Projet Blobwar

Hossam ElOuati, Mohamed Aziz Zeroual

April 2021

1 Introduction :

Le projet consiste à réaliser, concevoir et programmer proprement une IA pour le fameux jeu "Blobwar". La conception du jeu se fera en langage de programmation Rust.

2 Principe du jeu :

Le jeu de Blobwar s'effectue sur un échiquier 8 cases sur 8, chaque joueur dispose d'un ensemble de pions ou blobs qui sont susceptibles d'être déplacés en cases voisines. Le déplacement du blob sur une case adjacente implique sa duplication sur la case cible. On peut aussi sauter à une distance de deux cases, on regarde dans ce cas les cases voisines de la case d'arrivée, les blobs occuppants ces cases changent de couleur.

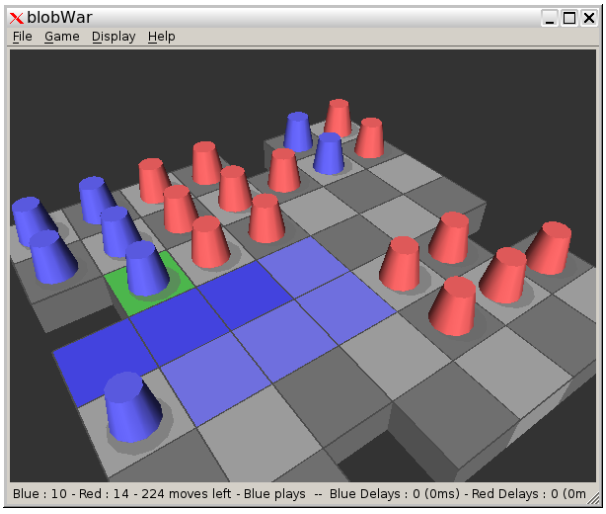

Figure 1: Blobwar

3 Les algorithmes nécessaires :

Pour garantir la conception subtile du jeu, respecter ses paramètres et les contraintes qui'ils imposent, on proposera de coder en particulier deux algorithmes; Min-Max, et Alpha-Beta.

3.1 Algorithme Min-Max :

Le principe de l'algorithme Min-Max s'articule sur le fait de choisir un "meilleur" successeur dans un arbre donné où chaque niveau est successivement à maximiser et à minimiser. L'algorithme parcourt l'arbre en profondeur d'abord. Cet algorithme s'adapte à notre situation comme étant une stratégie pour le jeu Blobwar. On considère que la maximisation est le choix du meilleur coup possible et que la minimisation représente le choix de l'adversaire en supposant que celui-ci choisit aussi la meilleure position pour lui (selon nous).

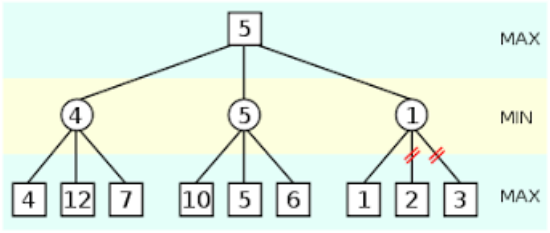

Figure 2: Maximisation et minimisation des niveaux

Le choix s'éffectue de telle façon de privilégier la valeur minimale de l'ensemble des valeurs proposées dans une situation donnée, puis de choisir le maximum de toutes ses valeurs minimales.
En pseudo-code, on peut élaborer une première idée de l'algortihme:

```
function MINIMAX(N) is
begin
   if N is a leaf then
        return the estimated score of this leaf
   else
        Let N1, N2, .., Nm be the successors of N;
        if N is a Min node then
           return min{MINIMAX(N1), .., MINIMAX(Nm)}
        else
           return max{MINIMAX(N1), .., MINIMAX(Nm)}
end MINIMAX;
```

3.2 Algorithme Alphabêta

Cette méthode consiste à réduire le nombre de noeuds explorés par la stratégie Minmax: pour chaque noeud, elle calcule la valeur de la position ainsi que deux valeurs, alpha et bêta.
Alpha : une valeur toujours plus petite que la vraie evaluation. Au debut du parcours, Alpha est évaluée -INFINI pour un noeud quelconque et la valeur de l' evaluation pour une feuille. Ensuite, pour un noeud à maximiser, Alpha vaut la plus grande valeur de ses successeurs et, pour un noeud à minimiser, elle est celui du predecesseur.
Bêta : une valeur toujours plus grande que la vraie evaluation. Au debut du parcours elle vaut + INFINI pour un noeud quelconque et la valeur de l'evaluation pour une feuille. Ensuite, pour un noeud à maximiser, elle est celui du prédecesseur et, pour un noeud à minimiser, bêta vaut la plus petite valeur de ses successeurs.
Ce qui est garanti en utilisant ces deux valeurs :
– La valeur d'un noeud ne sera jamais plus petite que alpha et jamais plus grande que bêta. (c'est un type d'encadrement de valeur)
– Alpha ne décroît jamais, beta ne croît jamais.
– Quand un noeud est visite en dernier, sa valeur est celle de alpha si ce noeud est à maximiser, et celle de bêta sinon.

3.2.1 Optimisation de la méthode Alphabêta :

On peut proposer 3 approches d'améliorations :
-Envisager un Tri pour ameliorer l'ordre d'examen des noeuds. Cela permet de faire beaucoup plus de coupes.
-Proposer une réduction: plus l'intervalle de recherche (bêta - alpha) est petit, plus il y aura de coupes.
-Sauvegarde des résultats dans le cas de leurs réapparitions, puisqu'une position donnée est accessible par différents mouvements.