Le solitaire V2

# Présentation

Jeu de plateau à un joueur

Le plateau est constitué d’un ensemble de cases carrées jointives par les bords.

Il existe principalement deux configurations :

|  |  |
| --- | --- |
| Le plateau classique (33 cases) | Le plateau français (37 cases) |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | |  |  |  |  |  |  |  | |  |  |  |  |  |  |  | |  |  |  |  |  |  |  | |  |  |  |  |  |  |  | |  |  |  |  |  |  |  | |  |  |  |  |  |  |  | |  |  |  |  |  |  |  | | |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | |  |  |  |  |  |  |  | |  |  |  |  |  |  |  | |  |  |  |  |  |  |  | |  |  |  |  |  |  |  | |  |  |  |  |  |  |  | |  |  |  |  |  |  |  | |  |  |  |  |  |  |  | |

Au départ, chaque case est occupée par une pierre, sauf au moins une case.

Dans les jeux classique et français, seule la case centrale est vide.

Les cases contigües sont celles qui partagent un même bord. Il n’y a pas de mouvements en diagonale.

Un mouvement consiste à choisir une pierre A et à la faire sauter par-dessus une pierre B contigüe pour autant que A atterrisse dans une case vide du plateau. Alors B est retiré. Cela revient à déterminer trois cases contiguës alignées, les deux premières contenant une pierre, la troisième à une des extrémités est vide, et d’inverser la présence des pierres dans ces trois cases.

Si on note x la présence d’une pierre, o une case sans pierre, les 4 mouvements de base sont :

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| x x o | → | o o x |  | o x x | → | x o o |  | x x o | → | o o x |  | o x x | → | x o o |

Chaque mouvement fait donc décroitre de exactement 1 unité le nombre de pierres sur le plateau.

L’objectif est d’arriver à ne conserver qu’une seule pierre sur le plateau.

La position de la dernière pierre n’est pas déterminante.

S’il y a N pierres sur le plateau initial, il faut exactement N-1 mouvements pour atteindre une solution (si elle existe).

# Objectif

Evidemment, l’intérêt du jeu est d’être résolu manuellement. Mais je ne suis pas doué. Et je me pose une question existentielle : y a-t-il une solution quelle que soit la pierre de départ retirée d’un plateau complet ?

Lançons-nous dans l’écriture d’un programme de recherche automatique de solutions.

Bon, le plateau classique n’offre que peu de résistance aux premières versions de recherche automatique. Le Français en revanche se révèle plus ardu.

Le volume des situations se révèle très élevé. La mémoire du PC explose.

En fait, la situation « classique » peut être résolue naïvement, et plus rapidement encore dès qu’on tient compte des symétries. Mais le plateau Français est plus coriace.

D’où la stratégie suivante :

* Tenter de gérer une mémoire des situations déjà rencontrées pour ne pas lancer de recherches redondantes.
* Exploiter les symétries pour encore diminuer les redondances.
* Minimiser l’empreinte de chaque situation pour optimiser l’utilisation de la mémoire.

Et surtout, devant les échecs répétés des premières tentatives, mettre au point la stratégie suivante :

* Pour un plateau donné, calculer la liste des situations (aux symétries près) où une pierre manque exactement, appelons cet ensemble EI.
* A partir des situations de EI, calculer toutes les situations (aux symétries près) possibles en jouant un nombre déterminé de coups, nombre nommé ND, par exemple 9. L’ensemble est nommé ED. Stocker cet ensemble dans un fichier réutilisable ED.dat .
* Cet ensemble enregistre aussi, pour chaque situation SD de ED, la liste des situations initiales qui peuvent y mener (à une symétrie près).
* Calculer l’ensemble des situations gagnantes où il n’y a qu’une pierre sur le plateau (aux symétries près), ensemble qu’on nomme EG.
* Calculer à partir des situations de EG toutes les situations dites finales qui peuvent mener en un nombre donné de mouvements, nombre nommé NF, par exemple 12, à l’une de ces situations gagnantes (aux symétries près). L’ensemble est noté EF. Elles sont stockées dans un fichier EF.dat .
* Mettre au point un programme qui sait répondre à une sollicitation pour s’arrêter en sauvegardant son travail en cours. Ce programme peut reprendre à partir de cette position, mais en ayant libéré toute la mémoire des situations intermédiaires qu’il doit recalculer.
* Ce programme teste une à une les situations de ED. Et donc sait enregistrer la situation actuelle lors d’une demande d’arrêt pour pouvoir la reprendre au redémarrage.
* Pour chaque situation SD de ED, s’il trouve un ensemble de mouvements qui mène à une situation SF de EF (aux symétries près), alors il marque comme résolues les situations de EI (non encore résolues) qui peuvent mener à SD et leur associe cette situation SD. Il enregistre aussi les mouvements qui ont mené de SD à SF.
* Il peut aussi calculer dans un second temps une suite de mouvements qui mène de ce SI à ce SD (à une symétrie près). En fait il recalcule des mouvements inverses qui permettent de passer de SD à un équivalent SI’ de SI. Il recalcule rapidement SF grâce aux mouvements qu’il a enregistrés auparavant, puis finit en recalculant une suite de mouvements qui mènent de SF à une situation gagnante SG de EG.
* En combinant ces trois suites de mouvements, on reconstitue une suite de mouvements qui mène de SI’ à SG. Le programme enregistre alors cette suite complète de mouvements.
* Ces situations de EI résolues sont flaggées dans EI. Si toutes sont flaggées, alors le programme a trouvé une solution quelle que soit la situation initiale de l’ensemble EI d’origine.

EI

ED

EF

EG

ND mouvements

NF mouvements

N pierres initialement sur le plateau de N+1 cases, ou plus

N – ND pierres

dans les situations

de ED

NF + 1 pierres

dans les situations

de EF

N – ND – NF – 1

mouvements

séparent ED de EF

## Fiche de jeu

C’est un fichier texte décrivant une situation initiale.

Exemple de contenu de fiche de jeu :

xxx

xxxxx

xxxxxxx

xxxxxxx

xxxxxxx

xxxxx

xxx

Une situation est une suite de lignes non vides constituées d’espaces ‘ ‘, de lettres ‘x’ ou ‘o’.

Le ‘x’ désigne une pierre, le ‘o’ une case vide.

L’algorithme calcule automatiquement la largeur du plateau L.

Petite sophistication : si le plateau initial est entièrement couvert de pierres (il n’y a pas de ‘o’), alors les situations initiales sont celles où une pierre exactement est enlevée du plateau (aux symétries près). Sinon, il n’y a qu’une situation initiale, et c’est celle décrite dans ce plateau.

## Données persistantes

Le programme constitue initialement une banque de données dans un sous-répertoire qui porte le nom de la fiche de jeu, au même emplacement que cette fiche de jeu.

Il y stocke une copie de la fiche de jeu, qu’il complète au fur et à mesure avec l’information lui permettant de reprendre le travail, et les solutions trouvées.

Il y stocke aussi les fichiers ED.dat et EF.dat qu’il ne recalcule pas lors des reprises.

## Calcul des données persistantes

Lors de l’initialisation, les paramètres ND et NF sont réglés à 1. Puis l’utilisateur pourra plus tard demander à augmenter cette taille, de nouveaux fichiers ED.tmp, EF.tmp sont calculés et remplacent les anciens après confirmation. On ne prévoit pas de diminution des valeurs de ces paramètres. En cas de problème, il faut effacer les fichiers manuellement et relancer l’initialisation, les versions où nd=nf=1 sont recréées.

## Fichier Pilote

Exemple de contenu de « fichier pilote » :

Lors d’une reprise, la description du plateau initial doivent être identiques à ceux de la fiche de jeu.

<?xml version="1.0" encoding="utf-8"?>

<solitaire>

<plateau>

xxx

xxxxx

xxxxxxx

xxxxxxx

xxxxxxx

xxxxx

xxx

</plateau>

<parametres nd="9" nf="12" />

<reprise idx="999999999" />

<solution incomplete="true">

<plateau>

oxx

xxxxx

xxxxxxx

xxxxxxx

xxxxxxx

xxxxx

xxx

</plateau>

<mouvement d="4" s="3" />

<mouvement d="18" s="11" />

</solution>

<presolution idxSD="56789">

<initial idxSI="2" />

<mouvement d="4" s="3" />

<mouvement d="18" s="11" />

</presolution>

</solitaire>

Le paramètre reprise/@idx pointe sur la position de l’enregistrement de situation dans le fichier ED.dat à partir duquel reprendre les recherches.

Les paramètres parametres/@nd et parametres/@nf, contiennent la taille (nombre de pierres) des situations décrites dans les fichiers ED.dat et EF.dat.

Une solution est une suite de mouvements à effectuer pour résoudre le jeu.

Un mouvement est un couple (d, s). ‘d’ est l’indice de la pierre déplacée, s est l’indice de la pierre par-dessus laquelle passe d. L’indice de la case vide d’arrivée est facile à calculer. Voir plus loin une explication sur l’intérêt de l’attribut incomplete.

Une presolution est l’enregistrement de :

* L’indice idxSD dans ED.dat de la situation SD pour laquelle on a trouvé une suite de mouvements qui mènent à une situation SF de EF.dat
* La liste des indices idxSI dans EI des situations initiales SI (ou de l’une de ses symétries SI’) à partir desquelles on peut rejoindre SD.
* La liste des mouvements pour passer de SD à SF.

Remarque : pour une situation SD donnée, il se peut que plusieurs situations SI non encore résolues lui soient associées. Il peut donc y avoir plusieurs lignes initial associées à une pré-solution.

Lors d’une reprise, les situations de EI sont recalculées, et comparées aux situations des éventuelles situations déjà trouvées (aux symétries près), décrites par les entrées solution et presolution afin de déterminer quelles situations n’ont pas encore trouvé de solution.

Remarque : on suppose ici que les SI sont classés dans un ordre identique lors de la recherche initiale et lors des reprises.

## Description des situations

Le plateau est inscrit dans un rectangle minimal implicite, appelé rectangle encadrant. Ce rectangle est divisé en carrés dans certains desquels se situent les cases du plateau. Ces carrés sont numérotés à partir de 0, de gauche à droite, à partir du haut. Le nombre de cases du rectangle encadrant est inférieur à 256, aussi on utilise le byte pour coder les indices de ses cases et réduire la taille mémoire et disque nécessaires.

Le plateau initial est décrit par

* La largeur de ce rectangle implicite.
* L’ensemble des indices des cases qui le composent.

Toute situation est décrite par

* L’ensemble des indices des cases occupées par une pierre

Voici les indices des cases des deux plateaux envisagés.

On y a fait figurer des indices de ligne et de colonne qui seront utilisées pour faciliter la mise au point du programme.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Les coordonnées et indices des cases du plateau   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |  |  |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | |  |  | -3 | -2 | -1 | 0 | 1 | 2 | 3 |  |  |  | -3 | -2 | -1 | 0 | 1 | 2 | 3 | | 0 | -3 | 0 | 1 | 2 | 3 | 4 | 5 | 6 |  | 0 | -3 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | | 1 | -2 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |  | 1 | -2 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | | 2 | -1 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |  | 2 | -1 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | | 3 | 0 | 21 | 22 | 23 | 24 | 25 | 26 | 27 |  | 3 | 0 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | | 4 | 1 | 28 | 29 | 30 | 31 | 32 | 33 | 34 |  | 4 | 1 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | | 5 | 2 | 35 | 36 | 37 | 38 | 39 | 40 | 41 |  | 5 | 2 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | | 6 | 3 | 42 | 43 | 44 | 45 | 46 | 47 | 48 |  | 6 | 3 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | |

# Initialisation

Une fois la description textuelle du plateau effectuée, sa largeur est déterminée, sa suite de cases aussi.

Le programme détermine les symétries du plateau. Il prépare ainsi les calculs à effectuer pour dresser la liste des situations obtenues par symétrie de l’une d’elles pour déterminer si elle a déjà été rencontrée.

Le programme détermine aussi la liste des mouvements à envisager. Un mouvement « envisageable » est une suite ordonnée de 3 cases du plateau contiguës, alignées. Leur sens de parcourt est important. Il est représenté par le triplet des indices des cases. Par exemple le triplet (2,3,4) sera utilisé pour effectuer un mouvement à partir d’une situation où les cases 2 et 3 sont occupées par une pierre, et la case 4 est vide. Le triplet (4,3,2) sera utilisé pour effectuer un mouvement à partir d’une situation où les cases 4 et 3 sont occupées par une pierre, et la case 2 est vide.

Ces données sont rapides à calculer, elles sont recalculées à chaque reprise.

Les ensembles EI et EG sont eux aussi très rapides à calculer, et recalculés à chaque reprise.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Les situations initiales, une seule pierre retirée, avec mise en évidence des symétries.   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | Situation 1 | | | | | | |  | Situation 2 | | | | | | |  | Situation 3 | | | | | | |  | Situation 4 | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | Situation 5 | | | | | | |  | Situation 6 | | | | | | |  | Situation 7 | | | | | | |  | Situation 8 | | | | | | | |

On peut remarquer que les situations 2 et 8 autorisent chacune un seul mouvement, aux symétries près pour la situation 8, et que ce mouvement aboutit à la même situation. Donc les situations 2 et 8 développent dès le premier mouvement exactement les mêmes situations dérivées.

Les situations gagnantes sont les duales de ces situations initiales où chaque pierre est remplacée par une case vide, et la case vide remplacée par une pierre. Sur le plateau Français il y en a donc aussi 8.

# Volume

Calcul grossier du nombre de situations différentes :

Sur un plateau de N cases, chaque case peut contenir ou non une pierre. Le nombre théorique de situations différentes est donc

Pour 33 cases, cela représente environ 9 milliards de cas.

Pour 37 cases, cela représente environ 140 milliards de cas.

On peut réduire ces cas d’un facteur 8 en considérant les symétries possibles.

Le nombre de cas dans les premières étapes est lui aussi réduit. Mais on peut penser que plus de mouvements sont effectués, et plus de situations potentielles sont atteignables.

# Les symétries

Le plateau est composé de cases données par leur indice dans un rectangle englobant dont on connait la largeur L et la hauteur H. Pour un indice de case i donné, on lui calcule des coordonnées cartésiennes x=i%L, y = I/L ;

Les coordonnées de ces cases sont situées dans deux intervalles identifiés : [xMin, xMax], [yMin, yMax].

Ici, on est assuré que xMin = yMin = 0, xMax = L-1, yMax = H-1.

Une « symétrie » est une rotation ou une symétrie axiale qui conserve globalement toutes les cases du plateau. On envisage les 3 rotations de 90°, 180°, 270°, dont le centre reste à déterminer, et les symétries axiales selon des axes parallèles aux axes du repère ou à ses deux diagonales principales.

L’objectif est de dresser pour chaque symétrie trouvée un tableau T de taille L\*H, tel que T[i] fournit l’indice de la case image de la case d’indice i par la symétrie.

Ainsi, lorsqu’on voudra les symétries d’une situation donnée, situation elle-même inscrite dans le rectangle englobant, et fournissant la liste de ses cases occupées, il sera aisé d’exploiter ces tableaux.

Pour que le plateau soit conservé, il est nécessaire, mais pas suffisant, que son rectangle englobant le soit. Cela est le cas pour la rotation 180° et les deux symétries principales. Cela n’est le cas pour les autres rotations et symétries que si le rectangle est un carré.

Le centre du rectangle englobant joue un rôle essentiel dans la recherche de symétries. Mais dans le cas général, ses coordonnées peuvent ne pas être entières. Soient

ses coordonnées. Si le rectangle est un carré , les coordonnées de son centre sont soit toutes deux entières, soit toutes deux demi-entières. Leur somme ou leur différence est donc entière.

Si

Rotation 90° autour du centre du carré englobant

Rotation 180° autour du centre du rectangle englobant

Rotation 270° autour du centre du carré englobant

Symétrie du rectangle selon un axe vertical

Symétrie du rectangle selon un axe horizontal

Symétrie du carré selon une parallèle à la 1ère diagonale

Symétrie du carré selon une parallèle à la 2ième diagonale

# Résolution

## Synopsis

Il y a de nombreuses étapes à franchir pour trouver les solutions.

1. Etablir la liste des situations initiales à partir du plateau fourni dans la fiche de jeu (à une symétrie près)
2. Etablir l’ensemble ED.dat des situations de départ SD différentes (à une symétrie près), obtenues par application d’un nombre fixe de mouvements ND sur les situations initiales. Chaque situation de départ est associée à la liste des situations initiales qui mènent à cette situation de départ (à une symétrie près). Cela facilite l’étape 6)
3. Etablir la liste des situations gagnantes SG (à une symétrie près), celles qui n’ont qu’une pierre sur le plateau.
4. Etablir l’ensemble EF.dat des situations finales SF différentes à une symétrie près, obtenues par application d’un nombre fixe de mouvements inverses NF sur les situations gagnantes SG. On ne fait pas l’effort d’associer la liste des situations gagnantes SG à chaque situation finale SF.
5. Pour chaque situation de départ SD, on cherche un chemin vers une situation finale SF. En fait, on va ignorer celles des situations de départ SD dont les situations initiales SI associées ont toutes déjà trouvé (au moins) un début de solution. On enregistre comme presolution ces situation de départ SD, le chemin trouvé pour atteindre SF et la liste des situation initiales SI associées à la situation de départ qui n’avaient pas encore trouvé de début de solution.
6. On reprend chaque presolution, et on recherche un chemin inverse qui mène de sa SD aux situations initiales SI encore associées (à une symétrie près). Chacune devient une solution où incomplete="true". Ces solutions enregistrent un chemin de la situation initiale SI jusqu’à la situation finale SF, mais pas encore jusqu’à la situation gagnante SG.
7. On reprend chaque solution incomplete pour chercher maintenant un chemin complémentaire qui mène de la situation finale SF à une situation gagnante SG.

Il a fallu séparer les étapes 6 et 7 parce que l’étape 6 constitue des stocks de situations déjà rencontrées valables toutes presolutions confondues, alors qu’une solution incomplète, pour être résolue, doit être calculée séparément des autres.

## Fonctionnel

Choix d’une [fiche de jeu](#_Fiche_de_jeu) « chemin/nom.xml » qui décrit le plateau initial.

### Options proposées

* Initialiser
  + Le répertoire chemin/nom n’existe pas ou une incohérence a été détectée. Quand cette option est proposée, les autres sont indisponibles
* Consolider les solutions
  + Des presolution sont présentes dans le fichier pilote.
* Rechercher
  + Toutes les SI n’ont pas de presolution ou de solution
* Régler ND (selon la saisie faite)
  + Toutes les SI n’ont pas de presolution ou de solution
* Régler NF (selon la saisie faite)
  + Toutes les SI n’ont pas de presolution ou de solution
* Suspendre
  + Si une opération (asynchrone) est en cours, ou en attente d’arrêt, c’est la seule option proposée, sinon elle n’est pas proposée. La choisir permet de demander à l’opération en cours de s’arrêter.

### Incohérences

* Absence du fichier pilote, ou mauvais format de celui-ci, ou plateau différent de celui de la fiche de jeu, ou données manquantes dans le fichier (paramètres ou index de reprise)
  + Suppression de tous les fichiers du répertoire, réinitialisation à partir de la fiche de jeu
* Sinon, fichier ED.dat ou EF.dat ou les deux manquant ou de mauvaise taille
  + Suppression des fichiers ED.dat, EF.dat, des pré-solutions du fichier pilote, ajustement des paramètres nd=nf=1, reprise/@idx=0, reconstruction de ED.dat, EF.dat. Les éventuelles solutions déjà découvertes sont conservées.

RQ : les vérifications de cohérence se font au moment du drop down du menu, mais pas sur le contenu des fichiers ED.dat, EF.dat (ce serait trop long). Une incohérence peut être découverte au moment de leur chargement. A ce moment, le fait est signalé, l’opération avortée, le fichier en cause est supprimé. Ainsi au prochain drop down du menu, seule l’option Initialisation est proposée.

### Initialisation

Si la situation décrite dans le fichier fourni n’a aucune case inoccupée, alors l’outil recherche une solution pour chaque situation issue de ce plateau complet, duquel il retire une pierre (aux symétries près). Sinon il recherche une solution pour la situation fournie uniquement (EI contient alors cette unique situation).

La valeur de reprise/@idx est initialisée à 0, et celle de parametres/@nd et parametres/@nf à 1.

Si le plateau initial est complet et qu’il a NC cases (et donc NC pierres), alors ED.dat contient les situations accessibles après ND mouvements appliqués à des situations initiales ayant NP = NC – 1 pierres : le nombre de pierres dans les situations de ED.dat est NP – ND .

Si le plateau initial est incomplet, qu’il a NC cases et NP pierres ( NP < NC ), alors ED.dat contient les situations accessibles après ND mouvements appliqués à cette situation initiale ayant NP pierres : le nombre de pierres dans les situations de ED.dat est NP – ND .

Le fichier EF.dat contient les situations obtenues après NF mouvements inverses sur les situations à 1 pierre, ces situations ont donc NF + 1 pierres.

Ces deux fichiers contiennent des tableaux de NC octets qui donnent les indices des pierres des situations calculées initialement.

De plus, dans le fichier ED.dat, derrière chaque description de situation dans ED.dat on ajoute un tableau de EI.Count octets correspondant au nombre de situations de EI. Chacune des cases contient la valeur 0 qui indique que la situation initiale ne permet pas d’atteindre la situation de ED, ou une de ses situations symétriques. La valeur 0xFF indique que cette situation de départ est atteignable depuis la situation de EI correspondante.

Les valeurs des indices de pierre et ceux des associations avec les SI sont contrôlées au chargement. Chaque indice de pierre doit désigner une case du plateau, et les valeurs d’association doivent être 0 ou 0xFF.

### Incrémentation de ND

L’ensemble des situations initiales EI est calculé à partir de la description du plateau. C’est une liste indexée.

Chaque situation SD de ED.dat est lue successivement avec son tableau d’association avec les SI de EI. Lorsque ND=1, on considère chaque situation SI de EI, et son tableau d’association est rempli de 0.

Pour SD donné, on recherche les nouvelles situations SD+1 obtenues par application d’un mouvement.

Si cette situation SD+1 n’a pas encore été rencontrée, son tableau d’association est le même que celui de SD. Sinon on ajoute à son tableau les indicateurs de SD à 0xFF.

Les situations SD+1 sont ensuite enregistrées dans un fichier ED.tmp

Puis ED.dat est supprimé et ED.tmp renommé en ED.dat

Pour pouvoir mener l’opération, il faut conserver en mémoire les nouvelles situations SD+1 jusqu’au traitement de toutes les situations de SD, car on ne peut prévoir quand le tableau d’association risque d’être mis à jour, et n’enregistrer ces données dans un fichier ED.tmp qu’après ce traitement complet.

On met à jour le paramètre nd dans le fichier pilote, on remet l’index de reprise à 0, on en supprime les pré-solutions, et on sauvegarde le fichier pilote.

### Incrémentation de NF

EG est calculé à partir de la description du plateau.

Chaque situation SF de EF.dat est lue successivement. Lorsque NF=1, on considère chaque situation de EG.

Pour SF donné, on recherche les nouvelles situations SF+1 obtenues par application d’un mouvement inverse.

Si cette situation SF+1 n’a pas encore été rencontrée, on l’enregistre dans un fichier EF.tmp

A la fin du traitement, on met à jour le paramètre nf dans le fichier pilote, on remet l’index de reprise à 0, on en supprime les pré-solutions, et on sauvegarde le fichier pilote.

En cas de problème de sauvegarde de EF.tmp, le supprimer.

### Consolider une pré-solution

On charge depuis ED.dat la SD concernée, et on reconstruit par mouvements inverses toute l’arborescence des situations menant à SD depuis une situation initiale SI. On va y découvrir toutes les SI correspondantes, dont celles désignées par la ou les lignes initial associées à cette SD (à une symétrie près). Cette recherche nous donne une liste de mouvements inverses joignant SD à SI (ou à une symétrie SI’ de SI). On en déduit la suite de mouvements menant de SI’ à SD.

On convertit la pré-solution en une solution « incomplète » pour chaque SI’ trouvée qui n’a pas encore de solution (à une symétrie près).

Dans la conversion de la pré-solution en solution, le plateau de la situation initiale de la solution sera celui de la situation SI’ trouvée par mouvements inverses, pas celui SI de la situation initiale de EI désignée par idxSI.

Depuis SD, on reconstruit SF grâce aux mouvements enregistrés dans la pré-solution. Puis on cherche depuis EF une situation gagnante. Cela nous fournit une suite de mouvements qui résout SF.

En joignant les trois suites de mouvements, on établit les mouvements permettant de résoudre SI’.

### Arranger ED.dat

Situation envisagée : Nous avons calculé les SD pour une valeur donnée de ND, ces situations sont maintenant dans ED.dat.

Comme nous l’avons vu, chaque SD est associée à une liste de situations initiales (non vide).

Certaines SD peuvent être obtenues à partir de plusieurs SI. Si on résout cette SD, du coup on résout chaque SI associée. Ces SI auront des mouvements initiaux différents, mais ensuite leurs mouvements depuis SD jusqu’à la solution gagnante sont les mêmes.

Voir [Statistiques](#_Statistiques) pour consulter le tableau récapitulatif des combinaisons de SD rencontrées.

Il peut être intéressant de tenter de résoudre ces SD en priorité.

On peut réorganiser les situations de ED.dat pour placer en début de fichier, donc en début de traitement, les SD qui pourraient aboutir à résoudre le plus de situations.

De plus, à un moment de la recherche, on a augmenté l’indice de reprise, et éventuellement trouvé une solution à certaines SI (à une symétrie près). On peut aussi réorganiser ED.dat en supprimant les SD déjà traitées, celles qui ne le sont pas mais adressent uniquement des SI déjà résolues par ailleurs, et en réorganisant les SD restantes selon leur nombre de SI non résolues.

Algorithme :

Lire le pilote, idxReprise, lister les SI présentes dans les solutions (à une symétrie près)

Lire chaque entrée de ED.dat à partir de idxReprise

Pour chacune des SD, dresser à partir de sa liste de SI la liste de ses SI non résolues.

Ignorer les SD dont toutes les SI sont résolues.

Classer les SD par liste distincte de SI non résolues.

Classer cette liste distincte par nombre de SI décroissant.

Pour chaque liste classée, enregistrer chacune de ses SD dans un nouveau fichier

Remplacer l’ancien ED.dat par ce nouveau fichier.

### Recherche

Le programme sélectionne successivement chaque description de situation de départ stockée dans ED.dat (en sautant, lors d’une reprise, celles déjà testées) pour tenter de lui trouver une situation finale de EF.dat (à une symétrie près).

Si pour une situation SD de ED.dat donnée, il trouve une situation SF de EF.dat, il considère chaque SI associé à SD dans ED.dat et pour ceux qui n’ont pas encore de solution ou de pré-solution (à une symétrie près), il enregistre la pré-solution correspondante.

## Optimisations

Pour éviter de systématiquement allouer et désallouer (ou tout au moins demander au Garbage Collector de désallouer) des quantités d’objets, on alloue des objets tampon qu’on utilise pour les calculs.

Un premier objet, le Plateau, contient

* un bool[] de la taille du rectangle englobant permettant de savoir si une case du rectangle est ou non une case du plateau.
* la liste byte[] des indices des cases du plateau, de taille NP du nombre de cases du plateau.

Il contient aussi des informations géométriques

* Taille du rectangle englobant
* Informations sur les symétries du tableau
  + Le plateau Français offre 8 symétries
* Informations sur tous les mouvements potentiels (à condition que les pierres soient disposées selon les règles) : triplets d’indices (Pierre déplacée, Pierre enjambée, Case vide de réception)
  + Pour le plateau français, il y a 92 mouvements potentiels

Pour une situation donnée, décrite par la liste byte[] des indices des pierres de la situation, de taille P donnée entre 1 et NP – 1 , on va devoir

* Déduire chaque situation obtenue par application, quand il est possible, d’un des mouvements potentiels. Ces nouvelles situations ont P-1 pierres.
* Déduire de chaque nouvelle situation les images par les symétries
* Contrôler si ces symétries ont déjà été rencontrées ou non.

Donc

* On enregistre dans les fichiers ED.dat et EF.dat uniquement les suites d’indices (au format byte) des pierres de la situation
* On stocke dans des Hashset des objets Situation qui se résument à ces octets
* On manipule une situation dans un objet SituationEtude alloué initialement.

Situation et SituationEtude exposent tous deux des méthodes GetHashCode() et Equals(), qui donnent le même hashcode et rendent vrai quand ils sont comparés l’un à l’autre et contiennent les mêmes pierres, même si les descriptions de la liste de ces pierres diffèrent. Ainsi, une nouvelle situation établie dans SituationEtude (et ses symétries) peut être testée dans le Hashset des objets Situation déjà rencontrés, mais si la situation est nouvelle, alors l’objet de type SituationEtude produit un nouvel objet Situation qui sera ajouté à ce Hashset.

Pour pouvoir réaliser ceci en conformité avec les caractéristiques du C# et de sa collection Hashset, on a fait dériver Situation et SituationEtude d’une même classe abstraite SituationBase. Et les Hashset contiennent des objets (dérivés de) SituationBase.

L’objet de type SituationEtude contient deux tableaux, chacun de la taille du rectangle englobant :

* bool[] Pierres
* bool[] ImagePierres

Le premier, Pierres, traduit la liste des pierres de la situation qu’on souhaite étudier.

On le manipule pour effectuer un mouvement, et comme source pour calculer ses images par symétrie.

On restaure les pierres en effectuant le mouvement inverse une fois que le mouvement en cours a été étudié.

Le second, ImagePierres, est l’image du premier par l’une des symétries du plateau appliquée au second. C’est celui-ci qui est utilisé dans GetHashCode() et Equals().

Pour ajouter au stock une nouvelle situation, on pourrait aussi bien utiliser Pierres que la dernière ImagePierres calculée. On a choisi Pierres.

Le plateau est constitué de cases contenues dans un rectangle englobant. Ce tableau englobant est tel que les coordonnées minimales des cases de ce plateau sont 0, et les coordonnées maximales sont égales à la dimension – 1 du rectangle. Autrement dit, le rectangle englobant est ajusté aux cases du plateau. Une fois établies les dimensions (Largeur, Hauteur), on associe à chaque case du rectangle (et donc du plateau) un indice (de type byte) de 0 à Largeur\*Hauteur – 1. Le plateau est alors la suite (ordonnée) des indices de ses cases.

Chaque symétrie est décrite par un tableau byte[] d’indirections. Ce tableau a la taille du rectangle englobant mais seules les cases correspondant à une case du plateau sont renseignées, avec l’indice de la case du plateau obtenue par symétrie.

La situation étudiée est décrite par un tableau bool[] de la taille du rectangle englobant.

Soit le plateau décrit par byte[] Plateau, la symétrie décrite par byte[] Symetrie, une situation décrite par son tableau bool[] Pierres. Pour chaque indice idx tel que Pierres[idx] soit vrai, idx représente une case du plateau. Alors Symetrie[idx] est l’indice de la case du plateau obtenue par la symétrie correspondante. Il nous reste à renseigner ImagePierres [Symetrie[idx]] = true pour constituer l’image de la situation décrite par bool[] Pierres.

En fait, comme on effectue une recherche récursive en profondeur, on gère un tableau de SituationEtude, chaque élément étant dédié à l’étude des situations suivant leur nombre de pierres.

## Statistiques

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Plateau Français | | | | | |
| EI | 8 |  |  |  |
| ND | 11 |  | NF | 13 |
| SD |  |  | SF | 75 432 581 |
| ED.dat | 250 484 460 |  | EF.dat | 1 056 056 134 |

Pour le plateau français, avec ND=11, j’ai fait une statistique du nombre de cas regroupés par SI associées.

En vert les situations initiales pour lesquelles on a trouvé une solution.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Situations initiales | | | | | | | | Nombre de SD | Résolu |  |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |  |  |  |
|  |  |  |  |  |  | x |  | 394460 | x |  |
|  |  |  |  |  | x |  |  | 1018299 |  |  |
|  |  |  |  | x |  |  |  | 813315 | x |  |
|  |  |  |  | x |  | x |  | 612183 | x |  |
|  |  |  | x |  | x |  |  | 2173604 |  |  |
|  |  | x | x |  |  |  |  | 845205 |  |  |
|  |  | x | x |  | x |  |  | 1501252 |  |  |
|  | x |  |  |  |  |  | x | 984710 |  |  |
| x |  |  |  |  |  |  |  | 918235 | x |  |
| x |  |  |  |  |  | x |  | 1382980 | x |  |
| x |  |  |  | x |  |  |  | 333364 | x |  |
| x |  |  |  | x |  | x |  | 1546616 | x |  |
| **Total** | | | | | | | | **12524223** | 6001153 | 47,92% |

J’ai arrangé ED.dat après un certain temps de calcul (purge des situations déjà testées, tri en fonction du nombre de SI décroissant qu’elles adressent).

Pertinence des hashcodes.

J’ai fait une statistique sur les hashcodes des situations finales lorsque nf=13 (donc sur les situations à 14 pierres qui mènent à une situation gagnante) :

Il y a 75 432 581 telles situations, 74 329 954 hashcodes différents

Nombre de hashcodes en fonction du nombre de situations que chacun représente :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| nb situations par hashcode | 1 | 2 | 3 | 4 |
| nb hashcodes ayant ce nb de situations | 73 239 255 | 1 078 868 | 11 734 | 97 |

Le calcul du HashCode n’est pas « parfait », mais pas trop mauvais.

# Compacité

On part de l’hypothèse qu’il existe une solution et on voudrait orienter la recherche sur les cas qui semblent les plus prometteurs pour limiter les temps de recherche.

Comment améliorer la recherche ?

Il pourrait sembler que lorsqu’une situation présente des trous énormes entre pierres, il peut paraitre difficile d’aller chercher les pierres éloignées pour les éliminer.

Par exemple, la situation suivante (à 15 pierres) n’a pas de solution.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | |  | -3 | -2 | -1 | 0 | 1 | 2 | 3 | | -3 |  |  |  |  |  |  |  | | -2 |  | x |  |  |  |  |  | | -1 |  |  | x |  | x |  |  | | 0 |  |  |  |  |  |  |  | | 1 | x | x | x | x |  |  | x | | 2 |  | x | x | x | x | x |  | | 3 |  |  | x |  | x |  |  | |

N : Nombre de points, indicés de 0 à N-1 ;

(xi,yi) : coordonnées du point d’indice i ;

G : centre de gravité des points, G a pour coordonnées (xG, yG)

Vi : variance du point d’indice i par rapport à G

En appliquant ce calcul à l’ensemble des SF de EF.dat pour Nf=13, donc pour des situations à 14 pierres, on obtient l’ensemble des variances des situations finales, dont on peut calculer la moyenne et la variance : environ 9,30

Et si on calcule la variance de la situation perdante ci-dessus (qui possède 15 pierres, on obtient 4,60

Cette démarche n’est pas pertinente. Parmi toutes les 75 432 581 situations finales (et donc pouvant aboutir à une situation gagnante), la suivante a une variance de 7,92 :

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | |  | -3 | -2 | -1 | 0 | 1 | 2 | 3 | | -3 |  |  | x | x | x |  |  | | -2 |  |  | x |  |  |  |  | | -1 |  |  |  |  |  |  |  | | 0 | x |  |  |  |  | x | x | | 1 | x | x |  |  |  | x | x | | 2 |  | x |  |  |  | x |  | | 3 |  |  |  | x |  |  |  | |

Et pourtant elle a une solution …

La moyenne des variances des SF s’établit à 1,72. Sachant que toutes ces variances sont positives, ça semble vouloir dire que ces situations sont en général très compactes. Mais la variance des variances atteint 9,30.

# Occupation mémoire

Le stock de situations déjà rencontrées occupe rapidement la mémoire vive.

Dans les premières versions du programme, nous avons eu des OutOfMemory exceptions. Ces exceptions sont déclenchées lorsqu’un objet requiert trop de mémoire.

Pour augmenter la taille des objets, plusieurs réglages :

* Compiler pour Any CPU ou 64 bits,
* Exécuter sur une machine 64 bits
* Dans App.Config de l’application (l’exécutable), placer les lignes suivantes :

<configuration>

\*\*\*

<runtime>

<gcAllowVeryLargeObjects enabled="true" />

</runtime>

</configuration>

* Décocher l’option « Préférer 32 bits »

Attention : vérifier ces réglages en mode Debug comme en mode Release.

Tout cela permet effectivement d’augmenter la quantité de ressources mémoire accaparée par le programme, mais un autre problème survient alors : le swap des pages mémoire.

J’ai alors trouvé sur Internet des instructions permettant d’être averti lorsque le moment où la mémoire RAM commence à manquer. Une classe SurveillanceMemoire met en œuvre ce mécanisme, et lorsque le moment arrive, le programme libère de la mémoire en vidant les slots du Stock concernant les situations les plus proches de EF.

# EF.dat sur disque

## Piste d’amélioration

Le chargement des 75 432 581 situations finales à 14 pierres prend beaucoup de place en mémoire. La quantité restante est donc limitée pour stocker les situations intermédiaires rencontrées, et les besoins de libération mémoire fréquents.

On peut envisager de ne pas le charger en mémoire, mais de consulter l’information laissée sur disque.

Le besoin est de tester si une situation est présente ou non dans EF.dat.

Evidemment, il n’est pas envisageable de consulter les entrées de EF.dat les unes après les autres pour s’en assurer.

## Recherche dichotomique dans le fichier

On peut comparer des situations de Np pierres en comparant octet par octet leurs deux tableaux triés, à la manière d’une comparaison lexicale, pour les ordonner.

Pour une situation donnée, dans les tableaux Classiques et Français, il y a 8 situations équivalentes par symétrie. Certaines peuvent être strictement identiques. En tout état de cause il existe (au moins) une situation la « plus petite » au sens de la comparaison vue ci-dessus.

On peut prévoir de ne stocker que ces situations minimales entre toutes les situations équivalentes par symétrie. Lorsqu’on voudra tester si une situation est dans l’ensemble, il suffira de considérer la situation équivalente minimale. Ainsi on limite le nombre de recherches.

Ensuite, on classe les entrées de EF.dat entre elles et on recherche par dichotomie dans le fichier toute situation ayant atteint le nombre de pierres correspondant à une SF.

Le fichier EF.dat a été trié en chargeant ses anciennes entrées dans un SortedSet équipé de ce comparateur.

Statistiques :

nf==13, nbPierres==14, nombre de SF : 75 432 581, taille de EF.dat : 1 056 056 134 octets

Chargement « naïf » dans le SortedSet en moins de 3 minutes.

Chargement avec choix parmi les situations symétriques en moins de 8 minutes.

Sauvegarde sur disque des situations triées en moins de 12.30 secondes

Evidemment lors de la recherche, la mémoire vive est moins encombrée, l’initialisation extrêmement courte, mais hélas le temps de scrutation des candidats au statut de SF est très augmenté.

Evaluation du nombre de niveaux a parcourir lors d’une recherche

Le nombre de nœuds est 75 432 581

A chaque test (négatif), on retient une des deux moitiés de part et d’autre de la situation centrale.

Le calcul nous amène à 26 niveaux maximum pour tester une situation qui n’est pas finale.

n niveau de recherche

q Nombre maximal de situations dans la moitié restante

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| n | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| q | 75 432 581 | 37716290 | 18858145 | 9429072 | 4714536 | 2357268 | 1178634 | 589317 | 294658 | 147329 |
|  |  |  |  |  |  |  |  |  |  |  |
| n | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| q | 73664 | 36832 | 18416 | 9208 | 4604 | 2302 | 1151 | 575 | 287 | 143 |
|  |  |  |  |  |  |  |  |  |  |  |
| n | 20 | 21 | 22 | 23 | 24 | 25 | 26 |  |  |  |
| q | 71 | 35 | 17 | 8 | 4 | 2 | 1 |  |  |  |

On peut envisager de charger en mémoire les premiers niveaux de ces situations charnières.

s Nombre de situations charnières dans un niveau

S Nombre cumulé avec les niveaux précédents

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| n | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| s | 0 | 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 |
| S | 0 | 1 | 3 | 7 | 15 | 31 | 63 | 127 | 255 | 511 |
|  |  |  |  |  |  |  |  |  |  |  |
| n | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| s | 512 | 1024 | 2048 | 4096 | 8192 | 16384 | 32768 | 65536 | 131072 | 262144 |
| S | 1 023 | 2 047 | 4 095 | 8 191 | 16 383 | 32 767 | 65 535 | 131 071 | 262 143 | 524 287 |
|  |  |  |  |  |  |  |  |  |  |  |
| n | 20 | 21 | 22 | 23 | 24 | 25 | 26 |  |  |  |
| s | 524288 | 1048576 | 2097152 | 4194304 | 8388608 | 16777216 | 33554432 |  |  |  |
| S | 1 048 575 | 2 097 151 | 4 194 303 | 8 388 607 | 16 777 215 | 33 554 431 | 67 108 863 |  |  |  |

Si on charge les 65 535 situations charnières à consulter pour arriver au niveau 16, on aura alors 16 situations à comparer sans accéder au fichier EF.dat, puis ensuite 10 accès au fichier.

## Recherche par BTree

Pour limiter les accès disque (mais en chargeant à chaque lecture plus de données), j’envisage alors de stocker dans le fichier EF.dat une structure de type B-Tree.

Un premier nœud, dit root ou racine, contient *m+1* index de type uint32 et *m* tableaux de *nf+1* octets.

Les tableaux sont les représentations des SF. Ils sont ordonnés.

Les index désignent le début d’autres nœuds de même structure, dits enfants du nœud racine qui est alors leur parent. Les SF de ces fils sont ordonnés, bien sûr, mais respectent aussi l’ordre suivant :

L’enfant d’indice 0 a des SF toutes inférieures à celles de son parent.

L’enfant d’indice *m* a des SF toutes supérieures à celles de son parent.

L’enfant d’indice *j*, *0<j<m*, a des SF comprises entre celles d’indices *j-1* et *j* du parent.

Comment choisir *m* ? On lit un SSD, alors il n’y a à ma connaissance pas trop de contraintes d’optimisation. Le nœud a une taille de *4\*(m+1) + nf\*m* . On va utiliser la dichotomie pour rechercher dans les *m* situations du nœud la position de la situation testée. Donc on charge un nœud (m1 millisecondes) et on cherche parmi les *m* situations (m2 millisecondes). L’idée est de choisir *m* pour minimiser m1+m2.

Petit exercice : dans une liste de C cellules, une recherche dichotomique de la valeur X teste X avec la cellule du milieu M pour connaitre la position de X par rapport à M. En cas d’inégalité, la recherche reprend avec la partie à gauche de M ou la partie à droite de M. Pour que ces deux parties soient de taille égale, il faut que C soit impair. Et en généralisant, il faut aussi que chaque partie soit impaire. En cas de recherche totalement infructueuse, on aboutit à une partie à un seul élément, évidemment impaire. Donc C devrait avoir l’expression suivante : C = 2p – 1 ,où *p* est la profondeur de recherche.

Contrôle : si 2n – 1 est le nombre de situations (indicées de 0 à 2n – 2) auprès desquelles tester la situation X candidate, alors après comparaison infructueuse de X avec la situation centrale (d’indice (2n – 2 + 0)/2 = 2n-1 – 1), la moitié des situations restantes devient : (2n – 1 – 1) / 2 = 2n-1 – 1.

Soit p la profondeur maximale de recherche dichotomique souhaitée.

Soit n le nombre de situations contenues dans la moitié retenue après une comparaison infructueuse avec la situation moyenne.

|  |  |
| --- | --- |
| Profondeur atteinte | Nombre de SF |
| 1 | 2p – 1 |
| 2 | 2p-1 – 1 |
| 3 | 2p-2 – 1 |
| … |  |
| p | 21 – 1 = 1 |

Si on prévoit une profondeur de recherche dichotomique p=4, ce qui représente 4 tests au maximum, le nombre optimal de situations est 15.

Soit p la profondeur maximale de recherche dichotomique

Soit m le nombre optimal de situations dans un nœud

Soit t la taille du nœud en octets

Avec nf=13, donc Nombre de Pierres = 14, on a les statistiques suivantes :

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **p** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| **m** | 1 | 3 | 7 | 15 | 31 | 63 | 127 | 255 | 511 | 1023 | 2047 | 4095 | 8191 | 16383 | 32767 |
| **t** | 22 | 58 | 130 | 274 | 562 | 1138 | 2290 | 4594 | 9202 | 18418 | 36850 | 73714 | 147442 | 294898 | 589810 |

Si on veut limiter la profondeur de l’arbre, on augmente cette profondeur de recherche dichotomique, ainsi que la quantité d’octets lus sur le disque pour charger un nœud.

On peut aussi conserver en mémoire les nœuds de premier niveau chargés pour là encore limiter les accès au disque.

Rappel :

Entiers sur 32 bits.

|  |  |  |  |
| --- | --- | --- | --- |
| int | -2 147 483 648 to 2 147 483 647 | –231 to 231–1 | Signed 32-bit integer |
| uint | 0 to 4 294 967 295 | 0 to 232–1 | Unsigned 32-bit integer |

On remarque que ce B-Tree n’a pas besoin d’être construit au fur et à mesure de la découverte des situations. On peut découvrir toutes les situations en mémoire, puis les trier, puis construire les nœuds du B-Tree, toujours en mémoire, pour finalement enregistrer toute la structure sur disque dans le fichier EF.dat . Le seul risque est de saturer la mémoire lors de cette opération.

Rappels : structure en B-Tree

Un paramètre essentiel d’un B-Tree est un entier *m>0* appelé ordre du B-Tree.

Un B-Tree est un arbre (une seule racine) constitué de nœuds et de liens orientés, sans boucles, chaque nœud sauf le nœud dit racine ayant un seul antécédent (dit père).

Chaque nœud contient au plus *m* situations. S’il en contient *m’, 1 ≤ m’ ≤ m*, alors il contient *0* ou *m’+1* liens vers ses fils.

Les *m’* situations d’un nœud sont ordonnées.

La plus grande situation du fils de position *0* est < à la plus petite situation du père.

La plus petite situation du fils de position *m’* est > à la plus grande situation du père.

Les situations du fils de position *j, 0 < j < m’*, sont comprises entre la situation du père de position *j-1* et celle de position *j*.

Un nœud sur le disque est donc constitué d’un tableau d’octets.

Pour un nœud contenant exactement *m* situations :

Les 4\*(m+1) premiers octets sont des entiers non signés sur 32 bits qui renseignent sur l’offset depuis la le début du nœud actuel pour accéder au début du nœud fils. Ou, si le nœud n’a pas de fils, ce tableau est entièrement constitué de 0 (il n’y a pas d’offset valant 0 séparant un nœud père d’un de ses fils).

On remarque, étant donné que les offsets sont des entiers non signés, qu’un nœud fils est toujours placé APRES son nœud père dans le fichier.

Les Np \* m octets suivants contiennent les indices des pierres des *m* situations du nœud.

Si le nœud contient *m’, 0<m’<m* situations, alors ces situations sont décrites dans les *m’* premières zones de données (et les éventuels fils dans les *m’+1* premières zones des indices), et les tableaux suivants sont tous constitués de 0. Un tableau de situation dont les deux premiers octets sont 0 est donc la marque que lui et les suivants ne sont pas des situations.

Tout nœud a une longueur fixe, même s’il contient moins de *m* situations : TN = *4\*(m+1)+*Np*\*m* octets. Les indices donnent le nombre de nœuds qui séparent un père de son fils. Il faut donc le multiplier par TN pour obtenir le décalage en nombre d’octets.

## Initialisation de EF.dat

Les situations finales de taille Nf sont toutes calculées et insérées dans un SortedSet.

Construction du premier nœud :

Le sorted set contient N éléments. Supposons que N ≥ *m+1*. Il s’agit de choisir *m* éléments qui vont constituer le contenu de ce nœud. Il seront le plus régulièrement répartis de manière à fractionner la liste ordonnée des situations du sorted set en *m+1* sous-ensembles qui constituerons le contenu des *m+1* sous-arbres des fils du nœud , tous séparés par une situation (il y en a donc *m*) qui elle ira dans le nœud. On cherche une valeur *d* telle que *d* est le cardinal des *f* premiers sous-ensembles et *d*-1 celui des suivants. On doit donc avoir N-*m* = *f*\**d* + (*m*+1-*f*)\*(*d*-1). C’est-à-dire N = *d*\*(*m*+1) + *f*-1.

La valeur *d* est le quotient de N par *m*+1.

Encore faut-il que *d*-1 > 0, car dans le cas contraire on aura des nœuds vides. C’est le cas pour N ≥ *m*+1.

La valeur *f*-1 est l’indice du dernier sous-ensemble de cardinal *d*. C’est le reste de la division de N par *m*+1.

Lorsque N ≤ *m*, alors toutes les situations vont dans le nœud qui n’a pas de fils.

Cette manière de faire déroge à une règle des B-Trees, à savoir qu’aucun nœud (sauf éventuellement la racine) ne doit contenir moins d’un certain nombre d’éléments, en général *m/2*.