25/04/2024
INM-1

GEMICIOGLU Utku
GULTEKIN Kivanc
UNAY Arhan

RAPPORT DU PROJET INF402

HISTOIRE DU JEU

Futoshiki (不等式), également appelé “Plus ou Moins”, est un jeu de logique originaire du Japon. Son nom signifie “inégalité”. Il est également orthographié “hutosiki” (en utilisant la romanisation Kunrei-shiki). Futoshiki a été développé par Tamaki Seto en 2001. Ce jeu est arrivé en Europe à partir de la fin 2006.

On retrouve dans plusieurs journaux nationaux et qui a fait sa première apparition dans les éditions du samedi du journal “The Guardian”. Les journaux britanniques suivants publient aussi des grilles de Futoshiki : The Daily Telegraph, Dundee Courier, i, The Times.

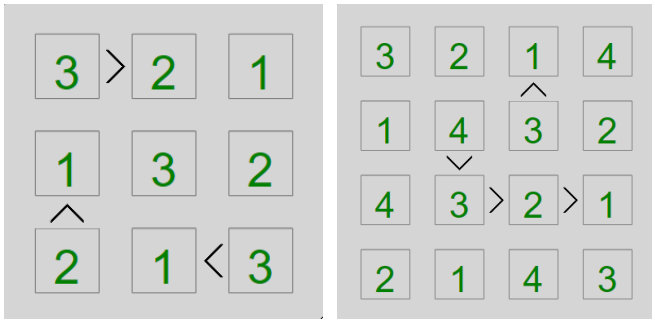
COMMENT JOUER FUTOSHIKI (REGLES ET CONTRAINTES)

Nous avons une grille carrée ; chaque carré peut contenir un chiffre de 1 à la taille de la grille, et certains carrés ont des signes d'indice entre eux. Notre objectif est de remplir entièrement la grille avec des chiffres tels que :

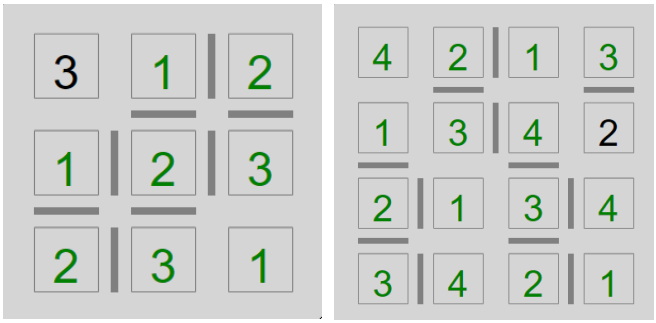
- Chaque ligne ne contient qu'une occurrence de chaque chiffre
- Chaque colonne ne contient qu'une occurrence de chaque chiffre
- Tous les signes d'indice sont satisfaits.

Il y a deux modes pour ce jeu, 'Inégal' et 'Adjacent'.
En mode 'Inégal', les signes d'indice sont des symboles supérieurs indiquant que la valeur d'un carré est supérieure à celle de son voisin. Dans ce mode, tous les indices peuvent ne pas être visibles, en particulier à des niveaux de difficulté plus élevés.
En mode 'Adjacent', les signes d'indice sont des barres indiquant que la valeur d'un carré est numériquement adjacente (c'est-à-dire supérieure ou inférieure d'une unité) à celle de son voisin. Dans ce mode, tous les indices sont toujours visibles : l'absence d'une barre signifie donc que la valeur d'un carré n'est définitivement pas numériquement adjacente à celle de son voisin.

EXEMPLES POUR LE MODE ‘INEGAL’



EXEMPLES POUR LE MODE ‘ADJACENT’



MODELISATION

Chaque cellule doit contenir un chiffre : Pour chaque cellule (i, j) de la grille, on peut avoir une variable booléenne X_{ijk} pour chaque chiffre k possible. La contrainte est alors que pour chaque cellule, exactement un de ces X_{ijk} doit être vrai. Cela peut être exprimé par une clause disjonctive (OR) pour chaque k, et une clause conjonctive (AND) pour chaque paire de k.

$$\bigwedge_{i=1}^{n}\bigwedge_{j=1}^{n}\left(\bigvee_{k=1}^{n} x_{ijk}\right)\wedge\bigwedge_{k\neq l}\neg(x_{ijk}\wedge x_{ijl})$$

Nous obtenons la modélisation complète de la première contrainte pour la cellule (1,1) :

$$(x_{111}\vee x_{112}\vee x_{113})\wedge\neg(x_{111}\wedge x_{112})\wedge\neg(x_{111}\wedge x_{113})\wedge\neg(x_{112}\wedge x_{113})$$

Et puis nous avons appliqué ce que nous avons fait ci-dessus pour une grille 3x3 et nous avons obtenu pour toutes les cellules la modélisation suivante :

$$\begin{aligned}&(x_{111}\vee x_{112}\vee x_{113})\wedge\neg(x_{111}\wedge x_{112})\wedge\neg(x_{111}\wedge x_{113})\wedge\neg(x_{112}\wedge x_{113})\wedge\\&(x_{121}\vee x_{122}\vee x_{123})\wedge\neg(x_{121}\wedge x_{122})\wedge\neg(x_{121}\wedge x_{123})\wedge\neg(x_{122}\wedge x_{123})\wedge\\&(x_{131}\vee x_{132}\vee x_{133})\wedge\neg(x_{131}\wedge x_{132})\wedge\neg(x_{131}\wedge x_{133})\wedge\neg(x_{132}\wedge x_{133})\wedge\\&(x_{211}\vee x_{212}\vee x_{213})\wedge\neg(x_{211}\wedge x_{212})\wedge\neg(x_{211}\wedge x_{213})\wedge\neg(x_{212}\wedge x_{213})\wedge\\&(x_{221}\vee x_{222}\vee x_{223})\wedge\neg(x_{221}\wedge x_{222})\wedge\neg(x_{221}\wedge x_{223})\wedge\neg(x_{222}\wedge x_{223})\wedge\\&(x_{231}\vee x_{232}\vee x_{233})\wedge\neg(x_{231}\wedge x_{232})\wedge\neg(x_{231}\wedge x_{233})\wedge\neg(x_{232}\wedge x_{233})\wedge\\&(x_{311}\vee x_{312}\vee x_{313})\wedge\neg(x_{311}\wedge x_{312})\wedge\neg(x_{311}\wedge x_{313})\wedge\neg(x_{312}\wedge x_{313})\wedge\\&(x_{321}\vee x_{322}\vee x_{323})\wedge\neg(x_{321}\wedge x_{322})\wedge\neg(x_{321}\wedge x_{323})\wedge\neg(x_{322}\wedge x_{323})\wedge\\&(x_{331}\vee x_{332}\vee x_{333})\wedge\neg(x_{331}\wedge x_{332})\wedge\neg(x_{331}\wedge x_{333})\wedge\neg(x_{332}\wedge x_{333})\end{aligned}$$

Chaque chiffre apparaît exactement une fois par ligne et par colonne : Pour chaque ligne i et chaque chiffre k, on peut avoir une clause disjonctive qui stipule qu'au moins un des X_{ijk} pour chaque colonne j est vrai. Faites de même pour chaque colonne et chaque chiffre. De plus, pour chaque paire de cellules dans une même ligne (ou colonne), on peut avoir une clause conjonctive qui stipule que X_{ijk} et X_{ijk} ne peuvent pas être vrais en même temps.

$$\bigwedge_{i=1}^{n}\bigwedge_{k=1}^{n}\left(\bigvee_{j=1}^{n} x_{ijk}\right)\wedge\bigwedge_{j\neq j'}\neg(x_{ijk}\wedge x_{ij'k})$$

Pour la première ligne : $(x_{111}\vee x_{121}\vee x_{131})\wedge\neg(x_{111}\wedge x_{121})\wedge\neg(x_{111}\wedge x_{131})\wedge\neg(x_{121}\wedge x_{131})$

$(x_{112}\vee x_{122}\vee x_{132})\wedge\neg(x_{112}\wedge x_{122})\wedge\neg(x_{112}\wedge x_{132})\wedge\neg(x_{122}\wedge x_{132})$

$(x_{113}\vee x_{123}\vee x_{133})\wedge\neg(x_{113}\wedge x_{123})\wedge\neg(x_{113}\wedge x_{133})\wedge\neg(x_{123}\wedge x_{133})$

On fait la même chose pour la 2ème (en utilisant X_{2jk}) et la 3ème ligne (X_{3jk}) et on lie toutes les expressions avec un connecteur 'ET'. Maintenant, tous ce qu'on a fait en haut, on va les refaire pour toutes les lignes d'une colonne et connecter avec 'ET'.

Pour première colonne : $(x_{111}\vee x_{211}\vee x_{311})\wedge\neg(x_{111}\wedge x_{211})\wedge\neg(x_{111}\wedge x_{311})\wedge\neg(x_{211}\wedge x_{311})$

$(x_{112}\vee x_{212}\vee x_{312})\wedge\neg(x_{112}\wedge x_{212})\wedge\neg(x_{112}\wedge x_{312})\wedge\neg(x_{212}\wedge x_{312})$

$(x_{113}\vee x_{213}\vee x_{313})\wedge\neg(x_{113}\wedge x_{213})\wedge\neg(x_{113}\wedge x_{313})\wedge\neg(x_{213}\wedge x_{313})$

On fait la même chose pour la 2ème (en utilisant X_{i2k}) et la 3ème colonne (X_{i3k}) et on lie toutes les expressions avec un connecteur 'ET'.

Respect des inégalités (supériorité – infériorité) : Pour chaque inégalité donnée dans la grille, on peut avoir une clause disjonctive qui stipule que si une cellule (i, j) est supérieure à une cellule (i', j'), alors pour chaque paire de chiffres k<l, X_{ijk} et $X_{i'j'l}$ ne peuvent pas être vrais en même temps.

$$\bigwedge_{(i,j) > (i',j')} \bigwedge_{k=1}^{n} \bigwedge_{l=k+1}^{n} \neg(x_{ijk} \wedge x_{i'j'l})$$

Supposons que nous ayons une inégalité qui stipule que la cellule (1,1) est supérieure à la cellule (1,2). Pour chaque paire de chiffres k<l, X_{ijk} et $X_{i'j'l}$ ne peuvent pas être vrais en même temps. Les clauses seraient :

En convertissant dans une forme normale conjonctive, on imagine comme si 'cellule (1,2) >= cellule (1,1)' et comme c'est faux, en prenant l'inverse de cette hypothèse, on arrive à ce qui est voulu des règles du jeu.

$$\neg(x_{111} \wedge x_{121}) \wedge \neg(x_{111} \wedge x_{122}) \wedge \neg(x_{111} \wedge x_{123}) \wedge \neg(x_{112} \wedge x_{122}) \wedge \neg(x_{112} \wedge x_{123}) \wedge \neg(x_{113} \wedge x_{123})$$

Maintenant, on fait la même formulation pour chaque signe d'inégalité et on les lie avec le connecteur 'ET'. A la fin, on distribue les négations à l'intérieur des parenthèses afin de transformer sous FNC.

Respect des adjacences : Pour chaque paire de cellules adjacentes (i, j) et (i', j') on peut avoir une clause disjonctive qui stipule que si X_{ijk} est vrai, alors ni $X_{ij(k-1)}$ ni $X_{ij(k+1)}$ ne peuvent être vrais, pour chaque k.

$$\bigwedge_{(i,j) \sim (i',j')} \bigwedge_{k=2}^{n-1} \neg(x_{ijk} \wedge x_{i'j'(k-1)}) \wedge \neg(x_{ijk} \wedge x_{i'j'(k+1)})$$

Comme on a fait pour la version 'inégal', on raisonne avec l'inverse, si 2 chiffres non successifs se trouvent côte à côte, on ne peut pas satisfaire le jeu puisque cela enfreint les règles du jeu.

$$\neg(x_{111} \wedge x_{121}) \wedge \neg(x_{111} \wedge x_{123}) \wedge \neg(x_{112} \wedge x_{121}) \wedge \neg(x_{112} \wedge x_{122}) \wedge \neg(x_{112} \wedge x_{123}) \wedge \neg(x_{113} \wedge x_{122}) \wedge \neg(x_{113} \wedge x_{123})$$

Maintenant, on fait la même formulation pour chaque signe d'adjacence et on les lie avec le connecteur 'ET'. A la fin, on distribue les négations à l'intérieur des parenthèses afin de transformer sous FNC.

En guise de conclusion; après avoir obtenu toutes les formules des 4 contraintes en forme normale conjonctive (ℓ_1, ℓ_2, ℓ_3, ℓ_4), on lie avec le connecteur 'ET' selon le mode du jeu.

Pour le mode 'inégal', on lie les formules ℓ_1, ℓ_2 et ℓ_3.

Pour le mode 'adjacent', on lie les formules ℓ_1, ℓ_2 et ℓ_4 car toutes ces conditions doivent être réalisées en même temps.

JEU.PY

Le fichier `jeu.py` contient le code principal de l'application du jeu Futoshiki. Il utilise le module Tkinter pour créer une interface graphique permettant au joueur d'interagir avec le jeu. Comme il s'agit d'un jeu japonais, nous avons ajusté le thème et la musique de notre application en conséquence.

Tout d'abord, le fichier importe les modules nécessaires, tels que `Tkinter`, `sys` et `pygame`, ainsi que les modules personnalisés `modelisation_2` et `mode_dimacs`, qui contiennent les fonctionnalités de modélisation et de résolution du jeu. Ensuite, le fichier initialise Pygame pour la gestion de la musique dans le jeu et récupère les paramètres de la ligne de commande, tels que la taille du tableau, le mode de jeu et le solveur à utiliser.
Une fois les paramètres récupérés, le fichier crée la fenêtre de jeu Tkinter avec un fond d'écran et initialise les fonctionnalités audio à l'aide de Pygame.

Puis, le fichier génère le tableau de jeu en fonction des paramètres fournis et l'affiche dans la fenêtre graphique à l'aide de la fonction `print_board`. Il crée également les boutons permettant au joueur d'interagir avec le jeu. Chaque bouton est associé à une fonctionnalité spécifique tels que le bouton ‘Resoudre’ pour voir la bonne réponse du jeu (si l'utilisateur souhaite résoudre le jeu à nouveau après avoir vu la réponse, les cadres des nombres passent à deux au lieu d'un pour indiquer clairement que la réponse a été examinée) , le bouton ‘Nouveau Jeu’ pour créer une table différente, le bouton ‘Effacer’ pour nettoyer la table lorsque l'utilisateur fait une erreur ou souhaite la résoudre à nouveau et les boutons de contrôle de la musique.

Selon le choix d’utilisateur (minisat ou notre solveur), les principes des quelques fonctions changent. Par exemple, les fonctions `valider_solution_sat` et `valider_solution` visent toutes deux à vérifier si une grille de solution respecte les contraintes imposées par les signes dans un jeu Futoshiki. Cependant, elles adoptent des approches distinctes pour atteindre cet objectif. La fonction `valider_solution_sat` utilise un solveur SAT (satisfiabilité booléenne) pour évaluer la satisfiabilité de la grille de solution. Elle traduit les contraintes des signes et la grille de solution en une forme normale conjonctive (FNC), puis ajoute ces clauses au solveur SAT pour déterminer si une solution satisfaisante existe. En revanche, la fonction `valider_solution` procède directement à la vérification des contraintes des signes sans recourir à un solveur SAT. Elle parcourt chaque cellule de la grille de solution et vérifie si les signes entre les cellules respectent les contraintes spécifiées. Ainsi, bien que les deux fonctions aient le même objectif, elles diffèrent dans leur méthode de validation, l'une utilisant un solveur SAT et l'autre effectuant une vérification directe des contraintes.

Pour un autre exemple, la fonction `generer_jeu_valide’ utilise une approche classique pour générer un jeu valide de Futoshiki en utilisant un solveur personnalisé. Elle génère d'abord une grille de jeu et des signes, puis crée des indices (hints) pour la grille générée. La validité du jeu est assurée en vérifiant que la solution générée respecte toutes les contraintes du Futoshiki. Contrairement, `generer_jeu_valide_sat’ utilise un solveur SAT pour garantir la validité du jeu généré. Elle génère une solution de jeu en utilisant le solveur SAT, puis crée des indices pour la grille générée. Comme elle repose sur un solveur SAT, cette approche assure une validité stricte du jeu en s'assurant que toutes les contraintes du Futoshiki sont respectées, sans recourir à une vérification supplémentaire.

LA FONCTION ‘print_board’ DANS LE FICHIER JEU.PY

La fonction print_board est chargée de gérer l'affichage du tableau de jeu dans l'interface graphique. Elle prend plusieurs paramètres (jeu_grille, signes, hint_grille, taille et affiche_tous) pour contrôler l'apparence du tableau.

Dans un premier temps, la fonction commence par afficher les signes de comparaison entre les cellules à l'aide de la fonction afficher_signes. Ces signes comprennent des symboles tels que ">" ou "<" pour les comparaisons horizontales, ainsi que "v" ou "^" pour les comparaisons verticales. Ensuite, la fonction affiche la grille de jeu elle-même en appelant la fonction afficher_grille. Cette grille peut afficher soit toutes les cellules avec leurs valeurs numériques, soit seulement les cellules vides si le paramètre affiche_tous est faux.

La fonction utilise des boucles for pour parcourir toutes les cellules de la grille de jeu. Selon la valeur de `affiche_tous`, elle affiche soit les valeurs numériques des cellules, soit des zones d'entrée pour que le joueur puisse saisir des chiffres dans les cellules vides. Pour chaque cellule, la fonction vérifie les signes de comparaison horizontaux et verticaux à l'aide des données fournies dans la liste `signes`. Elle utilise des étiquettes (`Label`) pour afficher ces signes à côté des cellules correspondantes.

Lorsque la cellule correspondante dans la grille des indices est vide, un champ de saisie est créé à l'aide du widget `Entry`. Ce champ de saisie est configuré avec divers paramètres pour assurer une apparence cohérente avec le reste de l'interface graphique. Le champ de saisie est placé dans la grille à la position appropriée, permettant ainsi à l'utilisateur de saisir un chiffre dans cette cellule. Pour garantir que l'utilisateur entre uniquement des chiffres dans le champ de saisie, une fonction de validation est définie. Cette fonction est appelée à chaque fois que l'utilisateur tape une touche dans le champ de saisie. Elle vérifie si la saisie est un chiffre valide en utilisant la méthode `register` du widget `Entry`. Ainsi, seuls des chiffres valides peuvent être saisis dans les cellules du jeu.

Enfin, la fonction crée un bouton "Valider" qui permet au joueur de valider sa solution. Ce bouton appelle une fonction `create_and_validate` qui vérifie si la solution proposée par le joueur est correcte. En fonction du résultat, un message de succès ou d'échec est affiché en bas du tableau.

MODELISATION.PY

Ce fichier contient un ensemble de fonctions pour la création, l'initialisation, la validation et la génération de grilles de jeu. La fonction creer_hint_grille génère un tableau d'indices pour aider les joueurs, en fonction du niveau de difficulté spécifié. Tout d'abord, elle initialise un tableau d'indices vide de la même taille que la grille de jeu d'origine. Ensuite, un choix aléatoire est effectué pour déterminer si des indices seront ajoutés à la grille et, le cas échéant, combien d'indices seront ajoutés.

La méthode utilisée pour ajouter des indices dépend du niveau de difficulté sélectionné. Par exemple, dans le mode "facile", un indice est ajouté pour chaque cellule de la grille, tandis que dans le mode "moyen", deux cellules reçoivent des indices, et dans le mode "difficile", trois cellules reçoivent des indices.

Pour déterminer les cellules à recevoir des indices, des cellules sont choisies au hasard dans la grille, et les valeurs de ces cellules sont copiées dans le tableau d'indices. Ce processus se répète pour le nombre requis de cellules, en veillant à ce que les cellules déjà choisies ne soient pas sélectionnées à nouveau.

Nous avons ajusté les modes de difficulté du jeu en nous basant sur des exemples trouvés sur le site de l'UE. Pour ce faire, nous avons calculé la proportion des signes pour chaque niveau de difficulté et nous avons adapté notre fonction initialisation_signes en conséquence. Cette fonction initialise une liste de signes en fonction du mode de jeu spécifié, en assignant des signes tels que ">", "<", "^" et "V" avec des proportions différentes selon le niveau de difficulté choisi. En intégrant ces ajustements, nous avons veillé à ce que chaque mode de difficulté offre une expérience de jeu équilibrée et conforme aux attentes des joueurs, en termes de complexité et de défi.

Les fonctions de ce fichier remplissent différentes fonctions dans la gestion du jeu. Elles permettent notamment d'initialiser une grille de jeu avec des zéros, de compter le nombre de signes de comparaison dans une grille de signes, de créer des indices pour les joueurs, d'initialiser une liste de signes de comparaison en fonction du mode de jeu spécifié, d'afficher les grilles, de mélanger les grilles, de vérifier la validité des chiffres dans les cellules par rapport aux contraintes des signes, de générer des solutions pour les grilles de jeu, de valider si une solution respecte les contraintes des signes, et enfin de générer des jeux de Futoshiki valides avec des grilles de jeu et de signes conformes au mode de jeu spécifié.

Notre solveur n'est pas un solveur SAT, mais plutôt un solveur basé sur le backtracking, qui teste les chiffres un par un jusqu'à trouver une solution. Autrement dit, lors de la création du jeu, nous vérifions s'il existe une solution (en fonction de sa satisfaisabilité), et nous générons un jeu avec une solution valide. Si un jeu sans solution est généré, nous réessayons jusqu'à obtenir un jeu satisfaisant.

Dans certains jeux, notamment ceux avec des grilles de grande taille, il peut y avoir plusieurs réponses possibles. Au lieu de comparer la réponse de l'utilisateur avec la réponse générée lors de la création du jeu, nous

soumettons la réponse de l'utilisateur à notre propre solveur et la soumettons à une phase de contrôle. Cela nous permet de vérifier si la réponse de l'utilisateur est valide, en garantissant ainsi une expérience de jeu cohérente et équitable, même pour les grilles les plus complexes.

En utilisant le backtracking, notre algorithme explore toutes les possibilités pour remplir les cellules de la grille, en respectant les contraintes imposées par les signes de comparaison et les règles du jeu. Ainsi, il trouve une solution en testant séquentiellement différentes combinaisons de chiffres pour chaque cellule, et s'assure que la solution finale est valide.

LA FONCTION 'generer_solution' DANS LE FICHIER MODELISATION.PY

La fonction `generer_solution` est cruciale pour créer une solution valide pour une grille de jeu Futoshiki. Elle utilise une technique de recherche récursive appelée backtracking pour explorer toutes les possibilités et trouver une solution qui respecte les contraintes du jeu.

Au début, la fonction initialise une grille de solution avec des zéros. Cette grille sera utilisée pour stocker les chiffres qui seront placés dans chaque cellule pour former la solution. Ensuite, elle appelle une fonction interne `backtrack` pour commencer le processus de recherche de solution.

La fonction `backtrack` est récursive et a un rôle de parcourir les cellules de la grille de manière systématique, en essayant différentes combinaisons de chiffres pour chaque cellule. Elle commence par vérifier si la ligne actuelle a été entièrement parcourue. Si c'est le cas, cela signifie que toutes les cellules ont été remplies avec succès, et la fonction retourne `True`, indiquant qu'une solution a été trouvée. Dans le cas contraire, la fonction `backtrack` détermine la prochaine cellule à remplir en se déplaçant d'abord horizontalement (colonne suivante dans la même ligne) puis verticalement (passage à la ligne suivante lorsque la fin de la ligne est atteinte).

Une fois la cellule sélectionnée, la fonction génère une liste de candidats possibles pour cette cellule en utilisant la fonction `est_numeros_valides` pour vérifier la validité de chaque chiffre candidat. Ces candidats sont triés en fonction de leur aptitude à remplir la cellule en cours, ce qui permet d'explorer les combinaisons les plus prometteuses en premier. Ensuite, la fonction parcourt chaque candidat et le place dans la cellule. Elle appelle ensuite récursivement elle-même pour traiter la cellule suivante. Si la fonction récursive retourne `True`, cela signifie qu'une solution a été trouvée, et la solution partielle est conservée. Sinon, la fonction annule le placement du candidat dans la cellule et continue à explorer les autres candidats.

Ce processus se poursuit jusqu'à ce que toutes les cellules soient remplies ou jusqu'à ce qu'il n'y ait plus de candidats possibles pour une cellule donnée. Si aucune solution valide n'est trouvée, la fonction retourne `None`, indiquant l'impossibilité de trouver une solution pour la grille donnée.

TABLE.PY

Le fichier `table.py` est un script Python qui utilise la bibliothèque Pygame pour créer et gérer l'interface graphique du menu principal du jeu Futoshiki. Pygame est une bibliothèque de développement de jeux en Python qui fournit des fonctionnalités pour créer des jeux interactifs.

Dans ce script, nous utilisons Pygame pour initialiser l'environnement graphique, définir la fenêtre du menu principal et gérer les événements utilisateur tels que les clics de souris. Le menu principal du jeu contient des boutons pour démarrer le jeu, activer/désactiver la musique et quitter le jeu. Les actions correspondantes sont déclenchées lorsque l'utilisateur interagit avec ces boutons.

Le script utilise également pour charger et jouer de la musique de fond, ainsi que des effets sonores pour les interactions utilisateur telles que les clics de bouton. La musique de fond crée une atmosphère immersive pour le menu principal, tandis que les effets sonores fournissent un retour audio lors des interactions utilisateur.

LA FONCTION ‘init_tkinter_()’ DANS LE FICHIER TABLE.PY

La fonction `init_tkinter_()` est essentielle pour créer une interface utilisateur fluide permettant à l'utilisateur de choisir les paramètres du jeu avant de démarrer. À travers cette fonction, une fenêtre Tkinter est créée pour guider l'utilisateur à travers les choix du solveur, du mode de jeu et de la taille du tableau.

Dans un premier temps, la fenêtre principale Tkinter est mise en place, affichant le titre "Choix du solveur" et une étiquette invitant l'utilisateur à faire son choix. Ensuite, lorsqu'un solveur est sélectionné parmi les options disponibles, la fonction réagit en ouvrant une nouvelle fenêtre pour choisir le mode de jeu. Cette fenêtre présente à son tour une sélection d'options, telles que "Entrainement", "Facile", "Moyen" et "Difficile".

Une fois le mode de jeu choisi, une autre fenêtre s'ouvre pour permettre à l'utilisateur de définir la taille du tableau de jeu. Cette étape est cruciale pour adapter l'expérience de jeu selon les préférences de l'utilisateur. Après que toutes les options ont été sélectionnées, la fonction lance le jeu avec les paramètres choisis. Cela assure une transition fluide de la configuration à l'expérience de jeu, offrant à l'utilisateur un contrôle complet sur son expérience.

MODE_DIMACS.PY

Ce fichier de code contient une série de fonctions essentielles pour générer, représenter et résoudre des grilles de Futoshiki en utilisant des méthodes de logique booléenne.

Dans ce fichier, on trouve tout d'abord des fonctions pour la création de grilles de Futoshiki valides. Ces grilles respectent les règles du jeu, notamment la disposition des chiffres dans les lignes et les colonnes ainsi que les contraintes de comparaison entre les cellules adjacentes. La génération de ces grilles se fait en tenant compte de la taille de la grille et du niveau de difficulté souhaité. La représentation des grilles se fait sous forme de matrices, où chaque cellule contient un chiffre et éventuellement un signe de comparaison avec les cellules voisines. Les signes de comparaison sont représentés par des symboles tels que '<' ou '>', et sont placés entre les cellules conformément aux règles du jeu. Pour résoudre ces grilles de Futoshiki, le fichier convertit les grilles et les contraintes en formules conjonctives normales (FNC), un format logique utilisé par les solveurs SAT (satisfiabilité booléenne). Les contraintes du jeu, telles que les comparaisons entre les cellules, sont traduites en clauses logiques qui peuvent être résolues par un solveur SAT. Enfin, une fois les contraintes converties en FNC, le solveur SAT est utilisé pour vérifier la satisfiabilité des clauses. Si une solution satisfaisant toutes les contraintes est trouvée, le solveur fournit une réponse valide, sinon il indique l'insatisfaisabilité de la grille.

Quelques Petites Explications Pour Les Fonctions de ‘mode_dimacs.py’

1. Contrôle des grilles de signes (controle_grille_signes) :

   Cette fonction vérifie si les signes placés dans la grille sont valides et se trouvent dans des cellules adjacentes. Elle parcourt chaque inégalité spécifiée dans la grille et vérifie si les signes sont valides ('<' ou '>'). Elle vérifie également que les inégalités sont placées entre des cellules adjacentes et ne comparent pas les mêmes cellules plusieurs fois.

2. Contrainte de cellule unique (contrainte_cellule_unique_bis) :

   Cette fonction génère les clauses pour garantir qu'une seule valeur est placée dans chaque cellule de la grille. Elle parcourt chaque cellule de la grille et génère des clauses pour garantir qu'exactement une valeur est vraie pour chaque cellule.

3. Contrainte de ligne et de colonne unique (contrainte_ligne_colonne_unique_bis) :

   Cette fonction génère les clauses pour garantir qu'une seule occurrence de chaque valeur est présente dans chaque ligne et chaque colonne de la grille. Elle parcourt chaque ligne et chaque colonne de la grille et génère des clauses pour garantir qu'exactement une valeur est vraie pour chaque ligne et chaque colonne.

4. Contrainte d'inégalités (contrainte_inegalites_bis) :
   Cette fonction génère les clauses pour prendre en compte les inégalités spécifiées dans le jeu Futoshiki. Elle parcourt chaque inégalité spécifiée et génère des clauses pour garantir que les comparaisons entre les cellules sont correctes selon les signes spécifiés.

5. Génération de la FNC (generer_fnc) :
   Cette fonction combine toutes les contraintes générées précédemment en une seule forme normale conjonctive (FNC). Elle utilise les fonctions précédentes pour générer les clauses correspondant à chaque contrainte, puis les combine en une seule FNC.

6. Conversion de la FNC au format DIMACS (fnc_a_dimacs) :
   Cette fonction convertit la FNC générée en un fichier au format DIMACS, utilisé par les solveurs SAT. Elle parcourt chaque clause de la FNC et les écrit dans un fichier texte au format DIMACS.

7. Conversion du fichier DIMACS au format SAT (dimacs_a_sat) :
   Cette fonction convertit un fichier DIMACS en un format SAT plus lisible, permettant de visualiser les clauses logiques. Elle lit le fichier DIMACS, puis organise les clauses de manière à ce qu'elles soient plus facilement lisibles et compréhensibles.

8. Conversion de DIMACS à 3-SAT (de_dimacs_a_3sat) :
   Cette fonction convertit un fichier DIMACS en une forme normale conjonctive (FNC) 3-SAT. Elle prend chaque clause du fichier DIMACS et la convertit en une clause 3-SAT, si nécessaire, afin de garantir que chaque clause contient au plus trois littéraux.

Nous avons eu des difficultés à télécharger et à intégrer un solveur SAT que nous pourrions charger et exécuter via un sous-processus pour notre projet d'entreprise. À la place, nous avons utilisé le solveur SAT fourni par le module pysat.solvers. Cependant, ce solveur SAT ne prend pas en entrée un fichier DIMACS, mais plutôt des clauses écrites en code. Par conséquent, nous avons dû écrire nos fonctions comme si nous utilisions un solveur SAT réel.

BIBLIOGRAPHIE

- https://www.chiark.greenend.org.uk/~sgtatham/puzzles/doc/unequal.html#unequal
- https://en.wikipedia.org/wiki/Futoshiki
- https://fr.wikipedia.org/wiki/Futoshiki
- https://www.futoshiki.com/