10 Projet : MCTS

À rendre au plus tard le vendredi 8/01/2020 sur Moodle
Le projet peut-être réalisé en binôme ou seul.
On donnera les fichiers sources (`.cs`) du programme ainsi qu'un rapport décrivant les expériences menées et les résultats obtenus.

Sur Moodle, on trouvera dans le fichier "MCTS.txt" plusieurs classes permettant de mettre en œuvre l'algorithme MCTS pour un jeu à deux joueurs.

10.1 Description des classes

On considère une énumération `Resultat` et trois classes `Position`, `Partie` et `Joueur` pour modéliser un jeu à deux joueurs. Les trois premières valeurs de l'énumération `Resultat` correspond aux trois résultats possibles d'une partie : le joueur 1 gagne, le joueur 0 gagne, partie nulle. La dernière valeur `Resultat.indertermine` correspond à une partie qui n'est pas terminée et dont l'issue n'est pas déterminée.

La classe abstraite `Position` définit un type d'objet représentant l'état du jeu à un moment donné de la partie. En plus d'un constructeur, la classe `Position` a 6 membres :

- Un attribut `bool j1aletrait` qui détermine quel joueur a le trait (`true` si le joueur 1 a le trait)
- Une propriété `Eval` qui renvoie le résultat de la partie (une des 3 valeurs `Resultat.j1gagne`, `Resultat.j0gagne`, `Resultat.partieNulle`) si la position courante est une position terminale, c'est-à-dire pour laquelle l'issue du jeu est connue, ou sinon la valeur `Resultat.indetermine`. Elle doit être initialisée par le ou les constructeurs.
- Une propriété `NbCoups` qui renvoie le nombre de coups possibles à partir de la position courante (égale à 0 si la position courante est terminale). Elle doit être initialisée par le ou les constructeurs.
- Une méthode abstraite `void EffectuerCoup(int i)` qui modifie la position courante de sorte qu'elle corresponde à la position obtenue après avoir joué le (i+1) ème coup possible. Cette méthode doit aussi mettre à jour les propriétés `Eval` et `NbCoups`. L'ordre dans lequel les coups sont numérotés est arbitraire mais doit être toujours le même pour une position donnée.
- Une méthode abstraite `Position Clone()` qui renvoie une copie de la position courante. Aucune référence ne doit être partagée entre la copie et l'objet courant.
- Une méthode abstraite `void Affiche()` qui affiche une représentation de la position sur la console.

La classe abstraite `Joueur` définit un type d'objet représentant un joueur et a deux membres :

- la méthode abstraite `int Jouer(Position p)`, qui doit renvoyer l'indice du coup choisi à partir de la position `p` (cet indice est compris entre 0 et `p.NbCoups-1`).
- la méthode virtuelle `void NouvellePartie() { }` qui est invoquée par la classe `Partie`. Au besoin, dans une classe dérivée de `Joueur`, on redéfinira cette méthode pour initialiser l'instance avant une nouvelle partie.

La classe `Partie` fait jouer chaque joueur à son tour et afficher les positions jusqu'à la fin de la partie.

La classe `JoueurMCTS` dérivée de la classe `Joueur` met en œuvre l'algorithme MCTS (voir sur `https://en.wikipedia.org/wiki/Monte_Carlo_tree_search`) pour un temps fixé. L'attribut `float a` est un paramètre positif qui intervient pour le choix du coup le plus prometteur lors de la phase de sélection. La position fille choisie est celle pour lequel

$$\frac{W+a}{C+a}$$

est maximum où W est le nombre de victoires et C le nombre de traversées.

10.2 Les jeux

Le jeu des allumettes : C'est un jeu très simple : initialement, il y a un certain nombre d'allumettes sur la table. Chaque joueur, à son tour, prend 1, 2 ou 3 allumettes. Celui qui prend la dernière allumette a perdu.

Puissance 4 : Sur une grille de 7 colonnes de 6 cases, chaque joueur met à tour de rôle un pion de sa couleur en haut d'une colonne, et celui-ci descend en bas de la colonne. Celui qui en aligne 4 de sa couleur, verticalement, horizontalement ou en diagonale, a gagné.

10.3 Travail à faire

1. Écrire une classe `PositionA` qui dérive de la classe `Position` pour représenter une position dans le jeu des allumettes ains q'une classe `JoueurHumainA` qui dérive de la classe `Joueur` qui demande un coup à l'utilisateur dans le jeu des allumettes.
2. Vérifier que vos classes `PositionA` et `JoueurHumainA` permettent de faire jouer `JoueurHumainA` et `JMCTS` au jeu des allumettes.
3. Écrire une classe `PositionP4` qui dérive de la classe `Position` pour représenter une position dans le jeu Puissance4.
4. Écrire une classe `JoueurHumainPuissance` qui dérive de la classe `Joueur` qui demande un coup à l'utilisateur.
5. Reprendre la question (2) pour le jeu Puissance4.
6. Lors d'une partie on souhaite que la classe `JMCTS` puisse réutiliser l'arbre calculé au coup précédent (en réalité seulement le sous-arbre correspondant au coup de l'adversaire). Modifier la classe `JMCTS` en conséquence. On pourra utiliser la méthode virtuelle `bool Equals(object obj)` pour comparer deux instances de `Position`, et on prendra soin de la redéfinir dans la classe `PositionP4`. On devra aussi redéfinir la méthode `NouvellePartie` pour `JMCTS`.
7. Pour un temps limite de 100 ms, déterminer empiriquement une bonne valeur du paramètre a en organisant un championnat entre plusieurs joueurs. Pour simplifier on ne considérera que des valeurs entières pour a.

8. Dans la phase de simulation, au lieu de jouer une seule partie au hasard, on peut en jouer N en parallèle et utiliser le nombre de victoires obtenues pour la phase de rétropropagation. Concevoir une classe `JMCTSp` dérivant de `Joueur` qui met œuvre cet algorithme. On prendra en compte le fait que la classe `Random` n'est pas thread-safe.
9. Pour $N = 4$ reprendre la question 4 pour `JMCTSp`. Pour un temps de 100 ms, entre `JMCTS` et `JMCTSp` avec $N = 4$, quel algorithme est le plus performant ?
10. Une autre manière de paralléliser est la suivante :
    On considère une classe `NoeudP` avec les mêmes attributs que `Noeud` sauf `cross` et `win` qui sont des tableaux d'entiers de taille N. N threads explorent simultanément le même arbre (formé d'instance de `NoeudP`) : le thread numéro i met à jour `cross[i]` et `win[i]` lors de sa phase de rétropropagation. Lors de la phase de sélection, chaque thread utilise les statistiques $\sum_{i=0}^{N-1}$ `win[i]` et $\sum_{i=0}^{N-1}$ `cross[i]`. Il n'est pas nécessaire d'utiliser des verrous (`lock`).
    Concevoir les classes `NoeudP` et `JMCTSP` dérivant de `Joueur` pour mettre en œuvre cet algorithme.
11. Pour un temps limite de 100 ms et $N = 4$, quel est l'algorithme le plus performant entre `JMCTS`, `JMCTSp` et `JMCTSP` ?