# RAPPORT PRROJET:

# JEU MUSASHI


**Réalisé par:**

- **Mostafa Sissi**
- **Ilham Hafid**

**Encadré par:**
**Mme ADDOU**

# SOMMAIRE

# INTRODUCTION

Le mot "résolution" est à prendre dans son sens large. Il s'agit de poser, d'analyser et de représenter des situations concrètes, tout autant que de les résoudre.  L'un des vastes champs dont il intervient l'IA en appliquant différentes méthodes rationnelles de prises de décision est les jeux opposant un adversaire (Humain) à la machine .Et parmi les stratégies utilisées est celle du Minimax, du Alpha Beta...

Dans notre cas, nous intéressons au jeu du "Musashi". Ce dernier est un jeu combinatoire qui oppose deux adversaires qui placent à tour de rôle des pions dans une grille composée de 33 cases, le joueur "Musashi" possède un seul pion et il tente de gagner les 16 policiers.

Nous allons donc implémenter en C++ ce jeu dans deux versions différentes, à savoir la version graphique et la version console, utilisant les stratégies du minimax et de l'alpha beta.

# 1. MODÉLISATION DU PROBLÈME :

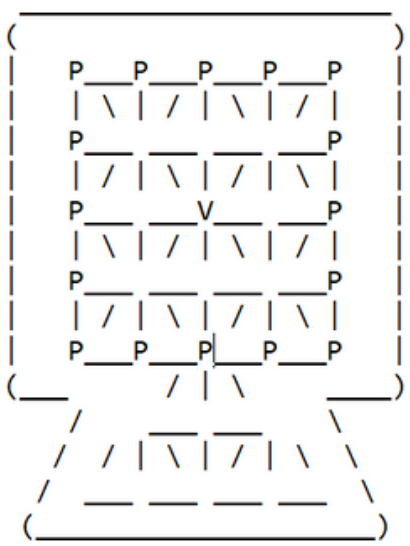
Le jeu "Musashi" se joue dans une grille de taille 7*5. Nous avons deux types de pions, un pour la machine ('P': policier) et un autre pour l'utilisateur ('V': voleur).

- **Espaces d'état :**

**La grille du jeu :** elle s'agit d'une matrice de taille 7x5 gérée par la classe "Plateau".
**Le tour :** il s'agit d'un caractère qui prend la valeur 'V' dans le tour du musashi; et la valeur 'P' dans le tour des policiers( de la machine).
**Le 'h' :** il s'agit d'un entier retourner par la fonction heuristique définie dans notre cas.
**La 'profondeur' :** elle reflète la profondeur choisie pour les stratégies de la recherche 'minimax et alphabeta'.

- **Etat initial VS états finaux :**

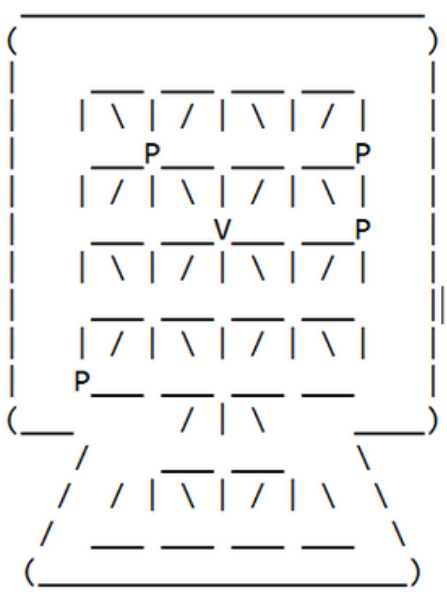
**L'état initial :** est représenté par la grille avec le "musashi" centré au milieu du carré haut (matrice 5x5), les policiers situés dans la frontière de la même matrice et les autres cases sont initialisées par le 'vide'.



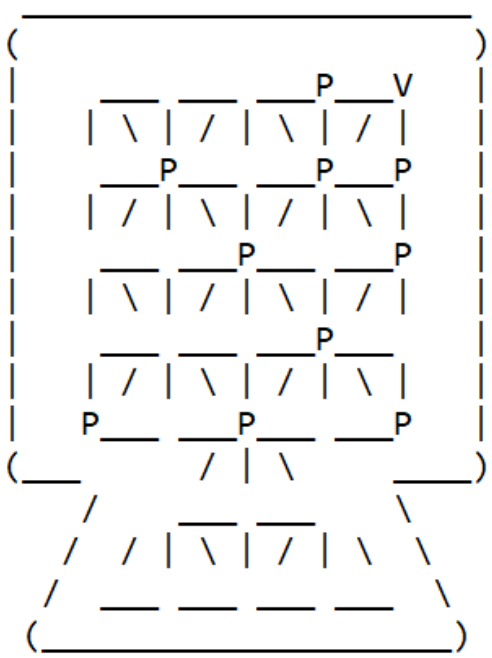
**Les états finaux :**

Il existe deux types des états finaux à savoir le cas où:

-Le 'musashi' a gagné: caractérisé par l'existence dans la grille d'au plus quatre policiers.



-La machine a gagné:
caractérisé par le confinement du 'musashi' par les policiers (aucun déplacement du 'musashi' est possible).



- **Règles de productions :**

Le jeu se joue à tour de rôle et les règles de productions sont :

-Déplacer vers le haut;
-Déplacer vers le bas;
-Déplacer vers la gauche;
-Déplacer vers la droite;
-Déplacer vers le haut droit;
-Déplacer vers le haut gauche;
-Déplacer vers le bas droit;
-Déplacer vers le bas gauche;

**Condition d'ajout :** La case de destination doit être "VIDE"  c'est-à-dire que dans plateau[i][j] ne devrait contenir ni policier ni 'Musashi' et la direction du déplacement doit être possible .

- **La fonction heuristique h :**

Dans notre réalisation, la fonction d'évaluation des nœuds, est une méthode de la classe "Minimax  qui prend en paramètre une configuration de la grille et qui lui attribue une valeur entière qui reflète le nombre du vide entouré par le 'Musashi'.

Le code source :

```
// la fonction heuristique : sert a calculer le cout d'un etat
int minimax::heuristique(plateau& P){
    // renvoie le nombre de vide voisin
        int nbrDevide = 0;
        for(int i=0;i<7;i++){
            for(int j=0;j<5;j++){
                if(P.monPlateau[i][j]==voleur){
                        for (int dir=0;dir<8;dir++){
                            deplacement d ;
                            d.posi.x = i;
                            d.posi.y=j;
                            d.direction=dir;
                            d.posf = d.positionSuivant();
                            if(d.verifierMvt() && d.estVide(P)) nbrDevide++;}
                                                }
                            }

                                }
        return nbrDevide;
}
```

- **Utilisation de la fonction heuristique dans le programme :**

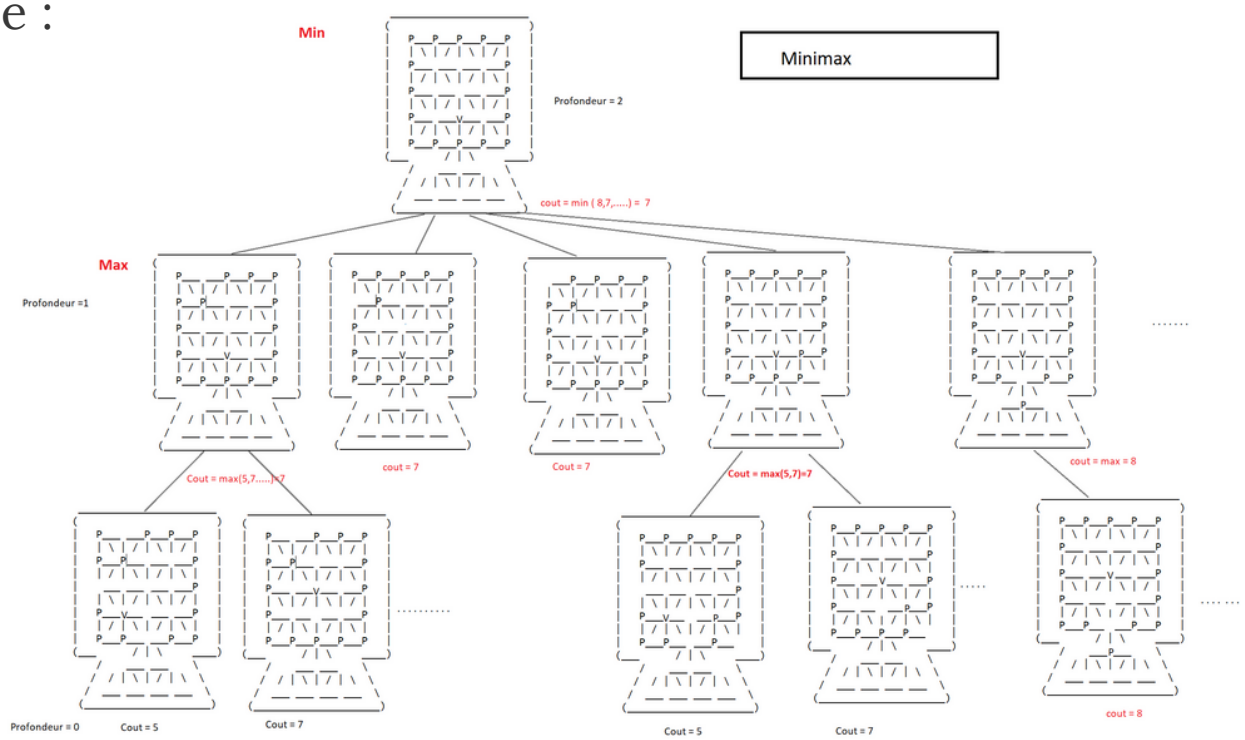
La fonction heuristique est utilisée essentiellement dans les fonctions "minimax" et "alphabeta"
et permet d'avoir la valeur du coup si la profondeur à laquelle est appelée la fonction "minimax" ou "alphabeta" est nulle ou si on se retrouve à un état final.

- **Représentation graphique de l'arbre de recherche pour la stratégie MiniMax:**

La stratégie du minimax a été conçue afin d'intervenir dans des jeux opposant deux types de joueurs :

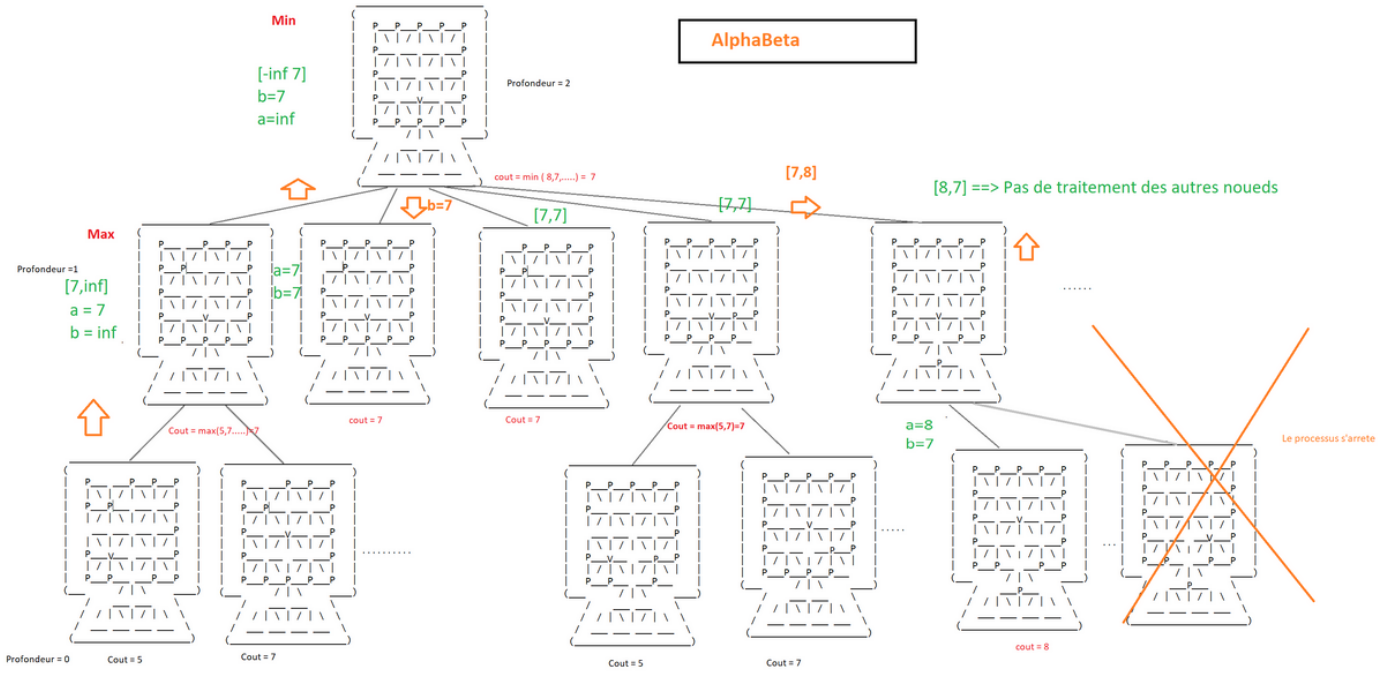
1. Le **Min** : qui essaie de minimiser les coups du max, il représente la machine.
2. Le **Max** : qui essaie de maximiser ses coups dans le but de gagner, il représente le joueur.

Exemple :

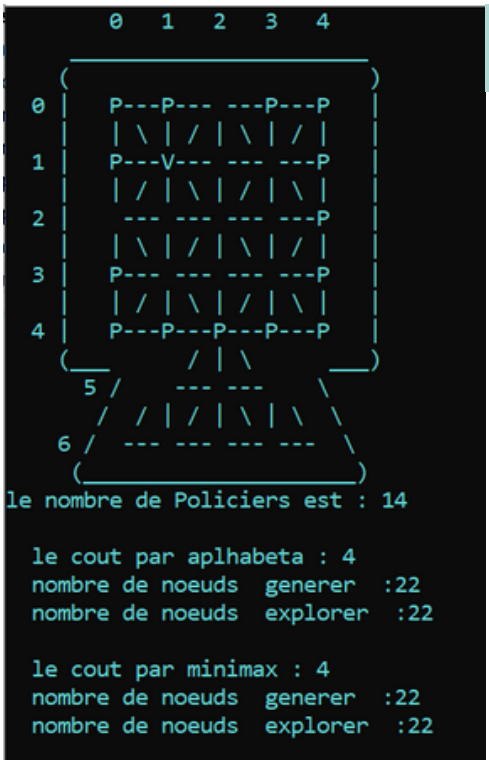


- **Représentation graphique de l'arbre de recherche pour la stratégie Alphabeta:**

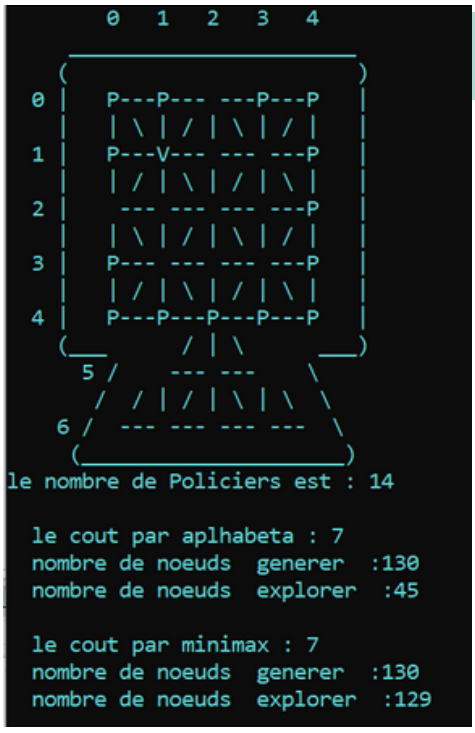
La stratégie alphabeta est une amélioration de la stratégie minimax. Elle permet de parcourir moins de nœuds et est par conséquent plus optimale par rapport au minimax.



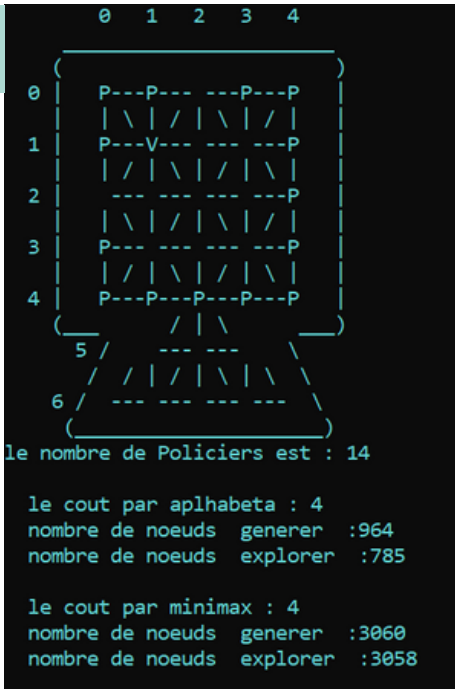
P = 1




P = 2

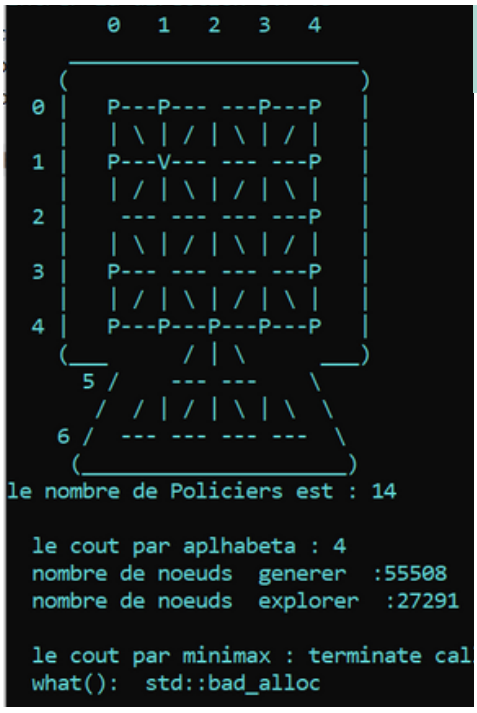



P = 3

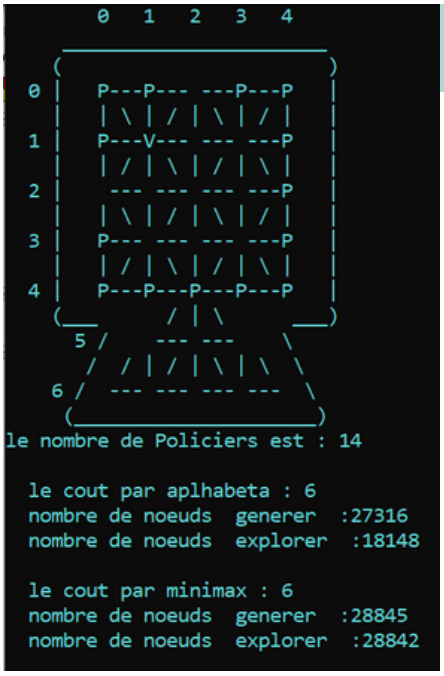



# COMPARAISON ENTRE LE MINIMAX ET L'ALPHABETA

P = 5




P = 4



- **Minimax VS Alphabeta:**

Comme on remarque dans la figure précédente, le programme alphaBeta est optimal par rapport au programme minimax.
La profondeur qu'on peut atteindre par le programme alphaBeta est p = 6, alors que pour le programme minimax peut atteindre la profondeur  p=5.

# 2.IMPLÉMENTATION DU CODE :

- **Les classes utilisées :**

1. Class Interface:

**Rôle:** Gestion d'affichage de l'entrée du jeu, du menu principal et du menu des trois niveaux (easy, medium, hard).

Elle se compose de ces méthodes suivantes:

- **setChoixMenu()** : responsable de la lecture du choix du joueur parmi le menu principal.

-**affiche()** : responsable de la lecture des fichiers du design d'écriture.

-**setChoixLevel()** : responsable de la lecture du choix du joueur parmi le menu des niveaux.

-**afficheInterface()** : responsable de la combinaison des deux méthodes précédentes.

-**getChoixLevel()** : getter du choix du joueur.

2. Class Plateau:

**Rôle:** Gestion des différents traitements relatifs directement à la grille du Jeu tels que :

l'affichage du plateau de jeu avec ses pions avec la méthode **afficheToi()**.

l'initialisation de la grille c'est-à-dire, mettre une grille à l'état initial (rôle du constructeur **plateau()**).

déplacer le pion choisi pour chaque joueur avec la méthode **updatePlateau()**.

vider ou manger les policiers s' il est possible avec la méthode **mongePion()**.

vérifier si le voleur "Musashi" a la possibilité d'effectuer un déplacement avec capture des policiers avec la méthode **capturePossible()**.

vérifier si le jeu a atteint sa fin avec la méthode **gameOver()**.

vérifier si le voleur n'a aucune autre possibilité de se déplacer avec la méthode **etatfinale()**.

donner la position actuelle du musashi avec la méthode **positionDuVoleur()**.

3.Class Deplacement:

**Rôle:** Gestion des déplacements effectués :

calcul de la position suivante à partir de la direction entrée par le joueur.

NOTE: On a crée une énumération '**direction**' qui contient les directions possibles (8 au maximum). De plus on a crée une structure '**position**' dans laquelle on stocke les coordonnées (x, y).

vérifier les mouvements, si le pion va effectuer un déplacement vers les places voisines par la méthode **verifierMvt().**

vérifier si la direction souhaitée est vide par la méthode **estVide().**

4.Class Player:

**Rôle:** Gestion des joueurs.

changer le tour des joueurs avec la méthode **update().**

afficher le tour correspond au joueur avec la méthode **afficheTour().**

5.Class MiniMax:

**Rôle:** Gestion de l'intelligence par la stratégie minimax.

calcul de la fonction heuristique (nombre de vide entourés par le musashi) avec la méthode **heuristique().**

générer les successeurs d'un état avec la méthode **genersuccesseur()**.

calculer le coup d'un nœud parmi la liste 'listeNœuds' avec la méthode **Minimax()**.

retourner l'état convenable par la méthode **joueurMinimax().**

gérer la liste '**listeNoeuds**' soit en terme d'extraction par la méthode **extraire()** soit en la vidant avec la méthode **viderListeNoeud().**

6.Class AlphaBeta:

**Rôle:** Gestion de l'intelligence par la stratégie alphabeta.

D'une manière équivalente de celle de la classe MiniMax, on crée la classe AlphaBeta, la seule chose qui se défère est le principe implémenter dans la méthode alphaBeta.

7.Class SDL:

**Rôle:** Gestion de la version graphique du jeu.

gestion des erreurs de différents éléments de la SDL avec la méthode **SDL_ExitWithError().**

dessiner les images importées avec la méthode **DessinerForme().**

savoir l'emplacement des voleurs par pixels à partir du tableau du travail avec la méthode **dessinerVoleurs().**

savoir l'emplacement du 'Musashi' par pixels à partir du tableau du travail avec la méthode **getMusashi().**

gérer le déroulement du jeu du niveau 'easy' contre l'ordinateur avec la stratégie minimax du profondeur 2 avec la méthode **CaseMinimax_easy().**

gérer le déroulement du jeu du niveau 'meduim' contre l'ordinateur avec la stratégie minimax du profondeur 4 avec la méthode **CaseMinimax_meduim().**

gérer le déroulement du jeu du niveau 'hard' contre l'ordinateur avec la stratégie alphabeta du profondeur 5 avec la méthode **CaseAlphaBeta_hard()**.

gérer le déroulement du jeu tout entier dès son début à sa fin avec la méthode **JeuGraphique().**

- **Exécution du programme:**

1. **Version Console:**

Ouverture de notre jeu est faite de la manière suivante:

Après un menu globale s'affiche pour choisir l'option convenable parmi les suivantes:

1. Play / 2.Rules /3.Exit

Le choix de la première option envoie l'utilisateur à choisir le niveau du jeu de la manière suivante:

1. Easy / 2. Medium / 3.Hard


Ces niveaux corresponds respectivement à:

Niveau facile: Ce niveau correspond à la profondeur 2 (avec minimax).

Niveau moyen: Ce niveau correspond à la profondeur 4 (avec minimax).

Niveau difficile: Ce niveau correspond à la profondeur 5 (avec alphabeta).

Le choix de la première option envoie l'utilisateur à choisir le niveau du jeu de la manière suivante:

1. Easy / 2. Medium / 3.Hard


Ces niveaux corresponds respectivement à:

Niveau facile: Ce niveau correspond à la profondeur 2 (avec minimax).

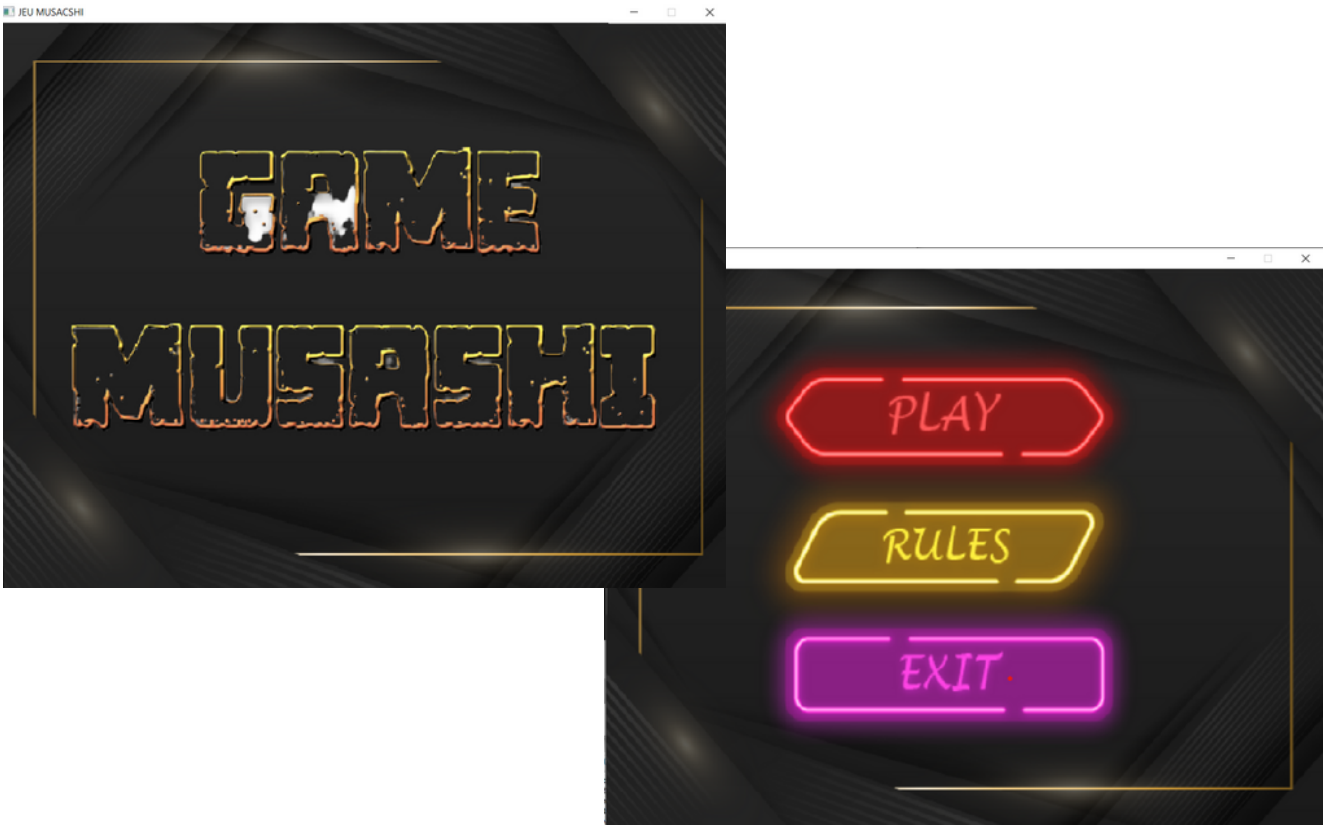
Niveau moyen: Ce niveau correspond à la profondeur 4 (avec minimax).

Niveau difficile: Ce niveau correspond à la profondeur 5 (avec alphabeta).

2.**Version Graphique:**

L'implémentation du jeu en mode graphique a été faite avec la bibliothèque SDL.

Ouverture de notre jeu est la suivante:




En cliquant sur le button "Play" on se retrouve avec l'interface suivante:

En cliquant sur le button "Rules" on se retrouve avec l'interface suivante:

# CONCLUSION

Le Jeu 'Musashi' est un jeu combinatoire qui se joue à deux. Ce projet est une expérience très intéressante vu qu'elle est le premier effectif contact avec le domaine d'intelligence artificielle, de plus est, il nous a permis d'appliquer les connaissances acquises dans cet élément de module, notamment les différentes stratégies de système de production tels que le minimax ou l'alphabeta.

Durant le progrès de notre projet on a subi comme difficultés d'une part l'optimisation des noeuds explorés. D'autre part, l'autoformation de la programmation sur une interface graphique (SDL). Mais nous avons pu surmonter ces difficultés au final.