Rendu EDD stratégie

Pinero, Borde, Bonnet

April 6, 2015

Contents

1 Stratégie fonctionnelle

La stratégie fonctionnelle choisie pour le rapport est la stratégie du coin. Pour cela, nous avions fait un premier algorithme qui se contente de jouer vers la gauche tant qu'on le peut, sinon il joue vers le bas encore une fois tant qu'on le peut, de même vers la droite puis vers le haut. Avec cet stratégie, sur 48 parties nous avons un score moyen de 2475, avec 6x64, 20x128, 20x256 et 2x512.

Dans un seconds temps, nous avons fait un second algorithme qui est basé sur le premier mais qui un coup sur deux change les mouvements favoris. Les premiers mouvements favoris sont dans cet ordre, gauche, bas, droite et haut, et les seconds mouvements sont dans cet ordre, bas, gauche, droite et haut. Avec cette stratégie, sur 48 parties nous avons un score moyen de 2259, avec 1x32, 3x64, 21x128 et 23x256.

Les deux stratégies se valent à peu près, mais c'est bien la première qui est la meilleure.

2 La libraire dynamique

La librairie A2_bonnet_borde_pinero_basic.so est générée dans le répertoire lib/ . Cette librairie implémente l'interface include/strategy.h . Elle contient une méthode appelée *A2_bonnet_borde_pinero_basic()* et qui retourne la stratégie.

3 Evaluation de la grille

Pour évaluer la grille, nous avons pris en compte quatre paramètres, et à chacun de ces paramètres nous leurs avons donné un coefficient pour leur donner plus ou moins d'importance.

3.1 Le nombre de tiles vides

On compte le nombre de tiles vides que l'on a après avoir joué.

3.2 La plus grande tile

Est la valeur de la tile avec la plus grande valeur.

3.3 La grille la plus progressive possible

Une grille est progressive lorsque la valeur des cases augmente ou descend quelle que soit la direction. Ainsi, 2 - 4 - 8 - 16 est acceptable, tout comme 32 - 8 - 4 - 2. Mais 2 - 8 - 2 - 16 ne l'est pas (à cause de la 3eme case). À chaque mouvement on doit vérifier si le mouvement d'après rendra la grille plus progressive. Si c'est le cas, le mouvement doit être effectué. Si non, trouver un autre mouvement.

3.4 La grille la plus régulière possible

pour pouvoir fusionner, les cases doivent comporter des valeurs identiques. Ainsi une suite 2 - 16 - 64 - 256 respecte la règle de progressivitè) mais aboutira à un échec car on ne pourra fusionner aucune case avec sa voisine. Il faut donc respecter un autre critère : la régularité. Et faire en sorte de ne pas avoir de cassure dans les séries. Entre créer une série 2 - 4 - 8 - 16 et créer une série 16 - 64 - 256 - 1024 on préférera donc la première, même si la possibilité d'obtenir un nombre fort comme 1024 peut être tentante a priori.

4 Réflexion avancée sur l'implémentation de expected max

Expected Max est un algorithme qui consiste à retourner la direction optimale à jouer. Elle est basée sur l'algorithme minimax . Le principe est de donner une valeur à la grille en fonction de plusieurs critères (cf. 3 Evaluation de la grille) ainsi nous jouerons le coup avec le maximum de chances de gagner combiné au coup avec le minimum de chances de perdre. Ici nous avons un exemple avec le morpion :

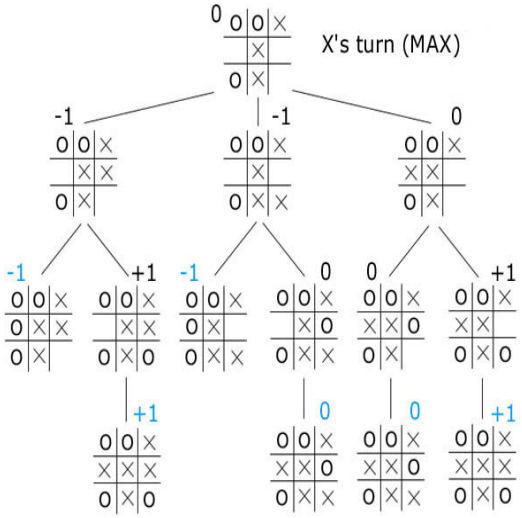

Figure 1: Exemple minimax

Dans la figure 1, on peut voir que le coup le plus intéressant à jouer est le troisième, car si on fait la somme de chaque branche de l'arbre (-2, 1, -2, -1, 0, 2), c'est celle-ci qui offre le plus grand score et la perte la moins importante.

L'une des limites de cette méthode est qu'il faille jouer à deux joueurs, or au 2048 on joue tout seul. C'est pour cette raison qu'on utilisera pas le théorème du minimax mais celui d'Expected Max . En effet ici on concidèrera le second joueur comme l'ordinateur qui remplira de façon aléatoire une des tiles vides. Cela engage de calculer, pour chaque grille et pour toutes les possibilités de remplissage d'une tile vide, la valeur de cette grille. Il faudra ensuite faire une moyenne de toutes les valeurs calculées afin de déterminer quel chemin est le plus intéressant. Pour un algorithme encore plus performant il serait avantageux de pouvoir étendre notre arbre à une profondeur N choisie au début du programme. Il parait évident qu'une telle technique va avoir un coût en mémoire et calcule important. Hors dans la mesure où des limitations en temps d'exécution nous sont imposés il faudra trouver un équilibre entre résultats et performances.