Support de cours

Cours:

Initiation à la programmation (en C++)

Vidéo:

Puissance 4 - fonction joue 1ère version (partie 5)

Concepts (extraits des sous-titres générés automatiquement) :

Variable pleine. Case vide. Type de retour de notre fonction. Valeur de retour. Valeur pleine. Condition précédente. Not pleine. Premiers coups. Valeur de vérité. Toute nouvelle fonction. Premiers coups valides. Septième coup. Variantes d'écriture de cette fonction. Fin du programme. Ligne.

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

vers la vidéo

EPFL

Puissance 4 : fonction `joue` 1ère version
(Partie 5)

Initiation à la programmation (C++)

Vincent Lepetit, Jean-Cédric Chappelier et Jamila Sam

...

notes

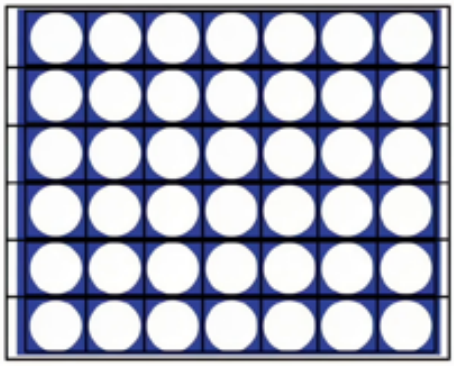
résumé

0m 0s

Fonction joue

```
void 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 :
  bool pleine(false);
  while (grille[ligne][colonne] != vide) {
    if (ligne == 0) {
      pleine = true
    --ligne;
  }

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



et imaginons qu'un joueur décide de tout le temps jouer dans la même colonne ; les deux joueurs décident de tout le temps jouer dans la même colonne. Qu'est-ce qui va se passer ? Évidemment on va commencer, ici pour simplifier je vais toujours prendre la même couleur, on va commencer par mettre un pion ici et puis on aura un pion ici puis on aura un pion là, etc. etc. jusqu'à ce qu'on mette un pion ici ; maintenant supposons qu'on redemande donc de rentrer ici dans la colonne 3, de nouveau un pion. Qu'est-ce qui se passe ? Donc on part depuis la ligne qui est en bas et puis jusqu'à ce qu'on trouve une case vide, on va faire « ligne-- » donc ici on va remonter notre ligne jusqu'ici, et puis là on arrive avec ligne qui vaut 0 mais on n'a toujours pas trouvé de case vide et donc qu'est-ce qu'on fait ? On fait ici « ligne-- » , et la question c'est qu'est-ce qui se passe ici ? alors pour l'instant peu importe ce qui se passe ici, parce que c'est une erreur de conception, une erreur d'algorithme ! Donc ce qu'il faut que l'on fasse, c'est que l'on corrige notre algorithme et que l'on recherche le fait que la case vide est bien dans le jeu et qu'on n'est pas sorti. Pour ça, on va introduire un booléen ceci va nous indiquer vrai ou faux si la colonne est pleine. Donc au départ bien sûr la colonne n'est pas pleine et donc on l'initialise à « faux ». Et donc lorsque l'on va faire ici notre parcours ce que l'on va faire, c'est que l'on va vérifier que l'on n'a pas débordé, donc pour ça si jamais on atteint la ligne ici 0, alors c'est qu'on a débordé et donc que le tableau est plein. Donc

notes

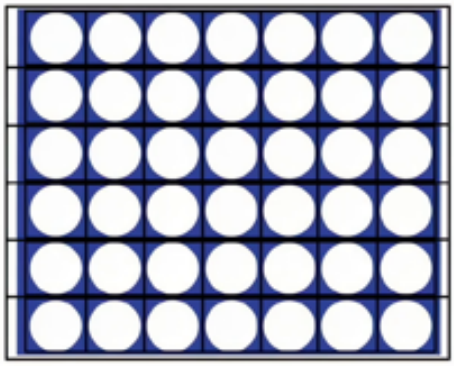
résumé

0m 1s

Fonction joue

```
void 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 :
  bool pleine(false);
  while (grille[ligne][colonne] != vide) {
    if (ligne == 0) {
      pleine = true
    --ligne;
  }

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



on dit : si la ligne est « = = 0 » alors on va dire que cette fois-ci la variable pleine, que l'on avait donc déclarée ici,

notes

résumé

est « vraie » puisqu'on a atteint la ligne 0 donc on sait que la colonne est pleine puisque la ligne 0 d'une colonne n'est pas vide que si cette colonne est pleine. Donc sinon, à ce moment-là on continue comme on faisait précédemment par diminuer donc la ligne de 1 à chaque fois. Corrigeons donc maintenant notre boucle pour qu'elle s'arrête dès que la colonne est pleine donc ici on faisait une boucle « while » c'est à dire qu'on peut « continuer tant que » donc ici on doit continuer tant que la grille, la colonne n'est pas pleine donc ici on rajoute « n'est pas pleine » donc que la colonne n'est pas pleine et que l'on a la condition précédente et que l'on n'a pas trouvé de case vide ; ça s'écrit exactement comme ce que l'on avait précédemment on ne change rien on rajoute juste le fait que la condition c'est que la colonne ne soit pas pleine. Donc on explique ici avec un commentaire que l'on a trouvé une case vide, éventuellement, ou alors que on a atteint le haut de la colonne pour s'arrêter. Enfin donc on va corriger la fin du programme si à ce moment-là la colonne n'était pas pleine, alors on fait comme précédemment, on remplit.

notes

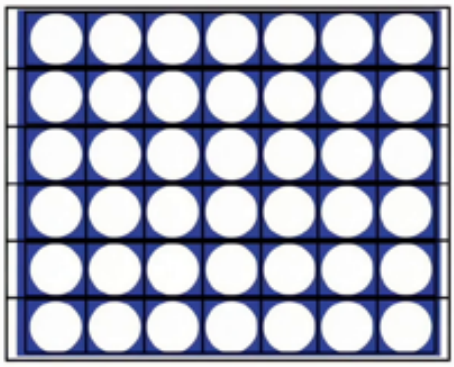
résumé

2m 1s

Fonction joue

```
void 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;
    }
  }

  // on remplit la case vide trouvée :
  if (not pleine) {
    grille[ligne][colonne] = couleur;
  }
}
```



On peut aussi à ce stade en profiter pour indiquer en valeur de retour si on a pu jouer ou si on n'a pas pu jouer.

notes

résumé

3m 13s

Donc dans ce cas là effectivement on a pu mettre la couleur désirée dans la colonne en question, et donc on pourrait par exemple dire que l'on retourne la valeur de vérité, « vrai », la valeur « true », pour dire que l'on a pu jouer et sinon c'est-à-dire que si la colonne était pleine, à ce moment-là, on retourne la valeur « false » ; on en profite pour changer le type de retour de notre fonction, qui maintenant retourne quelque chose et donc retourne ici un booléen, et on indique par un commentaire que si la colonne n'était pas pleine on a joué et on renvoie « true » et que sinon, à ce moment-là on n'a pas pu jouer et on renvoie « false ». Comme on a modifié notre fonction « joue » ici pour corriger une erreur que l'on avait envisagée, il est important de tester cette correction,

notes

résumé

3m 21s

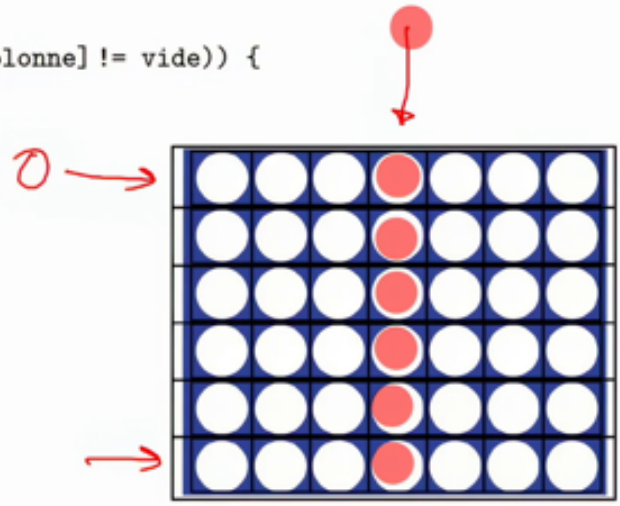
Tester la nouvelle fonction joue

```
bool joue(Grille& grille, size_t colonne, Couleur couleur)
{
  size_t ligne(grille.size() - 1);

  bool pleine(false);
  while ((not pleine) and (grille[ligne][colonne] != vide)) {
    if (ligne == 0) {
      pleine = true;
    } else {
      --ligne;
    }
  }

  if (not pleine) {
    grille[ligne][colonne] = couleur;
    return true;
  } else {
    return false;
  }
}
...
  for(int i(0); i < 10; ++i) {
    bool valide(joue(grille, 3, rouge));

    if (not valide) {
      cout << "impossible d'ajouter un pion sur cette colonne" << endl;
    }
  }
```



c'est toujours important de tester votre programme régulièrement donc maintenant on réécrit un autre « main » ici, qui nous permet donc de tester toujours la même chose, on initialise une grille que l'on a déclarée et puis on l'affiche, mais ici on va essayer de remplir disons par exemple, dix fois la même colonne. Pour cela on va donc faire appel à notre toute nouvelle fonction « joue » mais en jouant tout le temps ici donc dans la même colonne. Cette fonction « joue » retournant maintenant une valeur, ayant une valeur de retour de type booléen, on déclare ici une variable valide de type booléen pour récupérer la valeur de retour. Et dans notre boucle on va à chaque fois tester si le coup joué n'est pas valide ; alors à ce moment-là, on affiche qu'il est impossible de jouer dans la colonne et puis dans la boucle aussi à chaque fois, on affichera la grille. Donc ce qui va se passer dans le « main » c'est qu'on va vouloir jouer dix fois dans la colonne 3. Donc les six premiers coups vont se dérouler sans aucun souci, donc on va jouer ici les six premiers coups mais dès qu'on va vouloir jouer le septième coup, regardons ce qui va se passer : à ce moment-là on va donc appeler ici « joue grille 3 » , on va recevoir 3 et puis la couleur rouge ici, on va initialiser donc « ligne » à la taille de la grille - 1 puis on va initialiser la valeur pleine à « false » puisqu'au début avant de tester on suppose que la colonne n'est pas pleine, puisque c'est justement ce que l'on veut vérifier, et donc évidemment ici le test est vrai ; et donc on va faire cette partie du test, et donc ici

notes

résumé

4m 13s

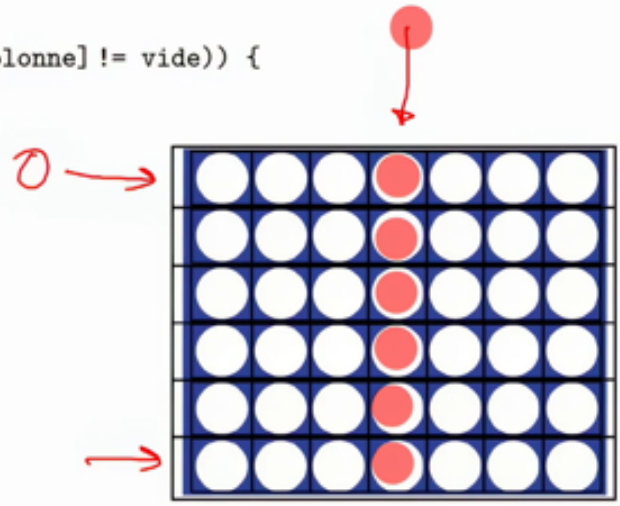
Tester la nouvelle fonction joue

```
bool joue(Grille& grille, size_t colonne, Couleur couleur)
{
  size_t ligne(grille.size() - 1);

  bool pleine(false);
  while ((not pleine) and (grille[ligne][colonne] != vide)) {
    if (ligne == 0) {
      pleine = true;
    } else {
      --ligne;
    }
  }

  if (not pleine) {
    grille[ligne][colonne] = couleur;
    return true;
  } else {
    return false;
  }
}
...
  for(int i(0); i < 10; ++i) {
    bool valide(joue(grille, 3, rouge));

    if (not valide) {
      cout << "impossible d'ajouter un pion sur cette colonne" << endl;
    }
  }
```



« grille[taille - 1][colonne 3] » n'est pas vide, donc ce test est correct. On est pas dans le cas où la ligne vaut 0 donc on décrémente la ligne, cette condition ici sera toujours vraie tant que la case testée n'est pas vide, donc on va faire ici « -- », « -- », « -- ». On va finalement donc arriver ici en 0, et à ce moment-là on va arriver à cette phase-là du test « pleine » est toujours « false » et la case n'est pas vide, on re-rentre donc dans la boucle ici, mais cette fois-ci « ligne » vaut 0 et donc comme ligne vaut 0 alors on met « pleine » à « true » et on n'exécute pas le « -- » donc on reste ici sur la ligne 0, ce qui fait que l'on reboucle encore une fois dans la boucle « while » et fois-ci la condition « not pleine » devient « false » et donc la boucle « while » sort ici et se termine. Donc on arrive dans ce « if » et cette fois-ci la condition « not pleine » est toujours fausse, « if » lui-même va rentrer dans le bloc « else » on voit donc que la grille n'a pas été modifiée et simplement on retourne « false » ici, ce qui fait donc que les six premiers coups valides étaient corrects et la boucle dans le « main » s'est déroulée,

notes

résumé

(la boucle « for » ici s'est déroulée) six fois mais, arrivé au septième coup, on a ici un « valide » qui devient « false » à cause de ce return « false » et donc ici le coup n'est pas valide et on affiche « impossible d'ajouter un pion ». dans cette colonne Comme dans notre « main » ici de test on n'a pas mis de conditions ici sur la validité du jeu, la boucle « for » va continuer à s'exécuter donc 10 fois, et on aura donc quatre fois l'affichage de ce message d'impossibilité de jouer. Voilà donc pour cette fonction « joue » qui nous permet, étant donné un numéro de colonne, une couleur et une grille, effectivement de jouer à l'endroit si la position est valide et d'indiquer si le coup n'est pas valide. Il existe différentes variantes d'écriture de cette fonction qui vont être présentées donc dans une séquence vidéo suivante. dans une séquence vidéo suivante.

notes

résumé

7m 25s