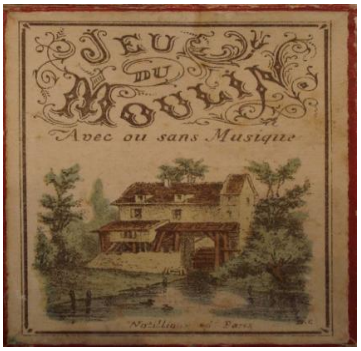
المملكة المغربية
Royaume du Maroc

Ecole Hassania des Travaux Publics المدرسة الحسنية للأشغال العمومية

Résolution de Problèmes

Rapport du jeu de MOULIN



Réalisé par :

IREHAL Fatima

El Kanfoudi Mohamed

Encadré par :

Madame ADDOU

Table des matières

1

1 INTRODUCTION

1.1 Contexte

Le jeu du moulin est un projet réalisé dans le cadre du module de “résolution de problèmes”.

L’objectif de ceci est d’implémenter une IA dans le jeu avec les deux stratégies

Minimax et Alpha-beta à partir des outils acquis dans le module.

1.2 Histoire du jeu

Des traces du jeu ont été découvertes au temple de Kurna (c’est-à-dire Cheikh Abd el-Gournah), sur l'une des rives du Nil, qui existait au XIV^e^ siècle av. J.-C. C’est du moins ce que croyait Henry Parker (*Ancien Ceylon*, Londres, 1909), mais il a depuis été démontré par les archéologues allemands qui ont fouillé le site, que les gravures de jeux présentes sur les dalles du toit effondré du "Temple des millions d'années de Séthi Ier" sont bien plus tardives. Le jeu du moulin encore visible porte en son centre une croix propre aux Coptes (déjà relevée par Parker), ce qui laisse entendre que, comme les autres jeux présents, il a été gravé après l’arrivée du Christianisme. Des plateaux ont aussi été découverts lors des fouilles du dernier niveau de la ville de Troie, tout comme dans des tombes de l'âge du bronze en Irlande[1]. Cependant, aucune référence sérieuse n’a jamais été produite pour Troie, et les tombes de l'âge du bronze en Irlande, découvertes au XIX^e^ siècle, ne peuvent être datées, qui laisse ouverte la question de l’ancienneté du jeu du moulin. Les historiens penchent aujourd’hui pour une date bien plus tardive, peut-être aux temps de la Rome antique. Le jeu est cité dans le *Livre des jeux* d'Alphonse X de Castille (sous le nom d'*alquerque de nueve*), ainsi que dans le Talmud[1].

« L'intérêt tout particulier de ce jeu ne tient pas seulement à son universalité (d'époques et de lieux), mais aussi à sa surprenante simplicité et à son intelligence. » Le jeu du moulin est répandu à travers le monde, où il est connu sous divers noms : jeu des mérelles (ou marelles), *Mühle* en Allemagne, *Nine Men's Morris* en Grande-Bretagne et aux États-Unis.

1.3 Présentation du jeu

« Jeu du Moulin », un jeu de société traditionnel européen qui trace ses origines de l'Egypte antique et de la Rome antique. Il affronte à deux personnes sur un plateau 7x7 formé par quatre carrés concentriques reliés au centre de leurs quatre côtés par des lignes perpendiculaires. Le jeu se déroule sur les 24 points du tableau (les 12 coins des carrés et les 12 intersections qu´ils forment avec les lignes perpendiculaires).rente pour chacun d´eux. Chaque joueur a 9 pions (9 hommes de Morris), de couleurs différentes.

Outils utilisés

Langage de programmation utilisé : C

Pour la version graphique on a opté pour la bibliothèque graphique sdl

On a utilisé comme outil CodeBlocks pour les 2 versions

1.4 Difficultés rencontrées

- Les problèmes qu'on a rencontré pendant la réalisation du projet c la difficulté de coordinance pendant la phase du travail à distance à cause de la mauvaise connexion.
- La gestion du temps.

2

2 Modélisation du problème

2.1 Espaces d'états

On a définie 2 structures principales qui caractérisent le joueur ; une structure **etat** qui a pour champs une matrice P contenant des lettres de 'a' à 'x' , cette matrice est définie comme une autre structure **place** qui contient tous les éléments qui caractérisent un joueur à savoir son numéro (Max ou Min), sa lettre, son couleur, et ses voisins(haut – bat – gauche – droite)

La grille est construite par une fonction s'appelle **afficher()**, chaque position dans la grille est caractérisée par P[k] représentant chaque intersection où on peut mettre un jeton

Objectif :

Chaque joueur a pour objectif la réalisation du plus grand nombre de moulin possible (un moulin représente l'alignement de trois pions de la même couleur) afin de capturer les jetons de l'adversaire. Le joueur qui a réussi à capturer 7 jetons gagne.

Le déroulement du jeu :

Le jeu se déroule en trois phases :

— La première phase : La grille est initialement vide, donc P[24].joueur=0. Les deux joueurs commencent à poser leurs 9 jetons à tour de rôle.

— La deuxième phase : à tour de rôle, Les deux joueurs déplacent un jeton à la fois vers une intersection voisine libre, après avoir posé tous les 9 jetons.

— La troisième phase : Celui qui ne possède plus que trois pions peut alors se déplacer en sautant où il veut.

Le jeu s'achève quand un joueur n'a plus que deux pions ou ne peut plus jouer, il est alors le perdant.

Règles du jeu :

- Le jeu commence avec un plateau vide entre deux joueurs Max et Min.
- Les joueurs Max et Min doivent placer à tour de rôle leurs pions (neuf pions au maximum) sur un point d'intersection libre du plateau, suivant la couleur qui leur est attribuée : ν pour Max et ν pour Min.
- Le but est de faire un « moulin : trois pions alignés ». Si un joueur forme un moulin, il retire un pion à son adversaire (en dehors d'un moulin éventuel).
- Quand les pions sont tous posés, les joueurs peuvent les glisser d'un point d'intersection à un autre point de la ligne (un mouvement à la fois).
- Quand un joueur n'a que trois pions, il peut sauter d'un point à un autre.
- Les pions formant un moulin sont intouchables par l'adversaire. Mais s'ils sont déplacés par le propriétaire, ils deviennent vulnérables.
- Le jeu continue ainsi jusqu'à ce qu'un joueur n'a que deux pions en sa possession. Dans ce cas, son adversaire est déclaré gagnant.

Stratégies :

On va se baser sur deux stratégies :

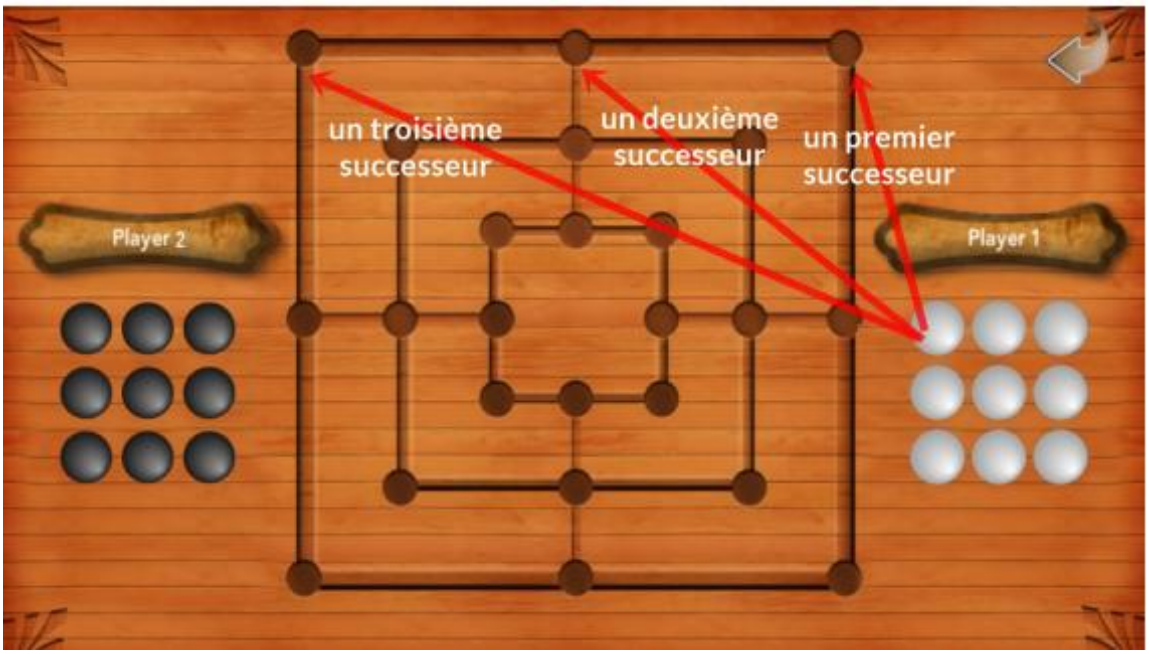
— Minimax : Il amène l'ordinateur à passer en revue toutes les possibilités pour un nombre limité de coups et à leur assigner une valeur qui prend en compte les bénéfices pour le joueur et pour son adversaire. Le meilleur choix est alors celui qui minimise les pertes du joueur tout en supposant que l'adversaire cherche au contraire à les maximiser.

— Alpha-beta : C'est une amélioration de Minimax, il permet de réduire le nombre de nœuds évalués par l'algorithme minimax pour une solution optimale.

2.2 Règles de production

2.2.1 Règles de production des successeurs lors de la 1er phase :

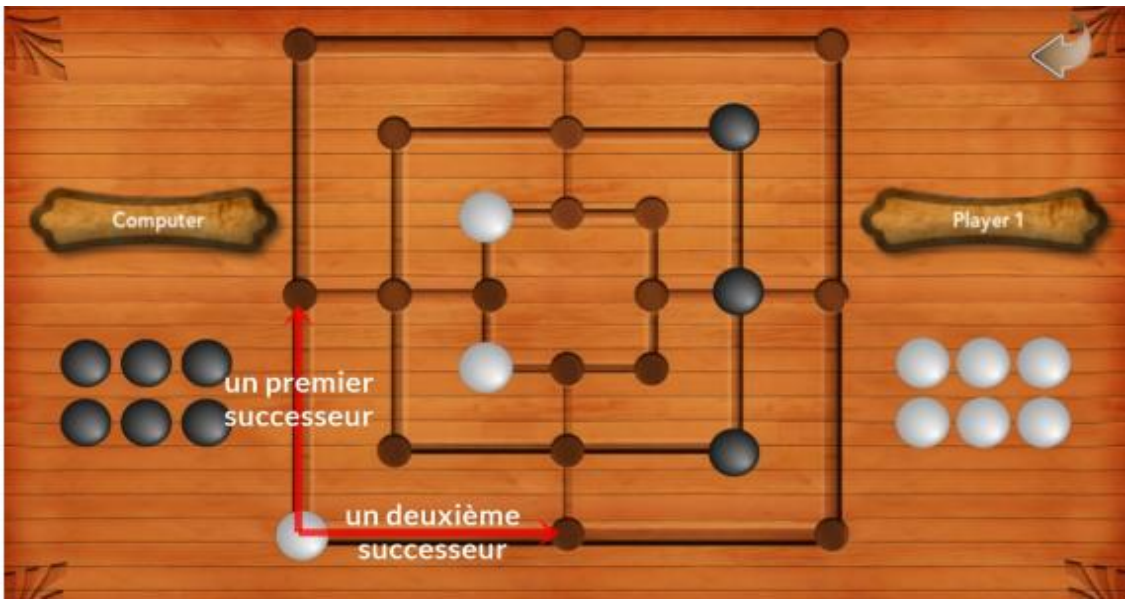
La génération des successeurs lors de la première phase du jeu se fait par la recherche des intersections qui ne contiennent pas des jetons.



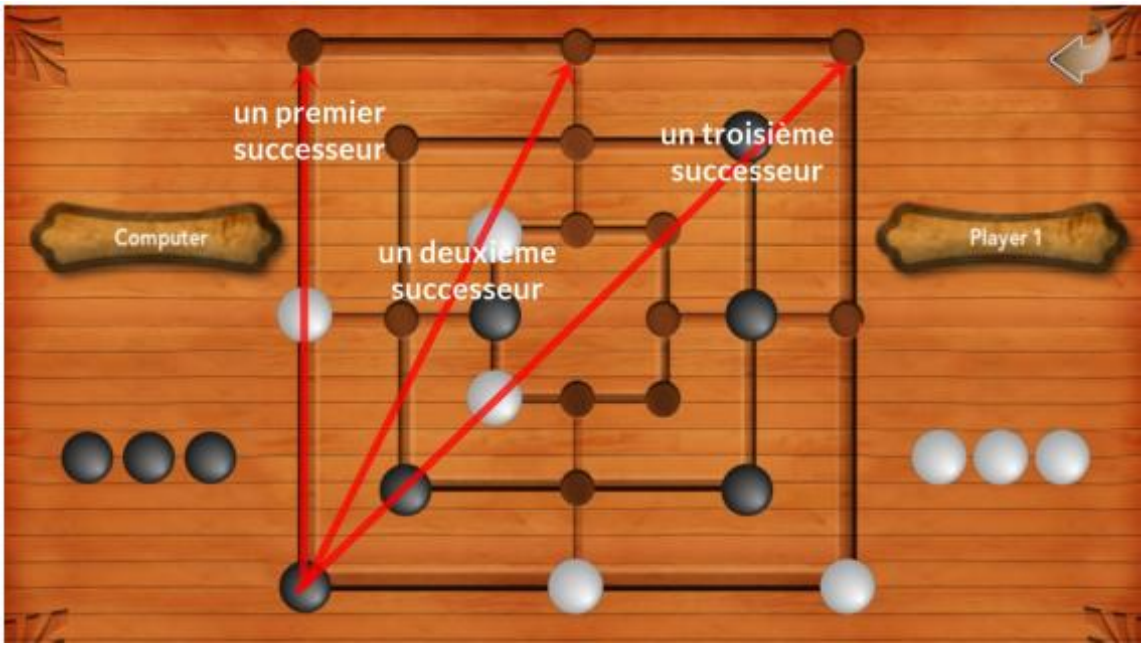
2.2.2 Règles de production des successeurs lors de la 2ième phase et la 3ième phase :

A fin de générer les successeurs de ces 2 phases à la foi on fait l'appel à une fonction :

« int Voisinages(int source,int PositionTest,etat c,int joueur) » qui retourne 0 si un joueur a une possibilité de déplacer dans la grille et -1 sinon, en vérifiant si la position test est un voisin possible pour déplacer la position source, et si l'un des joueurs a le nombre de jetons sur table égal à 3, on laisse le déplacer ou il veut dans le plateau.



2ième Phase



3ième Phase

2.3 La fonction heuristique

On cherche toujours à maximiser le cout du joueur est à minimiser celui de l'adversaire et de choisir état optimal qui donne plus de chance de moulins c'est pour cette raison on a définit la fonction heuristique (h(etat)) comme suit :

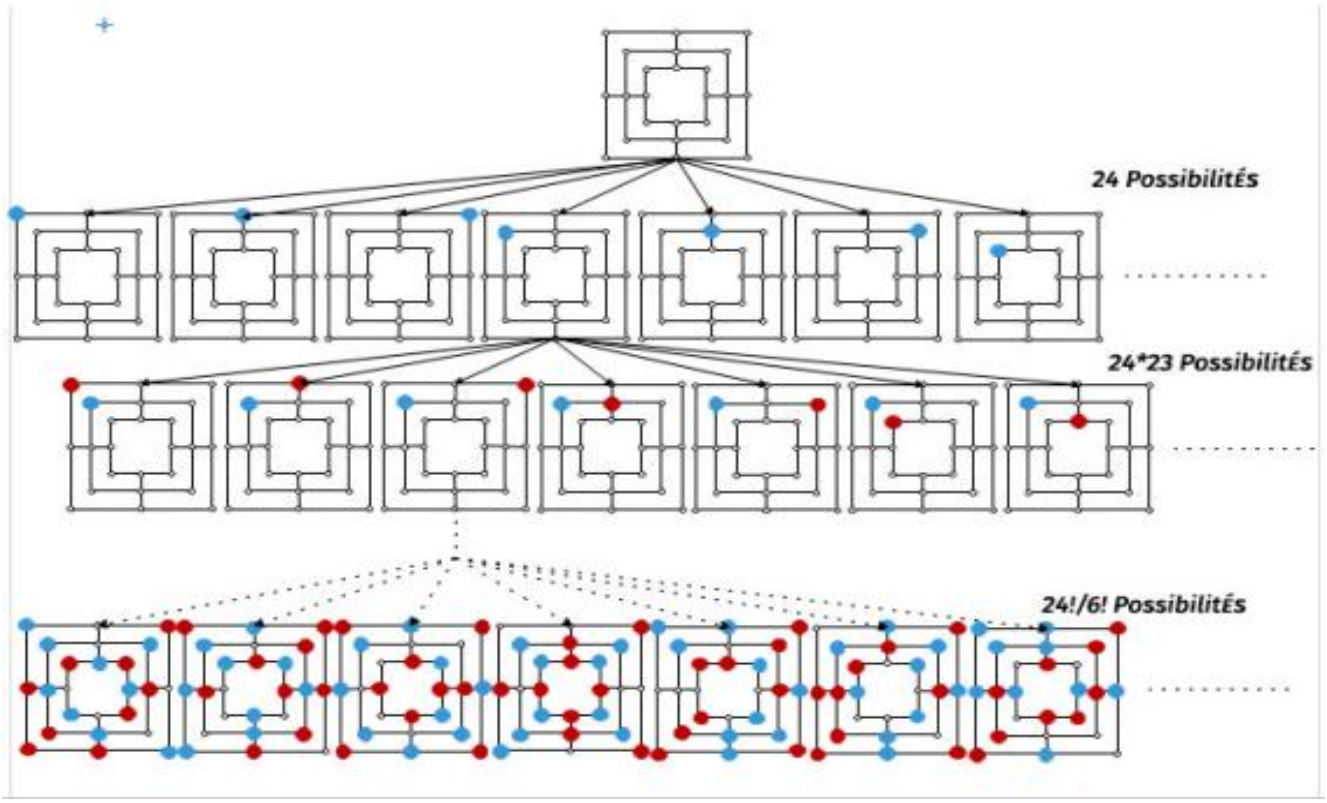
```
///la fonction heuristique
etat h(etat c)
{
    int nb1=0,nb2=0;
    for(int i=0; i<24 ; i++)
    {
        if(c.P[i].joueur==1)nb1++;
        if(c.P[i].joueur==2)nb2++;
        if(c.P[i].joueur==0&&moulin_fnc(c.P,i,1)==0)nb1+=3;
        if(c.P[i].joueur==0&&moulin_fnc(c.P,i,2)==0)nb2+=3;
        if(moulin_fnc(c.P,i,1)==0)nb1+=10;
        if(moulin_fnc(c.P,i,2)==0)nb2+=10;
    }
    c.ce=nb1-nb2;
    return c;
}
```

Pour chaque joueur on calcule (nb1, nb2) :

- le nombre de pion qui l'a sur le plateau
- le nombre de moulin qui n'est pas complet d'un joueur dans la position entrée c.-à-d. qui a besoin d'un pion pour construire un moulin.
- le nombre de moulin construit dans telle position donnée.

A la suite on calcule la différence de ces 2 nombre, et on affecte la valeur trouvée au cout de l'état entré en paramètre.

2.4 Graphe de résolution



Un arbre de résolution très vaste, on ne peut qu'utiliser un algorithme comme Minimax ou Alpha-beta qui va nous permettre d'élaguer certaines branches à l'aide d'une fonction heuristique.

3

3 Implémentation du code

3.1 Conception du code de la version Console :

3.1.1 Fonctions et stratégies implémentées :

On commencé par définir les structures dont on a besoin :

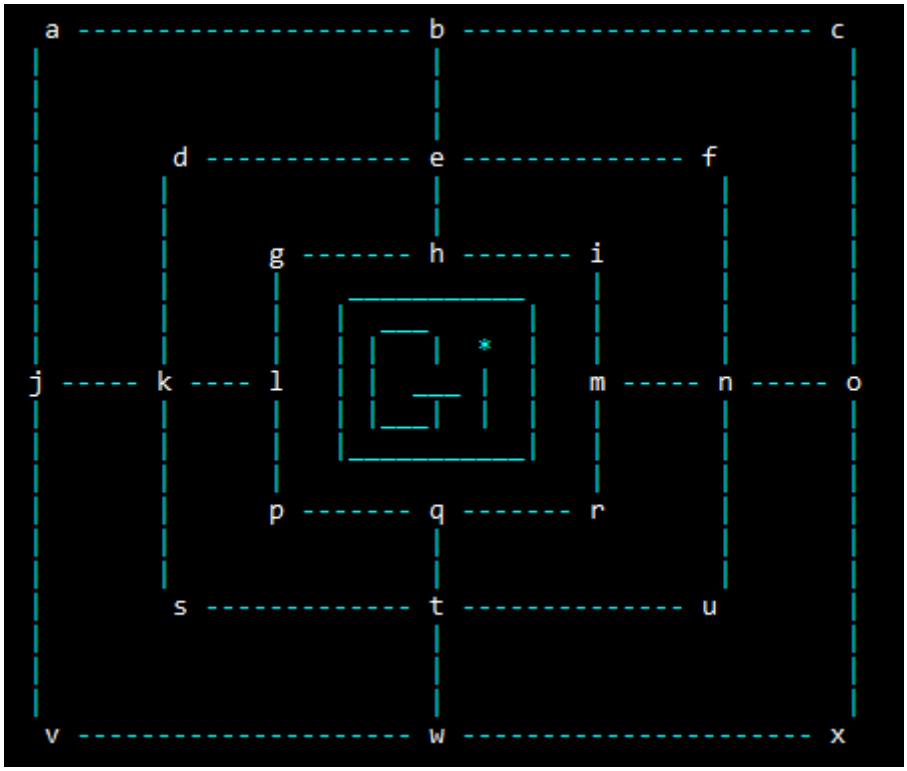
```
typedef struct place {
    char lettre;///la lettre de la case
    int joueur;///le joueur concerné par cette case
    int couleur;///la couleur de la case utilisé dans la versio console
    char v1,v2,v3,v4;///voisins de chaque position sur la grille
}place;
typedef struct etat {
    place P[24];///le plateau du jeu
    int pos1; ///nombre de pion sur table pour machine
    int pos2;///nombre de pion sur table pour joueur
    int ce; ///cout de chaque etat
    int pos;///position de chaque pion
    int presd;///le pion precedent de chaque etat
    int pC;///pion capture de l'adversaire

}etat;
typedef struct Noeud
{
    struct Noeud *suivant;
    etat Etat;
}Noeud;
```

Puis on définit les fonctions suivantes :

- « void Color(int flags) » et « void color(int BackC, int ForgC) » pour utiliser les couleurs, en incluant la bibliothèque <windows.h>.

- « void afficher(etat e) » : construit et affiche la grille comme suite :

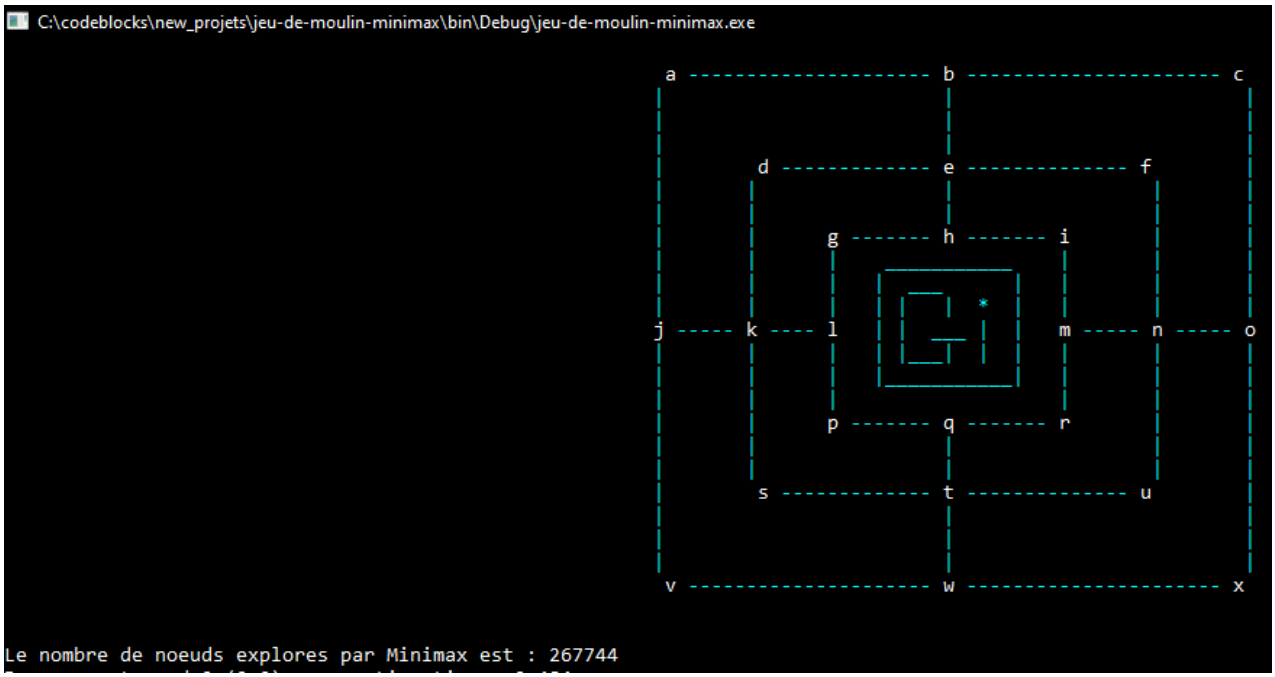


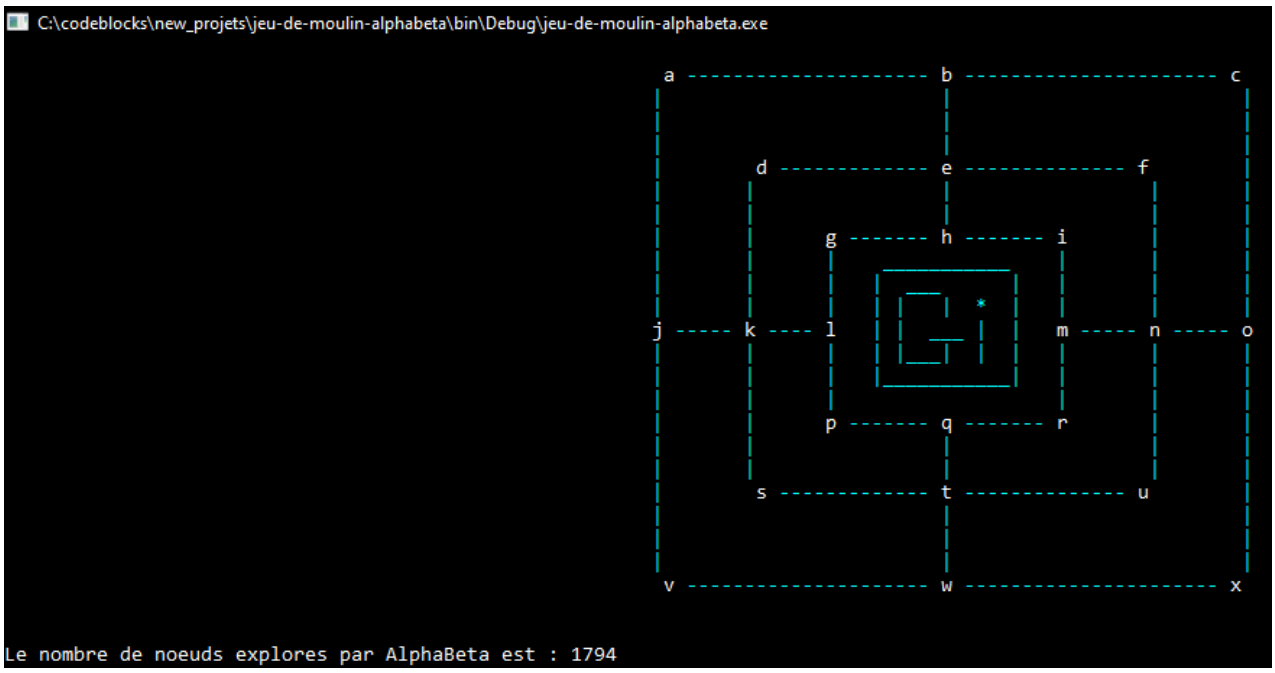
- « int moulin_fnc(place *UnPlateau,int position ,int joueur) » : verifie s'il y a un moulin dans une position et aussi si un pion appartient à un moulin.
- « int Voisinages(int source,int PositionTest,etat c,int joueur) » : retourne 0 si un joueur a une possibilité de déplacer dans la grille et -1 sinon, en vérifiant si la position test est un voisin possible pour déplacer la position source.
- « int indice_pion(char c ,etat e) » : trouver l'indice d'un pion sur la grille.
- « int pas_de_mouvement1(etat c) » : vérifie s'il n'y a pas de mouvement pour Max(Machine dans la version graphique).
- « int pas_de_mouvement2(etat c) » : vérifie s'il n'y a pas de mouvement pour Min(Utilisateur dans la version graphique).
- « void copier(etat source, etat *copie) » : copier un etat dans un autre etat, en copiant tout les champs.

- « etat h(etat c) » : la fonction heuristique qui s'occupe du calcul du score de chaque joueur.
- « etat Minimax(etat e, int p, int pions_a_placer, bool TourMax) » : en utilisant la stratégie minimax des jeux, cette fonction l'état optimale qui peu la machine jouer.
- « etat AlphaBeta(etat e, int p, int pions_a_placer, bool TourMax ,int a, int b) » : en utilisant la stratégie minimax des jeux, cette fonction l'état optimale qui peu la machine jouer en améliorantla stratégie minimax.

3.1.2 Comparaison entre Minimax et Alpha-beta en version console:

L'algorithme minimax effectue une exploration complète de l'arbre de recherche jusqu'à un niveau donné. L'élagage alpha-beta permet d'optimiser grandement l'algorithme minimax sans en modifier le résultat. Pour cela, il ne réalise qu'une exploration partielle de l'arbre. Lors de l'exploration, il n'est pas nécessaire d'examiner les sous-arbres qui conduisent à des configurations dont la valeur ne contribuera pas au calcul du gain à la racine de l'arbre. Dit autrement, l'algorithme Alpha-beta n'évalue pas des nœuds dont on est sûr que leur qualité sera inférieure à un nœud déjà évalué. Ceci est concrétisé en affichant le nombre des noeuds explorés par chaque algorithme.





3.2 Conception du code de la version graphique:

3.2.1 Fonctions et stratégies implémentées :

On utilisé les mêmes fonctions et stratégies que la version console en ajoutant des images et des sons à notre jeu, cela implique l'inclusion de :

- SDL.h ,
- SDL2.h ,
- SDL_image.h ,
- SDL_mixer.h
- et SDL_ttf.h

3.2.2 Initialiser le SDL

- SDL_Init(SDL_INIT_EVERYTHING) : pour initialiser le SDL , si il y a des erreurs on les affiche avec SDL_GetError()
- Mix_OpenAudio : pour initialiser l'audio n si il y a des erreurs on les affiche avec SDL_GetError()
- on crée des objets avec Mix_LoadWAV pour les effets et Mix_LoadMUS pour la musique (audio continu)

3.2.3 La création de la fenêtre

- SDL_CreateWindow : on crée la fenêtre avec le nom « Jeu de Moulin » et les dimensions 800*600

3.2.4 Choix d'utilisateur

- on crée un événement avec SDL_Event
- SDL_MOUSEBUTTONDOWN puis SDL_BUTTON_LEFT : on clique avec le bouton gauche de la souris, cela est enregistré dans notre événement avec une information sur les coordonnées (x,y) de la fenêtre en pixels .
- puis selon les informations des coordonnées on peut savoir le choix de l'utilisateur :

```
if(event.button.x>=369&&event.button.y>=15&&event.button.x<=
369+45&&event.button.y<=15+45)choix=1;
```

le carré noir dont on clique a des coordonnées (x,y) est selon la valeur de « choix » on ajoute l'image de pion de joueur qui arrive son tour :

```
case 1: destrect.x=369;destrect.y=15;break;
```

3.2.5 Ajouter des images : afficher(etat e) ;

```
if(e.P[i].joueur == 1){
    srcsurface=IMG_Load("src/images/pion1.png");
….}
```

3.2.6 Ajouter le son

```
Mix_PlayChannel(-1,effect5,0);
```

4

4 Conclusion :

Ce projet est une très belle expérience vu que c'est le premier contact qu'on vient d'avoir avec le domaine de l'intelligence artificiel, il nous a permet de mettre en œuvre les connaissances qu'on a acquis dans cet élément de module pour réaliser un vrai projet qui contient une vrai intelligence artificiel.