Groupe de binôme : Trieu Xuong YU et Jingyi Liang

Rapport de projet informatique C# : MCTS

Jeu d'allumettes :

- Implémentation d'une classe dérivé PositionA qui dérive de la classe Position pour représenter une position dans le jeu d'allumettes. Autrement dit, on veut savoir combien y'a-t-il des allumettes restants au moment donné. Pour ça, on a redéfini des méthodes abstraites de la classe mère, c'est-à-dire la classe Position. Et on a défini des propriétés dans le constructeur.

  →Dans le constructeur, on a défini la propriété NbCoups qui est entre 1 et 3 quand le nombre d'allumettes restants est supérieur que 3 sinon il vaut le nombre d'allumettes restantes. Et on définit Eval comme indéterminé par défaut de chaque position.
  →Parmi ces méthodes abstraites, on constate que la méthode EffectuerCoup sert à modifier la position courante et ainsi que de mettre à jour les propriétés Eval et NbCoups. Ici, EffectuerCoup sert à retirer i (paramètre de la méthode EffectuerCoup) allumettes au moment de la position courante. Sachant qu'on veut faire jouer entre 2 joueurs JMCTS, on doit retirer i+1 allumettes au lieu de i allumettes, car dans la méthode JeuHasard de la classe JMCTS, un moment on voit une appel de la méthode EffectuerCoup avec un paramètre gen.Next(q.NbCoups) qui vaut 0/1/2. Sachant qu'on ne peut pas retirer 0 allumette dans ce jeu, par ailleurs on doit retirer entre 1 et 3 allumettes à chaque tour ! Donc dans la méthode EffectuerCoup, on va retirer i+1 allumettes ici, c'est-à-dire gen.Next(q.NbCoups)+1. Mais en faisant cette action, ça peut provoquer une erreur dans la classe JoeurHumainA qu'on va parler tout de suite.

- Implémentation d'une classe dérivée JoueurHumainA qui dérive de la classe abstrait Joueur. Pour celle-ci, on a redéfini la méthode abstrait Jouer avec le paramètre p, la position courante. Cette méthode doit retourner une valeur entière qui est le paramètre de la méthode EffectuerCoups de la classe PositionA. Par contre ici, on a retourné une valeur entière de (nb – 1) et pas de nb, parce qu'on a changé le nombre d'allumettes à enlever dans la méthode EffectuerCoup, donc au lieu de retourner nb, on a choisi nb – 1 pour d'être cohérant avec la méthode EffectuerCoup qu'on a implémenté.

Groupe de binôme : Trieu Xuong YU et Jingyi Liang

Jeu de Puissance4 :

- Implémentation d'une classe dérivé PositionP4 qui dérive de la classe Position pour représenter une position dans le jeu Puissance4. Autrement dit, on veut savoir combien y'a-t-il des pions dans chaque colonne au moment donné. On considère ce jeu est mise en place dans un tableau de taille 6 lignes et 7 colonnes. On a redéfini certaines méthodes abstraites et on a construit quelques méthodes instances pour simplifier des méthodes très compliquées, par exemple la méthode EffectuerCoup.

  - Tout d'abord on a implémenté la méthode EffectuerCoup avec un paramètre i qui est la colonne qu'on veut mettre notre pion dessous.
    - La première chose qu'il faut faire c'est que on doit vérifier si il y'a déjà une ou plusieurs colonnes qui sont déjà remplies ou pas. Et ensuite on pourra décaler indice i n fois avec n qui est le nombre des colonnes déjà remplies avant la i-ème colonne.
    - Pour la mise à jour le NbCoups d'après d'avoir fait jouer un des deux joueurs, on a besoin juste de vérifier si la position qu'on vient de jouer est situé à la première ligne, si oui, on effectue une décrémentation au Nbcoups.
    - Pour la mise à jour le Eval d'après d'avoir fait jouer un des deux joueurs, on a besoin de vérifier si il y'a quatre pions sont alignés horizontalement, ou verticalement, et ou diagonalement. Pour celle-ci, on a créé une double boucle de for pour parcourir chaque valeur du tableau, on vérifier les 3 conditions (4 alignements horizontalement, verticalement et diagonalement) avec une autre boucle de for pour parcourir les 3 valeurs à côté / dessus/ à diagonale de droite à gauche et de gauche à droite. On vérifie si ces 3 valeurs sont pareilles que chaque tab[lig, col]. Si oui, alors Eval va d'être j0.gagne ou j1.gagne et NbCoups devient à 0.
    - Et ensuite on doit vérifier si un des deux joueurs a gagné, ou c'est un match nul. On a implémenté une méthode qui est verif_Match_nul pour celle-ci. Dans cette méthode, on va juste parcourir chaque colonne de la première de ligne du tableau pour voir si la valeur à la première ligne de chaque colonne est-elle différente à 0. Si oui, alors le résultat est un match nul.

  - Ensuite on a juste redéfini la méthode abstrait Clone en copiant tous les éléments de la position courante, c'est-à-dire le NbCoups et Eval. On a aussi redéfini la méthode Affiche afin de se rendre une meilleure vision au jeu.

Groupe de binôme : Trieu Xuong YU et Jingyi Liang

- Implémentation d'une classe dérivée JoueurHumainP4 qui dérive de la classe abstrait Joueur. Pour celle-ci, on a redéfini la méthode abstrait Jouer avec le paramètre p, la position courante. La méthode retourne une valeur entière qui la colonne qu'on veut mettre notre pion dessous.

Un championnat de la league :

Pour un temps limité de 100ms, on veut déterminer une bonne valeur du paramètre a de la classe JMCTS en organisant un championnat entre plusieurs joueurs JMCTS avec différentes paramètres de a = 1 jusqu'à a = 50. Dans ce championnat, on a fait jouer chaque joueur contre chaque joueur. Et on aura une score sur n / 50 pour chaque joueur avec n qui est le nombre de partie gagné sur les 50 parties jouées.

Au final, on constate que le meilleur joueur est le JMCTS avec a = 11, ce joueur a gagné 38 / 50 à la première lancée, 35/50 à la deuxième lancée et 39 / 50 à la troisième lancée. En même temps la score de ce joueur est dominée aux autres joueur, donc on en déduit que a = 11 est une bonne valeur de paramètre a.

Parallélisassions - JMCTSp:

Dans la phase de simulation, au lieu de jouer une seule partie au hasard, on peut en jouer N partie en parallèle et utiliser le nombre de victoires obtenues pour la phase de rétropropagation. On a conçu une classe JMCTSp qui ressemble de la classe JMCTS dérivant de la classe Jouer qui met en œuvre cet algorithme.

La différente entre JMCTS et JMCTSp est que dans la méthode de JeuHasard, le JeuHasard de JMCTSp contient un paramètre en plus, une position p et une indice i. Car dans cette classe JMCTSp, on ne peut plus d'utiliser la même Random tout temps, car la classe Random n'est pas un thread safe. Donc on a initialisé et instancié un tableau de Random pour lui bien mettre dans la méthode de JeuHasard. Et puis pendant la simulation, on lance un thread pour qu'il puisse faire 4 parties en même temps, et puis on peut obtenir le nombre de victoire comme la somme des résultats de tous les threads pendant la rétropropagation.

Pour N = 4 pour JMCTSp et un temps limité de 100ms, on a lancé 50 parties entre JMCTS et JMCTSp, le résultat est plutôt équilibre, c'est un peu étrange comme résultat, parce que logiquement JMCTSp doit être un peu plus intelligent que JMCTS… Alors on constate que peut-être notre classe de PositionP4 n'est pas encore parfaite…

Groupe de binôme : Trieu Xuong YU et Jingyi Liang

Les difficultés rencontrées :

Selon le projet, la plus grosse difficulté pour nous est ne pas avoir une classe positionP4 parfaite, car le joueur JMCTS n'est pas très intelligent quand on joue avec lui, du coup on ne peut pas considérer JMCTS fonctionne parfaitement avec notre classe de PositionP4. Ce problème va provoquer des erreurs de certains tests par exemple pour tester la performance entre des joueurs…