
Notes rapides sur Visual Studio et sur l'API de Cyril

Les exemples viennent de l'activité dans laquelle les étudiants doivent concevoir un jeu simple à partir de l'application de Cyril. Ils sont évidemment valides peu importe l'application.

Rédaction : Cyril

Révision et mise en forme : Raphaël

En général, je pense que les informations sont claires et très utiles. Il manque juste quelques informations en plus afin de mieux comprendre l'utilité de tout. Mes commentaires sont en rouge.

I – Les types de fichiers

- Les fichiers `.h` décrivent une classe avec une série d'attributs et de fonctions membres, avec un niveau de visibilité (`private`, `public`, `protected`). C'est ce qu'on appelle une déclaration de classe
- Les fichiers `*.cpp` contiennent l'implémentation d'une classe qui a été déclarée dans les fichiers `*.h`
- Le fichier `*.vcxproj` définit un projet dans lequel se trouvent des fichiers `*.cpp` et `*.h`
- Le fichier `*.sln` définit une solution dans laquelle se trouvent plusieurs projets

Afin de simplifier votre apprentissage, nous aurons une seule solution qui contiendra un seul projet `WindowsApp...` afin de ne pas vous préoccuper des dépendances et des imports/exports de classe(s) de fonction(s).

II – Organisation des fichiers

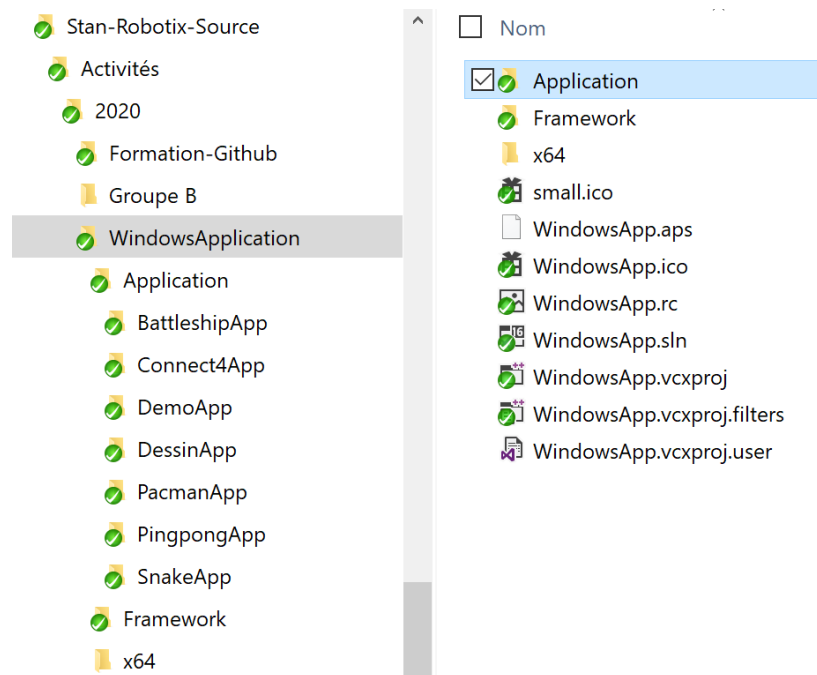
Vous avez deux hiérarchies de fichiers à vous préoccuper :

- Hiérarchie dans la solution
- Hiérarchie dans les répertoires Windows

Il est important de bien classer les fichiers afin d'éviter d'avoir trop de fichiers sous un même répertoire. Règle générale : plus la liste est longue, plus ce sera pénible de chercher un fichier en particulier, et plus vous allez perdre de temps en productivité dans ce que vous voulez vraiment réaliser, c'est-à-dire votre jeu.

III – Hiérarchie dans les répertoires Windows

Dans le cadre de la formation, nous avons la structure suivante

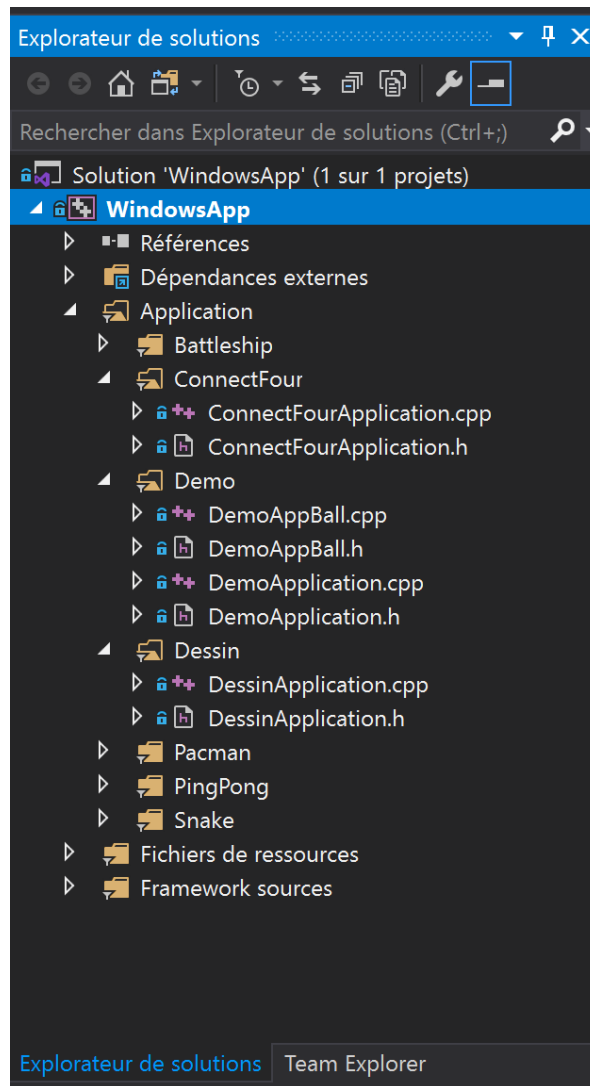


Dans le dossier WindowsApplication, vous allez retrouver la solution et le projet.

Dans le dossier WindowsApplication\Framework, vous allez retrouver le code qui vous permettra de créer un jeu simplifié. Vous ne devez en aucun cas modifier les fichiers qui s’y trouvent. Pourquoi? Tous les étudiants créeront un jeu... mais de la même façon. Il s’agit d’une simplification pour vous.

Dans le dossier WindowsApplication\Application, vous allez retrouver plus jeux (Battleship, Connect4, DemoApp...). Pour éviter les conflits, l’élève responsable de créer le jeu PingPong ne rajoutera des fichiers *.cpp et *.h seulement dans le répertoire WindowsApplication\Application\pingpong. Bref je veux éviter qu’il n’y ait de fichiers partagés (*.cpp et *.h) entre deux jeux.

IV – Hiérarchie dans la solution



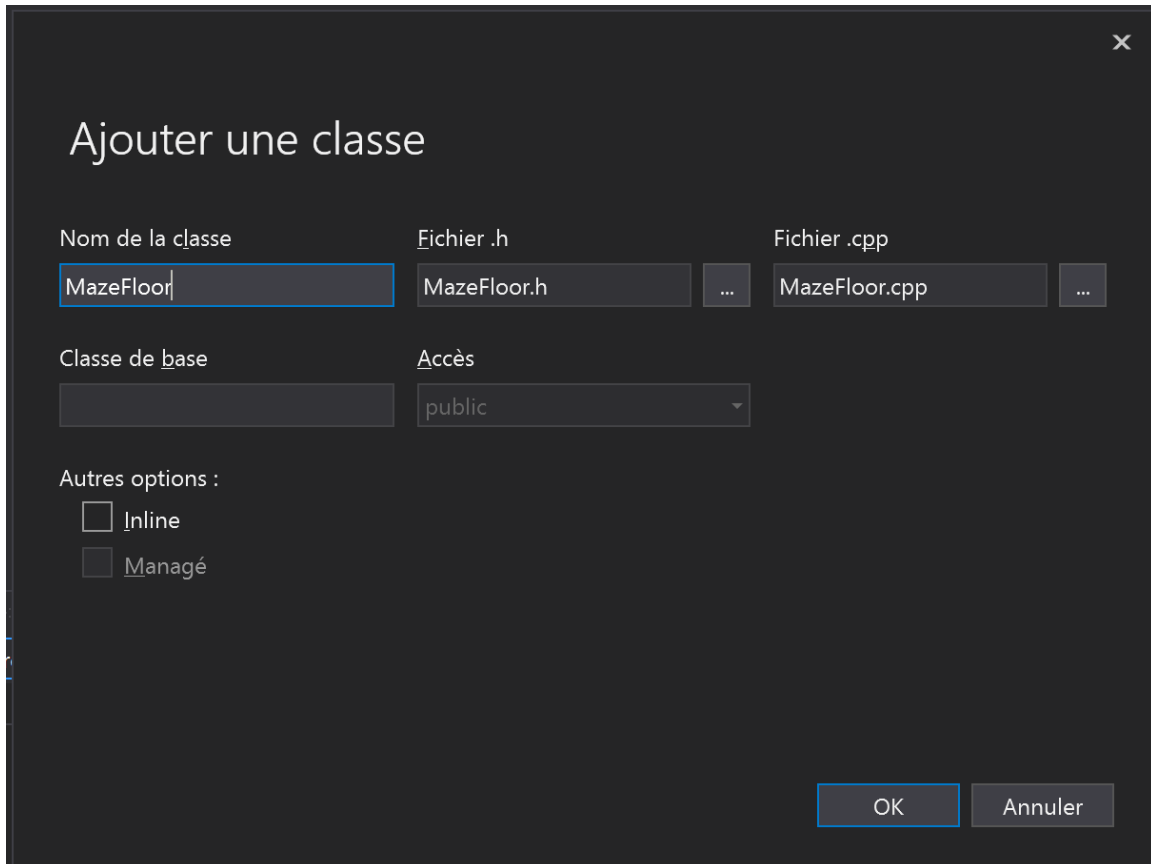
Quand vous ouvrez la solution Stan-Robotix-Source\Activités\2020\WindowsApplication\WindowsApp.sln, vous devriez voir la hiérarchie de la solution à droite. Si vous ne la voyez pas, faites CTRL+W, suivi de S.

Comme vous constaterez, la structure des dossiers est similaire à celle dans les répertoires Windows. C'est volontaire! Si je décidais de réaliser le jeu de Pacman, probablement une 20aine de fichiers seraient créés. Cependant, je ne pense pas qu'un élève apprécierait que je pollue son répertoire avec mes fichiers !

Pour créer un dossier dans l'explorateur de solutions, sélectionner un dossier dans votre répertoire, puis bouton droit de la souris pour afficher le menu contextuel, ensuite ce sera Ajouter\Nouveau Filtre. Et vous donner un nom. Dans le cadre du développement de votre jeu, je doute que vous aurez créé des dossiers. Mais si vous devez pour vous sentir mieux organiser, mettez-vous à l'aise !

V – Ajout d’une classe à un dossier de votre solution

Pour créer une nouvelle classe, cliquez droit sur n’importe quel dossier pour afficher le menu contextuel, ensuite ce sera Ajouter\Nouvelle classe.



The screenshot shows the 'Ajouter une classe' (Add Class) dialog box. It has a title bar with a close button (X). The main area contains several input fields and options:

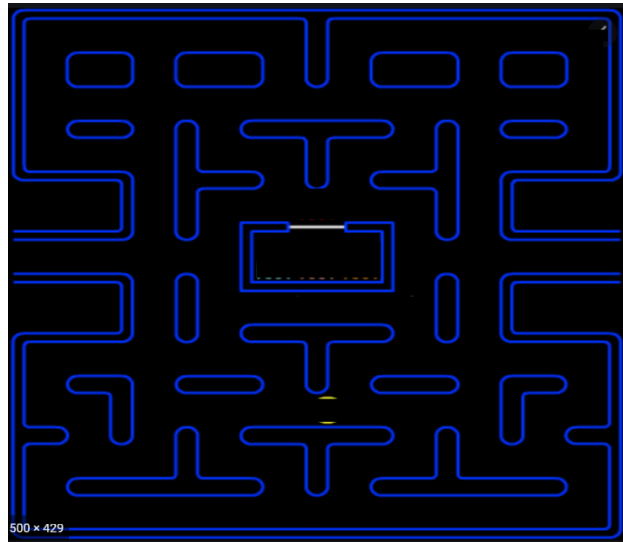
- Nom de la classe**: A text box containing 'MazeFloor'.
- Fichier .h**: A text box containing 'MazeFloor.h' and a browse button (...).
- Fichier .cpp**: A text box containing 'MazeFloor.cpp' and a browse button (...).
- Classe de base**: An empty text box.
- Accès**: A dropdown menu with 'public' selected.
- Autres options :**
 - ☐ **Inline**
 - ☐ **Managé**

At the bottom right, there are two buttons: 'OK' and 'Annuler' (Cancel).

Vous inscrivez le nom de la classe, puis autant que pour le .h, que pour le .cpp, cliquez sur les ... vous sélectionnez le bon répertoire. C’est-à-dire WindowsApplication\Application\PacmanApp, et vous cliquez sur OK. Les fichiers vont être créés au bon endroit dans les répertoires Windows, mais il va falloir déplacer les deux fichiers dans votre propre répertoire : à partir de la fenêtre des solutions, glisser les nouveaux fichiers (e.g. MazeFloor.cpp et MazeFloor.h) dans votre propre répertoire dans la solution.

VI – Comment intégrer une nouvelle classe dans votre projet

Dans le cadre de Pacman, nous avons créé une classe MazeFloor... qui est une classe vide. Le requis (le but, *requirement* en anglais) de la classe MazeFloor, est de dessiner à l'écran le labyrinthe à l'écran, comme montré ci-dessous



Dans mon exemple, je ne chercherai pas comment développer le labyrinthe... car il faut aussi penser à la stratégie de comment déplacer les fantômes et Pacman. Aussi pourquoi se limiter à un seul type de labyrinthe....

Ce que nous allons montrer est tout simplement d'afficher **une ligne à l'écran**.

Dans PacmanApplication.h, nous devons ajouter:

- `#include "MazeFloor.h"`
- Ajouter une propriété à la classe `PacmanApplication` ceci : `MazeFloor mMazeFloor;`

Puisque le but est de dessiner le labyrinthe, il faut rajouter une fonction à la classe MazeFloor pour dessiner, donc

- MazeFloor.h: `void paint(HDC ihdc, RECT& iPaintArea);`
- MazeFloor.cpp: `void MazeFloor::paint(HDC ihdc, RECT& iPaintArea) { }`

Mais en rajoutant la signature dans MazeFloor.h, les types HDC et RECT sont soulignés en rouge.

Définition : Que signifie signature d'une fonction ? C'est tout simplement la fonction elle-même, son type de retour et ses paramètres. Elle peut être dans une classe, ou dans un fichier .h

Définition : HDC. HDC est un identifiant associé à la fenêtre qui vous permettra de dessiner à l'écran.

Définition : RECT. RECT est tout simplement une structure (une classe) qui possède 4 propriétés. (left, right, Top, bottom). Quand la fonction paint est appelé, iPaintArea vous donne les coordonnées de la zone dans laquelle vous pourrez dessiner.

Cela veut dire que l'éditeur C++ ne comprend pas ces deux types. Il faut donc rajouter ceci :

```
#include <Windows.h>

/* Information importante, peut-être à mettre en gras (ou souligner...)? */
```

Dans cette inclusion sont définis tous les types que vous aurez besoin pour une application Windows, en plus de toutes les fonctions pour dessiner, le protocole http.... Bref tout ce qu'il y a d'imaginable spécifique à la plateforme Windows.

Ensuite dans

```
void MazeFloor::paint(HDC ihdc, RECT& iPaintArea)
{
    ::MoveToEx(ihdc, iPaintArea.left, iPaintArea.top, nullptr);

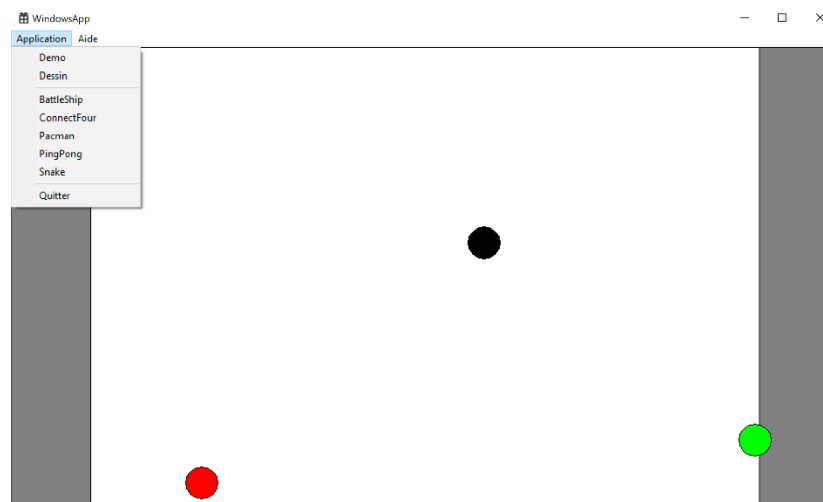
    ::LineTo(ihdc, iPaintArea.right, iPaintArea.bottom);
}
```

La dernière étape est de modifier la fonction comme suit

```
void PacmanApplication::paint(HDC ihdc, RECT& iPaintArea)
{
    mMazeFloor.paint(ihdc, iPaintArea);
}
```

Une ligne allant de haut à gauche vers de bas à droite.

VII – Votre application



Par défaut, le jeu qui est lancé automatiquement est DemoBall. Mais si vous voulez accéder à votre propre jeu, sélectionnez-le à partir du menu principal.