

***L3 MP:X31I020***

*Algorithmique et Structures*

*de données 3*

*Projet*

*2021\_2022*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ATALLA Salim

NISO Aram

L3 – Informatique\_Groupe\_584I

Avant de commencer le rapport de notre projet, voici les commandes qui peuvent compiler et exécuter le code enfin pour qu’il puisse être mis en œuvre en mode console :

• javac GUI.java Plateau.java Region.java Brave.java Temeraire.java Joueur.java Case.java

Fichier.java

• jar -cvf GUI.jar GUI.class Plateau.class Region.class Brave.class Temeraire.class Joueur.class Case.class Fichier.class

• java -cp ./GUI.jar GUI :

## Projet Caméléon :

Nous avions à travailler sur un projet en Java, le projet consistait d’implémenter un jeu qui contient une IA simple enfin pour que l’utilisateur puisse jouer contre l’ordinateur.

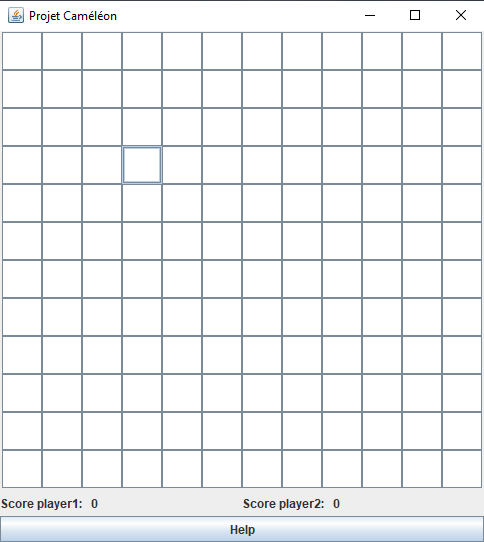
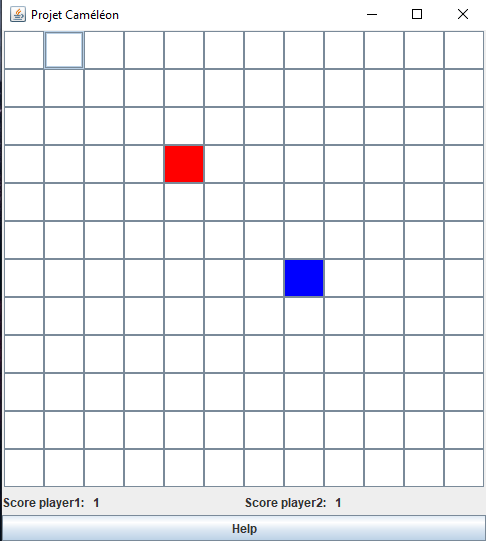
Le projet était divisé en plusieurs parties enfin pour nous faciliter les taches à faire et produire.

Le but de ce projet était de mettre en œuvre les connaissances requises en cours, notamment l’utilisation des arbres, qui était la base de notre projet, mais aussi pour nous aider à nous améliorer en programmation JAVA et surtout comment choisir des structures de données qui minimisent la complexité de nos méthodes et nos fonctions.

Nous allons voir cela en détaille en parcourant le rapport.

Tout d’abord voici la description du jeu implémenté :

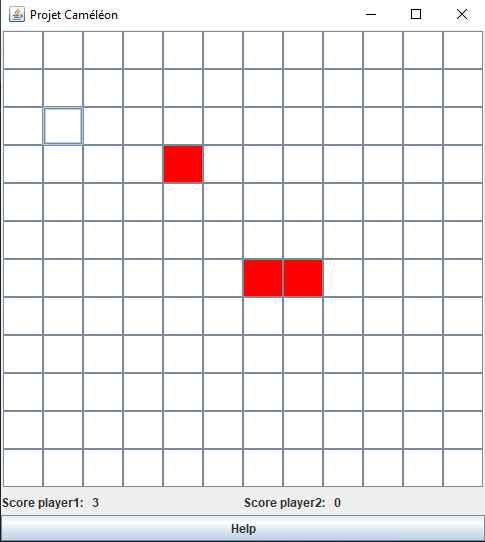
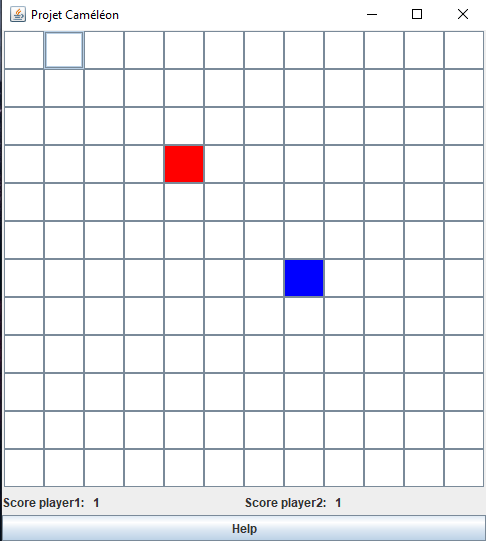
En amont, le but de jeu est de faire jouer un utilisateur humain contre un ordinateur, la surface graphique de jeu est un simple plateau qui contient des cases, auxquelles on va appliquer une coloration, Rouge pour l’utilisateur Humain et Bleu pour l’ordinateur. Le jeu se termine lorsque toutes les cases seront coloriées, à la fin le joueur qui aura colorié le plus des cases alors il gagnera.



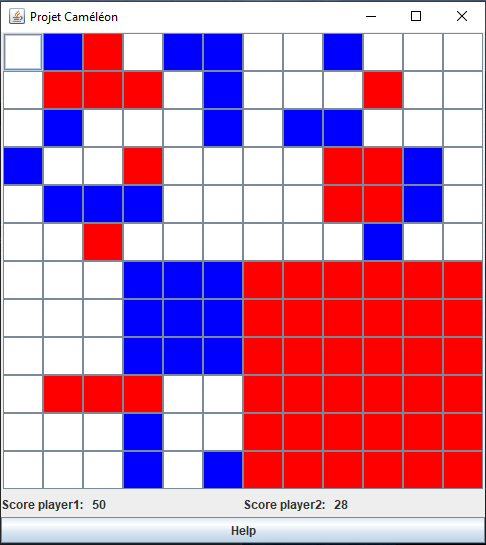
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Voici les détailles du jeu :  
- Le jeu est implémenté en deux versions :   
La première et la plus simple est appelée Brave, en effet, dans cette version, les deux joueurs jouent simplement le jeu, chacun des deux joueur à son tour colorie une case, et lorsqu’il n’y a plus aucune case blanche alors le score détermine le gagnant.

>> Cette version possède une seule règle, toute case qui est sur le point d’être colorier et qui a des cases voisines déjà coloriées, alors les cases voisines deviennent de la même couleur que elle, si et seulement si elles sont déjà coloriées mais bien si elles sont blanches alors elles restent blanches.



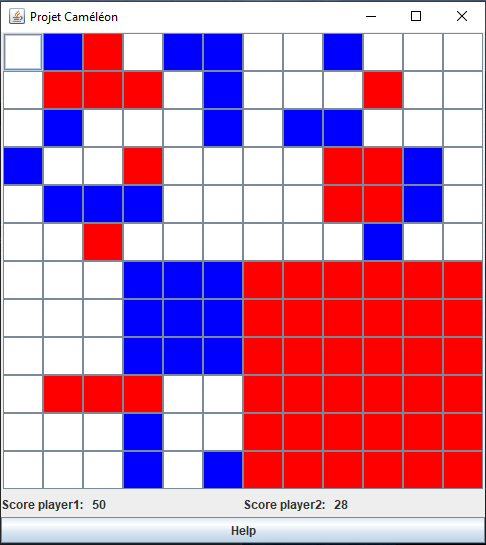
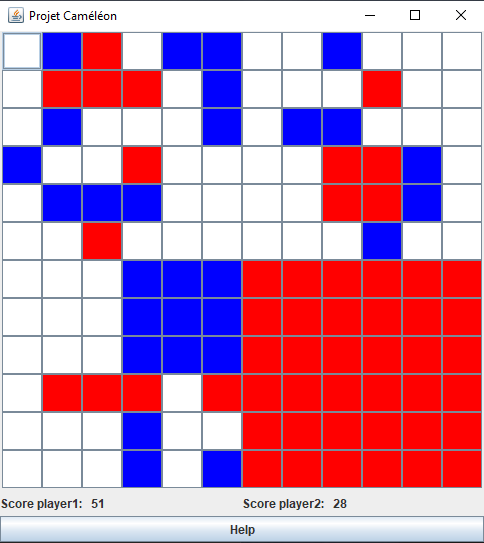
La deuxième qui est un peu plus complexe est appelé Téméraire, en effet, dans cette version, les deux joueurs jouent sur le plateau mais avec plusieurs règles qui définissent cette version.



Tout d’abord il faut savoir que dans cette version le plateau est divisé dans ce que l’on appelle des régions, il y a des petites régions s’appellent feuilles mais aussi des grandes régions, comme illustré dans les photos ci-dessus :

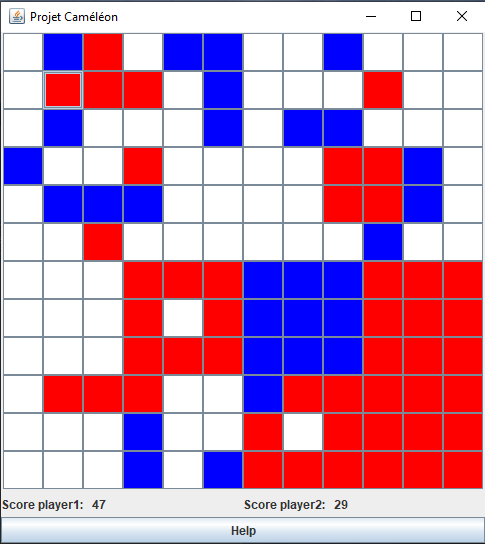
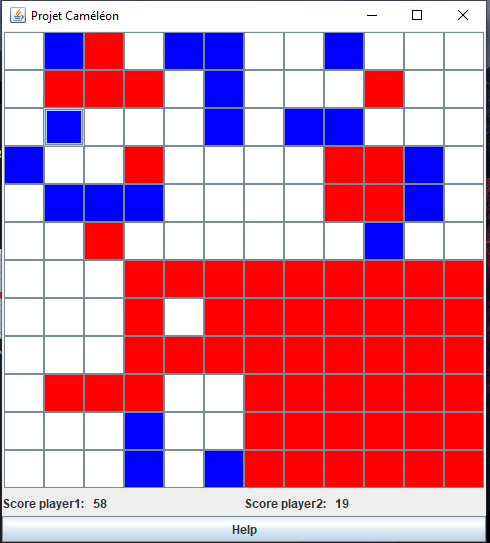
Grande règion  
Petite règion (Feuille)

• La première règle est à peu près comme dans la version Brave, toute case qui est sur le point d’être colorier et qui a des cases voisines déjà coloriées, alors les cases voisines deviennent de la même couleur qu’elle, si et seulement si elles sont déjà coloriées et que leur région n’est pas acquise mais bien si elles sont blanches alors elles restent blanches, ou bien si leur région est acquise alors leur couleur ne change pas.

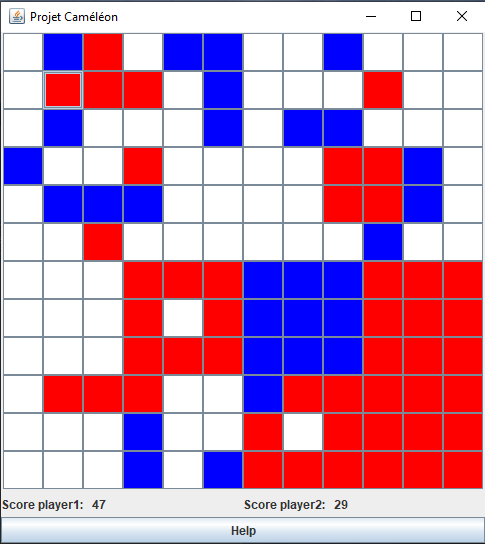
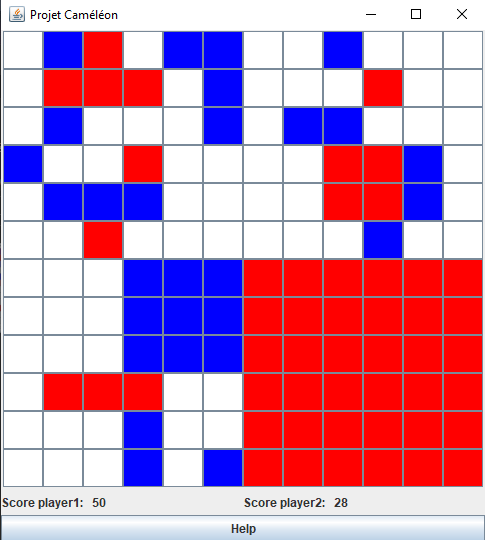


• Si une petite région est complètement coloriée par une même couleur alors elle sera appelée petite région acquise, donc les cases ne peuvent plus être prises par l’autre joueur.

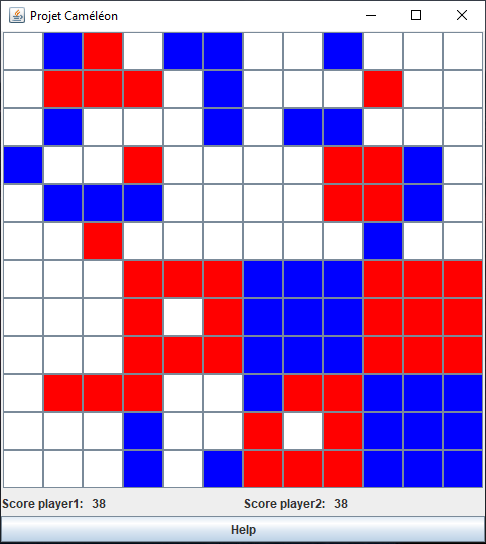
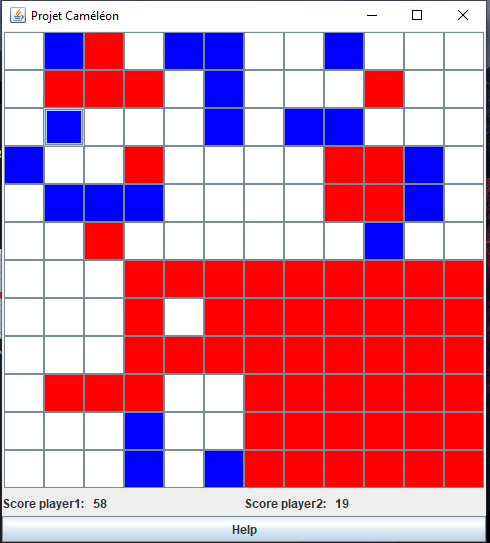
• Si un joueur J possède 3 petites régions dans une grande régions alors toutes les petites régions de ces grandes régions deviennent de la couleur du joueur J une fois toutes les cases jouées, et alors cette grande région devient grande région acquise, donc les cases ne peuvent plus être prises par l’autre joueur.



• Si un joueur J joue la case du milieu d’une petite région, et que cette dernière est la dernière case dans cette petite région alors, toute la petite région sera prise par le joueur J.



• Si un joueur J possède la moitié d’une grande région et que l’autre joueur possède l’autre moitié, alors le dernier joueur qui a joué dans cette grande région possédera toute la grande région.



Les différentes stratégies du jeu :  
En fait, le jeu peut être jouer en deux stratégies, celle-ci est choisie par l’utilisateur au début du jeu.

• La première est appelée la stratégie Gloutonne. Cette stratégie est appliquée pour les deux versions du jeu. Elle est simple parce qu’elle vise le gain immédiat, en effet elle regarde toutes les cases blanches et puis elle calcule la case qui lui rapporte le plus des points, et elle la joue, donc elle ne regarde pas du tout au long terme, pour acquérir une région ou protéger une région par exemple.

• La deuxième est appelée la stratégie ‘IA’ ou même la stratégie ‘intelligente’. Cette stratégie est appliquée que pour la version Téméraire. Elle est censée être plus intelligente que la stratégie gloutonne, donc elle est appliquée sur la version Téméraire enfin pour calculer la meilleure case en fonction des Régions acquises ou qui peuvent être acquises prochainement. Donc on peut dire que cette stratégie vise le long terme.

Le choix des structures des données :

Les deux versions du jeu ont été implémentées avec la même structures des données, les deux versions demandent les mêmes calcules, mais la version Téméraire a plus de contraintes par rapport à la version Brave, en plus la version Téméraire a objet en plus qui est la Région, que l’on peut l’imaginer comme les longitudes et les latitudes sur le Plateau qui gère les contraintes de la version Téméraire.

Le plateau a été implémenté avec un tableau, celui-ci est une super classe pour les classes Brave et Téméraire donc les deux versions utilisent des tableaux.

Mais la Région a été implémenté avec un QuadTree pour minimiser la complexité, et il faut noter que le Région est un objet de la classe Téméraire.

Les différentes classe et méthodes du code :

Classe Plateau : responsable du plateau. O(n)

Rôle : Initialiser le plateau vide ou rempli partiellement.  
Paramètre : initVide Boolean  
Complexité : O(n)  
Procédure RemplirPlateau (boolean initVide) ;  
  
Rôle : Remplir le plateau partiellement.  
Paramètre : Rien  
Complexité : O(log n )  
Procédure RemplirPlateauPartiel () ;  
  
Rôle : Si la case est valide (existe dans le plateau).  
Paramètre : c Case   
Complexité : ϴ(1)  
Fonction estCaseExiste (Case c) : Boolean ;  
  
Rôle : Si la case vide.  
Paramètre : c Case  
Complexité : ϴ(1)  
Fonction estCaseLibre (Case c) : Boolean ;  
  
Rôle : Ajouter un point au score du joueur passé en paramètre.  
Paramètre : J Joueur   
Complexité : ϴ(1)  
Procédure addPoint (Joueur J) ;  
  
Rôle : Enlever un point du score de l'autre joueur.  
Paramètre : J Joueur   
Complexité : ϴ(1)  
Procédure suppAutrePoint (Joueur J) ;  
  
Rôle : Retourner le score du joueur passé en paramètre.  
Paramètre : J Joueur   
Complexité : ϴ(1)  
Fonction CalculeScore (Joueur J):entier ;

Rôle : Colorier la case séléctionnée.  
Paramètre : c Case , J Joueur   
Complexité : ϴ(1)  
Procédure colorier\_case (Case c, Joueur J)  
  
Rôle : Supprimer la case choisit de la liste des cases vides.  
Paramètre : c Case   
Complexité : O(n) où n est le nombres de case vide  
Procédure supp\_case (Case c) ;  
  
Rôle : L'affichage du plateau sur le console.  
Paramètre : Rien   
Complexité : ϴ(n)  
Procédure afficherPlateau () ;  
  
Rôle : Affichage le score des joueurs à chaque coup.  
Paramètre : Rien   
Complexité : ϴ(1)  
Procédure afficherScore () ;

Classe Region : responsable des régions sur le plateau. O(n)

Rôle : créer les régions  
Paramètre : p Plateau  
Complexité : O(m) où m = log2(sqr(n)/3) donc ϴ(1)  
Fonction diviseRegion (Plateau p):Region ;  
  
Rôle : chercher la feuille qui contient la case c   
Paramètre : c Case, longueur\_racine entier   
Complexité : O(m) où m = log2(sqr(n)/3) donc ϴ(1)  
Fonction rechercheFeuille (Case c, int longueur\_racine): Region ;  
  
Rôle : chercher la région de taille longueur et qui contient la case c  
Paramètre : c Case , longueur entier , longueur\_racine entier   
Complexité : O(m) où m = log2(sqr(n)/3) donc ϴ(1)  
Fonction recherche (Case c, int longueur, int longueur\_racine) : Region ;  
  
Rôle : Vérifier si une feuille est acquise.  
Paramètre: p Plateau   
Complexité : ϴ(1) car 9 itération au pire et tout le temps   
Fonction estFeuilleAcquise (Plateau p) : Boolean ;  
  
Rôle : Vérifier si une grande région est acquise.  
Paramètre : p Plateau  
Complexité : O(m) où m = log2(sqr(n)/3) donc ϴ(1)  
Fonction estAcquise (Plateau p) : Boolean ;

Rôle : Colorier une région lorsqu’elle devient acquise.  
Paramètre : p Plateau , J joueur  
Complexité : O(n) où n est la taille de la région  
Procédure ColorierRegion (Plateau p, Joueur J) ;  
  
Rôle : vérifier si la région est bien remplie.  
Paramètre : p Plateau  
Complexité : O(n) où n est la taille de la région  
Fonction estRemplie (Plateau p) : Boolean ;  
  
Rôle : colorier les régions en appliquant les règles 4 et 5 du jeu.  
Paramètre : p Plateau , J joueur  
Complexité : Ω(n) O(n)  
Fonction RemplirRegion (Plateau p, Joueur J) : Region ;  
  
Rôle : Afficher la région sur le console.  
Paramètre : p Plateau  
Complexité : O(n) où n est la taille de la région  
Procédure afficherRegion (Plateau p) ;  
  
Classe Temeraire : responsable de la version Téméraire et extends de la classe plateau. O(n)

Rôle : Retourner le nombre des cases qui peut J gagné s'il choisit la case c.  
Paramètre : c Case , J joueur  
Complexité : ϴ(1) car négligeable  
Fonction EvalCase (Case c, Joueur J) : entier ;  
  
Rôle : Chercher la case qui peut gagne le plus pour le joueur J en paramètre.  
Paramètre : J Joueur  
Complexité : O(n) où n est le nombre de cases vide.  
Fonction chercher\_meilleur\_case (Joueur J) : Case ;  
  
Rôle : Jouer Glouton avec Brave.  
Paramètre : c Case , J Joueur  
Complexité : O(n) où n est le nombre de cases vide.  
Fonction JouerGlouton (Case c, Joueur J):Case ;  
  
Rôle : Retourner le nombre des cases que le joueur J peut gagner s'il choisit la case c, de manière plus intelligent de EvalCase().  
Paramètre : c Case , J Joueur  
Précondition : Rien  
Complexité : on peut le négliger aussi ϴ(1)  
Fonction EvalCaseIA (Case c, Joueur J) : entier ;  
  
Rôle : Chercher la meilleur cases en utilisant l'évaluation intelligent EvalCaseIA().  
Paramètre : J Joueur  
Complexité : O(n) où n est le nombre des cases vide.  
Fonction chercher\_meilleur\_case\_IA(Joueur J):Case ;

Rôle : Jouer Temeraire avec la recherche intelligent.  
Paramètre : c Case , J Joueur  
Complexité : O(n) où n est le nombre des cases vide.  
Fonction JouerIATemeraire (Case c, Joueur J):Case ;  
  
Rôle : Applique le coloriage de la case séléctionnée avec les changements.  
Paramètre : c Case , J Joueur  
Complexité : O(n)  
Fonction Colorier (Case c, Joueur J):Boolean ;

Classe Brave : responsable sur le version Brave et extends de la classe Plateau. O(n)

Rôle : Retourner le nombre des cases qui peut J gagné s'il choisit la case c.  
Paramètre : c Case , J Joueur  
Complexité : ϴ(1)  
Fonction EvalCase (Case c, Joueur J):entier ;  
  
Rôle : Chercher la case qui peut gagner le plus pour le joueur J en paramètre.  
Paramètre : J Joueur  
Complexité : O(n) où n est le nombre des cases vide.  
Fonction chercher\_meilleur\_case(Joueur J): Case ;  
  
Rôle : Jouer Glouton avec Brave.  
Paramètre : c Case , J Joueur  
Complexité : O(n) où n est le nombre des cases vide.  
Fonction JouerGlouton (Case c, Joueur J) : Case ;  
  
Rôle : Applique le coloriage de la case sélectionnée avec les changements.  
Paramètre : c Case , J Joueur  
Complexité : ϴ(1) mais on fait un affichage de plateau donc O(n)  
Fonction Colorier (Case c, Joueur J): Boolean;  
  
Il y a aussi les classes :

- Case : responsable sur les cases. ϴ(1)  
-Fichier : responsable pour ouvrir un fichier. O(n)  
-Joueur : responsable sur les joueurs. ϴ(1)  
Ces classes n’ont pas de méthodes mais juste le constructeur.

**La complexité totale est donc O(n)**

Enfin il y a la classe Gui, cette classe est responsable et gère l’interface graphique du jeu.

Les méthodes demandées :

Procédure RemplirPlateau( initVide : Boolean)

Variables : entiers i,j

Début :

plateau ← allocation d’un tableau de caractères de taille (nb\_cases)2  
//tableau à 2 dimensionpour i allant de 0 à nb\_cases-1 faire

pour j allant de 0 à nb\_cases-1 faire

plateau[i][j] ← casesVides

finPour

finPour

si (!initVide) alors

RemplirPlateauParteil()

finSi

Fin

//Jouer Glouton Brave et Téméraire  
Fonction JouerGlouton( c:Case, J:Joueur ) : Case  
Variables : Case meilleur  
Début

meilleur ← chercher\_meilleur\_case(J)  
supp\_case(c)  
retourner (meilleur)

Fin

Fonction JouerIATemeraire( c:Case, J:Joueur ) : Case  
Variables : Case meilleur  
Début

meilleur ← chercher\_meilleur\_case\_IA(J)  
supp\_case(c)  
retourner (meilleur)

Fin

//pour Brave  
Fonction EvalCase(c:Case, J:Joueur):entier  
variables : caractère id\_autre

entiers nb1, nb2, i, j  
Case c\_tmp

Début

si(J.id = J2.id) alors id\_autre←J1.id

sinon id\_autre←J2.id

finSi

nb1←0  
nb2←0  
si(estCaseLibre(c)) alors

nb1++  
nb2++  
pour i allant de -1 à 1 faire

pour j allant de -1 à 1 faire

c\_tmp← allocation d’une nouvelle Case (c.i+i, c.j+j)  
si (estCaseExiste(c\_tmp)) alors

si (plateau[c\_tmp.i][c\_tmp.j] = id\_autre ) alors nb1++  
finSi

finSi  
si ( plateau[c\_tmp.i][c\_tmp.j]!= caseVide ) alors nb2++  
finSi

finPour

finPour

finSi  
si ( nb2 > nb1 et nb1 = 9 ) alors retourner ( nb2 )  
finSi  
retourner ( nb1 )

Fin

//pour Téméraire  
Fonction EvalCase(c:Case, J:Joueur):entier  
variables : caractère id\_autre

entiers nb1, nb2, i, j  
Case c\_tmp  
Region r1, r2

Début

si(J.id = J2.id) alors id\_autre←J1.id

sinon id\_autre←J2.id

finSi  
nb1←0  
nb2←0  
r1← R.rechercheFeuille(c)  
si(estCaseLibre(c)) alors

nb1++  
nb2++  
pour i allant de -1 à 1 faire

pour j allant de -1 à 1 faire

c\_tmp← allocation d’une nouvelle Case (c.i+i, c.j+j)  
si (estCaseExiste(c\_tmp)) alors

si (plateau[c\_tmp.i][c\_tmp.j] = id\_autre ) alors

r2← R.rechercheFeuille(c\_tmp)  
si ( r1!=r2 et !r2.estAcquise(this) ou r1=r2 ) alors  
 nb1++  
finSi

finSi

finSi  
si ( plateau[c\_tmp.i][c\_tmp.j]!= caseVide ) alors nb2++  
finSi

finPour

finPour

finSi  
si ( nb2 > nb1 et nb1 = 9 ) alors retourner ( nb2 )  
finSi  
retourner ( nb1 )

Fin

Fonction RemplirRegion( p : Plataeu, J : Joueur ) : Region  
variables : entiers j1, j2  
Début

si ( estAcquise(p)) alors

si ( appartenant = p.J1.id ) alors ColorierRegion(p, p.J1)

sinon ColorierRegion(p, p.J2)

finSi

sinon

si ( estFeuille ) alors retourner ( this )

finSi

j1← 0  
j2← 0  
si ( haut\_gauche.estAcquise(p) ) alors

si ( haut\_gauche.appartenant = p.J1.id ) alors j1++

sinon si ( haut\_gauche.appartenant = p.J2.id ) alors j2++  
finSi

finSi  
si ( haut\_droite.estAcquise(p) ) alors

si ( haut\_droite.appartenant = p.J1.id ) alors j1++

sinon si ( haut\_droite.appartenant = p.J2.id ) alors j2++  
finSi

finSi  
si ( bas\_gauche.estAcquise(p) ) alors

si ( bas\_gauche.appartenant = p.J1.id ) alors j1++  
sinon si ( bas\_gauche.appartenant = p.J2.id ) alors j2++  
finSi

finSi  
si ( bas\_droite.estAcquise(p) ) alors

si ( bas\_droite.appartenant = p.J1.id ) alors j1++  
sinon si ( bas\_droite.appartenant = p.J2.id ) alors j2++  
finSi

finSi  
si ( estRemplie(p) ) alors

si ( j1 = 2 et j2 = 2 ) alors ColorierRegion (p, J)  
sinon si ( j1 ≥ 2 et j2 < j1 ) alors ColorierRegion ( p, p.J1 )  
sinon si ( j2 ≥ 2 et j1 < j2 ) alors ColorierRegion ( p, p.J2 )  
finSi

finSi  
haut\_gauche.RemplirRegion(p, J )  
haut\_droite.RemplirRegion(p, J )  
bas\_gauche.RemplirRegion(p, J )  
bas\_droite.RemplirRegion(p, J )

finSi  
retourner ( this )

Fin

Fonction CalculeScore( J:Joueur):entier  
Début

si ( J1.id = J.id ) alors retourner ( J1.Score)  
sinon si ( J2.id = J.id ) alors retourner ( J2.Score)  
sinon retourner 0  
finSi

Fin

Avant que l’utilisateur se met à jouer alors il verra un Menu avec plusieurs choix à faire.  
Comme par exemple :  
- Initialiser le jeu avec un fichier près rempli ou non.  
- si non alors il faudra fixer la taille du plateau.  
-Intialiser un plateau vide ou un plateau près rempli par hasard  
-L’utilisateur peut choisir de jouer contre l’ordinateur mais il peut également choisir de jouer contre un autre joueur humain s’il le souhaite.  
-Choisir la version du jeu, Brave ou Téméraire.  
-si Brave alors le jeu débute directement.-  
-si Téméraire et contre l’ordinateur alors il faudra choisir la stratégie du jeu donc Gloutonne ou IA.

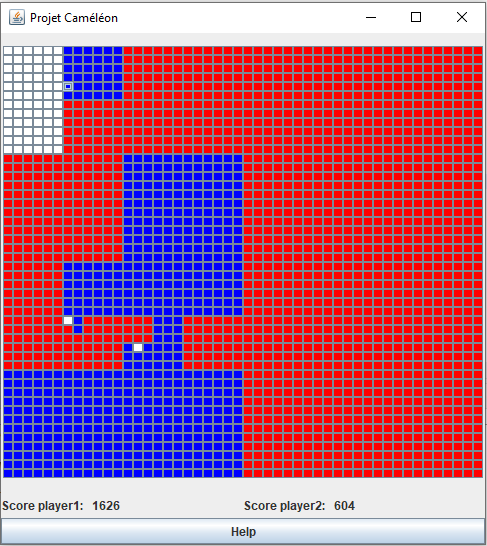
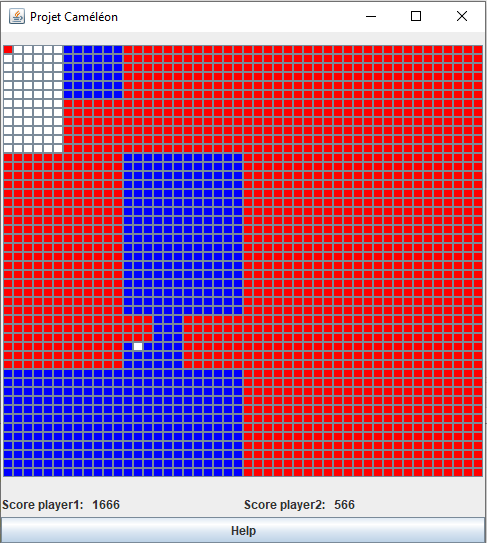
Difficulté rencontrées :

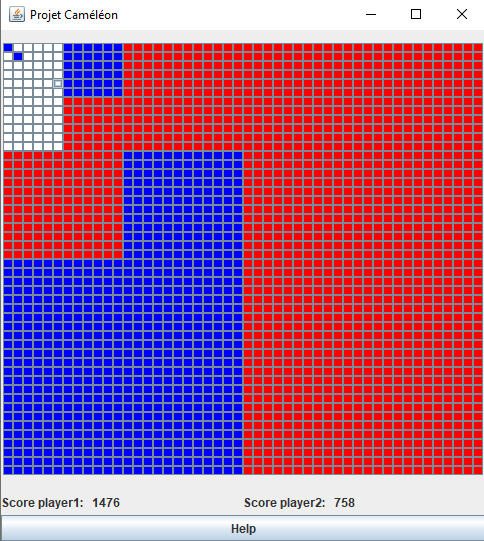
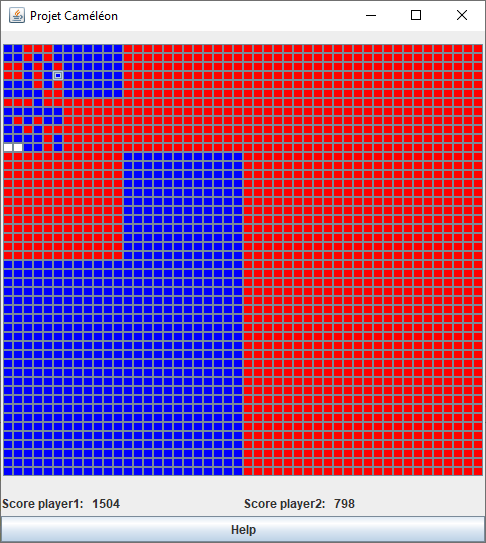
L’une des choses les plus compliquées pour nous c’était de comprendre le sujet et les consignes, et  
cela est dû à notre niveau de langue.   
Sinon concernant le travail même, alors l’un des points les plus compliqués était le coloriages d’une région lorsque celle-ci est acquise mais aussi de bien gérer les régions et surtout de bien choisir la bonne structure de donnée.  
  
Résoudre les difficultés et traverser les obstacles :

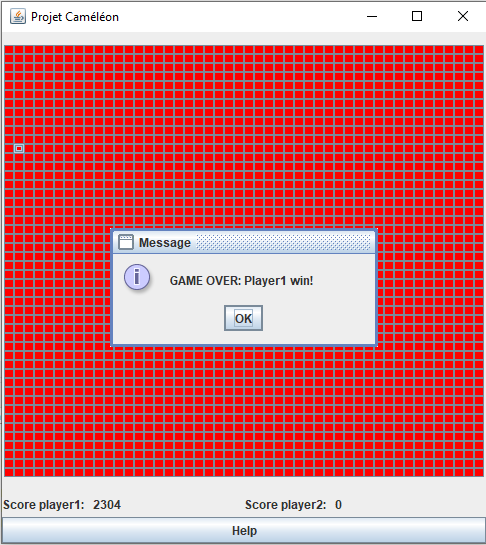
Ce n’était pas facile au début, nous avons changé le travail plusieurs fois, modifié le code plusieurs fois car nous n’étions pas du tout sûr de notre travail.  
Mais grâce au cours sur madoc, les réponses du prof et les explications pendant les travaux dirigés alors nous avons pu résoudre les problèmes que l’on avait, il y a aussi eu de la recherche sur internet de notre coté comme comment bien implémenter une structure de donnée ou comment gérer certaines données.  
  
Les point à améliorer :

Ce code est fonctionnel, le jeu est implémenté comme demandé avec les bonnes règles et les bonnes stratégies. Les deux versions du jeu sont bien fonctionnelles et jouables avec une interface graphique mais aussi une simulation sur la terminale du l’ordinateur.  
Il faut noter que ce code reste à améliorer pour avoir des meilleures complexités surtout pour la structure des données des deux versions, mais il y a aussi la fonctions pour chercher la meilleure case est améliorable enfin pour réduire la complexité et pourquoi pas chercher une meilleure

## Jeux de Tests :







## Conclusion :

En conclusion, ce projet était très important pour nous, non seulement vis à vis de la matière mais aussi pour appliquer nos connaissances acquises en classe et surtout les approfondir de plus en plus.  
Ce projet nous a été très utile et il nous a beaucoup aidé pour comprendre l’utilisation des arbres  
comme structure des données.