UNIVERSITÉ
CAEN
NORMANDIE

Rapport Aide à la résolution de Kakuro

Projet de substitution

PIGNARD Alexandre - 21701890
BOCAGE Arthur - 21806332

L3 Informatique - Promotion 2020-2021

1er juin 2021

Table des matières

1 Introduction

Le **Kakuro** est un jeu logique semblable aux mots croisés. Le jeu est originaire du Jpaon où sa popularité est immense. Le jeu est similaire aux mots fléchés dans lesquels une même combinaison de chiffres ne peut être utilisé deux fois dans la même grille. Bien que le jeu ne soit parvenu en France que vers les années 2004 et 2005 dans le sillage du sudoku, le jeu reste connu depuis longtemps.

Source : Wikipedia

Le but de ce projet est de créer un logiciel permettant de fournir une aide à la résolution de grilles de kakuro, il ne s'agit pas de proposer une résolution automatique mais de fournir des outils à un joueur afin qu'il résolve de lui même une grille de jeu.

Pour réaliser ce projet, nous avons choisi de l'écrire en Python, langage que nous maitrisions et qui nous offre un large panel de possibilités que cela soit pour réaliser les différentes fonctions dont nous avions besoin mais aussi pour les choix d'interfaces graphiques que nous avions à notre disposition grâce à ce langage.

2 Utilisation du programme

Pour utiliser notre programme, vous pouvez vous rendre dans le répertoire *Kakuro_Helper* et lancer la commande :

```
$ python3 Main.py
```

Afin d'éxecuter le logiciel il faut avoir installé Python 3 sur sa machine ainsi que PyQt5 et pickle, l'interface graphique que nous avons utilisés. Si ces dépendances sont installées, le logiciel devrait fonctionner sur Linux, Winwows et Mac Os.

3 Conception du programme : Back-end

Le projet est divisé en deux grosses parties, la première est le back-en et se trouve dans le répertoire du même nom. La deuxième est le front-end et se trouve elle aussi dans un répertoire qui porte son nom.

Le code présent dans le répertoire back-end constitue le cœur du logiciel. Il correspond aux méthodes qui vont permettre de créer un grille, y associer des valeurs à jouer ou encore une "heat map". Il s'agit du travail en arrière plan que l'utilisateur ne verra pas et qui sera utilisé par la partie front-end pour interagir avec l'utilisateur. Le back-end correspond à la partie que l'utilisateur ne verra pas, il s'agit du travail en arrière plan que le front-end va afficher à l'utilisateur.

Le back-end est constitué de deux sous parties : la partie Logic et les Ressources qui pourront être utilisées.

La partie Logic contient les fonctions du logiciel et la partie Ressources contient, comme son nom l'indique, les ressources nécessaires au bon fonctionnement du back-end (dictionnaire des sommes).

3.1 Grid_Logic : la création de la grille

Le fichier *Grid_Logic.py* est le fichier Python qui va permettre de créer la grille de Kakuro.
La grille est générée de façon assez classique : il s'agit de listes dans lesquels on trouve d'autres listes.
La méthode *grid_Maker_Creator* permet de créer une grille vide et la rempli avec une liste d'instructions.
Dans notre logiciel, une grille classique est constituée de six fois six cases, soit trente-six cases. Chaque case est en fait un tableau on a donc trente-six tableaux.

Il y a six types de case :

Case jouable/non jouable

Case de type "heat" (utilisée pour la heat map qui sera détaillée plus loin)

Liste des objectifs de la case

Liste des valeurs possibles de la case

Liste des valeures Contraintes

Case sélectionnée

3.2 Heat_Mapping_Logic : coloration d'une grille

Le fichier *Heat_Mapping_Logic.py* contient les fonctions qui vont nous permettre de réaliser une "heat map". Il s'agit de la coloration de la grille qui va permettre de mettre en avant certaines cases en fonction de leur potentiel. Concrètement, plus une case est intéressante à jouer plus elle sera colorée. Pour ce faire, dans le tableau une valeur sera attribuée en fonction du nombre de cases libres dans chaque lignes de la grille. Cette valeur est un poids, plus elle est élevée et plus la case sera colorée sur l'interface graphique.

Le fichier est constitué des fonctions suivantes :

heat_Mapping_Row : elle permet de parcourir les cases via un double for, quand elle trouve des cases libres dans une ligne elle incrémente le poids.

heat_Mapping_Column : même fonctionnement que la fonction ci-dessus mais parcours dans le sens des colonnes.

Set_Heat_Mapping : permet de définir le poids de bases et de retourner le Kakuro coloré.

3.3 Possible_Values_Mapping_Logic : guide du remplissage des cases

Le fichier *Possible_Values_Mapping_Logic.py* permet de créer le système de recommandation de coups à jouer. C'est ce qui permet d'afficher dans des cases vides la recommandation de chiffres à placer.

C'est l'un des fichier le plus important car il contient la fonction qui permet de charger la ressource sums.pkl.

Les fonctions *objective_Propagation_Row* et *objective_Propagation_Row* permettent de parcourir la grille (verticalement et horizontalement) et permettent d'attribuer des valeurs dans le Kakuro en fonction des lignes.
Ces deux fonctions sont executées par la methode *set_Objective_Propagation*.

Quant à la méthode $findCommonNumberForLenght$, elle permet de créer deux tableaux constitué des solutions possibles à la résolution d'une ligne de Kakuro et ce en parcourant la grille horizontalement et verticalement. Cette méthode est appelée dans *set_Possible_Values_Mapping*.

3.4 Saving_Logic : le fichier de sauvegarde

Sans le fichier *Saving_Logic.py* la sauvegarde et le chargement de Kakuro serait impossible car c'est ce fichier qui permet de les gérer.
Le code utilise le module pickle pour ce faire et utilise des fichiers au format *.pkl*.
Il y a deux méthodes de chargement : $load\_Default\_Save$ qui va permettre de charger un fichier du nom de "save.pkl". On ne peut pas choisir de fichier en particulier. La deuxième méthode de chargement (très similaire) est $load\_Kakuro\_From\_File$, qui prend en argument le nom d'un fichier et qui l'ouvre.
La méthode *push_Save* permet d'écrire un fichier au format *.pkl* du nom de *save.pkl*.

La méthode *dynamic_Load* permet de charger un fichier de la façon suivante :

- Vérifie si un fichier "save" est disponible

- Si aucun fichier n'est trouvé alors un Kakuro dit "basique" est crée (il s'agit d'un Kakuro tout prêt fait).

- En revanche, si un fichier "save" est trouvé, alors on le charge.

get_All_Saves permet quand à elle de renvoyer les nom de tous les fichiers présents dans le dossier *Saves*.

4 Conception du programme : Front-end

Dans notre logiciel, le front-end sert d'interface homme-machine afin de proposer une grille simple et de permettre à l'utilisateur d'interagir avec cette dernière via des boutons notamment.

L'interface graphique a été réalisée grâce à l'outil *PyQt*5. Cette dernière est divisée en plusieurs fenêtres (constituées de widgets) (que l'on appellera windows, de leur nom en anglais et dans les fichiers) et qui permettront d'exécuter des tâches spécifiques que nous détaillerons plus loin.

Voici la liste de ces fenêtres :

- Une fenêtre d'accueil permettant de sélectionner une grille

- Une fenêtre de jeu permettant d'avoir accès aux options d'aide à la résolution de la grille de Kakuro que l'on aura sélectionné.

- Une fenêtre de création de grille qui permettra de créer et sauvegarder sa propre grille de Kakuro.
  Enfin, c'est dans la partie *Front_End* que l'on trouve les dossiers de sauvegardes, à savoir *Save* et *Save_Creator_Result*.

Pour ce qui est du style de l'interface graphique nous avons opté pour un design simple et sombre (pour préserver le confort des yeux) qui permet de se concentrer sur la grille et les fonctionnalités proposées. On a donc une palette de couleurs allant du gris foncé au gris clair et du orange pastel pour la heat map.

4.1 Main_Window : initialisation de l'IU

Main_Window.py est le fichier qui va permettre d'initialiser l'interface graphique. Il importe le back-end ainsi que toues les windows et évidement les modules nécessaires au bon fonctionnement de *PyQt*.

Dans cette classe il y a quatres grandes méthodes :

- _init_
- startSolverWindow
- startCreatorWindow
- startLoaderWindow

Voici les détails de ces méthodes :

La méthode _init_ permet de paramétrer l'interface graphique et d'appeler le back-end. On y trouves les paramètres les plus importants comme la taille de la fenêtre, l'appel du dictionnaire et de la grille de Kakuro, les paramètres de styles généraux ainsi que le lancement de la fenetre *LoaderWindow*.

startSolverWindow, *startCreatorWindow* et *startLoaderWindow*, comme leur nom l'indique clairement, permettent de lancer les fenêtre de résolution de grilles, de création de grilles ainsi que la fenêtre de choix de grille.

Le fichier *Main_Window.py* est donc la base de l'interface graphique, nous allons maintenant détailler les trois autres window ainsi que les widgets qu'elles utilisent.

4.2 Solver_Window : la fenêtre principale

Si l'on ne devait garder qu'une seule fenêtre pour ce logiciel, ce serait evidemment la fenêtre *Solver_Window*. En effet il s'agit de la fenêtre centrale de l'application, celle qui propose les outils d'aide à la résolution de grilles de Kakuro.
L'affichage de la fenêtre est simple : en tête un menu permettant de choisir ou non l'affichage de la heat map et/ou des valeurs possible ainsi que la possibilité de charger un autre Kakuro et de se rendre sur la page de création de grilles.

Pour ce qui est du code au sein de *Solver_Window.py*, il est agencé de la manière suivante :

- Un constructeur
- Une fonction d'initialisation de la fenêtre qui appelle les widgets nécessaires
- Une méthode *createMenuLayout* qui permet de créer le menu de la tête de fenêtre
- Ainsi que la plus grosse méthode : *createKakuroSolverLayout* qui permet de créer la grille centrale et d'y interagir.

createKakuroSolverLayout est la méthode qui permet l'affichage de la grille au milieu de la fenêtre et qui permet d'appeler les fonctions d'aide fournies par le back-end.
Elle permet d'afficher la grille, d'afficher la heat map et de fournir les valeurs possibles dans les cases à jouer. Cette méthode contient aussi les styles de la grille et de la heat map.
Pour cette dernière nous avons utilisé un code couleur assez subtil : le *rgba*, ce dernière nous permet de gérer les traditionnelles valeurs rouge, vert, bleu mais aussi l'opacité des couleurs, ce dernier paramètre est particulièrement utile lors de l'utilisation de la heat map car on peut profiter du poids de chaque case pour adapter l'opacité de la couleur et donc l'intensité de la heat map.