Rapport Projet : Jeu de moulin

OUTIDRARINE Mohamed : **harrymomment@gmail.com**

EL BOUALAOUI Fairouz : **fairouzelboualaoui@gmail.com**

Encadrement : Mme. ADDOU

Table des matières

1

Introduction

1.1 Contexte

Le jeu du moulin est le projet réalisé dans le cadre du module de “résolution de problèmes”. Le but est d’implémenter une IA dans le jeu avec les stratégies Minimax et Alpha-beta à partir des outils acquis dans le module.

1.2 Présentation du jeu

Le jeu affronte à deux personnes sur un tableau 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.

1.3 Language,Outils,bibliothèques et Frameworks utilisés

- Langage de programmation utlisé : C++
- Pour la version graphie et Android : nous avons opter pour la bibliothèque graphique Cocos2d, c’est une bibliothèque libre opensource, pour realiser des jeux avec le langage C++
- Outils utilisés :
  - CodeBlocks pour la version console
  - Microsoft Viasual studio pour la version graphique

  - Google Android Studio pour la vesion
  - Github pour le partage synchroniser le travail en groupe
- La bibliothèque vectors pour stocker les successeurs générés dans Alphabeta et Minimax
- Adobe Photoshop pour la création des images graphiques utilisés dans la version graphique et la version Android

1.4 Difficultés rencontrées

Lors du réalisation de ce projet on a rencontré des difficultés qu'on a pu surmonter, dans cette partie on va citer quelques unes de ces difficultés :

- Le jeu nécessite des animation, parce que on doit réaliser des animation lorsqu'on bouge les jetons, c'est pour cela il a fallut trouver une bibliothèque graphique qui contient cette option de réaliser les animation, c'est pour cette raison qu'on opté pour la bibliothèque graphie cocos2d et pas SDL ou une autre bibliothèque
- La nature du jeu rend son implémentation un peut difficile, en effet le jeu se déroule en 3 phases, qui sont différentes en matière de génération des successeurs, effectivement les successeurs qui doivent être générés lors de la première phase, doivent être différents des successeurs générés lors de la deuxième phases, et de même pour la troisième phase, ce qui nous oblige à réaliser un algorithme de recherche minimax et alphabeta pour chaque phase du jeu
- Implémentation de l'algorithme Alpha-beta et Minimax, surtout pour la deuxième phase du jeu, justement parceque l'algorithme alphabeta ou minamx doit nous retourner deux valeurs en sortie, en effet il doit retourner la position du jeton qu'on doit bouger et la position de la destination vers laquel on doit le placer

2

Modélisation du problème

2.1 Structuration du problème

Le joueur est représenté par la classe Player contenant comme membres “int finaldepth” et “int nbr-noeuds”. Cette classe représente un joueur humain bien qu’une intelligence artificielle.
La grille du jeu est représenté par une classe Board composé de “int intersection-remplie[24]” représentant chaque intersection où on peut mettre un jeton, ,“int player” qui représente le joueur actuelle et qui a la valeur 1 par défaut, “int times”, “int noir” et “int blanc”.

Objectif :

Chaque joueur a pour objectif la réalisation du plus grand nombre de moulin possible (un moulin représente l’alignement de trois de la même couleur) afin de capturer les jetons de l’adverse. 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 state[24]=0. Les deux joueurs commencent à poser leur 9 jetons à tour de rôle.
- La deuxième phase :
  à tour de rôle, Les deux joueurs déplacent un jetons à la fois vers une intersection voisine libre.
- La troisième phase :
  Une fois qu’un joueur n’a que trois jetons, la troisième phase commence et ce dernier peut se déplacer vers n’importe quel intersection vide.

Règles du jeu :

— En cas de construction d'un moulin, on peut capturer un jeton de l'adversaire.

— On ne peut capturer que le jetons qui ne construisent pas un moulin.

— Durant la deuxième phase, on n'a droit à déplacer un jetons qu'aux intersections voisines vides.

— Les pièces capturées ne sont jamais rejouées sur le plateau et restent capturées pour le reste de la partie.

— Le jeu est terminé lorsqu'un joueur perd en étant réduit à deux pièces ou en étant incapable de bouger.

Stratégies :

On va se baser sur deux stratégies :

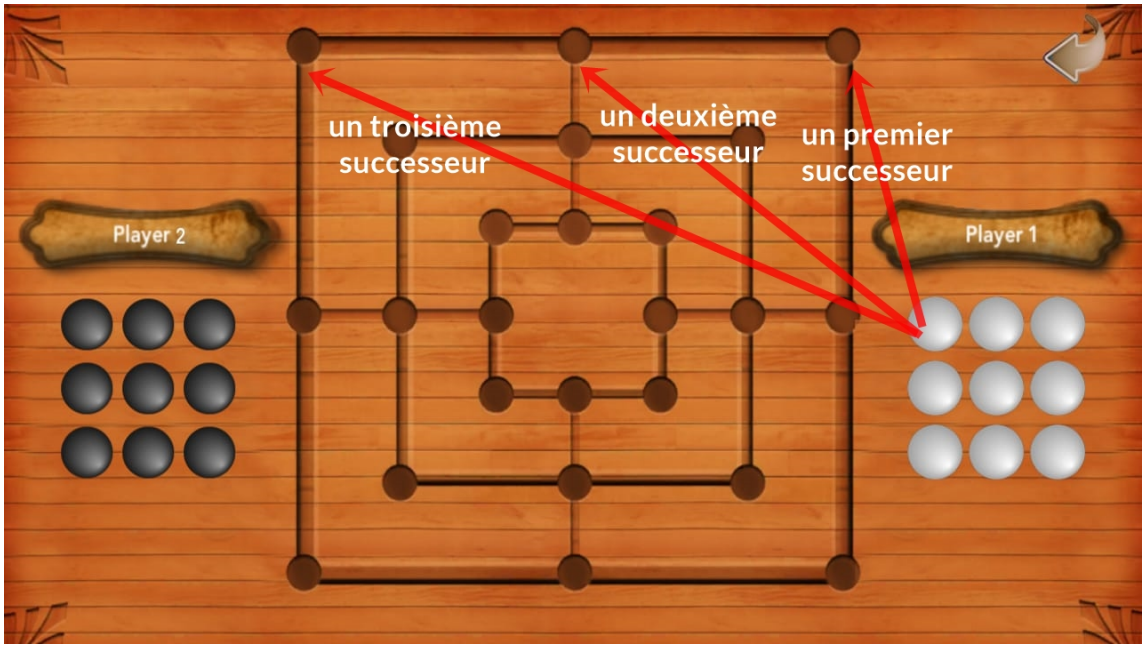
— Minimax : Il amène l'ordinateur à passer en revue toutes les possibilités pour un nombre limité de coups et à leur assigner une valeur qui prend en compte les bénéfices pour le joueur et pour son adversaire. Le meilleur choix est alors celui qui minimise les pertes du joueur tout en supposant que l'adversaire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

2.2 Règles de production

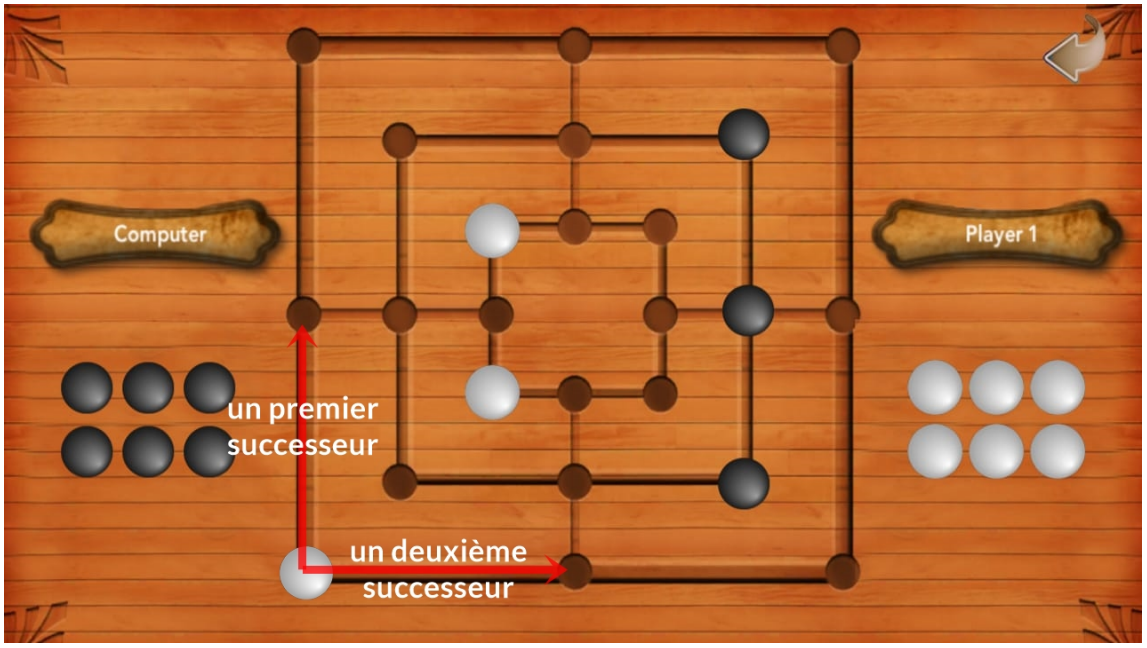
2.2.1 Production des successeurs lors de la 1ere phase :

Pour générer les successeurs lors de la première phase du jeu on cherche les intersection qui ne contient pas des jetons et on les remplie et puis on insère dans la liste des successeurs.



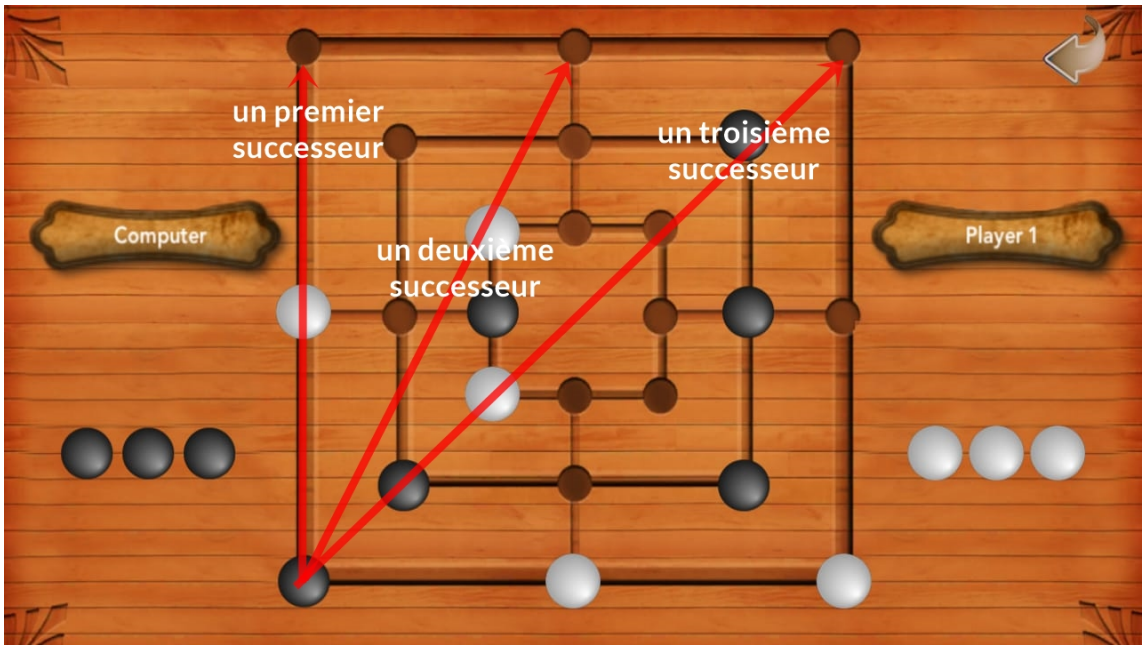
2.2.2 Production des successeurs lors de la 2eme phase :

Pour générer les successeurs lors de la deuxième phase du jeu on fait de tell façon à chercher les intersections qui contient des jetons et on appel une fonction qui nous retourne un vecteur, ce vecteur contient les voisins vide de ce jetons et puis on bouge ce jetons dans chaqu'un de ces voisins vide et au même temps on insère dans la liste des successeurs.

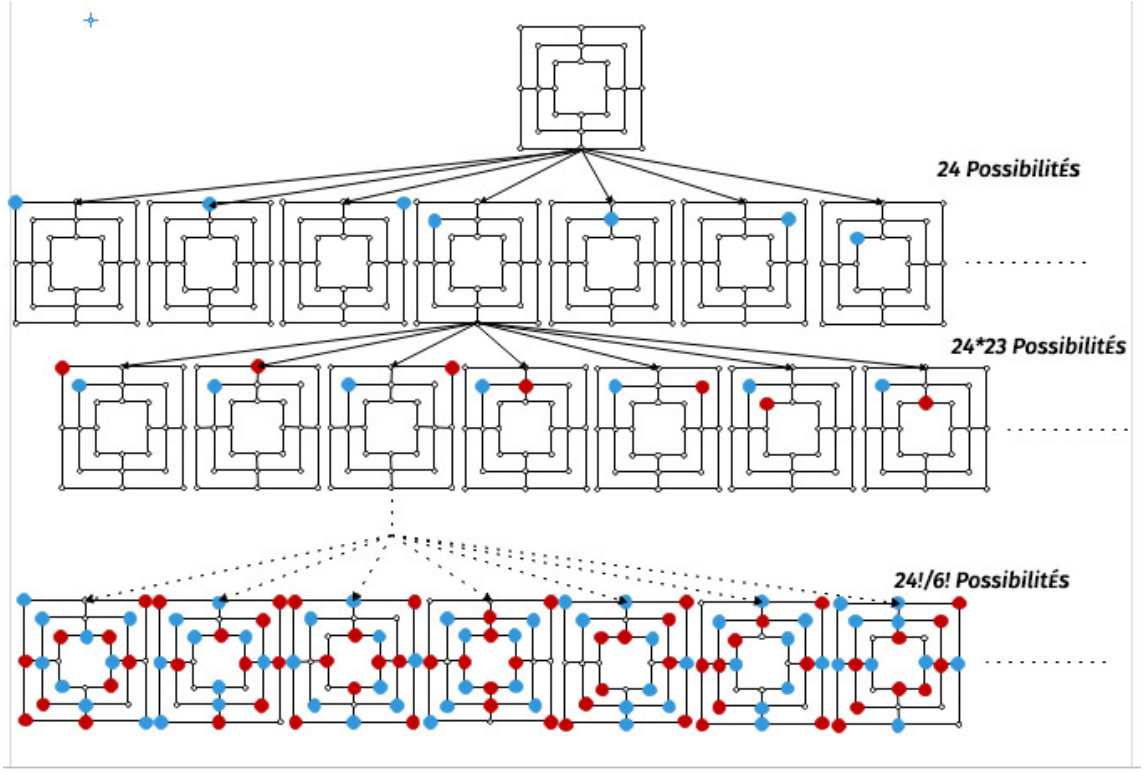


2.2.3 Production des successeurs lors de la 3eme phase :

Pour la troisième phase du jeu on cherche les intersections qui contient des jetons et puis on les bouge vers tous les intersections vide du plateau et au même temps on insère dans la liste des successeurs.



2.3 Graphe de résolution



Avec 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.

2.4 La fonction heuristique (fonction d'évaluation)

2.4.1 Première phase du jeu

Pour la première phase du jeu on donne la priorité aux moulins formés par l'IA c'est pour cela on a imlémenté la fonction heuristque comme la suite :

— Si un jeton va former un moulin avec une profondeur égale à 1 on donne une valeur à ce noeud qui est égale à : 2000.

— Si un jeton va empêcher la formation d'un moulin par l'adversaire on évalue ce noeud avec une valeur égale à 1000 et ceci pour donner la priorité aux moulins formés par l'IA car il car de toute façon les moulin de Min qui sont entrain d'être construites vons être détruite par la capture de l'un d'eux

— Si un jeton va former un moulin avec une profondeur supérieure à 1 et inférieure à la profondeur maximale on évalue le noeud à une valeur égale à 400/profondeur, et ceci

pour valoriser les noeuds noeuds qui sont plus proche à être formés. car la première phase du jeu ne tient pas durant tout le jeu.

— Si on atteint la profondeur maximale on évalue les noeuds avec 100*noeuds construits par Max - 50*Noeuds contruits par min, et ceci pour valoriser les noeuds qui permettent la construction des moulin par Max

2.4.2 Deuxième phase du jeu

— Si on arrive à la profendeur maximale on évalue le noeuds avec une valeur égale à : 3*nombre de moulins formés pour valoriser les noeuds qui permets de former plus de moulins

— Si on est pas dans la profondeur maximale mais un moulin peut être fromé on évalue le noeuds à une valeur égale à 1000/profendeur, pour valoriser les noeuds les plus proches à être formés.

2.4.3 Troisième phase du jeu

— Si on arrive à la profendeur maximale on évalue le noeud avec une valeur égale à 100*nombre de noeuds construits par Max

— si dans un noeud un moulin va etre construit par Max on évalue le noeud avec une valeur égale à 100000/profendeur, pour valoriser les noeuds les plus proches à être contstuits.

— si dans un noeuds max va empêcher la formation d'un moulin par Min, on évalue ce nouds avec une valeur égtale à 10000.

3

Implémentation du code

3.1 Conception du code de la version console :

Le jeu est composé de deux classes :

— La classe **Player** : Cette classe représente le joueur et contient comme membres :

  — “int finaldepth” qui représente la profondeur maximale et

  — ‘int nbr_noeuds” représentant le nombre de noeuds.

‘ La classe est aussi composé de 3 méthodes :

  — “int alphabeta1(int state[24],int depth,int ismax,int in)” qui est l’implémentation de l’algorithme Minimax pour la première phase

  — “int alphabeta2(int state[24],int depth,int ismax, int* dest,int it,int initial[24])” cette fonction joue le role de l’algorithme alphabeta pour le deuxième phase du jeu

  — “int alphabeta3(int state[24],int depth,int ismax,int in)” cette fonction joue le role de alphabeta pour la troisième phasee du jeu

— La classe **Board** : Elle représente la grille du jeu avec un tableau de 24 cases (state[24]), contient aussi les membres suivantes :

  — “int intersection_remplie[24]” qui désigne les intersections remplies.

  — “int player” qui indique quel joueur est en train de jouer.

  — “int times”, utilisé pour la première phase du jeu pour indiquer le nombre total des jetons joués.

  — “int noir” et “int blanc” représentant la couleur de chaque joueur.

  —

Pour les méthodes de la classe, ce sont les suivantes :

— “void color(int t,int f)” : elle a comme rôle de manipuler les couleurs de la console.

— “void drop(int i)” et “void Board ::drop2(int i)” : Colorier les cases remplies selon chaque joueur.

— “void afficher_grille()” : sert à afficher la grille.

— “void ingame()” : C’est la méthode qui s’occupe du déroulement du jeu en entier.

— “int heuristique(int player,int* table,int p)” : Elle s’occupe du calcul du score de chaque joueur.

— “bool moulinformation(int in, int player,int* table)” : Elle retourne un booléen qui indique si un jeton peut former un moulin dans une intersection indiqué comme paramètre.

— “bool validmove(int,int)” : Elle sert à tester si un emplacement et vide et par par conséquent si on peut mettre un jeton dedans.

— void “voisin(int i,vector<int>* v,int state[24],int with)” : indique les voisins de chaque intersection.

3.2 Conception du jeu graphique et version Android

Pour la bibliothèque graphie qu’on a utilisé cocos2d, pour implémenter la version graphique et la version Android du jeu elle traite le code avec des scènes, la manière de fonctionnement de cette bibliothèque est de telle façon que chaque cas d’utilisation est une scène, ce qui fait que le code de est divisés sur des classes qui représentent des scènes du jeu, voici les scènes qui composent notre jeu :

— SplashScene : cette classes qui est aussi une scene remprésente l’écrant qui apprait lors de l’ouverture du jeu

— MaiMenuScene : cette classes contient le menu pricipale du jeu, qui cotient les buttons 1 joeur

— GameScene : cette classe contient le traitement d’une partie entre 2 joeurs.

— GameSceneIA : cette partie contient le traitement qui se fait lors d’une partie entre un humain et une marchine.

— GameOverScene : cette classe cotient l’écrant qui apparait lors de la fin d’une partie

— Player : cette classe contient les algorithmes AlphaBeta pour chaque phase du jeu.

voici les scènes qui composent notre de jeu :

3.3 Comparaison entre Minimax et Alpha-beta

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.

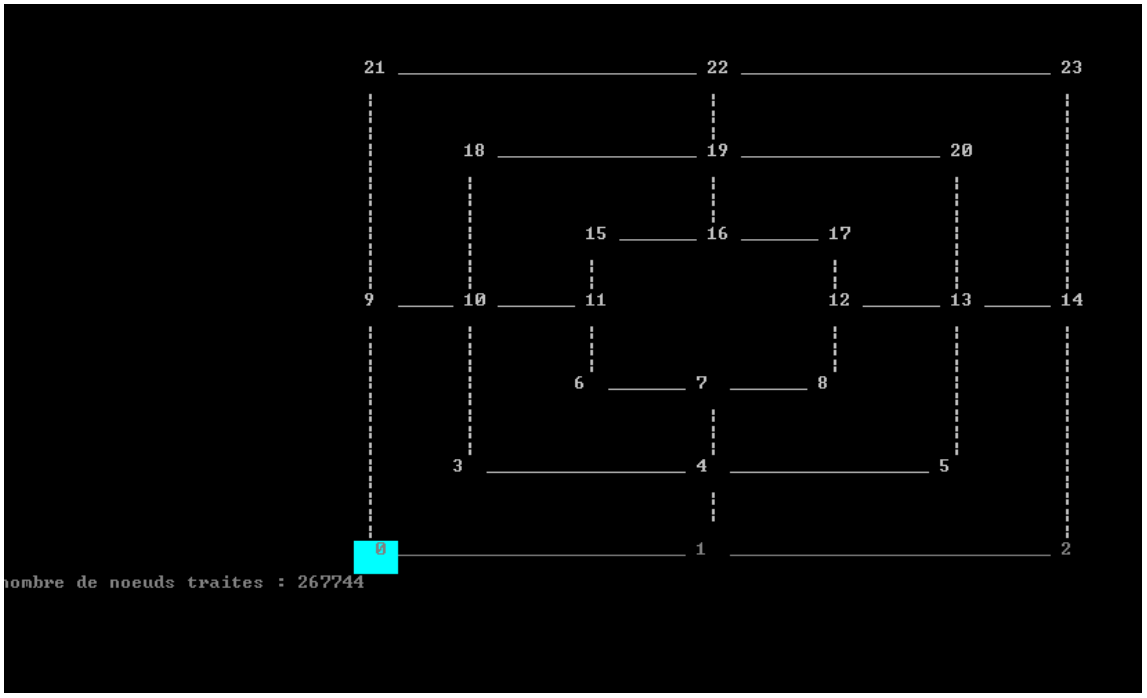


Figure 1 – Le nombre de noeuds explorés par Minimax

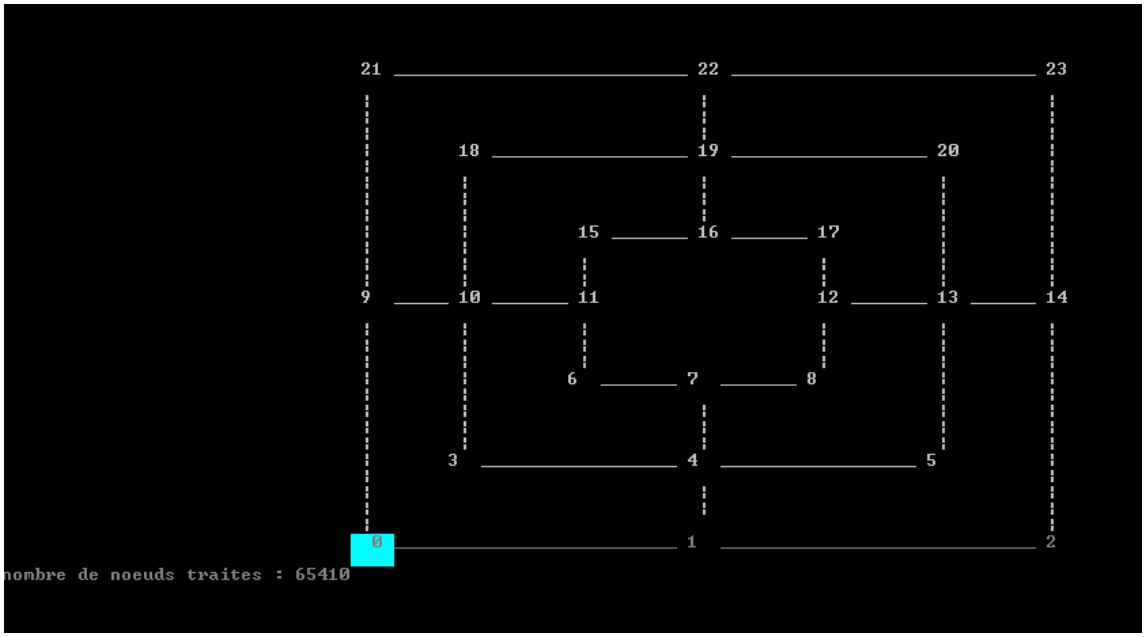


FIGURE 2 – Le nombre de noeuds explorés par Alpha-beta

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 oeuvre les connaissances qu'on a acquis dans cet élément de module pour réaliser un vrai projet qui contient une vrai intelligence artificiel.

Références

[1] *Heuristic Evaluation Function for Nine Men's Morris,*
www.kartikkukreja.wordpress.com/2014/03/17/heuristicevaluation-function-for-nine-mens-morris/

[2] *Minimax woes !,*
www.gamedev.net/forums/topic/543332-minimax-woes/

[3] *Minimax Algorithm C++,*
https ://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-4-alpha-beta-pruning/

[4] *Alpha Beta Pruning,*
http ://eric-yuan.me/alpha-beta-pruning/