HUREL Arnaud
Elève-Ingénieur en master Informatique : Système Intelligent

Métaheuristique et jeu

Projet de construction d'une IA du jeu 2048

A l'attention de Monsieur CAZENAVE

À Paris le 9 mars 2015

Table des matières

1 Introduction sur le 2048

Le jeu 2048 a été créer en 2014 par Gabriele Cirulli à l'age de 19 ans, le but de ce jeu est de faire glisser des tuiles sur un grille de 4x4 cases afin d'obtenir une tuile égale à 2048. En effet, les tuilles ont pour valeur des puissance de 2. A chaque tour, le jeu fait apparaitre de manière aléatoire soit une tuile 2, soit une tuile 4 (90% pour la tuile 2 et 10% pour la tuile 4). Pour jouer, l'utilisateur a le choix entre quatres coup : Nord, Sud, Est et Ouest. Ceux-ci auront, respectivement, un effet de comprésion des tuiles vers le haut, le bas, la droite et la gauche.

Le jeu ainsi que les IAs ont été développer en python, vous pouvez y acceder en toute liberté sur mon Github à l'adresse suivante : https ://github.com/phatryun/2048

2 Algorithmes utilisés

Une fois le jeu créé, nous avons mis en place différents algortithme en vu de créer un intelligence artificiel capable de résoudre se casse tête. Nous avons alors regarder sur internet afin de se renseigner sur les différents algorithme utilisé

Pour des raisons techniques, lors de nos test nous avons choisi de faire tourner nos algortimes que avec une profondeur de 4.

2.1 Algorithme Naif

Cet algorithme consiste à déterminer le meilleur coup parmis tous ceux possible. Il peut être considéré comme naif, car il vas tout simplement effectuer tous les coups de manière exhaustif sans logique particulière. Dans notre fichier source il corresdond aux fonctions *nextMove* et *nextMoveRecur*. Voici son derroulement :

- Tant que la profondeur de recherche n'est pas égale à celle voulut, on va effectuer chaque coup possible (Nord, Sud, Est, Ouest)
- On va, ensuite, ajouter de manière aléatoire la nouvelle tuile pour ensuite calculer le score heuristique de la grille obtenue.
- Puis on va étudier ces fils qui lui retourneront leur meilleur score calculé par l'heuristique choisit.
- De ces dernier, on retiendra le coup pour le quel le score de l'heuristique à été le plus élévè.
- Enfin on va ajouter à son propore score celui de son meilleur fils mais de façon pondéré.

2.2 Expectimax

Cet algorithme est répartie en deux étapes : la première consiste à rechercher le meilleur coup possible calculé en maximisant le score obtenu grâce à une heuristique et ensuite de minimiser l'impacte de l'ajout aléatoire des nouvelles tuiles. Dans notre code nous le retrouverons à travers la fonction *player_max* qui aura pour objectid de maximiser le choix du coup et la fonction *player_expect* qui va minimiser les placements de la nouvelle tuile.

Voici plus en détail leurs déroulement :

- Premièrement, notre condition d'arret sera quand la profondeur de recherche sera égale à 0. Alors ont calculera notre score grâce à notre heuristique.
- La première étape consiste à tester tous les coup possible du moment qu'ils sont autorisé.
- Si ils le sont, alors on va passer à la deuxième partis de l'algo qui va minimiser le positionnement aléatoire de nos nouvelles tuiles
- Pour cela on va repérer tous les emplacements vide de notre grille et pour chaque emplacement on va ajouter une tuile pour repartir dans la première étape (calcul du score max).

– De ces score obtenue on va effectuer un calcul de score moyen pour obtenir une estimation de score en fonction de toutes les positions de nouvelles tuile possible

3 Heuristiques utilisé

3.1 Utilisation du score

La première heuristique qu'y nous est venue à l'esprit est celle maximisant le score de la grille. En effet, celui ci est automatiquement calculé à chaque fusion de tuile. Il fut donc facile de mettre en place cette heuristique qui va préférer le coup rapportant le plus de point. Malheureusement cette heuristique n'est pas tres concluante (comme le prouve les résultats obtenues ci-dessous). En effet, celle-ci va préférer les fusions de tuiles direct au défaut des fusions qui pourraient se produire au coup d'après et qui rapporteraient plus de point.

3.2 Gradient

L'heuristique du gradient permet de favoriser les grilles qui ont pour valeur maximale des tuille de dans un coin de la grille. Ainsi pour chaques grilles, on va calculer sont score en fonction des 4 gradients et on retournera le meilleur score. Le score est le résultat d'une somme des multiplication entre les valeurs des tuiles et les poids contenu dans la matrice des gradients.
Cette heurisitque est facile à mettre en place mais elle est très couteuse car en effet, pour chaque appel de cette fonction on va calculer 4 score. Néanmois on obtient de bon résultats

3.3 Ordre des valeurs des cases

Cet heuristique permet de preferer le choix de construction d'une suitre parfaite. En effet, avec un système de pondération des valeurs de la grille, nous allons définir une suite parfaite qui pour notre cas commancera en bas à droite de notre grille.

3.4 Conjugaison d'heuristiques avec ajout de la maximisation des cases libres

De ces deux dernières heuristiques, nous avons également rajouté un impacte sur le nombre de case vide. En effet, nous allons dans ce cas maximiser le nombre de case vide restant après le coup dans le but de faire durer le plus longtemps possible la partie.

4 Résultats obtenus

Les résultats obtenus ont été réalisés sur une machine Debian 64bit doté de 4 processeurs Intel Core i5-2520M CPU 2.50GHz et de 8Go de RAM. Pour des raisons technique, nous avons réalisé les test avec comme profondeur maximum de 4. Dans les tableaux ci-desous nous observons la fréquence d'atteinte de palier sur 25 tests.

Algorithme	Naif				
Heuristique	Score	Gradiant	Gradient ++	ordre	ordre ++
0	13	3	0	4	0
1024	12	5	0	1	0
2024	0	13	0	13	0
4096	0	4	0	7	0
8192	0	0	0	0	0
temps moyen	0	239	0	209	0

Algorithme	Expectimax				
Heuristique	Score	Gradiant	Gradient ++	ordre	ordre ++
0	25	10	0	2	0
1024	0	13	0	3	0
2024	0	2	0	18	0
4096	0	0	0	2	0
8192	0	0	0	0	0
temps moyen	32	183	0	94	0