Projet Caméléon [1]
Programmation en JAVA 1.7 et en binôme (obligatoirement)

1 Objectifs

Le but du projet est d'implémenter le jeu décrit ci-dessous, entre un humain $J1$ et l'ordinateur $J2$. L'humain a la couleur rouge et l'ordinateur a la couleur bleu. Le gagnant est le joueur qui, lorsque le plateau est rempli, a colorié le plus de cases avec sa couleur.

2 Description du jeu

Le jeu se déroule sur un plateau carré qui fait 3×2^n cases de côté (c'est-à-dire 3×2^n lignes et 3×2^n colonnes), que l'on imagine découpé successivement en régions carrées de plus en plus petites, comme pour construire un quadtree :

- Le plateau est lui-même une région carrée de 3×2^n cases de côté.
- Il est découpé en quatre régions carrées, chacune de $3 \times 2^{n-1}$ cases de côté.
- Et ainsi de suite, récursivement, chaque région carrée de 3×2^k de côté, avec $k \geq 1$, est découpée en quatre régions carrées de $3 \times 2^{k-1}$ cases de côté.
- Le découpage s'arrête aux régions carrées de côté 3.

Le terme de *région* utilisé par la suite fait référence exclusivement aux régions carrées à 3×2^k cases de côté obtenues par ce découpage, avec $k = 0, 1, 2, \ldots, n$. Une région est *petite* si elle a 3 cases de côté et *grande* dans les autres cas.

Les colonnes du plateau sont numérotées de 1 à 3×2^n de gauche à droite, et les lignes sont numérotées de 1 à 3×2^n de haut en bas. Une case est identifée par ses coordonnées (i, j), où i est le numéro de sa ligne et j le numéro de sa colonne. Elle appartient à exactement une petite région. Le *voisinage* d'une case est l'ensemble d'au plus 8 cases qui ont soit un côté soit un coin en commun avec cette case.

Nous considérons que le plateau initial est soit vide, soit partiellement colorié de sorte qu'il y ait autant de cases rouges que de cases bleues. Les autres cases sont supposées blanches. Si une région entière, quelle que soit sa taille, est coloriée de la même couleur, elle est dite *acquise* (par le joueur qui possède cette couleur). Toutes ses cases sont alors dites *acquises*. A noter qu'il s'agit (quand-même) d'une acquisition temporaire.

Le joueur J1 commence le jeu. Ensuite, tour à tour, chaque joueur J choisit une case blanche (i, j), ce qui déclenche le changement de couleur de plusieurs cases. Les règles de recoloriage définissent la version du jeu, Caméléon Brave ou Caméléon Téméraire.

Règles pour le Caméléon Brave :

R_1. La case (i, j) prend la couleur du joueur J.
R_2. Toute case qui est voisine de la case (i, j) et qui est déjà coloriée prend la couleur du joueur J.

1. Les caméléons carrés n'étant pas légion, vous pouvez lire cette histoire (en français) qui vous changera les idées : https ://www.ebooksgratuits.com/details.php ?book=912

Règles pour le Caméléon Téméraire :

R_1. La case (i, j) devient de la couleur du joueur J.

R'_2. (attention, changement) Toute case qui est voisine de la case (i, j) et qui est déjà coloriée mais n'est pas acquise devient de la couleur du joueur J.

R_3. Si la case (i, j) était la dernière case non-coloriée d'une petite région, alors la petite région est acquise par le joueur J et toutes ses cases deviennent de la couleur du joueur J.

R_{4-5}. Supposons qu'une région P_k de 3×2^k cases de côté, avec $k \geq 0$, du plateau vient d'être acquise par un joueur X, quel qu'il soit (J ou son adversaire). Soit P_{k+1} l'unique région de $3 \times 2^{k+1}$ cases de côté qui contient P_k. Alors :

R_4. si P_{k+1} est totalement coloriée, et le joueur X a acquis au moins deux des quatre sous-régions de taille 3×2^k de P_{k+1}, alors toute la région P_{k+1} est acquise par le joueur X et toutes ses cases deviennent de la couleur du joueur X.

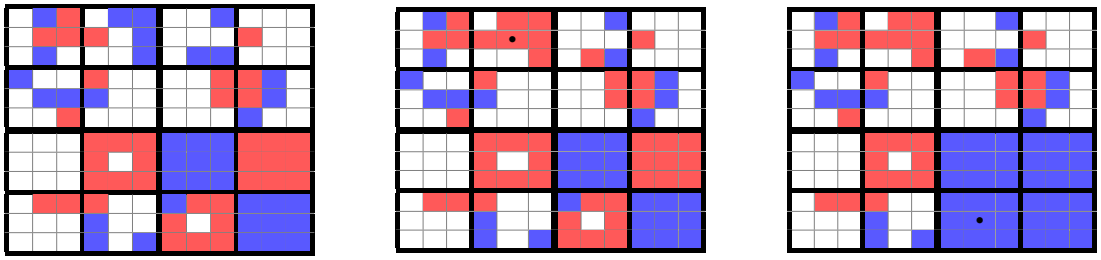
R_5. Si P_{k+1} est totalement coloriée, et le joueur X n'a acquis qu'une des quatre sous-régions de taille 3×2^k de P_{k+1}, alors toute la région P_{k+1} est acquise par l'adversaire de X et toutes ses cases deviennent de la couleur de l'adversaire de X.

Observation. Notez ici que si l'une des règles R_4 ou R_5 a été appliquée pour une région P_k, alors P_{k+1} est acquise par un joueur et donc il faut vérifier à nouveau, mais cette fois-ci pour P_{k+1}, si les règles R_4 ou R_5 s'appliquent. Et cela récursivement tant qu'une nouvelle région plus grande est acquise par un joueur.

Exemple. Soit le plateau de gauche ci-dessous.

Versions Brave. Si le joueur J1 (rouge) choisit la case (2,5), les règles R_1 et R_2 s'appliquent et le plateau devient celui du milieu (le • indique la case choisie). Si, à la place, il choisit la case (6,8), la case (6,8) et les trois cases bleues situées en dessous, qui sont ses voisines, deviennent rouges (non montré sur le dessin).

Version Téméraire. Si, sur le plateau du milieu, le joueur J2 (bleu) joue la case (6,12) alors la case (6,12) devient bleue mais aucun autre changement n'est effectué, puisque les deux cases rouges situées en dessous, qui sont ses voisines, sont acquises par J1 (non montré sur le dessin). Par contre si, sur le plateau du milieu, le joueur J2 joue la case (11,8) alors le plateau devient celui de droite. La règle R_3 s'applique d'abord et le joueur J2 récupère la petite région à l'intersection des lignes 10 à 12 avec les colonnes 7 à 9. Ensuite, la règle R_4 s'applique et le joueur J2 acquiert la région plus grande, celle de 3×2 cases de côté, puisqu'il détient 3 sous-régions de cette région. Les images ne le montrent pas, mais si les trois autres régions de 3×2 cases de côté du plateau étaient rouges, alors tout le plateau deviendrait rouge par la règle R_5 !



3 Travail demandé

En bref, le programme que vous allez écrire doit permettre de :

- récupérer le plateau initial selon les formalités définies plus bas.
- représenter le plateau à l'écran (une forme basique suffit amplement).
- reporter sur le plateau chaque coup joué par l'humain ou l'ordinateur, et le coloriage qui s'en suit, selon les deux variantes Brave *et* Téméraire.
- implémenter plusieurs stratégies de jeu pour l'ordinateur, en suivant les consignes ci-dessous.

Les détails sont fournis ci-dessous.

Sur tous les aspects du jeu, on visera d'abord à **minimiser la complexité en temps, au pire**, mais l'espace mémoire devra également être utilisé avec parcimonie. Il faudra suivre les consignes suivantes :

C1. Pour la complexité en temps, vous choisirez (sauf lorsque c'est imposé) les structures de données le plus efficaces, afin que pour chaque tâche de l'algorithme la complexité minimum soit garantie. Les structures de données peuvent être de plusieurs types, adaptés à vos besoins, pourvu que le but d'efficacité soit atteint.

C2. Pour stocker le plateau et gérer une partie des informations le concernant, vous pouvez utiliser un tableau. Par contre, le parcours du tableau sans raison précise et bien fondée est à proscrire. Si la tâche souhaitée peut être accomplie en ne parcourant qu'un sous-tableau bien ciblé et/ou une structure de données plus efficace stockant les informations utiles à la réalisation de cette tâche, vous devez le faire.

Dans chacune des variantes du jeu, il faut implémenter la stratégie gloutonne suivante, pour le joueur J2 (l'ordinateur).

Stratégie gloutonne. Par définition, une stratégie gloutonne est une stratégie qui vise le gain immédiat maximum, sans vision à long terme. Pour notre jeu, cette stratégie se définit comme suit : lorsque son tour arrive, le joueur J2 choisit la case blanche qui lui permet, une fois que les règles du jeu ont été appliquées et que son tour est fini, d'avoir un score maximum, où le score est défini par :

score(J2) = nombre de cases bleues - nombre de cases rouges.

Vous devez :

- gérer les cases, et le score que J2 peut atteindre s'il choisit chaque case blanche, à l'aide de structures de données de sorte que la complexité *au pire* de chaque coup joué par J2 (incluant le choix de la case à jouer, la mise à jour du plateau et la mise à jour des structures de données) soit aussi petite que possible.
- pour la version Brave, choisir les structures de données adaptées.
- pour la version Téméraire, en plus des structures pour la variante Brave, utiliser un quadtree pour gérer les régions acquises ou non, et évaluer le score atteignable par chaque case blanche. Dans ce quadtree, les feuilles de l'arbre seront les régions acquises les plus grandes (non-incluses dans d'autres régions acquises) et les petites régions qui ne sont pas acquises. Ces dernières pourront contenir des informations sur les cases blanches à l'intérieur de la région concernée.

Deuxième stratégie. Pour la version Téméraire, vous proposerez une deuxième stratégie pour J2, aussi intelligente que possible, et vous l'implémenterez suivant les mêmes consignes C1 et C2.

Vous devez également proposer un menu, demandant à l'utilisateur ce qu'il veut faire (le programme appelera ensuite les méthodes nécessaires pour répondre à sa demande) :

- s'il veut démarrer par un plateau vide, ou déjà partiellement rempli. Dans le deuxième cas, le plateau sera fourni dans un fichier texte, dont le nom ne doit pas être imposé, de la forme suivante : la première ligne contient le nombre de cases par côté. Les lignes suivantes contiennent les lignes du plateau, où chaque case est représentée par R/B/A selon qu'elle est rouge, bleue ou blanche. Par exemple, pour le plateau fourni sur la figure précédente, à gauche, le fichier contiendra :

  12
  ABRABBAABAAA
  ARRRABAAARAA
  ABAAABABBAAA
  BAARAAAARRBA
  ABBBAAAARRBA
  AARAAAAAABAA
  AAARRRBBBRRR
  AAARARBBBRRR
  AAARRRBBBRRR
  ARRRAABRRBBB
  AAABAARARBBB
  AAABABRRRBBB

- s'il veut afficher le plateau actuel (vous pouvez oublier cette option si vous affichez le plateau en permanence).
- s'il veut jouer, et avec quelle stratégie pour l'ordinateur.
- s'il veut afficher le score de chaque joueur ; l'utilisateur doit pouvoir connaître le score à tout moment du jeu, pas seulement à la fin.
- s'il veut connaître l'évaluation d'une case blanche, telle que calculée par J2 pour son propre usage [2] ; l'utilisateur doit pouvoir faire cette demande à tout moment du jeu, pas seulement à la fin.

Enfin, votre programme doit contenir, parmi d'autres fonctions que vous implémenterez, les fonctions suivantes (dont les noms doivent se retrouver tel quels dans votre programme, de sorte qu'un lecteur de votre code puisse aller directement à la fonction qui l'intéresse) :

RemplirPlateau qui saisit le plateau fourni en entrée.
JouerGloutonBrave, JouerGloutonTemeraire qui implémentent la stratégie gloutonne pour chacune des variantes du jeu.
JouerIATemeraire qui implémente votre stratégie intelligente.
EvalCaseBrave, EvalCaseTemeraire qui évaluent le score que J2 peut atteindre, selon la version, s'il choisit la case passée en paramètre.
RemplirRegion qui modifie le quadtree selon les règles indiquées (R_3, R_4, R_5) lorsque la dernière case d'une petite région est remplie.
CalculeScore qui calcule le score du plateau actuel.

Vous choisirez également des noms clairs pour vos structures de données : *Quadtree* et/ou *ABR* et/ou *AVL* etc.

Le programme sera lancé en ligne de commande et pourra être utilisé indépendemment d'Eclipse ou autre IDE.

2. c'est promis, on ne trichera pas, on veut juste savoir !

4 Rendu de TP

Les programmes doivent être implémentés par vous-mêmes en Java 1.7 et doivent fonctionner parfaitement sur les machines du CIE, sous Linux. L'appel à des structures de données optimisées existantes (du genre TreeSet) est interdit : vous devez implémenter vous-mêmes vos structures de données. Seuls les tableaux et les listes chaînées peuvent être utilisés tels que fournis par Java. Aucun appel à une méthode mise à disposition dans les bibliothèques Java (par exemple un tri d'une liste) n'est autorisé si vous ne connaissez pas le fonctionnement de la méthode et sa complexité (ce que vous montrerez dans les calculs de complexité demandés ci-dessous).

Un rapport d'au maximum 12 pages sera fourni, incluant :

- les commandes de compilation et exécution en mode console
- la description du jeu, les difficultés rencontreés, les choix effectués (y compris ceux liés aux structures de données)
- une vue globale de votre programme, sous la forme d'un algorithme (c'est-à-dire en pseudo-code) dont les instructions sont des appels à des procédures/fonctions ; la spécification et les paramètres de chaque procédure/fonction seront précisément décrits.
- pour chaque méthode parmi celles de la liste demandée :
  * le détail de la procédure/fonction, sous forme algorithmique (c'est-à-dire en pseudo-code) ;
  * l'explication de son fonctionnement ;
  * sa complexité
- la complexité au pire d'un coup joué par J2, dans chacune des versions
- la complexité globale de l'algorithme permettant de jouer jusqu'à ce que le plateau soit rempli, dans chacune des versions, en supposant que chaque coup de J1 est fait en $O(1)$.
- des jeux de données commentés, mettant en lumière des cas où J1 ou J2 gagne, en fonction des stratégies utilisées.

Important

- Les fichiers du programme, ainsi que les jeux de données, seront mis dans un répertoire appelé Nom1Nom2 (où Nom1 et Nom2 sont les noms des deux étudiants du binôme). Ce répertoire sera ensuite archivé sous le nom Nom1Nom2.**zip**. L'archive sera déposée sur Madoc, au plus tard le **samedi 4/12/2021 à 23h55** (Madoc n'acceptera pas de retard).
- La dernière séance est consacrée à une démo de 10 minutes par projet, pour laquelle des plateaux vous seront fournis en entrée. Vous devez donc impérativement suivre les consignes concernant le format du fichier en entrée. Cette dernière séance **n'est donc pas une séance de travail sur le projet**.
- *Tout* est important pour la notation. En particulier, il sera accordé beaucoup d'attention au respect des consignes et à la recherche d'une complexité minimum - garantie d'une efficacité maximum - pour votre méthode, via la mise en place des structures de données les plus adaptées.