Projet C ♯ - M2 ISIFAR

Liang Jingyi

January 2021

1 Introduction

Dans ce projet de C sharp, on a l'occasion d'étudier l'implémentation de l'algorithme Monte Carlo Tree Search (MCTS), un algorithme de prise de décision. Le jeu nous sont proposé pour pouvoir appliquer est le jeu Tron.

Il y a 4 phases pour réaliser cet algorithme de prise de décision:

• Sélection: Depuis la racine, on sélectionne successivement des enfants jusqu'à atteindre une feuille(c'est à dire que les dernières positions possibles), le choix se fait par une fonction de choix comme en bas:

$$\phi(a, W, C) = \frac{a+W}{a+C}$$

• Expansion: si cette feuille n'est pas finale, créer un enfant (ou plusieurs) en utilisant les règles du jeu et choisir l'un des enfants.

• Simulation: simuler une exécution d'une partie au hasard depuis cet enfant, jusqu'à atteindre une configuration finale.

• Rétropropagation (Backpropagation): utiliser le résultat de la partie au hasard et mettre à jour les informations sur la branche en partant du nœud enfant vers la racine.

Notre example le jeu Tron est un jeu de plateau à 2 joueurs et à coups simultanés. Le jeu se joue sur une où les joueurs se déplacent simultanément. A chaque tour, chaque joueur doit se déplacer d'une case (vers le haut ou vers la bas ou à gauche ou à droite) et arriver sur une case libre, c'est-à-dire une case où ne se trouve pas son adversaire et où ne se trouve pas une pierre. De plus chaque joueur laisse une pierre sur la case qu'il vient de quitter. Si les deux joueurs arrivent sur la même case, ils sont téléportés sur leur case de départ. Le gagnant est le dernier joueur qui peut se déplacer. Si aucun joueur ne peut se déplacer, la partie est nulle.

2 Questions préliminaires: 1-2

Ces questions s'agissent de la mise en place du jeu Tron. Tout d'abord, l'implementation d'une classe derivée PositionTron contients 4 phases:

2.1 Construire le plateau

La construction commence par la creation du tableau de dimension 2, ici j'ai choisi un tableau de 7*7, mais effectivement ça marche aussi avec différents dimensions. Le problème que j'ai rencontré c'est de vérifier que les joueurs ne se déplacent pas hors des bords, pour cela, j'ai remplacé le plateau 7*7 par un plateau 9*9 et rempli les 4 bords avec le numéro 8, on les considère comme des pierres autour du plateau. En même temps les coordonnées de la case départ des joueurs devraient être changées aussi dans le nouveau plateau.

Dans la méthode PositionTron il y a 4 paramètres: la coordonnée de la case départ du joueur 1 et la coordonnée de la case départ du joueur 0, dans un instance j'ai mis (1,1) pour le joueur 1 et (3,3) pour le joueur 0. Mais ceci pourrait modifier par l'utilisateur.

• Le parcours du joueur 1 est rempli par le numéro 3 (comme des pierres), et la position courante du joueur 1 est représentée par le numéro 1.

• Le parcours du joueur 0 est rempli par le numéro 4 (comme des pierres), et la position courante du joueur 0 est représentée par le numéro 2.

```
static int nbL = 7; //nombre de ligne du plateau
static int nbC = 7; //nombre de colonnes du plateau
1 个引用
public PositionTron(int pJ10, int pJ11, int pJ00, int pJ01)
//Construire PositionTron avec les cases de depart de deux joueurs
{
    this.plateau = new int[nbL + 2, nbC + 2];
    for (int i = 0; i < nbL + 2; i++)
    {
        for (int j = 0; j < nbC + 2; j++)
        {
            if (i == 0 || i == (nbL + 1) || j == 0 || j == (nbC + 1))
            {
                plateau[i, j] = 8;//remplir les bords avec le numéro 8
            }
        }
    }
    this.pJ1 = new int[2] { pJ10 + 1, pJ11 + 1 };//modifier les coordonées des cases de départ
    this.pJ0 = new int[2] { pJ00 + 1, pJ01 + 1 };//modifier les coordonées des cases de départ
    this.pJ1D = new int[2] { pJ10 + 1, pJ11 + 1 };
    this.pJ0D = new int[2] { pJ00 + 1, pJ01 + 1 };
    plateau[pJ1[0], pJ1[1]] = 1;
    plateau[pJ0[0], pJ0[1]] = 2;
    this.Eval = 0;
```

Figure 1: Construction du Plateau

2.2 Calculer le nombre de coups

Pour calculer le nombre de coups, j'ai besoin de regarder les 4 directions à côté depuis la position courante, si la case est nulle, c'est un coup possible, la valeur du nombre de coups est entre 0

et 4 (contient 0 et 4), j'ai utilisé un double boucle pour découvrir les 4 directions et aussi j'ai créé deux tableaux direc1 et direc2 pour entregistrer les coordonnées des coups possibles pour joueur1 et joueur2.

```
int nbcoup1 = 0;
int t = 0;
for (int p = -1; p < 2; p++)
{
    for (int q = -1; q < 2; q++)
    {
        if (Math.Abs(p + q) == 1 && this.plateau[this.pJ1[0] + p, this.pJ1[1] + q] == 0)
        {
            nbcoup1++;
            direc1[t, 0] = p;
            direc1[t, 1] = q;

        }
        if (Math.Abs(p + q) == 1)
        { t++; }

    }
}
this.NbCoups1 = nbcoup1;
```

Figure 2: Construction du Nbcoups1

2.3 Effectuer le coup et mettre à jour les propriétés

J'ai implémenté la méthode EffectuerCoup avec deux paramètres i et j qui est le choix de la direction du joueur. On modifie la position courante de sorte qu'elle corresponde à la choix du joueur. Ici, encore j'ai utilisé un double boucle et aussi défini les différents mouvements avec différents choix du Joueur.

Quand il y a 4 coups possibles,les joueurs ont 4 choix entre 0 et 3:

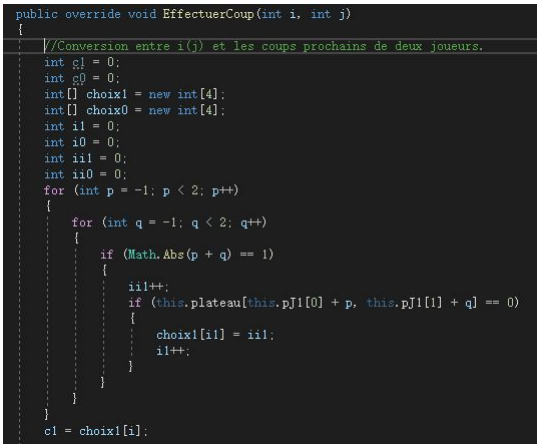
si le choix du joueur est 0, on bouge vers le haut; si le choix du joueur est 1, on bouge à gauche; si le choix du joueur est 2, on bouge à droit; si le choix du joueur est 3, on bouge vers le bas.

Quand il y a moins de 4 coups possibles, les joueurs ont moins de 4 choix, mais l'ordre du mouvement reste la même (c'est à dire l'ordre le haut,gauche,droit,le bas en suprimmant les directions impossible).

Quand les joueur croisent sur la même case après effectuer le coup, on les fait retourner au point départ.

Ensuite on devrait mettre à jour le nombre de coups Nbcoups1 et Nbcoups0 depuis la nouvelle case, la méthode est la même. Pour la mise à jour le Eval, on défini Eval=1 si joueur1 a gagné; Eval=-1 si joueur0 a gagné; Eval=0 si la partie est nulle ou pas encore fini.

Travaille 2 nous a demandé d'implémenter une classe dérivée JoueurHumainTron qui dérive de la classe abstrait joueurS. Pour celle-ci, j'ai redéfini la méthode abstrait "Jouer" avec les paramètres p (la posotion courante) et asj1 (boolean si joueur1 a le trait).Je fais entrer le choix de l'utilisateur et retourne la valeur entière qui est le paramètre de la méthode EffectuerCoups de la classe PositionTron.

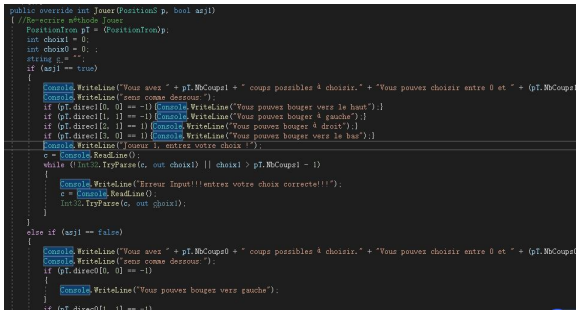


```
public override void EffectuerCoup(int i, int j)
{
    //Conversion entre i(j) et les coups prochains de deux joueurs.
    int c1 = 0;
    int c0 = 0;
    int[] choix1 = new int[4];
    int[] choix0 = new int[4];
    int i1 = 0;
    int i0 = 0;
    int ii1 = 0;
    int ii0 = 0;
    for (int p = -1; p < 2; p++)
    {
        for (int q = -1; q < 2; q++)
        {
            if (Math.Abs(p + q) == 1)
            {
                ii1++;
                if (this.plateau[this.pJ1[0] + p, this.pJ1[1] + q] == 0)
                {
                    choix1[i1] = ii1;
                    i1++;
                }
            }
        }
    }
    c1 = choix1[i];
```

Figure 3: Méthode EffectuerCoup1(1)

```
//Renouveler pJ1
if (c1 == 1 && plateau[this.pJ1[0] - 1, this.pJ1[1]] == 0)
{
    plateau[this.pJ1[0], this.pJ1[1]] = 3;
    this.pJ1[0] = this.pJ1[0] - 1;
    this.pJ1[1] = this.pJ1[1];
}
else if (c1 == 3 && plateau[this.pJ1[0], this.pJ1[1] + 1] == 0)
{
    plateau[this.pJ1[0], this.pJ1[1]] = 3;
    this.pJ1[0] = this.pJ1[0];
    this.pJ1[1] = this.pJ1[1] + 1;
}
else if (c1 == 4 && plateau[this.pJ1[0] + 1, this.pJ1[1]] == 0)
{
    plateau[this.pJ1[0], this.pJ1[1]] = 3;
    this.pJ1[0] = this.pJ1[0] + 1;
    this.pJ1[1] = this.pJ1[1];
}
else if (c1 == 2 && plateau[this.pJ1[0], this.pJ1[1] - 1] == 0)
{
    plateau[this.pJ1[0], this.pJ1[1]] = 3;
    this.pJ1[0] = this.pJ1[0];
    this.pJ1[1] = this.pJ1[1] - 1;
}
```

Figure 4: Méthode EffectuerCoup1(2)



```
public override int Jouer(Position2 p, bool asj1)
{ //Re-ecrire méthode Jouer
    PositionTron pT = (PositionTron)p;
    int choix1 = 0;
    int choix0 = 0;
    string c = "";
    if (asj1 == true)
    {
        Console.WriteLine("Vous avez " + pT.NbCoups1 + " coups possibles à choisir." + "Vous pouvez choisir entre 0 et " + (pT.NbCoups1
        Console.WriteLine("sens comme dessous:");
        if (pT.direc1[0, 0] == -1) {Console.WriteLine("Vous pouvez bouger vers le haut");}
        if (pT.direc1[1, 1] == -1) {Console.WriteLine("Vous pouvez bouger à gauche");}
        if (pT.direc1[2, 1] == 1) {Console.WriteLine("Vous pouvez bouger à droit");}
        if (pT.direc1[3, 0] == 1) {Console.WriteLine("Vous pouvez bouger vers le bas");}
        Console.WriteLine("Joueur 1, entrez votre choix !");
        c = Console.ReadLine();
        while (!Int32.TryParse(c, out choix1) || choix1 > pT.NbCoups1 - 1)
        {
            Console.WriteLine("Erreur Input!!!entrez votre choix correcte!!!");
            c = Console.ReadLine();
            Int32.TryParse(c, out choix1);
        }
    }
    else if (asj1 == false)
    {
        Console.WriteLine("Vous avez " + pT.NbCoups0 + " coups possibles à choisir." + "Vous pouvez choisir entre 0 et " + (pT.NbCoups0
        Console.WriteLine("sens comme dessous:");
        if (pT.direc0[0, 0] == -1)
        {
            Console.WriteLine("Vous pouvez bougez vers gauche");
        }
```

Figure 5: Méthode Jouer dans la classe JoueurHumainTron

3 JMCTSS et le championnat

Travail 3 nous a demandé de modifier la classe JMCTSS pour qu'elle soit plus performante , c'est à dire que l'algorithme garde en mémoire des parcours qui sont été calculé. Donc j'ai essayé de ajouter une méthode "Equals" dans la classe PositionTron,cette méthode me permet de comparer deux positions et choisir le noeud fils à garder. Malheusement cela fonctionne pas assez bien. Pour une visibilité, je vous monte la partie de code concernée:

```
public override bool Equals(Object obj)
{
    if ((obj == null) || !this.GetType().Equals(obj.GetType()))
        return false;
    PositionTron position = (PositionTron)obj;
    bool equal = ((this.NbCoups1 == position.NbCoups1) && (this.Eval == position.Eval))|| (this.NbCoups0 == position.NbCoups0) && (this.Eval == position.Ev
    for (int i = 0; i < nbL; i++)
    {
        for (int j = 0; j < nbC; j++)
        {
            if (this.plateau[i, j] != position.plateau[i, j])
            {
                equal = false;
                break;
            }
        }
        if (!equal)
            break;
    }
    return equal;
}
```

Figure 6: Méthode Equals

Ensuite, on veut trouver une valeur optimale de "a" pour intervenir dans le choix de la position fille, la positon fille choisie est celle pour lequel $\frac{W+a}{C+a}$, W est la somme des gains et C est le nombre de traversées.Pour cela, on pourrait lancer un champinnat entre plusieurs joueurs avec les différents "a", et ce qui a le plus de victoire est le meilleur, on pourra trouver empiriquement le "a" optimal en répétant cette action.

Dans ce championnat(dans le code c'est "champinnat") j'ai pris 50 joueurs et j'ai fait jouer chaque joueur contre chqaue joueur,et j'ai pris $a \in [0, 50]$, au début les résultats que j'ai reçus sont pas très clairs, les différences entres les différents "a" sont pas évidents. Au final, en répétant le championnat, j'ai constaté que le paramètre optimal est: a=22. (En fait, les résultants sont bien quand la valeur de "a" est entre 10-20):

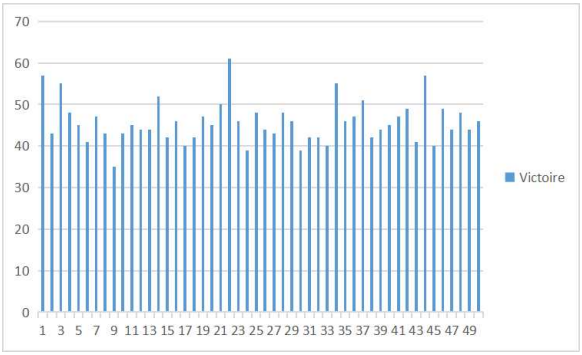


Figure 7: Résultat de nombre de victoires de JMCTSS

On voit que le paramètre optimal est a = 22.

4 Parallélisations-JMCTSSp Threading et optimisation

On cherchera par la suite à paralléliser la phase de similation. 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 JMCTSSp qui ressemble à la classe JMCTSS qui met en œuvre cette amélioration.

4.1 Réalisation de JMCTSSp

La différente entre JMCTSS et JMCTSSp est que dans la méthode de JeuHasard, le JeuHasard de JMCTSSp contient un paramètre en plus, une position p comme dans JMCTSS et une indice i. Car dans cette classe JMCTSSp, 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 dans la simulation de la méthode "jouer", on lance un thread pour qu'il puisse faire plusieurs 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 JMCTSSp et un temps limité de 100ms, j'ai lancé le championnat (dans le code c'est la méthode "champinnatp") de 50 joueurs, et j'ai fait jouer chaque joueur contre chaque joueur avec $a \in [0, 50]$, Comme le résultat de JMCTSS,les différences entres les différents "a" sont pas évidents. Aprés lancer plusieurs fois, j'ai constaté que le paramètre optimal est: a=12.

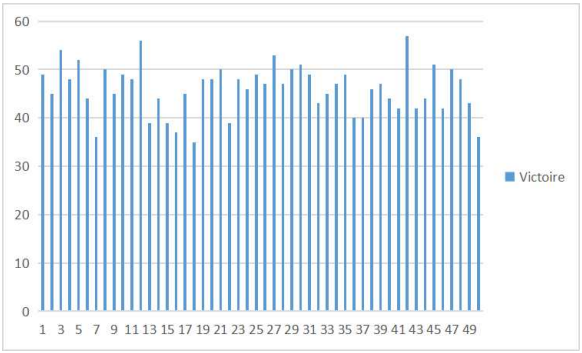


Figure 8: Résultat de nombre de victoires de JMCTSSp

On voit que le paramètre optimal est a = 12.

5 Performance de JMCTSS et de JMCTSSp

Pour comparer le performance de JMCTSS et JMCTSSp quand N=2, j'ai écrit un méthode "VersusTron" qui nous permet de faire jouer 2 joueurs différents plusieurs parties pour voir les résultats et trouver le plus performant algorithme. Le code comme dessous: J'ai lancé 100 par-

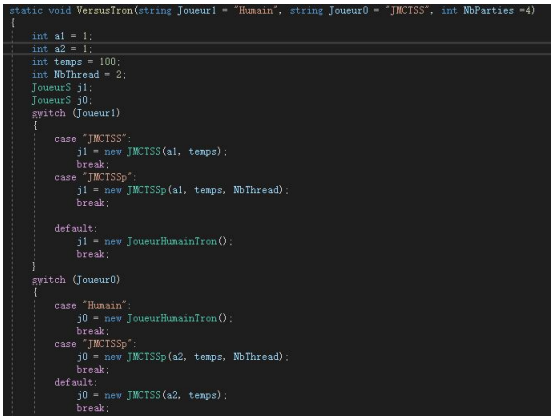


Figure 9: L'algorithme VersusTron(1)

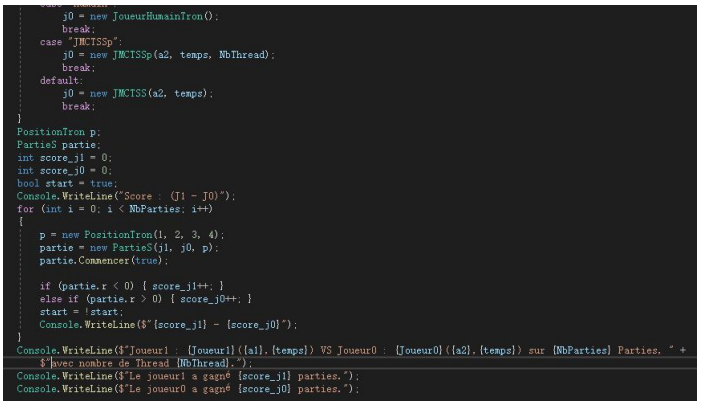


Figure 10: L'algorithme VersusTron(2)

ties avec différents "a"("a1" représent "a" de JMCTSS; "a2" représente "a" de JMCTSSp), j'ai constaté que JMCTSSp est un peu plus performant que JMCTSS. Le score est comme dessous:

• Quand a1=1 , a2=1: 32-51 parties de victoire pour JMCTSSp ; 30-48 partie de victoire pour JMCTSS. Avec a1, a2 égale à 1, je vois pas trop de différence entre JMCTSS et JMCTSSp, parfois JMCTSS a gagné plus de partie dans une opération.

• Quand a1=12 , a2=12: 45-55 parties de victoire pour JMCTSSp ; 40-48 partie de victoire pour JMCTSS. Pour JMCTSSp "a" optimal que j'ai constaté est 12, donc avec ce "a", JMCTSSp est clairement mieux que JMCTSS, et c'est logique.

• Quand a1=22 , a2=12: 40-48 parties de victoire pour JMCTSSp ; 45-53 partie de victoire pour JMCTSS. Avec a1, a2 égale à 22 et 12, le résultat n'est pas comme j'imagine , JMCTSS a gagné plus. C'est un peu biarre parce que théoriquement JMCTSSp doit être plus intélligent.

• Quand a1=22 , a2=22: 40-56 parties de victoire pour JMCTSSp ; 39-46 partie de victoire pour JMCTSS.

6 JMCTSSP : Deuxième approche de multi-threading

La deuxième approche est JMCTSSP, on doit écrire une autre classe "NoeudSP" avec les même attributs que "NoeudS" sauf "cross" et "win" qui sont des tableaux d'entiers de taille N. Donc j'ai défini la classe NoeudSP, la partie de code concernant comme dessous:

```
public class NoeudSP
{
    static Random gen = new Random();

    public PositionS p;
    public NoeudSP pere;
    public NoeudSP[,] fils;
    public int[] win, cross;
    public int N;
    public int[] indiceMeilleurFils1, indiceMeilleurFils0;
    3 个引用
    public NoeudSP(NoeudSP pere, PositionS p, int nb_thread)
    {
        this.pere = pere;//
        this.p = p;//
        this.fils = new NoeudSP[this.p.NbCoups1, this.p.NbCoups0];//
        this.N = nb_thread;
        this.win = new int[nb_thread];
        this.cross = new int[nb_thread];
        for (int i = 0; i < N; i++)
        {
            this.win[i] = 0;
            this.cross[i] = 0;
        }
        this.indiceMeilleurFils1 = new int[nb_thread];
        this.indiceMeilleurFils0 = new int[nb_thread];
    }
```

Figure 11: L'algorithme NoeudSP(1)

```
public int totale_win
{
    get
    {
        int totale_W = 0;
        for (int i = 0; i < this.N; i++)
        {
            totale_W += win[i];
        }
        return totale_W;
    }
}
6 个引用
public int totale_cross
{
    get
    {
        int totale_C = 0;
        for (int i = 0; i < this.N; i++)
        {
            totale_C += cross[i];
        }
        return totale_C;
    }
}
```

Figure 12: L'algorithme NoeudSP(2)

Malheusement l'implémentation de JMCTSSP n'est pas parfait, mon but est de lancer plusieurs threads afin d'actualiser "win[i]" et "cross[i]" pendant retropogation mais cela n'a pas marché pas très bien.

7 Conclusion

Au cours de ce projet, j'ai mis en pratique les connaissances du cours:

• Dans un premier temps à intégrer au sein du jeu Tron, l'algorithme Monte Carlo Tree Search pour qu'on ait un algorithme de prise de décision performante.

• Dans un second temps, j'ai amélioré JMCTSS en utilisant threading, et le résultat montre que le nouvel algorithme JMCTSSp est légèrement plus intélligent que JMCTSS.

Néanmoins, j'ai rencontré aussi des difficultés comme l'implémentation de JMCTSSP.

Pour conclure, ce projet me permet de renforcer mes connaissance du cours dans un cas concret, c'est une très bonne expérience.