Master MIAGE 1

Design Patterns - Devoir

Le jeu d'Othello

L'objectif de l'exercice est de mettre en place une ébauche de solution informatique pour permettre à l'ordinateur de de jouer à l'Othello et surtout de décider quel coup jouer. Nous allons nous appuyer sur l'exercice précédent et voir comment la notion d'arbre peut s'appliquer ici.

Pour les lecteurs qui ne connaissent pas ce jeu quelques petits rappels:

- le jeu se joue sur un plateau de 64 cases;
- chaque joueur joue alternativement soit en posant une pierre de sa couleur sur une case vide du plateau à condition d'effectuer une prise; ou soit en passant son tour (parce qu'il ne peut pas prendre des pierres adverses)
- le principe de prise est simple : si la pose d'un pierre permet d'encadrer une ligne de une ou plusieurs pierre ennemies, celles-ci sont capturées et prennent la couleur de leur adversaire. La pose d'une pierre doit se faire nécessairement sur une case vide connexe à une case non vide
- le jeu se termine soit quand toutes les cases sont occupées; soit s'il ne reste plus qu'une seule couleur sur le plateau;
- Gagne celui qui a le plus de pierres de sa couleur sur le plateau.

Le déroulement d'une partie d'Othello est donc séquencé par les coups joués alternativement par les deux joueurs. Chaque coup donne une nouvelle configuration du plateau de jeu comme illustrée dans le schéma suivant

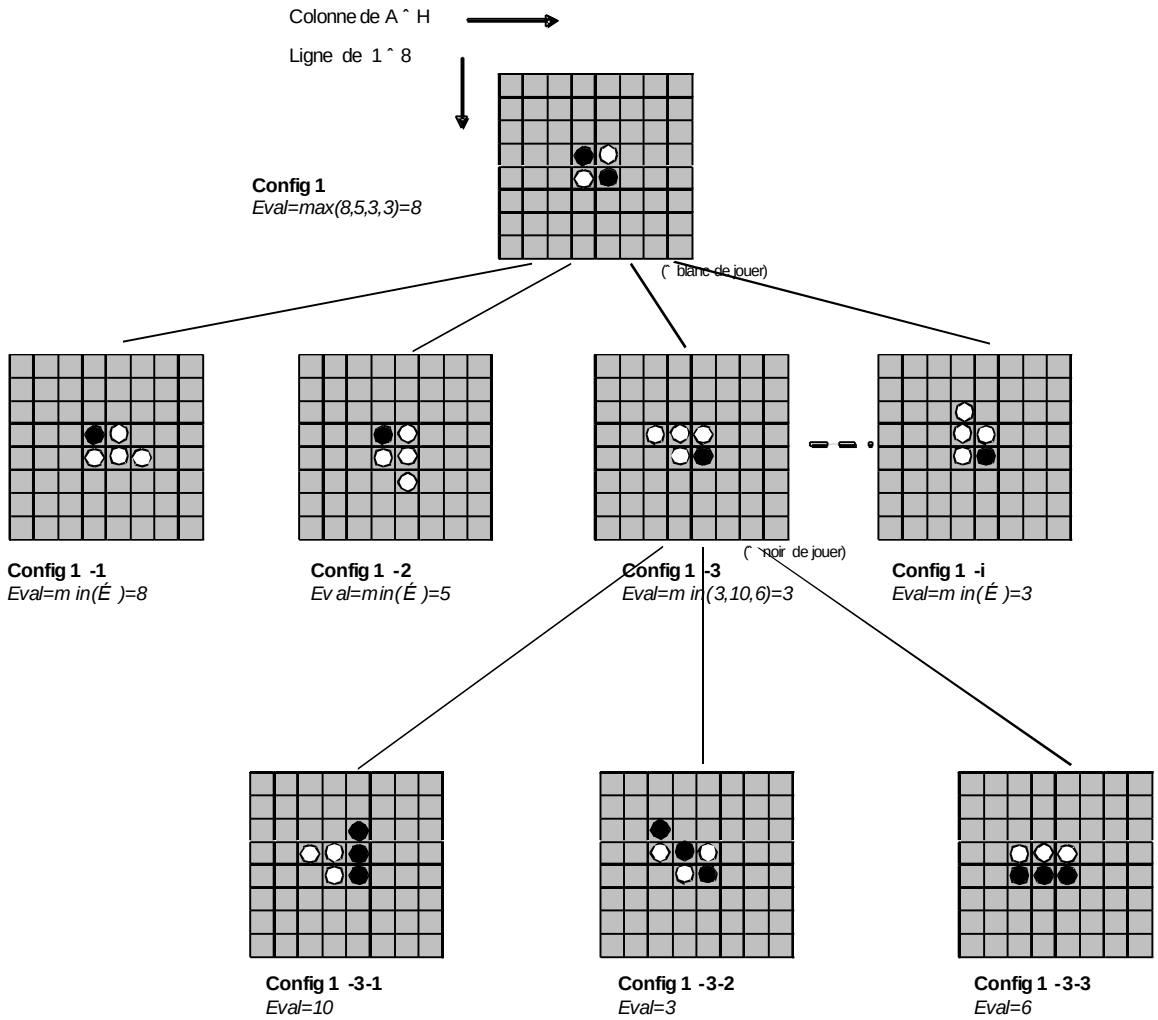

Figure 1 - Arbre de coup d'une partie d'Othello

Le schéma se lit comme suit : le plateau mis à la racine est la configuration initiale à partir de laquelle on raisonne[1] (donc l'ordinateur raisonnera) pour déterminer quel coup jouer, on se place du point de vue du joueur blanc. Si blanc joue C4 depuis cette configuration 1, il passe en configuration 1-3 et c'est à noir de jouer et on recommence ce raisonnement. Ceci permet de construire un arbre de coup dont chaque noeud est une configuration obtenue avec l'effet du coup joué. La construction de cet arbre lors du processus de raisonnement s'arrête au niveau de profondeur *fixé* arbitrairement: on ne raisonne pas a plus de X niveaux de profondeur[2]. Une branche peut toutefois s'arrêter plus tôt Si elle aboutit à la perte ou au gain de la partie (i.e. une des deux couleurs est complètement éliminée du plateau).

Question 1

Identifiez les objets présents dans le schéma de l'exemple et proposez le diagramme d'objet représentant cet arbre de coup[3]. Commentez ce diagramme pour montrer votre perception du problème. Quel design pattern vous proposez-vous d'utiliser ?

Question 2

Donnez le diagramme de classes correspondant.

Nous allons maintenant voir comment l'ordinateur pourrait «raisonner» pour identifier, parmi la liste des coups qui lui est possible de jouer à son tour lequel est le meilleur. Tout d'abord et parce que cela n'apporte rien à cet exercice, nous supposons avoir un algorithme d'évaluation d'une configuration fonctionnant comme suit: on donne l'état des 64 cases du plateau, le point de vue de l'évaluation (blanc ou noir) et l'on récupère une valeur indiquant l'évaluation de la position de jeu : 1 = très mauvais à 10 = très bon. Vous pouvez supposer que cet algorithme est une méthode prenant en paramètre l'état du plateau de jeu et une indication du point de vue (blanc ou noir) suivant lequel produire l'évaluation. La signature de cet algorithme appelé FEval est la suivante :

Entier FEval(Plateau~n Plateau, Couleur pointDeVue)

Voilà maintenant comment se déroule le « raisonnement».
Chaque configuration terminale (les feuilles de l'arbre) récupère son évaluation (pour la figure 2, cette évaluation est faite suivant le point de vue de blanc) via l'algorithme FEval.
Le principe ensuite pour remonter l'évaluation vers la racine où le choix du coup à jouer s'effectue est de considérer que chacun joue de son mieux. Donc noir jouera le coup qui l'avantage le plus et blanc de même. Chaque niveau de l'arbre correspond à une couleur : blanc joue puis noir puis blanc. L'évaluation des configurations situées à des noeuds non terminaux s'effectue donc autrement que par l'algorithme d'évaluation FEval. Quand c'est blanc qui joue, il choisit le coup ayant l'évaluation maximum (il joue de son mieux). Quand c'est à noir de jouer, on considère qu'il choisit le coup ayant la plus mauvaise évaluation pour blanc, on prend donc le minimum (donc la meilleure pour noir) et ainsi de suite en remontant l'arbre. C'est le principe de l'algorithme du MinMax[4].
Dans la suite, on ne s'intéresse pas à la génération de cet arbre de configuration mais à son évaluation.

Question 3

Chaque noeud de l'arbre des coups doit savoir s'évaluer. Eu utilisant le polymorphisme, positionnez dans votre diagramme de classe élaboré en question 2 et expliquez une ou plusieurs implémentations (donner le corps de la méthode ainsi que la ou les valeurs qu'elle retourne, si elle en retourne dans votre solution) de la méthode *evalue(Couleur unPoinfDeVue).* Cette méthode doit permettre de déterminer le coup à jouer à partir de la configuration courante en se plaçant du côté du joueur ayant la couleur unPointDeVue.

Question 4

De même positionner une méthode *joue(Couleur unPointDeVue)* lancée par l'ordinateur quand c'est à son tour de jouer et donner son algorithme.

---

[1] Dans l'exemple ci-dessus on part de la position de jeu de début mais le principe est le même en cours de partie

[2] Ceci est un moyen de déterminer le niveau de difficulté pour nous qui jouons contre l'ordinateur. Plus on lui permet de construire un arbre profond plus il peut analyser et produire le meilleur coup. Mais et c'est la contrepartie, il prend plus de temps pour répondre.

[3] Le diagramme doit rester lisible. Il est évident que la représentation de tous les objets que l'on peut identifier à partir du schéma sera peu lisible : Ne montrez que les objets pertinents et ne répétez pas des groupes d'objet similaires plusieurs fois ce qui surcharge inutilement le diagramme d'objets.

[4] Cet algorithme est un pionnier dans le domaine de la théorie des jeux et n'est plus utilisé aujourd'hui dépassé qu'il est par les algorithmes de classe alpha-beta.