Support de cours

Cours:

Initiation à la programmation (en C++)

Vidéo:

Puissance 4 - fonctions est_ce_gagne et compte

Concepts (extraits des sous-titres générés automatiquement) :

Pion de la couleur du joueur. Nouvelle fonction. Grille ligne. Résultat de ma fonction. Boucle principale. Condition d'arrêt. Valeur de retour de ma fonction. Couleur du joueur. Variable de type booleéen. Façon suivante. Pions rouges. Dernier argument. Colonne de la case de départ. Nombre de pions. Variable de type booléen.

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

vers la vidéo

EPFL

Puissance 4 : fonctions `est_ce_gagne` et `compte`

Initiation à la programmation (C++)

Vincent Lepetit, Jean-Cédric Chappelier et Jamila Sam

...

notes

résumé

0m 0s

Retour sur la fonction `main`

```
Couleur couleur_joueur(jaune);

do {
  demande_et_joue(grille, couleur_joueur);

  affiche(grille);


  // on change la couleur pour la couleur de l'autre joueur:
  if (couleur_joueur == jaune) {
    couleur_joueur = rouge;
  } else {
    couleur_joueur = jaune;
  }
} while(    );
```

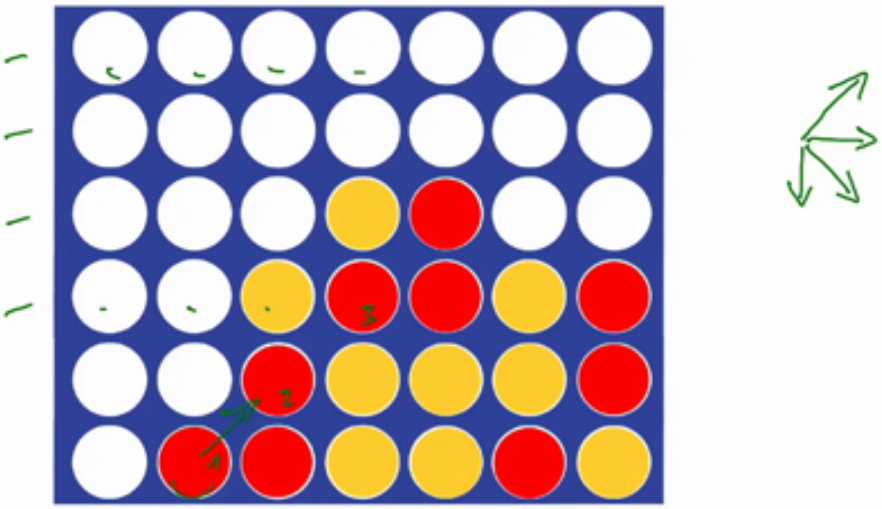
Dans la vidéo précédente, vous aviez vu comment écrire la boucle principale qui permet aux joueurs de jouer chacun leur tour. Il manquait juste

notes

résumé

0m 1s

Fonction est_ce_gagne : Stratégie



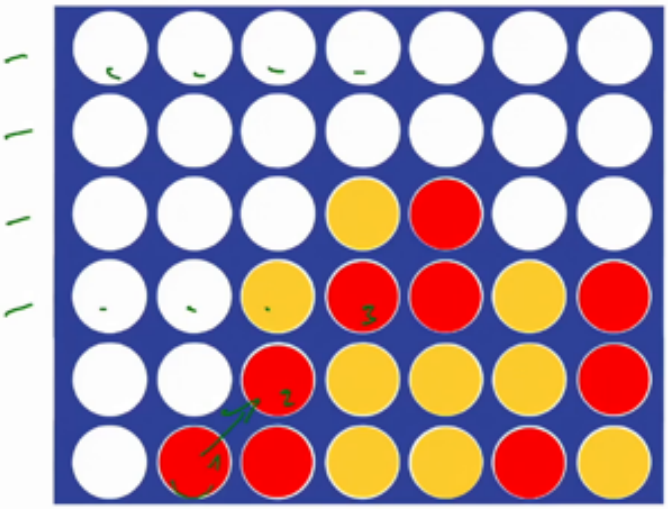
la condition d'arrêt, c'est-à-dire qu'il manquait ce qu'il faut écrire ici pour que la boucle s'arrête quand un des deux jours a gagné ou quand la grille est pleine. Alors pour l'instant nous allons nous concentrer sur le cas où la boucle doit s'arrêter quand un des deux joueurs a gagné. Et pour cela je vais introduire simplement une nouvelle fonction que je vais appeler « est_ce_gagne » et que je vais appeler de cette façon-ci : je vais mettre le résultat de ma fonction « est_ce_gagne » dans une variable que je vais appeler « gagne » ; ce sera une variable de type booleéen. « gagne » va recevoir la valeur de retour de ma fonction « est_ce_gagne ». Ma fonction « est_ce_gagne » va prendre en argument la grille et la couleur du joueur qui vient juste de jouer. Alors il faut que je déclare également ma variable « gagne » ; et comme je dois m'en servir dans la condition d'arrêt, il faut que je déclare « gagne » en dehors de la boucle, c'est-à-dire quelque part ici. Et maintenant je peux utiliser la variable « gagne » dans ma condition d'arrêt. Et ça va s'écrire très simplement de la façon suivante : on va répéter la boucle tant que le joueur n'a pas gagné. Il nous faut donc écrire cette fonction « est_ce_gagne », ce qui est sans doute la partie la plus difficile de notre programme. Alors comment allons-nous nous y prendre ? Ayant en fait plusieurs façons d'écrire la fonction « est_ce_gagne », la stratégie que nous vous proposons est la suivante : nous allons parcourir la grille ligne par ligne et colonne par colonne jusqu'à trouver un pion de la couleur du joueur qui vient de jouer. Disons par exemple que c'est le joueur rouge qui vient de jouer et que l'on trouve ce pion-ci ; nous

notes

résumé

0m 11s

Fonction est_ce_gagne : Stratégie

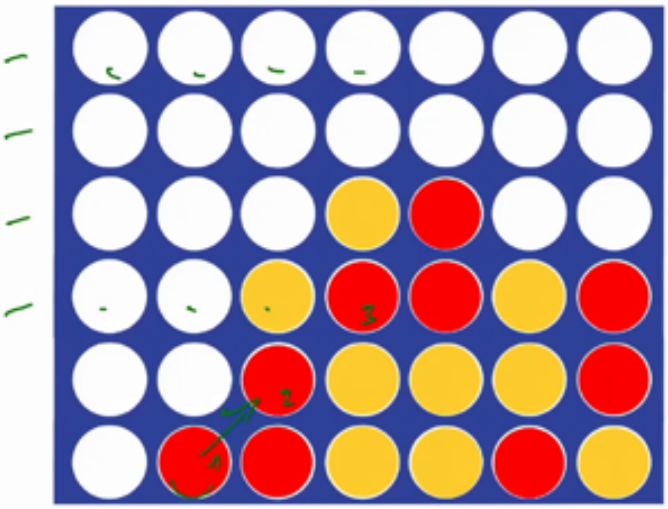


allons partir de ce pion et parcourir la grille dans plusieurs directions horizontalement, verticalement et en diagonale pour compter combien il y a de pions, à partir de ce pion-ci, qui sont de la même couleur. Si par exemple on considère cette direction-ci, on arrive à compter 1, 2, 3, 4 pions rouges, et donc on sait que le joueur a gagné. Si en revanche on a parcouru toute la grille et qu'on a toujours compté moins que 4 pions, on sait que le joueur n'a pas gagné. On peut en fait remarquer qu'on n'est pas obligé de considérer toutes les directions et en fait nous allons nous contenter de compter les pions dans quatre directions seulement. Et nous allons choisir par exemple vers le haut et la droite, vers la droite, vers le bas et la droite, et vers le bas. Alors pourquoi on peut se contenter de considérer ces quatre directions ? J'ai dit que nous allions parcourir la grille ligne par ligne et colonne par colonne. Supposons encore une fois que c'est le joueur rouge qui vient de jouer. Le premier pion que nous allons trouver est celui-ci. Nous allons compter tout d'abord les pions, dans cette direction-ci. Nous n'allons compter qu'un seul pion, celui-ci. Même chose pour cette direction-ci et cette direction-ci : il n'y a qu'un seul pion dans ces trois directions. Dans cette direction-ci, il y a deux pions, mais ce n'est pas suffisant pour gagner ; et nous allons continuer à parcourir la grille. Le prochain pion rouge que nous allons trouver, est celui-ci, dans cette direction-ci et cette direction-ci, nous allons compter que deux pions ; ce n'est pas suffisant pour gagner ; dans cette direction-ci et cette direction-ci, nous allons compter qu'un seul pion, ce n'est toujours pas suffisant pour gagner. Donc nous allons continuer à parcourir la grille jusqu'à atteindre

notes

résumé

Fonction `est_ce_gagne` : Stratégie



ce pion-ci. Nous allons compter les pions dans cette direction-ci et nous allons compter cette fois 1, 2, 3, 4 pions

notes

résumé

Fonction est_ce_gagne

```
// gagne = est_ce_gagne
```

et on sait que le joueur rouge a gagné. Pour prouver que le joueur rouge a bien aligné ses quatre pions, on peut donc soit partir de ce pion-ci et aller dans cette direction-ci soit partir de ce pion là et aller dans cette direction-là. On est donc pas obligé de considérer ces deux directions. On peut se contenter d'une seule. Et c'est la même chose pour les six autres directions. On va regarder uniquement dans cette direction-ci et pas dans cette direction-ci dans cette direction-ci et pas celle-ci dans cette direction-ci et pas celle-ci. Il nous reste donc que quatre directions à considérer. Il ne nous reste plus qu'à coder cette stratégie en écrivant la fonction « est_gagne ». Alors je rappelle qu'on va appeler la fonction « est_ce_gagne » de la façon suivante ; c'est-à-dire qu'on va mettre la valeur de retour de la fonction dans une variable de type booléen

notes

résumé

4m 1s

Fonction est_ce_gagne

```
// gagne = est_ce_gagne(grille
```

et que la fonction atteint la grille en argument

notes

résumé

4m 59s

Fonction est_ce_gagne

```
// gagne = est_ce_gagne(grille, couleur_joueur);
bool est_ce_gagne(const Grille& grille, Couleur couleur_joueur)
{
  for(size_t ligne(0); ligne < grille
```

et la couleur du joueur qui vient jouer ; l'en tête de cette fonction va tout simplement être celui-ci ; le résultat sera de type booléen ; et on va passer la grille par référence constante, puisque la grille n'a pas besoin d'être modifiée dans cette fonction « est_ce_gagne ». Et le deuxième paramètre sera de type « Couleur ». Notre stratégie consiste à parcourir la grille ; ce qu'on va faire

notes

résumé

5m 2s

Fonction est_ce_gagne

```
// gagne = est_ce_gagne(grille, couleur_joueur);
bool est_ce_gagne(const Grille& grille, Couleur couleur_joueur)
{
  for(size_t ligne(0); ligne < grille.size(); ++ligne) {
    for(size_t colonne(0); colonne < grille[ligne].size(); ++colonne) {
      Couleur couleur_case(grille[lig
```

à l'aide de deux « boucles for ». Et comme on a besoin des coordonnées du pion, on va utiliser deux « boucles for classiques » qui vont s'écrire de la façon suivante. Les variables ligne et colonne vont contenir les coordonnées du pion à partir duquel on va compter le nombre de pions alignés. Et pour simplifier un peu l'écriture de code, je vais introduire une variable locale, que je vais appeler « couleur_case »,

notes

résumé

5m 33s

pour contenir la couleur de ce pion. Si la couleur de la case est la même que la couleur du joueur qui vient de jouer,

notes

résumé

6m 1s

Fonction est_ce_gagne

```
// gagne = est_ce_gagne(grille, couleur_joueur);
bool est_ce_gagne(const Grille& grille, Couleur couleur_joueur)
{
  for(size_t ligne(0); ligne < grille.size(); ++ligne) {
    for(size_t colonne(0); colonne < grille[ligne].size(); ++colonne) {
      Couleur couleur_case(grille[ligne][colonne]);

      if (couleur_case == couleur_joueur) {
```

alors je vais compter les pions alignés à partir de cette case dans les quatre directions qui nous intéressent.

notes

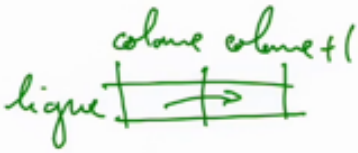
résumé

6m 11s

Fonction est_ce_gagne

```
// gagne = est_ce_gagne(grille, couleur_joueur);
bool est_ce_gagne(const Grille& grille, Couleur couleur_joueur)
{
  for(size_t ligne(0); ligne < grille.size(); ++ligne) {
    for(size_t colonne(0); colonne < grille[ligne].size(); ++colonne) {
      Couleur couleur_case(grille[ligne][colonne]);

      if (couleur_case == couleur_joueur) {
        if (// en diagonale, vers le haut et la droite:
            compte(grille, ligne, colonne, -1, +1) >= 4 or
            // horizontalement, vers la droite:
            compte(grille, ligne, colonne,  0, +1) >= 4 or
```



Comme c'est un peu compliqué de compter les pions alignés, je vais introduire une fonction que je vais appeler « compte ». De quels arguments la fonction « compte » a-t-elle besoin ? Tout d'abord, bien sûr de la grille, des coordonnées de la case à partir de laquelle on va compter les pions alignés. Donc c'est tout simplement les variables « ligne » et « colonne »; et puis également de la direction dans laquelle on veut compter les pions. Alors comment peut-on définir cette direction ? Supposons par exemple que je vais compter les pions en diagonale vers le haut et vers la droite. Alors dans ce cas pour passer d'une case à l'autre il va falloir soustraire 1 à la ligne et ajouter 1 à la colonne. Et donc je vais utiliser ces deux valeurs-là, ce -1 et ce +1, en arguments à ma fonction « compte » pour dire que je vais aller vers le haut et vers la droite. Je vais faire renvoyer à ma fonction « compte » le nombre de pions alignés et de la même couleur ; et je sais que si ce nombre de pions alignés est supérieur ou égal à 4, alors le joueur a gagné. Alors pourquoi supérieur ou égal à 4 plutôt que simplement égal à 4 ? C'est parce que bien sûr on peut aligner plus que quatre pions. Alors évidemment il faut que je considère aussi les trois autres directions et ça va s'écrire de la façon suivante pour compter les pions horizontalement Eh bien je ne vais pas changer la ligne ; c'est pour ça que j'utilise 0 en argument ici. Par contre, la colonne va changer ; je vais passer à la colonne suivante. C'est pour ça que j'utilise la valeur +1 en dernier argument.

notes

résumé

6m 18s

Fonction compte

Ensuite, pour compter les pions en diagonale vers le bas et la droite, eh bien, pour passer d'une case à l'autre je vais ajouter 1 à la ligne ; c'est pour ça que j' utilise la valeur +1 ici. Et puis j'ajoute également 1 à la colonne ; et c'est pour ça que j'utilise la valeur +1 ici. Enfin, il nous reste une dernière direction pour compter les pions verticalement vers le bas je vais passer d'une case à l'autre en ajoutant 1 à la ligne c'est pour ça que j'ai la valeur +1 ici. Par contre, la colonne ne va pas changer et j'utilise la valeur 0 ici. Si l'une de ces quatre conditions est vraie, alors je sais que le joueur a gagné. Alors il suffit qu'une de ces conditions soit vraie, et c'est pour ça que j'ai utilisé « or » ici. Et dans ce cas je vais faire « envoyer true » à la fonction. Je vais fermer les accolades que j'avais ouvertes précédemment. Et si j'arrive à ce point-ci de ma fonction, ça veut dire que j'ai parcouru toute la grille sans jamais passer par ici, puisque sinon je serais déjà sorti de ma fonction. Et je sais que dans ce cas-là, eh bien, je n'ai pas trouvé au moins quatre pions alignés. Et je sais que le joueur n'a pas encore gagné. Donc je fais « renvoyer false » à ma fonction.

notes

résumé

8m 1s

Fonction compte

```
// if (compte(grille, ligne, colonne, |
```

Mais évidemment il faut encore écrire la fonction « compte ». Cette fonction « compte » on va l'appeler par exemple ainsi ; c'est-à-dire qu'elle va renvoyer le nombre de pions alignés

notes

résumé

9m 30s

Fonction compte

```
// if (compte(grille, ligne, colonne, -1, +1) >= 4 ...
unsigned int compte(const Grille& grille,
                    size_t ligne_depart, size_t colon
```

qui est forcément un nombre positif ou nul. Donc, c'est pour ça que je vais utiliser le type « unsigned int » pour définir le type de résultat de ma fonction. Le premier paramètre de ma fonction est donc la grille que je vais passer encore une fois par référence constante. Les deux paramètres suivants sont simplement la ligne et la colonne

notes

résumé

9m 41s

Fonction compte

```
// if (compte(grille, ligne, colonne, -1, +1) >= 4 ...
unsigned int compte(const Grille& grille,
                    size_t ligne_depart, size_t colonne_depart,
                    int dir_ligne, int dir_colonne)
{
  unsigned int compteur(0);

  size_t ligne(ligne_depart);
  size_t colonne(colonne_depart);

  while (grille[ligne][colonne] == grille[l
```

de la case de départ à partir de laquelle je vais compter les pions et les deux derniers paramètres de ma fonction sont les deux valeurs qui vont définir la direction et qui peuvent être des entiers négatifs ou positifs. Donc cette fois-ci je vais utiliser le type « int » Et je vais appeler ces paramètres « dir_ligne » et « dir_colonne » pour signaler que c'est deux valeurs qui vont correspondre aux lignes et à la colonne pour une direction donnée. Je vais me servir d'une variable compteur initialisée à 0 et du même type que la valeur de résultats de la fonction « compte » pour compter le nombre de pions alignés. Que doit faire la fonction « compte » exactement ? Elle doit partir de la case de coordonnées « ligne_depart » et « colonne_depart ». Et elle doit parcourir la grille dans la direction donnée tant que l'on trouve des pions de la même couleur que le pion qui est en « ligne_depart » « colonne_depart ». Alors comment on va faire ça ? On va utiliser les variables que je vais appeler « ligne » et « colonne », qui seront respectivement initialisées à « ligne_depart » et « colonne_depart ». Et à chaque tour de boucle pour passer d'une case à l'autre, je vais ajouter « dir_ligne » à la variable ligne et « dir_colonne » à la variable colonne. Comme on va parcourir la grille tant qu'on trouve des pions de la même couleur que le pion qui est en « ligne_depart » « colonne_depart », eh bien on va utiliser une « boucle while » ! Et à chaque tour de la « boucle while » on aura trouvé un nouveau pion de la même couleur et on va incrémenter la variable « compteur ». Comment tout cela va-t-il s'écrire ? Eh bien

notes

résumé

10m 1s

Fonction compte

```
// if (compte(grille, ligne, colonne, -1, +1) >= 4 ...
unsigned int compte(const Grille& grille,
                    size_t ligne_depart, size_t colonne_depart,
                    int dir_ligne, int dir_colonne)
{
  unsigned int compteur(0);

  size_t ligne(ligne_depart);
  size_t colonne(colonne_depart);

  while (grille[ligne][colonne] == grille[l
```

on va commencer par déclarer les variables ligne et colonne, initialisées à « ligne_depart » et « colonne_depart ». On va donc utiliser une « boucle while » dont la condition d'arrêt dépend de la couleur du pion en ligne et colonne. Et on va continuer tant que la couleur de ce pion est la même

notes

résumé

que la couleur du pion qui est en « ligne_depart » « colonne_depart ». A chaque tour de boucle, on va donc incrémenter la variable « compteur » puisqu'on a trouvé un nouveau pion de la même couleur que le premier. Et puis on va ajouter « dir_ligne » et « dir_colonne » à ligne et colonne. Notez au passage que j'ai ajouté des espaces ici et ici pour aligner les signes = et + sur ces deux lignes, ça rend le code plus clair ça met en évidence que les variables ligne et colonne sont de même nature.

notes

résumé

12m 1s

Fonction compte

```
// if (compte(grille, ligne, colonne, -1, +1) >= 4 ...
unsigned int compte(const Grille& grille,
                    size_t ligne_depart, size_t colonne_depart,
                    int dir_ligne, int dir_colonne)
{
  unsigned int compteur(0);

  size_t ligne(ligne_depart);
  size_t colonne(colonne_depart);

  while (ligrille[ligne][colonne] == grille[ligne_depart][colonne_depart]) {

    ++compteur;
    ligne   = ligne   + dir_ligne;
    colonne = colonne + dir_colonne;

  }

  return compteur;
}
```

Quand on sort de la boucle, c'est qu'on a trouvé un pion qui n'était pas de la bonne couleur et on peut renvoyer la valeur contenue dans la variable compteur. Sauf que ce n'est pas tout à fait correct et j'espère que vous voyez pourquoi à ce stade, comme pour la fonction « joue » si on ajoute pas de conditions à notre « boucle while », il se peut très bien qu'on sorte du tableau « grille ». Comme pour la dernière version alternative de la fonction « joue »

notes

résumé

12m 35s

Retour sur la fonction est_ce_gagne

```
bool est_ce_gagne(const Grille& grille, Couleur couleur_joueur)
{
  for(size_t ligne(0); ligne < grille.size(); ++ligne) {
    for(size_t colonne(0); colonne < grille[ligne].size(); ++colonne) {
      Couleur couleur_case(grille[ligne][colonne]);

      if (couleur_case == couleur_joueur) {
        if (// en diagonale, vers le haut et la droite:
            compte(grille, ligne, colonne, -1, +1) >= 4 or
            // horizontalement, vers la droite:
            compte(grille, ligne, colonne,  0, +1) >= 4 or
            // en diagonale, vers le bas et la droite:
            compte(grille, ligne, colonne, +1, +1) >= 4 or
            // verticalement, vers le bas:
            compte(grille, ligne, colonne, +1,  0) >= 4) {
          return true;
        }
      }
    }
  }

  return false;
}
```

je vais utiliser la même solution. Et je vais rajouter un test sur la variable ligne pour vérifier qu'elle est bien inférieure à la taille de « grille ». Et la même chose pour la variable colonne.

notes

résumé

13m 1s

Pour finir, je vais revenir sur la fonction « est_ce_gagne » pour faire remarquer que si la ligne de la case de départ a pour indice 0, 1, ou 2, eh bien ce n'est pas la peine de chercher dans cette direction-ci ! Par exemple, puisqu'on sait déjà que le joueur n'aura pas pu aligner quatre pions dans cette direction-là on peut aligner au maximum deux / trois pions si on part de la ligne numéro 2. Même chose quand on part du bord droit de la grille et qu'on compte le nombre de pions vers la droite ou quand on part du bas de la grille et qu'on compte le nombre de pions vers le bas. Donc la fonction « est_ce_gagne » fait plus d'opérations que nécessaire et je vais ajouter des conditions pour éviter de compter les pions quand on est sûr qu'il n'y aura pas au moins quatre pions alignés.

notes

résumé

13m 14s

Retour sur la fonction est_ce_gagne

```
bool est_ce_gagne(const Grille& grille, Couleur couleur_joueur)
{
  for(size_t ligne(0); ligne < grille.size(); ++ligne) {
    for(size_t colonne(0); colonne < grille[ligne].size(); ++colonne) {
      Couleur couleur_case(grille[ligne][colonne]);

      if (couleur_case == couleur_joueur) {
        if (// en diagonale, vers le haut et la droite:
            compte(grille, ligne, colonne, -1, +1) >= 4 or
            // horizontalement, vers la droite:
            compte(grille, ligne, colonne,  0, +1) >= 4 or
            // en diagonale, vers le bas et la droite:
            compte(grille, ligne, colonne, +1, +1) >= 4 or
            // verticalement, vers le bas:
            compte(grille, ligne, colonne, +1,  0) >= 4) {
              return true;
        }
      }
    }
  }

  return false;
}
```

Alors, ça va s'écrire de la façon suivante :

notes

résumé

14m 1s

donc on considère la direction en diagonale vers le haut et la droite et pour qu'il y ait une chance d'avoir au moins quatre pions alignés, il faut donc que la ligne soit supérieure ou égale à 3 et que la colonne ne soit pas trop grande et vous pourrez vérifier qu'en fait la colonne doit être inférieure au nombre de colonnes moins 4, ça va s'écrire de la façon suivante. Dans le cas où on cherche les pions alignés horizontalement il suffit d'avoir un test sur la colonne, il sera le même que précédemment. Le test pour la direction en diagonale vers le bas à la droite va s'écrire de cette façon-ci. Cette fois-ci je vérifie que la ligne n'est pas trop grande et que la colonne non plus. Dans le cas où on compte les pions vers le bas, il suffit de vérifier que la ligne n'est pas trop grande. Maintenant vous pouvez remarquer que j'ai introduit plusieurs répétitions puisque cette expression apparaît ici, ici et ici. Et celle-ci apparaît également ici. Alors, pour éviter ça, je vais introduire deux constantes qui vont contenir les valeurs de ces expressions. Ça va s'écrire de cette façon je vais appeler mes constantes « ligne_max » et « colonne_max » et je vais les utiliser là où j'ai utilisé mes expressions précédentes. Voilà ! Cette fois-ci c'était vraiment la version finale de nos fonctions « est_ce_gagne » et « compte ». Il nous reste à gérer le cas où la grille devient pleine sans qu'aucun des deux joueurs n'ait gagné. Et c'est ce que nous allons voir dans la dernière vidéo. dans la dernière vidéo.

notes

résumé

14m 2s