

Support de cours

Cours:

Introduction à la programmation orientée objet (en C++)

Vidéo:

W11-05-morpions-CPP-pt2

Concepts (extraits des sous-titres générés automatiquement) :

Classe jeumorpion. Programmeur-utilisateur. Code résultant. Valeur de ce pointeur. Dernier point important. Programmeurs de la classe jeumorpion. Nombreux écueils. Forme d'entiers. Version jouable. Détails intimes d'implémentation. Forme d'un tableau. Ligne de code. Première case de la grille de jeu. Monde extérieur. Attribut de type grille.



[vers la recherche de séquences vidéo](#)

(dans Introduction à la programmation orientée objet (en C++).)



[vers la vidéo](#)

Center for Digital Education. Plus de matériel de soutien pédagogique ici :

<https://www.epfl.ch/education/educational-initiatives/cede/educational-technologies-gallery/boocs-en/>

Encapsulation et abstraction : étude de cas

(Partie 2)

Introduction à la programmation orientée objet (en C++)

Jean-Cédric Chappelier, Jamila Sam et Vincent Lepetit

...

notes

résumé

0m 0s



Exemple : jeu de Morpion (2)

```
typedef array<int, 9> Grille;

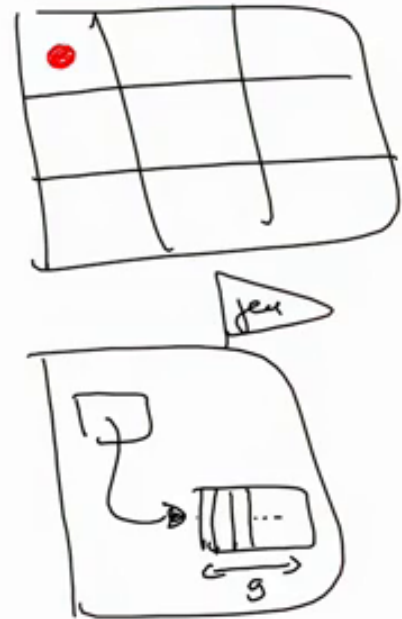
class JeuMorpion {
public:
    void initialise() { grille = new Grille; }
    Grille* get_grille() { return grille; }

private:
    Grille* grille;
};
```

Le joueur rond coche la case en haut à gauche :

```
JeuMorpion jeu;
jeu.initialise();
(*jeu.get_grille())[0] = 1;
```

Convention : 1 représente un rond, 2 une croix et 0 une case vide



Pour placer un rond dans la première case de la grille de jeu, notre programmeur-utilisateur n'a d'autre choix que d'aller regarder comment est concrètement implémentée la classe JeuMorpion. Il doit en effet savoir que sa variable jeu contient désormais un objet dont l'un des attributs est un pointeur sur un tableau de 9 cases à 1D et il doit savoir que placer un rond revient pour lui selon ses conventions

notes

résumé

0m 1s



Exemple : jeu de Morpion (3)

Ce code est parfaitement fonctionnel mais ... pose **beaucoup** de problèmes :

- ▶ L'utilisateur de la classe `JeuMorpion` doit savoir que les **cases sont stockées sous forme d'entiers** dans un **tableau 1D, ligne par ligne** (et non colonne par colonne)
- ▶ Il doit savoir que la valeur entière 0 correspond à une case non cochée, que 1 correspond à un rond, et que la valeur 2 correspond à une croix.
- ▶ L'utilisateur doit connaître « le codage » des données

```
JeuMorpion jeu;
jeu.initialise();
(*jeu.get_grille())[0] = 1;
```

à mettre un "1" dans cette case. Il va donc accéder à la valeur de ce pointeur par le biais de `jeu.get_grille`. Ensuite le `*jeu.get_grille` va lui donner un accès au contenu pointé par ce pointeur donc à la grille et au niveau de la case zéro de cette grille, il va mettre la valeur "1". S'il est suffisamment attentif et précautionneux, notre programmeur-utilisateur de la classe `JeuMorpion` peut parfaitement aboutir au final à une version jouable et fonctionnelle du jeu. Simplement en chemin, il a rencontré de nombreux écueils et problèmes qu'en principe il n'aurait pas du avoir à affronter. Un des problèmes fondamentaux avec cette première version de la classe `JeuMorpion` est que celui qui l'utilise ne peut absolument pas le faire proprement sans en connaître les détails intimes d'implémentation. Il doit par exemple absolument savoir que les cases sont stockées sous la forme d'entiers sous la forme d'un tableau en 1D et avec des conventions de stockage ligne par ligne par exemple. De son côté, il doit également adopter un certain nombre de conventions arbitraires comme par exemple que le "0" représente une case non cochée, le "1" le rond et le "2" une croix. Ces conventions ne seront pas forcément adoptées par d'autres programmeurs.

notes

résumé

0m 37s



Exemple : jeu de Morpion (4)

```
JeuMorpion jeu;  
jeu.initialise();  
(*jeu.get_grille())[0] = 1;
```

- Le code est complètement cryptique pour une personne qui n'est pas intime avec les entrailles du programme. 0, 1, 2 ? Que cela signifie-t-il ? Impossible de le deviner juste en lisant ce code. Il faut aller lire le code de la classe `JeuMorpion` (ce qui devrait être inutile), et en plus ici `JeuMorpion` n'est même pas documentée !

private:
Grille* grille;

`jeu.grille[0] = 1`

de la classe `JeuMorpion` et donc le code résultant sera difficile à lire pour chacun. En conclusion, celui qui utilise la classe `JeuMorpion` ne peut pas utiliser proprement cette classe sans connaître le codage intime des données dans cette classe et sans prendre de son côté un certain nombre de conventions arbitraires. Par ailleurs, le code auquel va aboutir notre programmeur-utilisateur de la classe `JeuMorpion` va lui-même être complètement critique. Quelqu'un qui lirait cette ligne de code serait dans l'incapacité de la comprendre. Que veut dire ce "0" ? Que veut dire ce "1" ? On ne peut comprendre cette ligne de code sans aller regarder l'intérieur de la classe `JeuMorpion`. Ce code produit par le programmeur-utilisateur n'est malheureusement pas robuste à d'éventuels changements faits par le programmeur de la classe `JeuMorpion` dans ses choix d'implémentations. Supposons par exemple qu'il décide d'abandonner son implémentation de la grille sous la forme d'un tableau à 1D et qu'il passe à un tableau à 2D. Eh bien il faudrait réécrire cette ligne de code qui n'est plus adaptée. Donc celui qui utilise la classe `JeuMorpion` est impacté par des modifications faites à l'intérieur de la classe. Il devrait par exemple réécrire cette ligne de code comme ceci, si on passe à un tableau à 2D. Supposons qu'il ait écrit de très nombreuses lignes de code se basant sur ces choix d'implémentations, eh bien l'intégralité de son code est à revoir et éventuellement à réécrire. Enfin, dernier point important et dangereux : par le biais de cette fonctionnalité, le programmeur-utilisateur a accès à la valeur d'un pointeur qui lui donne un accès direct à la grille de jeu telle qu'implémentée. Ceci permettra, nous allons voir un petit peu plus tard, de nombreuses manipulations sur cette grille qui ne sont pas souhaitables. Donc en fait, même si le programmeur-concepteur de `JeuMorpion` a effectivement bien pris la précaution de déclarer comme privé son attribut de type

notes

résumé

1m 49s



Exemple : jeu de Morpion (4)

```
JeuMorpion jeu;
jeu.initialise();
(*jeu.get_grille())[0] = 1;
```

- Le code est complètement cryptique pour une personne qui n'est pas intime avec les entrailles du programme. 0, 1, 2 ? Que cela signifie-t-il ? Impossible de le deviner juste en lisant ce code. Il faut aller lire le code de la classe `JeuMorpion` (ce qui devrait être inutile), et en plus ici `JeuMorpion` n'est même pas documentée !

private:
Grille grille;*

jeu.grille[0] = 1

grille ; ce qui interdit ce genre de manipulation directe de la grille via l'accès à l'attribut, il offre néanmoins la valeur de ce pointeur au monde extérieur via `get_grille` ce qui permet de faire exactement la même manipulation qu'ici mais d'une façon un petit peu plus indirecte. Ceci casse donc complètement l'intérêt de définir l'attribut en privé. Le fait de l'avoir défini en privé ici devient tout simplement complètement inutile. De façon générale lorsque dans une classe, un attribut est un pointeur vers un objet, comme par exemple le tableau chez nous, offrir un accès à l'extérieur, à la valeur de ce pointeur est mauvais pour l'encapsulation car il permet d'accéder aux détails d'implémentations. aux détails d'implémentations.

notes

résumé