# Note rapide sur Visual Studio

## 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 contient l’implémentation d’une classe qui a été déclaré dans le fichier .h
* Le fichier \*.vcxproj définit un projet dans lequel se trouve des fichiers \*.cpp et \*.h
* Le fichier \*.sln définit une solution dans laquelle se trouve 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 de fonction.

## Organisation des fichiers

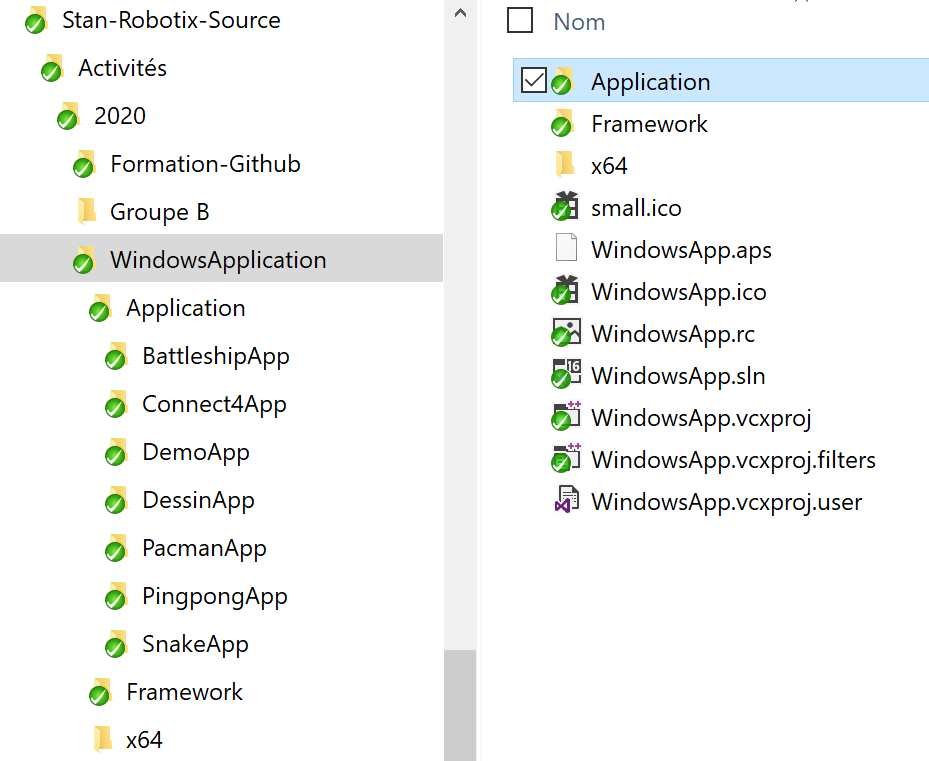
Vous avez deux hiérarchies de fichiers à vous préoccuper.

* 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 longue, plus ce sera pénible de chercher un fichier en particulier, et plus de temps vous allez perdre en productivité ce que vous voulez vraiment réaliser : votre jeu.

### Hiérarchie dans les répertoires Windows:

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

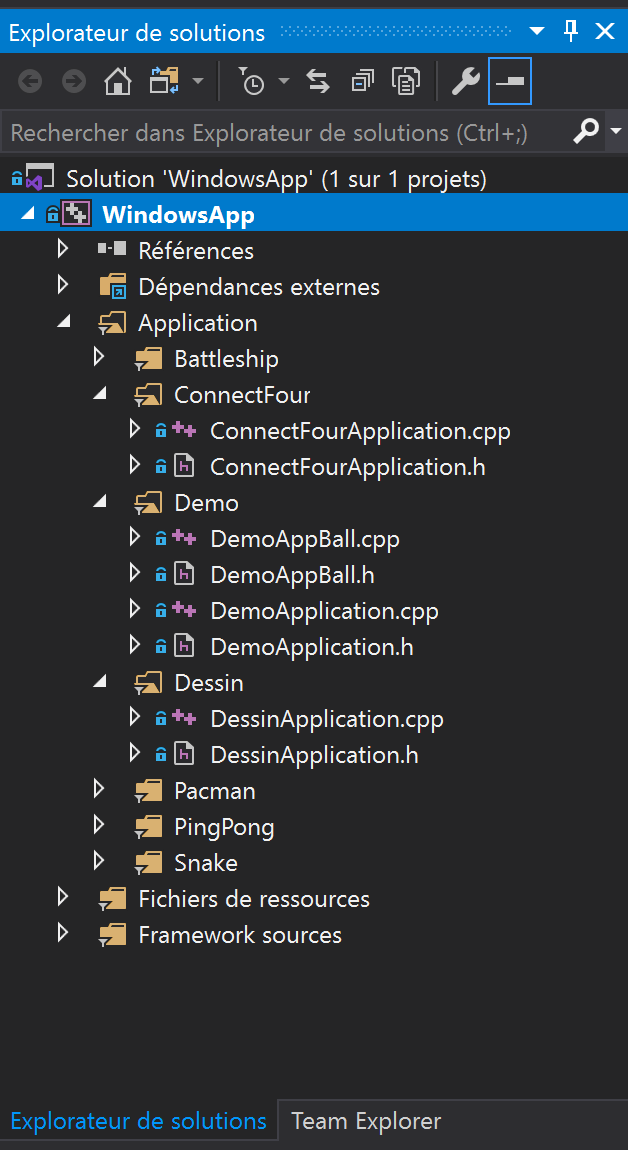


Dans le dossier WindowsApplication, vous allez retrouver la solution et le projet.

Dans le dossier WindowsApplication/Framework, vous allez retrouvez 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 retrouvez 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 ne veux éviter qu’il y ait aucun fichier partagé (\*.cpp et \*.h) entre deux jeux.

### Hiérarchie dans la solution :



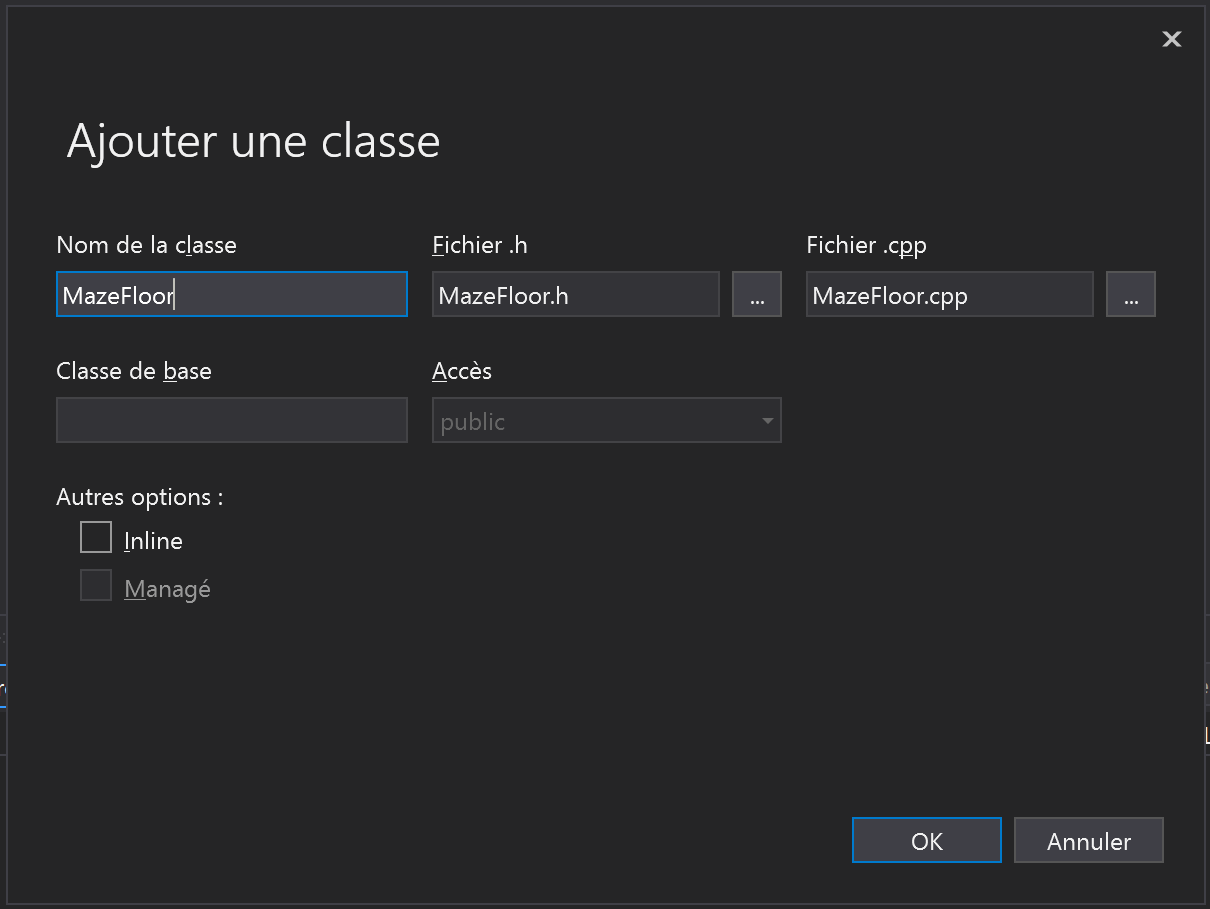
Quand vous ouvrez la solution Stan-Robotix-Source\Activités\2020\WindowsApplication\ WindowsApp.sln, vous devriez voir la hiérarchie de la solution à droite. Si vous ne la voyez pas, faites CTRL+W, suivi de S.

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. Par contre 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!!!

Ajout d’une classe à un dossier de votre solution.

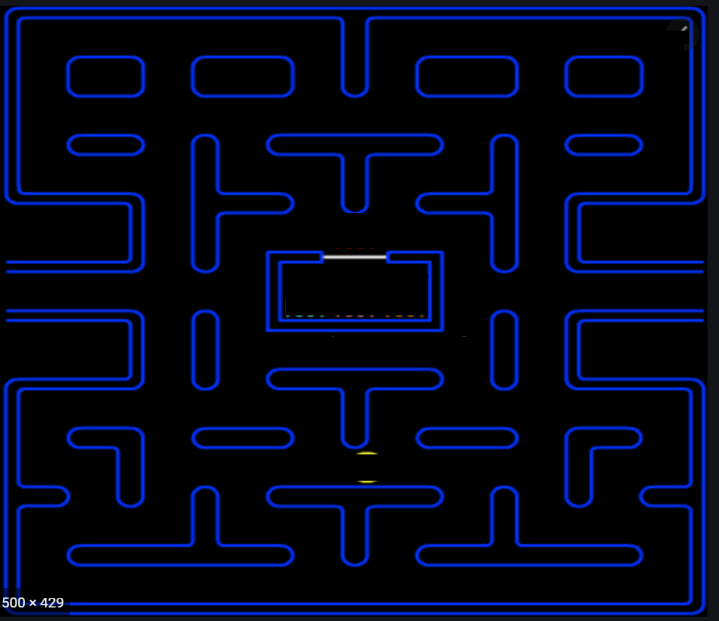
Pour créer une nouvelle classe, cliquez droit sur n’importe quel dossier pour afficher le menu contextuel, ensuite ce sera Ajouter/Nouvelle classe.



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 créer au bon endroit dans les répertoires Windows, mais il va falloir déplacer les deux fichiers dans votre propre répertoire : à partir du fenêtre des solutions, glisser les nouveaux fichiers (e.g. MazeFloor.cpp et MazeFloor.h) dans votre propre répertoire dans la solution.

## Comment intégrer une nouvelle classe dans votre projet.

Dans le cadre de Pacman, nous avons donc créer 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 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;

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

#include <windows.h>

Dans cette inclusion sont définies 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.