Jeu des sept couleurs: Implémentation et réalisation d'une IA

Antonin Garret
Jimmy Rogala

Mars 2016

Table des matières

1 Règles du jeu **2**

2 Voir le monde en 7 couleurs **2**

3 A la conquête du monde **3**

4 La stratégie de l'aléa **4**

5 La loi du plus fort **4**

6 Les nombreuses huitièmes merveilles du monde **5**

1 Règles du jeu

Nous allons implementer le jeu des 7 merveilles du monde en C.
Les règles du jeu des 7 couleurs sont :

— Un tableau d'une certaine taille est rempli de 7 couleurs aléatoirement.
— La case en bas à gauche (resp : en haut à droite) est de la couleur v (resp : ^) du joueur 0 (resp : 1).
— Chaque tour le joueur 0 (resp : 1) choisit une couleur parmi les 7 couleurs. Toute les cases de la couleur choisie qui sont juxtaposées à une case du joueur 0 (resp : 1) directement ou indirectement prennent la couleur du joueur 0 (resp : 1).
— Le jeu termine quand un joueur a rempli la majorité du tableau : il est le gagnant.

Nous allons d'abord aborder la réalisation du jeu en lui-même (et des choix de l'implémentation) puis parler de la réalisation de plusieurs IA pour ce même jeu.

2 Voir le monde en 7 couleurs

Le langage imposé fut le C. Il permet un contrôle rigoureux de la mémoire.

Question 2.1 :Remplir le monde

Le choix le plus intuitif était d'utiliser un tableau pour représenter le plateu de jeu. C'est ce que nous avons fait. L'autre choix cohérent et possible consistait à définir les cases en tant qu'objet et les voisins des cases mais le C n'est pas pratique pour faire de la programmation orientée objet. Nous avons fait le choix de représenter en interne les couleurs par les nombre de 2 à 9, et les joueurs 0 et 1 par le nombres 0 et le nombre 1. Des fonctions de traduction permettent ensuite d'afficher les couleurs correspondantes pour l'utilisateur.

La fonction *initgame()* permet d'initialiser le plateau de jeu. On remplit le tableau de chiffres de 2 à 9 aléatoirement, puis la case en bas à gauche (resp : en haut à droite) de la couleur du joueur 0 (resp : joueur 1). La fonction est en $\mathcal{O}(n)$ (on notera à partir de maintenant n le nombre de case du tableau)

Question 2.2 : Jouer un coup

Nous avons réaliser la méthode proposée pour jouer un coup. Elle consiste à parcourir le tableau de jeu et à trouver toute les cases de la couleur choisie par le joueur actuel qui sont adjacentes à l'une de ses cases. Si on a mis à jour au moins une case au cours de ce parcours, on effectue un nouveau parcours. On s'arrête lorqu'on à effectué un parcours complet sans changement. Le moyen le plus simple pour tester la correction de cette méthode est de l'appliquer quelques fois sur le tableau de jeu en affichant celui-ci et de vérifier que tout se passe comme prévu.

Dans le pire des cas, on effectuera $n-2$ parcours (si l'on modifie exactement une case à chaque parcours). Comme chaque parcours vérife les n cases du tableau, on a une complexité dans le pire des cas de $\mathcal{O}(n^2)$.

La plupart du temps on effetuera que quelques parcours (car il faut une grande zone de même couleur pour effectuer de nombreux parcours, ce qui est statistiquement peu probable), on a donc une complexité moyenne de $\mathcal{O}(n)$. Ce pendant, il est possible de faire mieux

Question 2.3 : Jouer un coup : meilleure alternative

La methode précedente est peu efficace. La methode choisi consiste à faire comme un front d'onde : On applique la fonction récursivement en commencant par la case du joueur. Si la case est de la couleur du joueur, on réapplique la fonction sur ses voisins. Si la case est de la couleur choisie par le joueur, on change la couleur de la case et on vérifie si la case a des voisins de la même couleur pour réappliquer la fonction sur les voisins. Sinon, on ne fait rien. Nous avons utilisé un autre tableau pour retenir les cases déjà parcourues donc on a une complexité spatiale en $\mathcal{O}(n)$ permettant une complexité temporelle linéaire par rapport aux nombre de cases du joueur après le coup, soit en $\mathcal{O}(n)$ dans le pire des cas.

Un moyen de vérifier si la fonction est correcte est de print le boardTemp (notre tableau permettant d'éviter les boucles infinis). Si celui ci est en adéquation avec l'état du jeu, alors la fonction est probablement correcte.

Question 2.3 : Autre possibilité

On aurait pu (dans l'idée des fonctions de remplissage dans les logiciels de dessins comme paint) peindre les cases du joueur en la couleur voulue, puis appliquer sur la case en bas à gauche (resp. en haut à droite) pour le joueur 0 (resp. 1) une fonction, elle aussi récursive, qui appliquée à une case, si elle est de la couleur choisie, la repeint de la couleur du joueur puis est réappliquée à ses voisins, et ne fait rien sinon. On repeint ainsi les cases voulues de la couleur du joueur, et ceci evite la complexité spatialle en $\mathcal{O}(n)$

3 A la conquête du monde

Le jeu a été fait de sorte que changer l'algorithme qui choisit le coup joué par un joueur (IA simple, IA compliqué ou joueur humain) soit simple. Il nous suffit donc de créer une fonction qui renvoie une couleur en prenant en argument le joueur qui jouer un coup (0 ou 1), sauf si dans le cas de la stratégie aléatoire.

Question 3.1 : Joueur vs Joueur

Une simple boucle qui demande à l'utilisateur de choisir une couleur et qui vérifie qu'il a bien rentré une lettre valide permet à un joueur de jouer. On ne

vide pas le buffer entre chaque coup et un joueur malveillant peut jouer pour l'autre joueur.

Question 3.2 : Condition d'arrêt

Le jeu peut s'arrêter lorsqu'un des deux joueurs possèdent plus de 50% de la surface de jeu. En effet, il peut alors être désigné gagnant car peu importe les coups joués par la suite, son adversaire ne pourra jamais avoir plus de cases que lui. Nous avons implémenté cette condition d'arrêt, mais on pourrait en réalité dans certains cas arrêter le jeu avant qu'elle soit remplie, par exemple lorsqu'un joueur arrive à couper l'accès à une portion du tableau de jeu à son adversaire, et qu'elle est suffisamment grosse pour garantir que quelque soient les coups joués par la suite, il possèdera finalement plus de 50% du terrain.

4 La stratégie de l'aléa

Question 4.1 et 4.2 : Un début d'IA

L'IA la plus simple reste l'aléatoire, la premiere fonction est simple à faire. La deuxième est plus compliqué mais il suffit de garder en mémoire les couleurs quand on passe dessus.

5 La loi du plus fort

Question 5.1 : Enfin une bonne IA

Le passage de l'IA aléatoire pas trop bête à cette fonction est simple : Au lieu de seulement mémoriser les couleurs, on mémorise le nombre de fois qu'on voit la couleur. Une attention particulère est portée au fait que l'on doit vérifier les voisins de chaque case colorée (comme pour jouer un coup). La compléxité est la même que pour jouer un coup.

Question 5.2 : Un monde équitable ?

Quand on fait jouer l'aléatoire contre le glouton, le glouton l'emporte souvent. Quand on fait le glouton contre le glouton, le premier joueur l'emporte le plus souvent (avec une différence moindre par rapport à Aléa VS Glouton). Pour rendre le jeu équilibré, on peut par exemple faire un monde symetrique (symetrie axialle par rapport à la diagonale séparant les 2 joueurs) et d'alterner le joueur 1 et le joueur 2. Ceci est réalisable sur plusieurs parties mais pour ne privilieger aucun joueur sur une seule partie, l'idée a été de forcer un nombre de case proches de chaque couleur pour chaque joeur (qu'il y ait autant de rouge que de vert que de bleu etc). Ceci empecherait le fait qu'un joueur domine parce qu'il commence (mais ceci ne garantie rien, l'algorithme étant aléatoire).

6 Les nombreuses huitièmes merveilles du monde

Les questions suivantes n'ont pas été implémentées. Une expérience de pensée est faite dessus.

Question 6.1 et 6.2 : Un début de Min-Max

La meilleure façon de faire les questions suivantes est de penser en termes d'heuristique et d'arbre : L'hégémonique est comme le glouton, sauf que l'heuristique n'est pas la même. Il suffit de créer une fonction qui calcule le périmètre et de l'utiliser dans un min max de hauteur 1 pour avoir la réponse à la 6.1 (il suffit d'utiliser une hauteur de 2 pour avoir la réponse à la 6.2). La complexité de l'algorithme pour une prévision sur les k prochains coups est en $\mathcal{O}(7^k)$ appels de la fonction *coup*, soit $\mathcal{O}(7^k n)$.

Synthèse du projet

Observations diverses

Les résultats sont bien en adéquation avec l'intuition (contrairement à certains autres jeux).

Ce qui a marché

Le canevas de code permet de se focaliser directement sur les questions qui nous intéressent. Le jeu en lui-même est suffisament simple pour pouvoir finir l'implémentation en quelques heures (pas comme le jeu de dames ou d'echec) et suffisament compliqué pour pouvoir poser des problème intéressants, comme la recherche d'une meilleure heuristique (le morpion par exemple est trop simple).

Ce qu'on peut ajouter

— Pour donner envie de s'attaquer aux problèmes 7.x pourquoi ne pas mettre en place un concours en ligne d'ia sous la forme d'un tournoi mis à jour chaque 5 minutes entre tous nos algorithmes ? (La méthode a été utilisé en sup dans ma prépa et avait eu un franc succés).

— Le projet pourrait être tout simplement posé sous la forme d'un défi : Mettre au point une démarche pour avoir un algorithme efficace dans le jeu des 7 couleurs (Nous expliquer seulement les règles du jeu). Ceci pourrait nous forcer à recréer le jeu pour tester nos algorithmes.