Support de cours

Cours:

Initiation à la programmation (en Java)

Vidéo:

Init-JAVA-07-6-final-OG

Concepts (extraits des sous-titres générés automatiquement) :

Existence d'une méthode. Cas de la grille pleine. Programme principal. Sortir de la boucle. Couleur du joueur alterne. Situations de match nul. Terme de notre étude de cas. Première chose naturelle. Pions jaunes. Fin de chaque itération de la boucle. Codage effectif de la méthode. Encodage de la méthode. Cases de la dernière ligne. Ensemble des cas. Dernières instructions.

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

vers la vidéo

EPFL

Puissance 4 : finalisation

Initiation à la programmation (Java)

Jamila Sam, Vincent Lepetit et Jean-Cédric Chappelier

...

notes

résumé

0m 0s

Nous sommes arrivés à ce stade presque au terme de notre étude de cas « Puissance 4 »;

notes

résumé

0m 1s

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);
```

nous savons faire jouer en alternance les deux joueurs nous savons tester si un joueur a gagné. Il nous reste à peaufiner un peu le programme principal pour indiquer lequel des deux joueurs a gagné et pour faire face aux situations de match nul. C'est l'objet de la séquence d'aujourd'hui.

notes

résumé

0m 6s

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);
```

joueur X → jaune
O → rouge

Pour rappel notre jeu de puissance 4 tel que prévu jusqu'ici fait jouer en boucle deux joueurs : le joueur 'X' qui joue les pions jaunes et le joueur 'O' qui joue les pions rouges, les deux joueurs jouent en alternance c'est-à-dire qu'à la fin de chaque itération de la boucle on garantit que la couleur du joueur alterne entre le jaune et le rouge, et les deux joueurs continuent de jouer jusqu'à ce que l'un des deux ait gagné. Ce programme mérite cependant d'être un peu peaufiné. La première chose naturelle à faire est lorsque nous sortons de la boucle, ce qui signifie que l'un ou l'autre des joueurs a gagné, de faire afficher au programme lequel des deux est effectivement le vainqueur ; la seconde chose à faire est de faire face à la situation où les deux joueurs jouent en alternance sans gagner jusqu'à remplir complètement la grille et à ce moment-là il n'y a pas de gagnant, il s'agit d'une situation de match nul. Cette situation n'est pas couverte par le programme actuel

notes

résumé

0m 21s

Retour sur la fonction `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.prin
  }
```

ce qui ne correspond pas à une situation très réaliste. Commençons par modifier notre programme de sorte à ce qu'il affiche qui est le vainqueur donc il s'agit naturellement d'ajouter au programme après la boucle un certain nombre d'instructions, ces dernières vont naturellement tester la couleur du joueur lorsque l'on sort de la boucle et si cette couleur est jaune ceci signifie que le joueur vainqueur est le joueur rouge. Pourquoi le joueur rouge ici et non pas le jaune ? Et bien, pour rappel, les dernières instructions que nous exécutons avant de sortir de la boucle sont celles qui consistent à faire alterner la couleur du joueur. Si cette couleur a été alternée de sorte à ce que la couleur en sortant de la boucle soit la couleur jaune, ce qui correspondrait à cette situation, cela signifie que le joueur qui effectivement gagne la partie est celui de la couleur alternative donc ici la couleur rouge ce qui explique pourquoi lorsque nous sortons de la boucle avec la couleur jaune cela signifie bien que ce soit le joueur rouge qui a effectivement gagné. Donc ici il s'agit bien sûr de compléter pour couvrir l'ensemble des cas et donc dans l'autre cas il faudra simplement faire afficher le fait que le joueur gagnant est cette fois bien le joueur jaune qui correspond au joueur 'X' dans notre modification. Voilà nous avons ainsi couvert l'ensemble des cas. Nous prenons ici le soin de commenter ce test un petit peu contre intuitif en indiquant que au préalable on a effectivement changé la couleur du joueur ce qui explique la nature du test que nous effectuons ici. Comme indiqué au début de cette séquence il peut aussi se présenter des situations où les deux joueurs vont jouer en alternance et vont remplir totalement la grille sans pour autant que l'un des deux ne remporte la partie ; il va

notes

résumé

1m 13s

Retour sur la fonction `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.prin
  }
```

falloir donc compléter notre code de sorte à pouvoir également considérer cette situation c'est-à-dire qu'il va falloir sortir de la boucle si l'un des deux a gagné, ce qui est couvert par le test actuel mais également sortir de la boucle si la grille est complètement pleine, ce qui n'est pas couvert et qui correspond à une situation de match nul. Que peut-il se produire dans le cas où nous ne considérons pas le cas de la grille pleine. Et bien ce qui risque de se produire c'est qu'une fois que la grille est remplie nous allons demander à un joueur de jouer un coup valide sur la grille, ce qui sera impossible, et donc nous allons boucler indéfiniment sans qu'il ne puisse jamais y avoir de joueur gagnant, nous allons sans cesse demander à un joueur d'introduire un pion valide, ce qui lui est impossible, vu que la grille est complètement remplie et dans ce cas-là nous ne pourrons jamais sortir de la boucle. Ceci correspond bien évidemment à un comportement erroné du programme auquel il faut remédier pour que le programme puisse se dérouler correctement dans toutes les situations possibles. Les modifications a apporter à notre programme pour tester le cas de la grille pleine sont assez naturelles. Donc nous allons devoir répéter les traitements « demander aux joueurs de jouer en alternance » tant qu'aucun des deux n'a gagné, ce qui est déjà testé actuellement, et tant que la grille n'est pas pleine ; ce qui peut s'écrire assez naturellement comme ceci. Et nous anticipons donc ici l'existence d'une méthode « plein » qui va nous aider à tester si la grille est déjà remplie ou non. Cette modification aura une incidence sur le déroulement du reste du programme. En effet, désormais nous ne serons plus dans la situation unique possible où il y a fatalement un gagnant, mais

notes

résumé

Retour sur la fonction `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.prin
}
```

il y a également la situation où nous sortons de la boucle sans qu'il ait de gagnant, ce qu'il faut aussi signaler de façon appropriée. Ici, il faudrait modifier notre programme de sorte à couvrir également cette situation, donc en testant : est-ce que l'on est sorti de la boucle parce qu'il y a un gagnant et dans ce cas là effectivement nous devons signaler le gagnant comme nous le faisions tout à l'heure. Sinon, cela signifie qu'il y a une situation désormais de match nul

notes

résumé

Fonction plein

et nous devons également le signaler par un message approprié. Voilà nous aurons ainsi couvert tous les cas possibles. Voici l' encodage de la méthode « plein », donc on a on va demander en alternance au joueur rouge et au joueur jaune sur quelle colonne il veut placer son jeton, supposons que la main soit au joueur jaune si ce dernier décide de jouer sur la première colonne et bien le remplissage se fait automatiquement des bas en haut ce qui veut dire que nous allons effectivement remplir dans le cas présent la toute première case de la grille ; ceci a une incidence sur la façon de tester que la grille est pleine il n'est en effet plus nécessaire de tester l'intégralité des cases de la grille pour savoir si elles sont d'une couleur donnée ou si elles sont vides il suffira de tester les cases de la dernière ligne remplie de la grille de jeu. En effet dans cette situation si l'intégralité des cases de cette grille ont déjà une couleur, cela signifie que l'on a déjà effectué tout le remplissage en partant du bas et cela signifie effectivement que la grille est déjà pleine. Intéressons-nous maintenant au codage effectif de la méthode « plein ». Nous avons vu que du point de vue de la méthodologie lorsque nous abordons le codage d'une méthode, il faut toujours d'abord se poser la question du « quoi »,

notes

résumé

5m 1s

Fonction plein

```
// plein(grille)
static boolean plein(int
```

que fait la méthode, de quelles données a-t-elle besoin pour pouvoir travailler que livre-t-elle comme informations assorties, ce sont des éléments que nous avons abordé ici de façon très rapide parce qu'il s'agit là d'une fonction très simple au travers de notre façon de l'anticiper, d'anticiper son utilisation, dans le programme principal ; donc nous avons anticipé que nous allions invoquer cette méthode « plein » sur une grille et qu'en sortie évidemment elle allait nous sortir un résultat qui vraisemblablement allait être un booléan c'est-à-dire une valeur booléenne

notes

résumé

6m 13s

Fonction plein

```
// plein(grille)
static boolean plein(int[][] grille)
{
  // Si |
```

nous indiquant si la grille est effectivement pleine. Ceci correspond assez naturellement à cet entête-ci pour notre méthode « plein ». Le « quoi » étant clairement établi nous pouvons nous intéresser au comment, comment effectivement programmer le corps de cette méthode « plein », cet un algorithme que nous avons esquissé dans notre discussion de tout à l'heure

notes

résumé

6m 43s

Fonction plein

```
// 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;
```

à savoir que si l'on trouve sur la première ligne, c'est-à-dire la dernière dans le sens du remplissage, une case vide, cela signifie tout simplement que la grille n'est pas pleine. C'est ce traitement que nous allons mettre en œuvre, ceci peut naturellement se mettre en œuvre au travers d'une « boucle for », une itération sur un ensemble de valeurs, nous allons itérer sur l'ensemble des cellules de la première ligne, ce qui s'écrit « grille[0] », et tester si la cellule en question est vide cela signifie que la grille n'est pas pleine et donc nous devons sortir de notre méthode « plein » en indiquant que la grille n'est pas pleine, en retournant « false ».

notes

résumé

7m 3s

Si nous avons parcouru l'intégralité des cellules de la première ligne sans jamais trouver de cellules vides alors nous sortons de la boucle et à ce moment-là nous pouvons signifier que la grille est pleine et nous devons faire retourner à notre méthode « plein » le résultat « true » ; à noter que parmi les erreurs fréquentes chez les débutants dans ce genre de situations est d'écrire l'algorithme, de le retranscrire de façon erronée, en testant si la cellule est vide alors retourner « false », ce qui correspond à quelque chose de correct, mais de tout de suite après retourner un « true » ce qui évidemment ne correspond pas à l'algorithme que nous voulons mettre en œuvre. Pourquoi ? Parce qu'il faut bien sûr avoir testé l'intégralité des cases de la première ligne de la grille avant de pouvoir statuer sur le fait que la grille est effectivement pleine ; et ceci ne peut pas être connu à l'intérieur du corps de la boucle mais on doit d'abord sortir de la boucle avant de pouvoir retourner « true ». On fait juste attention à ce genre de petits détails que l'on retrouve assez souvent dans les copies des débutants. Voilà ceci termine le codage de notre petite application de « Puissance 4 », il y a bien évidemment de très nombreuses façons de programmer cette application et nous n'avons fait ici qu'en présenter une parmi d'autres ; le but de cette petite étude de cas était essentiellement de vous montrer comment on aborde la programmation d'une telle application lorsqu'on le fait pour la première fois. Plusieurs idées clé à retenir : la première, toujours procéder de bas en haut, commencer par identifier les types essentiels les types de données essentielles, les fonctions fondamentales, avant d'affiner. Chacune des fonctionnalités et méthodes fondamentales peut à son tour être modularisée c'est-à-dire

notes

résumé

7m 37s

faire appel à d'autres méthodes, plus de détails, qui vont l'aider à réaliser les traitements souhaités. Nous avons vu que pour le codage de chacune des fonctionnalités de chacune des méthodes il est souhaitable de procéder avec une certaine méthodologie, se poser toujours la question du « quoi », que fait la méthode, de quelles informations elle a besoin pour pouvoir travailler quelles sont les informations qu'elle livre un retour ; ensuite seulement se poser la question du comment et réfléchir aux algorithmes qui permettront d'implémenter les traitements souhaités. Enfin nous avons vu qu'il fallait aller à l'essentiel avant de nous intéresser aux aspects de prospectique, de peaufinage, et point tout à fait fondamental, qu'il fallait systématiquement tester au fur et à mesure que l'on progresse dans le codage de l'application. Voilà, c'était un certain nombre de messages que nous souhaitions faire passer au travers de cette étude de cas et qui nous l'espérons vous seront utiles lorsque vous aborderez par vous mêmes la programmation d'une telle application. la programmation d'une telle application.

notes

résumé