Carnet De Bord Takuzu Solver :

Répartitions tâches :

Nous avons fait la modélisation du code ensemble, nous avons mis nos idées sur papier, puis ensuite nous les avons codé ensemble, nous avons décidé que Loan s'occupera du Carnet De Bord, du diaporama et des fonctions suivantes : *grille_to_fichier, est_complete, generer_grille_unique_takuzu*
Tandis que Timothé s'occupera des fonctions : *generer_grille_takuzu, remplir_grille, est_valide, resoudre_takuzu, solveur.*
Toutes ces fonctions sont stockées dans takuzu_algo.py.
Le fichier Takuzu principal a été codé par nous deux au moment de la mise en commun et fichier takuzu_regle_verif.py a été réalisé l'année dernière. Idem pour le fichier takuzu_1NSI_eleve.py, mais celui-ci a été modifié légèrement pour ajouter les boutons re-générer et solution à l'interface graphique.

Modélisation :

Nous avons cherché un moyen de générer une grille aléatoire à partir d'une grille vide, la première idée que nous avons trouvé, c'est de générer une grille de façon complètement aléatoire et de vérifier si cette grille correspond aux règles du jeu Takuzu. (Explication des limites de cette modélisation dans la partie problème rencontré)

La deuxième idée à été d'utiliser une méthode pseudo-aléatoire qui place un nombre aléatoire mais vérifie dans un second temps si le chiffre passé est en accord avec les règles du Takuzu. Pour vérifier si le nombre aléatoire placé correspond aux règles nous testons les règles suivantes : autant de 1 que de 0 et pas plus de 2 chiffres identiques côte à côte sur chaque ligne et chaque colonne. (nous nous sommes aperçu de l'oublie de la règle 3 à la fin du projet)

Cette méthode est constituée en 2 parties : Une partie que place des nombres puis une autre qui vérifie que ces nombres soit bien placé

1. On place les nombres dans la liste par ligne, en premier le premier emplacement de la ligne 1, puis le deuxième emplacement de la ligne 1 et ainsi de suite, jusqu'au bout de la ligne, pour ensuite passer à la ligne suivante (ligne 2). Pour choisir la valeur à mettre dans une case du Takuzu on choisit aléatoirement une valeur entière entre 0 et 1, on place cette valeur et si celle-ci ne convient pas, on place l'autre choix (si 0 est choisi et qu'elle ne convient pas, on met 1). Pour cela on utilise une liste ([0,1]) qu'on mélange à chaque placement à l'aide de la méthode sample de la librairie random.

2. Pour vérifier dans un second temps si la grille est valide nous utilisons un algorithme récursif qui au moment du dépilement annule le placement de ces valeurs si elles ne correspondent pas aux règles du Takuzu.

Une fois notre fonction remplissage de grille finie, nous avons réfléchi à la méthode de solve :
Un algorithme qui permet de résoudre n'importe quelle grille de Takuzu en respectant les règles. Pour cela nous avons tout d'abord pensé à faire un programme qui résoudrait la Grille avec la même logique d'un humain.

C'est-à-dire suivre ces 3 étapes:

1. La première étape identifier les lignes ou colonnes lorsque 2 nombres identiques sont côte à côte et dans ce cas placer les nombres « opposés » (si 1 placé 0 et inversement). Exemple: … 1 1 … => 0 1 1 0
2. La seconde étape identifier les lignes et colonnes ou il reste uniquement 1 chiffre à placé et donc placé le chiffre correspondant en suivant la règle : "Il y a autant de 0 et de 1 sur une ligne ou colonne"
3. La troisième étape identifier ce schéma Ex : 0 … 0 ou 1 … 1
   Comme il n'est pas possible d'avoir plus de 3 même chiffre côte à côte, on place le nombre « opposé » au milieu. Ex : 0 1 0 ou 1 0 1

En suivant ces 3 étapes, en lien avec la logique qu'un humain utiliserait. Il serait théoriquement possible de compléter n'importe quelle grille. Enfin c'est ce que l'on pensait car après plus de réflexion, dans le cas ou le programme n'identifie aucun de ces 3 cas. La grille ne pourra pas être résolue, or, il est possible que cette grille ait une solution. Dans le cas où le joueur doit prendre une initiative et tenter de rajouter un nombre sans suivre la logique des 3 étapes. De plus, selon le cahier des charges demandé, en optant pour cette méthode en 3 étape,s notre fonction de résolution de la grille ne serait pas en capacité de trouver plusieurs solutions pour une même grille.

Nous avons donc abandonné cette idée de la "logique humaine" et avons cherché une autre méthode.
Nous nous sommes aperçu que notre fonction qui permet de Générer une grille, était proche de la fonction solve, et qu'avec quelques modifications elle pourrait aussi permettre de résoudre des grilles.

Pour que le projet soit clair, nous avons séparé notre code en 4 fichiers :
le fichier Takuzu Principale, qui importe les 3 autres fichiers et qui permet de lancer une partie en console, une partie avec interface graphique.

Une fois notre code finalisé et fonctionnel, nous avons testé de manière approfondie afin de connaître ces limites. Nous sommes donc arrivés aux résultats qu'il faut que la taille de la grille soit un nombre pair compris entre 3 et 13. Car au-dessus d'une grille de 12x12, notre

programme ainsi et la machine n'est pas capable de fournir une grille dans un temps raisonnable.

Concernant la complexité de notre fonction de solve, celle-ci est liée au nombre de case vide mais aussi à la probabilité de 9/10 (probabilité expérimentale avec un grille de 8x8) pour que la grille soit générée sans erreurs au niveau des colonnes, pour chaque case vide il y a à chaque fois 2 possibilités 0 ou 1. Pour n cases vides dans la grille on peut donc estimer que la complexité est de 2^n. La complexité du solveur de Takuzu est exponentielle par rapport au nombre de cases vides dans la grille. A ajouter à cela le fait que la grille peut être générée une deuxième fois ou troisième… si il y a des colonnes identiques. La complexité de notre méthode est de k*(2^n) avec k le nombre de grilles générées avant résultat final.

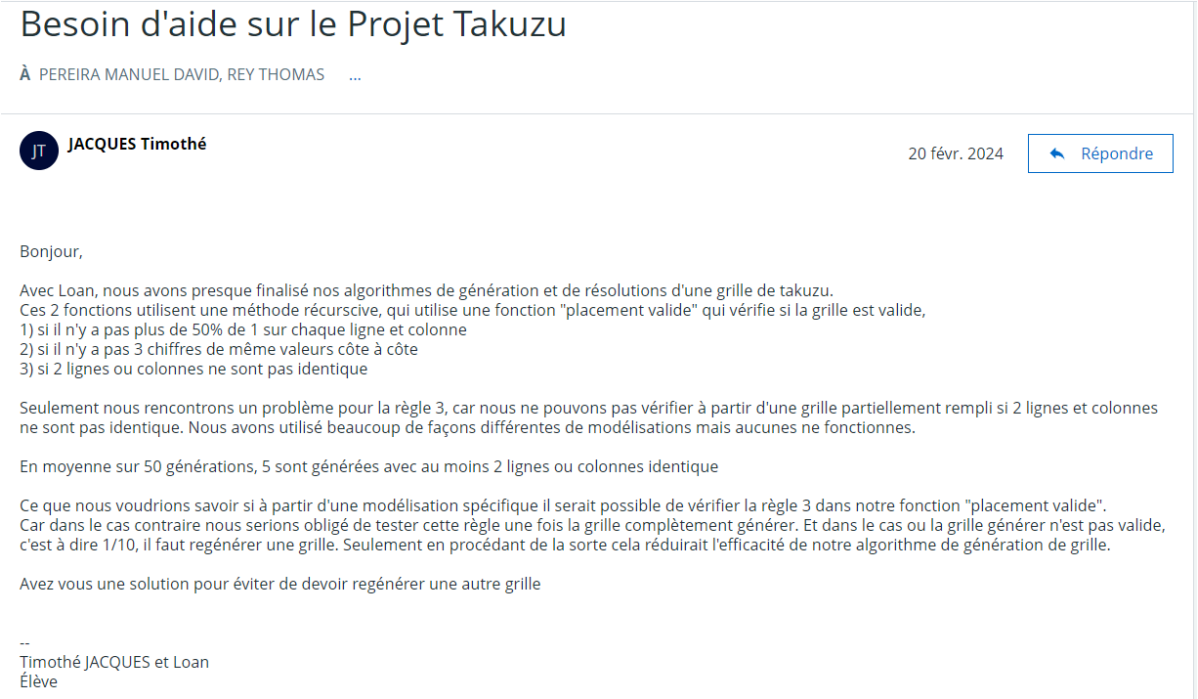
Problèmes rencontrés :

Le premier problème que nous avons rencontré, avec notre toute première idée de modélisation, la génération d'une grille de jeu complète pouvait être très longue, à partir du moment où la grille faisait plus de 6x6. Avec cette méthode, la probabilité de générer une grille qui respectait les règles du jeu, avec une méthode aléatoire, était très faible.

Dans ma fonction *def solve_recursive(grid, solutions):* j'ai rencontré un problème car pour enregistrer la solution il fallait faire une copie de ma liste, pour cela j'ai voulu recréer une liste par compréhension mais celle ci ne fonctionnait pas car j'avais écris: [ligne for ligne in grille]
J'ai donc essayer de faire une copie avec le for i in range
[[grille[ligne][colonne] for colonne in range(len(grille[ligne]))] for ligne in range(len(grille))]
Pour finalement arriver à simplifier le code sous cette forme, c'est donc là que j'ai vu mon erreur
[[colonne for colonne in ligne] for ligne in grille]

Un problème auquel j'ai fait face est celui de la fonction résoudre qui s'appelle de manière récursive en boucle. Faisant crash mon pc, car la mémoire ram atteint 100%. Finalement, je me suis rendu compte que la condition d'arrêt qui était fixée était incomplète car il manquait un return lorsque les événements récursifs devaient être dépilés.

A la fin nous avons essayé de tester nos grilles générées avec les fonctions codées l'année passée. Et à notre grande surprise les solutions que nous avions générées étaient fausses. C'est donc après plusieurs heures de recherche que nous nous sommes aperçu que nous avions oublié la règle 3 du takuzu « 2 lignes ou 2 colonnes ne peuvent être identiques. »
Cette règle était présente dans le code de l'année passé mais pas dans le code de la fonction est_valide(*grille*, *ligne*, *colonne*):
Nous avons cherché à l'ajouter, mais nous avons réussi à mettre en place la vérification pour uniquement pour les lignes identiques , nous n'y sommes pas arrivé à mettre en place la vérification des colonnes identiques. Et c'est à ce moment que vous avez reçu notre message:



Besoin d'aide sur le Projet Takuzu

À PEREIRA MANUEL DAVID, REY THOMAS ...

JT **JACQUES Timothé** 20 févr. 2024 Répondre

Bonjour,

Avec Loan, nous avons presque finalisé nos algorithmes de génération et de résolutions d'une grille de takuzu.
Ces 2 fonctions utilisent une méthode récurscive, qui utilise une fonction "placement valide" qui vérifie si la grille est valide,
1) si il n'y a pas plus de 50% de 1 sur chaque ligne et colonne
2) si il n'y a pas 3 chiffres de même valeurs côte à côte
3) si 2 lignes ou colonnes ne sont pas identique

Seulement nous rencontrons un problème pour la règle 3, car nous ne pouvons pas vérifier à partir d'une grille partiellement rempli si 2 lignes et colonnes ne sont pas identique. Nous avons utilisé beaucoup de façons différentes de modélisations mais aucunes ne fonctionnes.

En moyenne sur 50 générations, 5 sont générées avec au moins 2 lignes ou colonnes identique

Ce que nous voudrions savoir si à partir d'une modélisation spécifique il serait possible de vérifier la règle 3 dans notre fonction "placement valide".
Car dans le cas contraire nous serions obligé de tester cette règle une fois la grille complètement générer. Et dans le cas ou la grille générer n'est pas valide, c'est à dire 1/10, il faut regénérer une grille. Seulement en procédant de la sorte cela réduirait l'efficacité de notre algorithme de génération de grille.

Avez vous une solution pour éviter de devoir regénérer une autre grille

--
Timothé JACQUES et Loan
Élève

Comme la grille se remplit progressivement au début, toutes les colonnes sont identiques on ne peut donc pas vérifier que les colonnes sont différentes.

Finalement nous avons opté pour ne pas instaurer la “vérification des colonnes” lors de la création de notre grille. De manière aléatoire, il y a erreur 1 fois sur 10. On génère donc une grille et dans la cas où la grille générée est fausse on en re-génère une.
Nous avons créé des fonctions de test pour voir si sur plus de 1000 grilles celle-ci était juste et si le temps de génération n’était pas trop long.
A partir de maintenant toutes nos grilles générées étaient justes, seulement lors de la méthode de résolution certaines grilles étaient fausses lorsqu’il y avait plus d’une solution. Nous nous sommes aperçu que cela provenait du fait que nous avions mis “le code pour vérification des colonnes” uniquement dans la fonction de génération des grilles et nous avions oublié la fonction de solve.

Mode d’emploi pour le programme :

Pour tester notre programmes, il suffit d'aller dans le fichier takuzu_principale et d'exécuter le fichier, vous verrez dans un premier temps :

Les variables: taille et pourcentage qui peuvent être modifiés selon ce que souhaite l’utilisateur
Pour générer une grille il y a 2 possibilités: Soit avec le fonction *generer_grille_unique_takuzu* soit avec la fonction *generer_grille_unique_takuzu_unique*

Pour finir il y a aussi la possibilité d'utiliser une interface graphique avec la fonction: *takuzu_graphique*