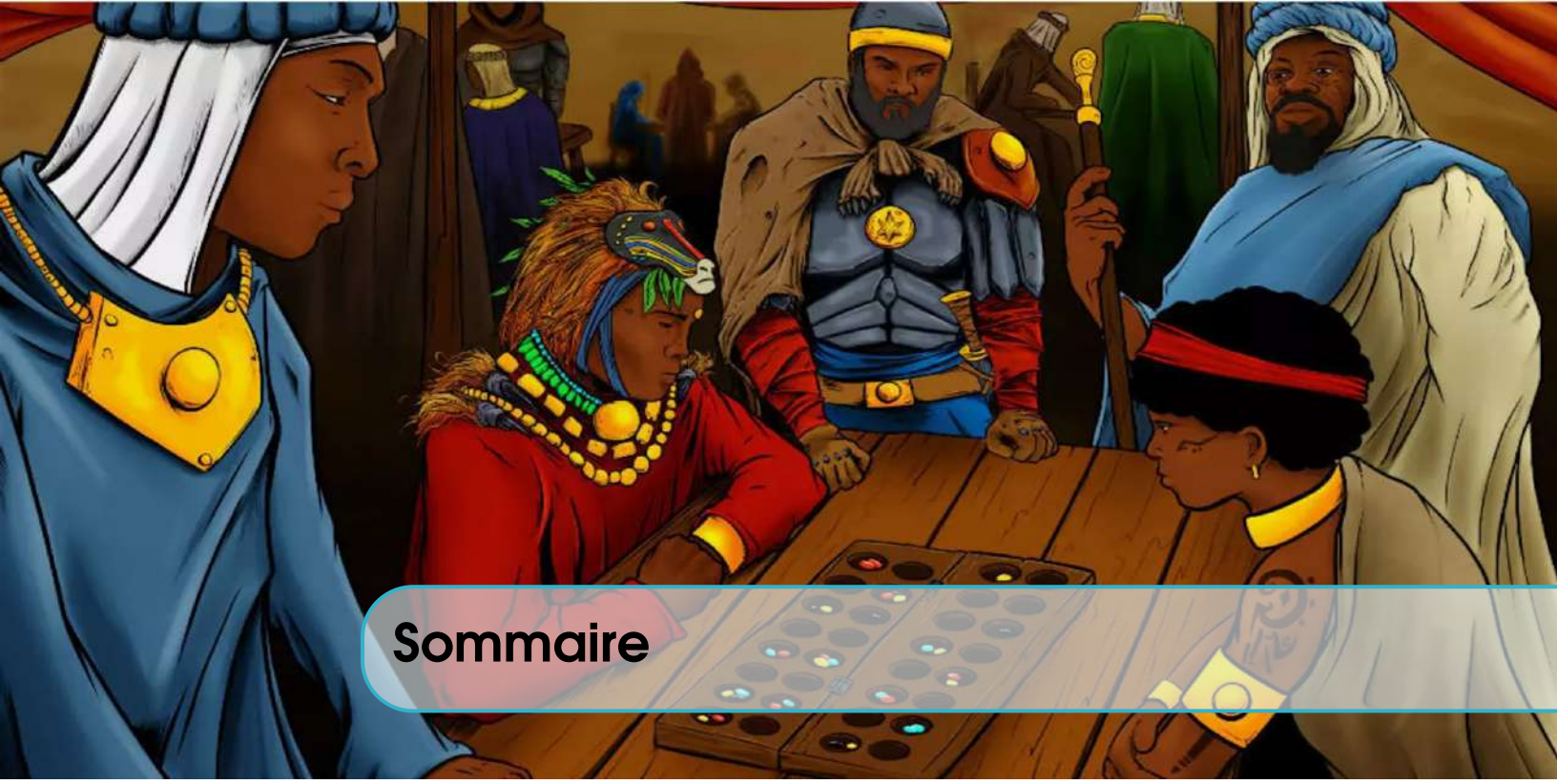
IA Game

Implémentation IA pour le jeu Awale

NERSISSIAN Tigran, TOPRAK Erdal

UNIVERSITÉ
CÔTE D'AZUR

PROJET DE IA GAME, UNIVERSITÉ DE NICE
Ce projet a été réalisé grâce aux enseignements de M.Régin.
Janvier, 2021



Sommaire

1. Introduction

1.1 Membres du projet

Notre projet est composé de 2 membres, nous avons NERSISSIAN Tigran et TOPRAK Erdal. C'est grâce à un effort concerté que nous sommes arrivés à implémenter le jeu, ces règles spécifiques et les améliorations.

1.2 Description du projet

Notre projet consistait à mettre en place le jeu d'Awale avec des règles spécifiques comme le plateau de taille variable (merge) et l'affamage ce qui diffère du jeu original. Le jeu d'Awale est un jeu d'origine d'Africaine. Notre plateau se compose de 2 rangées avec 12 trous (puis 6), chaque trous possède 4 graines en début de partie. Le but du jeu est de récolter plus de graines que l'adversaire. Dans notre variante le joueur 1 possède toutes les cases impaires et le jeu s'arrête lorsque qu'il y a moins de 8 graines sur le plateau.

1.3 État de l'art

Il faut savoir que l'awale est un jeu combinatoire déterministe où les informations sont complètes et parfaites, c'est à dire qu'on peut prédire tout les états de jeu "futur" en connaissant les conditions initiales.Les deux joueurs possèdent les mêmes informations (aucune n'est dissimulés). Il faut savoir qu'une variante de l'Awale à été résolue mathématiquement c-a-d tout l'arbre complet du jeu à été calculé, la partie se solde par un match nul dans le cas où 2 IA jouent les coups parfaits.

2. Méthodologie

Notre première approche a été d'implémenter un algorithme MinMax voir figure 2. Mais le problème c'est que le MinMax effectue une exploration complète jusqu'à une profondeur fixé de tout les états de jeu possible tant dit que dans la pratique nous n'avons pas besoin de connaître l'ensemble de l'arbre du jeu pour une profondeur fixé. c'est pour cela que nous utilisons une variante du MinMax ,Le MinMax Alpha-Beta prunning voir figure 2.

```
function minimax(node, depth, maximizingPlayer) is
    if depth = 0 or node is a terminal node then
        return the heuristic value of node
    if maximizingPlayer then
        value :=
        for each child of node do
            value := max(value, minimax(child, depth-1, FALSE))
        return value
    else (* minimizing player *)
        value := +
        for each child of node do
            value := min(value, minimax(child, depth-1, TRUE))
        return value
```

Figure 2.1: Algorithme pseudo-code du MinMax . Source Wikipédia

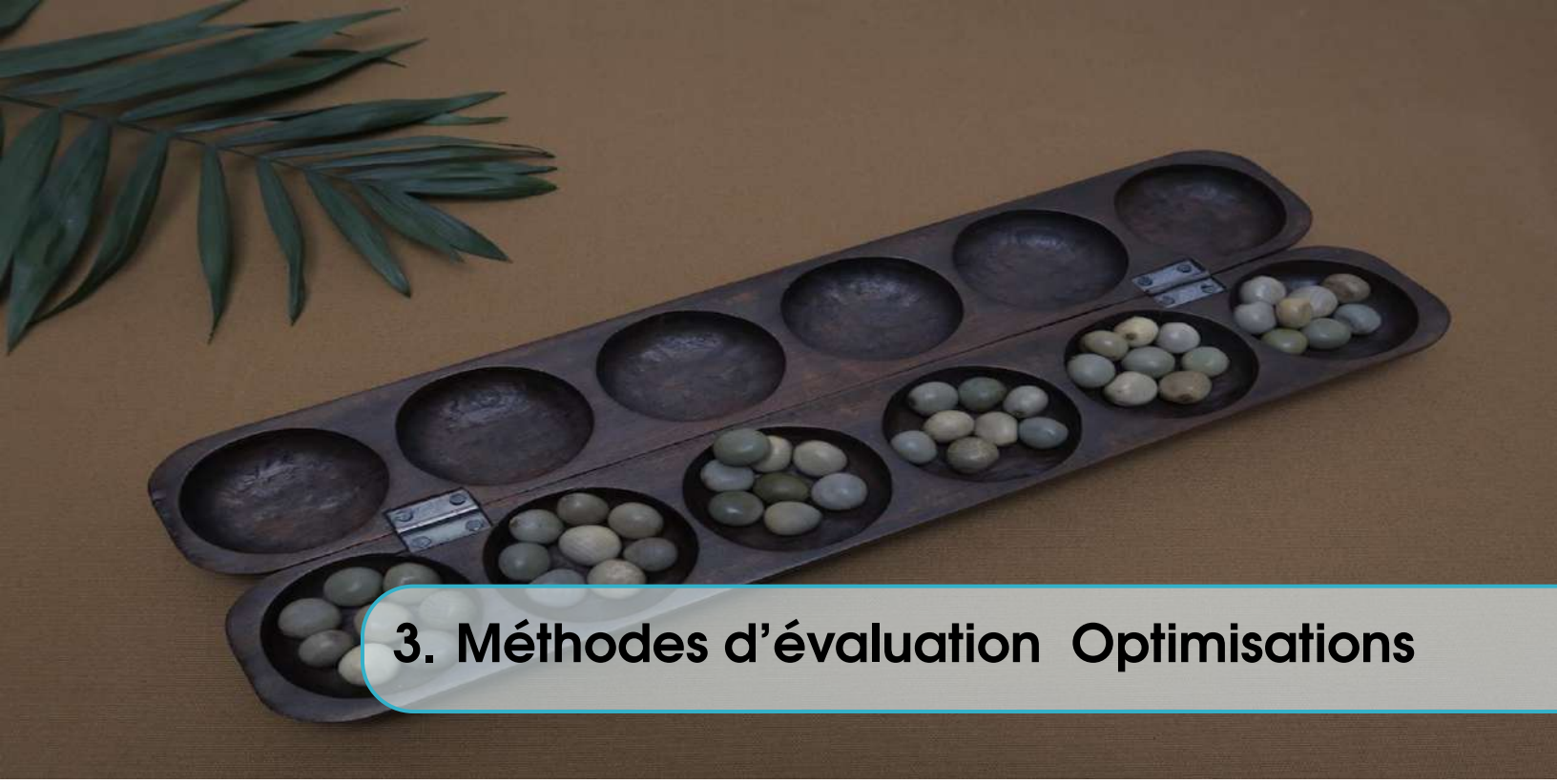
Par exemple après avoir joué et récolté les graines on vérifie si l'adversaire est affamé dans ce cas on prends la totalité des graines qu'il y a sur le plateau de jeu. voir 2

```
fonction alphabeta(noeud, A, B) /* A est toujours inf rieur    B */
   si noeud est une feuille alors
       retourner la valeur de noeud
   sinon si noeud est de type Min alors
           v = +inf
           pour tout fils de noeud faire
               v = min(v, alphabeta(fils, A, B))
               si A >= v alors  /* coupure alpha */
                   retourner v
               B = Min(B, v)
    sinon
           v = -inf
           pour tout fils de noeud faire
               v = max(v, alphabeta(fils, A, B))
               si v >= B alors /* coupure beta */
                   retourner v
               A = Max(A, v)
    retourner v
```

Figure 2.2: Algorithme pseudo-code du MinMax Alpha-Beta prunning. Source Wikpedia

```
int cpt = 0;
if (computer_play) {
    for (int i = 0; i < SIZE6; i++) {
        cpt += pos_next->cells12[2 * i + 1];
    }
}
else {
    for (int i = 0; i < SIZE6; i++) {
        cpt += pos_next->cells12[2 * i];
    }
}
if (cpt == 0){
    if (computer_play) {
        for (int i = 0; i < SIZE6; i++) {
            pos_next->seeds_computer += pos_next->cells12[2 * i];
            pos_next->cells12[2 * i]=0;
        }
    }
    else {
        for (int i = 0; i < SIZE6; i++) {
            pos_next->seeds_player += pos_next->cells12[2 * i + 1];
            pos_next->cells12[2 * i + 1]=0;
        }
    }
}
```

Figure 2.3: Code cpp extrait de la méthode playMove



3. Méthodes d'évaluation Optimisations

3.1 Évaluation

Nous avons une méthodes d'évaluation basique qui est simplement la différence des graines prises, vu que les score sont symetriques autours de 0 nous pouvons facilement l'adapter pour l'algorithme NegaMax alpha-beta que nous aurions bien voulu implémenter.voir(3.1) Aussi lors que le joueur peut gagner, on privilégie la branche qui nous permets de gagner avec le moins de coups joué possibles.voir(3.1)

```
int evaluation(Position * pos, int computer_play, int depth) {
    return pos->seeds_computer - pos->seeds_player;
}
```

Figure 3.1: Méthodes d'évaluation

3.2 Profondeur Variable

Aussi nous ne pouvons, d'entrée de jeu, avoir une profondeur élevée car le MinMax Alpha-Beta n'arrive pas à élaguer beaucoup des branches de l'arbre du jeu au début (ce qui est attendu) de ce fait il a la même complexité que le MinMax normal. Nous utilisons une stratégie qui consiste à compter le nombre de graine qu'il a sur le plateau et d'adapter dynamiquement notre profondeur en fonction aussi nous pouvons donc anticiper le merge du plateau et donc augmenter notre profondeur avant qu'il y ai moins 54 graines sur le plateau. Il faut savoir que lorsque on joue un coup, on boucle sur la quantité de graine présente dans la case joué de ce faite la complexité global de notre algorithmes dépends de la quantité de case jouable et du nombre de graines présente dans ces case.

```
if (finalPosition(pos_current, computer_play, depth)) {
    if (pos_current->seeds_computer > 48) {
        return 96 + depthMax - depth;
    } else {
        if (pos_current->seeds_player > 48) {
            return -96;
        } else {
            if ((pos_current->seeds_computer == 48) &
            (pos_current->seeds_player == 48)) {
                return 0;
            }
        }
    }
}
```

Figure 3.2: Code extrait depuis la méthode MinMax

```
if (finalPosition(pos_current, computer_play, depth)) {
    if (pos_current->seeds_computer > 48) {
        return 96 + depthMax - depth;
    } else {
        if (pos_current->seeds_player > 48) {
            return -96;
        } else {
            if ((pos_current->seeds_computer == 48) &
            (pos_current->seeds_player == 48)) {
                return 0;
            }
        }
    }
}
```

Figure 3.3: Code extrait depuis la méthode IAplay

3.3 Optimisations

L'une de nos principal piste d'amélioration a était d'utiliser le compilateur pour optimiser le code au lieu de l'optimiser nous même. Voir les figures suivantes pour s'en convaincre.

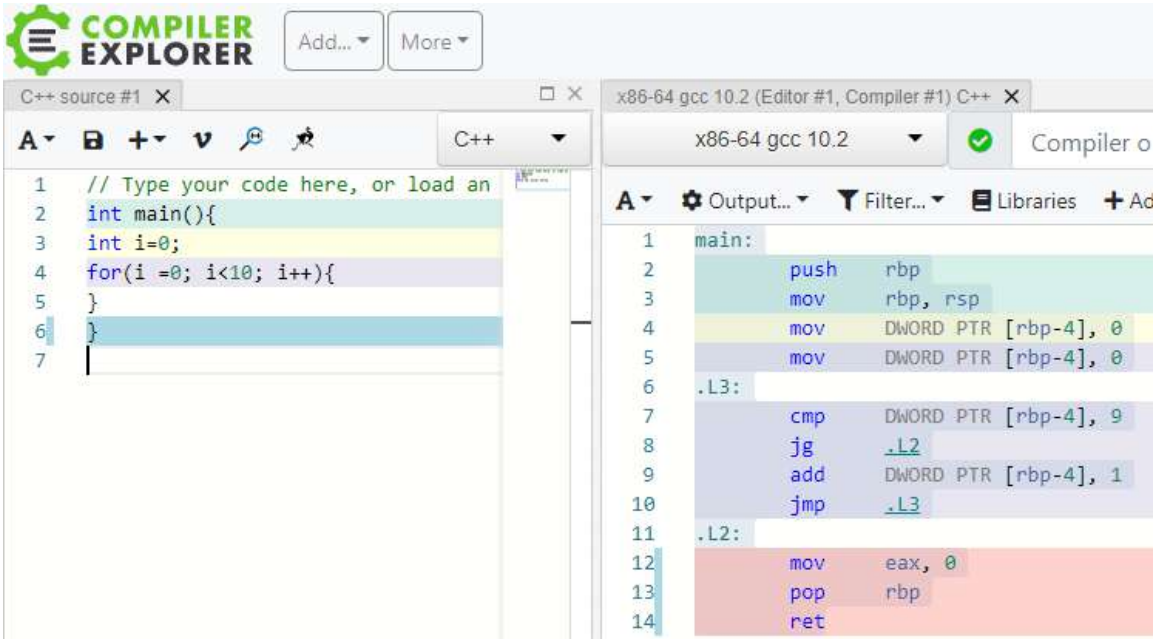


Figure 3.4: Boucle simple avec comparaison

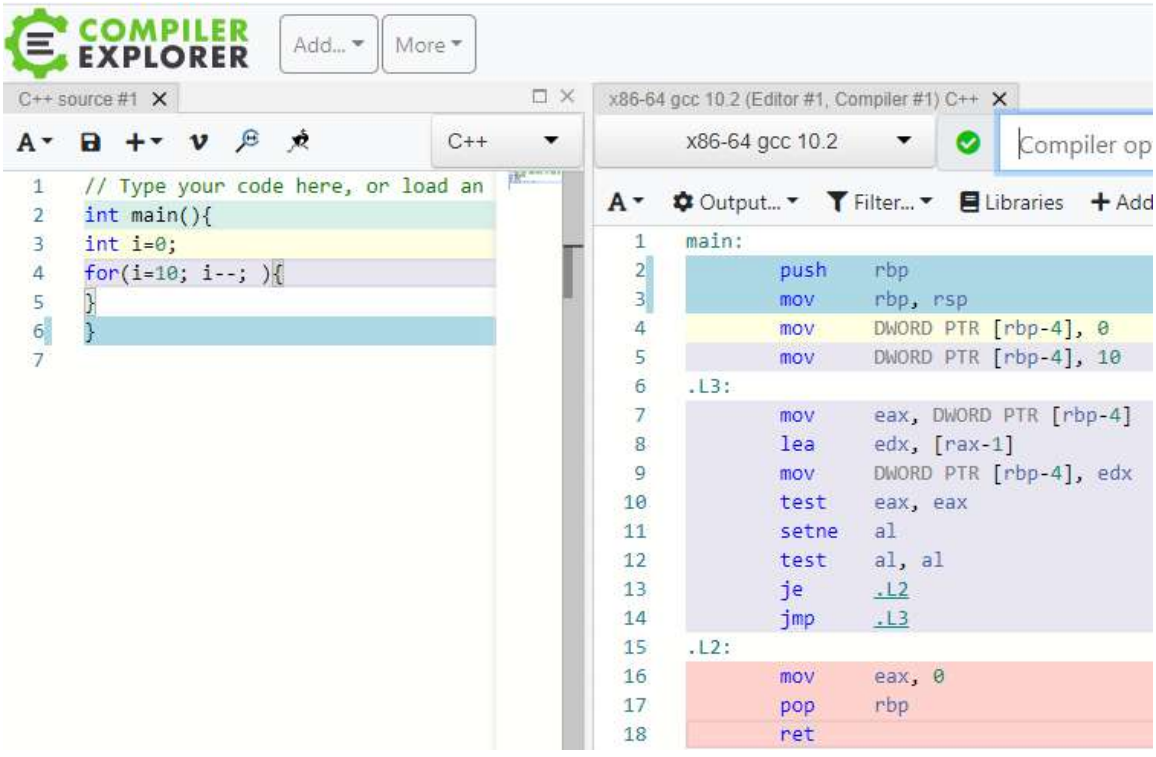


Figure 3.5: Boucle optimiser sans comparaison

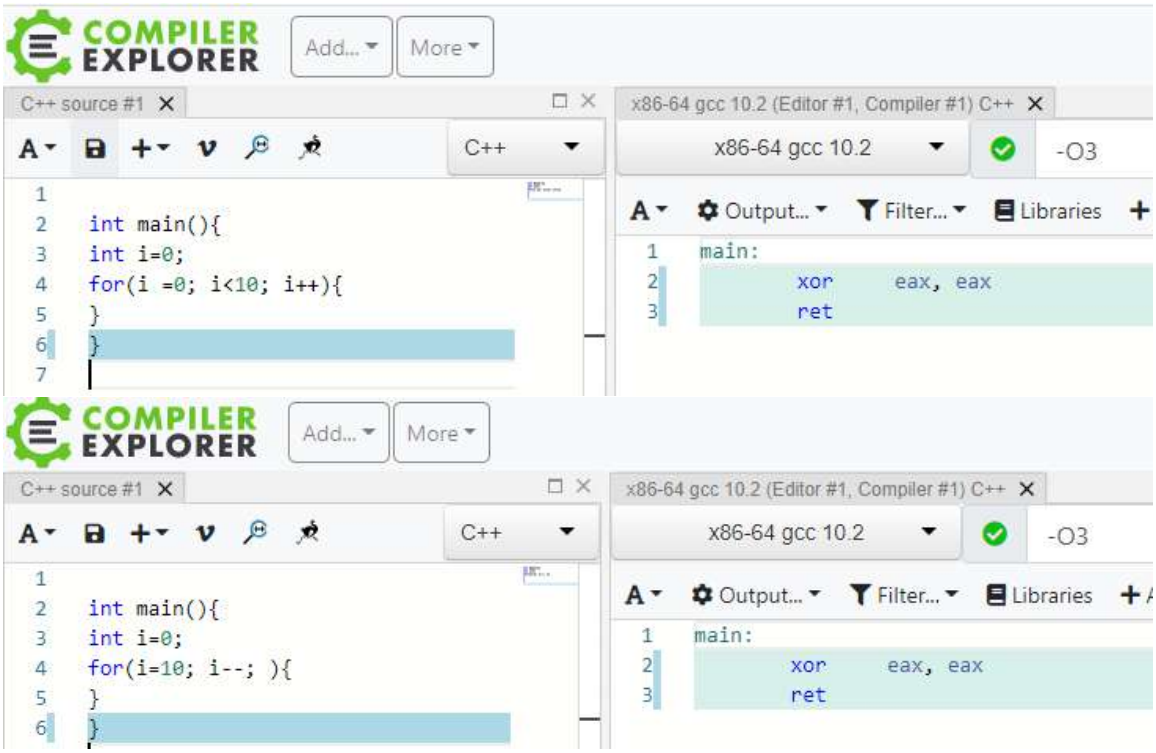


Figure 3.6: Les deux boucles compilés avec l'option -O3 sur Godbolt donne le même Code Assembleur

4. Résultats

```
root@localhost:~/IA_GAME# ls
impaire  impaire.cpp  paire  paire.cpp
root@localhost:~/IA_GAME#
```

Figure 4.1: Nos deux fichiers nous permettant de jouer une partie paire ou impaire

```
root@localhost:~/IA_GAME# ./paire
FIRST ?
0

Please type number between 1 to 24 :
```

Figure 4.2: Lors du lancement de "paire" on met 0 et on attend le premier coup de notre adversaire

```
Please type number between 1 to 24 :1
Human Score 0

IA Score 0

_0_ _5_ _5_ _5_ _5_ _4_ _4_ _4_ _4_ _4_ _4_ _4_
_4_ _4_ _4_ _4_ _4_ _4_ _4_ _4_ _4_ _4_ _4_ _4_

IA PLAY ################## 2 ##################
Human Score 0

IA Score 0

_0_ _0_ _6_ _6_ _6_ _5_ _5_ _4_ _4_ _4_ _4_ _4_
_4_ _4_ _4_ _4_ _4_ _4_ _4_ _4_ _4_ _4_ _4_ _4_


Please type number between 1 to 24 :
```

Figure 4.3: Un exemple de coup de notre adversaire et notre réponse dans la situation "paire"

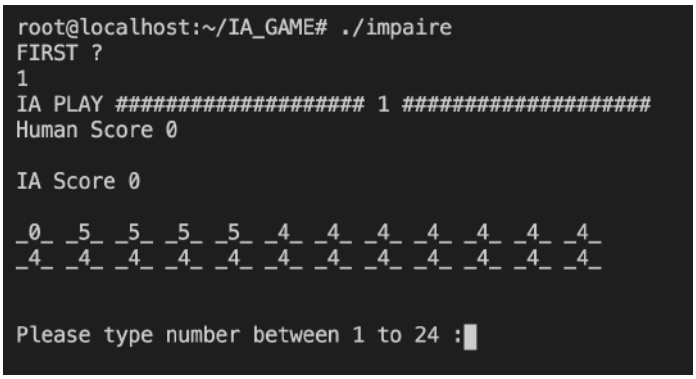


Figure 4.4: Lors de lancement d'"impaire" on met 1 et on reçoit immédiatement le coup de notre IA

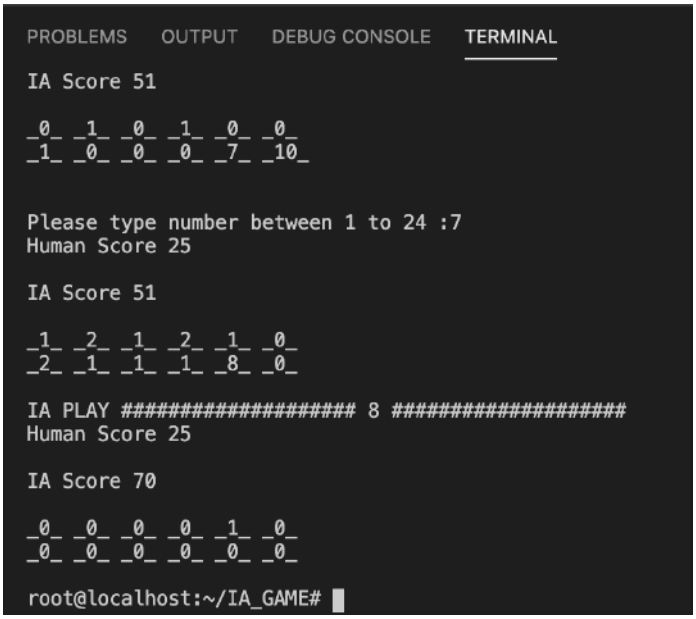


Figure 4.5: Exemple de fin de partie dans la situation "paire"

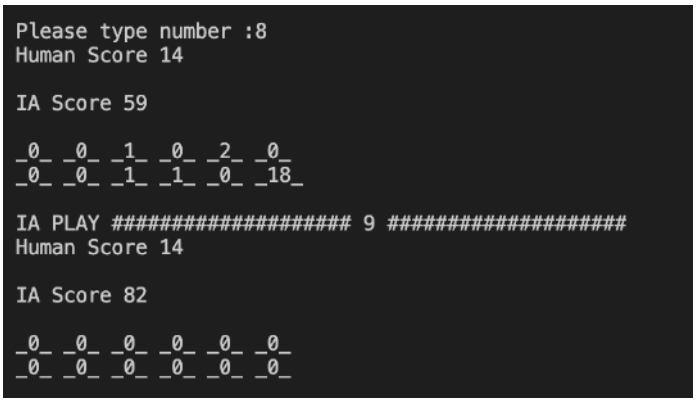
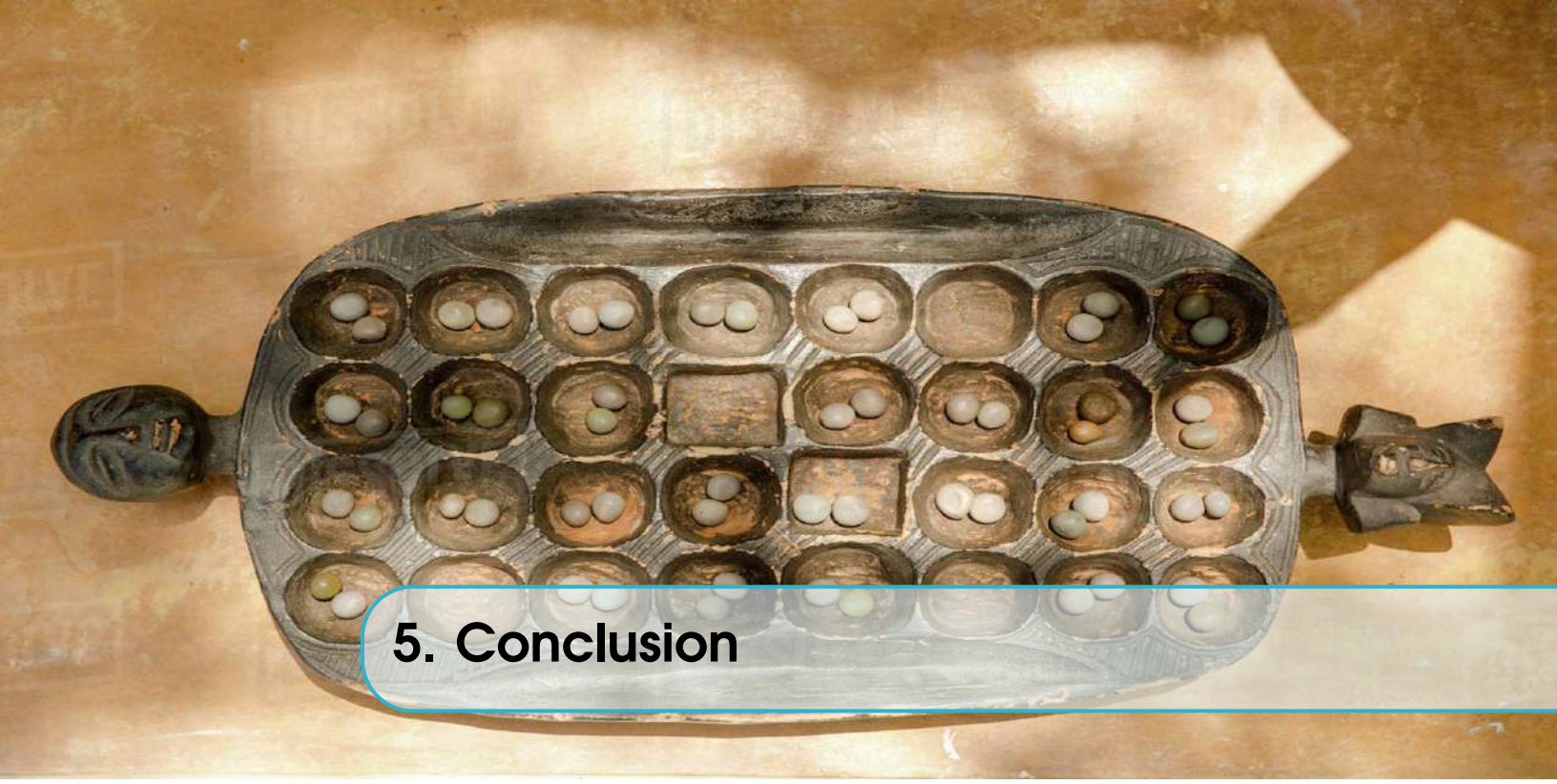


Figure 4.6: Exemple de fin de partie où un des joueurs est affamé



5. Conclusion

Ce projet de groupe nous à permit d'acquérir de nombreuses compétences qui nous étaient inconnues il y a encore quelques semaines. Cet effort collectif nous a permis d'apprendre le concept de minimax, d'alpha-beta prunning. Nous avons également appris certains concepts d'optimisation du compilateur. Nous avons essayé de paralléliser le code avec OpenMP nous avons pu notamment paralléliser les boucles for et faire une réduction sur la valeur que l'on cherche à minimiser/maximiser (dans la pratique il faut rajouter "omp parrallel for reduction(min:MinValue) reduction (max:MaxValue)" par exemple). Cependant la communication entre les threads coûtent plus que le gain fournit par notre l'implémentations du multi-threading, de ce faits nous avons abandonner cette piste. Nous avons améliorer notre fonction d'évaluation et également expérimenter avec le concept d'opening fixe selon des stratégies connues.