Cours d'introduction à la programmation (en Java)

Programmer un Puissance 4

Jamila Sam
Jean-Cédric Chappelier
Vincent Lepetit

Faculté I&C

Le jeu

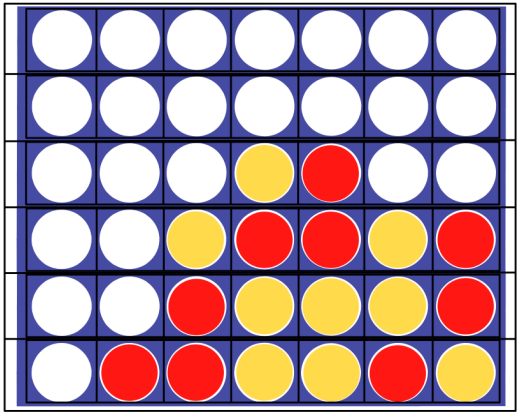
```
 | | | | | | | |
 | | | | | | | |
 | | | |X| | | |
 | | |X|O|O|X|O|
 | | |O|X|X|X|O|
 | |O|O|X|X|O|X|
==1=2=3=4=5=6=7==

Joueur O :  entrez un numéro de colonne
5

 | | | | | | | |
 | | | | | | | |
 | | | |X|O| | |
 | | |X|O|O|X|O|
 | | |O|X|X|X|O|
 | |O|O|X|X|O|X|
==1=2=3=4=5=6=7==

Le joueur O a gagne !
```

Développement

1. N'essayez pas d'écrire tout le programme en une seule fois !
   - Décomposez le problème en sous-problèmes pour développer le programme par étapes ;
   - À chaque étape, testez le code développé.
2. Tout d'abord, identifiez les types nécessaires ;
3. Identifiez les fonctions qui portent sur ces types, écrivez-les et testez-les au fur et à mesure ;
4. Quand une fonction est difficile à écrire, créez une fonction pour chaque point difficile.

Identifier les types

type_des_elements[][]



Type des éléments

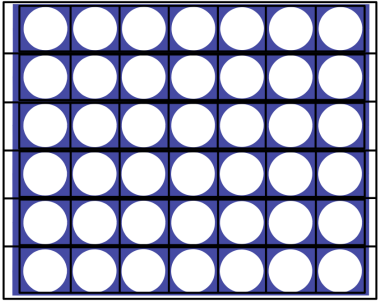
type_des_elements ?

Nous allons utiliser `int` et définir les constantes :

```
private final static int VIDE  = 0;
private final static int JAUNE = 1;
private final static int ROUGE = 2;
```

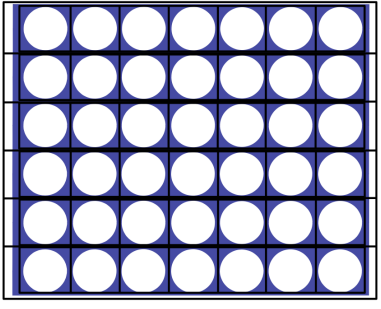
Types de notre programme

```
private final static int VIDE  = 0;
private final static int JAUNE = 1;
private final static int ROUGE = 2;

...

int[][] grille;

grille[0][0] = VIDE;
grille[2][3] = JAUNE;
```



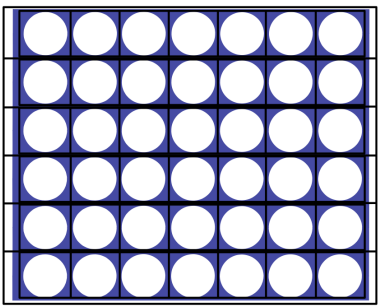
Premières méthodes

- `initialise` : méthode qui initialise une grille ;
- `affiche` : méthode qui affiche une grille.



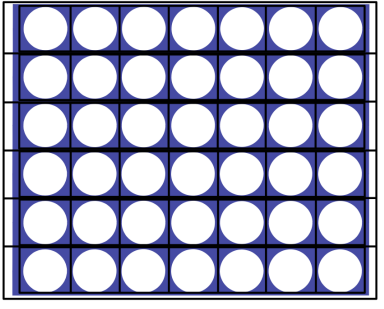
Méthode initialise

```
private final static int VIDE  = 0;
private final static int JAUNE = 1;
private final static int ROUGE = 2;

...

static void initialise(int[][] grille)
{
  for(int i = 0; i < grille.length; ++i) {
    for(int j = 0; j < grille[i].length; ++j) {
      grille[][j] = VIDE;
    }
  }
}

...

int[][] grille = new int[6][7];
initialise(grille);
```



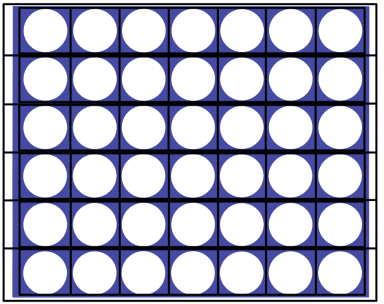
Méthode initialise

```
private final static int VIDE  = 0;
private final static int JAUNE = 1;
private final static int ROUGE = 2;

...

static void initialise(int[][] grille)
{
  for(int i = 0; i < grille.length; ++i) {
    for(int j = 0; j < grille[i].length; ++j) {
      grille[i][j] = VIDE;
    }
  }
}

...

int[][] grille = new int[6][7];
initialise(grille);
```



Méthode affiche

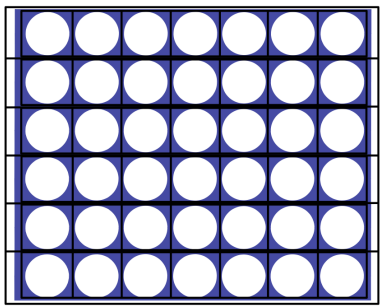
```
private final static int VIDE  = 0;
private final static int JAUNE = 1;
private final static int ROUGE = 2;

...
// affiche O pour une case rouge, X pour une case jaune
static void affiche(int[][] grille)
{
  for(int[] ligne : grille) {
    for(int cellule : ligne) {
      if (cellule == VIDE) {
        System.out.print(' ');
      } else if (cellule == ROUGE) {
        System.out.print('O');
      } else {
        System.out.print('X');
      }
    }
    System.out.println();
  }
}

...

int[][] grille = new int[6][7];
initialise(grille);
```



Tester `initialise` et `affiche`

```
public static void main(String[] args)
{
  int[][] grille = new int[6][7];

  initialise(grille);
  grille[2][3] = JAUNE;
  grille[2][4] = ROUGE;
  affiche(grille);
}
```

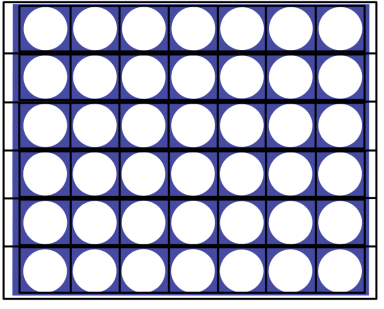
```
␣␣␣␣␣␣␣
␣␣␣␣␣␣␣
␣␣␣XO␣␣
␣␣␣␣␣␣␣
␣␣␣␣␣␣␣
␣␣␣␣␣␣␣
```



Retour sur affiche

```
// affiche O pour une case rouge, X pour une case jaune
static void affiche(int[][] grille)
{
  System.out.println();
  for(int[] ligne : grille) {
    System.out.print(" |");
    for(int cellule : ligne) {
      if (cellule == VIDE) {
        System.out.print(' ');
      } else if (cellule == ROUGE) {
        System.out.print('O');
      } else {
        System.out.print('X');
      }
      System.out.print('|');
    }
    System.out.println();
  }

  System.out.print('=');
  for(int i = 1; i <= grille[0].length; ++i) {
    System.out.print("=" +  i);
  }
  System.out.println("==\n");
}
```

```
| | | | | | | |
| | | | | | | |
| | | |X|O| | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
==1=2=3=4=5=6=7==
```



Jouer...

A ce stade, nous avons vu :

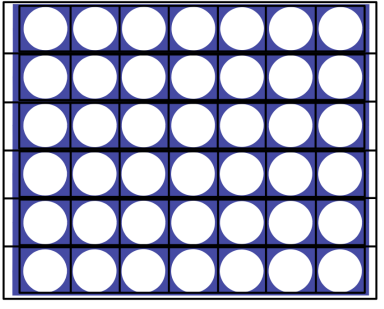
- les structures de données choisies
- 2 fonctionalités simples : `initialise` et `affiche`

☞ Voyons maintenant comment jouer :

- demander à un joueur où il joue
- valider son coup
- demander à l'autre joueur
  c'est-à-dire alterner les joueurs
- vérifier si l'un gagne (ou si le jeu est plein)

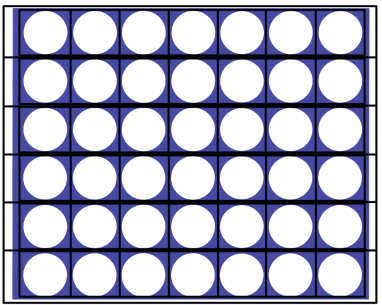
Méthode joue

```
static void joue(int[][] grille, int colonne, int couleur)
{
  // on parcourt la colonne en partant du bas jusqu'à trouver une case vide :
  int ligne = grille.length - 1;

  while (grille[ligne][colonne] != VIDE) {
    --ligne;
  }

  // on remplit la case vide trouvée :
  grille[ligne][colonne] = couleur;
}

...

joue(grille, 3, rouge);
joue(grille, 2, jaune);
joue(grille, 3, rouge);
```



Tester la méthode joue

```
static void joue(int[][] grille, int colonne, int couleur)
{
  // on parcourt la colonne en partant du bas jusqu'à trouver une case vide :
  int ligne = grille.length - 1;

  while (grille[ligne][colonne] != VIDE) {
    --ligne;
  }
  // on remplit la case vide trouvée :
  grille[ligne][colonne] = couleur;
}
...
static void  main()
{
  int[][] grille = new int[6][7];

  initialise(grille);
  affiche(grille);

  joue(grille, 3, ROUGE);
  affiche(grille);

  joue(grille, 2, JAUNE);
  affiche(grille);

  joue(grille, 3, ROUGE);
  affiche(grille);
}
```



Méthode joue

```
static  joue(int[][] grille, int colonne, int couleur)
{
  // on parcourt la colonne en partant du bas jusqu'à trouver une case vide,
  // ou jusqu'en haut de la colonne si la colonne est pleine :
  int ligne = grille.length - 1;

  boolean pleine = false;
  while ((!pleine) && (grille[ligne][colonne] != VIDE)) {
    if (ligne == 0) {
      pleine = true;
    } else {
      --ligne;
    }
  }

  // si on n'est pas arrivé jusqu'en haut de la colonne, on remplit la case vide trouvée,
  // sinon c'est que la colonne est pleine et le coup n'est pas valide :
  if (!pleine) {
    grille[ligne][colonne] = couleur;
    return true;
  } else {
    return false;
  }
}
```

Méthode joue

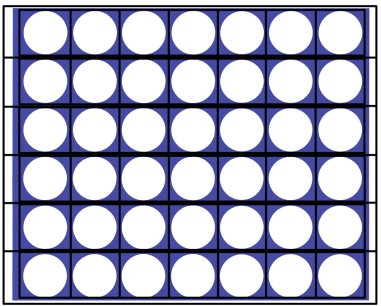
```
static boolean joue(int[][] grille, int colonne, int couleur)
{
  // on parcourt la colonne en partant du bas jusqu'à trouver une case vide,
  // ou jusqu'en haut de la colonne si la colonne est pleine :
  int ligne = grille.length - 1;

  boolean pleine = false;
  while ((!pleine) && (grille[ligne][colonne] != VIDE)) {
    if (ligne == 0) {
      pleine = true;
    } else {
      --ligne;
    }
  }

  // si on n'est pas arrivé jusqu'en haut de la colonne, on remplit la case vide trouvée,
  // sinon c'est que la colonne est pleine et le coup n'est pas valide :
  if (!pleine) {
    grille[ligne][colonne] = couleur;
    return true;
  } else {
    return false;
  }
}
```

Tester la nouvelle méthode `joue`

```
public static void main(String[] args)
{
  int[][] grille = new int[6][7];

  initialise(grille);
  affiche(grille);

  for(int i = 0; i < 10; ++i) {
    boolean  valide = joue(grille, 3, ROUGE);

    if (!valide) {
      System.out.println("impossible d'ajouter un pion sur cette colonne");
    }

    affiche(grille);
  }
}
```

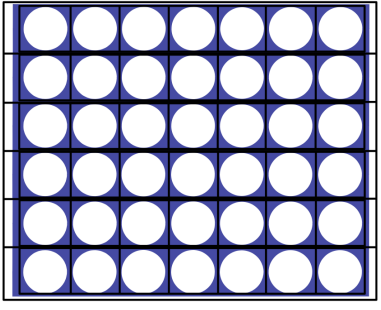
Tester la nouvelle méthode joue

```
static boolean joue(int[][] grille, int colonne, int couleur)
{
  int ligne = grille.length - 1;

  boolean pleine = false;
  while ((!pleine) && (grille[ligne][colonne] != VIDE)) {
    if (ligne == 0) {
      pleine = true;
    } else {
      --ligne;
    }
  }

  if (!pleine) {
    grille[ligne][colonne] = couleur;
    return true;
  } else {
    return false;
  }
}
...
  for(int i = 0; i < 10; ++i) {
    boolean valide = joue(grille, 3, ROUGE);
    if (!valide) {
      System.out.println("impossible d'ajouter un pion sur cette colonne");
    }
    affiche(grille);
  }
```



Une version alternative de la méthode joue

```
static boolean joue(int[][] grille, int colonne, int couleur)
{
  // si la colonne est pleine, le coup n'est pas valide :
  if (grille[0][colonne] != VIDE) {
    return false;
  }

  // on parcourt la colonne en partant du bas jusqu'à trouver une case vide :
  int ligne = grille.length - 1;
  while (grille[ligne][colonne] != VIDE) {
    --ligne;
  }

  // on remplit la case vide trouvée :
  grille[ligne][colonne] = couleur;
  return true;
}
```



Encore une autre version de la méthode joue

```
static boolean joue(int[][] grille, int colonne, int couleur)
{
  int ligne = grille.length - 1;

  // on parcourt la colonne en partant du bas jusqu'à trouver une case vide.
  //
  // Si le test (ligne >= 0) devient faux, c'est qu'on a
  // soustrait 1 à ligne quand elle valait 0, ce qui arrive quand la
  // colonne est pleine.
  while ((ligne >= 0) && (grille[ligne][colonne] != VIDE)) {
    --ligne;
  }

  // si ligne >= 0, on a trouvé une case vide, on la remplit,
  // sinon c'est que la colonne est pleine et le coup n'est pas valide :
  if (ligne >= 0) {
    grille[ligne][colonne] = couleur;
    return true;
  } else {
    return false;
  }
}
```

Jouer...

A ce stade, nous avons vu :

- les structures de données choisies (`Grille`, `Couleur`)
- 2 fonctionalités simples : `initialise` et `affiche`
- une fonctionalité plus avancée : `joue`

Pour jouer, il faut :

- demander à un joueur où il joue
- valider son coup
- demander à l'autre joueur
  c'est-à-dire alterner les joueurs
- vérifier si l'un gagne (ou si le jeu est plein)

Jouons !

A ce stade, nous avons vu :

- les structures de données choisies (`Grille`, `Couleur`)
- 2 fonctionalités simples : `initialise` et `affiche`
- 1 fonctionalité plus complexe : `joue`
  qui vérifie si un coup est valide et place le jeton dans le jeu

Mais pour vraiment jouer, il nous manque :

- demander à un joueur où il joue
- la boucle principale qui alterne les joueurs
- vérifier si l'un des deux gagne (ou si le jeu est plein)

Jouons !

```
private static Scanner clavier = new Scanner(System.in);

public static void main(String[] args)
{
  int[][] grille = new int[6][7];

  initialise(grille);
  affiche(grille);

  int couleurJoueur = JAUNE;
  do {
    if (couleurJoueur == JAUNE) {
      System.out.println("Joueur X :  entrez un numéro de colonne");
    } else {
      System.out.println("Joueur O :  entrez un numéro de colonne");
    }

    int colonne = clavier.nextInt();
}
```

Jouons !

```
public static void main(String[] args)
{
  int[][] grille = new int[6][7];

  initialise(grille);
  affiche(grille);

  int couleurJoueur = JAUNE;
  do {
    if (couleurJoueur == JAUNE) {
      System.out.println("Joueur X :  entrez un numéro de colonne");
    } else {
      System.out.println("Joueur O :  entrez un numéro de colonne");
    }

    int colonne = clavier.nextInt();
    // les indices des tableaux commencent par 0 en Java:
    --colonne;

    boolean valide = joue(grille, colonne, couleurJoueur);
    if (!valide) {
      System.out.println(" > Ce coup n'est pas valide");
    }

    affiche(grille);
}
```

Jouons !

```
  do {
    if (couleurJoueur == JAUNE) {
      System.out.println("Joueur X :  entrez un numéro de colonne");
    } else {
      System.out.println("Joueur O :  entrez un numéro de colonne");
    }

    int colonne = clavier.nextInt();
    // les indices des tableaux commencent par 0 en Java:
    --colonne;

    boolean valide = joue(grille, colonne, couleurJoueur);
    if (!valide) {
      System.out.println(" > Ce coup n'est pas valide");
    }

    affiche(grille);
    // on change la couleur pour la couleur de l'autre joueur:
    if (couleurJoueur == JAUNE) {
      couleurJoueur = ROUGE;
    } else {
      couleurJoueur = JAUNE;
    }
  } while(    );
}
```

Jouons !

```
  do {
    demandeEtJoue(grille, couleurJoueur);

    affiche(grille);
    // on change la couleur pour la couleur de l'autre joueur:
    if (couleurJoueur == JAUNE) {
      couleurJoueur = ROUGE;
    } else {
      couleurJoueur = JAUNE;
    }
  } while(    );
}
```

Méthode demandeEtJoue

```
void demandeEtJoue(int[][] grille, int couleurJoueur)
{
  boolean valide;

  do {
    System.out.print("Joueur ");
    if (couleurJoueur == JAUNE) {
      System.out.print("X");
    } else {
      System.out.print("O");
    }
    System.out.println(" :  entrez un numéro de colonne");

    int colonne = clavier.nextInt();
    // les indices des tableaux commencent par 0 en Java:
    --colonne;

    valide = joue(grille, colonne, couleurJoueur);
    if (!valide) {
      System.out.println(" > Ce coup n'est pas valide");
    }
  } while(!valide);
}
```

Retour sur la méthode joue

```
static boolean joue(int[][] grille, int colonne, int couleur)
{
  // Si le numéro de colonne n'est pas valide, le coup n'est pas valide :
  if (colonne >= grille[0].length) {
    return false;
  }

  // on parcourt la colonne en partant du bas jusqu'à trouver une case vide,
  // ou jusqu'en haut de la colonne si la colonne est pleine :
  int ligne = grille.length - 1;

  boolean pleine = false;
  while ((!pleine) && (grille[ligne][colonne] != VIDE)) {
    if (ligne == 0) {
      pleine = true;
    } else {
      --ligne;
    }
  }

  // si on n'est pas arrivé jusqu'en haut de la colonne, on remplit la case vide trouvée,
  // sinon c'est que la colonne est pleine et le coup n'est pas valide :
  if (!pleine) {
    grille[ligne][colonne] = couleur;
    return true;
  } else {
    return false;
  }
```

Où en sommes nous ?

A ce stade, nous avons fait :

- les structures de données choisies (`Grille`, `Couleur`)
- 2 fonctionalités simples : `initialise` et `affiche`
- 2 fonctionalité plus complexe : `joue` et `demandeEtJoue`
- la boucle principale qui alterne les joueurs

Mais pour vraiment jouer, il nous manque :

- vérifier si l'un des deux gagne (ou si le jeu est plein)

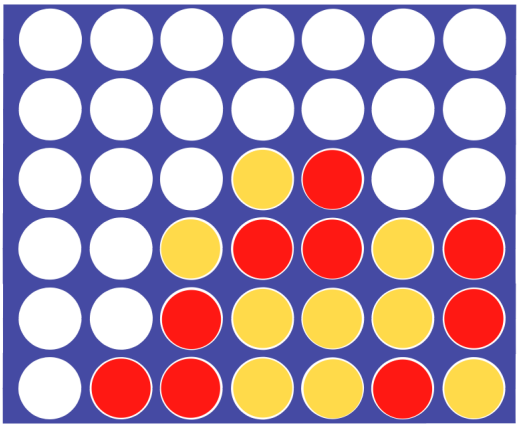
Retour sur la méthode `main`

```
int couleurJoueur = JAUNE;
boolean gagne;

do {
  demandeEtJoue(grille, couleurJoueur);

  affiche(grille);

  gagne = estCeGagne(grille, couleurJoueur);

  // on change la couleur pour la couleur de l'autre joueur:
  if (couleurJoueur == JAUNE) {
    couleurJoueur = ROUGE;
  } else {
    couleurJoueur = JAUNE;
  }
} while();
```

Retour sur la méthode `main`

```
int couleurJoueur = JAUNE;
boolean gagne;

do {
  demandeEtJoue(grille, couleurJoueur);

  affiche(grille);

  gagne = estCeGagne(grille, couleurJoueur);

  // on change la couleur pour la couleur de l'autre joueur:
  if (couleurJoueur == JAUNE) {
    couleurJoueur = ROUGE;
  } else {
    couleurJoueur = JAUNE;
  }
} while(!gagne);
```

Méthode `estCeGagne` : Stratégie



Méthode estCeGagne

```
// gagne = estCeGagne(grille, couleurJoueur);
static boolean estCeGagne(int[][] grille, int couleurJoueur)
{
  for(int ligne = 0; ligne < grille.length; ++ligne) {
    for(int colonne = 0; colonne < grille[ligne].length; ++colonne) {
      int couleurCase = grille[ligne][colonne];

      if (couleurCase == couleurJoueur) {
        if (
          // en diagonale, vers le haut et la droite:
          (compte(grille, ligne, colonne, -1, +1) >= 4) ||
          // horizontalement, vers la droite:
          (compte(grille, ligne, colonne, 0, +1) >= 4) ||
          // en diagonale, vers le bas et la droite:
          (compte(grille, ligne, colonne, +1, +1) >= 4) ||
          // verticalement, vers le bas:
          (compte(grille, ligne, colonne, +1, 0) >= 4)
        ) {
          return true;
        }
      }
    }
  }

  return false;
}
```

Méthode estCeGagne

```
// gagne = estCeGagne(grille, couleurJoueur);
static boolean estCeGagne(int[][] grille, int couleurJoueur)
{
  for(int ligne = 0; ligne < grille.length; ++ligne) {
    for(int colonne = 0; colonne < grille[ligne].length; ++colonne) {
      int couleurCase = grille[ligne][colonne];

      // pour chaque case qui contient un pion de la bonne couleur,
      // on compte les pions de la meme couleur dans 4 directions:
      if (couleurCase == couleurJoueur) {
        if (
          // en diagonale, vers le haut et la droite:
          (compte(grille, ligne, colonne, -1, +1) >= 4) ||
          // horizontalement, vers la droite:
          (compte(grille, ligne, colonne, 0, +1) >= 4) ||
          // en diagonale, vers le bas et la droite:
          (compte(grille, ligne, colonne, +1, +1) >= 4) ||
          // verticalement, vers le bas:
          (compte(grille, ligne, colonne, +1, 0) >= 4)
        ) {
          // si le nombre de pions pour au moins une des directions
          // est au moins 4, le joueur a gagne:
          return true;
        }
      }
    }
```

Méthode estCeGagne

```
      // pour chaque case qui contient un pion de la bonne couleur,
      // on compte les pions de la meme couleur dans 4 directions:
      if (couleurCase == couleurJoueur) {
        if (
          // en diagonale, vers le haut et la droite:
          (compte(grille, ligne, colonne, -1, +1) >= 4) ||
          // horizontalement, vers la droite:
          (compte(grille, ligne, colonne, 0, +1) >= 4) ||
          // en diagonale, vers le bas et la droite:
          (compte(grille, ligne, colonne, +1, +1) >= 4) ||
          // verticalement, vers le bas:
          (compte(grille, ligne, colonne, +1, 0) >= 4)
        ) {
          // si le nombre de pions pour au moins une des directions
          // est au moins 4, le joueur a gagne:
          return true;
        }
      }
    }
  }

  // si on a parcouru toute la grille sans trouver au moins 4 pions
  // alignes, le joueur n'a pas gagne, du moins pas encore:
  return false;
}
```

Méthode compte

```
// if (compte(grille, ligne, colonne, -1, +1) >= 4 ...
static int compte(int[][] grille,
                  int ligneDepart, int colonneDepart,
                  int dirLigne, int dirColonne)
{
  int compteur = 0;

  int ligne = ligneDepart;
  int colonne = colonneDepart;

  while (grille[ligne][colonne] == grille[ligneDepart][colonneDepart] &&
         ligne   >= 0 && ligne   < grille.length &&
         colonne >= 0 && colonne < grille[ligne].length) {

    ++compteur;
    ligne   = ligne   + dirLigne;
    colonne = colonne + dirColonne;

  }

  return compteur;
}
```

Méthode compte

```
static int compte(int[][] grille,
                  int ligneDepart, int colonneDepart,
                  int dirLigne, int dirColonne)
{
  int compteur = 0;

  int ligne = ligneDepart;
  int colonne = colonneDepart;

  // on part de la case (ligneDepart, colonneDepart) et on parcourt la grille
  // dans la direction donnee par (dirLigne, dirColonne)
  // tant qu'on trouve des pions de la meme couleur que le pion de depart.
  while (grille[ligne][colonne] == grille[ligneDepart][colonneDepart] &&
         ligne   >= 0 && ligne   < grille.length &&
         colonne >= 0 && colonne < grille[ligne].length) {

    ++compteur;
    ligne   = ligne   + dirLigne;
    colonne = colonne + dirColonne;

  }

  return compteur;
}
```

Retour sur la méthode `estCeGagne`

```
static boolean estCeGagne(int[][] grille, int couleurJoueur)
{
  for(int ligne = 0; ligne < grille.length; ++ligne) {
    for(int colonne = 0; colonne < grille[ligne].length; ++colonne) {
      int couleurCase = grille[ligne][colonne];

      if (couleurCase == couleurJoueur) {
        if (
          // en diagonale, vers le haut et la droite:
          (compte(grille, ligne, colonne, -1, +1) >= 4) ||

          // horizontalement, vers la droite:
          (compte(grille, ligne, colonne, 0, +1) >= 4) ||

          // en diagonale, vers le bas et la droite:
          (compte(grille, ligne, colonne, +1, +1) >= 4) ||

          // verticalement, vers le bas:
          (compte(grille, ligne, colonne, +1, 0) >= 4)
        ) {
          return true;
        }
      }
    }
  }
```

Retour sur la méthode `estCeGagne`

```
{
  for(int ligne = 0; ligne < grille.length; ++ligne) {
    for(int colonne = 0; colonne < grille[ligne].length; ++colonne) {
      int couleurCase = grille[ligne][colonne];

      if (couleurCase == couleurJoueur) {
        if (
          // en diagonale, vers le haut et la droite:
          (compte(grille, ligne, colonne, -1, +1) >= 4) ||

          // horizontalement, vers la droite:
          (compte(grille, ligne, colonne, 0, +1) >= 4) ||

          // en diagonale, vers le bas et la droite:
          (compte(grille, ligne, colonne, +1, +1) >= 4) ||

          // verticalement, vers le bas:
          (compte(grille, ligne, colonne, +1, 0) >= 4)
        ) {
          return true;
        }
      }
    }
  }
```

Retour sur la méthode estCeGagne

```
for(int ligne = 0; ligne < grille.length; ++ligne) {
  for(int colonne = 0; colonne < grille[ligne].length; ++colonne) {
    int couleurCase = grille[ligne][colonne];

    if (couleurCase == couleurJoueur) {
      if (
        // en diagonale, vers le haut et la droite:
        (compte(grille, ligne, colonne, -1, +1) >= 4) ||

        // horizontalement, vers la droite:
        (compte(grille, ligne, colonne, 0, +1) >= 4) ||

        // en diagonale, vers le bas et la droite:
        (compte(grille, ligne, colonne, +1, +1) >= 4) ||

        // verticalement, vers le bas:
        (compte(grille, ligne, colonne, +1, 0) >= 4)
      ) {
        return true;
      }
    }
  }
}

return false;
```

Retour sur la méthode estCeGagne

```
    for(int colonne = 0; colonne < grille[ligne].length; ++colonne) {
      int couleurCase = grille[ligne][colonne];

      if (couleurCase == couleurJoueur) {
        if (
          // en diagonale, vers le haut et la droite:
          (compte(grille, ligne, colonne, -1, +1) >= 4) ||

          // horizontalement, vers la droite:
          (compte(grille, ligne, colonne, 0, +1) >= 4) ||

          // en diagonale, vers le bas et la droite:
          (compte(grille, ligne, colonne, +1, +1) >= 4) ||

          // verticalement, vers le bas:
          (compte(grille, ligne, colonne, +1, 0) >= 4)
        ) {
          return true;
        }
      }
    }
  }

  return false;
}
```

Retour sur la méthode estCeGagne

```
      int couleurCase = grille[ligne][colonne];

      if (couleurCase == couleurJoueur) {
        if (
          // en diagonale, vers le haut et la droite:
          (compte(grille, ligne, colonne, -1, +1) >= 4) ||

          // horizontalement, vers la droite:
          (compte(grille, ligne, colonne, 0, +1) >= 4) ||

          // en diagonale, vers le bas et la droite:
          (compte(grille, ligne, colonne, +1, +1) >= 4) ||

          // verticalement, vers le bas:
          (compte(grille, ligne, colonne, +1, 0) >= 4)
        ) {
          return true;
        }
      }
    }
  }

  return false;
}
```

Retour sur la méthode `estCeGagne`

```
      if (couleurCase == couleurJoueur) {
        if (
          // en diagonale, vers le haut et la droite:
          (compte(grille, ligne, colonne, -1, +1) >= 4) ||

          // horizontalement, vers la droite:
          (compte(grille, ligne, colonne, 0, +1) >= 4) ||

          // en diagonale, vers le bas et la droite:
          (compte(grille, ligne, colonne, +1, +1) >= 4) ||

          // verticalement, vers le bas:
          (compte(grille, ligne, colonne, +1, 0) >= 4)
        ) {
          return true;
        }
      }
    }
  }

  return false;
}
```

Retour sur la méthode `estCeGagne`

```
      if (couleurCase == couleurJoueur) {
        if (
          // en diagonale, vers le haut et la droite:
          (compte(grille, ligne, colonne, -1, +1) >= 4) ||

          // horizontalement, vers la droite:
          (compte(grille, ligne, colonne, 0, +1) >= 4) ||

          // en diagonale, vers le bas et la droite:
          (compte(grille, ligne, colonne, +1, +1) >= 4) ||

          // verticalement, vers le bas:
          (compte(grille, ligne, colonne, +1, 0) >= 4)
        ) {
          return true;
        }
      }
    }
  }

  return false;
}
```

Retour sur la méthode `estCeGagne`

```
      if (couleurCase == couleurJoueur) {
        if (
          // en diagonale, vers le haut et la droite:
          (ligne >= 3 && colonne <= grille[ligne].length - 4 &&
           compte(grille, ligne, colonne, -1, +1) >= 4) ||

          // horizontalement, vers la droite:
          (colonne <= grille[ligne].length - 4 &&
           compte(grille, ligne, colonne, 0, +1) >= 4) ||

          // en diagonale, vers le bas et la droite:
          (ligne <= grille.length - 4 && colonne <= grille[ligne].length - 4 &&
           compte(grille, ligne, colonne, +1, +1) >= 4) ||

          // verticalement, vers le bas:
          (ligne <= grille.length - 4 &&
           compte(grille, ligne, colonne, +1, 0) >= 4)
        ) {
          return true;
        }
      }
    }
  }

  return false;
```

Retour sur la méthode `estCeGagne`

```
    if (couleurCase == couleurJoueur) {
      final int ligneMax = grille.length - 4;
      final int colonneMax = grille[ligne].length - 4;

      if (
        // en diagonale, vers le haut et la droite:
        (ligne >= 3 && colonne <= colonneMax &&
         compte(grille, ligne, colonne, -1, +1) >= 4) ||

        // horizontalement, vers la droite:
        (colonne <= colonneMax &&
         compte(grille, ligne, colonne, 0, +1) >= 4) ||

        // en diagonale, vers le bas et la droite:
        (ligne <= ligneMax && colonne <= colonneMax &&
         compte(grille, ligne, colonne, +1, +1) >= 4) ||

        // verticalement, vers le bas:
        (ligne <=  &&
         compte(grille, ligne, colonne, +1, 0) >= 4)
      ) {
        return true;
      }
    }
  }
```

Retour sur la méthode `estCeGagne`

```
    if (couleurCase == couleurJoueur) {
      final int ligneMax = grille.length - 4;
      final int colonneMax = grille[ligne].length - 4;

      if (
        // en diagonale, vers le haut et la droite:
        (ligne >= 3 && colonne <= colonneMax &&
         compte(grille, ligne, colonne, -1, +1) >= 4) ||

        // horizontalement, vers la droite:
        (colonne <= colonneMax &&
         compte(grille, ligne, colonne, 0, +1) >= 4) ||

        // en diagonale, vers le bas et la droite:
        (ligne <= ligneMax && colonne <= colonneMax &&
         compte(grille, ligne, colonne, +1, +1) >= 4) ||

        // verticalement, vers le bas:
        (ligne <= ligneMax &&
         compte(grille, ligne, colonne, +1, 0) >= 4)
      ) {
        return true;
      }
    }
  }
```

Retour sur la méthode `main`

```
    demandeEtJoue(grille, couleurJoueur);

    affiche(grille);

    gagne = estCeGagne(grille, couleurJoueur);

    // on change la couleur pour la couleur de l'autre joueur:
    if (couleurJoueur == JAUNE) {
      couleurJoueur = ROUGE;
    } else {
      couleurJoueur = JAUNE;
    }
  } while(!gagne);

  // attention, on a change la couleur pour la couleur de l'autre joueur !
  if (couleurJoueur == JAUNE) {
    System.out.println("Le joueur O a gagne !");
  } else {
    System.out.println("Le joueur X a gagne !");
  }
}
```

Fini ?

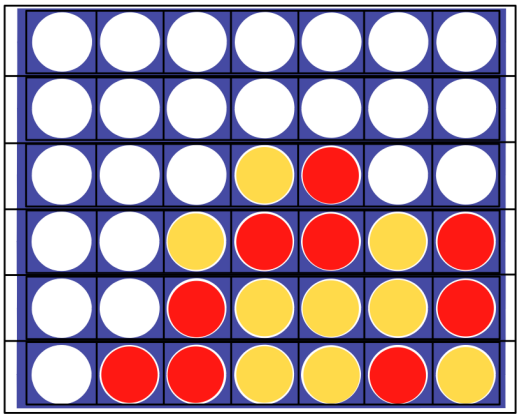
Retour sur la méthode `main`

```
    demandeEtJoue(grille, couleurJoueur);

    affiche(grille);

    gagne = estCeGagne(grille, couleurJoueur);

    // on change la couleur pour la couleur de l'autre joueur:
    if (couleurJoueur == JAUNE) {
      couleurJoueur = ROUGE;
    } else {
      couleurJoueur = JAUNE;
    }
  } while(!gagne && !plein(grille));

   // attention, on a change la couleur pour la couleur de l'autre joueur!
  if (couleurJoueur == JAUNE) {
    System.out.println("Le joueur O a gagne!");
  } else {
    System.out.println("Le joueur X a gagne!");
  }
}
```

Retour sur la méthode `main`

```
    // on change la couleur pour la couleur de l'autre joueur:
    if (couleurJoueur == JAUNE) {
      couleurJoueur = ROUGE;
    } else {
      couleurJoueur = JAUNE;
    }
  } while(!gagne && !plein(grille));

  if (gagne) {
    // attention, on a change la couleur pour la couleur de l'autre joueur !
    if (couleurJoueur == JAUNE) {
      System.out.println("Le joueur O a gagne !");
    } else {
      System.out.println("Le joueur X a gagne !");
    }
  } else {
    System.out.println("Match nul !");
  }
}
```

Fonction `plein`



Méthode `plein`

```java
// plein(grille)
static boolean plein(int[][] grille)
{
  // Si on trouve une case vide sur la premiere ligne, la grille n'est pas pleine :
  for(int cellule : grille[0]) {
    if (cellule == VIDE) {
      return false;
    }
  }

  // Sinon, la grille est pleine :
  return true;
}
```