Support de cours

Cours:

Initiation à la programmation (en C++)

Vidéo:

Puissance 4 - moteur de jeu (partie 1)

Concepts (extraits des sous-titres générés automatiquement) :

Couleurs des joueurs. Besoin d'un joueur. Couleur du joueur courant. Petit bout de code. Not valide. Numéro de colonne. Couleur du joueur. Chemin difficile de la programmation de notre jeu. Joueur jaune. Version actuelle. Exemple de modularisation possible. Petit commentaire. Validité du coup. Valeur de retour de l'appel. Int main.

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

vers la vidéo

EPFL

Puissance 4 : moteur de jeu
(Partie 1)

Initiation à la programmation (C++)

Vincent Lepetit, Jean-Cédric Chappelier et Jamila Sam

...

notes

résumé

0m 0s

Jouons !

EPFL

A ce stade, nous avons vu :

- les structures de données choisies (`Grille`, `Couleur`)
- 2 fonctionalités simples : `initialise` et `affiche`
- 1 fonctionalité plus complexe : `joue`
  qui vérifie si un coup est valide et place le jeton dans le jeu

Continuons donc sur le chemin difficile de la programmation de notre jeu puissance 4

notes

résumé

0m 1s

Jouons !

EPFL

i

où à ce stade nous avons de quoi représenter le jeu : la « Grille », de quoi représenter les couleurs des joueurs ; nous avons de quoi initialiser le jeu et l'afficher, et enfin nous avons une fonction un peu plus complexe qui nous permet de jouer un pion dans le jeu, si tant est que le coup indiqué est valide. Mais à ce stade là, nous ne sommes toujours pas capable de effectivement jouer. Donc, essayons de jouer.

notes

résumé

0m 9s

Jouons !

```
int main()
{
  Grille grille;
  initialise(grille)
  return 0;
}
```

Pour cela, essayons d'écrire une méthode « main » qui nous permet effectivement de jouer. Donc on commence comme d'habitude : « int main » et puis le corps du « main » et commençons donc bien sûr par définir un jeu c'est-à-dire ...une grille, que l'on va initialiser ; tout ceci est normal,

notes

résumé

0m 34s

Jouons !

```
int main()
{
  Grille grille;

  initialise(grille);
  affiche(grill

  return 0;
}
```

et puis au départ bien sûr il faut la montrer aux joueurs

notes

résumé

0m 49s

Jouons !

```
int main()
{
  Grille grille;

  initialise(grille);
  affiche(grille);

  Couleur couleur_j

  return 0;
}
```

donc on va aussi l'afficher. Voila, que faisons nous ensuite ? Nous avons besoin d'un joueur on va donc le représenter par une couleur. Donc pour ça on va commencer par déclarer une variable [de type] « Couleur » qui sera la couleur du joueur courant.

notes

résumé

0m 52s

Jouons !

```
int main()
{
  Grille grille;

  initialise(grille);
  affiche(grille);

  Couleur couleur_joueur(jaune);

  |

  return 0;
}
```

Et on va l'initialiser, donc par exemple à « jaune ». Qu'est ce que l'on veut faire ensuite ? Ce que l'on veut faire c'est bien sûr se faire dérouler le jeu, tant que l'on peut jouer, et donc, une boucle- est ce que ici c'est une boucle « while » ou une boucle « do-while » ? Est ce qu'on va commencer par tester quelque chose ? Ou est ce qu'on va commencer par jouer ?

notes

résumé

1m 5s

Jouons !

```
int main()
{
  Grille grille;

  initialise(grille);
  affiche(grille);

  Couleur couleur_joueur(jaune);

  do {

  return 0;
}
```

Ici toutes les conditions sont bonnes le jeu est initialisé, on a un joueur donc on peut commencer par jouer et donc ça va être une boucle « do-while ». Donc on écrit notre « do », et puis, qu'est ce qu'on va faire dans cette boucle ?

notes

résumé

1m 25s

Jouons !

```
int main()
{
  Grille grille;

  initialise(grille);
  affiche(grille);

  Couleur couleur_joueur(jaune);

  do {

  return 0;
}
```

La première chose à faire, bien sûr,

notes

résumé

1m 37s

Jouons !

```
int main()
{
  Grille grille;

  initialise(grille);
  affiche(grille);

  Couleur couleur_joueur(jaune);

  do {
    if (couleur_joue

  return 0;
}
```

c'est de demander au joueur de nous indiquer un coup. Commençons donc par afficher un message au joueur pour lui indiquer qu'on attend qu'il nous entre un coups, et dans ce message indiquons même la couleur du joueur Pour ça on pourrait utiliser une fonction « affiche_joueur » ; mais je vais ici l'écrire directement

notes

résumé

1m 39s

Jouons !

```
int main()
{
  Grille grille;

  initialise(grille);
  affiche(grille);

  Couleur couleur_joueur(jaune);

  do {
    if (couleur_joueur == jaune) {
      cout << "Joueur |

  return 0;
}
```

pour qu'on voit concrètement de quoi il s'agit.

notes

résumé

1m 58s

Jouons !

```
int main()
{
  Grille grille;

  initialise(grille);
  affiche(grille);

  Couleur couleur_joueur(jaune);

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

    size_t colonne;
    cin >> colonne;
    --colonne;  // les indices des tableaux commencent par

  return 0;
}
```

Donc, si la couleur du joueur est jaune alors je vais afficher : « joueur jaune entre un coup » et sinon, si c'est donc le joueur rouge j'affiche que c'est le joueur avec la convention donc rouge, c'est-à-dire 'O' qui doit rentrer un coup c'est-à-dire, un numéro de colonne. Ensuite on va donc bien sûr devoir saisir sa réponse donc pour ça, on va déclarer par exemple, une colonne donc une variable de type « size_t » puisqu'on l'utilisera pour indexer notre grille, dans laquelle donc on récupère la réponse du joueur, et il ne faut pas oublier que les joueurs vont jouer entre 1 et 7, alors que nous, nous jouons, puisqu'en C++ ces tableaux sont indexés entre 0 et 6. Donc pour ça il faut absolument décrémenter,

notes

résumé

2m 3s

soustraire 1 à colonne. Maintenant qu'on a un numéro de colonne qui nous a été fourni par le joueur on peut le jouer en utilisant donc la fonction « joue » que l'on avait développée dans les vidéos précédentes. Donc pour ça on déclare un booléen « valide » qui va tester la validité du coup et on appelle, donc ici, la fonction « joue » dans l'initialisation du booléen. Voila donc peut être un petit commentaire sur cette ligne donc ici on déclare une variable « valide » de type booléen qu'on initialise directement avec la valeur de retour de l'appel donc de notre fonction joue à laquelle je vous rappelle on passait la grille, la colonne, ici qu'on vient de déclarer et la couleur du joueur courant.

notes

résumé

2m 49s

Jouons !

```
int main()
{
  Grille grille;

  initialise(grille);
  affiche(grille);

  Couleur couleur_joueur(jaune);

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

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

    bool valide( joue(grille, colonne, couleur_joueur) );

  return 0;
}
```

Donc comme cette fonction teste si le coup a été valide et on a récupéré ça ici dans la variable « valide », il faut bien sûr bien en informer le joueur

notes

résumé

3m 34s

Jouons !

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

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

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

    affiche(grille);

    // on change la couleur pour la couleur de l'autre joueur:
    if (couleur_joue

  return 0;
}
```

si le coup n'était pas valide et donc pour ça si 'not valid' alors on affiche un message prévenant le joueur. Une fois le coup joué ou alors, le joueur averti du fait que son coup n'était pas valide, dans tout les cas, on affiche à nouveau le jeu pour que le joueur voit de quoi il s'agit, et puis, décalons un peu tout ça, on va pouvoir maintenant passer à l'autre joueur, donc tout simplement si,

notes

résumé

3m 42s

la couleur du joueur courant est jaune, alors la couleur du joueur courant doit devenir rouge. Et sinon bien sûr, donc « else » c'est que la couleur du joueur courant doit être jaune. Là, encore une fois ce petit bout de code ici, exactement comme l'affichage précédent tout à l'heure, on aurait pu en faire une fonction qui s'appellerait par exemple alterne joueur mais ici c'est suffisamment simple dans ce « main » je voulais donc vous montrer globalement comment on pouvait organiser le jeu.

notes

résumé

4m 13s

Jouons !

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

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

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

  affiche(grille);

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

  return 0;
}
```

Voila donc à ce stade on a demandé un coup au joueur, récupéré sa réponse, on a testé la validité du coup, si il était valide on a joué effectivement le pion dans la grille, si il n'était pas valide on a averti le joueur. Dans tout les cas, on a affiché le jeu et on a inversé donc les couleurs des joueurs, donc on a terminé un tour de jeu

notes

résumé

4m 44s

ce qui fait que on peut maintenant finir notre boucle do-while qui nous permet donc d'alterner les tours de jeux. La question c'est maintenant, quel test mettre ici pour cette condition d'arrêt de jeu. Je vais laisser ça pour l'instant en suspens puisque ça sera l'objet de la prochaine séquence vidéo donc à ce stade je vais simplement laisser un blanc. Ce qui termine notre version presque complète, il manque juste ce test ici qui termine donc la version presque complète du jeu.

notes

résumé

5m 3s

Jouons !

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

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

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

    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(    );

  return 0;
}
```

À ce stade on peut maintenant critiquer un peu ce que l'on a fait, et essayer d'améliorer notre code. Ce qui ce passe ici, c'est que le « main »

notes

résumé

5m 37s

est beaucoup trop grand, puisque, vous voyez, il ne tient même pas sur un transparent qui est bien une preuve que la fonction « main » est trop grande. Donc il va falloir, ce qu'on appelle « modulariser », essayer de mettre des petits bouts dans des fonctions. Et donc je vous propose ici de regrouper tout ceci dans une fonction parce que, qu'est ce que c'es que cette partie ? C'est la partie où vous demandez au joueur de rentrer un coup. On testait la validité et on jouait ce coup, et si le coup n'était pas valide on avertissait le joueur. Tout ceci correspond donc bien à une fonctionnalité donc on peut mettre ça dans une fonction, et on pourrait la nommer, par exemple « demande_et_joue ».

notes

résumé

5m 42s

Jouons !

EPFL

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

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

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

    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(    );

  return 0;
}
```

demande_et_joue

C'est un exemple de modularisation possible, il y en a évidement plusieurs autres et si vous voulez, vous pouvez créer vos propres modularisations. Ce qui compte c'est donc d'identifier les concepts et surtout donc de décomposer pour pouvoir rendre le code beaucoup plus modulaire, beaucoup plus lisible et beaucoup plus maintenable de façon indépendante. Donc revenons à la méthodologie sur les fonctions, si on a identifié comme ça un bout de code qui est, soit répété, pas de copier coller, soit important en soi, qui est notre cas ici. Ce qu'on commence par faire, donc on veut le mettre dans une fonction

notes

résumé

6m 26s

Jouons !

```
  do {

    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(    );

  return 0;
}
```

ce qu'on commence par faire c'est donc, de le couper, simplement.

notes

résumé

7m 1s

Fonction demande_et_joue

```
// demande_et_joue(grille, couleur_joueur);
```

Donc voila, on a supprimé toute cette partie du code et on remplace ce bout de code donc par un appel à notre fonction « demande_et_joue ». Donc, qu'est ce qu'elle va devoir recevoir ? Bien sûr, le jeu, la grille la couleur du joueur courant et cela devrait suffire pour cette fonction.

notes

résumé

7m 2s

Fonction demande_et_joue

```
// demande_et_joue(grille, couleur_joueur);
void demande_e
```

Passons donc maintenant à l'écriture de cette fonction. Je vous ai mis ici en commentaire pour rappel comment on faisait appel à cette fonction. Donc la fonction « demande_et_joue » on l'appelle directement, donc son type de retour ça sera « void »,

notes

résumé

7m 19s

elle s'appelle « demande_et_joue ». Ensuite elle va recevoir donc, une grille, qu'on avait définie de type « Grille ». Est ce que la grille est modifiée ? Oui, donc « demande_et_joue » va permettre de jouer dans la grille donc la grille est modifiée donc on doit faire un passage par référence. Et ensuite on passe la couleur du joueur, est ce que la couleur du joueur est modifiée ? La réponse est clairement non. Donc un passage par valeur suffit ici. Voila donc, le prototype de cette fonction « demande_et_joue », on peut maintenant passer à son corps et là c'est tout simple, il suffit simplement de recoller tout le bout de code qu'on avait supprimé du « main ».

notes

résumé

7m 32s

Fonction demande_et_joue

```
void demande_et_joue(Grille& grille, Couleur couleur_joueur)
{
  if (couleur_joueur == jaune) {
    cout << "Joueur X :  entrez un numéro de colonne" << endl;
  } else {
    cout << "Joueur O :  entrez un numéro de colonne" << endl;
  }

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

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

À ce stade là, lorsque vous pratiquez ce genre d'exercices, il est bon un, de relire votre code pour vérifier si vous n'avez rien oublié, pas fait d'erreur. Deux, le compiler, parce que peut être vous auriez pu faire des erreurs de noms par exemple ici on aurait peut être pu appeler ça « couleur » au lieu de l'appeler « couleur_joueur » et donc du coup,

notes

résumé

8m 9s

Fonction demande_et_joue

```
void demande_et_joue(Grille& grille, Couleur couleur_joueur)
{
  cout << "Joueur ";
  if (couleur_joueur == jaune) {
    cout << 'X';
  } else {
    cout << 'O';
  }

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

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

les aspects ici « couleur_joueur » n'auraient plus compilé. Et enfin, trois, tester bien sûr votre fonction. Continuons dans ce même esprit de modularisation et améliorons un peu là, cette partie là du code. Donc j'ai déjà mentionné précédemment dans la vidéo, qu'on pourrait en faire une fonction, qui serait du genre « affiche_joueur ». Mais même ici sans en faire disons, une fonction, on peut quand même encore arranger ce code en regroupant les parties communes, ne faite jamais de copier coller, là moi je vois du copier coller, je vois joueur, qui a été, ici recopié deux fois, je vois l'espace, « : entrez un numéro de colonne endl » qui a été répété deux fois. Donc ici on va pouvoir réarranger un peu ce code, en commençant donc par afficher la partie commune. Donc ici on affiche le message « joueur » et là je ne mets pas de « endl » puisque je vais continuer à vouloir afficher derrière 'X' ou 'O' et continuer le message, donc ici il n'y a pas de « endl ». Ensuite je garde donc le test qu'il y avait tout à l'heure sur la couleur, donc si la couleur du joueur est jaune j'affiche 'X' si non, j'affiche 'O'. Et à ce stade là, c'est cette partie là en fait qu'on pourrait modulariser et en faire une fonction qui serait, « affiche_couleur » par exemple, qui afficherait systématiquement donc, un 'X' si la couleur qu'elle reçoit en paramètre est jaune et un 'O' si ce n'est pas le cas.

notes

résumé

8m 25s

Terminons donc par l'affichage de la partie commune qui suit l'affichage de la couleur du joueur donc, ici le message « entrez un numéro de colonne » et bien sûr le retour à la ligne.

notes

résumé

9m 49s

Fonction demande_et_joue

```
void demande_et_joue(Grille& grille, Couleur couleur_joueur)
{
  cout << "Joueur ";
  if (couleur_joueur == jaune) {
    cout << 'X';
  } else {
    cout << 'O';
  }
  cout << " :  entrez un numéro de colonne" << endl;

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

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

Terminons en améliorant un autre aspect cette fois ci, algorithmique, de notre fonction « demande_et_joue ». Que se passe-t-il si un joueur donne un coup invalide ? Dans la version actuelle tout ce qui va ce passer, c'est simplement qu'on va afficher le fait que le coup est invalide et puis ensuite on va passer à l'autre joueur. Ce qui ne correspond pas à la vrai réalité du jeu. Donc pour cela qu'est ce qu'il faut faire ? C'est : tant que on reçoit un coup invalide, il ne faut pas lâcher le joueur, il faut continuer à lui redemander. C'est donc à nouveau une boucle do-while puisqu'on commence par demander au joueur d'entrer un coup puis on vérifie si le coup est valide et tant que le coup n'est pas valide, on le redemande.

notes

résumé

10m 0s

Donc on a une boucle ici 'do', sur tout le bloc qui consiste à demander au joueur d'entrer un coup. Puis qui vérifie si coup est valide et enfin, l'informe du fait que le coup n'était pas valide. Donc tout ce bloc on le met ici dans un « while ». « while » quoi ? « while » tant que le coup n'est pas valide, donc « while (not valide) ». donc « while (not valide) ».

notes

résumé

10m 42s