Université de Caen Normandie
Département Mathématiques & Informatique
L2 informatique, 2022
Unité UE2

Sécurité et aide à la décision

Jeu d'infection

Grégory Bonnet

---

1 Jeu d'infection

Soit une grille vide de 7×7 cases et deux joueurs, `Rouge` et `Bleu`. Chaque joueur débute la partie avec deux pions de sa couleur, respectivement en bas à gauche et haut à droite pour `Bleu` et dans les deux autres coins pour `Rouge` (voir figure 1). C'est le joueur `Bleu` qui joue en premier.

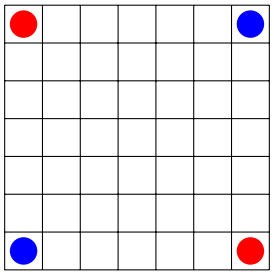

FIGURE 1 – Position initiale du jeu

Le gagnant est celui qui a **le plus de pions en fin de partie**. La partie se termine quand :

1. un des joueurs n'a plus de pion de sa couleur sur le plateau,
2. les deux joueurs doivent passer leur tour,
3. le plateau de jeu revient dans un état qui a déjà été joué.

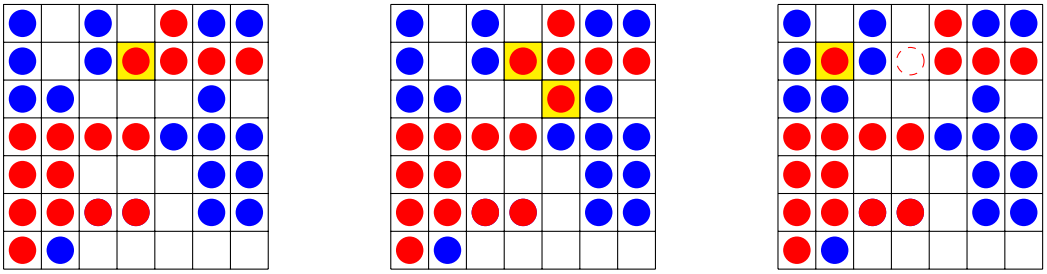

FIGURE 2 – Le pion rouge dans la case jaune (figure à gauche) peut entre autres être cloné en diagonale bas droite (figure du milieu) ou sauter à gauche (figure de droite)

Lorsque vient son tour de jouer, un joueur choisit un de pions de sa couleur et peut soit le `cloner`, soit le faire `sauter` (voir figure 2). Cloner un pion consiste à placer un nouveau pion sur une case libre à une distance de un (quelle que soit la direction) du pion choisi. Sauter consiste à déplacer le pion choisir une **case libre à une distance de deux** dans une des huit directions possibles, sachant que ce mouvement permet de passer par-dessus un autre pion quel qu'en soit le propriétaire. Si le joueur ne peut ni cloner un de ses pions, ni en faire sauter un alors il *doit* **passer son tour**.

À l'issue de ce coup, tous les pions adverses qui sont adjacents au pion déplacé par un saut, ou adjacent au pion nouvellement créé par clonage sont infectés et sont transformés en des pions de la couleur du joueur actif (voir figure 3). Après cela, c'est au joueur suivant de jouer.

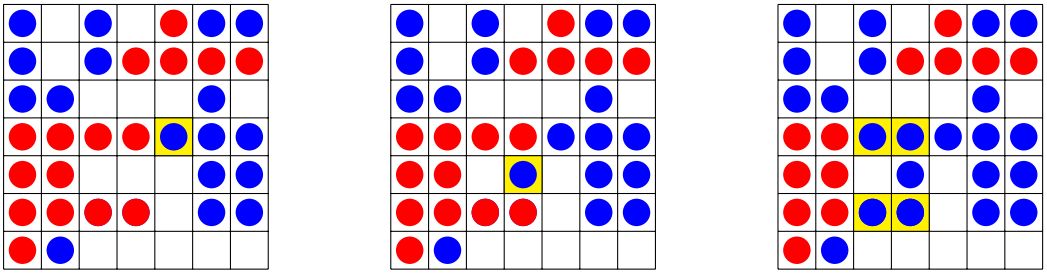

FIGURE 3 – Le pion bleu dans la case jaune (figure à gauche) se clone en diagonale bas gauche (figure du milieu) et infecte les pions rouges adjacents (figure de droite)

2 Travail à réaliser

2.1 Implémentation du jeu

Implémentez au choix en **Python** ou **Java** le jeu présenté ci-dessus.

Pour cela, créez une classe `State` représentant un état du jeu : grille, pions, joueur actif. Privilégiez une représentation simple : il n'est pas utile de définir des classes pour représenter les pions ou les joueurs. Implémentez dans la classe `State` une méthode `isOver` qui retourne un booléen indiquant si la partie est finie ou non, une méthode `getMove` qui prend un joueur en entrée et retour l'ensemble des coups possibles de ce joueur (il peut être nécessaire de définir une classe `Move` pour représenter une coup), une méthode `getScore` qui prend un joueur en entrée et retourne l'évaluation de l'état par ce joueur, et une méthode `play` qui prend un coup en entrée et retourne un **nouvel** état (ce dernier point est très important pour la récursion des algorithmes de recherche).

Pour la méthode `getScore`, considérez l'évaluation comme étant la proportion de pions que possède le joueur par rapport à son adversaire (il est peut être intéressant de conserver dans la classe `State` le nombre de pions de chaque joueur plutôt que de les recalculer à chaque fois). Ainsi, si N_B est le nombre de pions de `Bleu` et N_R le nombre de pions de `Rouge`, les évaluations de `Bleu` S_B et de `Rouge` S_R sont donc :

$$S_B = \frac{N_B}{N_B + N_R} \qquad S_R = \frac{N_R}{N_B + N_R}$$

Implémentez un programme principal permet de faire jouer aléatoirement les deux joueurs.

Algorithm 1 Exemple de programme principal pour un jeu aléatoire

```
game ← new State(initial)
while not game.isOver() do
  player ← game.getCurrentPlayer()
  move ← random(game.getMove(player))
  game ← game.play(move)
end while
```

2.2 Algorithmes de recherche

Écrivez une implémentation de (1) `minimax` ou `negamax` et (2) un élagage `alphabeta` (ou sa version `negamax`). Ces implémentation sont des classes dont les constructeurs doivent prendre en entrée le joueur qui raisonne et la profondeur de raisonnement.

L'algorithme `minimax` vu en cours ne calcule que la valeur de l'état passé en paramètre. Pour obtenir l'action à jouer, ajoutez une méthode `getBestMove` qui prend en entrée un état et retourne le meilleur coup à jouer.

Algorithm 2 Exemple de méthode `getBestMove`

```
Require: state, deepness
  bestaction ← ∅
  bestvalue ← -∞
  for all action ∈ state.getMove(player) do
    nextstate ← state.play(action)
    value ← minimax(nextstate, deepness)
    if value > bestvalue then
      bestvalue ← value
      bestaction ← action
    end if
  end for
  return bestaction
```

Attention, dans le cas de `alphabeta`, cette méthode doit être adaptée pour tenir compte des valeurs de α et β trouvées lors des évaluations précédentes dans la boucle.

2.3 Expérimentation

Votre programme principal doit pouvoir être exécuté **en ligne de commande** en lui passant trois paramètres :

1. un entier indiquant la profondeur de raisonnement de `Bleu`,

2. un entier indiquant la profondeur de raisonnement de `Rouge`,
3. un booléen indiquant l'utilisation ou non d'un élagage `alphabeta`.

Implémentez dans vos algorithmes de recherche un compteur pour obtenir le nombre de nœuds explorés au cours de la partie.

En considérant que la profondeur de raisonnement est la même pour les deux adversaires, tracez des courbes représentant le nombre total de nœuds explorés par `minimax` puis `alphabeta` au cours de la partie en fonction la profondeur de raisonnement du joueur `Bleu`. Faites le au moins pour les profondeurs `1 à 5`. Intégrez ces courbes dans un fichier PDF et commentez-les.

3 Rendu du travail et notation

Le TP devra être rendu pour le **mardi 1er mars 2022**, **9h00** au plus tard sur eCampus : `https://ecampus.unicaen.fr/mod/assign/view.php?id=442164`. Il s'agira d'une archive (ZIP, TAR, GZ au choix) contenant :

— votre code source (compilable ou exécutable sans les scories laissées par un IDE),
— un fichier `readme.txt` afin d'indiquer la marche à suivre,
— un court rapport au format PDF présentant les résultats d'expérimentation.

Vous pouvez former un **groupe de quatre étudiants** (ou moins, mais jamais plus). Vos noms doivent être indiqués dans le rapport et le `readme.txt`. La notation reposera sur (1) l'implémentation des règles du jeu, `minimax` et `alphabeta` ; (2) les expérimentations réalisées ; (3) la propreté et la lisibilité du code qui devra être commenté. Un petit bonus sera donné à ceux qui auront implémenté les versions `negamax` des algorithmes.