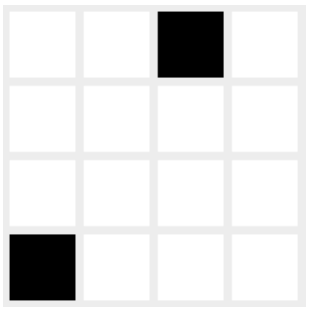
L'énigme des N reines

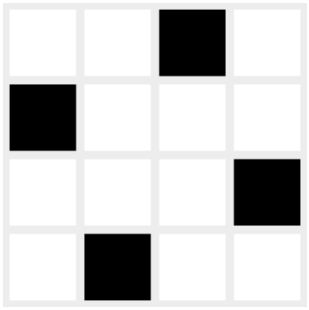
POnGL, TP1, révisions de programmation Java.

1 Présentation

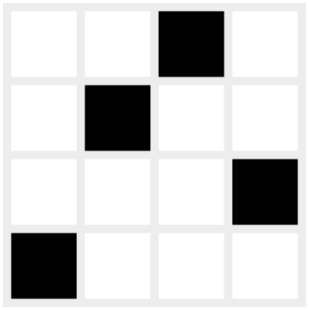
Le problème des N reines s'énonce ainsi : peut-on placer N reines sur un échiquier de taille N, de sorte qu'aucune reine ne soit menacée par une autre ? Autrement dit, nous disposons d'un tableau à deux dimensions carré de côté N, et nous voulons placer N points dans ce tableau de sorte qu'il n'y ait jamais deux points sur une même ligne, une même colonne, ou une même diagonale.



Configuration valide



Solution complète



Configuration invalide

Dans ce TP, nous allons créer un plateau de jeu graphique qui proposera à un joueur de résoudre cette énigme. Le joueur pourra effectuer les deux actions suivantes sur le plateau :

— Cliquer sur une case libre pour y placer une reine.

— Cliquer sur une case occupée pour retirer la reine qui s'y trouve.

On ajoutera également au jeu deux boutons :

— Un bouton de validation, qui indique si la configuration actuelle du plateau est acceptable (c'est-à-dire qu'aucune des reines présentes n'en menace une autre, indépendamment du nombre de reines présentes).

— Un bouton de demande d'indice. Si la configuration actuelle peut être complétée en une solution, alors l'interface propose au joueur une case sur laquelle jouer. Sinon elle indique que le joueur est bloqué.

Dans tout ce TP, on utilisera la mini-bibliothèque `IG` fournie, et on complètera le squelette de fichier `NReines.java`. Le tout peut être téléchargé à l'adresse :
`http://www.lri.fr/~blsk/POnGL/NReines.zip`

2 Mise en place de l'interface graphique

Mots-clés : `if`, `for`, création d'un objet, lecture d'un argument, conversion d'une chaîne de caractères en un entier, surcharge d'un constructeur.

Pour commencer, on veut créer un plateau dans lequel un clic fait passer une case du blanc au noir ou inversement.

1. Compléter les constructeurs des classes `Plateau` et `Case`, ainsi que la méthode `clicGauche` de la classe `Case`. On donnera pour l'instant au plateau une taille par défaut de `8` et les signatures de ces constructeurs et méthodes seront les suivantes :
   ```
   public Plateau()
   public Case(Plateau p)
   public void clicGauche()
   ```
   Pour colorer une case en noir, on pourra utiliser l'appel de méthode
   `setBackground(Color.BLACK);`

2. Créer un deuxième constructeur pour la classe `Plateau`, de signature
   `public Plateau(int taille)`
   auquel la taille du plateau est passée en argument, et modifier la méthode `main` pour qu'elle utilise ce nouveau constructeur avec une taille différente de `8` (par exemple `6`).
   Note : il faudra modifier la ligne
   `super(8, 8);`
   pour que l'affichage tienne compte de ces dimensions différentes.
3. Modifier la méthode `main` pour que la taille soit maintenant passée comme un argument en ligne de commande. Cet argument sera le premier élément du tableau `args`, c'est-à-dire un objet de type `String` qu'il va falloir convertir en un `int`. Il existe une méthode pour cela : à vous de trouver son nom sur internet ou dans la documentation.

Mode d'emploi pour donner des arguments à la méthode `main` :
— Depuis un terminal, pour créer un tableau de taille `6` il faudra lancer le programme avec la ligne de commande
  `java NReines 6`
— Depuis Eclipse, au lieu de sélectionner *Run as Java application*, il faudra sélectionner la ligne inférieure *Run configuration...*, puis aller dans l'onglet *Arguments* et donner l'argument `6` dans le principal champ de texte (il suffit de le faire une fois).

3 Validation d'une configuration

Mots-clés : tableaux à deux dimensions, `if`, `for`, `for each`.

L'objectif de cette partie est de programmer le comportement du bouton de validation. Pour chaque méthode écrite dans cette partie il vous est demandé de créer des tests, que vous inclurez dans la méthode `main`. N'hésitez pas à créer des tableaux à la main, et à faire appel à des méthodes comme `System.out.println( ... )` pour suivre les résultats de vos tests. Remarque : pour certaines questions, il sera pertinent d'ajouter des attributs aux classes présentes dans le squelette.

1. Pour pouvoir parcourir les cases de l'échiquier, il faudra que celles-ci soient regroupées dans un tableau à deux dimensions, qui sera un attribut de l'une des classes de l'exercice. Oui, mais laquelle ? Et dans quelle classe définir les méthodes qui devront agir sur ce tableau ?
2. Écrire une méthode de signature
   `private int compteLigne(Case[] l)`
   qui compte le nombre de reines présentes dans la ligne `l`, et déduisez-en une méthode de signature
   `private boolean verifieLignes()`
   qui renvoie `true` si et seulement si aucune ligne de notre plateau ne contient plus d'une reine. Pour ces deux méthodes on utilisera des constructions de type `for each`.
3. Procéder de même pour les colonnes, puis pour les diagonales. On utilisera cette fois des constructions `for` traditionnelles.
4. En déduire une méthode de signature
   `public boolean verifieConfiguration()`
   vérifiant la validité d'une configuration, et compléter la méthode `void clicGauche()` de la classe `Validation`. On veut que le bouton s'affiche en vert si la configuration est valide, et en rouge sinon.

4 Exploration des solutions et proposition d'un indice

Mots-clés : récursivité, backtrack.

L'objectif de cette partie est de programmer le comportement du bouton de demande d'indice. Comme à la partie précédente, chaque méthode écrite devra être méthodiquement testée.

On va pour cela utiliser une méthode qui a pour objectif de tester toutes les manières possibles de compléter le plateau jusqu'à trouver une solution. Les deux principaux ingrédients sont :

- À chaque étape, choisir une case, y placer une reine, et rappeler la méthode *récursivement* avec cette nouvelle configuration de départ.
- Si on ne trouve pas de solution, revenir au dernier choix et essayer une autre case (*backtrack*).

Plus précisément, on utilisera la procédure suivante :

(A) Vérifier la validité de la configuration actuelle.

(B) Déterminer la première ligne non occupée.

(C) Placer une reine sur la première case de cette ligne.

(D) Revenir à l'étape (A) avec cette nouvelle configuration.

Si on trouve une solution complète, c'est-à-dire si on a placé une reine sur chaque ligne et que l'ensemble est valide, alors on s'arrête. Si on arrive à une configuration invalide, alors on annule le choix qui avait été fait à l'étape (C), et on essaye à nouveau avec la deuxième case de la ligne, puis la troisième, et ainsi de suite jusqu'à avoir trouvé une solution ou avoir vu qu'aucune des cases ne permettait d'arriver à une solution.

1. Écrire la procédure proposée sous la forme d'un algorithme plus précis (sur papier).
2. Écrire une méthode de signature
   `public boolean verifieResolubilite()`
   qui met en œuvre cet algorithme, et compléter la méthode `void clicGauche()` de la classe `Indice`. On veut que le bouton s'affiche en vert si la configuration peut être complétée en une solution, et en rouge sinon.
3. Compléter le code précédent pour que dans le cas où une solution est trouvée, le jeu affiche en bleu une case dans laquelle il serait judicieux de placer une reine. On pourra introduire des attributs `int indiceL` et `int indiceC` dans lesquels stocker les coordonnées de l'une des cases testées lors de l'exploration.

5 Questions bonus

Mots-clés : lancement et rattrapage d'exceptions.

1. Dans la méthode `main`, faire en sorte que si un argument est fourni, alors un plateau de taille correspondante est créé, mais que si aucun argument n'est fourni (ou si l'argument n'est pas un entier valide), alors la taille par défaut `8` est utilisée.
2. Lors d'un appel à `verifieConfiguration()`, faire en sorte qu'en cas de configuration invalide, deux reines se menaçant l'une l'autre sont affichées en rouge.