

INITIATION A MONOGAME

R1.01 – Algorithmie et Programmation en C#

N. Gruson - IUT Annecy

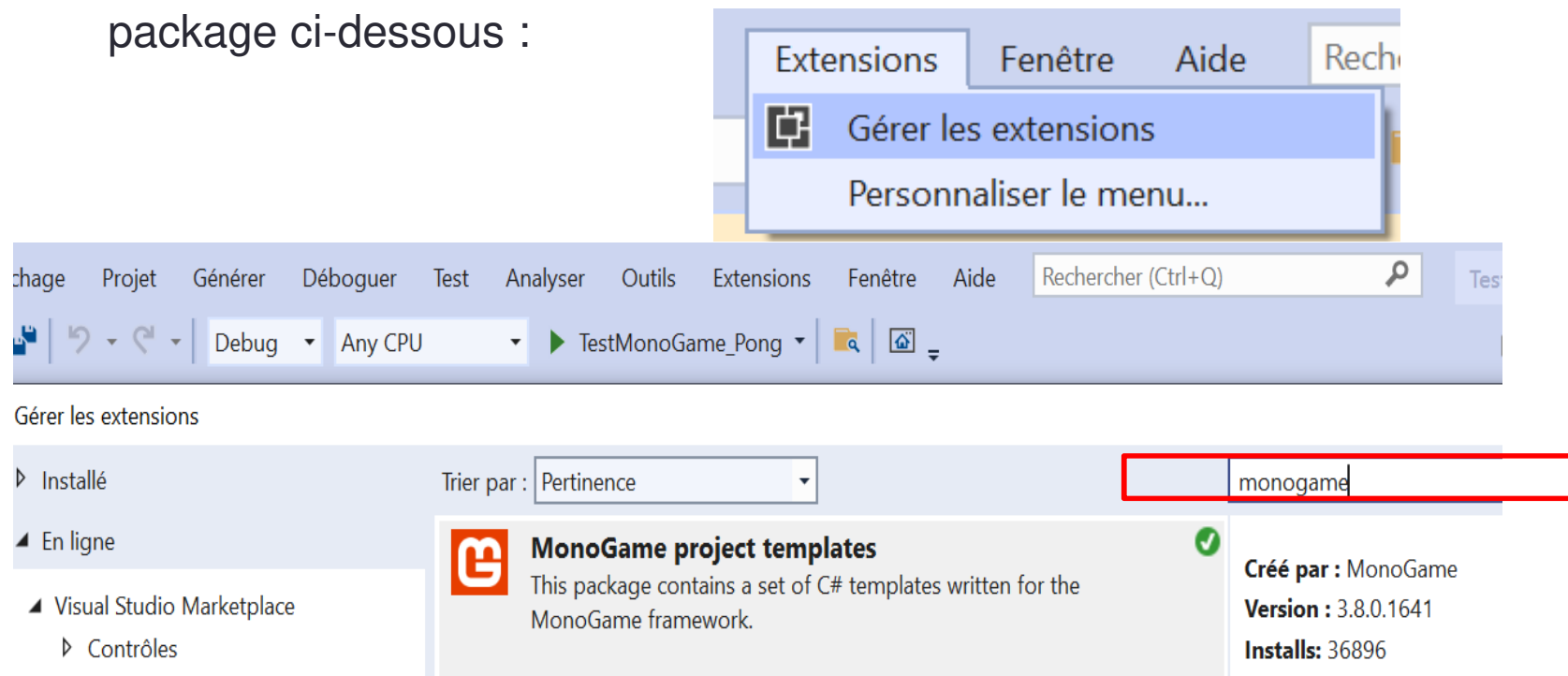
Installation Monogame (1/3)

- **Framework** :
 - contient une librairie (ensemble de classe et de méthodes supplémentaires),
 - impose un cadre, **une organisation**, un squelette, **une méthode de travail pour faire tourner le code fait par le développeur.**
- Installation sous forme d'extension
=> une seule installation, quelque soit le nombre de projet
- Assistant pour créer un projet monogame type

Le framework exécute le code du développeur.

Installation Monogame (2/3)

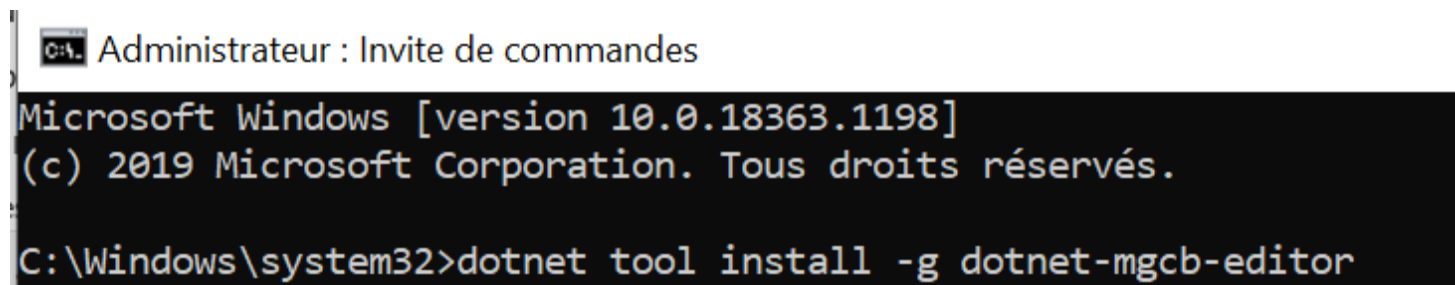
1. Dans le menu Extensions/ Gérer les extensions/
Cherchez dans la zone de recherche puis ajoutez le
package ci-dessous :



Installation Monogame (3/3)

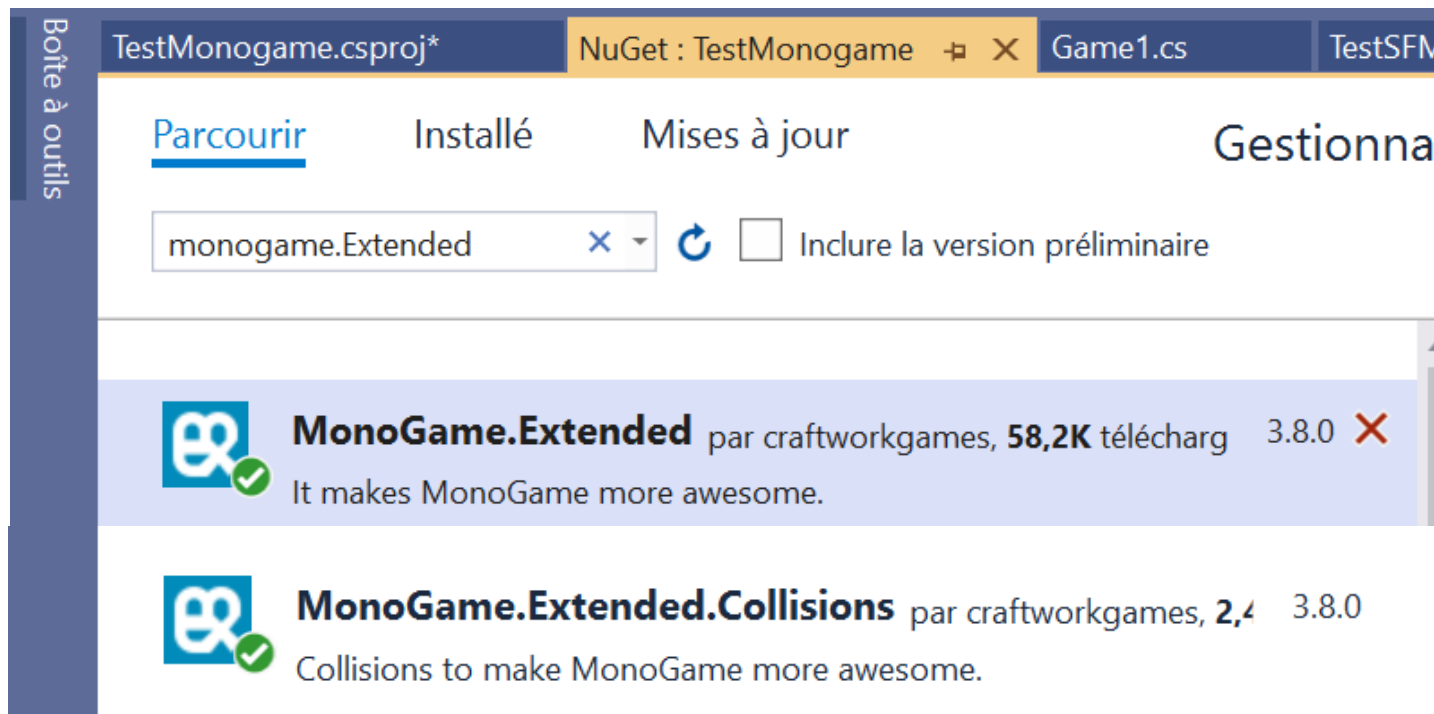
2. Installez l'éditeur de fichier « mgcb-editor » spécifique à monogame. C'est un éditeur pour configurer, ajouter des images au projet. Dans une invite de commande, tapez la commande suivante :

```
dotnet tool install -g dotnet-mgcb-editor
```



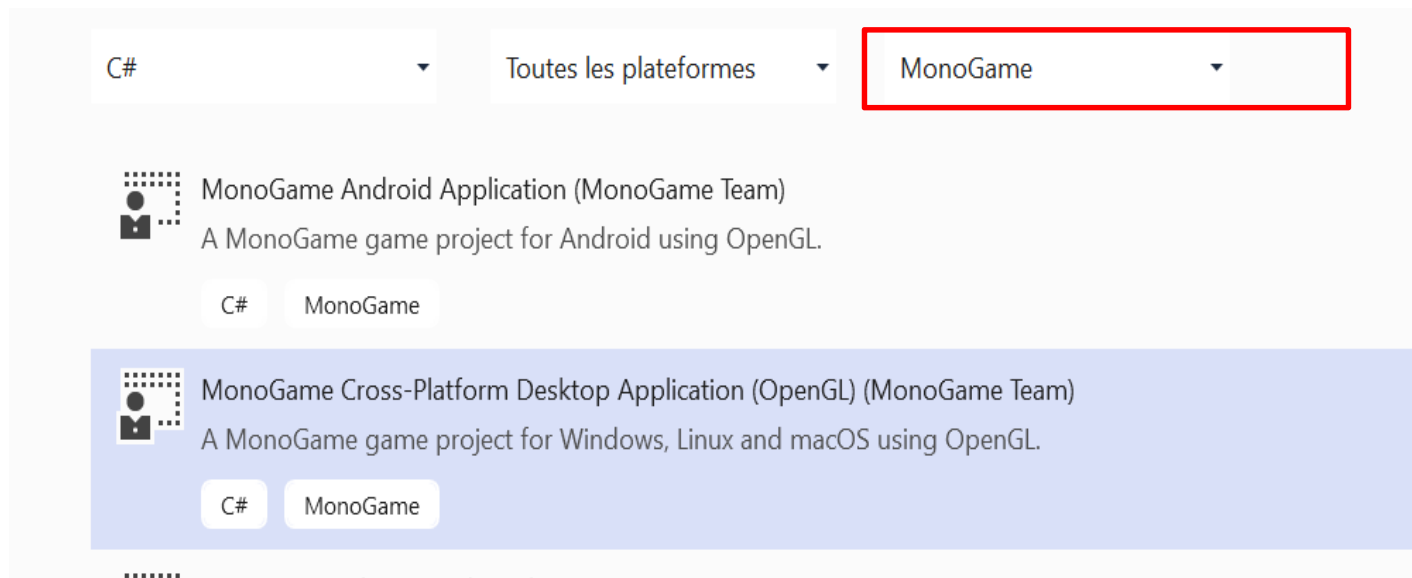
Installation de librairies additionnelles à Monogame

- Pour certains projets de fin de module, il sera utile d'ajouter à vos projets les librairies additionnelles suivantes :
 - Monogame.Extended pour gérer entre autre des scènes
 - Monogame.Extended.Collision pour gérer les collisions



Création d'un projet

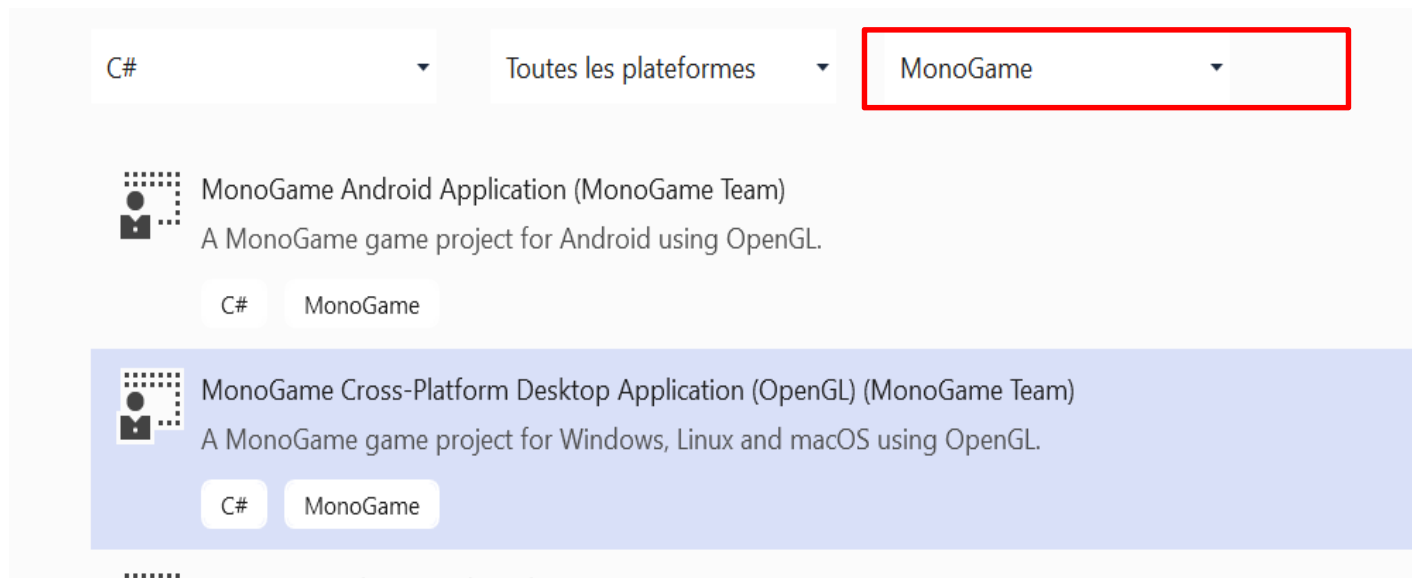
1. Créez un projet de type « Monogame Cross-Platform Desktop » :



Création d'un projet

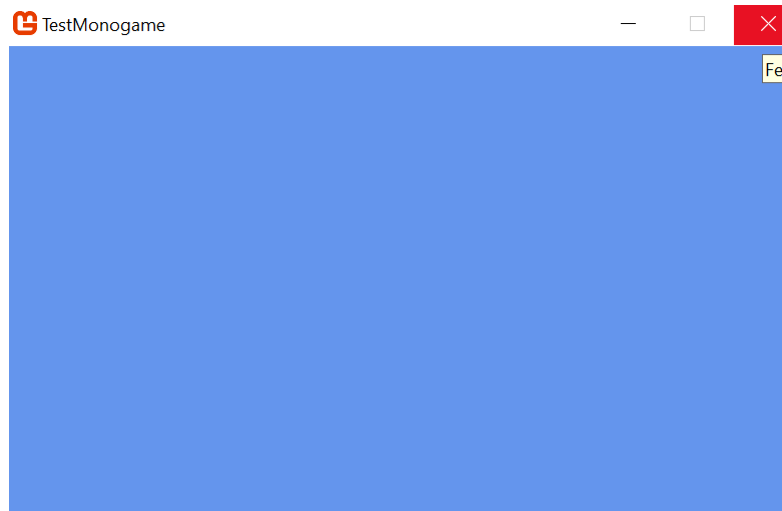
Bien choisir le type de projet !

« Monogame Cross-Platform Desktop » :



Exécution d'un projet

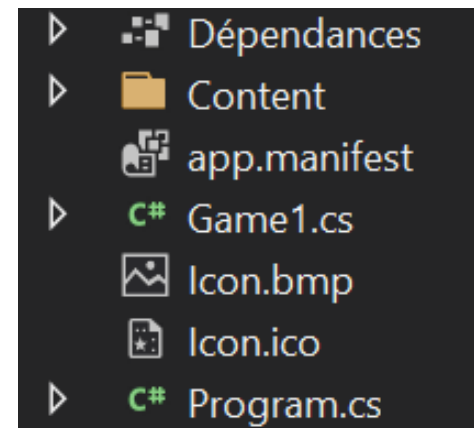
Exécutez le projet : on obtient une fenêtre de jeux :



Contenu d'un projet

Le projet contient :

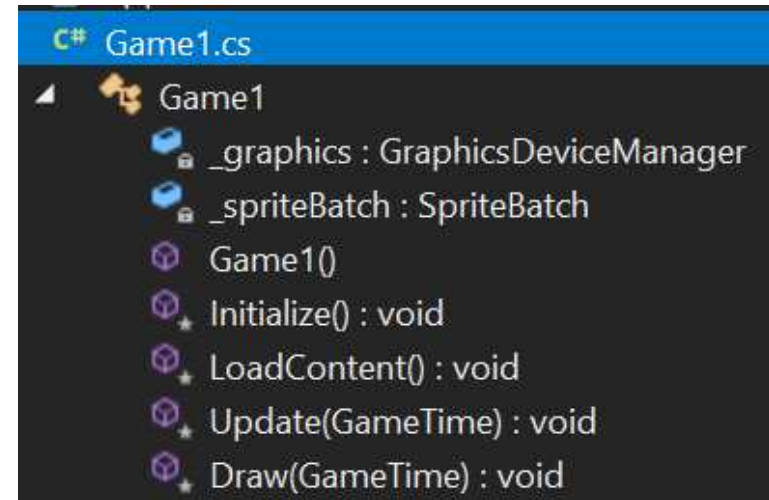
- 2 Classes :
 - Program.cs :
 - crée un objet Game1
 - lance la méthode run
 - Game1.cs : classe qui définit le jeu
- 1 répertoire Content : pour stocker les futurs contenus : images, sons...
- Les icones par défaut visibles dans la barre de la fenêtre de jeu



Contenu de la classe de jeu Game...

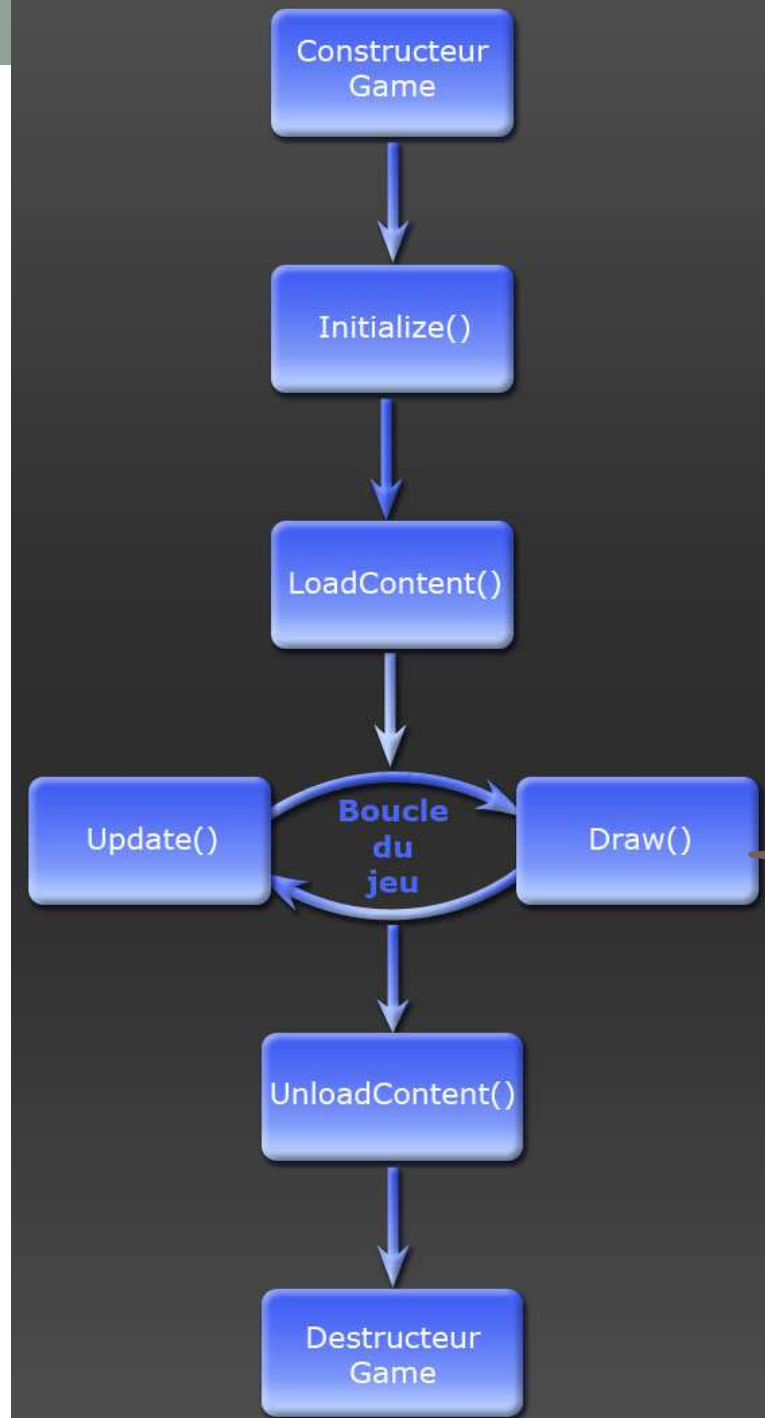
La classe de jeu contient :

- 2 champs :
 - `_graphics` : objet pour définir :
 - la taille de la fenêtre
 - certains paramètres de rendu de la carte graphique
 - `_spriteBatch` : objet pour dessiner images, textes
- 5 méthodes :
 - Le constructeur : initialise les champs
 - `Initialize` : met en place tout la logique de départ du jeu en initialisant les champs avec les valeurs voulues
 - `LoadContent` : charge sons et images
 - `Update` : contient la logique
 - `Draw` : dessine le résultat de la logique calculée par update



Boucle de jeu monogame

la méthode Run (héritée de la classe Game)
appelée dans le main déclenche tout le
processus : Initialize, LoadContent, Update,....



La classe de jeu

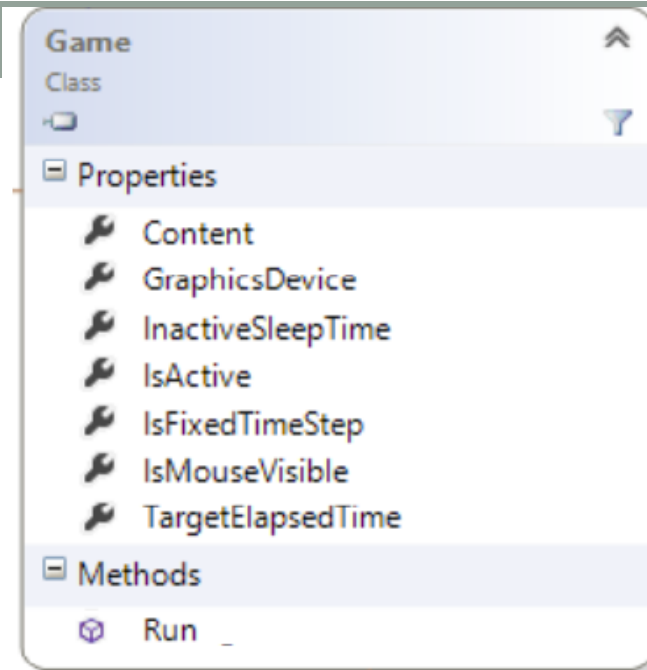
Code par défaut :

```
public class Game1 : Game
{
    private GraphicsDeviceManager _graphics;
    private SpriteBatch _spriteBatch;

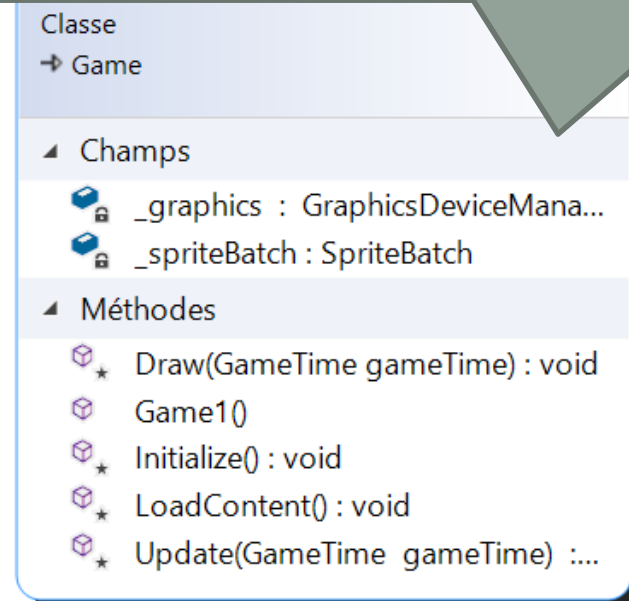
    public Game1()
    {
        _graphics = new GraphicsDeviceManager(this);
        Content.RootDirectory = "Content";
        IsMouseVisible = true;
    }

    protected override void Initialize()
    { base.Initialize(); }

    protected override void LoadContent()
    {
        _spriteBatch = new SpriteBatch(GraphicsDevice);
    }
}
```



— Autre convention de nommage pour distinguer les champs des variables locales (sans mettre this)



La classe de jeu

Code par défaut :

```
protected override void Update(GameTime gameTime)
```

```
{  
    if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed  
        || Keyboard.GetState().IsKeyDown(Keys.Escape))  
        Exit();  
}
```

```
// TODO: Add your update logic here
```

Quitte l'appli si la touche Echap est appuyée

```
    base.Update(gameTime);  
}
```

```
protected override void Draw(GameTime gameTime)
```

```
{  
    GraphicsDevice.Clear(Color.CornflowerBlue);  
}
```

```
// TODO: Add your drawing code here
```

Couleur de fond bleu par défaut

```
    base.Draw(gameTime);  
}
```

Taille de la fenêtre

- Pour changer la taille de la fenêtre, faites le dans la méthode Initialize :

```
_graphics.PreferredBackBufferWidth = 1200;  
_graphics.PreferredBackBufferHeight = 600;  
_graphics.ApplyChanges();
```

- Pour ensuite connaître la taille pour des tests ou autre :

```
int largeur = GraphicsDevice.Viewport.Width;  
int hauteur = GraphicsDevice.Viewport.Height;
```

- Pour mettre en plein écran, faites le dans le constructeur :

```
_graphics.IsFullScreen = true;
```

_graphics est un champ privé pour configurer entre autre Graphics qui est une propriété héritée

Afficher une image

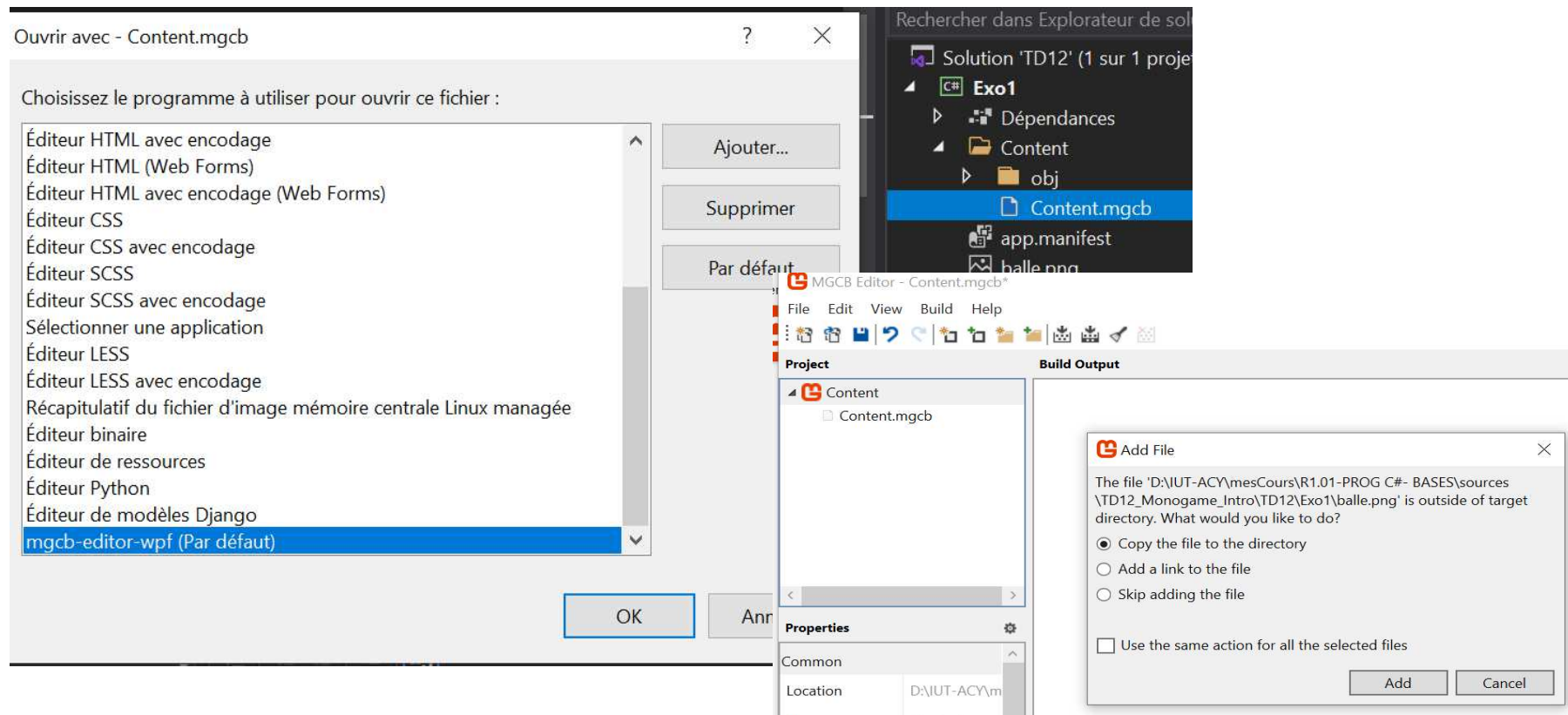
<https://www.freepng.fr/>

4 étapes :

1. Téléchargez une image libre de droit ou créez votre propre image puis ajouter l'image au projet à partir de l'éditeur « mgcb-editor-wpf »
2. Ajoutez un champ de type Texture2D à la classe de jeu et un Vector2 pour gérer le positionnement de l'image.
3. Chargez l'image dans la texture dans la méthode LoadContent et initialisez le positionnement de départ dans Initialize.
4. Dessinez la texture dans la méthode Draw.

Afficher une image - étape 1/4

1. Ajoutez l'image au projet à partir de l'éditeur « mgcb-editor-wpf » :
 1. Ouvrez le fichier Content.mgcb
 2. Ajoutez un élément existant (recherchez votre image)
 3. « Buildez »



Afficher une image - étape 2/4

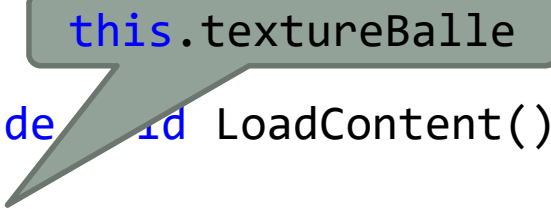
2. Ajoutez un champ de type Texture2D à la classe de jeu et un Vector2 pour gérer le positionnement de l'image.

```
public class Game1 : Game
{
    private Texture2D _textureBalle;
    private Vector2 _positionBalle;
```

Il n'y aura pas d'encapsulation ici !

Afficher une image - étape 3/4

3. Chargez l'image dans la texture dans la méthode LoadContent et initialiser le positionnement de départ dans Initialize.



```
protected override void LoadContent()
{
    this.textureBalle = Content.Load<Texture2D>("balle");
    _spriteBatch = new SpriteBatch(GraphicsDevice);
}

protected override void Initialize()
{
    //TODO: Add your initialization logic here
    _positionBalle = new Vector2(0, 1);
    base.Initialize();
}
```

Afficher une image - étape 4/4

4. Dessinez la texture à la position voulue dans la méthode Draw de Game à l'aide de la méthode Draw de SpriteBatch



```
protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(Color.CornflowerBlue);

    // TODO: Add your drawing code here
    _spriteBatch.Begin();
    _spriteBatch.Draw(_textureBalle, _positionBalle, Color.White);
    _spriteBatch.End();
    base.Draw(gameTime);
}
```

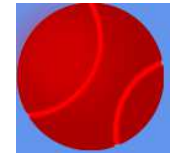
Draw de la classe SpriteBatch

Permet de mettre un filtre coloré sur l'image

Modifier une image

On peut ajouter :

- Un filtre coloré : (ex : rouge ici)



```
_spriteBatch.Draw(_textureBalle, _positionBalle, Color.Red);
```

- De la transparence



```
_spriteBatch.Draw(_textureBalle, _positionBalle, Color.White* 0.5f);
```

- Des effets miroirs ou autre avec la signature suivante :

```
public void Draw(Texture2D texture, Rectangle destinationRectangle, Rectangle? sourceRectangle,  
Color color, float rotation, Vector2 origin, SpriteEffects effects, float layerDepth)
```

Programmer le mouvement d'une image

Il suffit de travailler la position de l'image dans la méthode Update:
Ex: faire une balle qui rebondit de bas en haut.

```
protected override void Update(GameTime gameTime)
{
    if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed
        || Keyboard.GetState().IsKeyDown(Keys.Escape))
        Exit();
    // TODO: Add your update logic here
    // Si la balle touche le bord haut ou le bord bas, on inverse le pas
    if (_positionBalle.Y == 0 ||
        _positionBalle.Y + _textureBalle.Width == GraphicsDevice.Viewport.Height)
        _pas = - _pas;
    // on modifie le Y du vecteur indiquant la position de la balle
    _positionBalle.Y += _pas;

    base.Update(gameTime);
}
```

_pas est un champ
privé initialisé à 1

Améliorer le mouvement d'une image

Si votre programme prend beaucoup de ressources, il faut adapter le déplacement au temps écoulé entre 2 Update :

```
protected override void Update(GameTime gameTime)
{
    if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed
        || Keyboard.GetState().IsKeyDown(Keys.Escape))
        Exit();
    // TODO: Add your update logic here
    // Si la balle touche le bord haut ou le bord bas, on inverse le pas

    if (_positionBalle.Y <= 0 ||
        _positionBalle.Y + _textureBalle.Width >= GraphicsDevice.Viewport.Height)
        _pas = - _pas;
    // on modifie le Y du vecteur indiquant la position de la balle
    _positionBalle.Y += _pas * (float)gameTime.ElapsedGameTime.TotalSeconds;

    base.Update(gameTime);
}
```

_pas est un champ
privé de type float
initialisé à 100

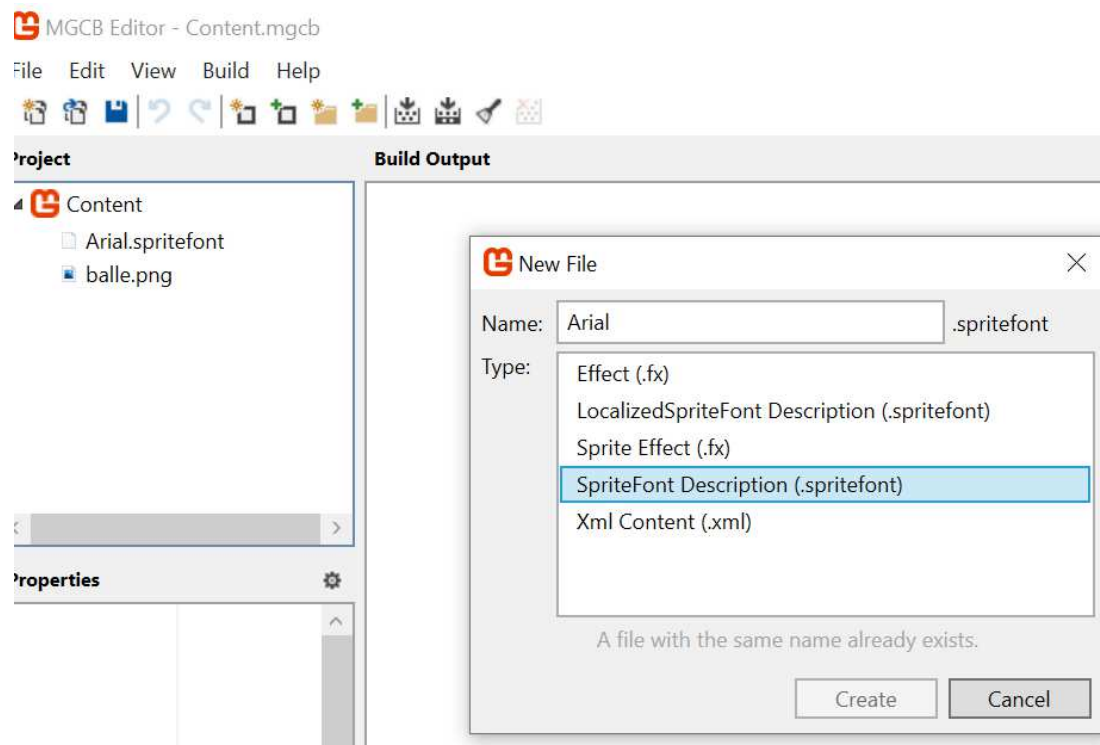
Afficher un texte

4 étapes :

1. Ajoutez un fichier de description de la font à utiliser : un fichier SpriteFont au projet à partir de l'éditeur « mgcb-editor-wpf »
2. Ajoutez un champ de type SpriteFont à la classe de jeu. (et un Vector2 pour gérer le futur positionnement du texte)
3. Chargez la police dans la méthode LoadContent et initialisez le positionnement du text dans Initialize.
4. Dessinez le text dans la méthode Draw.

Afficher un texte – étape 1/4

1. Ajoutez un fichier de description de la font à utiliser : un fichier SpriteFont au projet à partir de l'éditeur « mgcb-editor-wpf » :
 - Ouvrez le fichier qui gère le contenu : « content.mgcb »
 - Ajoutez un nouvel item de type « SpriteFont »



Afficher un texte – étape 2/4

2. Ajoutez un champ de type `SpriteFont` à la classe de jeu et un `Vector2` pour gérer le positionnement du texte.

```
public class Game1 : Game
{
    private SpriteFont _police;
    private Vector2 _positionTexte;
```

Afficher un texte – étape 3/4

3. Chargez la police dans la méthode **LoadContent** et initialisez le positionnement du text dans **Initialize**.

```
protected override void LoadContent()
{
    _police = Content.Load<SpriteFont>("Font");
    // ...
}
```



Nom du fichier SpriteFont

```
protected override void Initialize()
{
    _positionTexte = new Vector2(200, 1);
    // ...
}
```

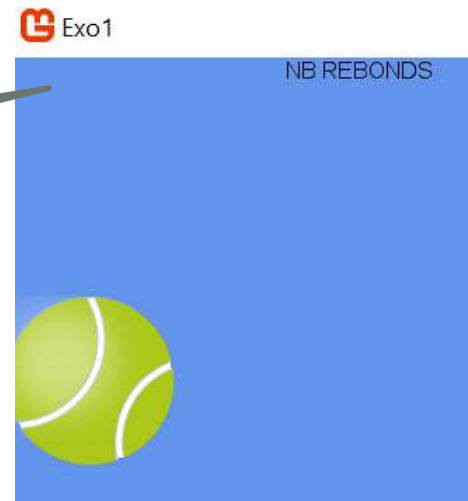
Afficher un texte – étape 4/4

4. Dessinez le text dans la méthode Draw.

```
protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(Color.CornflowerBlue);

    // TODO: Add your drawing code here
    _spriteBatch.Begin();
    _spriteBatch.DrawString(_police, "NB REBONDS", _positionTexte, Color.Black);
    // ...
    _spriteBatch.End();
}
```

Pour changer le style, la taille...: modifiez directement le fichier Font.SpriteFont



Gérer les sons

<https://lasonotheque.org/>

Il faut différencier :

- **La musique de fond : classe Song + MediaPlayer**
- **Des effets sonores : classe SoundEffect (et SoundEffectInstance)**

Jouer une musique

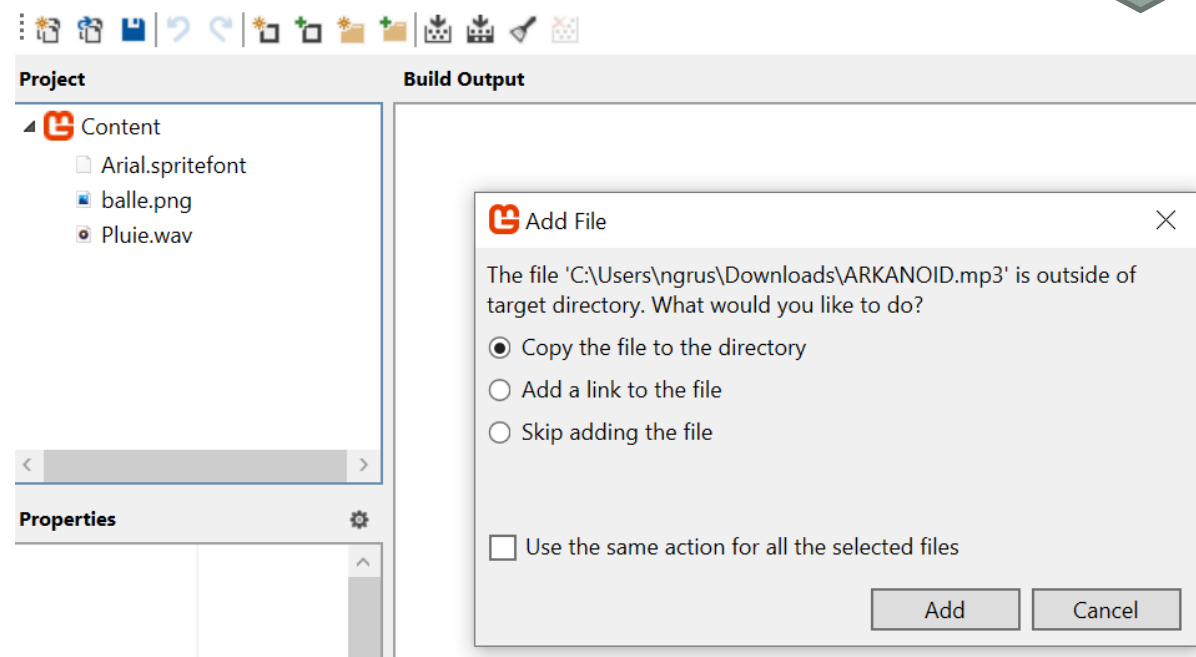
4 étapes :

1. Ajoutez la musique au fichier de contenu « content.mgcb »
2. Ajoutez un champ de type Song à la classe de jeu et
`using Microsoft.Xna.Framework.Media;`
3. Chargez la musique dans la méthode LoadContent et la jouer à l'aide de la classe : MediaPlayer

Jouer une musique – étape 1/3

1. Ajoutez la musique au projet à partir de l'éditeur « mgcb-editor-wpf » :
 1. Ouvrez le fichier Content.mgcb
 2. Ajoutez un élément existant (recherchez votre musique)
 3. « Buildez »

Mp3 , ogg
attendu



Jouer une musique – étape 2/3

2. Ajoutez un champ de type Song à la classe de jeu :

```
using Microsoft.Xna.Framework.Media;
```

```
namespace Exo1  
{  
    public class Game1 : Game  
    {  
        private Song _musique;
```

Jouer une musique – étape 3/3

3. Chargez la musique dans la méthode LoadContent et jouez la à l'aide de la classe : MediaPlayer

```
protected override void LoadContent()  
{  
    _musique = Content.Load<Song>("ARKANOID");  
    MediaPlayer.Play(_musique);  
    //...  
}
```


Agir sur la musique

- Il est possible de mettre en pause :

```
MediaPlayer.Pause();  
MediaPlayer.Stop();
```

- Il est possible de configurer quelques options :

// Définit si le MediaPlayer est muet ou non

```
MediaPlayer.IsMuted = false;
```

// Définit si le MediaPlayer doit jouer en boucle

```
MediaPlayer.IsRepeating = true;
```

// Définit si les titres de la playlist sont joués dans le désordre (true)

```
MediaPlayer.IsShuffled = true;
```

// Définit le volume du lecteur, valeur comprise en 0 et 1

```
MediaPlayer.Volume = 0.8f;
```

Jouez un son

wav
attendu

4 étapes :

1. Ajouter la musique au fichier de contenu « content.mgcb »
2. Ajouter un champ de type `SoundEffect` à la classe de jeu et un :
`using Microsoft.Xna.Framework.Audio;`
3. Charger la musique dans la méthode `LoadContent`
4. Jouer la musique

Jouez un son

2. Ajouter un champ de type `SoundEffect` à la classe de jeu. (et un :

```
using Microsoft.Xna.Framework.Media;
```

```
private SoundEffect _sonRebond;
```

3. Charger la musique dans la méthode `LoadContent`

```
_sonRebond = Content.Load<SoundEffect>("sonRebond");
```

4. Jouer le son (ici, en cas de rebond) dans `Update` :

```
_sonRebond.Play();
```

Gérez les interactions utilisateurs

Il faut gérer les actions du joueur :

- Actions sur le clavier
- Actions avec la souris
- Actions avec une manette

A chaque fois, on récupère l'état du clavier, de la souris, on l'analyse et on fait les actions appropriées.

Gérez le clavier – étape 1/2

1. Définir un champ privé pour stocker l'état du clavier

```
namespace Exo1
{
    public class Game1 : Game
    {
        private KeyboardState _keyboardState;
```

Gérez le clavier – étape 2/2

2. Récupérez l'état du clavier dans la méthode Update et testez les touches de jeux. Ici : on fait bouger la balle avec les touches directionnelles:

```
protected override void Update(GameTime gameTime)
{
    if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed ||
        Keyboard.GetState().IsKeyDown(Keys.Escape))
        Exit();

    // TODO: Add your update logic here
    _keyboardState = Keyboard.GetState();
    if (_keyboardState.IsKeyDown(Keys.Up))
    { _positionBalle.Y--; }
```

GetState : opération lourde
Stockez le résultat afin de ne
l'appeler qu'une seule fois

Toutes les touches de clavier ont une
constante définie dans Keys

Gérez le clavier

On peut tester différentes actions :

- **IsKeyDown(Keys) :**
pour savoir si la touche passée en paramètre est enfoncée.
- **IsKeyUp(Keys) :**
pour savoir si la touche passée en paramètre est relâchée.
- **GetPressedKeys() :**
pour récupérer un tableau de *Keys* enfoncées en même temps

Gérez la souris – étape 1/2

1. Définir un champ privé pour stocker l'état de la souris

```
namespace Exo1
{
    public class Game1 : Game
    {
        private MouseState _mouseState;
```


Gérez la souris – étape 2/2

2. Récupérez l'état de la souris dans la méthode Update et testez . Ex : ici, on fait bouger la balle avec la position de la souris

```
protected override void Update(GameTime gameTime)
{
    if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed ||
        Keyboard.GetState().IsKeyDown(Keys.Escape))
        Exit();

    // TODO: Add your update logic here
    _mouseState = Mouse.GetState();
    _positionBalle.X = _mouseState.X;
    _positionBalle.Y = _mouseState.Y;
```

GetState : opération lourde
Stockez le résultat afin de ne
l'appeler qu'une seule fois

Gérez la souris

On peut récupérer grâce à l'état de la souris :

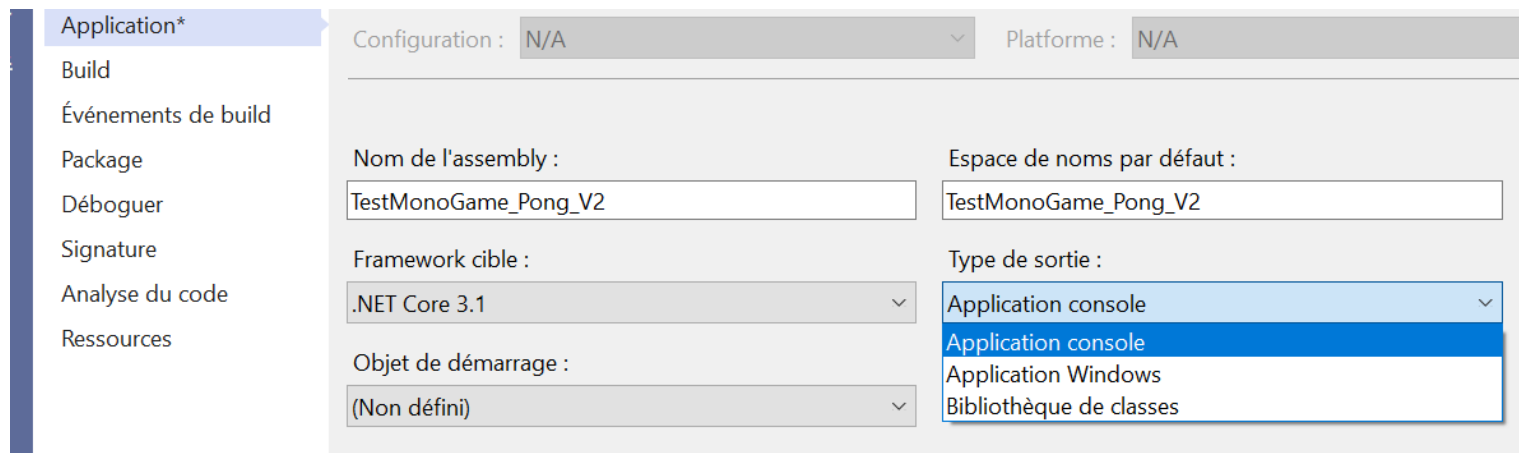
- **X et Y** : donne l'abscisse(X) et l'ordonnée(Y) de la souris par rapport au bord supérieur gauche de la fenêtre de jeu.
- **LeftButton / RightButton / MiddleButton / XBUTTON1/ XBUTTON2** :

```
if ( _mouseState.LeftButton == ButtonState.Pressed)
```
- **ScrollWheelValue** : donne la valeur du scroll de la molette depuis le début du jeu. Dans un sens, le scroll augmente la valeur, dans l'autre il la diminue (à vous découvrir quels sont ces sens ;).

Débuguer un projet Monogame

Si besoin d'afficher des infos dans la console pour déboguer, il faut activer la console dans les propriétés du projet

- Clic droit sur le projet/Propriétés
- Choisir pour type de sortie « Application Console »



Gérer des collisions

2 méthodes simples :

- Faire ses propres tests avec les coordonnées et les tailles des textures
- Utiliser Contains ou Intersects appliquées sur des rectangles initialisés à partir des coordonnées et des tailles des textures

Organiser son code

Pour gérer une image, il faut minimum 3 variables (voir plus) :

- Une pour la texture
- Une pour le positionnement
- Une pour le déplacement

Si dans le jeux, il y a besoin de 10, 15 images , imaginez le nb de variables....

Pour éviter cela, vous pouvez par exemple définir la classe Sprite :

Organiser son code

Sprite
Classe

▲ Champs

- 🔒 `_direction : Vector2`
- 🔒 `_position : Vector2`
- 🔒 `_speed : float`
- 🔒 `_texture : Texture2D`

▲ Propriétés

- 🔧 `Direction { get; set; } : Vector2`
- 🔧 `Position { get; set; } : Vector2`
- 🔧 `Speed { get; set; } : float`
- 🔧 `Texture { get; set; } : Texture2D`

▲ Méthodes

- 📦 `Contains(Point p) : bool`
- 📦 `Initialize(Vector2 position, Vector2 direction, int vitesse) : void`
- 📦 `Intersects(Sprite autre) : bool`
- 📦 `Sprite()`
- 📦 `Update(GameTime gameTime) : void`

On peut encore aller plus loin :
<http://sdz.tdct.org/sdz/le-developpement-de-jeux-video-avec-xna.html#Cratond039uneclasseSprite>

Pour en savoir plus ...

Pour les bases sur le fonctionnement de monogame (et donc de xna) :

<http://sdz.tdct.org/sdz/le-developpement-de-jeux-video-avec-xna.html>

Pour voir le diagramme de classe du framework :

<https://digitalrune.github.io/DigitalRune-Documentation/html/d9205cc0-780b-4302-a5bb-7f35d4268085.htm>

La doc officielle:

<https://www.monogame.net/>

La doc sur les classes :

<https://docs.monogame.net/>