Support de cours

Cours:

Initiation à la programmation (en Java)

Vidéo:

Init-JAVA-07-3-joue2-pt1

Concepts (extraits des sous-titres générés automatiquement) :

Méthode joue. Différentes lignes. Séquence vidéo précédente. Case vide. Coup valide. Tableau d'entiers. Dernière ligne du tableau. Entête de la méthode joue. Première case vide. Taille de grille. Version de la méthode joue. Problème de départ. Ligne. Version de la vidéo précédente. Séquence vidéo.

vers la recherche de séquences vidéo
(dans Initiation à la programmation (en Java).)

vers la vidéo

EPFL

Puissance 4 : révision de la méthode `joue`

(Partie 1)

Initiation à la programmation (Java)

Jamila Sam, Vincent Lepetit et Jean-Cédric Chappelier

...

notes

résumé

0m 0s

Dans la séquence vidéo précédente,

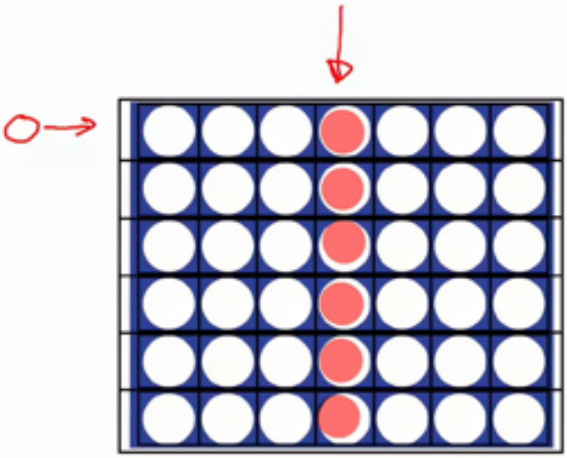
notes

résumé

0m 1s

Une version alternative de la méthode joue

```
static boolean joue(int[][] grille, int colonne, int couleur)
{
  |
}
```



nous avons vu une façon de programmer la méthode joue dont le but est de vérifier si une colonne indiquée par un joueur est un coup valide. Et si c'est le cas de, effectivement, mettre le pion dans la grille. Dans cette séquence vidéo-ci, je vous propose d'étudier 2 alternatives pour programmer cette méthode joue. Commençons donc par l'entête de la méthode joue. Il est clair que si on veut changer l'algorithme, qu'on veut changer le corps de la méthode, l'entête, lui, va rester le même. Et je vous rappelle donc que l'on reçoit une grille, un tableau d'entiers, qui est le jeu dans lequel on veut jouer, Donc ensuite un entier qui indique la colonne dans laquelle le joueur joue. Et enfin un troisième argument ici entier, qui indique la couleur du pion joué dans la colonne. Enfin la méthode joue retourne un boolean qui renvoie vrai, true, si le coup est valide et qui renvoie faux si le coup n'est pas valide. Changeons donc l'algorithme par rapport à la version précédente de la méthode joue et commençons par tester tout de suite la validité de ce coup. Le coup n'est valide que si la case en la position 0, en ligne 0, n'est pas pleine. En effet, si la case en position 0 dans la colonne dans laquelle on veut jouer n'est pas vide, c'est bien qu'il y avait des pions en dessous et que la colonne est pleine.

notes

résumé

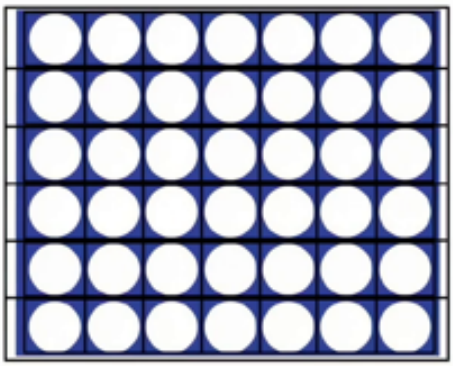
0m 6s

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;
}
```



Donc, comme d'habitude, commençons par écrire le commentaire, qui explique ce que l'on va faire. Si la colonne est pleine, alors le coup n'est pas valide. Et maintenant écrivons-le en JAVA. Si, if, la colonne est pleine, c'est-à-dire que la grille, à la position 0, et dans la colonne que l'on a reçue, n'est pas vide, c'est-à-dire n'est pas égale à la valeur qu'on avait prédéfinie vide, alors à ce moment-là le coup n'est pas valide, et il suffit simplement de quitter la méthode en retournant false. Voila qui simplifie donc grandement l'exploration ensuite de la grille pour placer le coup, puisque à ce stade-là, soit on est sorti avec false, soit on sait que la colonne n'est pas pleine et donc on va forcément trouver une case vide que l'on peut maintenant commencer à rechercher. Donc comme d'habitude, on met le commentaire, on va parcourir la colonne de bas en haut, pour rechercher la première case vide. Ce qui s'écrit donc en JAVA : on commence par nommer une variable ici, ligne pour parcourir les différentes lignes. Donc on initialise cette ligne à la dernière ligne du tableau, c'est à dire à la taille de grille - 1 donc je vous rappelle que les tableaux ont des index qui vont de 0 jusqu'à "taille -1". Donc ici, on initialise ligne à la taille de la grille - 1, puis ensuite on va parcourir le tableau en remontant. Donc tant que la grille à la position de la ligne qu'on est en train d'utiliser pour parcourir et de la colonne que l'on a reçu, n'est pas vide, à ce moment-là on décrémente la ligne, donc --ligne. A ce stade, on a donc trouvé une case vide et il suffit maintenant d'y jouer. On écrit comme d'habitude le commentaire et on écrit donc que grille, à la position ligne, colonne reçoit la couleur que l'on souhaite jouer. Donc le paramètre couleur

notes

résumé

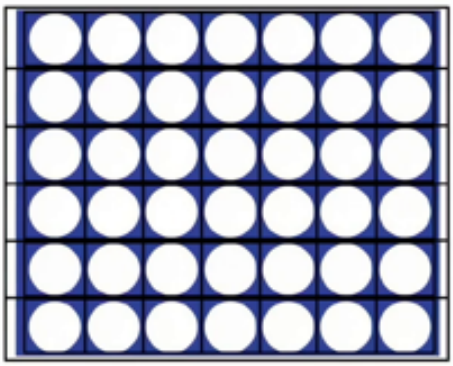
1m 25s

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;
}
```



que l'on a reçu. Et on peut terminer en renvoyant donc le fait que l'on a pu jouer, donc en renvoyant true. Voilà donc notre nouvelle version de la méthode joue

notes

résumé

et je vous laisse apprécier donc, indépendamment d'ailleurs de la taille des commentaires, la différence entre les deux méthodes.

notes

résumé

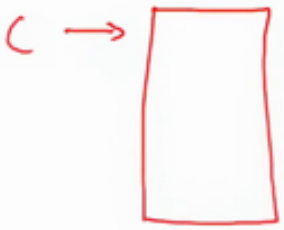
3m 37s

Encore une autre version de la méthode joue

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

  while ((|) && (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;
  }
}
```



Terminons enfin avec encore une autre version de la méthode joue qui cette fois repart de la version de la vidéo précédente, et va la remanier. Donc nous retrouvons ici la version de la vidéo précédente et l'idée est de s'abstenir d'introduire ce boolean ici pleine et d'essayer de faire autrement. Donc pour ça on va le supprimer, simplement, mais par quoi le remplacer ? Revenons au problème de départ, et je vous rappelle que donc, on avait introduit ce booleanpleine, en raison du fait que l'on décrémentait encore la ligne

notes

résumé

3m 49s

Encore une autre version de la méthode joue

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

  while ((ligne >= 0) && (grille[ligne][colonne] != VIDE)) {
    --ligne;
  }

  if (!pl) {
    grille[ligne][colonne] = couleur;
    return true;
  } else {
    return false;
  }
}
```

dans la toute toute première version lorsque celle-ci atteignait la position 0. Donc quand ligne était nulle, on continuait avec la première version à faire des -- et donc à descendre ici, hors du tableau. Donc il suffit maintenant simplement de l'écrire et de dire que on veut s'arrêter lorsque ligne est 0. On veut continuer tant que ligne est supérieure ou égale à 0. Dès que ligne atteindra 0, on veut que la boucle s'arrête. Donc on l'écrit, tant que ligne est >= 0, le 0 doit bien sûr être considéré, tant que ligne est supérieure ou égale à 0. Donc on peut continuer à supprimer toutes les utilisations de pleine. Voilà, ça simplifie donc notre boucle, quand même, et puis on va faire de la même façon, en bas, dans la partie finale, où on joue,

notes

résumé

4m 25s

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;
  }
}
```

et on va remettre le test ici, le même. Donc if ligne >= 0, en effet, si à ce stade là la ligne est supérieure ou égale à 0, c'est donc qu'on est bien dans le tableau, et donc on peut tout à fait jouer.

notes

résumé

5m 13s

Et puis, sinon, c'est qu'on est hors du tableau. Voilà donc la version finale avec les commentaires, bien sûr, qu'on ne se prive pas de rajouter pour rendre notre code compréhensible. Ce qui termine donc notre troisième version de la méthode joue. Ceci dit, il reste encore une erreur potentielle dans toutes nos versions de méthodes joue. Est-ce que vous pouvez voir de quoi il s'agit ? de quoi il s'agit ?

notes

résumé

5m 31s