# Rapport projet ALGOA : Blobwar

## 1 Introduction

Le but du projet est d'implémenter des algorithmes capables de déterminer les meilleurs blobs à jouer dans BlobWar, un jeu de plateau tour par tour à 2 joueurs. Nous parlerons notamment des stratégies gloutonne, min-max, alpha-bêta, et min-max parallèle. Nous présenterons aussi des analyses expérimentales sur le fonctionnement de ces algorithmes, afin de déterminer quelle méthode il sera préférable d'utiliser lors du tournoi entre les IA.

## 2 Implémentation des algorithmes

### 2.1 Stratégie gloutonne

La 1ère stratégie implémentée est la stratégie gloutonne. Elle consiste à choisir le mouvement, parmi tous les mouvements valides, qui maximise le gain immédiat (sur 1 coup) du plateau. Pour chaque coup, une simulation de ce qu'il ferait est calculée dans une version temporaire du plateau.

**Algorithm 1:** Stratégie gloutonne

**Input:** blobs, holes, cplayer, saveBestMove
**Output:** rien, le programme sauvegarde directement le meilleur coup

```
1  valid_moves ← [];
2  valid_moves ← computeValidMoves(valid_moves);
3  best_move ← valid_moves[0];
4  best_score ← −∞;
5  foreach mouvent valide mv de valid_moves do
6      applyMove(mv);
7      score ←estimateCurrentScore();
8      if score > best_score then
9          best_score ← score
10         best_move ← mv
11 saveBestMove(best_move);
```

### 2.2 Stratégie min-max

#### 2.2.1 Version non parallélisée

Le fonctionnement est le même que pour la stratégie gloutonne, à la différence que la profondeur observée sera plus grande. Il s'agit donc, à la place d'appeler `estimateCurrentScore()` directement (ligne 7), de faire un appel récursif à une fonction auxiliaire `minmax`.
Cependant, la complexité de cet algorithme est exponentielle, et le nombre de branches explorées va être très grand, comme remarqué dans Temps d'exécutions. Il faudra attendre d'implémenter l'élagage Alpha-bêta (2.3) pour réduire ce temps de calcul.

**Algorithm 2:** Fonction minmax récursive a appeler ligne 7 de la stratégie gloutonne

**Input:** int depth, bool maximizingPlayer
**Output:** le meilleur score atteint lors du parcours des profondeurs

```
1  valid_moves ← [];
2  valid_moves ← computeValidMoves(valid_moves);
3  if valid_moves est vide then
4      if maximizingPlayer then
5          return −∞
6      else
7          return ∞
8  if depth == 0 then
9      return estimateCurrentScore()
10 best_score ← maximizingPlayer ? -∞ : +∞
11 foreach mouvent valide mv de valid_moves do
12     applyMove(mv);
13     score ←minmax(depth - 1, !maximizingPlayer)
14     best_score ← maximizingPlayer ? max(score, best_score) : min(score, best_score)
15 return best_score
```

### 2.2.2 Version parallèle

Pour implémenter la version parallèle, on reprend l'algorithme Min-max, en créant une liste de la taille du nombre de threads utilisés. Ensuite, on parallélise le `for` sur `valid_moves`:

```
#pragma omp parallel for schedule(dynamic)
for(size_t i = 0; i < valid_moves.size(); ++i) {
    // traitement parallele
}
```

## 2.3 Stratégie alpha-bêta

### 2.3.1 Version non parallélisée

L'élagage Alpha-bêta consiste à supprimer des branches inutiles qui ne serviront pas. Cette technique est extrêmement efficace, car sur de gros arbres, c'est parfois plus de 50% des chemins qui seront éliminés. Une analyse du temps de réflexion est disponible en Moyennes des scores par stratégie

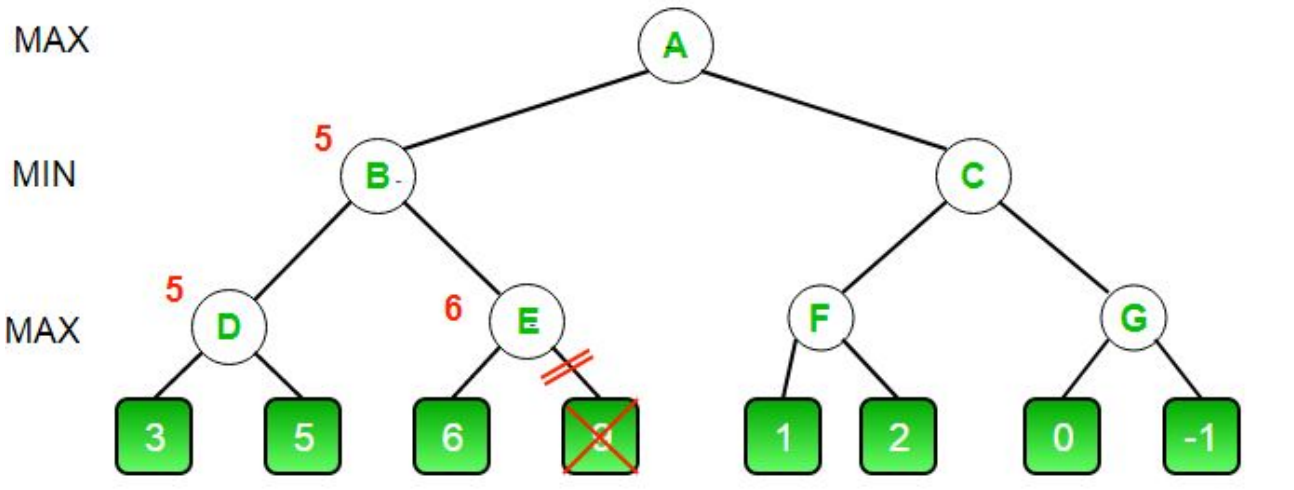


Figure 1: Illustration du temps

# 3 Autres changements

## 3.1 Fonction d'évaluation du plateau

Nous avons observé lors des parties que les algorithmes avec une profondeur élevée loupent des captures importantes et ne se focalisent sur la prise du centre et des coins de la carte, positions pourtant cruciales de notre point de vue. L'idée d'implémenter une heuristique nous est alors venue à l'esprit, mais nous n'avons pas eu le temps de le faire. Pour quand même favoriser l'algorithme à faire ces actions, nous avons modifié la fonction `estimateCurrentScore` de `strategie.cc`, pour mettre plus de points sur les actions de prise et de décalage vers les zones importantes.

# 4 Expérimentations

Pour effectuer un test de performances des différentes stratégies, nous avons eu comme objectifs les points suivants:

- Comparer les temps d'exécution des différents algorithmes.
- Analyser l'impact de la profondeur de recherche sur les performances.
- Évaluer l'efficacité des algorithmes dans des confrontations directes.
- Étudier l'impact de la configuration du plateau sur les performances.

## 4.1 Temps d'exécutions

Dans un premier temps, nous avons évalué les temps d'exécutions de chaque algorithme en fonction de la profondeur. 3

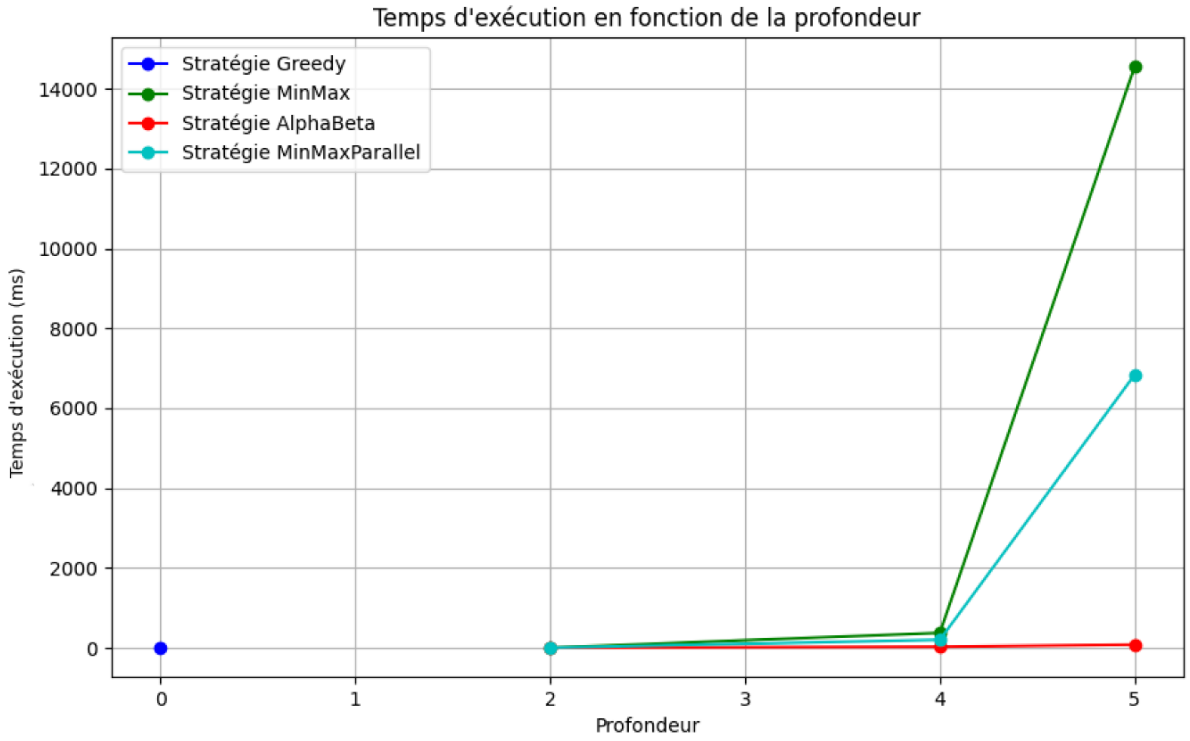


Figure 2: Temps d'exécution en fonction de la profondeur pour les différents algorithmes

On remarque que l'élagage alpha bêta est très efficace et supprime un grand nombre de branches qui ne seront pas explorées. Le parallélisme réduit aussi beaucoup le temps de calcul, mais n'est pas aussi efficace que l'élagage. La stratégie Greedy n'a pas de courbe car pas de profondeur à faire varier.

## 4.2 Combats entre algos

Voici aussi un tableau récapitulatif pour savoir quel algorithme bat lequel. Les combats ont tous été réalisés sur la carte standard. 1

| | Greedy | Min-max | Min-Max // | Alpha-bêta |
|---|---|---|---|---|
| Greedy | - | L | L | L |
| Min-max | W | - | L | D |
| Min-max // | W | W | - | D |
| Alpha-bêta | W | D | D | - |

Table 1: Résultats des confrontations entre les différentes stratégies

Il se lit en ligne : Greedy a perdu (L) contre Min-max, Min-max //, et Alpha-bêta. Il y a certains cas où le résultat est Draw (D). Il y a deux façons d'obtenir une égalité :

- La partie ne s'est pas arrêtée. Les deux joueurs jouaient en boucle.
- Le résultat a dépendu de l'algorithme qui avait le 1er coup.

## 4.3 Impact du choix de la carte sur les algos

Comme vu dans Combats entre algos, l'algorithme Min-max gagne à chaque fois contre Greedy dans la carte standard, mais il se trouve que sur la carte constrained et ring, c'est l'inverse.
Nous avons aussi remarqué un comportement étrange des blobs sur la fin de partie dans la carte chess.

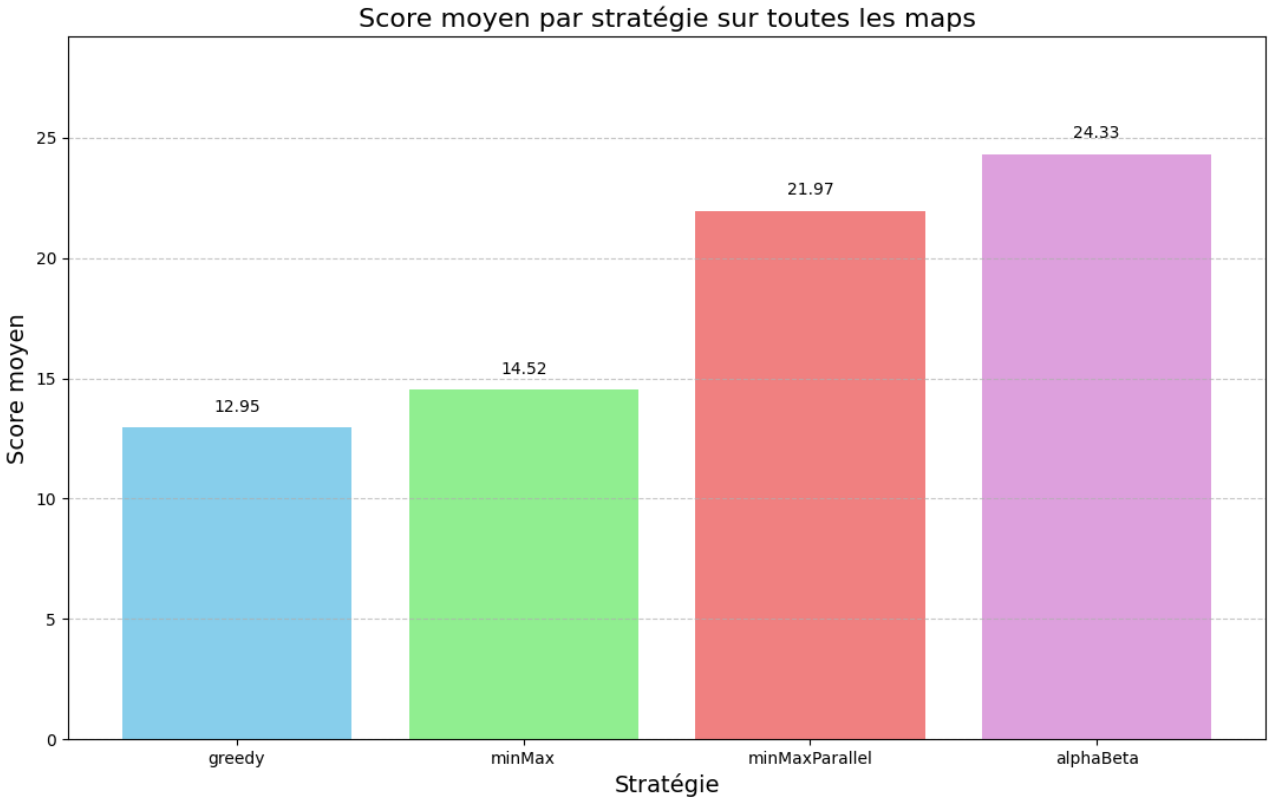


Figure 3: Moyennes des scores par stratégie

En moyenne, sur toutes les cartes, c'est `Alpha-bêta` qui l'emporte sur les autres algos que nous avons implémentés.