Université de Rennes

CYBERSCHOOL
ÉCOLE UNIVERSITAIRE
DE RECHERCHE EN CYBERSÉCURITÉ

Low Level Programming

Rapport projet de takuzu

Master 1 Cybersécurité / ISTIC

2023/2024

Rédigé par

Bassirou BADIANE

Encadrante

Isabelle Puaut

Table des matières

Introduction **3**

1 Heuristiques **3**

2 Algorithme pour la génération de grid **3**

3 Mode d'utilisation de l'application **3**
3.1 Solver Mode . . . 3
3.2 Generation Mode . . . 3

4 Organisation du code **4**
4.1 Gestion des options . . . 4
4.2 Fonctions utilitaires . . . 4
4.3 Gestion des grilles . . . 4

5 Description Synthétique du programme **4**
5.1 Démo du fonctionnement du programme . . . 4
5.1.1 Génération d'une grille ayant au moins une solution . . . 4
5.1.2 Résolution de la grille avec MODE_FIRST . . . 5
5.1.3 Résolution de cette grille avec MODE_ALL . . . 6

6 Stratégie de tests et Performance du programme. **6**

7 Limites du programme **8**

Conclusion **8**

Introduction

Le takuzu est un jeu de logique et de réflexion qui se joue sur une grille carrée. La grille est composée de cases vides, qu'on doit remplir en respectant ces règles principales :
Chaque ligne et chaque colonne doivent contenir le même nombre de 0 et 1. Il ne peut y avoir plus de deux symboles identiques côte à côte, que ce soit horizontalement ou verticalement. De plus, toutes les lignes doivent être différentes, pareilles pour les colonnes.

Au début du jeu, certaines cases sont déjà remplies avec des symboles, et on doit déduire les autres. Le but est de remplir toute la grille en respectant les règles du jeu.

1 Heuristiques

Le rôle des heuristiques est d'augmenter les chances de résoudre la grille Takuzu. Au delà des deux heuristiques données, jai rajoutée une autre dont l'implémentation fonctionne comme suit : elle parcourt chaque ligne et chaque colonne de la grille et cherche les cellules vides qui ont un nombre de 0 et de 1 différent. Lorsqu'une telle cellule est trouvée, ce qui signifie qu'il ne peut y avoir qu'une seule valeur possible pour cette cellule, l'heuristique remplit la cellule vide avec la valeur opposée (0 si le nombre de 0 est inférieur au nombre de 1, et 1 si le nombre de 1 est inférieur au nombre de 0). Cela permet de réduire l'espace de recherche en éliminant les possibilités contraires à cette contrainte.

2 Algorithme pour la génération de grid

La fonction `generate_grid` est responsable de la génération de grilles. Elle a deux modes :
Mode normal : on génère une grille qui a au moins une solution.
Mode unique : la grille générée a une seule solution.
La fonction `generate_grid` prend en paramètre un pointeur vers une structure `t_grid` qui représente la grille à générer, ainsi que la taille de la grille souhaitée. Elle commence d'abord par initialiser la grille à vide puis elle génère aléatoirement une grille consistente ayant le meme pourcentage de "0" et de "1" . Cela se fait en parcourant chaque cellule de la grille et en attribuant soit "0" soit "1" avec une probabilité égale. Enfin, elle fait appel à la fonction statique has_solution qu'on a défini pour vérifier, si la grille a au moins une solution. Si la grille n'a pas de solution, l'algorithme génère une nouvelle configuration initiale jusqu'à ce qu'une grille valide soit trouvée.
Pour la génération d'une solution unique, j'ai défini la fonction generateUniqueSolution qui a pour rôle de générer une solution unique pour une grille donnée en utilisant des heuristiques de résolution. Elle applique différentes stratégies pour remplir les cellules vides de la grille de manière à garantir qu'il existe une seule solution possible.

3 Mode d'utilisation de l'application

Le programme contient deux modes principaux : le mode de génération ("generation mode") et le mode de résolution ("solver mode").

3.1 Solver Mode

Ce mode permet de résoudre les grilles Takuzu fournies en tant qu'arguments à l'application. L'utilisateur peut spécifier une grille à résoudre en fournissant le nom de fichier contenant la grille et le programme recherche toutes les solutions possibles de la grille.

3.2 Generation Mode

Ce mode permet de générer des grilles Takuzu de différentes tailles. L'utilisateur peut spécifier la taille de la grille à générer en utilisant l'option "-g" suivie d'un nombre, par exemple "-g4" pour générer une grille de taille 4x4. Les tailles valides pour les grilles sont 4, 8, 16, 32 et 64.

4 Organisation du code

Le code est organisé en utilisant trois fichiers : grid.c, takuzu.c et utility.c, chaque fichier ayant son propre rôle et ses fonctions spécifiques.

4.1 Gestion des options

Le fichier takuzu.c est le point d'entrée principal du programme. Il contient la fonction "main" qui gère les options de ligne de commande et appelle les fonctions appropriées en fonction des options sélectionnées. Les options sont traitées à l'aide de la fonction "getopt_long". Par exemple, l'option "-h" affiche l'aide en appelant la fonction "PrintHelp". Le fichier takuzu.c utilise également des fonctions telles que "file_parser" pour analyser les fichiers de grille et "grid_solver" pour résoudre les grilles.

4.2 Fonctions utilitaires

Le fichier utility.c contient des fonctions utilitaires utilisées dans le programme. Il comprend des fonctions telles que "PrintHelp" pour afficher l'aide du programme, "grid_allocate" pour allouer de la mémoire pour une grille, "grid_free" pour libérer la mémoire d'une grille, "check_char" pour vérifier si un caractère est valide, et "grid_print" pour afficher une grille. Le fichier utility.c inclut également la fonction statique "get_size" qui détermine la taille d'une grille en analysant le fichier correspondant. La fonction "file_parser" est utilisée pour analyser le contenu des fichiers de grille et initialiser les grilles en conséquence.

4.3 Gestion des grilles

Le fichier grid.c contient des fonctions liées à la manipulation de grilles. La fonction principale dans ce fichier est "grid_solver" qui permet de résoudre une grille selon le mode choisi par l'utilisateur. Elle fait appel a deux fonctions statiques :
La fonction `has_solution` qui a pour but de vérifier si la grille a au moins une solution et d'afficher la solution s'il existe. Elle est définie récursivement et utilise le backtracking pour explorer le choix opposé lorsque le pemier choix renvoyé par le fonction grid_choice ne mène pas à une solution.
La fonction `search_solutions` qui permet de rechercher toutes les solutions possibles de la grille donnée. Cette fonction, définie récursivement, explore toutes les combinaisons possibles. Par conséquent on enregistre toujours le choix opposé de la fonction grid_choice et on explore ces deux choix afin de vérifier et d'afficher toutes les solutions.

5 Description Synthétique du programme

Les deux principaux modes du programme marchent correctement pour toutes les grilles de taille inférieure ou égale à 16x16. Au-delà de cette taille, le temps de calcul, pour générer ou résoudre une grille est souvent très long..
Les options et fonctions déinies sont correctement implémentées d'après tous mes tests.

5.1 Démo du fonctionnement du programme

Pour tester le fonctionnement du programme, commençons d'abord par générer une grille de taille 8x8 et essayons de résoudre cette grille avec le mode solver.

5.1.1 Génération d'une grille ayant au moins une solution

Pour générer ce type de grille, on fait appel à l'option -g.

```
(base) bassirou@badiane-yoga-slim:~/Desktop/takuzu$ time ./takuzu -g
Generating a grid of size 8 x 8 ...

Grid of size 8 x 8 generated !!!
_ 0 _ _ 0 _ 1 _
1 _ _ 0 _ _ _ _
0 _ 1 _ _ 0 _ 1
0 1 _ _ _ _ _ _
1 _ _ _ 0 _ 0 1
_ 1 0 _ _ 0 _ 1
1 _ _ _ 1 0 _ 0
_ _ _ _ _ _ _ 1

real    0m1,203s
user    0m1,199s
sys     0m0,004s
```

FIGURE 1 – Génération d'une grille 8x8 avec au moins une solution

Remarquons qu'on a pas besoin de spécifier la taille dans la commande car par défaut elle est égale à 8x8.

5.1.2 Résolution de la grille avec MODE_FIRST

On a enregistré dans le fichier oneSolutionDemo8x8.txt la grille générée précédemment afin de vérifier qu'il a au moins une solution. Pour faire appel au solver mode,le programme est exécuté sans l'option -g.

```
########################################################
Solution n° 13
1 0 1 1 0 0 1 0
1 1 0 0 1 1 0 0
0 0 1 1 0 0 1 1
0 1 1 0 1 1 0 0
1 0 0 1 0 1 0 1
0 1 0 1 0 0 1 1
1 0 1 0 1 0 1 0
0 1 0 0 1 1 0 1


########################################################
Number of solutions found 13
########################################################

real    0m0,004s
user    0m0,004s
sys     0m0,000s
(base) bassirou@badiane-yoga-slim:~/Desktop/takuzu$
```

FIGURE 2 – Solutions de la grille générée précédemment

On voit bien ici que cette grille a au moins une solution.

5.1.3 Résolution de cette grille avec MODE_ALL

Pour trouver toutes les solutions, on active l'option -a en solver mode

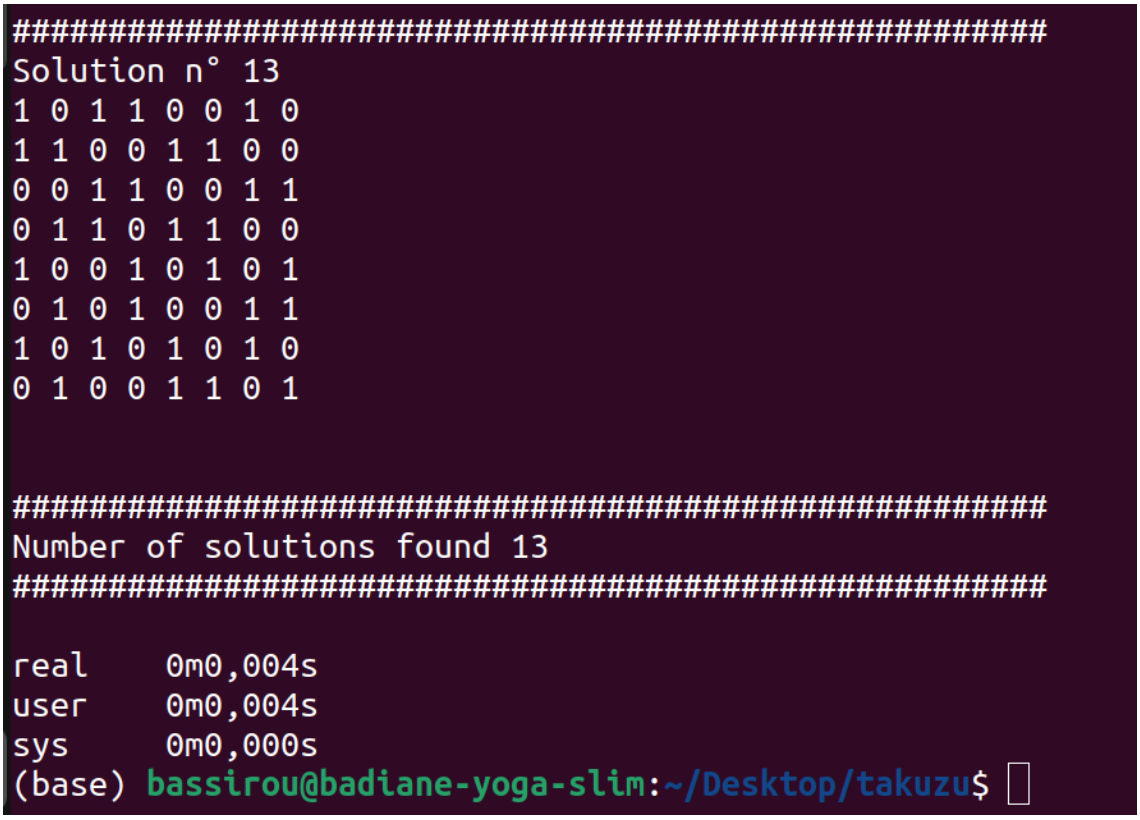

FIGURE 3 – Solutions de la grille générée précédemment

Avec le mode ALL, on remarque que cette grille admet au total 13 solutions.

6 Stratégie de tests et Performance du programme.

Afin de s'assurer que notre programme marche bien, on l'a rigoureusement testé avec pas mal de grilles. Le programme a ainisi réussi à trouver toutes les 72 solutions d'une grille 4x4 vide en 12ms

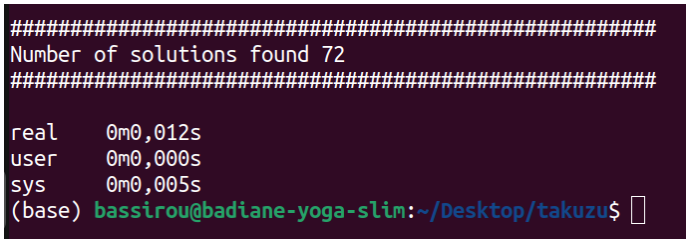

FIGURE 4 – Toutes les solutions d'une grille vide de taille 4x4

Et les 4 millions d'une grille vide de taille 8x8 en 7mn29s.

```
######################################################
Number of solutions found 4111116
######################################################

real    7m29,984s
user    5m39,058s
sys     0m37,731s
(base) bassirou@badiane-yoga-slim:~/Desktop/takuzu$
```

FIGURE 5 – Toutes les solutions d'une grille vide de taille 8x8

Le programme réussi aussi a générer et résoudre, sans le mode unique, des grilles de taille 16x16 en quelques millisecondes (dès fois ça peut prendre un peu plus de temps).

```
(base) bassirou@badiane-yoga-slim:~/Desktop/takuzu$ time ./takuzu -g16
grid size = 16 x 16
Generating a grid of size 16 x 16 ...

Grid of size 16 x 16 generated !!!
_ _ _ _ _ _ _ _ 1 _ _ _ _ _ _ _
0 _ _ _ _ 1 _ _ _ _ 0 1 _ 1 1 _
_ _ 0 0 _ _ _ 0 0 _ _ _ _ _ _ _
_ 1 _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ 1 _ _ _ 0 _ _ _ 1 _ _ 1 _
_ _ 0 _ _ _ _ 0 _ _ _ _ _ _ 1 _
0 _ _ _ _ _ _ _ 0 _ _ _ _ _ _ _
1 0 _ 0 _ _ _ _ 1 _ 0 _ 1 _ _ _
1 _ _ 0 _ _ _ _ _ _ _ _ _ 1 _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ 1 _ _ _ _ _ _ _ _ _ 0 _ _ _ _
_ _ _ 1 _ _ _ 0 1 0 _ 0 _ _ _ _
_ 1 _ _ _ 1 _ _ 0 1 _ _ _ _ _ _
_ 0 _ _ _ 0 _ _ _ 0 _ _ _ _ _ _
_ _ _ _ _ 1 _ _ _ _ _ 1 _ _ _ 0
_ _ 0 _ _ _ _ _ 1 _ _ _ _ _ _ 1

real    0m0,363s
user    0m0,363s
sys     0m0,000s
```

FIGURE 6 – Grid de taille 16x16 généré en 363ms

Une solution associé est aussi trouvée en peu de temps 335ms.

```
(base) bassirou@badiane-yoga-slim:~/Desktop/takuzu$ time ./takuzu tests/grid16x16.txt
######################################################
Solution found
0 0 1 0 0 1 0 0 1 1 0 1 1 0 1 1
0 0 1 1 0 1 0 1 1 0 0 1 0 1 1 0
1 1 0 0 1 0 1 0 0 1 1 0 0 1 0 1
0 1 0 0 1 0 1 1 0 1 0 1 1 0 0 1
0 0 1 1 0 1 0 0 1 0 1 1 0 1 1 0
1 1 0 0 1 0 1 0 0 1 0 0 1 0 1 1
0 0 1 1 0 0 1 1 0 0 1 1 0 1 0 1
1 0 1 0 1 1 0 0 1 1 0 0 1 0 1 0
1 1 0 0 1 0 1 0 1 0 1 0 0 1 1 0
0 0 1 1 0 1 0 1 0 0 1 1 0 1 0 1
1 1 0 0 1 0 1 1 0 1 0 0 1 0 1 0
1 0 1 1 0 1 1 0 1 0 1 0 0 1 0 0
0 1 0 1 0 1 0 1 0 1 0 1 1 0 0 1
1 0 1 0 1 0 0 1 1 0 1 0 1 0 1 0
1 1 0 1 0 1 1 0 0 1 0 1 0 1 0 0
0 1 0 1 1 0 0 1 1 0 1 0 1 0 0 1



real    0m0,335s
user    0m0,334s
sys     0m0,000s
```

FIGURE 7 – Une solution associée à la grille 16x16 générée

Le programme est aussi capable de générer des grilles qui ont une unique solution en quelques millisecondes.

7 Limites du programme

A partir de la taille 16 la génération de grille avec mode unique commence a être très lente. Néanmoins la résolution est plus ou moins rapide pour une grille de taille 16x16 comme le montre la figure 7. La génération et la résolution de grilles de taille supérieure ou égale à 32x32 est très lente même sans le mode unique.

Conclusion

En conclusion, ce projet de Takuzu met en œuvre des heuristiques, du backtracking et une génération de grilles avec au moins une (ou une et une seule solution).Ces heuristiques permettent de prendre des décisions informées lors du remplissage de la grille, en se basant sur les contraintes déjà présentes. Cela permet d'éliminer certaines possibilités et de guider le processus de résolution. Le backtracking a été utilisé pour explorer de manière exhaustive les différentes combinaisons possibles et trouver la solution optimale.