Rapport du projet 7 colors

Marco Freire et Clément Legrand
L3 Info ENS, 2017/2018

1 Introduction

Seven colors est un jeu d'ordinateur inventé par Dmitry Pashkov, développé en 1991. Le principe est simple: deux joueurs essayent de conquérir la plus grande surface d'un terrain de jeu constitué de cases juxtaposées, de sept couleurs différentes. Pour cela, chaque joueur choisit à son tour une couleur, et conquiert toute case de cette couleur adjacente à la zone qu'il possède.

2 Réponses aux questions

2.1 Voir le monde en sept couleurs

Question 1

Il convient tout d'abord de créer un type `boardd` afin d'encapsuler un peu le code et de ne pas modifier le plateau de jeu n'importe comment. Ce type contient un tableau de caractères de taille `BOARDSIZE`^{2}, ainsi que deux entiers: `numcellsup` et `numcellsdown` indiquant le nombre de cases possédées par le joueur ^ et le joueur v respectivement. Afin de maintenir cette structure, nous avons implémenté les méthodes `getcell`, `getnumcellsup`, `getnumcellsup`, `setcell`, ainsi que `boardcreate` et `boardfree`, allouant un tableau de la bonne taille, et le libérant respectivement.

Question 2

Complexité dans le pire cas : Dans le pire des cas, à chaque itération, l'algorithme met à jour une et une seule case. Posons $n = BOARDSIZE$. Étant donné qu'il y a n^2 cases dans le tableau, au plus n^2 passages sont effectués. Lors d'un passage, n^2 cases sont examinées, chacune d'elles en un temps constant. La complexité dans le pire cas de cet algorithme est donc un $O(n^4)$. Cette borne est réellement atteinte: par exemple si une couleur forme une spirale rectangulaire jusqu'au centre. La complexité est donc un $\Theta(n^2)$.

Complexité amortie : Le cas énoncé précédemment est assez pathologique et ne peut se produire trop souvent. L'analyse de la complexité amortie est donc pertinente.

Question 3 (bonus)

Principe de l'algorithme implémenté : Plutôt que d'effectuer des parcours entiers du tableau en vérifiant si les cases ont besoin d'être mises à jour, il est préférable de se représenter la zone conquise par un joueur comme un graphe (non orienté), dont les sommets sont les toutes les cases du tableau et les arêtes lient chacune des cases conquises par le joueur aux quatre cases adjacentes sur le plateau. Il suffit alors d'effectuer un parcours de ce graphe en le mettant a jour progressivement.

Nous créons pour cela un second tableau `boardvisited`, de mêmes dimensions, dont les cases indiquent si le sommet (i, j) a été visité ou pas. Nous parcourons ensuite récursivement le graphe en commençant par la case du coin, qui appartient nécessairement au joueur. À chaque sommet visité, nous le marquons comme tel dans `boardvisited`, et s'il est de la bonne couleur ou appartient au joueur courant, nous marquons la case comme appartenant au joueur dans `board` et explorons ses voisines non visitées.

Complexité dans le pire des cas : Chaque case du tableau est visité un nombre fini borné de fois (au plus quatre dans notre implémentation), donc l'algorithme est un $O(n^2)$. Là encore, cette borne peut être atteinte si tout le tableau est de la même couleur. La complexité est donc un $\Theta(n^2)$.

2.2 À la conquête du monde

Question 4

Les fonctions nécessaires pour faire jouer un joueur contre un autre sont `playerinput` et `changeplayer`; l'implémentation contraint les joueurs à s'affronter sur la même machine.

Question 5

La partie peut s'arrêter si l'un des joueurs au moins détient plus de la moitié du plateau de jeu. Nous avons donc recours aux attributs `numcellsup` et `numcellsdown` du type `boardd`, afin de savoir si ceux ci excèdent la moitié du nombre total de cases.

2.3 La stratégie de l'aléa

Question 6

Question 7

2.4 La loi du plus fort

Question 8

Nous allons pour cela reprendre et modifier légèrement la fonction `updateboardoptimized` impémentée pour la question 3, mais en comptant le nombre de cases modifiables en jouant une couleur donnée au lieu de les modifier. Nous effectuons ensuite le choix maximisant ce nombre de cases.

Question 9

Pour que le combat soit "équitable", le mieux est de faire s'affronter les deux joueurs artificiels sur un plateau donné, puis d'effectuer l'affrontement à nouveau, mais en échangeant les positions de départ.

Question 10

2.5 Les nombreuses huitièmes merveilles du monde (bonus)

Question 11

Il apparaît facilement qu'en effectuant un parcours entier du plateau et en comptant les cases ayant parmi leurs voisines, une appartenant au joueur courant, il est possible d'obtenir le périmètre de la zone détenue par le joueur en $\Theta(n^2)$. Il suffit dès lors, pour chaque couleur possible, de créer une copie du plateau, d'effectuer sur celle- ci une mise à jour en jouant ladite couleur, et de compter le périmètre correspondant. La complexité de cet algorithme est donc un $\Theta(n^2)$.

Question 12

Une des manières d'implémenter le glouton prévoyant est d'effectuer pour chacune des couleurs une copie du plateau de jeu, puis de modifier celui-ci en jouant cette couleur, en comptant le nombre de cases ainsi gagnées. Puis d'utiliser la fonction de la question 8 pour chacune des couleurs afin d'obtenir le nombre maximal de cases gagnées en ayant fixé le premier coup. On choisit ensuite le premier coup afin de maximiser ce nombre. Cet algorithme effectue 7 copies du plateau en $\Theta(n^2)$, fait sept appels à `updateboardoptimized` en $\Theta(n^2)$ et fait appel 49 fois à la fonction `calcnewcellsoptimized` qui s'exécute en $\Theta(n^2)$. Il s'exécute donc en $\Theta(n^2)$. Si on veut explorer m coups consécutifs, cette complexité passe à un $\Theta(\exp(m) * n^2)$, car il faut alors effectuer 7 fois plus de copies du tableau pour chaque coup additionnel (sauf le dernier ou il est possible d'utiliser `calcnewcellsoptimized`).

2.6 Le pire du monde merveilleux des sept couleurs (bonus)

Question 13

3 Conclusion

A Annexe

A.1 Bibliographie

Ne sont cités ici que les ouvrages et sites dont nous nous sommes le plus servi.

[1] N. SCHABANEL : Algorithmes d'approximation et algorithmes randomisés. http://pauillac.inria.fr/~quercia/documents-info/Luminy-2003/schabanel/schabanel.pdf, may 2003. Cours donné lors du Stage de Luminy 2003 à destination des enseignants d'informatique de CPGE.