

RO203 - Jeux, graphes et RO

Zacharie Ales, Marc Etheve, Eric Soutil

2020 - 2021








PROJET









1 Objectif

L'objectif de ce projet est de résoudre des jeux de logique issus du site :

chiark.greenend.org.uk/~sgtatham/puzzles.

Vous choisirez deux des problèmes représentés dans le tableau suivant (la somme de leurs difficultés devra être au moins égale à 10).

	Nom	Difficulté	Objectif
	Unequal	1	Remplir en respectant les inégalités.
	Lightup	2	Placer des lampes pour éclairer l'ensemble des cases.
	Tents	3	Placer une tente pour chaque arbre.
	Unruly	4	Trouver un pavage de tuiles noires et blanches.
	Dominosa	4	Trouver un pavage en dominos.
	Undead	4	Remplir par des monstres.
	Signpost	4	Numéroter toutes les cases.
	Tracks	3	Relier par des rails.
	Rect	4	Trouver un pavage de rectangles.
	Range	5	Placer les cases noires.
	Flip	5	Rendre toutes les cases blanches.
	Towers	4	Remplir par des tours.

	Nom	Difficulté	Objectif
	Singles	6	Masquer les doublons des lignes et colonnes.
	Bridge	5	Associer le bon nombre de ponts aux sommets.
	Pegs	7	Retirer tous les jetons sauf un.
	Loopy	6	Trouver une boucle vérifiant les contraintes d'adjacence.
	Galaxies	8	Placer des barrières symétriquement.
	Pearl	8	Trouver une boucle passant par chaque perle.
	Palisade	9	Placer des palissades vérifiant des contraintes d'adjacence.
	Filling	9	Remplir par des chiffres.

Remarque 1 : Ces jeux sont également accessibles sur android (peut être également sur iOS).

Remarque 2 : Le programme *wine* peut permettre d'exécuter les fichiers .exe sous linux.

2 Travail demandé

Le travail demandé pour chacun de vos deux problèmes est décrit ci-dessous.

Remarque : Le code complet du jeu de sudoku est donné en guise d'exemple. N'hésitez pas à vous en inspirer pour chacun de vos deux jeux.

Partie 1 Implémentation

1. Modélisation du problème

Modéliser sur papier la résolution du jeu par un programme linéaire en nombres entiers.

Remarque : Le but de ces problèmes est de trouver une solution réalisable. Il n'y a donc généralement pas d'objectif à proprement parlé. Si plusieurs solutions sont possibles, il peut cependant être intéressant de fixer un objectif permettant de les distinguer (pour améliorer l'efficacité du *branch-and-bound*). Par exemple, s'il faut remplir une grille avec des chiffres, on peut chercher une solution qui minimise la valeur de la case en haut à gauche (ou n'importe quelle combinaison linéaire de valeur de plusieurs cases).

2. Définition manuelle d'une instance

Dans le répertoire `data` créer un fichier `instanceTest.txt` représentant une instance simple du jeu considéré. Par exemple, dans le cas du sudoku, on pourrait créer le fichier suivant représentant une grille de taille 4×4 :

Première ligne de la grille → 1, , , 4

↙ Case initialement vide

, 3, , 1

, , , 3

La première case de la dernière ligne contient un 3 → 3, , ,

3. Lecture d'une instance

- Compléter la méthode `readInputFile(path::String)`, figurant dans le fichier `io.jl`, permettant la lecture d'une instance du jeu considéré figurant au chemin `path` et retournant une ou plusieurs variables représentant cette instance.
- Tester votre méthode en lançant julia en mode console, puis en exécutant la commande :
`readInputFile("../data/instanceTest.txt").`

4. Implémentation du modèle

- Implémenter votre modèle dans la méthode `cplexSolve`, figurant dans le fichier `resolution.jl`. Cette méthode retournera :
 - un booléen indiquant si une solution optimale est obtenue ;
 - un réel indiquant la durée de la résolution ;
 - une ou plusieurs variables représentant la solution.
- Tester la résolution de l'instance figurant dans votre fichier `test` en lançant julia en mode console puis en exécutant les commandes :
`include("resolution.jl")`
`cplexSolve(readInputFile("../data/test.txt"))`

Remarque : Pour que cela fonctionne, les entrées de la méthode `cplexSolve` doivent correspondre aux sorties de la méthode `readInputFile`.

5. Affichage en ligne de commande

Créer deux méthodes `displayGrid()` et `displaySolution()` dans le fichier `io.jl` permettant respectivement d'afficher dans la console :

- une instance non résolue ;
- une instance résolue.

Exemple pour une grille de sudoku :

<pre>1, , , 4 , 3, , 1 , , , 3 3, , ,</pre>	<pre>----- 1 - - 4 - 3 - 1 ----- - - - 3 3 - - - -----</pre>	<pre>----- 1 2 3 4 4 3 2 1 ----- 2 1 4 3 3 4 1 2 -----</pre>
Instance au format texte	<code>displayGrid()</code>	<code>displaySolution()</code>

6. Génération d'instances

- Compléter la méthode `generateInstance()` figurant dans le fichier `generation.jl` afin qu'elle permette de générer une instance aléatoire du jeu considéré. Vous pourrez ajouter des paramètres à cette méthode afin de générer différents types d'instances. Par exemple dans le cas du sudoku, la taille de la grille et le pourcentage de cases remplies sont considérés.
- Compléter la méthode `generateDataSet()` du fichier `generation.jl` afin de générer un ensemble d'instances dans le répertoire `./data`.

Remarque : Il n'est pas nécessaire que les instances générées possèdent une unique solution.

7. Résolution de plusieurs instances

- a) Compléter la méthode `solveDataSet()` afin qu'elle permette d'effectuer la résolution de l'ensemble des instances.

Remarque : Les résultats de cplex pour chaque instance seront stockés dans un fichier texte figurant dans le répertoire `./res/cplex`. Chaque fichier devra contenir deux variables `solveTime` et `isOptimal`. Ces conditions sont nécessaires au bon fonctionnement des méthodes de visualisation des résultats que nous utiliserons par la suite.

- b) Compléter la méthode `heuristicSolve()` du fichier `resolution.jl` afin qu'elle réalise une résolution heuristique d'une instance.

8. Mise en forme des résultats

Utiliser les méthodes `resultsArray()` et `performanceDiagram()` du fichier `io.jl` afin d'obtenir deux représentations de vos résultats (*i.e.*, tableau de résultats et diagramme de performances).

Partie 2 Rapport (date limite : 2 mai)

Le rapport contiendra la description de vos programmes linéaires, vos heuristiques, vos générateurs d'instances. Vous commenterez les résultats de vos méthodes de résolution :

- temps de calcul en fonction de la taille ou du type d'instances (graphiques encouragés);
- une solution est-elle toujours trouvée?
- jusqu'à quelle taille la résolution est-elle possible?
- si le programme linéaire est trop long et que vous avez une fonction objectif qui n'est pas constante, quel est le gap après 10 minutes?

Définition - *Gap*

Ecart relatif entre la meilleure relaxation linéaire r et la meilleure solution entière connue z : $\frac{|r-z|}{z}$;

— ...

Vous fournirez également l'ensemble du code source de votre projet.

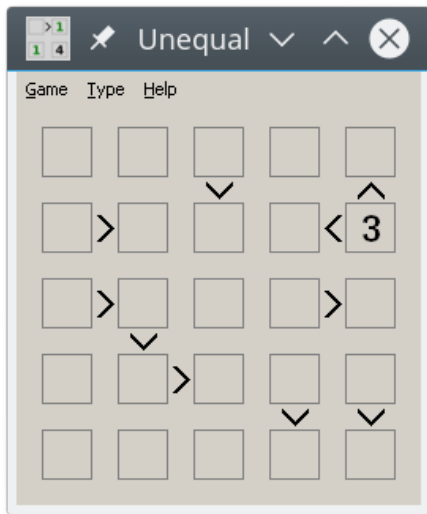
Partie 3 Soutenance (6 mai)

1. Présenter vos programmes de résolution;
2. Indiquer comment vous générez vos instances;
3. Indiquer si certaines contraintes n'ont pas pu être prises en compte (probable pour votre jeu le plus difficile). Dans ce cas, préciser :
 - a) si vous avez pu les prendre partiellement en compte;
 - b) s'il serait possible des les prendre totalement en compte en utilisant une formulation contenant un nombre exponentiel de variables ou de contraintes.
4. Mettre en avant les difficultés rencontrées.

3 Règles

Des règles en anglais plus détaillées peuvent être obtenues sur le site chiark.greenend.org.uk/~sgtatham/-puzzles

3.1 Unequal



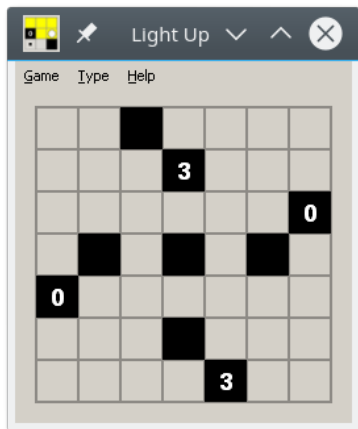
Exemple de grille initiale



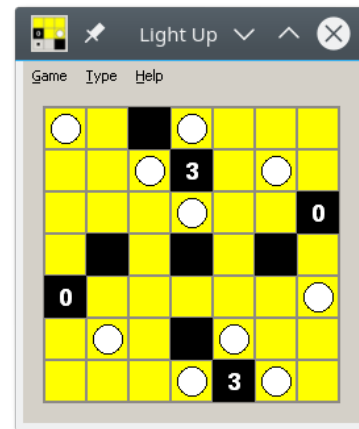
Exemple de grille résolue

- **Difficulté :** 2
- **Règles :** Remplir la grille de taille $n \times n$ par des chiffres de façon à ce que :
 - chaque ligne et colonne contienne les chiffres de 1 à n ;
 - les inégalités soient satisfaites.

3.2 Lightup



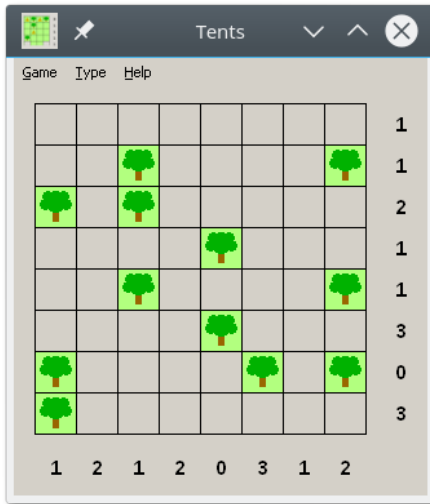
Exemple de grille initiale



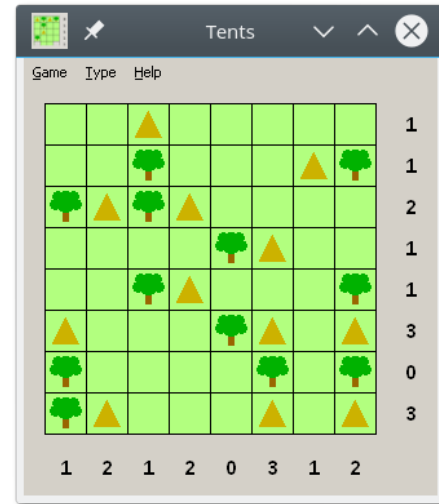
Exemple de grille résolue

- **Difficulté :** 2
- **Règles :** Placer des lampes afin d'éclairer l'ensemble des cases blanches. Une lampe éclaire toutes les cases blanches figurant sur la même ligne ou la même colonne qu'elle à moins qu'une case noire ne se trouve entre les deux. De plus :
 - Une lampe ne doit pas être éclairée par une autre lampe ;
 - Les cases noires comportant un chiffre x doivent être adjacentes à x lampes.

3.3 Tents



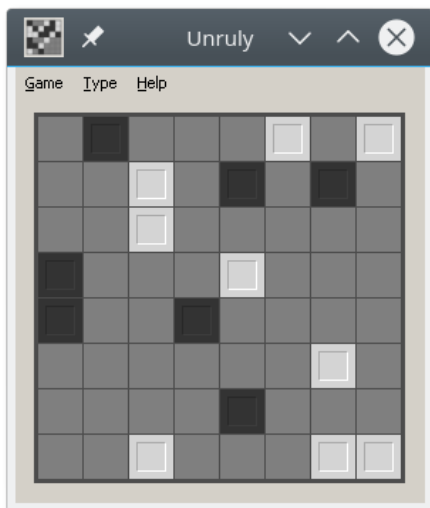
Exemple de grille initiale



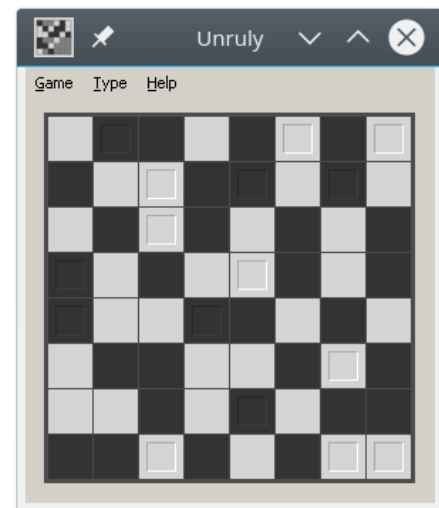
Exemple de grille résolue

- **Difficulté :** 3
- **Règles :** Placer des tentes de façon à ce que :
 - Chaque ligne et chaque colonne contienne le nombre de tentes indiquées ;
 - Chaque arbre ait une tente adjacente qui lui soit associée (il y a donc autant de tentes que d'arbres, mais un arbre peut, néanmoins, avoir plusieurs tentes qui lui sont adjacentes) ;
 - Deux tentes ne soient jamais adjacentes ou en diagonale l'une de l'autre.

3.4 Unruly



Exemple de grille initiale



Exemple de grille résolue

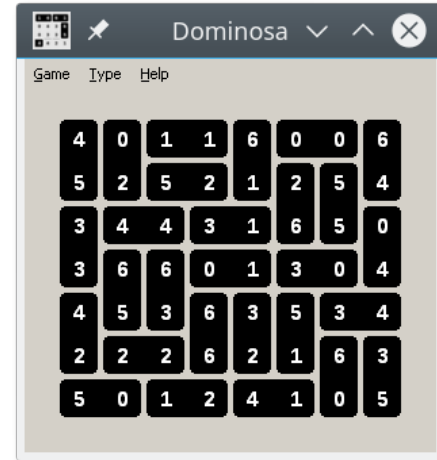
- **Difficulté :** 3
- **Règles :** Remplir une grille avec des cases noires et blanches de façon à ce que :
 - Chaque ligne et chaque colonne comporte autant de cases blanches que de noires ; et
 - Il n'y ait jamais plus de 2 cases successives d'une même couleur (horizontalement et verticalement).

3.5 Dominosa

The image shows the 'Dominosa' game interface. At the top left is a small 4x4 grid icon. To its right is a pencil icon. The title 'Dominosa' is centered at the top, followed by a downward arrow, an upward arrow, and a close button (X). Below the title bar is a menu with three items: 'Game', 'Type', and 'Help'. The main area contains an 8x8 grid of numbers. The numbers are arranged as follows:

4	0	1	1	6	0	0	6
5	2	5	2	1	2	5	4
3	4	4	3	1	6	5	0
3	6	6	0	1	3	0	4
4	5	3	6	3	5	3	4
2	2	2	6	2	1	6	3
5	0	1	2	4	1	0	5

Exemple de grille initiale

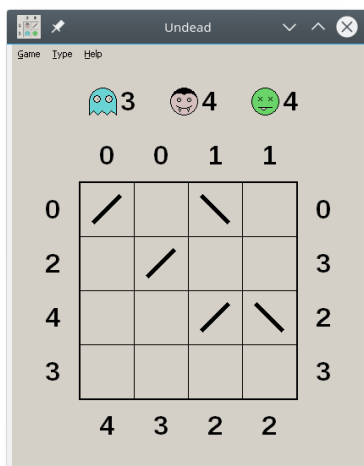
4	0	1	1	6	0	0	6
5	2	5	2	1	2	5	4
3	4	4	3	1	6	5	0
3	6	6	0	1	3	0	4
4	5	3	6	3	5	3	4
2	2	2	6	2	1	6	3
5	0	1	2	4	1	0	5

Exemple de grille résolue

— **Difficulté :** 3


— **Règles :** Regrouper par 2 les chiffres afin de former des dominos de façon à ce que tous les dominos soient différents.

3.6 Undead



		3	4	4		
		0	0	1	1	
0		/		\		0
2			/			3
4				/	\	2
3						3
		4	3	2	2	

Exemple de grille initiale

		3	4	4	
		0	0	1	1
0		/		\	
2			/		
4				/	\
3					
		4	3	2	2

Exemple de grille résolue

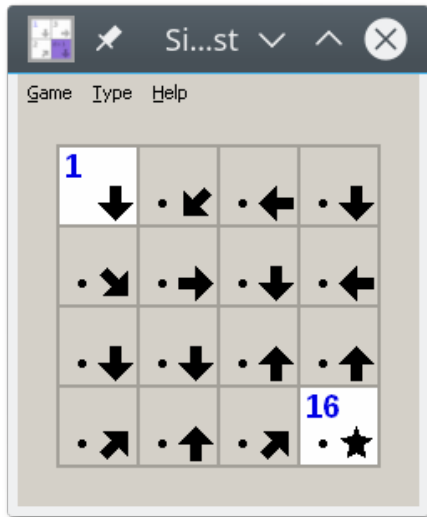
— **Difficulté :** 4

— **Règles :** Remplir les cases de la grille ne comportant pas de miroirs par des monstres (zombie, vampire ou fantômes). Les chiffres en bord de grille indiquent combien de monstres doivent être visibles par quelqu'un regardant la grille à partir de cette position, sachant que :

- les zombies sont toujours visibles;
- les fantômes ne sont visibles que s'ils sont reflétés;
- les vampire ne sont visibles que s'ils ne sont pas reflétés.

Le nombre de monstres de chaque type à placer est connu.

3.7 Signpost



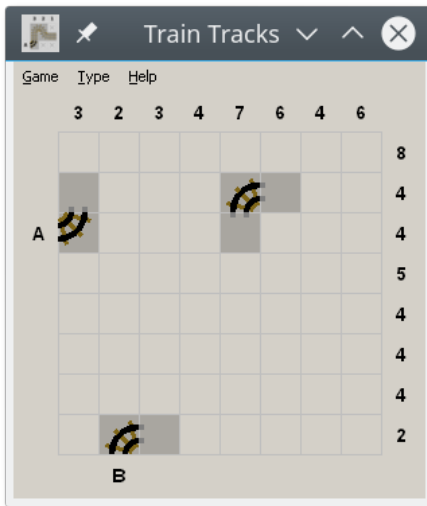
Exemple de grille initiale



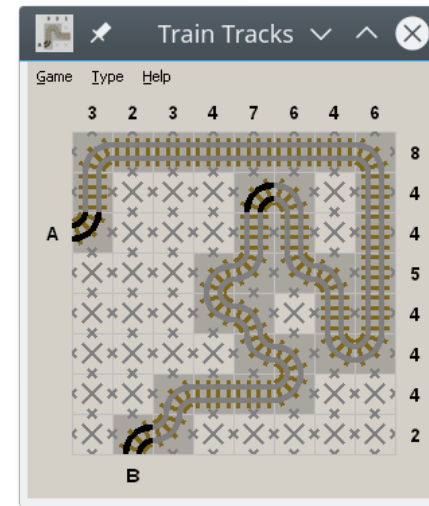
Exemple de grille résolue

- **Difficulté :** 4
- **Règles :** Assigner un numéro à chaque case de façon à ce que le successeur d'une case soit atteignable en suivant la flèche figurant sur la case.

3.8 Tracks



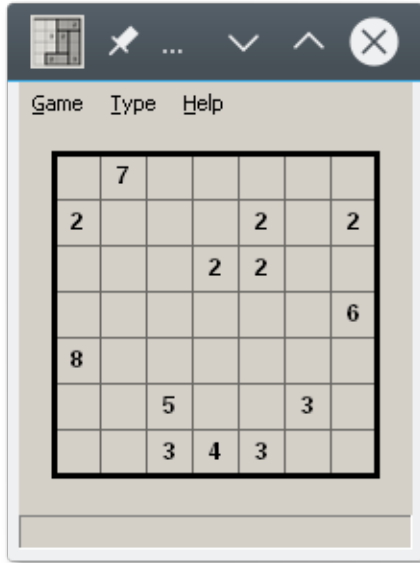
Exemple de grille initiale



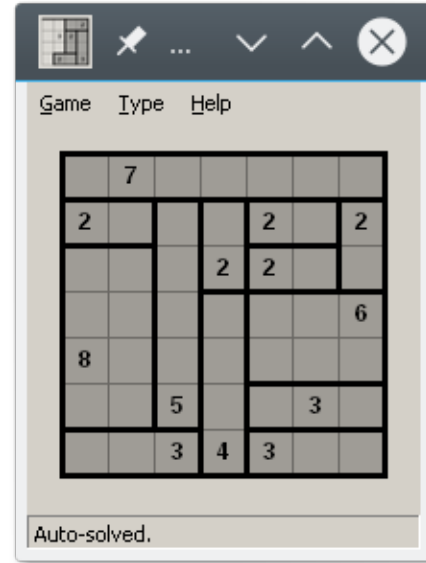
Exemple de grille résolue

- **Difficulté :** 4
- **Règles :** Construire un trajet allant de A en B de façon à ce que le nombre de cases utilisées par le trajet sur chaque ligne et chaque colonne corresponde aux valeurs en bord de grille.

3.9 Rect



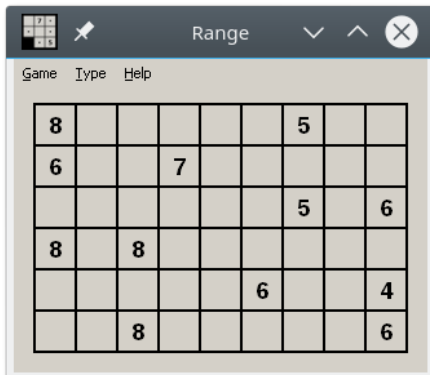
Exemple de grille initiale



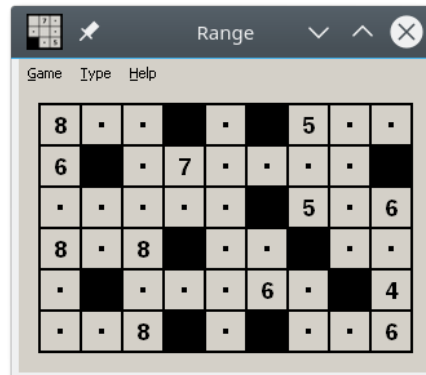
Exemple de grille résolue

- **Difficulté :** 4
- **Règles :** Remplir la grille avec des rectangle de façon à ce que :
 - Chaque chiffre x soit contenu dans un rectangle contenant exactement x cases ;
 - Chaque rectangle contienne exactement un chiffre.

3.10 Range



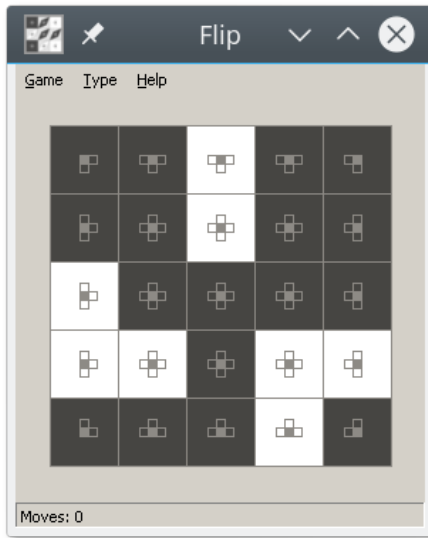
Exemple de grille initiale



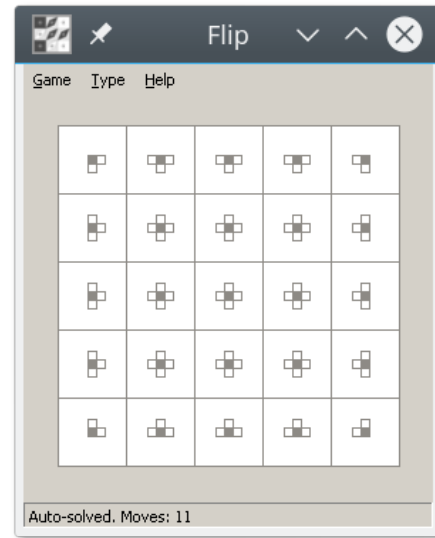
Exemple de grille résolue

- **Difficulté :** 5
- **Règles :** Placer des cases noires de façon à ce que chaque chiffre x puisse *voir* exactement x cases. Un chiffre voit une case si cette case est située sur la même ligne ou la même colonne que lui et si aucune case noire ne figure entre eux deux. De plus, l'ensemble des cases blanches doit former un ensemble connexe (*i.e.*, un ensemble de cases blanches ne peut être isolé des autres par des cases noires) et deux cases noires ne peuvent être adjacentes.

3.11 Flip



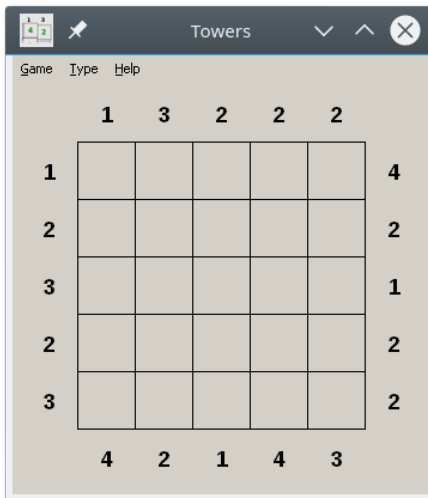
Exemple de grille initiale



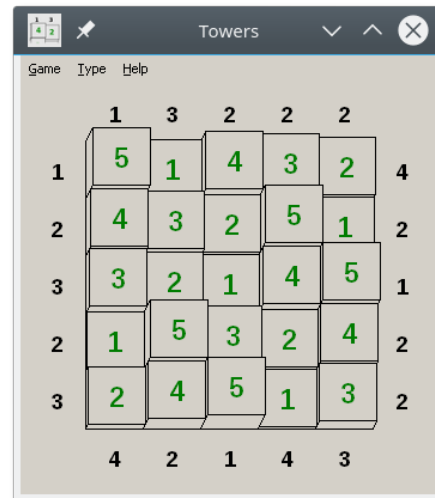
Exemple de grille résolue

- **Difficulté :** 5
- **Règles :** A chaque tour, une case est sélectionnée ce qui change la couleur de cette case et de ses 4 cases voisines. L'objectif est que toutes les cases soient blanches en un nombre de tours.

3.12 Towers



Exemple de grille initiale



Exemple de grille résolue

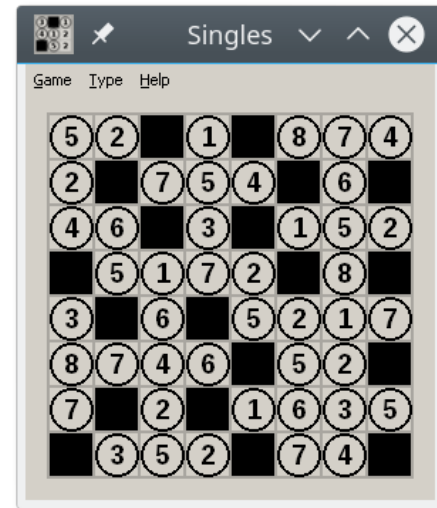
- **Difficulté :** 5
- **Règles :** Remplir la grille de taille $n \times n$ par des tours de taille 1 à n de façon à ce que :
 - Chaque ligne et chaque colonne contienne une tour de chaque hauteur ;
 - Les chiffres indiquent combien de tours doivent être visibles par quelqu'un regardant la grille à partir de cette position ;
 Par exemple, si la grille possède un chiffre 1, il n'y a qu'une tour de visible et c'est donc nécessairement une tour de taille n .

3.13 Singles



5	2	2	1	5	8	7	4
2	2	7	5	4	6	6	5
4	6	4	3	2	1	5	2
5	5	1	7	2	1	8	2
3	6	6	7	5	2	1	7
8	7	4	6	4	5	2	7
7	5	2	6	1	6	3	5
3	3	5	2	5	7	4	4

Exemple de grille initiale

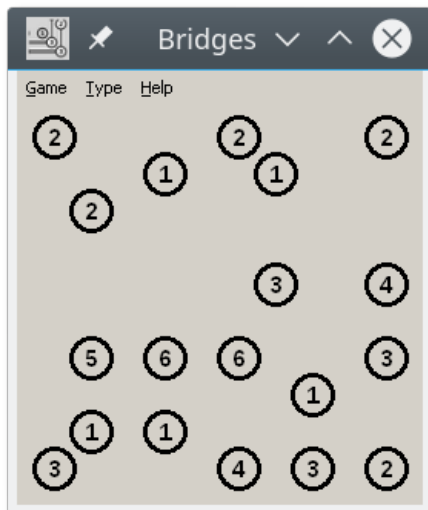



5	2		1		8	7	4
2		7	5	4		6	
4	6		3		1	5	2
	5	1	7	2		8	
3		6		5	2	1	7
8	7	4	6		5	2	
7		2		1	6	3	5
	3	5	2		7	4	

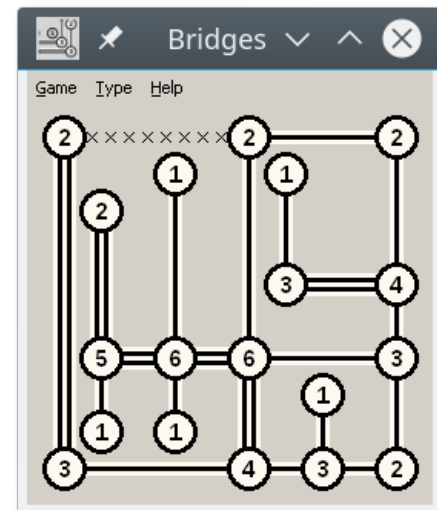
Exemple de grille résolue

- **Difficulté** : 6
- **Règles** : Masquer des case de façon à ce que :
 - aucun chiffre ne soit visible plus d'une fois sur chaque ligne et chaque colonne;
 - les cases masquées ne soient pas adjacentes (elles peuvent néanmoins être placées en diagonale);
 - l'ensemble des cases visibles est connexe.

3.14 Bridge



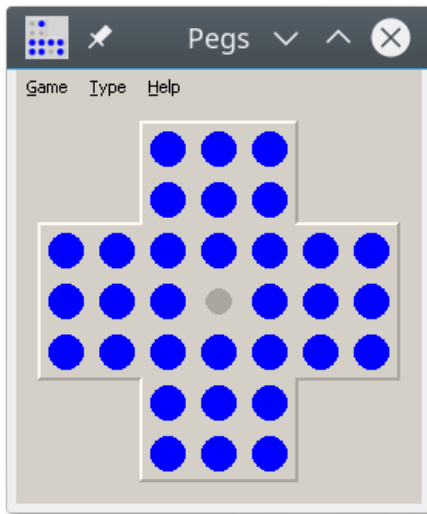
Exemple de grille initiale



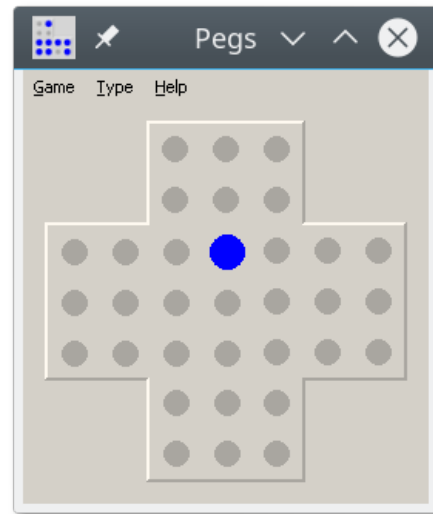
Exemple de grille résolue

- **Difficulté** : 6
- **Règles** : Ajouter des ponts reliant les sommets de façon à ce que :
 - le nombre de ponts partant d'un sommet soit égal au nombre indiqué sur le sommet;
 - aucuns ponts ne se croisent;
 - il existe un chemin entre toutes paires de sommets.

3.15 Pegs



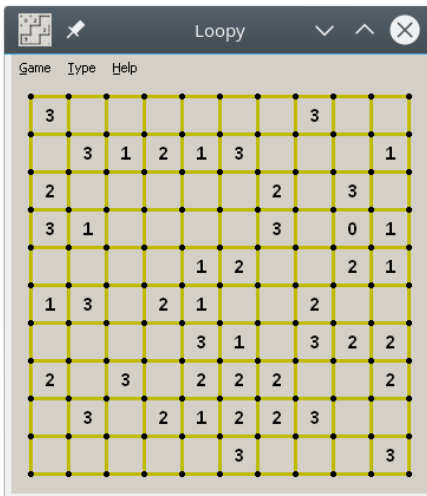
Exemple de grille initiale



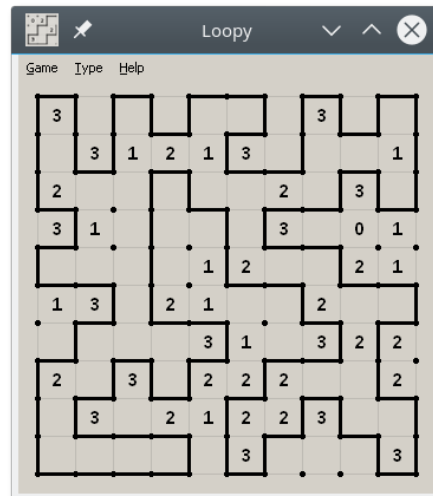
Exemple de grille résolue

- **Difficulté :** 6
- **Règles :** Retirer tous les jetons sauf un. A chaque tour, un jeton A est retiré en faisant sauter un jeton B par dessus lui. La case d'atterrissage du jeton B doit être libre.

3.16 Loopy



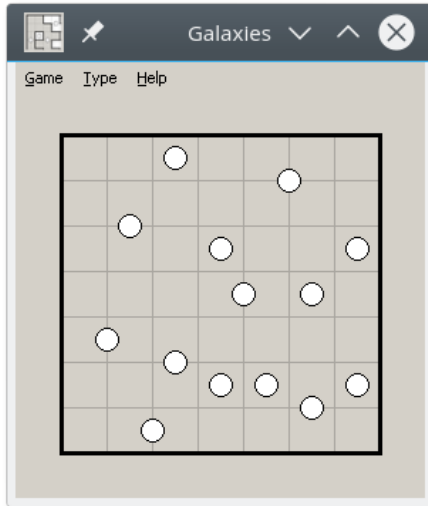
Exemple de grille initiale



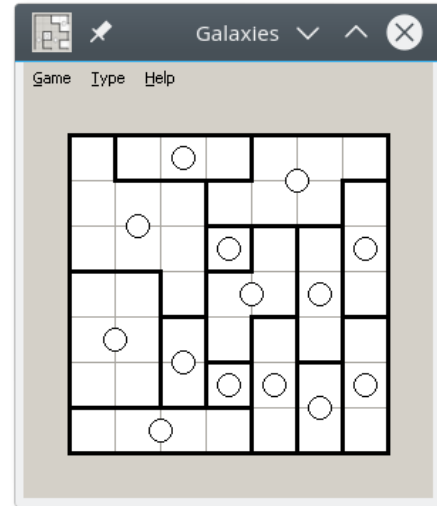
Exemple de grille résolue

- **Difficulté :** 8
- **Règles :** Tracer une boucle. La boucle doit passer par x cotés d'une case comportant le chiffre x .

3.17 Galaxies



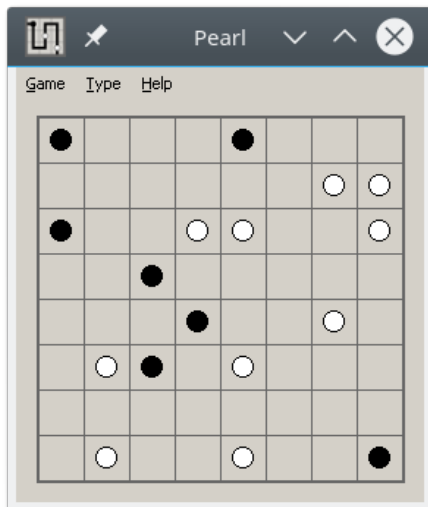
Exemple de grille initiale



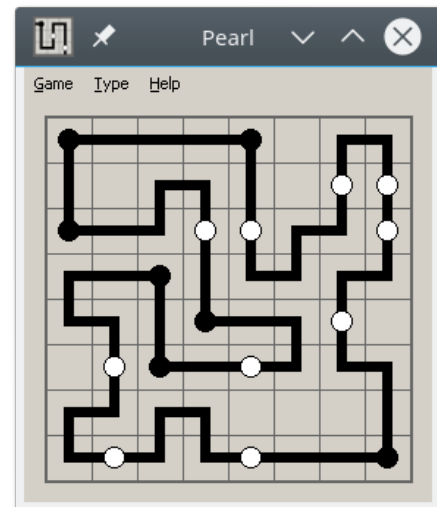
Exemple de grille résolue

- **Difficulté :** 8
- **Règles :** Tracer des séparations afin de diviser la grille en régions. Chaque région doit contenir un unique rond blanc (appelé *galaxie*) de telle sorte que la région soit symétrique par rapport à ce rond.

3.18 Pearl



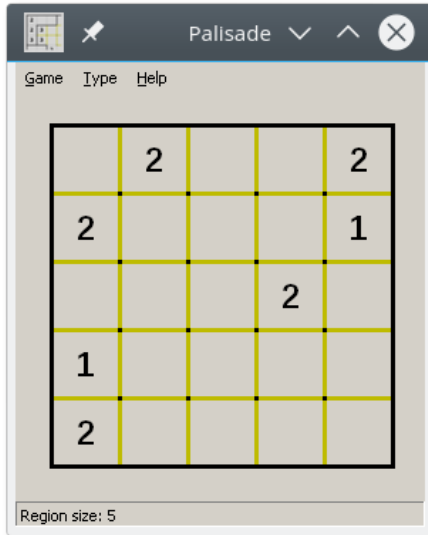
Exemple de grille initiale



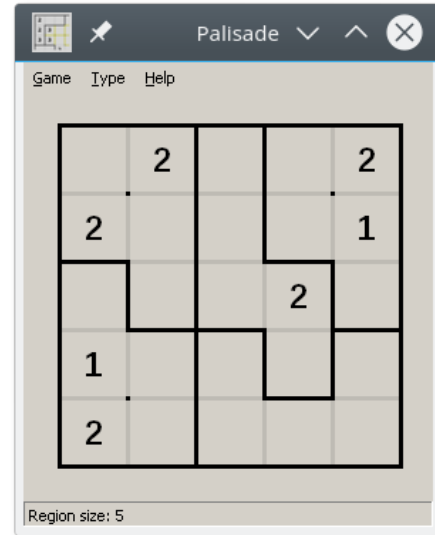
Exemple de grille résolue

- **Difficulté :** 8
- **Règles :** Tracer une boucle de façon à ce que :
 - En chaque case comprenant un rond noir, la boucle forme un angle droit qui ne soit pas directement connecté à un autre angle droit;
 - En chaque case comprenant un rond blanc, la boucle forme une ligne droite directement connectée à au moins un angle droit.

3.19 Palisade



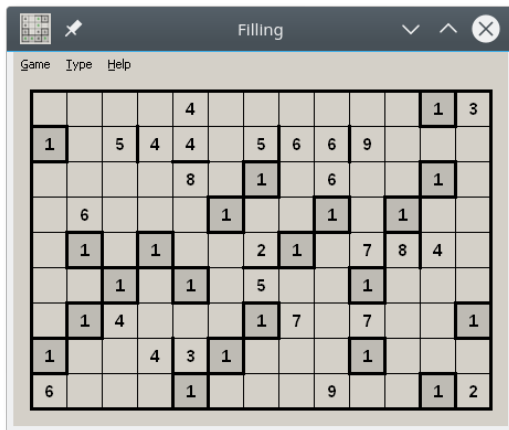
Exemple de grille initiale



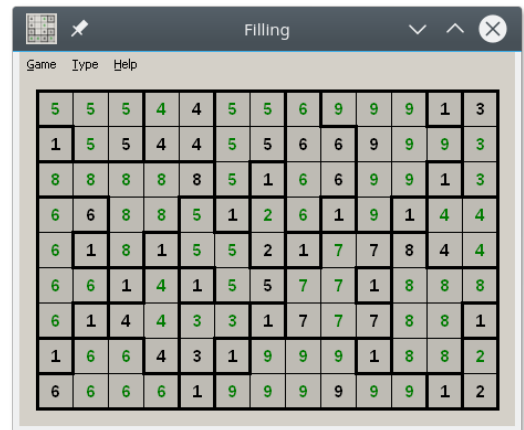
Exemple de grille résolue

- **Difficulté** : 8
- **Règles** : Placer des palissades afin de former des régions de façon à ce que :
 - Toutes les régions aient la même taille n (la valeur n fait partie des données du problème);
 - Chaque case comportant un chiffre x ait exactement x de ses cotés comportant une palissade.

3.20 Filling



Exemple de grille initiale



Exemple de grille résolue

- **Difficulté** : 9
- **Règles** : Remplir la grille par des chiffres de façon à ce que chaque chiffre x figure dans une zone connexe comprenant x case de valeur x .