Projet SDD1

TAKUZU

(Proposé par M. Nicolas FLASQUE)

Consignes & Informations générales :

⇒ Ce projet est à réaliser exclusivement en langage C

⇒ Organisation des équipes :
- Ce projet est à réaliser en binôme (**un seul trinôme est autorisé** si un **nombre impair** d'élèves)
- La liste des équipes est à remettre aux enseignants **au plus tard** à la fin de la première séance de suivi de projet

⇒ Dates clé :
- Date de publication : **28/03/2022**
- Date de présentation du projet : **04/04/2022**
- Date de suivi 1 : Semaine du **04/04/2022**
- Date de suivi 2 : Semaine du **09/05/2022**
- Date de soumission : **15/05/2022** à **23h59**
- Date de soutenance : Semaine du **16/05/2022**

⇒ Rendu final : Une archive **.zip** contenant
- Le code du projet contenant les fichiers **.c** et **.h**
- Le rapport en **.pdf**
- Un fichier **README.txt** donnant la liste des programmes et comment les utiliser en pratique. Ce fichier doit contenir toutes les instructions nécessaires à l'exécution ; il est très important pour expliquer à l'utilisateur comment se servir de l'outil.

⇒ Dépôt du projet :
- Zone de dépôt dédiée sur Moodle.

⇒ Évaluation
- Barème indicatif
- Barème détaillé : sera fourni plus tard
- Note finale du projet = Note code + Note rapport + Note soutenance
- Rappel : note projet = 15% de la note du cours « Algorithmique et Structures de données 1 »
- Les membres d'une même équipe peuvent avoir des notes différentes en fonction des efforts fournis dans la réalisation de ce projet.

⇒ Plagiat
- Tout travail présentant du plagiat sera sévèrement sanctionné

Organisation du code :

⇒ La notation du code tiendra compte principalement de :
- L'implémentation des fonctionnalités demandées : avancer au mieux mais PAS en hors sujet
- La qualité du code fourni : organisation en modules et en fonctions, **commentaires**, noms de variables significatifs, respect des noms de fichiers.
- Facilité d'utilisation de l'interface utilisateur

Notions pédagogiques traitées :

⇒ Pour vous aider à la réalisation de ce projet, vous pouvez vous appuyer sur :
- Les supports du cours TI202 (I, B, R)
- Livres, cours divers sur le web. **ALERTE PLAGIAT !!** il ne s'agit pas de copier des programmes entiers.
- Enseignants **Efrei** lors des séances de suivi du projet

Préambule

Le **Takuzu** est un jeu de grilles dans l'esprit du Sudoku, mais les chiffres utilisés pour remplir une grille sont uniquement le 0 et le 1

Une partie de **Takuzu** se joue sur une grille carrée, de dimension paire. Dans le cadre du projet, vous allez proposer deux niveaux :

1. L'un utilisant une matrice de taille 4 x 4 où les lignes sont numérotées de 1 à 4 et les colonnes de A à D
2. L'autre utilisant une matrice de taille 8 x 8 où les lignes sont numérotées de 1 à 8 et les colonnes de A à H

Les règles du **Takuzu** sont extrêmement simples :

1. Dans une ligne, il doit y avoir autant de 0 que de 1
2. Dans une colonne, il doit y avoir autant de 0 que de 1
3. Il ne peut pas y avoir deux lignes identiques dans une grille
4. Il ne peut pas y avoir deux colonnes identiques dans une grille
5. Dans une ligne ou une colonne, il ne peut y avoir plus de deux 0 ou deux 1 à la suite (on ne peut pas avoir trois 0 de suite ou trois 1 de suite)

À partir de ces règles, votre menu doit proposer 3 fonctionnalités principales :

1. Laisser un joueur résoudre une grille
2. Résoudre automatiquement une grille de **Takuzu**
3. Générer une grille de **Takuzu**

Pour vous entraîner et mieux comprendre le jeu, consulter : https://www.20minutes.fr/services/takuzu

Pour améliorer l'interface utilisateur, il est possible d'utiliser la librairie conio.h

Partie I : Laisser un joueur résoudre une grille (10 pts)

Dans cette étape, la grille à résoudre est totalement connue par la machine. Cependant, seule une partie des cases est affichée, et c'est au joueur de proposer les valeurs pour remplir la grille afin d'aboutir à la solution connue. Le programme se contente d'aider l'utilisateur de deux manières :

1. En indiquant la validité des coups joués, c'est-à-dire faire que les règles soient respectées ;
2. En donnant, pour une grille, des indices sur des coups invalides.

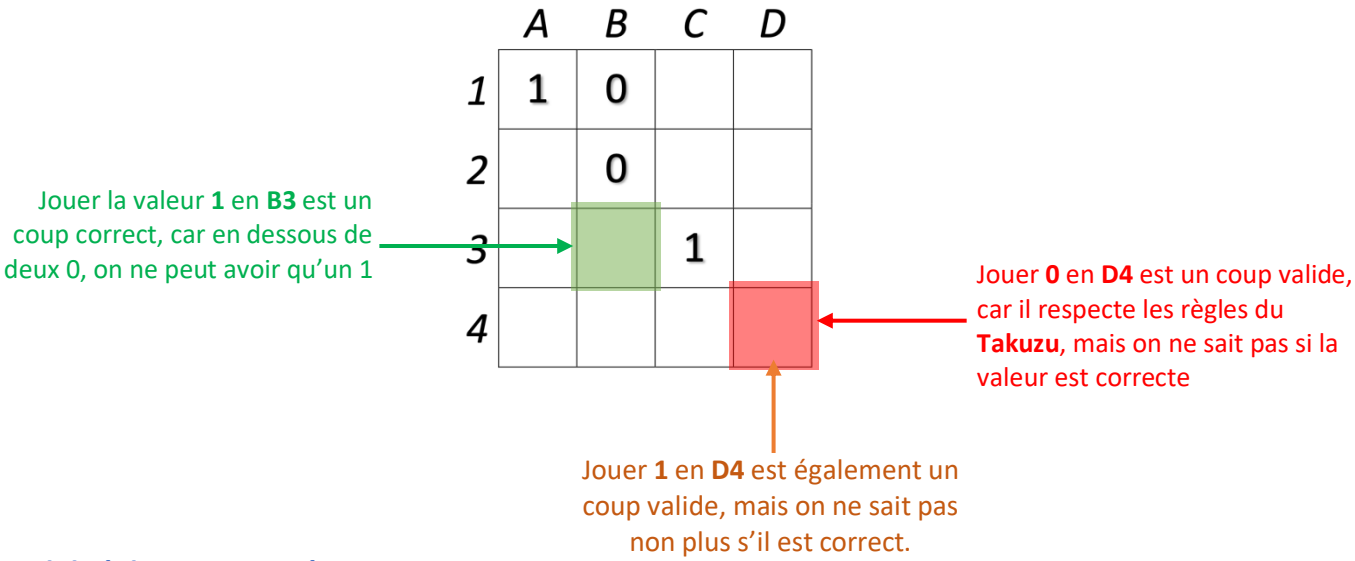
Coup correct Vs Coup valide

Un coup **valide** est un coup qui respecte les règles du **Takuzu**, mais qui n'est pas forcément la bonne réponse.

Un coup **correct** est un coup qui donne la bonne réponse (0 ou 1) pour une case donnée de la grille.

Exemple :

Soit la grille 4 x 4 suivante :



Validité des coups joués

À chaque proposition d'un coup par un joueur, le programme doit indiquer si le coup est valide ou non. Il y a plusieurs manières de faire. Dans ce projet, nous allons procéder comme suit :

1. Pour saisir un coup, le joueur doit saisir le numéro de la ligne et celui de la colonne
2. Le joueur doit ensuite insérer la valeur souhaitée (0 ou 1)
3. La machine lui affiche un texte pour indiquer si le coup est valide ou pas.
4. ~~Si le coup est correct, le programme doit indiquer que c'est correct et expliquer pourquoi il est correct~~
5. Lorsque le coup est invalide, le joueur perd une vie. Le programme doit lui indiquer la règle qui n'a pas été respectée et lui affiche un indice (quand cela est possible) pour l'aider à corriger son coup.
6. Si le coup est valide mais pas correct, afficher comme indice « Coup valide mais incorrect ! »
7. ~~L'utilisateur a droit à **3** essai pour corriger son coup s'il n'est pas valide~~
8. ~~Au bout de ces **3** coups, la machine lui propose deux choix :~~
   - ~~Tenter de jouer une autre case~~
   - ~~Obtenir un indice~~
9. Si l'utilisateur a consommé 3 vies, le programme s'arrête en indiquant un échec dans la résolution de la grille.

Votre programme devra tester, par des fonctions, la validité d'un coup en fonction des cases déjà découvertes par le joueur.

Il est recommandé d'utiliser des tableaux dynamiques dès cette partie.

Implémentation de la grille

En réalité, la grille est stockée dans un premier temps sous sa forme complète. Les valeurs de la grille complète (la solution à laquelle doit arriver le joueur) sont stockées dans un tableau 2D, ayant les valeurs 0 et 1. Nous appelons ce tableau la **solution**.

À ce tableau, nous allons associer une grille de la même dimension appelée **masque**. Cette dernière contiendra elle aussi les valeurs 0 et 1 mais ayant un sens différent. En effet :

1. La valeur 0 présente dans une case **masque[i][j]** signifie que sa case associée **solution[i][j]** doit être **invisible** pour le joueur au début de la partie.
2. La valeur 1 présente dans une case **masque[i][j]** signifie que sa case associée **solution[i][j]** doit être **visible** pour le joueur au début de la partie.

Enfin, la grille que voit le joueur est partiellement remplie au début de la partie avec des 0 et des 1 de la grille complète (au bon endroit évidemment); les cases non découvertes sont par exemple représentées par la valeur -1 ou toute valeur différente de 0 et de 1 ; on appellera ce tableau **grille_jeu**.

Illustration :

Soit la solution suivante (qui est valide bien sûr) ainsi qu'un masque

1	0	0	1
1	0	1	0
0	1	1	0
0	1	0	1

Grille **solution**

1	0	0	0
0	0	1	0
1	0	1	1
0	1	0	0

Grille **masque**

Le **masque** n'est pas une grille de **Takuzu** ! Les 0 du masque, notés en gris, indiquent les cases non visibles par le joueur en début de partie.

1			
		1	
0		1	0
	1		

Grille **grille_jeu**

Quelques indices à proposer au joueur

~~Le joueur peut demander jusqu'à 3 indices sur une partie s'il ne réussit pas un coup au bout de 3 essais.~~ Votre programme, à partir de la **grille_jeu**, doit repérer les coups corrects dictés par les règles du **Takuzu** :

1. Au-dessus, en dessous, à gauche, à droite d'une série de deux 0, il ne peut y avoir qu'un 1
2. Au-dessus, en dessous, à gauche, à droite d'une série de deux 1, il ne peut y avoir qu'un 0
3. Entre deux 0, il ne peut y avoir qu'un 1
4. Entre deux 1, il ne peut y avoir qu'un 0
5. En comparant une ligne (ou une colonne) déjà remplie avec une ligne (ou une colonne) à laquelle il manque 2 valeurs, si toutes les valeurs correspondent, alors on peut remplir la ligne (ou la colonne) à laquelle il manque deux valeurs.

Exemple :

Si dans la grille **grille_jeu** , nous avons les deux lignes suivantes :

0	1	1	0	1	0	1	0

0		1	0	1	0	1	

Alors pour compléter la deuxième ligne il faut un 1 et un 0. Puisqu'il ne peut y avoir 2 lignes identiques dans la grille, la deuxième ligne se complète forcément par :

0	0	1	0	1	0	1	1

Menu

Lorsqu'un utilisateur choisi de résoudre lui-même une grille, le programme doit lui demander d'abord de choisir la taille de la grille (4x4 ou 8x8) puis il lui affiche les 3 sous menus suivants :

1. Saisir manuellement un masque ➔ après la saisie le masque ainsi que la grille de jeu résultante doivent s'afficher pour vérifier que le masque saisi a bien été appliqué
2. Générer automatiquement un masque ➔ Le masque aléatoire généré ainsi que la grille de jeu résultante doivent être affichés pour vérifier le bon fonctionnement du programme.
3. Jouer (si cette étape est choisie, un masque est automatiquement généré aléatoirement)

Partie II : Résoudre automatiquement une grille (6 pts)

C'est la partie la plus simple du projet !

Dans cette partie, il est demandé :

1. D'écrire une fonction qui va résoudre un grille **grille_jeu** en appliquant les indices progressivement en fonction de la possibilité de les calculer. La fonction doit jouer le rôle du joueur humain de la partie I. Elle doit proposer pour une case une valeur aléatoire (0 ou 1) avant d'engager le processus qui vérifie la
2. D'afficher les étapes de résolution à l'écran : afficher l'état de la grille après l'application de chacun des indices jusqu'à la résolution finale en mettant des pauses dans le programme, voir la fonction `Sleep()` ou `sleep()` ; ou en demandant à l'utilisateur d'appuyer sur une touche entre chaque coup joué.

Partie III : Générer une grille (4 pts)

Dans cette partie, il est demandé de générer une grille de **Takuzu valide**. Afin d'y parvenir, il faut passer par les deux étapes suivantes :

1. Génération de lignes (ou de colonnes) valides en fonction de la taille de la grille choisie (4x4 ou 8x8)
2. Construction d'une grille complète valide à partir des lignes et colonnes valides.

Lignes / Colonnes valides

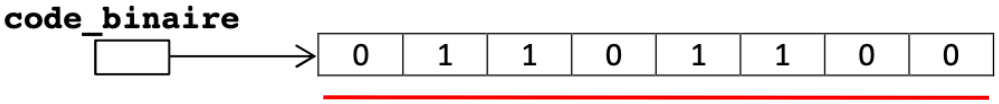
Pour construire un modèle de ligne (ou de colonne) valide, nous allons nous baser sur le codage binaire. En effet, étant donné qu'une ligne (colonne) n'est constituée que de 0 et de 1, l'on peut donc considérer la combinaison totale comme étant un seul nombre.

Ainsi, si la taille de la grille est de :

- 4 x 4 : alors une ligne (colonne) de **Takuzu** représente un nombre binaire se trouvant ente 0 et 15
- 8 x 8 : alors une ligne (colonne) de **Takuzu** représente un nombre binaire se trouvant ente 0 et 255.

Chaque nombre binaire généré, sera stocké dans un tableau 1D appelé **code_binaire**. Puis, pour chaque combinaison générée, vérifier si les règles de **Takuzu** sont respectées :

1. Compter le nombre de 0 et de 1 dans le tableau **code_binaire**
2. Regarder s'il n'y a pas de suite de plus de deux 0 ou deux 1 dans le tableau
3. …



Représentation binaire du nombre 108 : Exemple d'une ligne d'une grille de taille 8 x 8

NB : Ici, il est possible d'envisager une fonction `int verif_ligne()` qui prend en paramètres le tableau code_binaire d'une ligne donnée et qui retourne 1 si elle est valide, 0 sinon.

Construction de la grille complète

Sur la base des lignes (colonnes) valides, construire la grille complète progressivement.

Menu

Lorsqu'un utilisateur choisi de générer une grille de **Takuzu** automatiquement, le programme doit d'abord lui demander de choisir la taille de la grille à générer. Après le choix de la dimension, le programme doit afficher 2 sous menus :

1. Afficher l'ensemble des lignes (colonnes) valides ➔ Penser à la bonne structure de données pour les stocker au fur et à mesure.
2. Générer une grille de **Takuzu** ➔ L'affichage doit comporter des pauses qui nous permettent de suivre les étapes détaillées de la génération de la grille.

Ressources

Voici 4 grilles de **Takuzu** 8x8 assez simples, que votre programme doit pouvoir résoudre facilement. Pour ces grilles, l'on vous donne la solution. Les cases en grisé correspondent aux cases qui ne sont pas visibles au début de la partie (**masque**).

Pour les grilles 4x4, vous pouvez en créer vous-même quelques-unes très facilement.

1	0	1	1	0	1	0	0
1	0	1	0	1	0	0	1
0	1	0	1	1	0	1	0
0	1	0	1	0	1	1	0
1	0	1	0	0	1	0	1
0	1	0	0	1	0	1	1
0	0	1	1	0	1	1	0
1	1	0	0	1	0	0	1

0	0	1	0	1	0	1	1
1	0	0	1	0	1	0	1
0	1	0	1	1	0	1	0
0	1	1	0	0	1	1	0
1	0	1	0	0	1	0	1
1	0	0	1	1	0	0	1
0	1	1	0	1	0	1	0
1	1	0	1	0	1	0	0

1	0	0	1	0	1	0	1
0	0	1	1	0	0	1	1
1	1	0	0	1	1	0	0
1	1	0	1	0	1	0	0
0	0	1	0	1	0	1	1
0	0	1	1	0	1	0	1
1	1	0	0	1	0	1	0
0	1	1	0	1	0	1	0

1	1	0	1	0	1	0	0
1	0	1	0	1	0	0	1
0	1	0	1	0	0	1	1
0	1	1	0	0	1	1	0
1	0	1	0	1	1	0	0
1	0	0	1	1	0	0	1
0	1	0	1	0	1	1	0
0	0	1	0	1	0	1	1

Remarques générales :

⇒ Les saisies sécurisées sont à effectuer systématiquement même si elles ne sont pas toujours demandées explicitement
⇒ Ne pas hésiter à afficher des messages aux utilisateurs lorsqu'une action n'est pas possible.
⇒ Afficher le menu à la fin de chaque action pour pouvoir basculer vers une autre action.