

TABLE DES MATIERES

I - Introduction …………………………………………………………………………………3

# II – Modélisation du problème …………………………………………………………………………………4

# III – Implémentation du code ………………………………………………………………………………...9

# IV – Conclusion ………………………………………………………………………………...14

I – INTRODUCTION

* **Contexte**

Ce projet s’inscrit dans le cadre du module « résolution de problèmes ». L’objectif est de programmer le jeu de stratégie de Moulin en langage C/C++, et d’implémenter une intelligence artificielle IA sophistiquée en utilisant nos connaissances acquises pendant ce module.

* **Présentation de jeu**

Le jeu de Moulin est un jeu de société à deux joueurs. Entre les deux joueurs se trouve un tableau formé par quatre carrés concentriques reliées au centre de leurs quatre côtés par des lignes perpendiculaires. Chaque joueur, lors de son tour, doit mettre un de ses pions dans une intersection. 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.

* **Déroulement du jeu**

La partie se déroule sur un plateau à 24 points.Chacun des joueurs a 9 pions.  
  
Chacun des joueurs dépose à chacun de ses tours un et un seul pion sur tableau.  
Il le dépose où il veut du moment que la case est vide. Tant qu’il a des pions à poser il ne peut en déplacer aucun.

Une fois que tous les pions sont déposés il peut (et il doit) les déplacer sur les lignes pour autant qu’elles ne soient pas obstruées par un autre pion.

Si 3 pions d’une couleur forment une ligne, il y a un « moulin » .  
Au moment où un joueur forme un moulin, il DOIT prendre un pion adverse quelconque et de son choix, mais il ne peut pas prendre un pion qui fait partie d’un moulin (sauf si ce sont les 3 derniers pions qui restent à l’adversaire, ou si tous ses pions sont dans des moulins ) .

* **Travail réalisé**

**Version Console/SDL :**

Dans un premier temps, nous présentant la version console et graphique. Ensuite, on va présenter les différents algorithmes, structures et fonctions utilisés pour rependre au travail demandé tout en décrivant leurs rôles et leurs fonctionnements globales.

* **Outils et bibliothèque exploitée**

SDL :

Il nous offre plusieurs avantage, parmi :

- Permet de créer, d’un même code source, une interface graphique multiplateforme.

- Gestion plus stricte de la mémoire.

- Initialisation des données C++.

II – Modélisation du problème

* **Structuration du problème**

• Espaces d’états

Une Partie est représentée par la classe Player, qui contient comme fonction membre «void Color(int couleurDuTexte, int couleurDeFond)» qui aide à représenter la couleur de chaque pion(Rouge : Max, Bleu : Min). Un joueur peut être humain ou encore une intelligence artificielle IA.

**Remarque :** Vu qu’on s’intéresse à maximiser le score(Max), l’intelligence artificielle IA(CPU) sera représentée par le joueur Max qui commence toujours le jeu.

Le tableau de jeu est illustré par une fonction void AfficheTableau() qui contient un tableau de caractères : tab[37][80] **à 2 dimensions. Pour avoir l’état d’une case, il suffit de son numéro de ligne et de colonne comme indice et la tableau retournera l’information recherché. La modélisation a été fait comme ceci : Pour 0≤i≤80 et 0≤j≤37 :**

**si tab[i] [j]= ‘M’ alors la case (i, j) est remplie par un pion rouge (Max).**

**si tab[i] [j]= ‘m’ alors la case (i, j) est remplie par un pion bleue (Min).**

**si tab[i] [j] != ‘m’ ou ‘M’ ou ‘\272’ ou ‘\315’ alors la case ( i, j) est vide**

Remarque ***:*** Vu **que le tableau de Moulin ressemble à une matrice, c’est mieux de le représenter par un tableau de deux dimensions.**

• Objectif

Le but général du jeu est de parvenir à ce que l’opposant ait seulement 2 pions sur le tableau ou qu’il lui soit impossible de réaliser aucun mouvement ayant toutes ses pièces bloquées.

Pour ce, le but immédiat est d’essayer de faire des Moulins : Trois Pions en ligne) avec les propres pions, ce qui donne le droit de capturer un pion de l’adversaire.

• Etat initial

Dans l’état initial, le jeu commence avec le tableau vide. A la première phase chaque joueur à son tour place un de ses neuf pions dans un point libre quelconque du tableau. Pendant cette phase, il est interdit de déplacer les pions déjà placés sur le tableau.

• Etat final

Elle correspond à un état où l’un des deux joueurs gagne, et ce est dans deux cas :

- Quand il a réalisé 7 prises, et par conséquent son adversaire a seulement deux pions et ne plus former de moulins.

- Quand l’adversaire ne peut plus réaliser aucun mouvement parce qu’il a tous ses pions bloqués.

• Règles

On ne peut poser des pions que si le point est vide.

Rl1 : Positionner un pion sur la première ligne si elle est non pleine.

Rl2 : Positionner un pion sur la 2ème ligne si elle est non pleine.

Rl3 : Positionner un pion sur la 3ème ligne si elle est non pleine.

Rl4 : Positionner un pion sur la 4ème ligne si elle est non pleine.

Rl5 : Positionner un pion sur la 5ème ligne si elle est non pleine.

Rl6 : Positionner un pion sur la 6ème ligne si elle est non pleine.

Rl7 : Positionner un pion sur la 7ème ligne si elle est non pleine.

De même pour la colonne (Rc1...Rc7).

On représente alors un mouvement(ou position i) par le couple(Rli,Rci).

• Stratégie

La stratégie MinMax :

- Prévoir à l’avance les mouvements de l’adversaire.

- Prévoir un espace de recherche plus grand.

- Chercher le meilleur mouvement possible.

La stratégie AlphaBeta :

- Améliorer l’algorithme MinMax.

- Élaguer certaines branches explorées inutilement.

La fonction utilité :

- Max place les « M » alors que Min place les « m ». Pour un nœud :

- Si Max gagne, le score est de +560 +le nombre des pions pris-le nombre de ses pions pris par Min.

- Si Max perde, le score est de –560…

-Si Match nul, le score est de 0

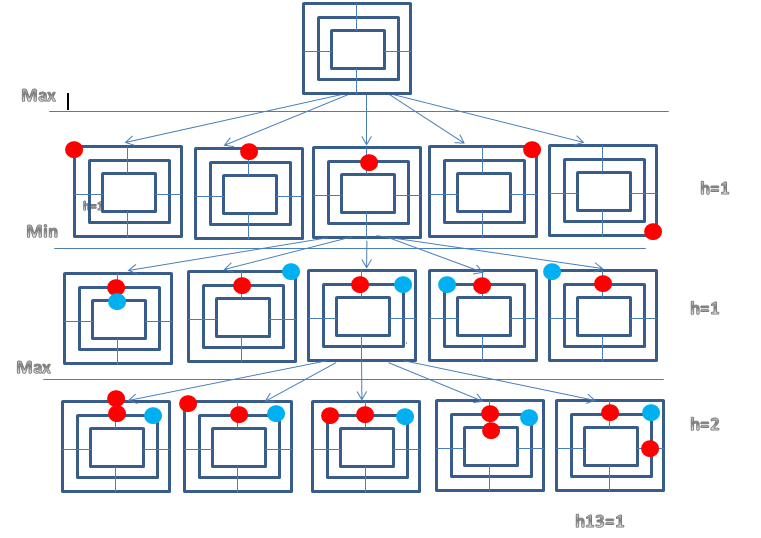
La fonction heuristique :

Et voici la fonction heuristique de Max pour chaque pion positionné dans un point (sur un ligne), on évalue selon le principe suivant :

* le nombre de pion de max sur la ligne.
* 0 si il existe un pion de min sur la ligne.

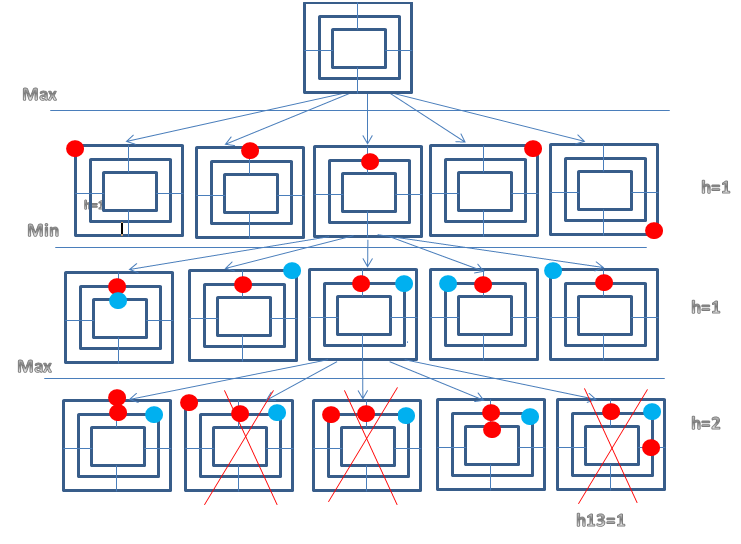
De même pour la colonne.

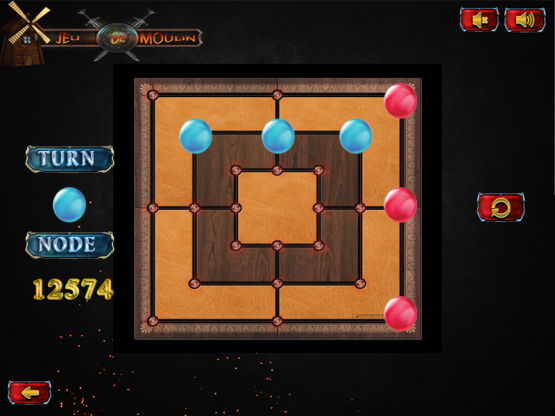
* **Graphe de résolution MiniMax**

Le nombre des nœuds explorés 56 nœuds.

* **Graphe de résolution AlphaBeta**

Le nombre des nœuds explorés 38 nœuds.



Voici la comparaison ente l’algorithme MinMax et Alpha-Beta en termes de nœuds explorées :

L’algorithme MinMax :

L’algorithme Alpha-Beta :



III – IMPLEMENTATION DU CODE

* **Structuration du problème**

• Espaces d’états

void CreateTable(): Dessiner le plateau vide.

void Menu(): Afficher le menu initial.

void MenuLevel() : Afficher le menu des niveaux du jeu.

void AfficherTableau() : Mettre le plateau à jour à chaque modification.

void lirePosition() : Récuperer la position du pion entrée par le joueur(min) et vérifie sa validité.

void lirePositionDeplacer() : Récupérer la position du pion à déplacer après la réctification de validité et demander par la suite le sens de déplacement(haut,bas,gauche,adroite).

void PositionDeplacer() : Stocker la position du pion déjà recupérée et faire appel à la fonction que se charge du déplacement.

void RemplirTableauMax() : Echanger le caractère du plateau vide par le caractère ‘M’ et faire appel à la fonction de coloriage.

void RemplirTableauMin() : Echanger le caractère du plateau vide par le caractère ‘m’ et faire appel à la fonction de coloriage.

void RandomPosition() : Choisir une position aléatoire du pion qui est valide(dans le niveau facile).

void RandomDeplacer() : Choisir un sens de déplacement aléatoire et valide.

void RandomDeplacerPionAdjacent(char pos,int i,int j) : Déplacer le pion choisi aléatoirement.

void DeplacerPionAdjacent(char pos,int i,int j) : Déplacer le pion choisi dans le sens convenable .

void DeplacementVulnerable() : Vérifier si un pion appartenant un moulin se déplace pour qu’il devient capturable par l’adversaire.

void Moulin() : Chercher les moulins et se charger de la capture du pion à retirer du plateau.

void CreateAvoidMoulin() : Chercher les futurs moulins de max et positionne un pion dans la position vide pour maximiser les moulins , et parallèlement empêcher min à construire des moulins.

void RetirerAdversaire(int a,int b) : capturer un pion de position(i,j).

bool AncienMoulinColonne(int i) : Stocker l’indice de ligne où un moulin est construit afin d’empêcher la captures de ces pions appartenant au moulin.

bool AncienMoulinLigne(int i) : Stocker l’indice de colonne où un moulin est construit afin d’empêcher la captures de ces pions appartenant au moulin.

void CompteurPion() : Compter le nombre de pions de max cM et min cm.

int Vide(Liste listeNoeuds) : Vérifier si une liste est vide en renvoyant 0 et 1 sinon.

void extraire(Etat\* e, Liste listeNoeuds) : Extraire l’état e de la sommet de liste des nœuds.

void Inserer(Etat actuel,Liste\* listeNoeuds) : Insérer l’état e dans la sommet de liste des nœuds.

void hLigne(Etat e1,int i,int j) : Calculer la fonction heuristique de ligne de chaque pion de max en choisisant le max du nombre de pions rouges situés dans la même ligne que le nœud et en stockant la position optimale dans laquelle max doit mettre son prochain pion afin de compléter/acheminer un moulin.

void hColonne(Etat e1,int i,int j) : le même fonctionnement de la fonction heuristique de ligne appliqué pour la colonne.

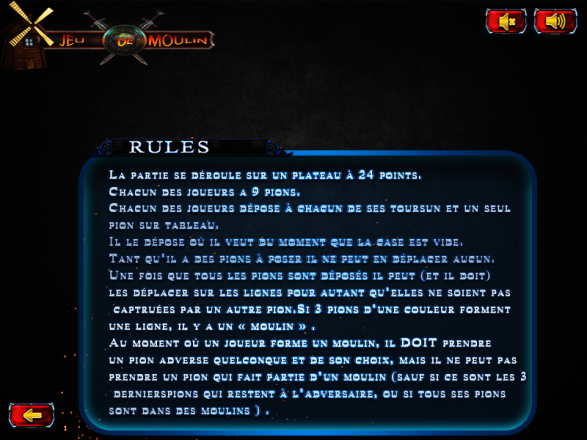
void RandomPositionDeffenseAttaque() : Vérifier s’il y a un moulin à compléter ou à empêcher avant de choisir une position aléatoire.

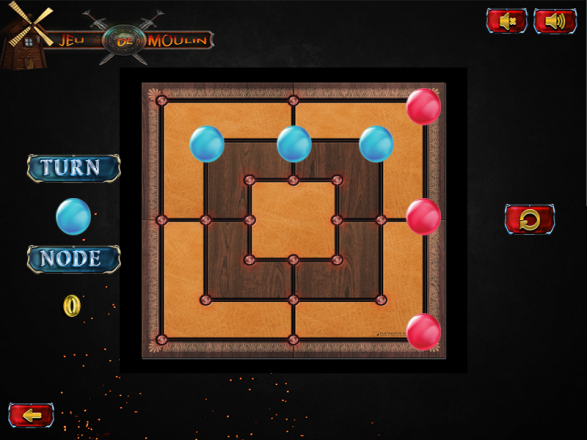
* **Simulation du programme**



L’exécution du programme est la suivante :







VI – CONCLUSION

Ce projet fut une très bonne expérience, vu qu’il a testé nos connaissances en résolution des problèmes acquises dans le 2eme semestre. On a également pu améliorer nos compétences en modélisation et développement des différents algorithmes de recherche. Cela nous a permet de construire une AI que nous même ne pouvons pas vaincre.