Rapport BlobWar

Greedy

La logique de l'algorithme glouton est relativement simple, puisqu'il suffit de calculer le coup avec la meilleur valeur, et de jouer ce coup. Il n'y a donc pas de notion de recherche récursive, ni d'anticipation des coups de l'adversaire. La seule chose un peu évolué est l'introduction d'un tableau de mouvement a jouer, dans le cas ou plusieurs mouvement aurait la meme valeur. Ce choix se fait alors de manière aléatoire parmi les mouvements candidats.

MinMax

De la meme manière, on retrouve ce choix d'avoir un résultat aléatoire parmi les choix candidats dans l'implémentation de minmax. Cette fois ci, on recherche le meilleur coup en prenant en compte les mouvement de l'adversaire, sur une certaine profondeur. On verra que, bien sûr, les résultats sont nettement meilleur qu'avec la logique Greedy, dans la partie *Analyse de l'efficacité.*

AlphaBeta

L'élagage AlphaBeta est une amélioration du code Minmax, et on le récent d'ailleurs dans les résultats d'analyse d'efficacité. Ici, on introduit en plus une valeur alpha et beta, qui représente en quelque sorte les bornes de valeur admissible. De cette manière, les branche trop faible sont élagué, et pas exploré. Cela provoque un gain d'efficacité dans le temps de calcul non négligeable.

Analyse de l'efficacité

J'ai tout d'abord fait jouer un algorithme contre lui-même, avec -lorsque c'est pertinent- la même profondeur dans chaque camps.

Greedy		
rouge	draw	bleu
67,00%	0,00%	33,00%

MinMax			
profondeur	rouge	draw	bleu
1	40,00%	0,00%	60,00%
2	60,00%	0,00%	40,00%
3	40,00%	0,00%	60,00%
4	0,00%	0,00%	100,00%

AlphaBeta			
profondeur	rouge	draw	bleu
1	0,00%	0,00%	100,00%
2	100,00%	0,00%	0,00%
3	0,00%	0,00%	100,00%
4	0,00%	0,00%	100,00%
5	0,00%	0,00%	100,00%

Tout d'abord, un fait intéressant. Alors que le fait de jouer en premier semble, pour Greedy, revêtir une importance particulière, on trouve plutôt le contraire pour minmax et AlphaBeta. Comme si plus la capacité d'anticipation de l'algorithme était développé, plus le fait de jouer en deuxième était important. Les pourcentage sont évidemment a titre indication, ayant été fait sur des sample de test de 5 a 10 occurrences, les marges d'erreurs sont évidemment très importante. On remarque néanmoins des résultats non négligeable.

	Greedy VS MinMax			MinMax VS Greedy		
profondeur	rouge	draw	bleu	rouge	draw	bleu
1	50,00%	10,00%	40,00%	60,00%	0,00%	40,00%
2	20,00%	0,00%	80,00%	90,00%	0,00%	10,00%
3	0,00%	0,00%	100,00%	80,00%	20,00%	0,00%
4	10,00%	0,00%	90,00%	100,00%	0,00%	0,00%

	Greedy VS AlphaBeta			AlphaBeta VS Greedy		
profondeur	rouge	draw	bleu	rouge	draw	bleu
1	20,00%	20,00%	60,00%	80,00%	0,00%	20,00%
2	0,00%	0,00%	100,00%	100,00%	0,00%	0,00%
3	0,00%	0,00%	100,00%	100,00%	0,00%	0,00%
4	0,00%	0,00%	100,00%	100,00%	0,00%	0,00%
5	0,00%	0,00%	100,00%	100,00%	0,00%	0,00%

On voit ici toute la supériorité d'AlphaBeta et de MinMax. En effet, des que ces algorithmes sont différents de Greedy (minimax(1) et Greedy étant identique), on voit une nette différence, la stratégie gloutonne perdant dans plus de 80% des cas dans le meilleur des cas.

	AlphaBeta VS MinMax(2)			MinMax(2) VS AlphaBeta		
profondeur	rouge	draw	bleu	rouge	draw	bleu
1	40,00%	0,00%	60,00%	60,00%	0,00%	40,00%
2	80,00%	0,00%	20,00%	0,00%	0,00%	100,00%
3	100,00%	0,00%	0,00%	0,00%	0,00%	100,00%
4	100,00%	0,00%	0,00%	0,00%	0,00%	100,00%

Comparaison intéressante ici, puisqu'il s'agit des deux meilleurs algorithmes. Et la conclusion est sans appel, AlphaBeta est meilleur des que les profondeurs sont comparable (ou plus élevé pour lui).

Tentative d'Optimisation d'AlphaBeta

J'ai tenté d'optimiser ma stratégie AlphaBeta en introduisant une mémoïsation des calculs, pour gagner en efficacité. Cette optimisation a eu un résultat plutôt mitigé. En effet, cela a permit de gagner en profondeur, entre 1 et 2. Néanmoins, j'ai la nette impression que le surcout de calcul, notamment la manipulation de table de hachage (qui est probablement extrêmement sous optimale dans mon code), rend ce gain très faible, surtout sur les profondeurs faible. Ce sentiment a été confirmé par le tournoi, ou mon code a été plutôt mauvais (1 victoire seulement). Une idée simple, a laquelle je n'avais pas pensé, pourrait être de simplement commencé le calcul a une profondeur plus élevé que 1 (pour le tournoi), mais a 4 ou 5 par exemple. Dans les confrontation contre MinMax et Greedy, l'algorithme non optimisé étant déjà très fort, on ne voit pas de différence notable.

Source

Toute les comparaisons de performance présenté ici, ainsi que d'autres (le meme algorithmes contre lui-même, mais avec des profondeurs différentes par exemple) se trouve en annexe, *dans le fichier Excel bench_blobwar.* Les données (ainsi que quelque autre, notamment pour l'optimisation) se trouve dans *des fichiers textes* dont le nommage est je pense pertinent. La génération de ces fichiers texte se trouve encore dans *main.rs*

Tout ce trouve aussi sur *https://github.com/vodkatypique/blob*