|  |  |
| --- | --- |
| Cours 420-EAB-LI Programmation orientée objet II Hiver 2015 | TP1 |

**Objectifs**: Après avoir complété ce travail, vous devriez être capable de:

1. Créer une application à partir d’un modèle de classes.
2. Manipuler une matrice (tableaux 2D) d’objets.
3. Gérer une interface graphique plus évoluée.

Mise en contexte

Nous voulons programmer un jeu composé d’une grille de 10 par 10 cases. Ce jeu se joue à deux joueurs. Chacun joue à tour de rôle, un avec des pions rouges et l’autre avec des pions bleus. Le premier qui réussit à placer 5 pions consécutifs en ligne gagne la partie. Les lignes de pions peuvent être **horizontales**, **verticales** ou **diagonales**.

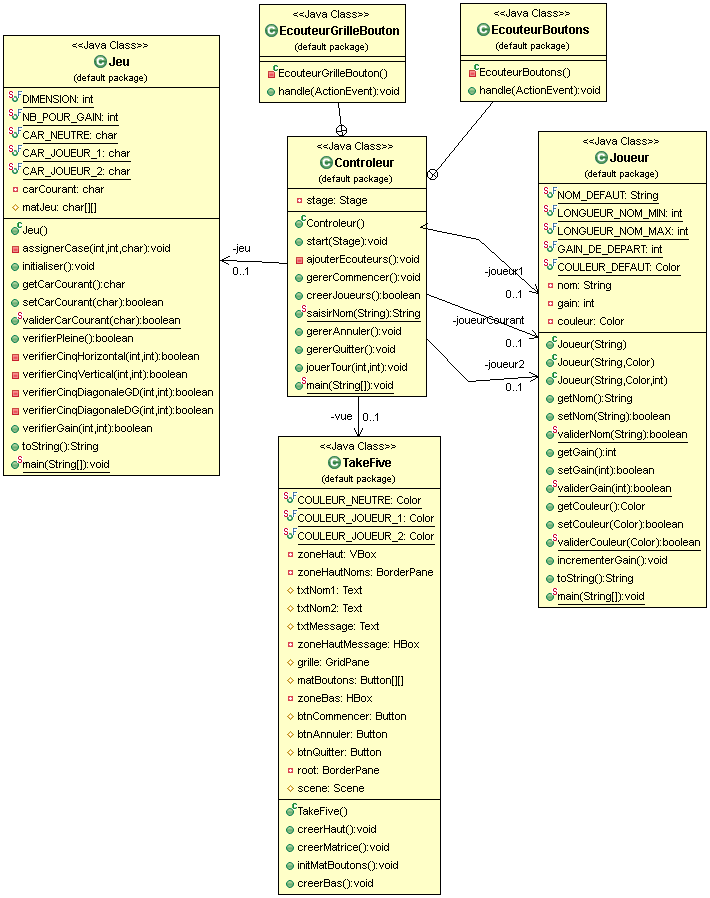
Vous devez donc coder ce jeu de façon à ce qu’il effectue les mêmes fonctionnalités que les exemples qui vous sont fournis sur LÉA.

Les joueurs peuvent jouer plusieurs parties et les gains de chacun seront affichés.

Pour réaliser ce jeu, vous devez utiliser 4 classes. Une classe **Joueur**, une classe **Jeu,** une classe **Gaincinq** (la vue) et une classe **Controleur**. **Une partie du code de ses classes est fournie. Vous devez obligatoirement partir de ces fichiers**. On vous fournit aussi un modèle de classes.

**Notez**: que dans la partie centrale de la fenêtre du jeu (dans la vue), nous avons une grille qui réfère à une matrice 10x10 de boutons cliquables qui possèdent une couleur (bleu, rouge ou blanc). Dans la classe Jeu, il y a aussi une matrice 10x10 de caractères (B, R ou espace).

|  |  |
| --- | --- |
| À l’ouverture | Pendant une partie |



Échéancier pour les remises

Semaine 1 : (30 points)

Au plus tard le : en début de laboratoire

* **Pour chaque classe demandée, vous référez à la JavaDoc et au diagramme de classes pour les infos sur la classe. Il est essentiel de respecter le diagramme et la JavaDoc.**
* À partir de la classe **Joueur** fournie, coder toutes les méthodes de cette classe.
* À partir de la classe **Jeu** fournie sur LÉA, coder toutes les méthodes de cette classe.
* Mettre des commentaires dans le code et la « JavaDoc » partout selon les normes.

**À remettre sur LÉA au début de la période de laboratoire** (8 avril):

Le document Word complet, un fichier « .zip », du nom de **Votre nom‑Lab1 sem1.zip** », du code source (.java) bien formaté et bien commenté (Java Doc) des classes **Joueur et Jeu**.

**package** TP1\_EAB;

**import** javafx.scene.paint.Color;

/\*\*

\* Classe représentant un joueur avec nom, gain et couleur.

\*/

**public** **class** Joueur

{

/\*\*

\* Nom par défaut du joueur

\*/

**public** **static** **final** String ***NOM\_DEFAUT*** = "Toto";

/\*\*

\* Longueur minimum d'un nom de joueur

\*/

**public** **static** **final** **int** ***LONGUEUR\_NOM\_MIN*** = 4;

/\*\*

\* Longueur maximum d'un nom de joueur

\*/

**public** **static** **final** **int** ***LONGUEUR\_NOM\_MAX*** = 15;

/\*\*

\* Gain de départ

\*/

**public** **static** **final** **int** ***GAIN\_DE\_DEPART*** = 0;

/\*\*

\* Couleur par défaut du joueur

\*/

**public** **static** **final** Color ***COULEUR\_DEFAUT*** = Color.***WHITE***;

/\*\*

\* Le nom du joueur

\*/

**private** String nom = "";

/\*\*

\* Les parties gagnées par le joueur

\*/

**private** **int** gain = 0;

/\*\*

\* La couleur du joueur

\*/

**private** Color couleur = **null**;

/\*\*

\* Constructeur avec paramètre nom. Met les gains du joueur à 0 et la couleur

\* par défaut.

\*

\* **@param** pNom le nom du joueur

\*/

**public** Joueur(String pNom)

{

**this**(pNom, Joueur.***COULEUR\_DEFAUT***, Joueur.***GAIN\_DE\_DEPART***);

}

/\*\*

\* Constructeur avec paramètres nom et couleur. Met les gains du joueur Ã 0.

\*

\* **@param** pNom le nom du joueur

\* **@param** pCouleur la couleur du joueur

\*/

**public** Joueur(String pNom, Color pCouleur)

{

**this**(pNom, pCouleur, Joueur.***GAIN\_DE\_DEPART***);

}

/\*\*

\* Constructeur avec paramètres nom, couleur et gain.

\*

\* **@param** pNom le nom du joueur

\* **@param** pCouleur la couleur du joueur

\* **@param** pGain les gain du joueur au dàpart

\*/

**public** Joueur(String pNom, Color pCouleur, **int** pGain)

{

**boolean** ok = setNom(pNom) && setCouleur(pCouleur) && setGain(pGain);

**if** (!ok)

{

setNom(Joueur.***NOM\_DEFAUT***);

setCouleur(Joueur.***COULEUR\_DEFAUT***);

setGain(Joueur.***GAIN\_DE\_DEPART***);

}

}

/\*\*

\* Obtenir le nom du joueur

\*

\* **@return** String, le nom du joueur

\*/

**public** String getNom()

{

**return** nom;

}

/\*\*

\* Modifier le nom du joueur.

\*

\* **@param** pNom le nouveau nom du joueur

\*

\* **@return** boolean, vrai si le nom a été modifié

\*/

**public** **boolean** setNom(String pNom)

{

**boolean** ok = *validerNom*(pNom);

**if** (ok)

{

**this**.nom = pNom;

}

**return** ok;

}

/\*\*

\* Valider le nom du joueur (n'est pas null et possède une longueur valide)

\*

\* **@param** pNom le nom à valider

\*

\* **@return** boolean, vrai si le nom est valide

\*/

**public** **static** **boolean** validerNom(String pNom)

{

**return** ((pNom != **null**) && (pNom.length() >= Joueur.***LONGUEUR\_NOM\_MIN***) &&

(pNom.length() <= Joueur.***LONGUEUR\_NOM\_MAX***));

}

/\*\*

\* Obtenir la valeur des gains du joueur.

\*

\* **@return** int, le valeur des gains.

\*/

**public** **int** getGain()

{

**return** gain;

}

/\*\*

\* Modifier la valeur des gains du joueur

\*

\* **@param** pGain la nouvelle valeur des gains du joueur

\*

\* **@return** boolean, vrai si la valeur a été modifiée

\*/

**public** **boolean** setGain(**int** pGain)

{

**boolean** ok = *validerGain*(pGain);

**if** (ok)

{

**this**.gain = pGain;

}

**return** ok;

}

/\*\*

\* Valider la valeur des gains du joueur. Doit être positive, 0 inclus.

\*

\* **@param** pGain la valeur à valider.

\*

\* **@return** boolean, vrai si la valeur est valide.

\*/

**public** **static** **boolean** validerGain(**int** pGain)

{

**return** (pGain >= 0);

}

/\*\*

\* Obtenir la couleur du joueur

\*

\* **@return** Color, la couleur du joueur

\*/

**public** Color getCouleur()

{

**return** couleur;

}

/\*\*

\* Modifier la valeur de la couleur du joueur

\*

\* **@param** pCouleur la nouvelle couleur

\*

\* **@return** boolean, vrai si la couleur a été modifiée

\*/

**public** **boolean** setCouleur(Color pCouleur)

{

**boolean** ok = Joueur.*validerCouleur*(pCouleur);

**if** (ok)

{

couleur = pCouleur;

}

**return** ok;

}

/\*\*

\* Valider la couleur du joueur

\*

\* **@param** pCouleur la couleur à valider

\*

\* **@return** boolean, vrai si le pointeur de l'objet n'est pas nul

\*/

**public** **static** **boolean** validerCouleur(Color pCouleur)

{

**return** (pCouleur != **null**);

}

/\*\*

\* Incrémente les gains de 1

\*/

**public** **void** incrementerGain()

{

**this**.setGain(**this**.getGain() + 1);

}

/\*\*

\* Construit une chaine de caractères représentant l'état de l'objet Joueur

\*

\* **@return** String, la chaine construite contenant nom et gains

\*/

**public** String toString()

{

**return** **this**.getNom() + ": " + **this**.getGain() + " Gain(s)";

}

/\*\*

\* Point d'entrée de la classe Joueur.

\*

\* **@param** args un tableau de "string", les arguments

\*/

**public** **static** **void** main(String[] args)

{

// **TODO** Auto-generated method stub

}

}

**package** TP1\_EAB;

/\*\*

\* Classe représentant un jeu avec matrice 10x10 et caractère du joueur

\* courant (R ou B).

\*

\*

\*/

**public** **class** Jeu

{

/\*\*

\* Dimension de la grille de jeu DIMENSION X DIMENSION

\*/

**public** **static** **final** **int** ***DIMENSION*** = 10;

/\*\*

\* Nombre de cases identiques consécutives pour faire un gain

\*/

**public** **static** **final** **int** ***NB\_POUR\_GAIN*** = 5;

/\*\*

\* Caractère d'une case qui n'est pas choisie

\*/

**public** **static** **final** **char** ***CAR\_NEUTRE*** = ' ';

/\*\*

\* Caractère associé au joueur 1 (R pour rouge)

\*/

**public** **static** **final** **char** ***CAR\_JOUEUR\_1*** = 'R';

/\*\*

\* Caractère associé au joueur 2 (B pour bleu)

\*/

**public** **static** **final** **char** ***CAR\_JOUEUR\_2*** = 'B';

/\*\*

\* Le caractère que doit prendre la case de la matrice lors d'un clique.

\* Change selon le joueur courant du jeu!

\*/

**private** **char** carCourant;

/\*\*

\* Un tableau 2D de caractères pour former la grille en mémoire

\*/

**protected** **char**[][] matJeu = **null**;

/\*\*

\* Création d'un jeu avec sa matrice de caractères initialisée avec des

\* espaces partout et le caractère courant initialisé à espace.

\*

\*/

**public** Jeu()

{

matJeu = **new** **char**[***DIMENSION***][***DIMENSION***];

setCarCourant(***CAR\_NEUTRE***);

initialiser();

}

/\*\*

\* Méthode accessoire...

\*

\* Permet de mettre un caractère désiré dans une des cases de la matrice.

\* Utilisé pour faire des tests unitaires.

\*

\* **@param** pLig la ligne du caractère

\* **@param** pCol la colonne du caractère

\* **@param** pCarac le caractère désiré

\*/

**private** **void** assignerCase(**int** pLig, **int** pCol, **char** pCarac)

{

matJeu[pLig][pCol] = pCarac;

}

/\*\*

\* Initialiser la matrice de caractères en mettant le caractère neutre

\* (espace) partout.

\*/

**public** **void** initialiser()

{

// Lignes

**for** (**int** i = 0; i < matJeu.length; i++)

{

// Colonnes

**for** (**int** j = 0; j < matJeu[i].length; j++)

{

matJeu[i][j] = ***CAR\_NEUTRE***;

}

}

}

/\*\*

\* Obtenir le caractère courant

\*

\* **@return** char, le caractère courant

\*/

**public** **char** getCarCourant()

{

**return** carCourant;

}

/\*\*

\* Modifier la valeur du caractère courant

\*

\* **@param** pCarCourant le nouveau caractère

\*

\* **@return** boolean, vrai si la valeur a été mise à jour

\*/

**public** **boolean** setCarCourant(**char** pCarCourant)

{

**boolean** ok = *validerCarCourant*(pCarCourant);

**if** (ok)

{

carCourant = pCarCourant;

}

**return** (ok);

}

/\*\*

\* Valider le caractère courant. Doit être neutre ou celui du joueur1 ou

\* celui du joueur2

\*

\* **@param** pCarCourant le caractère à valider

\*

\* **@return** boolean, vrai si valide.

\*/

**public** **static** **boolean** validerCarCourant(**char** pCarCourant)

{

**return** (pCarCourant == ***CAR\_NEUTRE*** || pCarCourant == ***CAR\_JOUEUR\_1*** || pCarCourant == ***CAR\_JOUEUR\_2***);

}

/\*\*

\* Vérifier s'il reste des cases à sélectionner dans la matrice. S'il reste

\* des cases de couleur neutre.

\*

\* **@return** boolean, vrai s'il n'y a plus de cases disponibles

\*/

**public** **boolean** verifierPleine()

{

**boolean** pleine = **true**;

**for** (**int** i = 0; i < matJeu.length && pleine; i++)

{

**for** (**int** j = 0; j < matJeu[i].length && pleine; j++)

{

// On cherche une case avec un espace

**if** (matJeu[i][j] == ***CAR\_NEUTRE***)

{

pleine = **false**;

}

}

}

**return** pleine;

}

/\*\*

\* Vérifier s'il y a NB\_POUR\_GAIN cases ayant le même caractère que le

\* caractère courant sur une ligne horizontale à partir des indices i et j.

\* Est appelée par verifierGain()

\*

\* **@param** i la ligne du bouton cliqué

\* **@param** j la colonne du bouton cliqué

\*

\* **@return** boolean vrai si NB\_POUR\_GAIN cases de suite sur la même ligne ont

\* le même caractère que le caractère courant.

\*/

**private** **boolean** verifierCinqHorizontal(**int** i, **int** j)

{

**int** compteur = 1;

**char** carCourant = matJeu[i][j];

// Vers la droite

**int** col = j + 1;

**while** ((col < matJeu[i].length) && (matJeu[i][col] == carCourant))

{

compteur++;

col++;

}

// Vers la gauche

col = j - 1;

**while** ((col >= 0) && (matJeu[i][col] == carCourant))

{

compteur++;

col--;

}

**return** (compteur >= ***NB\_POUR\_GAIN***);

}

/\*\*

\* Vérifier s'il y a NB\_POUR\_GAIN cases ayant le même caractère que le

\* caractère courant sur une ligne verticale (colonne) à partir des indices

\* i et j. Est appelé par verifierGain()

\*

\* **@param** i la ligne du bouton cliqué

\* **@param** j la colonne du bouton cliqué

\*

\* **@return** boolean vrai si NB\_POUR\_GAIN cases de suite sur la même colonne

\* ont le même caractère que le caractère courant.

\*/

**private** **boolean** verifierCinqVertical(**int** i, **int** j)

{

**int** compteur = 1;

**char** carCourant = matJeu[i][j];

// Vers le bas

**int** lig = i + 1;

**while** ((lig < matJeu.length) && (matJeu[lig][j] == carCourant))

{

compteur++;

lig++;

}

// Vers le haut

lig = i - 1;

**while** ((lig >= 0) && (matJeu[lig][j] == carCourant))

{

compteur++;

lig--;

}

**return** (compteur >= ***NB\_POUR\_GAIN***);

}

/\*\*

\* Vérifier s'il y a NB\_POUR\_GAIN cases ayant le même caractère que le

\* caractère courant sur une diagonale gauche/droite à partir des indices i

\* et j. Est appelé par verifierGain()

\*

\* **@param** i la ligne du bouton cliqué

\* **@param** j la colonne du bouton cliqué

\*

\* **@return** boolean vrai si NB\_POUR\_GAIN cases de suite sur la même diagonale

\* gauche/droite ont le même caractère que le caractère courant.

\*/

**private** **boolean** verifierCinqDiagonaleGD(**int** i, **int** j)

{

**int** compteur = 1;

**char** carCourant = matJeu[i][j];

// Vers bas et droite

**int** lig = i + 1;

**int** col = j + 1;

**while** ((lig < matJeu.length) && (col < matJeu[lig].length)

&& (matJeu[lig][col] == carCourant))

{

compteur++;

lig++;

col++;

}

// Vers haut et gauche

lig = i - 1;

col = j - 1;

**while** (lig >= 0 && col >= 0 && matJeu[lig][col] == carCourant)

{

compteur++;

lig--;

col--;

}

**return** (compteur >= ***NB\_POUR\_GAIN***);

}

/\*\*

\* Vérifier s'il y a NB\_POUR\_GAIN cases ayant le même caractère que le

\* caractère courant sur une diagonale droite/gauche à partir des indices i

\* et j. Est appelée par verifierGain()

\*

\* **@param** i la ligne du bouton cliquà

\* **@param** j la colonne du bouton cliquà

\*

\* **@return** boolean vrai si NB\_POUR\_GAIN cases de suite sur la même

\* diagonale droite/gauche ont le même caractère que le caractère

\* courant.

\*/

**private** **boolean** verifierCinqDiagonaleDG(**int** i, **int** j)

{

**int** compteur = 1;

**char** carCourant = matJeu[i][j];

// Vers haut et droite

**int** lig = i - 1;

**int** col = j + 1;

**while** ((lig >= 0) && (col < matJeu[lig].length)

&& (matJeu[lig][col] == carCourant))

{

compteur++;

lig--;

col++;

}

// Vers bas et gauche

lig = i + 1;

col = j - 1;

**while** ((lig < matJeu.length) && (col >= 0)

&& (matJeu[lig][col] == carCourant))

{

compteur++;

lig++;

col--;

}

**return** (compteur >= ***NB\_POUR\_GAIN***);

}

/\*\*

\* Méthode qui fait l'ensemble des validations pour voir s'il y a un gain.

\*

\* Vérifier s'il y a NB\_POUR\_GAIN cases ayant le même caractère que le

\* caractère courant dans les différentes directions : horizontale,

\* verticale et diagonales. Appelle les méthodes nécessaires pour faire le

\* travail.

\*

\* **@param** i la ligne du bouton cliqué

\* **@param** j la colonne du bouton cliqué

\*

\* **@return** boolean, vrai s'il a un gain

\*/

**public** **boolean** verifierGain(**int** i, **int** j)

{

**return** (verifierCinqHorizontal(i, j) || verifierCinqVertical(i, j)

|| verifierCinqDiagonaleGD(i, j) || verifierCinqDiagonaleDG(i,

j));

}

/\*\*

\* Retourne la matrice sous forme de dessin

\*

\* Chaque case est représentée par [ ] avec Ã l'intérieur soit un espace,

\* soit un R soit un B Exemple pour une 3x3: [ ][R][B] [B][ ][R] [B][R][ ]

\*

\* **@return** String, dessin de la matrice

\*/

**public** String toString()

{

String chaine = "";

**for** (**int** i = 0; i < matJeu.length; i++)

{

**for** (**int** j = 0; j < matJeu.length; j++)

{

chaine += (matJeu[i][j] == ***CAR\_NEUTRE***) ? "[ ]" : "["

+ matJeu[i][j] + "]";

}

chaine += "\n";

}

**return** chaine;

}

**public** **static** **void** main(String[] args)

{

Jeu jeu1 = **new** Jeu();

System.***out***.println(jeu1);

jeu1.assignerCase(1, 1, 'B');

jeu1.assignerCase(1, 2, 'B');

jeu1.assignerCase(1, 3, 'B');

jeu1.assignerCase(1, 4, 'B');

jeu1.assignerCase(1, 5, 'B');

System.***out***.println(jeu1);

System.***out***.println(jeu1.verifierGain(1, 1));

}

}

**package** TP1\_EAB;

**import** javafx.scene.paint.Color;

/\*\*

\* Classe représentant un joueur avec nom, gain et couleur.

\*/

public class Joueur

{

/\*\*

\* Nom par défaut du joueur

\*/

public static final String NOM\_DEFAUT = "Toto";

/\*\*

\* Longueur minimum d'un nom de joueur

\*/

public static final int LONGUEUR\_NOM\_MIN = 4;

/\*\*

\* Longueur maximum d'un nom de joueur

\*/

public static final int LONGUEUR\_NOM\_MAX = 15;

/\*\*

\* Gain de départ

\*/

public static final int GAIN\_DE\_DEPART = 0;

/\*\*

\* Couleur par défaut du joueur

\*/

public static final Color COULEUR\_DEFAUT = Color.WHITE;

/\*\*

\* Le nom du joueur

\*/

private String nom = "";

/\*\*

\* Les parties gagnées par le joueur

\*/

private int gain = 0;

/\*\*

\* La couleur du joueur

\*/

private Color couleur = null;

/\*\*

\* Constructeur avec paramètre nom. Met les gains du joueur à 0 et la couleur

\* par défaut.

\*

\* @param pNom le nom du joueur

\*/

public Joueur(String pNom)

{

this(pNom, Joueur.COULEUR\_DEFAUT, Joueur.GAIN\_DE\_DEPART);

}

/\*\*

\* Constructeur avec paramètres nom et couleur. Met les gains du joueur Ã 0.

\*

\* @param pNom le nom du joueur

\* @param pCouleur la couleur du joueur

\*/

public Joueur(String pNom, Color pCouleur)

{

this(pNom, pCouleur, Joueur.GAIN\_DE\_DEPART);

}

/\*\*

\* Constructeur avec paramètres nom, couleur et gain.

\*

\* @param pNom le nom du joueur

\* @param pCouleur la couleur du joueur

\* @param pGain les gain du joueur au dàpart

\*/

public Joueur(String pNom, Color pCouleur, int pGain)

{

boolean ok = setNom(pNom) && setCouleur(pCouleur) && setGain(pGain);

if (!ok)

{

setNom(Joueur.NOM\_DEFAUT);

setCouleur(Joueur.COULEUR\_DEFAUT);

setGain(Joueur.GAIN\_DE\_DEPART);

}

}

/\*\*

\* Obtenir le nom du joueur

\*

\* @return String, le nom du joueur

\*/

public String getNom()

{

return nom;

}

/\*\*

\* Modifier le nom du joueur.

\*

\* @param pNom le nouveau nom du joueur

\*

\* @return boolean, vrai si le nom a été modifié

\*/

public boolean setNom(String pNom)

{

boolean ok = validerNom(pNom);

if (ok)

{

this.nom = pNom;

}

return ok;

}

/\*\*

\* Valider le nom du joueur (n'est pas null et possède une longueur valide)

\*

\* @param pNom le nom à valider

\*

\* @return boolean, vrai si le nom est valide

\*/

public static boolean validerNom(String pNom)

{

return ((pNom != null) && (pNom.length() >= Joueur.LONGUEUR\_NOM\_MIN) &&

(pNom.length() <= Joueur.LONGUEUR\_NOM\_MAX));

}

/\*\*

\* Obtenir la valeur des gains du joueur.

\*

\* @return int, le valeur des gains.

\*/

public int getGain()

{

return gain;

}

/\*\*

\* Modifier la valeur des gains du joueur

\*

\* @param pGain la nouvelle valeur des gains du joueur

\*

\* @return boolean, vrai si la valeur a été modifiée

\*/

public boolean setGain(int pGain)

{

boolean ok = validerGain(pGain);

if (ok)

{

this.gain = pGain;

}

return ok;

}

/\*\*

\* Valider la valeur des gains du joueur. Doit être positive, 0 inclus.

\*

\* @param pGain la valeur à valider.

\*

\* @return boolean, vrai si la valeur est valide.

\*/

public static boolean validerGain(int pGain)

{

return (pGain >= 0);

}

/\*\*

\* Obtenir la couleur du joueur

\*

\* @return Color, la couleur du joueur

\*/

public Color getCouleur()

{

return couleur;

}

/\*\*

\* Modifier la valeur de la couleur du joueur

\*

\* @param pCouleur la nouvelle couleur

\*

\* @return boolean, vrai si la couleur a été modifiée

\*/

public boolean setCouleur(Color pCouleur)

{

boolean ok = Joueur.validerCouleur(pCouleur);

if (ok)

{

couleur = pCouleur;

}

return ok;

}

/\*\*

\* Valider la couleur du joueur

\*

\* @param pCouleur la couleur à valider

\*

\* @return boolean, vrai si le pointeur de l'objet n'est pas nul

\*/

public static boolean validerCouleur(Color pCouleur)

{

return (pCouleur != null);

}

/\*\*

\* Incrémente les gains de 1

\*/

public void incrementerGain()

{

this.setGain(this.getGain() + 1);

}

/\*\*

\* Construit une chaine de caractères représentant l'état de l'objet Joueur

\*

\* @return String, la chaine construite contenant nom et gains

\*/

public String toString()

{

return this.getNom() + ": " + this.getGain() + " Gain(s)";

}

/\*\*

\* Point d'entrée de la classe Joueur.

\*

\* @param args un tableau de "string", les arguments

\*/

public static void main(String[] args)

{

// TODO Auto-generated method stub

}

}

package TP1\_EAB;

/\*\*

\* Classe représentant un jeu avec matrice 10x10 et caractère du joueur

\* courant (R ou B).

\*

\*

\*/

public class Jeu

{

/\*\*

\* Dimension de la grille de jeu DIMENSION X DIMENSION

\*/

public static final int DIMENSION = 10;

/\*\*

\* Nombre de cases identiques consécutives pour faire un gain

\*/

public static final int NB\_POUR\_GAIN = 5;

/\*\*

\* Caractère d'une case qui n'est pas choisie

\*/

public static final char CAR\_NEUTRE = ' ';

/\*\*

\* Caractère associé au joueur 1 (R pour rouge)

\*/

public static final char CAR\_JOUEUR\_1 = 'R';

/\*\*

\* Caractère associé au joueur 2 (B pour bleu)

\*/

public static final char CAR\_JOUEUR\_2 = 'B';

/\*\*

\* Le caractère que doit prendre la case de la matrice lors d'un clique.

\* Change selon le joueur courant du jeu!

\*/

private char carCourant;

/\*\*

\* Un tableau 2D de caractères pour former la grille en mémoire

\*/

protected char[][] matJeu = null;

/\*\*

\* Création d'un jeu avec sa matrice de caractères initialisée avec des

\* espaces partout et le caractère courant initialisé à espace.

\*

\*/

public Jeu()

{

matJeu = new char[DIMENSION][DIMENSION];

setCarCourant(CAR\_NEUTRE);

initialiser();

}

/\*\*

\* Méthode accessoire...

\*

\* Permet de mettre un caractère désiré dans une des cases de la matrice.

\* Utilisé pour faire des tests unitaires.

\*

\* @param pLig la ligne du caractère

\* @param pCol la colonne du caractère

\* @param pCarac le caractère désiré

\*/

private void assignerCase(int pLig, int pCol, char pCarac)

{

matJeu[pLig][pCol] = pCarac;

}

/\*\*

\* Initialiser la matrice de caractères en mettant le caractère neutre

\* (espace) partout.

\*/

public void initialiser()

{

// Lignes

for (int i = 0; i < matJeu.length; i++)

{

// Colonnes

for (int j = 0; j < matJeu[i].length; j++)

{

matJeu[i][j] = CAR\_NEUTRE;

}

}

}

/\*\*

\* Obtenir le caractère courant

\*

\* @return char, le caractère courant

\*/

public char getCarCourant()

{

return carCourant;

}

/\*\*

\* Modifier la valeur du caractère courant

\*

\* @param pCarCourant le nouveau caractère

\*

\* @return boolean, vrai si la valeur a été mise à jour

\*/

public boolean setCarCourant(char pCarCourant)

{

boolean ok = validerCarCourant(pCarCourant);

if (ok)

{

carCourant = pCarCourant;

}

return (ok);

}

/\*\*

\* Valider le caractère courant. Doit être neutre ou celui du joueur1 ou

\* celui du joueur2

\*

\* @param pCarCourant le caractère à valider

\*

\* @return boolean, vrai si valide.

\*/

public static boolean validerCarCourant(char pCarCourant)

{

return (pCarCourant == CAR\_NEUTRE || pCarCourant == CAR\_JOUEUR\_1 || pCarCourant == CAR\_JOUEUR\_2);

}

/\*\*

\* Vérifier s'il reste des cases à sélectionner dans la matrice. S'il reste

\* des cases de couleur neutre.

\*

\* @return boolean, vrai s'il n'y a plus de cases disponibles

\*/

public boolean verifierPleine()

{

boolean pleine = true;

for (int i = 0; i < matJeu.length && pleine; i++)

{

for (int j = 0; j < matJeu[i].length && pleine; j++)

{

// On cherche une case avec un espace

if (matJeu[i][j] == CAR\_NEUTRE)

{

pleine = false;

}

}

}

return pleine;

}

/\*\*

\* Vérifier s'il y a NB\_POUR\_GAIN cases ayant le même caractère que le

\* caractère courant sur une ligne horizontale Ã partir des indices i et j.

\* Est appelée par verifierGain()

\*

\* @param i la ligne du bouton cliqué

\* @param j la colonne du bouton cliqué

\*

\* @return boolean vrai si NB\_POUR\_GAIN cases de suite sur la même ligne ont

\* le même caractère que le caractère courant.

\*/

private boolean verifierCinqHorizontal(int i, int j)

{

int compteur = 1;

char carCourant = matJeu[i][j];

// Vers la droite

int col = j + 1;

while ((col < matJeu[i].length) && (matJeu[i][col] == carCourant))

{

compteur++;

col++;

}

// Vers la gauche

col = j - 1;

while ((col >= 0) && (matJeu[i][col] == carCourant))

{

compteur++;

col--;

}

return (compteur >= NB\_POUR\_GAIN);

}

/\*\*

\* Vérifier s'il y a NB\_POUR\_GAIN cases ayant le même caractère que le

\* caractère courant sur une ligne verticale (colonne) Ã partir des indices

\* i et j. Est appelé par verifierGain()

\*

\* @param i la ligne du bouton cliqué

\* @param j la colonne du bouton cliqué

\*

\* @return boolean vrai si NB\_POUR\_GAIN cases de suite sur la même colonne

\* ont le même caractère que le caractère courant.

\*/

private boolean verifierCinqVertical(int i, int j)

{

int compteur = 1;

char carCourant = matJeu[i][j];

// Vers le bas

int lig = i + 1;

while ((lig < matJeu.length) && (matJeu[lig][j] == carCourant))

{

compteur++;

lig++;

}

// Vers le haut

lig = i - 1;

while ((lig >= 0) && (matJeu[lig][j] == carCourant))

{

compteur++;

lig--;

}

return (compteur >= NB\_POUR\_GAIN);

}

/\*\*

\* Vérifier s'il y a NB\_POUR\_GAIN cases ayant le même caractère que le

\* caractère courant sur une diagonale gauche/droite Ã partir des indices i

\* et j. Est appelé par verifierGain()

\*

\* @param i la ligne du bouton cliqué

\* @param j la colonne du bouton cliqué

\*

\* @return boolean vrai si NB\_POUR\_GAIN cases de suite sur la même diagonale

\* gauche/droite ont le même caractère que le caractère courant.

\*/

private boolean verifierCinqDiagonaleGD(int i, int j)

{

int compteur = 1;

char carCourant = matJeu[i][j];

// Vers bas et droite

int lig = i + 1;

int col = j + 1;

while ((lig < matJeu.length) && (col < matJeu[lig].length)

&& (matJeu[lig][col] == carCourant))

{

compteur++;

lig++;

col++;

}

// Vers haut et gauche

lig = i - 1;

col = j - 1;

while (lig >= 0 && col >= 0 && matJeu[lig][col] == carCourant)

{

compteur++;

lig--;

col--;

}

return (compteur >= NB\_POUR\_GAIN);

}

/\*\*

\* Vérifier s'il y a NB\_POUR\_GAIN cases ayant le même caractère que le

\* caractère courant sur une diagonale droite/gauche Ã partir des indices i

\* et j. Est appelée par verifierGain()

\*

\* @param i la ligne du bouton cliquà

\* @param j la colonne du bouton cliquà

\*

\* @return boolean vrai si NB\_POUR\_GAIN cases de suite sur la même

\* diagonale droite/gauche ont le même caractère que le caractère

\* courant.

\*/

private boolean verifierCinqDiagonaleDG(int i, int j)

{

int compteur = 1;

char carCourant = matJeu[i][j];

// Vers haut et droite

int lig = i - 1;

int col = j + 1;

while ((lig >= 0) && (col < matJeu[lig].length)

&& (matJeu[lig][col] == carCourant))

{

compteur++;

lig--;

col++;

}

// Vers bas et gauche

lig = i + 1;

col = j - 1;

while ((lig < matJeu.length) && (col >= 0)

&& (matJeu[lig][col] == carCourant))

{

compteur++;

lig++;

col--;

}

return (compteur >= NB\_POUR\_GAIN);

}

/\*\*

\* Méthode qui fait l'ensemble des validations pour voir s'il y a un gain.

\*

\* Vérifier s'il y a NB\_POUR\_GAIN cases ayant le même caractère que le

\* caractère courant dans les différentes directions : horizontale,

\* verticale et diagonales. Appelle les méthodes nécessaires pour faire le

\* travail.

\*

\* @param i la ligne du bouton cliquà

\* @param j la colonne du bouton cliquà

\*

\* @return boolean, vrai s'il a un gain

\*/

public boolean verifierGain(int i, int j)

{

return (verifierCinqHorizontal(i, j) || verifierCinqVertical(i, j)

|| verifierCinqDiagonaleGD(i, j) || verifierCinqDiagonaleDG(i,

j));

}

/\*\*

\* Retourne la matrice sous forme de dessin

\*

\* Chaque case est représentée par [ ] avec Ã l'intérieur soit un espace,

\* soit un R soit un B Exemple pour une 3x3: [ ][R][B] [B][ ][R] [B][R][ ]

\*

\* @return String, dessin de la matrice

\*/

public String toString()

{

String chaine = "";

for (int i = 0; i < matJeu.length; i++)

{

for (int j = 0; j < matJeu.length; j++)

{

chaine += (matJeu[i][j] == CAR\_NEUTRE) ? "[ ]" : "["

+ matJeu[i][j] + "]";

}

chaine += "\n";

}

return chaine;

}

public static void main(String[] args)

{

Jeu jeu1 = new Jeu();

System.out.println(jeu1);

jeu1.assignerCase(1, 1, 'B');

jeu1.assignerCase(1, 2, 'B');

jeu1.assignerCase(1, 3, 'B');

jeu1.assignerCase(1, 4, 'B');

jeu1.assignerCase(1, 5, 'B');

System.out.println(jeu1);

System.out.println(jeu1.verifierGain(1, 1));

}

}