



UNIVERSITÉ
CAEN
NORMANDIE

Rapport Aide à la résolution de Kakuro

Projet de substitution

PIGNARD Alexandre - 21701890
BOCAGE Arthur - 21806332

L3 Informatique - Promotion 2020-2021

June 2, 2021

Contents

1	Introduction	1
2	Utilisation du programme	1
3	Conception du programme : Back-end	2
3.1	Grid_Logic: la création de la grille	2
3.2	Heat_Mapping_Logic : coloration d'une grille	3
3.3	Possible_Values_Mapping_Logic : guide du remplissage des cases	4
3.4	Saving_Logic : le fichier de sauvegarde	4
4	Conception du programme : Front-end	7
4.1	Main_Window : La base de L'UI	8
4.2	Solver_Window : la fenêtre principale	9
4.3	Loader_Window, page de chargement de grilles	10
4.4	Creator_Window	10
4.5	Widgets utilisés	12
4.5.1	Content_Button_Widget	12
4.5.2	Divided_Label_Widget	13
4.5.3	Loader_Button_Widget	13
4.5.4	Morphing_Label	13
4.6	Dossiers de sauvegardes	13
5	Le logiciel dans son ensemble	14
6	Améliorations possibles	14
7	Annexes	15

1 Introduction

Le **Kakuro** est un jeu logique semblable aux mots croisés. Le jeu est originaire du Japon où sa popularité est immense. Le jeu est similaire aux mots fléchés dans lesquels une même combinaison de chiffres ne peut être utilisée 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 maîtrisons et qui nous offre un large panel de possibilités que cela soit pour réaliser les différentes fonctions dont nous avons besoin mais aussi pour les choix d'interfaces graphiques que nous avons 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'exécuter 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, Windows et Mac OS.

3 Conception du programme : Back-end

Le projet est divisé en deux grosses parties, la première est le back-end 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 une 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 remplir 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 éléments stockés dans une 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 valeurs Contraintes
- Cases sélectionnées

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 fichiers le plus important car il contient la fonction qui permet de charger la ressource *sums.pkl*.

Les fonctions *objective_Propagation_Row* et *objective_Propagation_Col* 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 exécutées par la méthode *set_Objective_Propagation*.

Quant à la méthode *findCommonNumberForLength*, elle permet de créer deux tableaux constitués 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 *PyQt5*. 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 : La base de L'UI

Main_Window.py est le fichier qui va permettre d'initialiser la fenêtre principale qui contiendra toutes les variables importantes du côté back-end pour la session actuelle et hébergera aussi la logique de navigation entre chaque fenêtre, en effet ,sous PyQt5, le changement dynamique d'une fenêtre n'est pas supporté avec l'architecture de base, c'est un initialisant un widget "Central" qu'on viendra override, puis ré-afficher que nous sommes arrivés à une interface dynamique.

- `_init_`
- `startSolverWindow`
- `startCreatorWindow`
- `startLoaderWindow`

Voici les détails de ces méthodes :

La méthode `_init_` permet de paramétrer l'interface graphique et d'instancier les variables back-end. On y trouve les paramètres les plus importants comme la taille de la fenêtre, les kakuro logiques, le dictionnaire des sommes, les paramètres de styles généraux ainsi que le premier appel à méthode d'instanciation de la fenêtre Loader, qui sera le point de départ pour l'utilisateur.

startSolverWindow, *startCreatorWindow* et *startLoaderWindow*, comme leur nom l'indique clairement, permettent de d'instancier les widgets des fenêtres de résolution de grilles, de création de grilles ainsi que la fenêtre de choix de grille et d'override le central widget avec le contenu venant d'être créer.

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 évidemment 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 sélection de filtres et de navigation
- Ainsi que la plus grosse méthode : *createKakuroSolverLayout* qui permet de créer la grille centrale.

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.

4.3 Loader_Window, page de chargement de grilles

Le fichier *Loader_Window* correspond à la page de chargement de grilles, cette page est une fenêtre très simple permettant de sélectionner la grille de Kakuro souhaitée : celle par défaut qui sert de démonstration au logiciel ou bien un Kakuro que l'on aura sauvegardé.

La window est simplement constituée d'une boîte "box" dans laquelle se trouve un bouton du nom du fichier à charger. Celui-ci dépend de ce que la fonction *createSelectionMenu* va renvoyer.

En effet, celle-ci va parcourir les sauvegardes et créer des boutons en fonctions de celles-ci. Si aucune sauvegarde n'est trouvée alors un bouton ramenant vers la grille de base est créé. Une fois cela fait, lorsque l'on clique sur le bouton, la méthode *loadDefaultKakuro* s'exécute et charge le Kakuro de base.

4.4 Creator_Window

Parlons à présent du fichier *Creator_Window.py*. Son nom l'indique : il sert à créer la fenêtre d'édition de grille ainsi que la sauvegarde de la grille créée. C'est une fenêtre importante car elle permet à l'utilisateur de créer ses propres grilles de Kakuro afin de s'aider sur les exemples concrets.

La fenêtre se présente de la façon suivante :

- Une grille de bouton servant à créer le Kakuro (des flèches permettent d'alterner entre case non jouable, jouable, ou case d'instructions).
- Le choix de la dimension de la grille
- Deux boutons : un vers la fenêtre d'aide à la résolutions de grilles et un autre permettant de sauvegarder la grille créée.

Voici les fonctions présentes dans le fichier *CreatorWindow.py*
Il y en a 5 :

- Un constructeur qui permet de définir les paramètres de base de la fenêtre.
- Une fonction *init_UI* qui permet de créer l'intérieur de la fenêtre : styles, création de *VBox*, *layouts* ainsi que l'affichage de la fenêtre
- *create_meneu_layout* qui permet de créer le menu du bas.
- *update_creator_kakuro_dimensions* qui gère le changement de taille de la grille
- *create_kakuro_creator_layout* gère quant à elle la création de la grille d'édition (grâce au widget *Morphing_Label*, détaillé plus loin).

4.5 Widgets utilisés

Dans cette dernière partie dédiée au code source, nous allons aborder les widgets utilisés par les windows du front-end.

Les widgets permettent de créer des portions de code facilement implémentable dans les windows ce qui rend le code moins lourd et permet de mieux le diviser.

Il y a quatre widgets que nous avons utilisé et qui vont être expliqués plus loin :

- *Content_Button_Widget.py*
- *Divided_Label_Widget.py*
- *Loader_Button_Widget.py*
- *Morphing_Label.py*

Voici leur utilité.

4.5.1 Content_Button_Widget

Ce widget est utilisé pour les boutons au sein des cases servant à sélectionner des valeurs à jouer. Il permet aussi de créer les contraintes. C'est le widget qui gère le contenu des cases au sein des grilles et est donc un widget particulièrement important.

La méthode *change_Content* permet de créer les boutons au sein des cases ainsi que de cacher un bouton si il est sélectionné.

Dans le fonctionnement, si un bouton est sélectionné alors la valeur ne peut pas être présente dans la ligne, on va donc utiliser les méthodes de propagation qui vont permettre de retirer la valeur dans toute la ligne afin qu'elle ne soit jouable qu'une fois.

4.5.2 Divided_Label_Widget

Ce widget sert à créer les neuf boutons au sein d'une case, ils sont tous créés au préalable et seront affichés ou non en fonction des lignes de jeu par le code qui utilisera ces boutons en question.

Ce widget est court et relativement simple et ne contient qu'un constructeur, il est créé pour être appelé dans le code des fenêtres.

4.5.3 Loader_Button_Widget

Loader_Button_Widget est utilisé dans la fenêtre *Loader_Window.py* et sert à charger le nom d'un fichier au format *.pkl* et à l'afficher dans un bouton sur la page de chargement de grilles.

Tout comme le widget *Divided_Label_Widget*, celui ci est aussi très court et n'est utilisé que dans *Loader_Window.py*.

4.5.4 Morphing_Label

Le dernier widget est *Morphing_Label.py*, il permet de créer des cases dans la creator window.

Il permet d'alterner entre case jouable, non jouable et d'instruction dans la grille de la fenêtre de création.

Le constructeur permet de spécifier les paramètres des cases : taille, contenu et type. La méthode *updateCaseType* permet de changer le type de la case et d'alterner entre les trois types.

4.6 Dossiers de sauvegardes

Pour ce qui est des dossiers de sauvegarde, il y en a deux: le dossier *Save* qui contient les sauvegardes à utiliser et le dossier *Save_Creator_Result* qui contient les nouvelles sauvegardes créées.

Pour utiliser une sauvegarde créée il faut donc la déplacer du dossier *Save_Creator_Result* vers *Save*.

5 Le logiciel dans son ensemble

Le logiciel est donc constitué du back-end, toute la partie "calcul et travail" et du front-end, la partie affichage.

Voici un exemple d'utilisation du logiciel : si un utilisateur veut résoudre une grille de Kakuro il peut lancer le logiciel et entrer dans l'éditeur de grilles, après avoir spécifié la taille de sa grille il peut commencer à l'éditer. Une fois cela fait il peut enregistrer sa grille et mettre la sauvegarde au format *.pkl* dans le dossier *Save*.

Une fois cela fait il peut désormais utiliser le logiciel pour charger son Kakuro et l'utiliser pour s'aider à résoudre sa grille grâce à la heat map et aux indications de coups à jouer.

6 Améliorations possibles

Le logiciel n'est pas parfait et certaines fonctionnalités n'ont pas été implémentées par manque de temps.

Voici une liste de fonctionnalités qui auraient trouvés leur place au sein de ce logiciel :

- Simplification du système de sauvegarde
- Choix du nom du fichier sauvegardé
- Une résolution automatique d'une grille de Kakuro afin de se corriger
- Amélioration du système de recommandation.
- Amélioration de l'interface

7 Annexes