Support de cours

Cours:

Initiation à la programmation (en C++)

Vidéo:

Puissance 4 - moteur de jeu (partie 2)

Concepts (extraits des sous-titres générés automatiquement) :

Coup valide. Corps de la fonction. Numéro de colonne. Dernier commentaire. Boucle do while. Dernier retour. Version précédente. Affichage de la question. On ré-affiche. Taille de la grille. Problème de cette version. Return false. Quizz des vidéos précédentes. Séquence vidéo. Boucle principale.

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

vers la vidéo

EPFL

Puissance 4 : moteur de jeu

(Partie 2)

Initiation à la programmation (C++)

Vincent Lepetit, Jean-Cédric Chappelier et Jamila Sam

...

notes

résumé

0m 0s

Fonction demande_et_joue

```
void demande_et_joue(Grille& grille, Couleur couleur_joueur)
{
  bool valide;

  cout << "Joueur ";
  if (couleur_joueur == jaune) {
    cout << 'X';
  } else {
    cout << 'O';
  }
  cout << " :  entrez un numéro de colonne" << endl;

  do {
    size_t colonne;
    cin >> colonne;
    --colonne;  // les indices des tableaux commencent par 0 en C++

    valide = joue(grille, colonne, couleur_joueur);
    if (not valide) {
      cout << " > Ce coup n'est pas valide" << endl;
    }
  } while(not valide);
}
```

Ce code, cependant ne compile pas parce que nous avons ici une utilisation de « valide » et la portée de « valide », qui a été déclaré ici est ce bloc là. Donc ici on a une utilisation de « valide » qui est au-delà de sa portée, ce qui fait que à ce stade là le compilateur va nous donner une erreur. Donc ce qu'il faut faire c'est sortir la déclaration de « valide » hors de la boucle. Donc, par exemple, ici je la mets tout en haut ou on pourrait la mettre juste avant la boucle. Donc ici, on déclare « valide » et on pourrait aussi l'initialiser par exemple, à 'false'. Il faut aussi bien sûr ici ne pas garder cette déclaration ici, parce que sinon on aurait deux variables « valid », on aurait une, celle ci, qui serait de portée locale ici au bloc contrôlé par le while. Et on aurait notre autre « valide » ici de portée le corps de la fonction, ce qui fait qu'évidemment, ça ne fonctionnerait pas donc évidemment ici on change cette déclaration initialisation en une affectation. Voila donc pour notre fonction « demande_et_joue » un dernier commentaire cependant, on aurait bien sûr évidemment pu mettre l'affichage de la question dans la boucle do while, ici c'est une variante, c'est-à dire-qu'à chaque fois qu'on a un coup valide, on ré-affiche à quel joueur c'est d'entrer le coup ; alors que dans la version précédente on fait simplement que répéter la saisie de sa réponse.

notes

résumé

0m 1s

Retour sur la fonction joue

```
bool joue(Grille& grille, size_t colonne, Couleur couleur)
{
  |
  // on parcourt la colonne en partant du bas...
  size_t ligne(grille.size() - 1);
  // ...jusqu'à trouver une case vide...
  // ...ou jusqu'en haut de la colonne si la colonne est pleine :
  bool pleine(false);
  while ((not pleine) and (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 (not pleine) {
    grille[ligne][colonne] = couleur;
    return true;
  } else {
    return false;
  }
}
```

Terminons enfin par un dernier retour sur la fonction « joue », lié à un des quizz des vidéos précédentes.

notes

résumé

1m 37s

Fonction demande_et_joue

```
void demande_et_joue(Grille& grille, Couleur couleur_joueur)
{
  bool valide;

  cout << "Joueur ";
  if (couleur_joueur == jaune) {
    cout << 'X';
  } else {
    cout << 'O';
  }
  cout << " :  entrez un numéro de colonne" << endl;

  do {
    size_t colonne;
    cin >> colonne;
    --colonne;  // les indices des tableaux commencent par 0 en C++

    valide = joue(grille, colonne, couleur_joueur);
    if (not valide) {
      cout << " > Ce coup n'est pas valide" << endl;
    }
  } while(not valide);
}
```

Le problème de cette version de « joue » c'est que le numéro de colonne n'a jamais été vérifié avant d'accéder ici dans le tableau. Il n'est pas vérifié ni dans la fonction « joue » ici, ni dans la fonction « demande_et_joue »,

notes

résumé

1m 45s

Retour sur la fonction joue

```
bool joue(Grille& grille, size_t colonne, Couleur couleur)
{
  // si le numéro de colonne n'est pas valide, le coup n'est pas valide :
  if |
  // on parcourt la colonne en partant du bas...
  size_t ligne(grille.size() - 1);
  // ...jusqu'à trouver une case vide...
  // ...ou jusqu'en haut de la colonne si la colonne est pleine :
  bool pleine(false);
  while ((not pleine) and (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 (not pleine) {
    grille[ligne][colonne] = couleur;
    return true;
  } else {
    return false;
  }
}
```

ici on fait simplement que faire confiance à l'utilisateur et on décrémente de 1 à 7 la colonne mais on ne vérifie pas si il nous a donné, par exemple 100 000 comme numéro de colonne, et donc, il faut maintenant corriger notre fonction « joue » pour garantir que l'accès à la colonne ici est effectivement valide. Donc on commence par, comme d'habitude, mettre le commentaire et puis ensuite, on l'écrit,

notes

résumé

2m 1s

Où en sommes nous ?

A ce stade, nous avons fait :

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

donc si le paramètre « colonne » est plus grand que la taille de la grille, alors, le coup n'est pas valide et donc on « return false » tout simplement mais il faut penser à systématiquement vérifier les conditions d'accès à un tableau. Ceci termine donc cette séquence vidéo où nous avons ajouté 3 fonctionnalités à notre programme,

notes

résumé

2m 25s

nous avons demandé au joueur d'entrer un coup. Nous avons codé la boucle principale qui permet de répéter le jeu et nous avons alterné les 2 joueurs. Reste, pour terminer complètement le jeu à terminer la boucle, donc la condition d’arrêt qu'on a laissée en blanc dans la boucle soit que le jeu donc est plein, soit que l'un des deux joueurs gagne. Ceci est l'objet de la prochaine séquence vidéo. séquence vidéo.

notes

résumé

2m 49s