**REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE**

**الجــــــــــــــــــــمهوريــــــــــــــــــــــــــــــة الجـــــــــــــــــــــزائريــــــــة الديــــمـــــــقراطيـــــــــــة الشــــــــــــــــعبية**

**MINISTERE DE L’ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE SCIENTIFIQUE**

**وزارة التــــــــعليــــــــــــــم العــــــــــــــــــــــالي والبــــــــــــــــــــــحث العلـــــــــــــــــــــــــــــــــــــــــــــــمي**



**Ecole nationale Supérieure d’Informatique**

**ex. INI (Institut National de formation en Informatique)**

**RAPPORT DE TP N°03 DE TPGO**

**Thème :**

|  |
| --- |
| ***Amélioration de fonction d’estimation MinMax pour les échecs*** |

**Réalisé par :**

* ADRAO Nassim
* BENCHOHRA Mohamed Amine

**Option :** SIQ

**Groupe :** 01

**Encadré par :** Mr. HADIM Boukhalfa

**Enoncé du problème :**

Le jeu d’échecs est très important dans la théorie de programmation et dans l’intelligence artificielle et cela est dû à sa complexité qui nécessite trop de temps de calcul.

Le jeu d’échecs (prononcer [eʃɛk]) oppose deux joueurs de part et d’autre d’un tablier appelé échiquier composé de soixante-quatre cases claires et sombres nommées les cases blanches et les cases noires. Les joueurs jouent à tour de rôle en déplaçant l'une de leurs seize pièces (ou deux pièces en cas de roque), claires pour le camp des blancs, sombres pour le camp des noirs. Chaque joueur possède au départ un roi, une dame, deux tours, deux fous, deux cavaliers et huit pions. Le but du jeu est d'infliger à son adversaire un échec et mat, une situation dans laquelle le roi d'un joueur est en prise sans qu'il soit possible d'y remédier.

Une estimation de la complexité du jeu d'échecs est le nombre de Shannon, soit , c'est-à-dire du nombre de parties différentes, ayant un sens échiquéen, possibles. Ce nombre est à distinguer du nombre, beaucoup plus élevé, de parties légales qu'autorisent les règles du jeu.

**Solution :**

Une des solutions pour le problème de jeu d’échecs est l’utilisation de l’algorithme MinMax (en annexe) qui est très implémenté dans les jeux à 2 joueurs. Son principe est simple : On construit un arbre dont la racine est l’état initial, les feuilles sont les états de fin de jeu, qui représentent le succès ou la perte

Or pour le jeu d’échec, cela est gourmand en temps et en ressources.

C’est pour cela on passe à l’heuristique et on essaye d’approcher les meilleurs résultats. L’heuristique se base sur deux aspects : La programmation dynamique et l’estimation

Pour de bons résultats, une bonne fonction d’estimation doit être élaborée, d’où l’importance de fonction d’estimation

Donc le but est d’améliorer la fonction d’estimation du MinMax pour optimiser les résultats.

La fonction d’estimation existante prend en considération le nombre de pièce. Est-il suffisant ?

La réponse est non, on doit aussi prendre en considération les mouvements possibles, des poids des cases pour chaque type. Pour cela on introduit les matrices

On va donc exploiter les algorithmes de parcours de graphe (DFS est pris comme exemple car son implémentation est simple) et puis tester la connectivité du graphe.

Une variance d’amélioration de l’algorithme consiste à traiter la composante connexe atteinte par le nœuds testé au lieu de parcourir tout le graphe (et par conséquent parcourir des composantes connexes indépendants du nœud dons elles ne s’affecte pas par la suppression de ce nœud).

**Comment cela marche :**

* Soit la matrice d’adjacence du graphe initialisée au début à 0 (tous ses éléments sont nuls).
* On remplit la matrice comme suit :
  + Si le nœud d’indice i est adjacent à un nœud d’indice j alors ou ( on exploite la symétrie de la matrice pour ne remplir que la matrice triangulaire inférieure et par conséquent gagner en espace).
* On effectue l’algorithme DFS (en annexe). Il est nécessaire d’éclairer certains points :
  + On exploite la symétrie de la matrice d’adjacence et utiliser la première ligne pour marquer les nœuds visités (nœud visité ⬄ ).
  + On a utilisé une variante de l’algorithme DFS qui a un paramètre supplémentaire : le nœud exclu de parcours. Ce paramètre est comparé avec les nœuds voisins pour l’exclure du résultat au lieu de supprimer les arcs et les restituer à chaque nœud du graphe. DFS modifié s’applique à un des voisins du nœud qui a subi un DFS normal.
* Pour chaque nœud du graphe :
  + A chaque marquage du nœud dans le DFS normal, on incrémente une variable , et on décrémente cette dernière dans le DFS modifié, et à la fin
    - Si égal à 1 : le nœud n’est pas un point d’articulation (on a pu rejoindre tous les voisins de i sans sa présence)
    - Sinon : le nœud est un point d’articulation.

**Calcul de la complexité :**

On considère ce qui suit :

* Pour effectuer le DFS que ce soit normal ou modifié, on parcourt cases, n le nombre de nœuds.
* Le premier processus est appliqué à tous les nœuds.

Alors :

**Annexe :**

1. **DFS() : // algorithme standard,** i l’identifiant du noeud

Marquer(i) comme visité ;

POUR chaque adjacent à

SI maqrue() = « non visité »

DFS();

FSI

FPOUR

1. **DFS() : //** utilisé dans notre algorithme, retourne le premier voisin rencontré, i l’identifiant du noeud

Marquer(i) comme visité ;

POUR chaque adjacent à

SI maqrue() = « non visité »

Si premier élément rencontré, le retourner

Incrémenter ;

DFS();

FSI

FPOUR

Retourner -1 ; //si aucun élément est voisin

**DFSModifié() : //** i l’identifiant du nœud, k l’identifiant du nœud qu’on veut exclure

Marquer(i) comme visité ;

POUR chaque adjacent à

SI maqrue() = « non visité » et

Décrémenter ;

DFS();

FSI

FPOUR

**EstPointArticulation() : //** testet si est un point d’articulation et retourne vrai, sinon faux

Rendre tous les nœuds non visité ;

1. =DFS() ; //

DFS() ; //

Si ==1 retourner faux ;

Sinon retourner vrai ;