Morpion solitaire

Projet Java avancé M1 2020/2021

1 Dates importantes

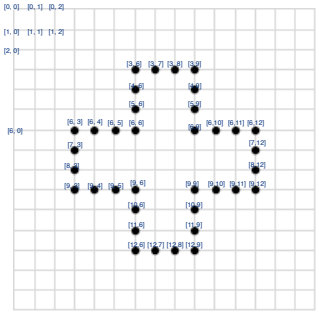
— Envoi des binômes et liens GIT par email avant le : 11 décembre 2020.
— Rendu du projet : au plus tôt le 10 janvier 2021, 23h.

2 Description

Le but du projet est de proposer une interface pour jouer au jeu du morpion solitaire et d'implémenter des algorithmes qui cherchent des solutions pour ce jeu.

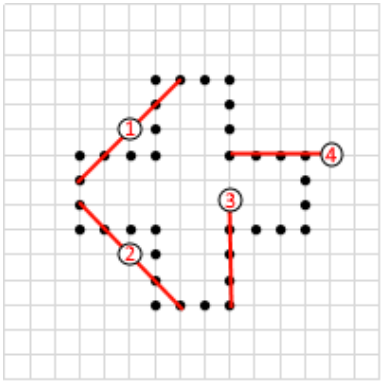
Il existe plusieurs plateaux et variantes, on va s'intéresser à 2 d'entre elles sur un même plateau.

Le plateau est représenté par la grille suivante :



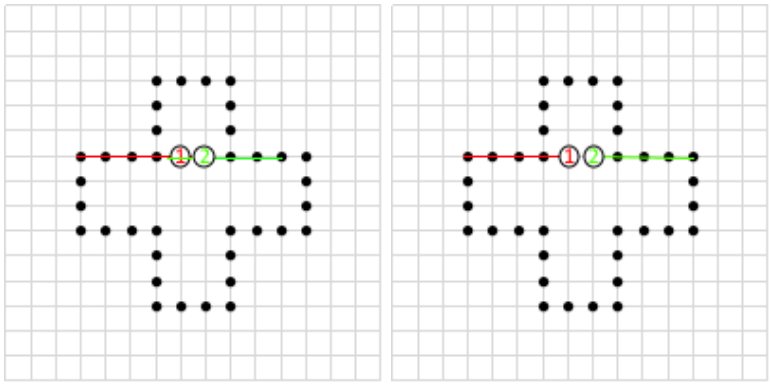
Le but du jeu est de jouer le plus de coup possible. Pour jouer un coup :
— il faut ajouter un nouveau point de façon à avoir 5 points alignés puis de tracer un trait sur ces 5 points.
— le trait ajouté peut ou ne peut pas recouvrir un trait existant dans la même direction selon la version (respectivement appelée 5T et 5D)

Voici un exemple représentant des coups dans les 4 directions existantes :

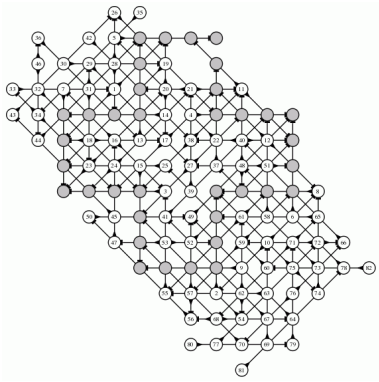


Il est important de représenter le point ajouté comme le numéro du coup joué. Sans cette indication, il est impossible de connaître l'ordre des coups de la partie et impossible de vérifier la validité de la grille.

Voici un exemple des versions 5T et 5D :



Voici un exemple de grille pour la version 5D (grille record) :



Il est attendu AU MINIMUM :

1. Un moteur de jeu qui permet à un utilisateur de jouer une partie en 5D ou en 5T.
2. Une méthode de recherche de solution automatique aléatoire.
3. Un interface graphique pour interagir et observer.

Plusieurs extensions sont possibles :

1. Pour la partie interactive : proposer un tableau des scores, afficher la ou les meilleurs grilles obtenues (sauvegarde de ces informations dans des fichiers)
2. Pour les méthodes de recherche : proposer l'affichage sous forme de graphique de la distribution des scores obtenues afin de comparer la performance de ces algorithmes.
3. Implémenter l'algorithme NMCS.
4. Proposer d'autres algorithmes de recherche de solutions.
5. Utiliser le multi-threading pour accélérer les algorithmes de recherche.
6. Proposer des plateaux de jeu différents (dénommée 5T# ou 5D#)

2.1 Par rapport à la méthode de programmation

La manière de coder est au moins aussi importante que le résultat final. Ainsi, la propreté du code, sa modularité, sa lisibilité et l'architecture choisie comptera largement dans la note finale. Vous devez faire des tests unitaires (pertinents) tout au long du projet avec JUnit. Votre code devra compiler avec aucune erreur ni warning.

Également, vous devez utiliser BitBucket ou GitHub pour votre projet. Ce sont des gestionnaires de versions utilisant GIT. Il sont gratuit en demandant un compte académique. Veillez à créer votre projet en **privé**. Vous devez m'ajouter à votre projet dès le début, avec mon email boris.doux@lamsade.dauphine.fr. Cela vous permettra de travailler facilement à plusieurs, à différent endroits. GIT se lie bien à Eclipse avec le plugin EGit. Veillez à bien commenter vos commit et push.

2.2 Méthodes de recherche

Vous allez devoir implémenter une méthode de recherche de solution aléatoire. De part sa nature, les résultats seront peu fiables mais avec un grand nombre de partie vous pourrez évaluer la qualité de l'approche et voir quels scores une telle méthode pour obtenir. C'est pourquoi vous pourrez proposer un graphique représentant la distribution des scores afin de comparer les différentes approches que vous aurez implémentées.

Pour implémenter l'algorithme NMCS, vous avez le pseudo-code suivant et pour plus de détail le lien suivant : NMCS

Algorithm 1 The NMCS algorithm.

```
NMCS (state, level)
if level == 0 then
  return playout (state, uniform)
end if
BestSequenceOfLevel ← ∅
while state is not terminal do
  for m in possible moves for state do
    s ← play (state, m)
    NMCS (s, level − 1)
    update BestSequenceOfLevel
  end for
  bestMove ← move of the BestSequenceOfLevel
  state ← play (state, bestMove)
end while
```

2.3 Morpion Solitaire

Vous pourrez trouver sur ce lien d'autres informations concernant le Morpion Solitaire, notamment pour les plateaux de jeu différents.

3 Conditions de rendu

Le projet est à effectuer en binôme (ou trinôme mais notation un peu plus sévère).

Votre projet est à rendre avant la date précisée plus haut sur mon mail. Un seul envoi par binôme ! **Il y aura un point en moins par heure de retard, une heure entamée est due.** Le format de rendu est une archive au format **ZIP** contenant :

- Un répertoire *src* avec les sources de votre implémentation java.
- Un répertoire *docs* contenant :
  - Une documentation pour l'utilisateur *user.pdf* décrivant à un utilisateur comment se servir de votre projet (entrées, sorties, utilisation, options...).
  - Une documentation pour le développeur *dev.pdf*, devant justifier les choix effectués, présenter l'architecture choisie, indiquer quelles ont été les difficultés rencontrées au cours du projet ainsi que la répartition du travail entre les membres du groupe. Ce rapport doit faire le point sur les fonctionnalités apportées, celles qui n'ont pas été faite (et expliquer pourquoi). Il ne doit pas paraphraser le code, mais doit rendre explicite ce que ne montre pas le code. Il doit montrer que le code produit a fait l'objet d'un travail réfléchi et minutieux (comment un bug a été résolu, comment la redondance dans le code a été évitée, comment telle difficulté technique a été contournée, quels ont été les choix, les pistes examinées ou abandonnées...). Ce rapport est le témoin de vos qualités scientifiques mais aussi littéraires (style, grammaire, orthographe, présentation).
  - Un rapport d'expérimentations étudiant les choix algorithmiques et leurs effets (telle optimisation, en quoi cette idée d'algorithme est mieux que la résolution aléatoire, telle utilisation des threads, etc).
  - Un répertoire vide *doc* pour la javadoc.

— Un fichier ant *build.xml* permettant de compiler, créer le jar, générer la javadoc, etc.
— Votre jar exécutable pouvant être lancés au moyen de la commande `java -jar` avec pour nom la concatenation du nom des membres du binome.

Votre projet doit pouvoir s'exécuter sans utiliser eclipse !

L'archive aura pour nom Nom1Nom2.zip, où Nom1 et Nom2 sont les noms des membres du binôme par ordre alphabétique. L'extraction de l'archive devra créer un dossier Nom1Nom2 contenant les élements précisés ci-dessus.

Il va sans dire que les différents points suivants doivent être pris en compte :

— Uniformité de la langue utilisée dans le code (anglais conseillé).
— Uniformité des conventions de nommage et de code (convention Java obligatoire).
— Gestion propre des différentes exceptions.
— Les sources doivent être commentées, dans une unique langue, de manière pertinente (pas de commentaire "fait un test" avant un if.).
— Le nom des variables, classes et méthodes doivent être choisis judicieusement.
— Le code devra être propre, les exceptions correctement gérées, les classes correctement organisées en packages. La visibilité des méthodes et champs doit être pertinente (privée ou non...).
— Le projet doit évidement être propre à chaque binôme. Un détecteur automatique de plagiat sera utilisé. Si du texte ou une portion de code a été empruntée (sur internet, chez un autre binome), il faudra l'indiquer dans le rapport. Tout manque de sincérité sera lourdement sanctionné.

La documentation (rapports, commentaires...) compte pour un quart de la note finale. On préférera un projet qui fonctionne bien avec peu de fonctionnalités qu'un projet bancal avec plus de fonctionnalités.