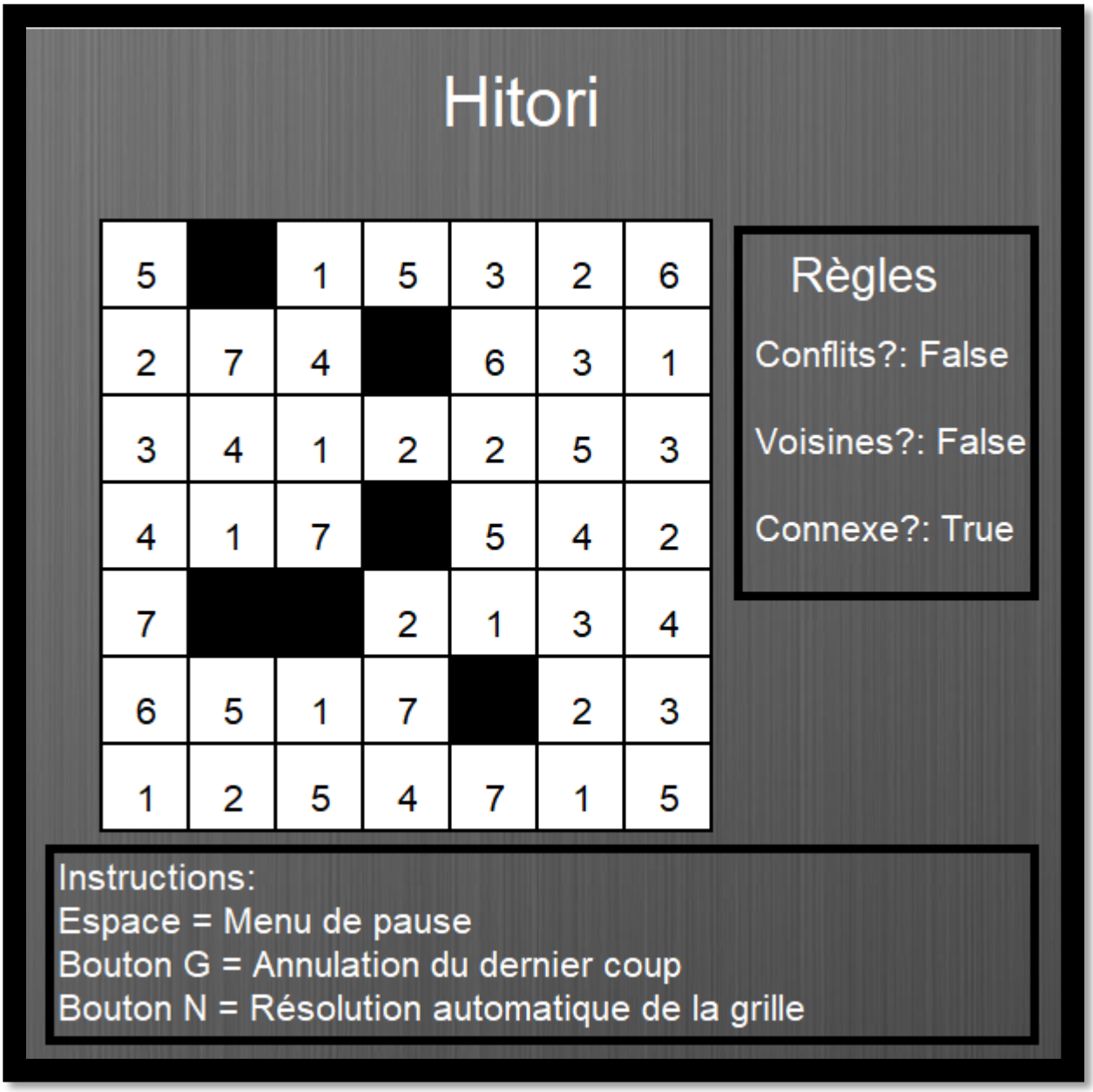
Hitori



<u>Diagne Ben :</u> Licence Maths/Info – Première année – UPEM - TD C – TP 4

<u>Belabbas Sofiane :</u> Licence Maths/Info – Première année – UPEM - TD C – TP 4

Présentation

Le jeu de l'Hitori (traduit en français « Laissez-moi tranquille ») est un casse-tête japonais prenant place dans une grille de jeu avec chacune des cases qui contiennent une valeur. L'objectif du joueur est de noircir ces cases pour faire en sorte de respecter les trois règles de l'Hitori :

-Aucune cellule similaire en conflit pour chaque ligne et colonne,

-Aucune cellule noircie voisine,

-Uniquement une seule zone visible présente.

La grille est complétée si le jeu respecte ces trois règles. Le joueur, de part pouvoir noircir les cases, peut annuler le dernier coup effectué, mais aussi avoir le choix d'utiliser le solveur automatique pour avoir la réponse à une grille.

Manuel

(En menu)

Naviguer dans le menu = Touches directionnelles

Sélectionner un mode = Entrée

(En jeu)

Noircir / Dénoircir une case = Clic Gauche

Menu de Pause = Espace

Annuler un coup = G

Solveur automatique = N

Sur quelles idées sommes-nous basées ?

Pour écrire notre programme, on s'est principalement basé sur upemtk, un module qui nous a été disposé pour la base du jeu. Notre programme s'est aussi basé sur plusieurs programmes écrits précédemment, comme le TP 5 du Klickety qui nous a beaucoup aidé à établir l'une des trois règles de l'Hitori, mais aussi le projet du semestre 1 (Bataille navale) pour la mise en place des menus par exemple.

Le jeu

Le programme est découpé en quatre tâches : Le chargement des niveaux, la mise en place des règles, l'interface graphique et le solveur. Les bases de ces quatre tâches sont contenues dans des fonctions qui seront appelés dans la boucle principale du jeu.

Tâche 1 : Représentation et chargement des niveaux

Cette tâche consiste à lire les valeurs d'une grille dans un fichier texte et la convertir en une liste de liste qui va être notre base de données principale.

Lire_grille(nom_fichier) (Fonctionne)

" " " Fonction qui renvoie une liste de listes décrivant les valeurs des cellules de la grille. " " "

- ➔ On fait en sorte de stocker tous les caractères du fichier *nom_fichier* dans une liste.
- ➔ On parcourt cette liste avec une boucle imbriqué : Si le caractère parcouru est un chiffre, on l'ajoute à une liste.
- ➔ A chaque fois qu'on passe par la première boucle, on ajoute la liste complétée à une autre liste pour qu'on tombe à la fin sur une liste de listes qu'on retournera.

Cette liste qu'on retourne va donc être la base de données de la grille qu'on nommera *grille*.

On a aussi deux autres fonctions pour le confort : Afficher_grille et Ecrire_grille, qui fonctionnent correctement.

Tâche 2 : Réalisation et moteur du jeu

Cette tâche nous invite à créer et appliquer les règles de l'Hitori. C'est ici qu'on introduit la liste *noircie.*

Première règle : Pas de conflit de cellules entre lignes et colonnes.

Sans_conflit (grille, noircie) (Fonctionne)

" " "Fonction qui établit la première règle de l'Hitori " " "

- ➔ On sépare la fonction en deux fonctions qui vont respectivement s'occuper des lignes et des colonnes**.** Pour les deux fonctions, une boucle imbriquée est écrite pour parcourir chaque cellule d'une ligne ou d'une colonne.
- ➔ On renvoie **False** si deux mêmes chiffres figurent dans une liste ou figurent dans deux listes différentes dans la même indice et **True** sinon.

Deuxième règle : Pas de cellules noircies voisines.

Sans_voisines (grille, noircie) (Fonctionne)

" " "Fonction qui établit la seconde règle de l'Hitori " " "

- ➔ On crée une boucle imbriquée qui parcourt chaque cellule de la grille.
- ➔ On vérifie d'abord si la cellule parcourue appartient à la liste *noircie,* puis on vérifie si ses 4 cases voisines appartiennent aussi à *noircie.*
- ➔ On renvoie **False** si c'est le cas, ou **True** si le cas échéant figure pour toutes les cellules.

Troisième règle : Une seule zone visible dans la grille.

Recherche_cases_reliées (grille, noircie, liste, x=0, y=0)

" " " Fonction qui renvoie une liste des coordonnées de toutes les cases blanches reliés par ses quatre coins." " "

- ➔ On vérifie si les coordonnées (x, y) de la cellule appartiennent à noircie
- ➔ Si ce n'est pas le cas, on l'ajoute à la liste (nommé *liste)* des cases blanches.

➔ On fait ensuite un appel récursif pour les quatre coins de la cellule pour que les cellules suivantes suivent les mêmes conditions. On renvoie la liste à la fin.

Connexe (grille, noircie) (Fonctionne)

" " " Fonction qui établit la troisième règle de l'Hitori " " "

➔ On prend le nombre de cases noires du jeu en prenant la taille de *noircie,* puis le nombre de cases blanches reliés en prenant la taille de la liste renvoyée par la fonction *recherche_cases_reliées* qu'on appelle.

➔ On vérifie si l'addition de ces deux nombres est égale au nombre total de cases de la grille. On renvoie **True** si oui et **False** sinon.

Tâche 3 : Interface graphique

Les menus

Affiche_menu (position) :

" " " Fonction qui affiche le menu du jeu. " " "

➔ On initialise d'abord la coordonnée de position du curseur

➔ On établit une condition avec la coordonnée de position pour modifier les coordonnées du rectangle dont ses valeurs dépendront de la coordonnée de position : Par exemple : si position = 0, le curseur serait tout en haut du menu.

Curseur_menu (position, touche) :

" " " Fonction qui gère les déplacements du curseur du menu. " " "

➔ On initialise d'abord la variable "position", qui est la coordonnée du curseur.

➔ On établit aussi une condition : Si l'évènement saisi est une touche directionnelle, alors la coordonnée de position du curseur va être modifié pour respecter la direction de la touche.

➔ La fonction va alors retourner les coordonnées modifiées pour ensuite être réutilisé par la fonction affiche_menu dans la boucle principale.

Le jeu

Dessine_grille (grille, noircie) :

" " "Fonction qui dessine la grille " " "

➔ On parcourt la liste *grille* avec une boucle imbriquée. Si les coordonnées de la cellule parcouru appartiennent à *noircie*, on crée un rectangle noir. Sinon, on crée un rectangle blanc.

Changer_état (x, y, grille, noircie) :

" " " La fonction responsable des changements d'états d'une cellule. " " "

➔ On parcourt la liste *grille* une nouvelle fois avec une boucle imbriquée. Si les coordonnées de la cellule parcouru est égale aux coordonnées du clic (x, y), deux

actions sont possibles : Si la cellule parcourue n'est pas dans la liste *noircie*, on l'ajoute dans celle-ci. Sinon, on la retire de la liste.

➔ On renvoie alors la liste *noircie* modifiée.

Annuler (noircie, stockage) :

" " " La fonction responsable des annulations de coup." " "

On introduit une nouvelle liste de listes *stockage* qui est là pour contenir l'historique du jeu. A chaque action dans la grille, on ajoute la liste *noircie* à *stockage.*

➔ Quand la fonction est appelée, on retire la mémoire actuelle du jeu en retirant la dernière liste de la liste stockage et *noircie* prend la toute dernière valeur de la liste *stockage* modifié.

➔ On renvoie alors *noircie.*

Tâche 4 : Solveur automatique

Résoudre (i, j, grille, noircie) (Ne fonctionne que pour le niveau 1) :

" " " Fonction qui gère le solveur automatique d'une grille " " "

On a élaboré un algorithme de recherche qui consiste en étudier la première règle de l'Hitori pour chaque cellule de la grille dans le cas où la deuxième et troisième règle est vraie et on ajoute ou retire les coordonnées (i, j) de la cellule de la liste *noircie* selon les résultats. On renvoie **None** si aucune solution n'a été trouvé et *noircie* sinon.

Répartition du travail

Ben s'est occupé du chargement des niveaux, l'interface graphique et la mise en place de la troisième règle de l'Hitori tandis que Sofiane s'est occupée de la tâche 2 à sa totalité. Nous nous sommes alliés pour faire la tâche 4.

Diagne Ben : 60 %

Belabbas Sofiane : 40 %

Difficultés rencontrées

Notre principale difficulté fût l'élaboration de la fonction connexe. Comparé aux deux autres règles, l'idée était non seulement dure à saisir, mais aussi à mettre en place. En effet, la fonction récursive à établir n'était pas évidente. Gérer cette fonction nous a permis de mieux comprendre comment marche la récursivité. Un autre problème qu'on n'a pas pu régler à temps est la consistance du solveur automatique.