

INFO-F-302 Informatique Fondamentale

Projet : Problèmes de Satisfactions de Contraintes et Utilisation de l'Outil ChocoSolver

El Haman Abdeslam Abdeslam Minhas Prabhdeep

Année académique 2016-2017

1 Problèmes d'échecs

1.1 CSP

Les CSP pour les questions 1 et 2 se trouvent dans **quest1-2.pdf**.

Soient n , $k1$, $k2$ et $k3$ des entiers positifs. Dans le problème d'échecs, nous considérons un échiquier de dimensions $n \times n$, ainsi que **k1** tours, **k2** fous et **k3** cavaliers.

Ces pièces ont une différente manière de se déplacer. Les tours peuvent aller verticalement ou horizontalement. Les fous, quant à eux, attaquent aux diagonales. Enfin, les cavaliers se déplacent en **L**, c'est-à-dire de deux cases dans une direction combinées avec une case perpendiculairement.

Nous pouvons compter deux problèmes dans les problèmes d'échecs, le problème d'indépendance ainsi que le problème de domination.

- Le problème d'indépendance consiste à déterminer s'il est possible d'assigner à chacune des pièces une position distincte sur l'échiquier de sorte qu'aucune pièce ne menace une autre pièce.
- Le problème de domination consiste à déterminer s'il est possible d'assigner à chacune des pièces une position distincte sur l'échiquier de sorte que chaque case soit occupée ou menacée par au moins une pièce.

De plus, nous ne considérons pas des pièces comme obstacles à d'autres pièces, c'est-à-dire que pour dominer une case, une pièce peut passer par dessus une autre.

Afin d'exprimer ces deux problèmes par un CSP équivalent, nous avons commencé par écrire les variables, le domaine ainsi que les notations utilisées dans les contraintes pour ceux-ci.

L'ensemble des variables est composé de toutes les coordonnées des pièces. Les tours sont notées t_i et t'_i , les fous f_i et f'_i et finalement les cavaliers h_i et h'_i .

Le domaine est compris entre **1** et **n**, **n** indiquant la taille de l'échiquier.

Pour les notations, nous avons la dimension de l'échiquier ($n \times n$), ainsi que les notations pour représenter les différentes pièces (k_1, k_2, k_3). De plus, **m** nous indique le nombre total de pièces que nous avons ($k_1 + k_2 + k_3$). Enfin, **y** représente la suite de toutes les coordonnées des pièces, celles-ci sont alternées. Ainsi, les coordonnées x sont représentées sans les guillemets et les coordonnées y avec. Les **2 x k1** premières sont les tours, les **2 x k2** suivantes les fous et les **2 x k3** dernières les cavaliers.

1.1.1 Question 1

La première question nous demandait d'exprimer une instance quelconque du problème d'indépendance. L'ensemble des contraintes prend 4 contraintes pour ce problème.

$$C = \{c_1 \cup c_2 \cup c_3 \cup c_4\}$$

$$c_1 = (y, \{(v_1, \dots, v_{2m}) \in D^{2m} \mid \forall 1 \leq i \neq j \leq m, (v_{2i-1} \neq v_{2j-1}) \vee (v_{2i} \neq v_{2j})\})$$

La première contrainte nous indique que deux pièces ne peuvent pas se trouver au même endroit sur l'échiquier. Nous l'avons implémentée comme ceci, pour un ensemble $(\mathbf{v1}, \dots, \mathbf{v2m}) \in D^{2m}$, il existe un **i** et **j**, différent et compris entre 1 et **m**, le nombre de pièce, tel que ces deux pièces ne peuvent pas avoir les mêmes coordonnées.

L'ensemble est jusque **v2m** car dans **y** nous avons le nombre de pièce multipliées par 2, pour avoir toutes les coordonnées. De la même manière, nous multiplions les indices **i** et **j** par deux, car nous les avons bornés jusque **m**, et pour avoir la coordonnée **y** nous avons besoin de la valeur juste à cotée qui se trouve dans **y**. Pour rappel, dans **y**, par exemple pour la première tour, nous avons mis t_1 , suivit de $t'1$, pour représenter ses coordonnées.

Pour les trois autres contraintes de ce problème, l'ensemble reste le même c'est seulement les conditions sur les deux pièces qui changent.

$$c_2 = (y, \{(v_1, \dots, v_{2m}) \in D^{2m} \mid \forall 1 \leq i \leq k_1, \forall 1 \leq j \leq m, (v_{2i-1} \neq v_{2j-1}) \wedge (v_{2i} \neq v_{2j})\})$$

La seconde contrainte se porte sur les tours, elle indique qu'aucune pièce ne peut se trouver sur l'horizontal ou la verticale d'une tour. Pour ceci, les bornes de **i** sont entre 1 et **k1**, afin de ne prendre en compte que les tours, et **j** se trouve entre 1 et **m**. Ces deux pièces ne peuvent être à la même colonne et à la même ligne.

$$c_3 = (y, \{(v_1, \dots, v_{2m}) \in D^{2m} \mid \forall k_1+1 \leq i \leq k_1+k_2, \forall 1 \leq j \leq m, \forall k \in \{-n, n\},$$

$$[((v_{2i-1} \neq v_{2j-1} + k) \vee (v_{2i} \neq v_{2j} + k)) \wedge ((v_{2i-1} \neq v_{2j-1} - k) \vee (v_{2i} \neq v_{2j} + k))]]$$

La troisième contrainte est sur la portée des fous, aucune pièce ne peut être sur les diagonales d'un fou. Pour cette contrainte, le **i** est entre **k1+1** et **k1+k2**, afin de n'avoir que les fous, étant donné que les **k1** représentent les tours nous faisons un saut du nombre de tour présentes. Les bornes de **j**, par contre, restent les mêmes, de 1 à **m**. Dans cette contraintes, nous devons tenir en compte un autre indice qui définira les déplacements vers les diagonales, nous l'avons appelé **k** et celui-ci est compris entre **{-n, n}**. Nous vérifions donc dans les 4 sens s'il n'y a pas d'autre pièces présentes.

$$c_4 = (y, \{(v_1, \dots, v_{2m}) \in D^{2m} \mid \forall k_1 + k_2 + 1 \leq i \leq m, \forall 1 \leq j \leq m,$$

$$\forall k \in \{-2, 2\}, \forall l \in \{-1, 1\},$$

$$[((v_{2i-1} \neq v_{2j-1} + k) \vee (v_{2i} \neq v_{2j} + l)) \wedge ((v_{2i-1} \neq v_{2j-1} + l) \vee (v_{2i} \neq v_{2j} + k))]]$$

Enfin, pour la dernière contrainte sur les cavaliers, nous devons vérifier qu'aucune pièce ne se trouve sur les déplacements en **L**. Pour cela, la borne inférieure de **i** est maintenant **k1+k2+1**, nous sautons donc toutes les tours et tous les fous. Et donc la borne supérieure est bien le nombre total de pièce : **m**. En plus de ceci, nous avons un **k** compris entre **{-2, 2}** et un **l** entre **{-1, 1}**, afin de pouvoir indiquer le déplacement vers une direction de deux cases suivit d'un mouvement perpendiculaire.

1.1.2 Question 2

Une instance quelconque du problème domination devait être exprimée pour la seconde question. L'ensemble des contraintes est composé de 2 contraintes.

$$C = \{c_1 \cup c_2\}$$

Où la première reste la même que pour la première question, 2 pièces ne peuvent pas être sur la même coordonnée dans l'échiquier.

$$c_1 = (y, \{(v_1, \dots, v_{2m}) \in D^{2m} \mid \forall 1 \leq i \neq j \leq m, (v_{2i-1} \neq v_{2j-1}) \vee (v_{2i} \neq v_{2j})\})$$

La deuxième contrainte indique qu'une pièce est soit :

- Occupée par une pièce
- Menacée par une tour
- Menacée par un fou
- Menacée par un cavalier

$$c_2 = (y, \forall 1 \leq i \leq n, \forall 1 \leq j \leq n, u1_{i,j} \cup u2_{i,j} \cup u3_{i,j} \cup u4_{i,j})$$

Les indices **i** et **j** sont bornés entre 1 et **n**, nous les passons en paramètre aux 4 ensembles exprimant les conditions citées ci-dessus.

$$u1_{i,j} = \{(v_1, \dots, v_{2m}) \in D^{2m} \mid \exists k \in \{1, \dots, m\}, v_{2k-1} = i \wedge v_{2k} = j\}$$

Le premier ensemble signifie que pour une case dont les coordonnées sont $\{\mathbf{i}, \mathbf{j}\}$, il existe une pièce quelconque ayant les même coordonnées. L'indice \mathbf{k} , appartenant à $\{\mathbf{1}, \mathbf{m}\}$, représente donc une pièce.

$$u2_{i,j} = \{(v_1, \dots, v_{2m}) \in D^{2m} \mid \exists k \in \{1, \dots, k1\}, v_{2k-1} = i \vee v_{2k} = j\}$$

Dans le second ensemble, nous voulons qu'une tour aie la même ligne ou la même colonne que la case $\{\mathbf{i}, \mathbf{j}\}$. L'indice \mathbf{k} symbolise les tours que nous avons, ses bornes étant entre 1 et $\mathbf{k1}$.

$$u3_{i,j} = \{(v_1, \dots, v_{2m}) \in D^{2m} \mid \exists k \in \{k1 + 1, \dots, k1 + k2\}, p \in \{-n, n\}, \\ ((v_{2k-1} = i + p \wedge v_{2k} = j + p) \vee ((v_{2k-1} = i - p \wedge v_{2k} = j + p))))\}$$

De même le troisième ensemble indique qu'il existe un fou dans les diagonales d'une case $\{\mathbf{i}, \mathbf{j}\}$. Ici, vu que nous considérons les fous, les bornes de \mathbf{k} sont comprises entre $\mathbf{k1} + \mathbf{1}$ à $\mathbf{k1} + \mathbf{k2}$. Et comme pour la première question, nous devons intégrer un autre indice, que nous avons appelé ici \mathbf{p} , afin de se déplacer dans les diagonales d'une case.

$$u4_{i,j} = \{(v_1, \dots, v_{2m}) \in D^{2m} \mid \exists k \in \{k1 + k2 + 1, \dots, m\}, p \in \{-2, 2\}, l \in \{-1, 1\}, \\ ((v_{2k-1} = i + k \wedge v_{2k} = j + l) \vee ((v_{2k-1} = i + l \wedge v_{2k} = j + k))))\}$$

Enfin le dernier ensemble exprime qu'il y a un cavalier aux déplacements \mathbf{L} d'une case se trouvant aux indices $\{\mathbf{i}, \mathbf{j}\}$. Avec le même raisonnement qu'avant, les indices \mathbf{k} , des cavaliers, sont compris entre $\mathbf{k1} + \mathbf{k2} + \mathbf{1}$ et \mathbf{m} . De plus, pour faire les déplacements en \mathbf{L} nous avons besoin de deux autres indices, appelés ici \mathbf{p} et \mathbf{l} .

1.2 ChocoSolver

La question 3 et la question bonus ont été implémentées dans le projet **CSPSolver**.

1.2.1 Question 3

Cette question nous demande d'implémenter un programme qui, étant donné une instance du problème de domination ou d'indépendance, en calcule une solution à l'aide de ChocoSolver.

Un format d'entrée et de sortie est fixé, où nous pouvons passer en paramètre les différentes options que nous voulons pour exécuter le programme. Afin de parser les arguments, nous avons utilisé la librairie Argparse4j. Pour cela, un fichier jar est donné qu'il faut rajouter dans le buildpath du projet en maven.

Certains paramètres comme la dimension, le nombre de tours, cavaliers, fous, sont obligatoires. En ce qui concerne le type de problème demandé, nous vérifions qu'il y a bien qu'un seul paramètre entré, soit un **"-i"** (indépendance) soit un **"-d"**.

Une fois que tous les arguments sont initialisés, nous créons un Model ainsi que toutes les pièces demandées par l'utilisateur. Ensuite, pour ces pièces nous allons appliquer le code pour l'indépendance ou la domination. Et enfin, faire appel à **model.getSolver().findSolution()** afin de trouver notre solution.

Afin d'afficher la solution, nous commençons par créer un tableau de dimension **n**. Puis, nous reprenons les positions x et y de toutes les pièces, ainsi que la première lettre de la pièce correspondante, et imprimons la solution.

Pour le projet, nous avons décidé de créer une structure générique, où nous avons les classes tour, fou et cavalier héritant d'une classe Pièce. Cette classe Pièce contient les méthodes que nous voulons implémenter pour tous les types de pièces existants, et nous permet de nous renvoyer les coordonnées x et y de celles-ci. De plus, dans cette classe nous avons créé une méthode qui, prenant en paramètre des coordonnées x et y, peut vérifier si une pièce y existe déjà.

Les trois autres classes comportent les mêmes méthodes. Dans la première méthode, checkIndependency, nous vérifions qu'aucune pièce ne se trouve aux déplacements que peut effectuer la pièce en question. Cette fonction parcourt toutes les pièces, vérifie d'abord qu'aucune autre n'est présentes aux coordonnées x et y, puis applique les différentes contraintes que nous avons exprimée dans le CSP plus haut. Par exemple, pour les tours se serait vérifier qu'il n'y a aucune pièce ayant la même ligne ou colonne.

La seconde méthode présente dans les 3 sous-classes, s'appelle inDomain, et prend en paramètre une case avec ses coordonnées. Elle nous renvoi, bien évidemment, le domaine tel que la case (x,y) se trouve dans le domaine de la pièce en question. Par exemple pour les fous, on vérifie pour les diagonales de cette case si une pièce fou s'y trouve.

La résolution du problème de domination se fait comme suit :

- Créer une liste de contraintes, everyConstraint
- Vérifier qu'aucune pièce ne se trouve à cet endroit
- Pour toutes les cases, créer une contrainte qui vérifie qu'elle se trouve dans le domaine d'une pièce, que ce soit une tour un fou ou un cavalier, ou qu'il y a une pièce présente à cette position.
- Ajouter cette contrainte, dans la première liste de contraintes : everyConstraint
- Dire au model de respecter toutes les contraintes pour toutes les cases

1.2.2 Question bonus

Pour la question bonus, nous avons créé une classe `PieceGenerique` qui, comme les fous, tours et cavaliers, hérite de la classe parent `Pièce`.

Nous représentons le domaine d'une pièce quelconque étant la fonction booléenne $\phi_{x,y}(i,j)$ qui renvoie vrai ou faux si la case/pièce (i,j) est dominée par la case (x,y) . C'est avec cette fonction qu'on peut créer les contraintes de domination et indépendance pour une pièce quelconque tel que :

$$c_{ind} = (y, \{(v_1, \dots, v_{2m}) \in D^{2m} \mid \forall k_1 + k_2 + k_3 \leq i \leq m, \forall k_1 + k_2 + k_3 \leq j \leq m, \\ \neg \phi_{v_{2i-1}, v_{2i}}(v_{2j-1}, v_{2j})\})$$

$$c_{dom} = (y, \{(v_1, \dots, v_{2m}) \in D^{2m} \mid \forall 1 \leq i \leq n, \forall 1 \leq j \leq n, \\ (\exists k, \phi_{v_{2k-1}, v_{2k}}(i, j) \vee (v_{2k-1} = i \wedge v_{2k} = j))\})$$

Nous demandons en arguments grâce au paramètre "**-g**" le nombre de pièce générique qu'un utilisateur demande. De plus nous avons défini le domaine d'une pièce avec des équations linéaires. Ces équations sont mises dans un fichier que l'utilisateur va mettre en paramètre grâce à **-file**. Les équations sont de la forme suivante : $\mathbf{a} * \mathbf{x} + \mathbf{b} * \mathbf{y} + \mathbf{c} * \mathbf{i} + \mathbf{d} * \mathbf{j} + \mathbf{e} = \mathbf{0}$, l'opérateur peut bien sûr être autre qu'un $=$ ($>$, $>=$, $<$, $<=$, $!=$). \mathbf{x} et \mathbf{y} représentent les coordonnées d'une pièce, et \mathbf{i} et \mathbf{j} celles de la pièce à dominer. Dans le fichier donné par l'utilisateur, le format est : a b c d e op., et nous supposons qu'il n'y a aucune erreur dans celui-ci.

Une fois que nous récupérons ces données, nous appliquons à la `pieceGenerique` le problème d'indépendance ou de domination.

Nous avons utilisé le `model.scalar()` afin de pouvoir implémenter les contraintes des équations linéaires. Le `model.scalar()` nous permet en effet d'y mettre en paramètres une liste des `IntVar`, une liste des différentes variables $\mathbf{x}, \mathbf{y}, \mathbf{i}$ et \mathbf{j} , ainsi que l'opérateur, et le coefficient. Afin de respecter le format du `model.scalar()`, nous passons le coefficient de l'autre côté de l'équation, et donc le mettons en négatif.

1.2.3 Question 4

L'implémentation de cette question se trouve dans le projet **HorsemanMinimizer**. Le CSP s'y trouve dans le fichier **quest4.pdf**.

Pour ce problème, un entier n est passé en paramètre, qui indique la taille de l'échiquier. Il nous a été demandé de minimiser le nombre de cavaliers pouvant dominer l'échiquier. La sortie comprend le nombre minimum de cavaliers requis ainsi qu'un tableau affichant les positions de ces cavaliers sur l'échiquier.

Nous commençons par initialiser toutes les cases du tableau. Ensuite, nous vérifions que chaque case respecte bien la contrainte d'être soit dominée, soit occupée par un cavalier.

L'ensemble des contraintes est défini comme ceci :

$$C = (y, \{z \in D^{n^2} \mid \forall 1 \leq i \leq n, \forall 1 \leq j \leq n, \exists k \in \{2, -2\}, \exists l \in \{1, -1\}, \\ a_{i+k, j+l} = 1 \vee a_{i+l, j+k} = 1 \vee a_{i, j} = 1\})$$

Comme dans les premières questions, le **k** représente 2 déplacements d'un cavalier dans un sens, suivit d'un déplacement perpendiculaire exprimé par **l**.

Une fois que ces contraintes sont rajoutées dans le model, il faut minimiser le nombre de cavalier. Nous faisons d'abord la somme de chevaliers grâce au `model.sum()`. Puis, `chocoSolver` propose un `model.setObjective(minimise)`, afin de minimiser le problème.

Nous itérons le problème jusqu'à avoir la meilleure solution, et affichons toutes les possibilités.

2 Surveillance de musée

Le code pour la dernière question, sur la surveillance de musée, se trouve dans **MuseumMonitorer**. Le CSP de cette question, quand à lui, est dans le fichier **musee.pdf**.

2.1 CSP

Pour les notations de ce problème, nous retrouvons :

- $n \times n$ la dimension de l'échiquier
- $a_{i,j}$ = case (i,j) du terrain (elle est égale à 1 si c'est un mur 0 sinon)
- y variable représentant l'orientation des caméras
- y' variables indiquant si une caméra est présente
- z contient les valeurs de y
- z' représentant les valeurs de y'
- yy' = concaténation de y et y'
- zz' = concaténation de z et z'

L'ensemble des variables est constitué de :

- Variables indiquant la direction de la caméra qui se trouve dans cette case
- Variables par case indiquant que la case est occupée par une caméra

Le problème possède deux domaines, le premier comprenant toutes les directions qu'une caméra peut prendre, $D_1 = \{Nord, Sud, Est, Ouest, \emptyset\}$.

Et le deuxième étant $D_2 = \{0, 1\}$.

L'ensemble des contraintes est défini par l'union de 3 contraintes.

$$c_1 = (yy', \{vv' \in D_1^{n^2} \times D_2^{n^2} \mid \forall 1 \leq i \leq n, \forall 1 \leq j \leq n, \\ ((v_{i,j} = \emptyset \wedge v'_{i,j} = 0) \vee (v_{i,j} \neq \emptyset \wedge v_{i,j} = 1))\})$$

Cette première contrainte, nous dit que $x'_{i,j} = 1$ ssi la caméra se trouve aux mêmes coordonnées. Nous avons les indices de **i** et **j** allant de 1 à **n**, la taille de l'échiquier, afin de prendre en considération toutes les cases de celui-ci.

$$c_2 = (y', \{v' \in D_2^{n^2} \mid \forall 1 \leq i \leq n, \forall 1 \leq j \leq n, (a_{i,j} = 1 \wedge v'_{i,j} = 0) \vee (a_{i,j} = 0)\})$$

La seconde contrainte permet de vérifier qu'une caméra ne se trouve pas dans un mur, ses coordonnées doivent être différentes d'un mu existant.

$$c_3 = (yy', \{vv' \mid \forall 1 \leq i \leq n, \forall 1 \leq j \leq n, (a_{i,j} = 1) \vee (v'_{i,j} = 1) \vee \\ \left(a_{i,j} = 0 \bigwedge \left((\exists k \in \mathbb{Z}, (v_{i+k,j} = Ouest \wedge (\forall 1 \leq l < k, v'_{i+l,j} = 0 \wedge a_{i+l,j} = 0))) \right. \right. \\ \bigvee \\ \left(\exists k \in \mathbb{Z}, (v_{i-k,j} = Est \wedge (\forall 1 \leq l < k, v'_{i-l,j} = 0 \wedge a_{i-l,j} = 0))) \right. \\ \bigvee \\ \left(\exists k \in \mathbb{Z}, (v_{i,j+k} = Sud \wedge (\forall 1 \leq l < k, v'_{i,j+l} = 0 \wedge a_{i,j+l} = 0))) \right. \\ \bigvee \\ \left. \left. \left(\exists k \in \mathbb{Z}, (v_{i,j-k} = Nord \wedge (\forall 1 \leq l < k, v'_{i,j-l} = 0 \wedge a_{i,j-l} = 0)) \right) \right) \right\}) \quad (1)$$

Enfin, la troisième et dernière contrainte exprime que chaque case doit être occupée par une caméra, ou être dominée par une caméra par le nord, le sud, l'est ou l'ouest.

Le $(a_{i,j})$ exprime la présence d'une caméra. Et comme on le voit, on vérifie pour chaque case dans ces 4 directions s'il y a une domination par une caméra.

2.2 ChocoSolver

L'utilisateur doit entrer en paramètre le nom du fichier dans lequel se trouve un tableau rectangulaire représentant la topologie de la salle à surveiller. Une case est soit vide, soit occupée par un obstacle. Un obstacle est symbolisé par *. De plus, les 4 murs sont toujours des obstacles. La sortie du code donne le nombre minimal de capteurs nécessaires, ainsi qu'un tableau montrant la position des caméras.

Dans l'implémentation du problème, nous commençons par parser le fichier mis en paramètre par l'utilisateur. Nous enregistrons dans une variable **murs**, contenant des booleans, le tableau. Nous travaillons avec des booleans, et donc s'il y a un obstacle a une position $\{x, y\}$, nous mettons le boolean à true, false sinon. Nous considérons qu'il n'y a pas d'erreurs dans le fichier mis en paramètre par l'utilisateur.

Nous devons donc minimiser le nombre de caméra présentes dans la pièce d'attente. Pour cela, nous devons respecter plusieurs contraintes. Premièrement, nous devons faire appliquer au model qu'une caméra existe seulement si son orientation n'est pas nulle, elle doit aller vers le nord, le sud, l'est ou l'ouest. Ensuite, il ne peut y avoir de caméra dans les murs, elles doivent être dans un espace vide. Finalement, chaque case doit être dominée ou occupée par une caméra. Afin de respecter cette dernière contrainte, pour chaque case nous vérifions pour ces 4 directions si une caméra y pointe.

Tout comme pour la question 4, nous itérerons toutes les solutions possibles jusqu'à avoir la meilleure.

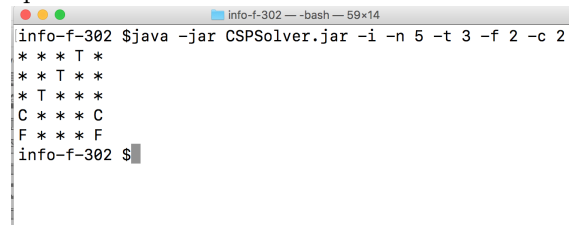
3 Résultats

Nous avons créé des fichiers .jar afin de pouvoir exécuter les différentes implémentations.

3.1 Exemples

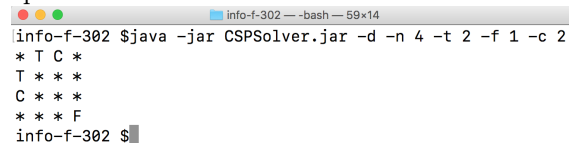
Voici quelques exemples :

- Problème d'indépendance avec un échiquier de dimension 5 x 5 , 3 tours, 2 fous ainsi que 2 cavaliers



```
info-f-302 $java -jar CSPSolver.jar -i -n 5 -t 3 -f 2 -c 2
* * * T *
* * T * *
* T * * *
C * * * C
F * * * F
info-f-302 $
```

- Problème de domination avec un échiquier de dimension 4 x 4 , 2 tours, 1 fous ainsi que 2 cavaliers



```
info-f-302 $java -jar CSPSolver.jar -d -n 4 -t 2 -f 1 -c 2
* T C *
T * * *
C * * *
* * * F
info-f-302 $
```

- Une implémentation de la question bonus, pour le problème d'indépendance, avec un échiquier de dimension 5 x 5 , 2 tours, 1 fous, 2 cavaliers

et 2 pièces génériques. Les équations linéaires de ces pièces génériques se trouvent dans le fichier pieceGen.txt.

```
info-f-302 $ java -jar CSPSolver.jar -i -n 5 -t 2 -f 1 -c 2
-g 2 -file pieceGen.txt
C * * * *
* T * * *
C * * * G
G * * * F
* * T * *
info-f-302 $
```

— Le fichier pieceGen.txt fournit par l'utilisateur prend cette forme :

```
1 2 1 4 4 =
1 2 -1 -4 -4 =
3 -2 4 5 6 =
```

— Pour la question4, minimisation des cavaliers, nous devons entrer en paramètre la taille de l'échiquier.

```
info-f-302 $ java -jar HorsemanMinimiser.jar 4
Nombre minimum: 4
****
****
*CC*
*CC*
info-f-302 $
```

— La dernière question prend en paramètre le nom du fichier contenant l'échiquier. Le code renvoie affiche toute les solutions jusqu'à atteindre l'optimale, celle qui utilise le moins de caméra.

```
info-f-302 $ java -jar MuseumMonitorer.jar Museumboard.txt
20
*****
*N*** E *
*SE *N O*
* * OE *
* N*****
* *S O O*
***N O***
*EE ON*
*N* O***
*****

19
*****
*N*** E *
*SE *N O*
* * OE *
* N*****
* *S O*
***N O***
*NE ON*
*N* O***
*****
```

```
info-f-302 — -bash — 61x27
*N*  O***
*****

12
*****
*S***  O*
* E *  O*
* *    O*
* N*****
* *E    *
***  O***
* N    O*
*N*  O***
*****

11
*****
* ***  O*
* O*  O*
* *    O*
* N*****
*N*    O*
***E   ***
* E     *
*N*  O***
*****
info-f-302 $
```