans des rectangles.

Recommencer : Le joueur a la possibilité de recommencer. Pour cela, lors du clic sur le bouton correspondant, on **vide le dictionnaire `etat`** et on remet à zéro le compteur des coups ainsi que le temps écoulé depuis le début de la partie.

Score : A chaque action du joueur (tracer un trait, effacer une croix, annuler…) on ajoute 1 à une variable qui **compte le nombre de coups**. Le contenu de la variable est par la suite **affiché à l'écran**.

Temps : Au lancement d'une partie, on enregistre dans une variable l'heure de début avec **`time.time()`**. On affiche à l'écran la **différence entre l'heure actuelle et l'heure de début** afin de voir le temps de la partie.

❖ Fonctions Importantes ❖

Voici quelques fonctions importantes classées dans un tableau. Pour en savoir plus sur le fonctionnement et implémentation d'une fonction, consultez ses docstrings et commentaires.

Slitherlink.py	affichage.py	fonction_acces.py	utilitaire.py	victoire.py	solveur.py
evenement_clic_gauche	affiche_coin affiche_chiffre affiche_segment	sommets_adjacents segments_adjacents statut_case	detection_clic_segment chargement_grille	indices_satisfaits longueur_coucle	solveur verification_case recursive

❖ Méthode de Travail ❖

Pour commencer, nous avons décidé de travailler assez tôt pour se libérer du temps afin de nous permettre de mieux nous préparer pour les examens. En effet, dès les vacances de printemps, nous avions déjà effectué les **tâches 1, 2, 3 du projet**.
Pour ces tâches, nous avions décidé de les faire ensemble puisque ces tâches étaient le cœur du projet. Ainsi, cela nous a permis d'avoir une **bonne base**, une bonne compréhension du programme ensemble.

Ensuite, comme nous avons avancé rapidement sur le programme, nous avons fait une pause pour ensuite reprendre le projet afin de réaliser la **4ème tâche, le solveur**.
A partir de ce moment, nous avons plus divisé le travail de façon **équitable** (50%-50%) afin d'avancer un peu plus vite sur des parties différentes du projet.
Dans un premier temps, nous avons fait une **première version d'un solveur** qui résolvait, en non graphique, la grille 1 en **quelques secondes** (~10 secondes) et la grille 2 **presque instantanément**. En graphique nous prenions autant de temps que celui du solveur partagé par un professeur sur Youtube (**~10 min**).

Puis après la consultation de Monsieur Meyer, nous avons **amélioré le solveur** en introduisant les segments interdits. Ainsi le temps de résolution en non graphique et en graphique a considérablement diminué. Les grilles 1 et 2 se **résolvent maintenant instantanément** en non graphique et en **quelques dizaines de secondes** en version graphique.
Enfin, après avoir instauré une interface plus intéressante et introduit quelques améliorations, nous avons fait le rapport et rendu le projet dans les délais.