Support de cours

Cours:

Initiation à la programmation (en Java)

Vidéo:

Init-JAVA-07-4-main-pt2

Concepts (extraits des sous-titres générés automatiquement) :

Tableau ligne colonne. Jeu complet. Séquence vidéo. Variable valide. Portée de la variable. Numéro de colonne. Méthode demandeetjoue. Genre de test. Test d'arrêt de la boucle. Séquences vidéos précédentes. Index de la grille. Répétition du jeu. Méthode joue. Exemple grille. Taille de la grille.

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

vers la vidéo

EPFL

Puissance 4 : moteur de jeu
(Partie 2)

Initiation à la programmation (Java)

Jamila Sam, Vincent Lepetit et Jean-Cédric Chappelier

...

notes

résumé

0m 0s

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);
}
```

Le code ne compile pas parce que la portée de la variable valide ici, qui est déclarée dans ce bloc, est justement tout ce bloc ici. Et donc la variable valide ici n'est pas définie. Donc il faut sortir la variable valide, la déclaration de la variable valide de la boucle, et la mettre avant la boucle pour pouvoir l'utiliser dans la condition du while. Evidemment si l'on fait ceci, il faut faire attention au fait que maintenant on a donc deux déclarations de valide, une ici, et une ici. Donc il faut bien sûr corriger celle-ci et la supprimer et on a donc plus qu'une affectation dans la boucle.

notes

résumé

0m 1s

Voilà, ce qui termine donc notre méthode demandeEtJoue, cette fois-ci, elle compile et elle devrait s'exécuter correctement.

notes

résumé

0m 49s

Retour sur la 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;
  }
}
```

Terminons cette séquence vidéo par un dernier retour à la méthode joue. En réponse à l'un des quizz posé dans une des séquences vidéos précédentes, et donc regardons ici ce qu'il fallait corriger à cette méthode joue. Le problème ici c'est qu'on accède à un tableau ligne colonne,

notes

résumé

0m 56s

sans avoir vérifier effectivement que la colonne que l'on reçoit est correcte.

notes

résumé

1m 13s

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) {
    retu
  // 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;
  }
}
```

On aurait pu imaginer que ce test puisse se faire avant d'appeler la méthode joue. On aurait par exemple pu le faire lorsque l'on demande au joueur d'entrer un numéro de colonne. S'il entre un numéro qui est trop grand, genre 100, on lui dit non ce coup n'est pas valide. Mais il est préférable de mettre ce genre de test au plus près de danger qu'il protège, et donc ici de le mettre dans la méthode joue qui reçoit donc ici comme paramètre une colonne, et donc de garantir et de vérifier que cette colonne est compatible avec son utilisation ici comme index de la grille. Et donc tout simplement, on va commencer, par évidemment comme d'habitude, mettre le commentaire et dire que si le numéro de colonne est plus grand que la taille n'est pas valide alors cela veut dire que le coup n'est pas valide. Ce qui s'écrit donc en java : Si colonne, que l'on a reçu donc comme paramètre, est plus grand que la taille de la grille, donc par exemple grille[0].length, alors à ce moment-là, tout simplement, on return false

notes

résumé

1m 19s

puisque le coup n'est pas valide. Voilà, ce qui conclut cette séquence vidéo. Il ne reste plus, pour avoir un jeu complet, qu'à implémenter le test d'arrêt de la boucle de répétition du jeu. Ce qui sera l'objet de la séquence vidéo suivante. la séquence vidéo suivante.

notes

résumé

2m 13s