XNA

officiellement XNA's Not Acronymed,

Christian VIAL

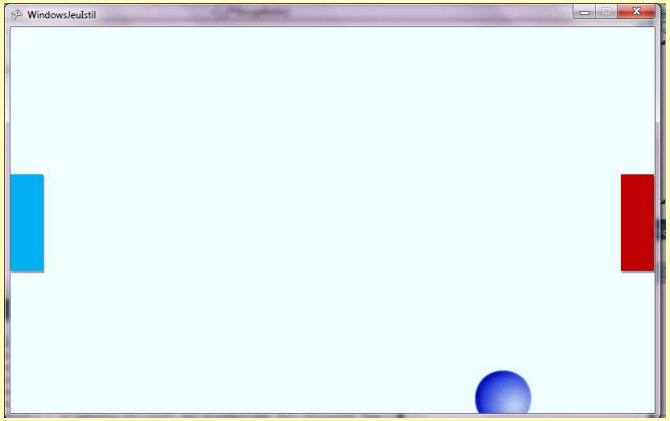
PARTIE INTRODUCTIVE

1. Introduction
2. Affichage de texte
3. Gestion d'un jeu
4. Sérialisation XML
5. TP A REALISER

PARTIE INTRODUCTIVE

Présentation du cours

- *XNA désigne une série d'outils fournis gratuitement par Microsoft qui facilite les développements de jeux pour les plates-formes Windows et Xbox 360 en réunissant un maximum d'outils en provenance de Microsoft et de ses partenaires (DirectX, Visual Studio, PIX, XACT).*

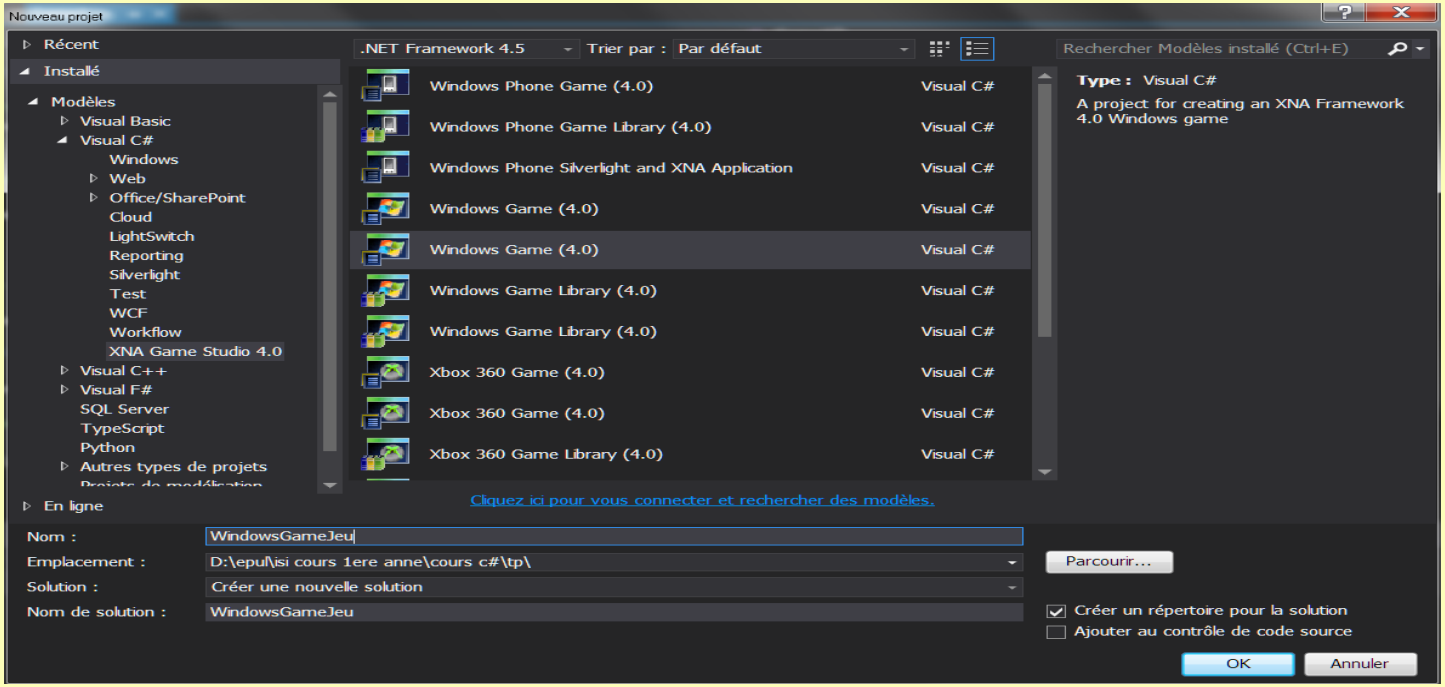


XNA

- **XNA**

Versions d'XNA : Game 4.0 pour 2010 & 2013

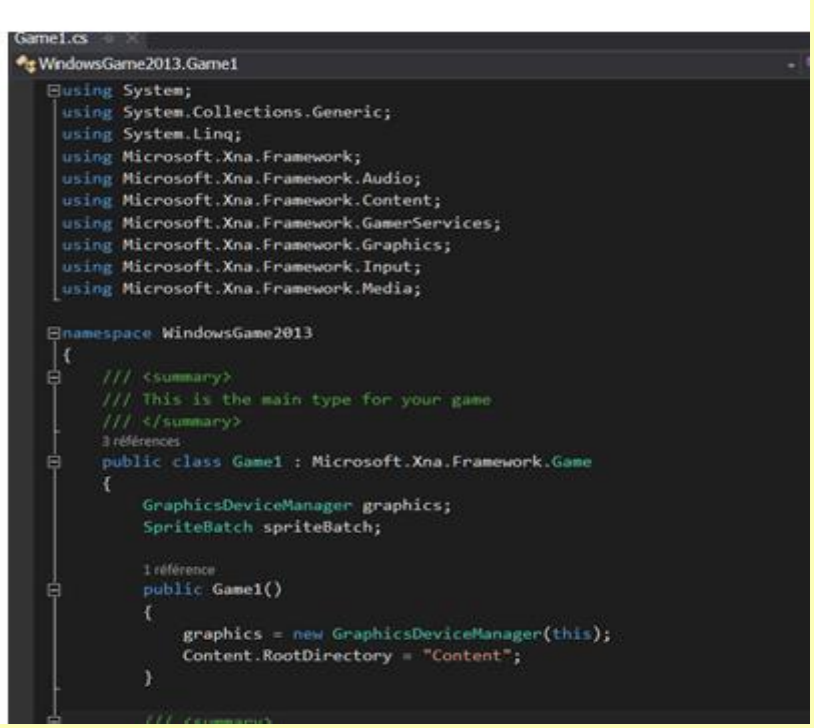
Développement d'une application

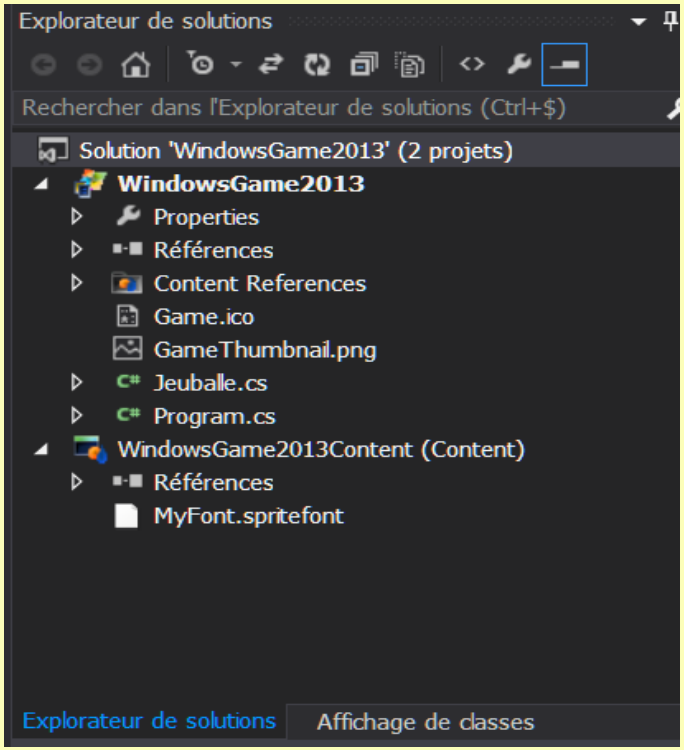


XNA

- **XNA**

On obtient :





Vous obtenez alors une classe nommée Game1 que vous allez renommer en JeuBalle.

XNA

- **XNA**

Code de la classe Jeu qui hérite de Microsoft.Xna.Framework.Game

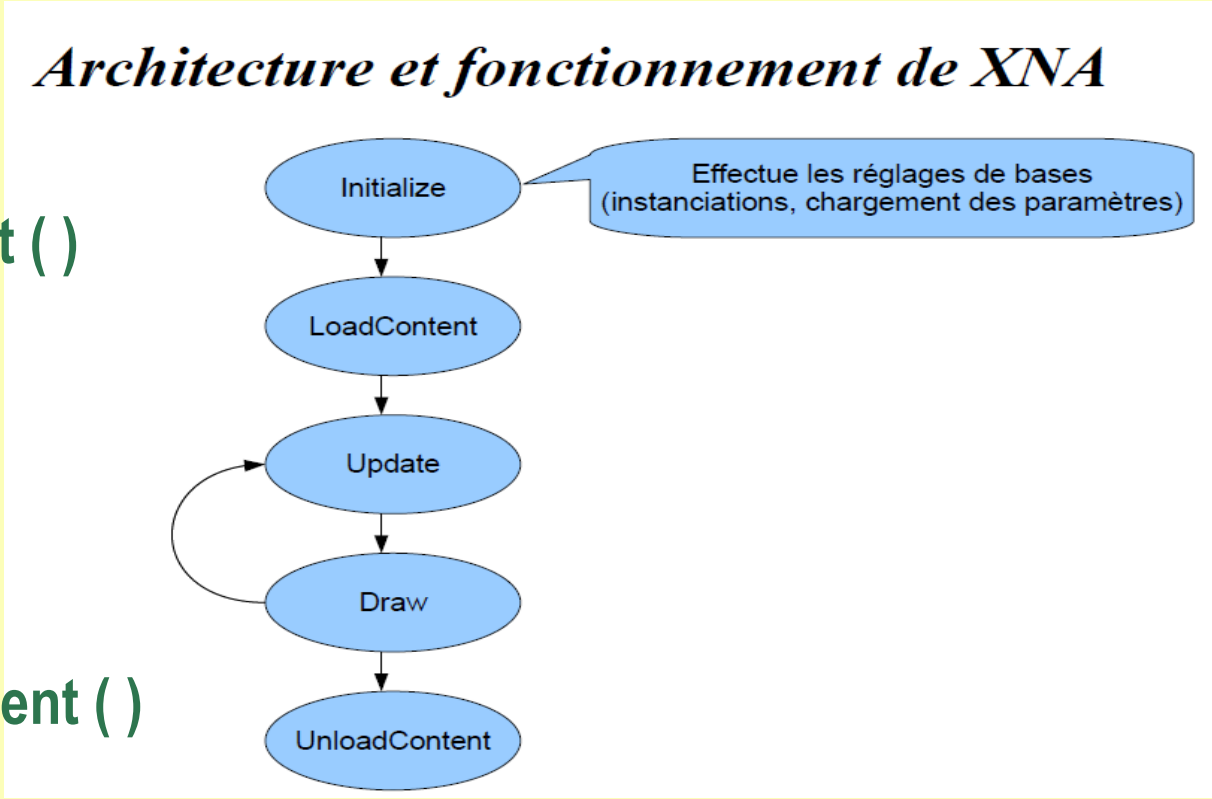
-Initialize ( )

-loadContent ( )

- Update ( )

- Draw ( )

-unloadContent ( )



XNA

- **XNA**

Etude de ces méthodes

-Constructeur de la classe

-Le constructeur se contente d'initialiser les membres :

```
public Jeu()
{
    graphics = new GraphicsDeviceManager(this);
    Content.RootDirectory = "Content";
}
```

GraphicsDeviceManager est une classe qui donne accès aux fonctionnalités de la carte graphique

XNA

- **XNA**

Etude de ces méthodes

-Taille de l'écran de sortie

-On définit une taille d'écran de sortie

```
protected override void LoadContent()
    {
        // Create a new SpriteBatch, which can be used to draw textures.
        spriteBatch = new SpriteBatch(GraphicsDevice);
        // TODO: use this.Content to load your game content here
        graphics.PreferredBackBufferWidth = 1024;
        graphics.PreferredBackBufferHeight = 620;
        graphics.ApplyChanges();
    }
```

XNA

- XNA

Etude de ces méthodes

-Initialize ()

-C'est ici que le développeur initialisera toutes les données de son jeu. Initialize est appelée juste après la méthode Load.

```
protected override void Initialize()
        {
            // TODO: Add your initialization logic here
              base.Initialize();
        }
```

XNA

- **XNA**

Etude de ces méthodes

-LoadContent ()

-Cette méthode gère le chargement et le déchargement des assets du jeu (son, images, objets 3D, fonts ...).

```
protected override void LoadContent()
  {
    // Create a new SpriteBatch, which can be used to draw textures.
        spriteBatch = new SpriteBatch(GraphicsDevice);
        // TODO: use this.Content to load your game content here
  }
```

XNA

- **XNA**

Etude de ces méthodes

-UnLoadContent ()

-Cette méthode gère le déchargement des assets du jeu (son, images, objets 3D, fonts ...).

```
protected override void UnLoadContent()
  {
    //   TODO: Unload any non ContentManager content here
  }
```

XNA

- **XNA**

Etude de ces méthodes

-Gestion de la vie du jeu Update

-Cette méthode assure la vie du jeu, elle est appelée 50 fois par seconde :

```
protected override void Update(GameTime gameTime)
{
        // Allows the game to exit
        if (GamePad.GetState(PlayerIndex.One).Buttons.Back ==
            ButtonState.Pressed)
            this.Exit();
        // TODO: Add your update logic here
        bougeBalle();
        base.Update(gameTime);
    }
```

XNA

- **XNA**

Etude de ces méthodes

-Gestion de la vie du jeu Draw

-Cette méthode Draw s'occupe exclusivement de l'affichage des données.

```
protected override void Draw(GameTime gameTime)
{          // Allows the game to exit
           GraphicsDevice.Clear(Color.CornflowerBlue);

          // TODO: Add your drawing code here
          base.Draw(gameTime);
}
```

- **Premier programme**
  - ***Affichage d'un texte au centre de l'écran***

    Nous commençons par définir un vecteur de position dans la classe. Nous l'initialiserons avec les coordonnées du centre de notre écran dans la méthode

```
private  Vector2 position;

Initialize ( )
    position.X = GraphicsDevice.DisplayMode.Width / 10;
    position.Y = GraphicsDevice.DisplayMode.Height / 4;
```

- **Premier programme : gestion de la police**
  - ***Utilisation d'une police***

    Pour afficher du texte, nous devons charger une police. Cette dernière sera contenue dans un fichier XML

    On déclare dans la classe une variable privée

```
//Police de caractères
private  SpriteFont textFont;
```

Dans la méthode LoadContent ( ), on charge la police

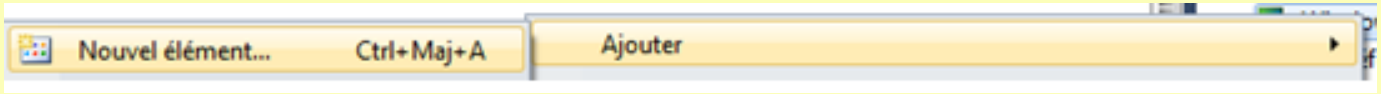
```
this.textFont = Content.Load<SpriteFont>("MyFont");
```

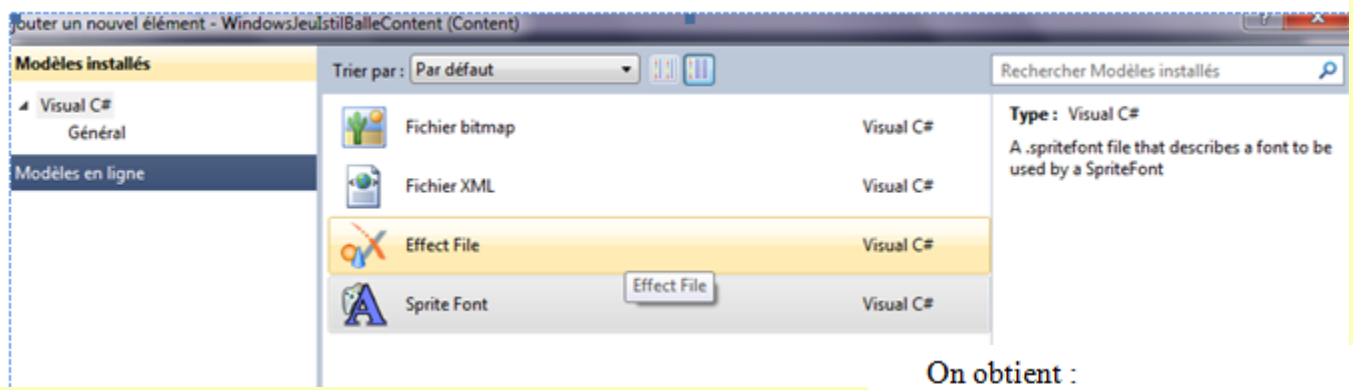
XNA

Premier programme : gestion de la police

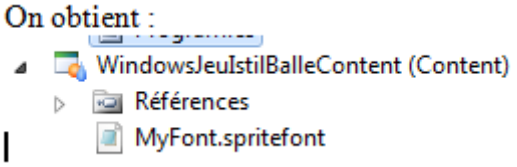
Ajout de la police

Pour afficher du texte, nous devons charger une police. Cette dernière sera contenue dans un fichier XML





On obtient :



XNA

- **Premier programme : gestion de la police**
  - ***Fichier Xml de la police***

MyFont désigne le nom du fichier xml

```
<?xml version="1.0" encoding="utf-8"?>
<XnaContent xmlns:Graphics="Microsoft.Xna.Framework.Content.Pipeline.Graphics">
  <Asset Type="Graphics:FontDescription">
    <FontName>Kootenay</FontName>
    <Size>14</Size>
    <Spacing>0</Spacing>
    <UseKerning>true</UseKerning>
    <Style>Regular</Style>
    <CharacterRegions>
      <CharacterRegion>
        <Start>&#32;</Start>
        <End>&#126;</End>
      </CharacterRegion>
      <CharacterRegion>
        <Start>&#176;</Start>
        <End>&#176;</End>
      </CharacterRegion>
    </CharacterRegions>
  </Asset>
</XnaContent>
```

XNA

- **Premier programme : Affichage du texte**
  - ***Utilisation de la méthode Draw***

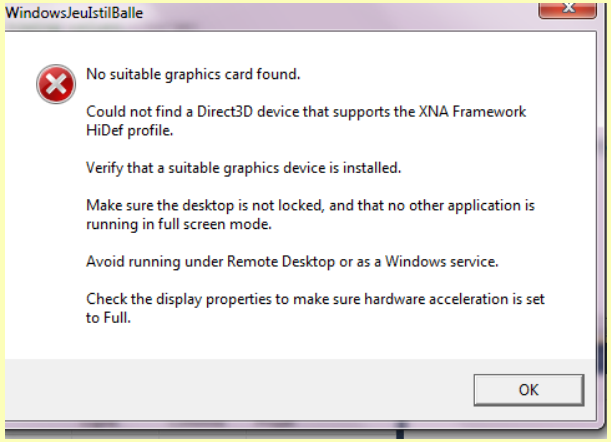
```
protected override void Draw(GameTime gameTime)
        {
            GraphicsDevice.Clear(Color.CornflowerBlue);
            spriteBatch.Begin();
            // TODO: Add your drawing code here
            string text1 = string.Format("Debut de notre premier jeu coordonnees  : X = {0}   Y =
{1}", position.X, position.Y);
            // on affiche le texte
            // police
            // texte
            // positin (x,y)
            // couleur de la fonte
            spriteBatch.DrawString(this.textFont, text1, position, Color.DarkRed);
            spriteBatch.End();
            base.Draw(gameTime);
        }
```

Il faut modifier le Game profile du projet (propriétés) :

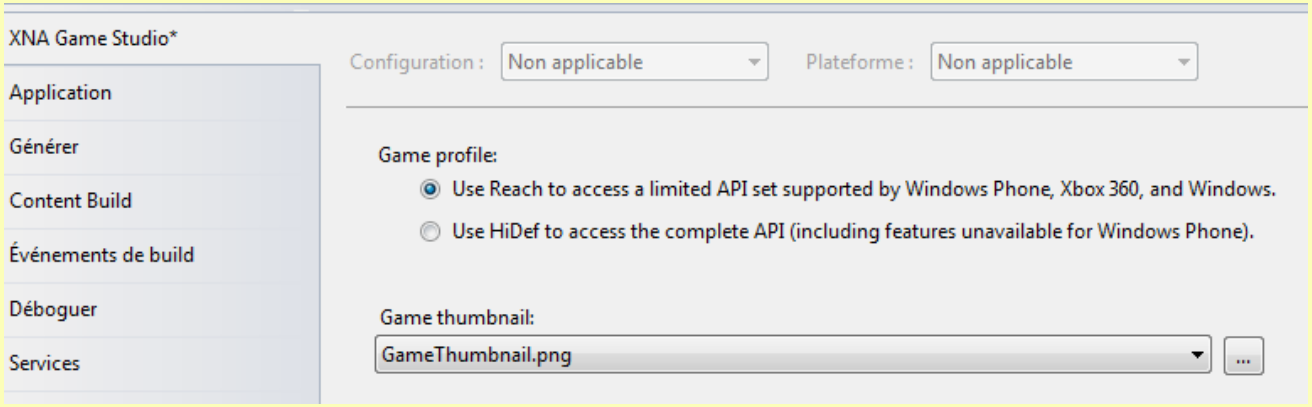
XNA

Premier programme : exécution

Génération d'une erreur



Il faut modifier le Game profile du projet (propriétés) :

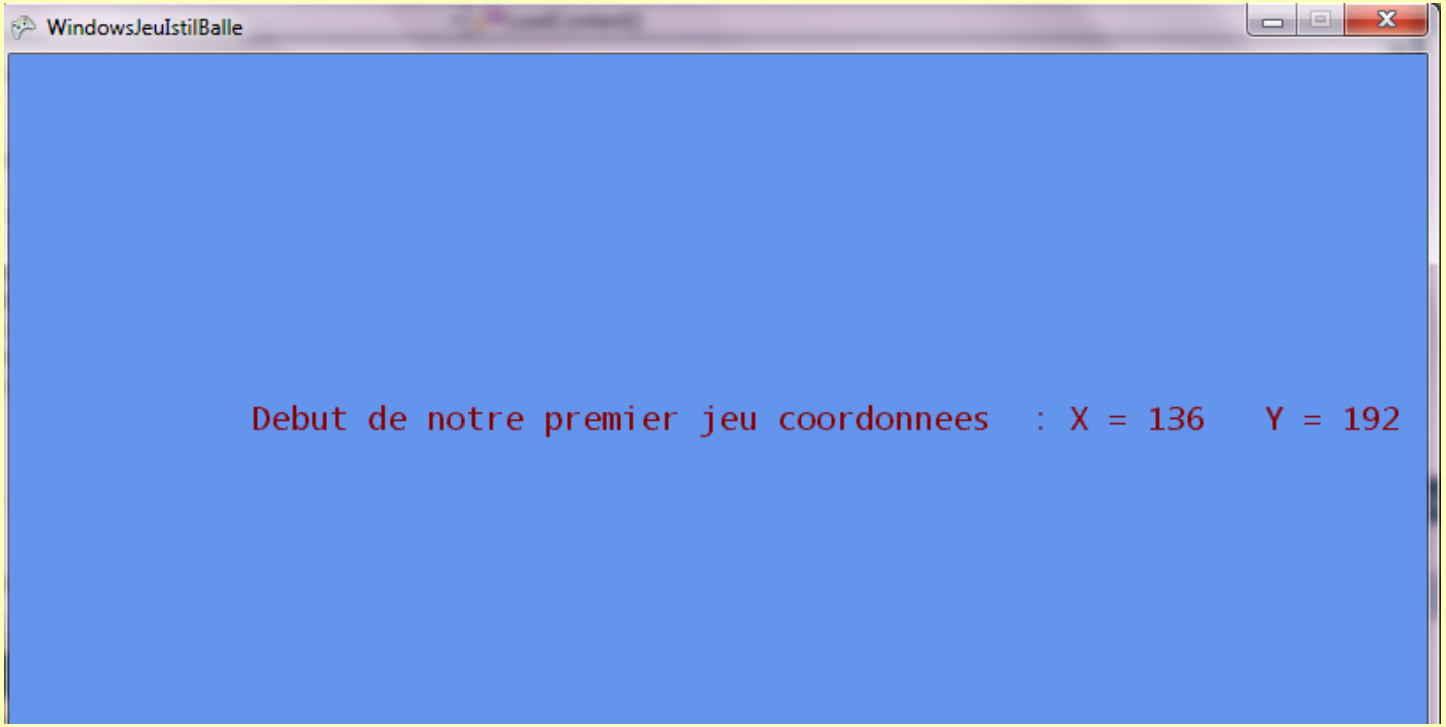


Il faut modifier le Game profile du projet (propriétés) :

XNA

Premier programme : exécution

- *Exécution*



XNA

Balle rebondissante

Christian VIAL

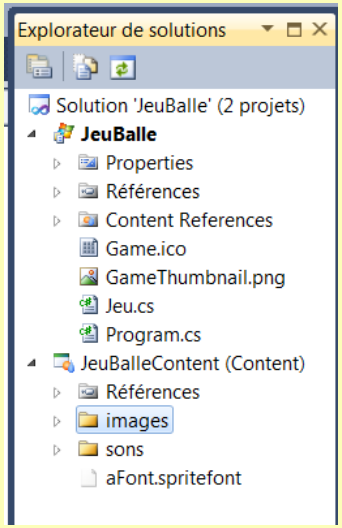
XNA

- **Balle rebondissante**

Développement de l'application

Le but de ce TP est vous initier à la programmation d'un jeu simple qui demande la mise en œuvre des possibilités du framework.
Démarrez un nouveau projet que vous nommerez

JeuBalle



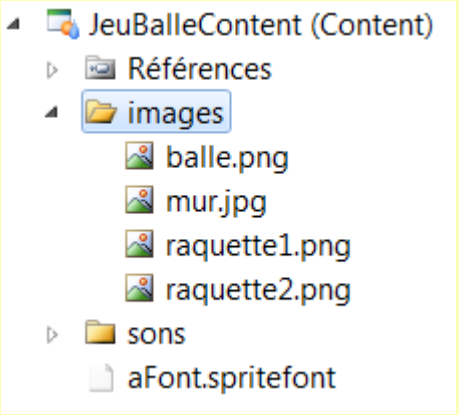
XNA

- **Balle rebondissante**

Affichage de la balle

Dans un premier temps, il faut récupérer une image de balle dans un fichier de type png. Ce fichier sera sauvegardé dans un répertoire de notre jeu nommé **mesingages**
Dans votre répertoire JeuBallelContent situé sous **JeuBalle**
Créez un nouveau répertoire nommé mesimages et ajoutez vos fichiers *.png (balle et barre)

Le répertoire images va apparaître dans votre projet



XNA

- **Balle rebondissante**

Les sprites ou objets animés

On définit un classe pour leur comportement :

- Position
- Taille
- vitesse

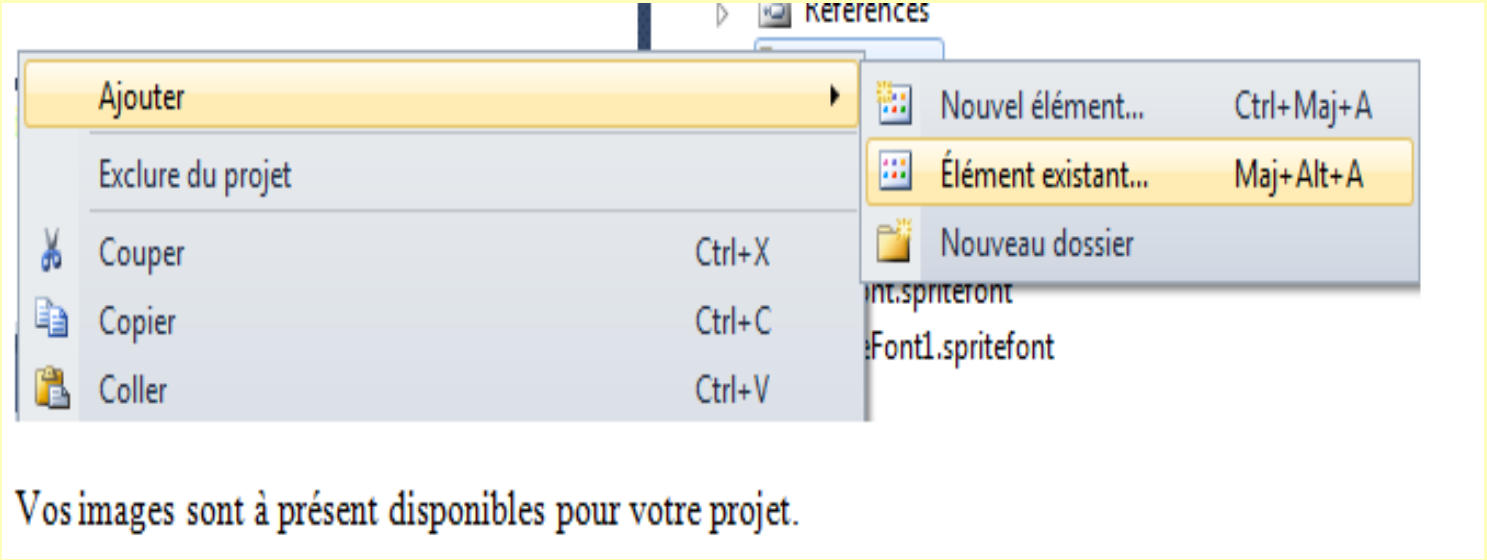
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;

namespace JeuBalle
{
    public class ObjetAnime
    {
        // On définit les propriétés de l'objet animé
        // Certains objets animés auront une vitesse nulle
        private Texture2D _texture;
        private Vector2 _position;
        private Vector2 _size;
        private Vector2 _vitesse;

        public Vector2 Vitesse
        {
            get { return _vitesse; }
            set { _vitesse = value; }
        }

        public Texture2D Texture
        {
            get { return _texture; }
            set { _texture = value; }
        }

        public Vector2 Position
        {
            get { return _position; }
            set { _position = value; }
```

XNA

- **Balle rebondissante**

Affichage de la balle

Ensuite vous devez ajouter (bouton droit sur le répertoire mesimages) les fichiers *.png



XNA

- **Balle rebondissante**

Affichage de la balle

Pour afficher la balle, il faut lui donner une position en 2D et une vitesse de départ sous la forme d'un vecteur . Pour réaliser les opérations liées à ce jeu, nous allons créer une classe que nous nommerons **Balle.cs** . Elle héritera de **DrawableGameComponent.**

Dans cette classe nous ajouterons un objet animé :

```
private ObjetAnime uneballe;
private SoundEffect soundRaquette;
private SoundEffect soundMur;
```

XNA

- **Constructeur de la classe Balle**

```
public Balle(Game game, int x,int y,int z,int w)
    : base(game)
{
    maxX = x;
    maxY = y;
    minX = z;
    minY = w;


   this.Game.Components.Add(this);

}
```

On récupère les coordonnées de l'écran de sortie.

XNA

- **Balle rebondissante**

```
public override void Initialize()
{
    // On définit une vitesse initale
    v_min = new Vector2(3, 2);
    v_max = new Vector2(10, 8);
    this.vitesse_initiale = v_min;
    this.position_initiale.X = maxX / 2;
    this.position_initiale.Y = maxY / 2;
    base.Initialize();
}
```

Dans la méthode Initialize ( ), nous définirons **la position de départ de la balle.**

XNA

- **Balle rebondissante**

Chargement de la balle

Dans le code de la méthode LoadContent(), nous pouvons charger le fichier balle.png dans un ObjetAnime

```
protected override void LoadContent()
{
    Vector2 taille;
    spriteBatch = new SpriteBatch(GraphicsDevice);
    uneballe = new ObjetAnime(Game.Content.Load<Texture2D>(@"images\balle"),
        this.position_initiale, Vector2.Zero , this.vitesse_initiale);

    soundRaquette = Game.Content.Load<SoundEffect>(@"sounds\rebond-raquette");
    soundMur = Game.Content.Load<SoundEffect>(@"sounds\rebond-terre_battue");

     taille.X= uneballe.Texture.Width;
     taille.Y = uneballe.Texture.Height;
     uneballe.Size = taille;
    base.LoadContent();
}
```

XNA

- **Balle rebondissante**

Chargement de la balle

La balle étant chargée, il suffit à présent de l'afficher avec la méthode Draw ( )

```
public override void Draw(GameTime gameTime)
{
    spriteBatch.Begin();
    spriteBatch.Draw(uneballe.Texture, uneballe.Position, Color.Azure);
    spriteBatch.End();
    base.Draw(gameTime);
}
```

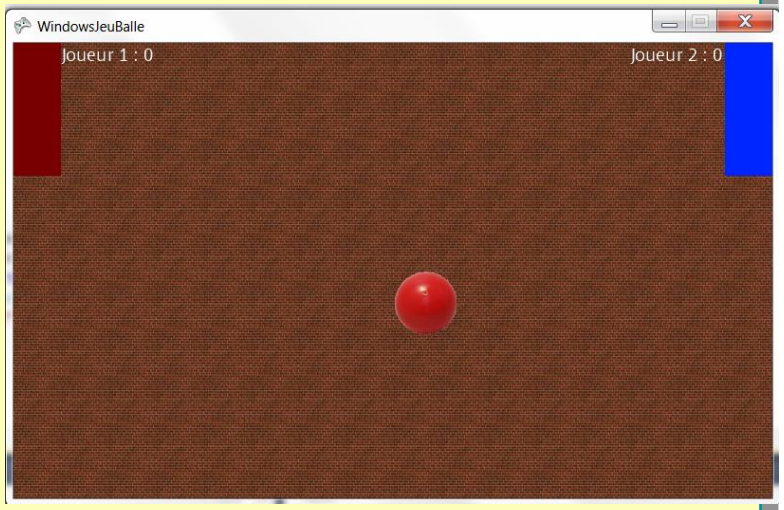
XNA

- **Balle rebondissante**

Appel de balle.cs

Dans la méthode Initialize de notre programme principal (jeu) , nous créons un objet de type Balle

Au lancement du jeu, la balle est affichée au centre de l'écran




```
spriteBatch = new SpriteBatch(GraphicsDevice);
// on définit les coordonnées de l'éran de sortie
maxX = this.GraphicsDevice.Viewport.Width;
maxY = this.GraphicsDevice.Viewport.Height;
minX = 0;
minY = 0;
Balle balle = new Balle(this, maxX, maxY, minX, minY);
```

XNA

- **Balle rebondissante**

Animation de notre balle

Nous allons à présent animer notre balle en lui donnant une vitesse par l'intermédiaire d'un vecteur. Au chargement, sa vitesse initiale est égale à V_min

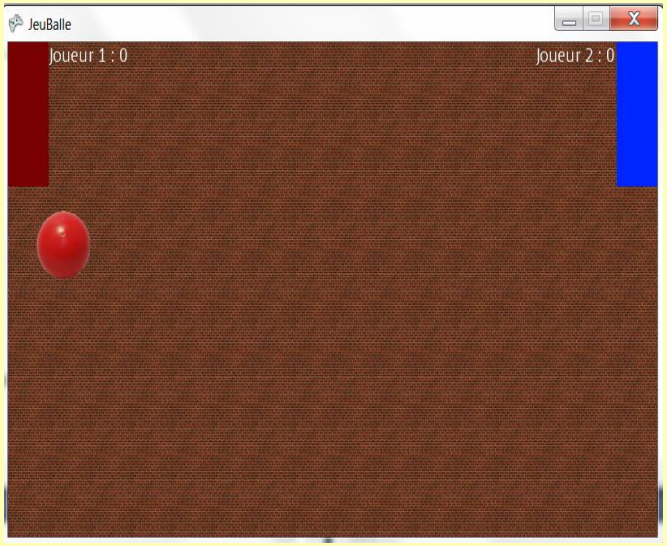
Nous allons créer une méthode nommée **bougeballe** () qui sera appelée par la méthode update () de la classe. Cette méthode va donner de nouvelles positions à la balle en fonction de la vitesse. **Si un mur est rencontré, on repart dans l'autre sens.**

XNA

- **Balle rebondissante**

Animation de notre balle

```
void bougeBalle2()
      {
          Vector2 v;
          Boolean collision_murs = false;
          //Bouger la balle par rapport a la vitesse
          //demarrage();
          // murs droit ou gauche
          uneballe.Position += uneballe.Vitesse;
          if (uneballe.Position.X <= minX
              || uneballe.Position.X + uneballe.Size.X >= maxX)
          {
              v = uneballe.Vitesse;
              v.X *= -1;

              if (v.X < v_max.X)
                  v.X *= 1.1f;
              uneballe.Vitesse = v;
              collision_murs = true;
          }
          if (uneballe.Position.Y + uneballe.Size.Y >= maxY
           || uneballe.Position.Y <= minY)
          {
              v = uneballe.Vitesse;
              v.Y *= -1;

              if (v.Y < v_max.Y)
                  v.Y *= 1.1f;
              uneballe.Vitesse = v;
              collision_murs = true;
          }
          if (collision_murs)
          {
              SoundEffectInstance soundInstMur = soundMur.CreateInstance();
              soundInstMur.Volume = 0.6f;
              soundInstMur.Play();
```



XNA

- **Balle rebondissante**

Construction des raquettes

La construction des raquettes demande la création de deux fichiers *.png que nous nommerons **raquettebleue** et **raquetterouge**. Les deux raquettes seront disposées de part et d'autre de l'écran et seront contrôlées par les touches D et E (raquettebleue) , P et M (raquetterouge).

Les raquettes sont des objets animés comme la balle. Nous définissons une classe raquette

XNA

Gestion des raquettes

```
public class Raquette : Microsoft.Xna.Framework.DrawableGameComponent
{
    SpriteBatch spriteBatch;

    private Joueur joueur;
    private string raquette_filename;
    private ObjetAnime uneraquette;
    private Vector2 position_initiale;

    private int maxY, maxX;
    private int minY;
    private int tailleraquetteX = 50;
    private int tailleraquetteY = 140;

    public const int VITESSE_RAQUETTES = 8;
    private Vector2 vitesse_initiale;

    private BoundingBox bbox;
    private Balle balle;

    public BoundingBox Bbox
    {
        get
        {
            return bbox;
        }
    }

    public ObjetAnime Uneraquette
    {
        get
        {
            return uneraquette;
        }
```

Le déplacement des raquettes est piloté par une variable nommée déplacement

private int deplacement =5;

// déplacement de 5 pixels

XNA

Position des raquettes

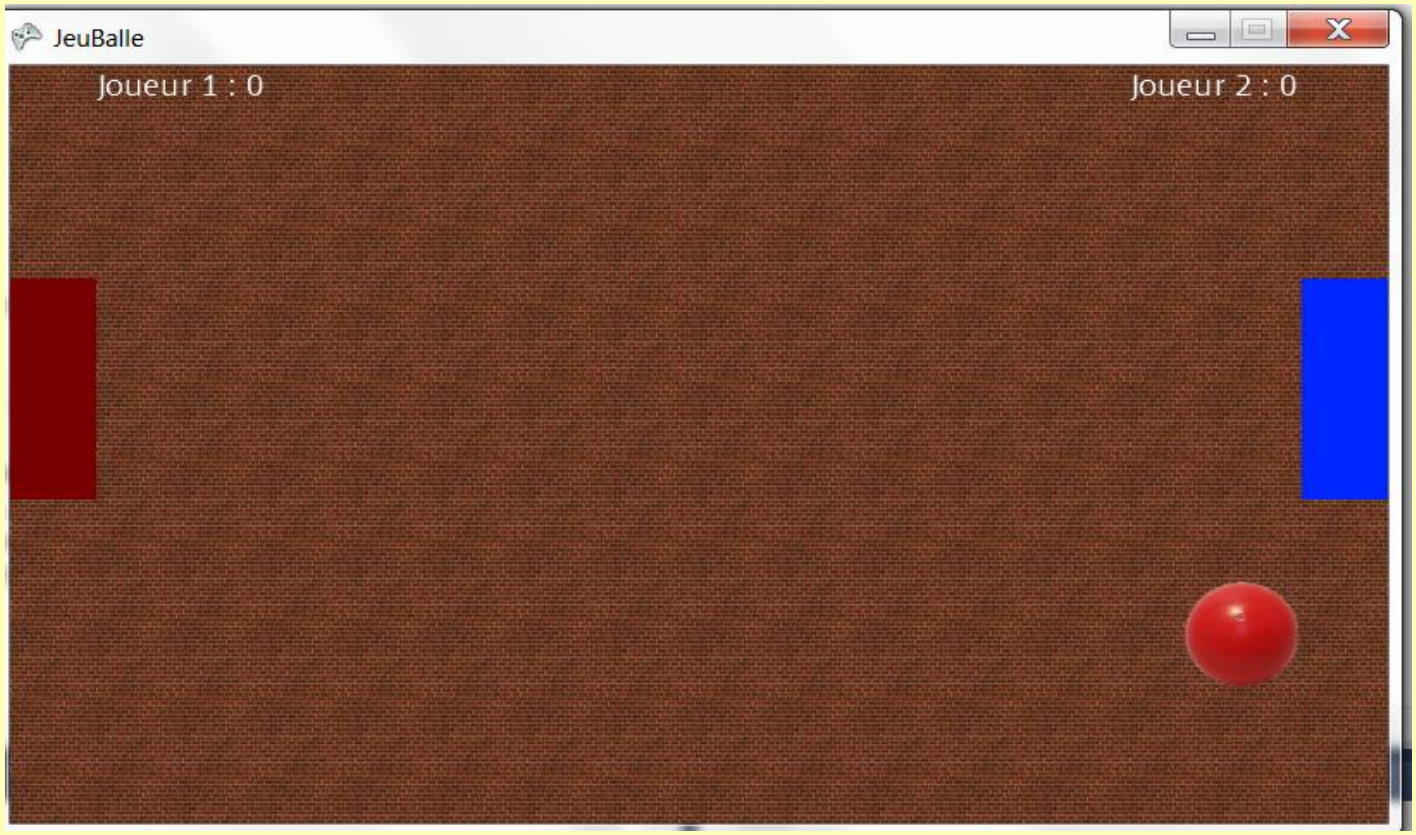
```
// par le constructeur on initialise les propriétés de la raquette
public Raquette(Game game, Joueur joueur, string raquette_filename,
    int vitesse, int tailleho, int tailleverticale)
    : base(game)
{
    this.joueur = joueur;
    this.raquette_filename = raquette_filename;
    this.vitesse_initiale = new Vector2(0, vitesse);
    this.Game.Components.Add(this);
    maxX = tailleho;
    maxY = tailleverticale - tailleraquetteY / 2;
    minY = 0;
}
```

```
// On affiche les raquettes
protected override void LoadContent()
{
    Vector2 taille;
    spriteBatch = new SpriteBatch(GraphicsDevice);

    if (joueur.NoJoueur == 1)
        this.position_initiale = new Vector2(0, (maxY - tailleraquetteY)/2);
    else if (joueur.NoJoueur == 2)
        this.position_initiale = new Vector2(maxX - tailleraquetteX, (maxY - tailleraquetteY)/2);
    else this.position_initiale = new Vector2(0, 0);
    uneraquette = new ObjetAnime(Game.Content.Load<Texture2D>(@"images\" + raquette_filename),
        position_initiale, Vector2.Zero, vitesse_initiale);
    taille.X = uneraquette.Texture.Width;
    taille.Y = uneraquette.Texture.Height;
    uneraquette.Size = taille;
    base.LoadContent();
}
```

XNA

- **Balle rebondissante**

Position des raquettes



XNA

- **Balle rebondissante**

Déplacement des raquettes

Le déplacement des raquettes se fait dans la méthode **update ( ).** Sa gestion se fait sur des touches de clavier. Il faut bien penser à bloquer les raquettes lorsqu'elles arrivent en fin de course.

```
// on récupère l'état du clavier
    KeyboardState keyboard = Keyboard.GetState();

   // si on appuie sur la touche D du clavier
   if (keyboard.IsKeyDown(Keys.D))
       raquettebleuePosition.Y += deplacement; // on déplace la barre vers le
bas  sinon si on appui sur E,
   else if (keyboard.IsKeyDown(Keys.E) )
       raquettebleuePosition.Y -= deplacement; // on monte la barre
```

XNA

- **Contrôle des touches**

Déplacement des raquettes

Le déplacement des raquettes se fait dans la méthode **update ( ).** Sa gestion se fait sur des touches de clavier. Il faut bien penser à bloquer les raquettes lorsqu'elles arrivent en fin de course.

```
// on récupère l'état du clavier
    KeyboardState keyboard = Keyboard.GetState();

   // si  le joueur 1  appuie sur la touche D du clavier
   if (Controls.CheckActionDown(joueur.NoJoueur))
       {
              ……..
      raquettebleuePosition.Y += deplacement;
    // on déplace la barre vers le bas  sinon si on appui sur E,
```

XNA

- **Balle rebondissante**

Déplacement des raquettes

Le contrôle des touches passe par la construction d'une classe Controls

```
/**
 * Cette classe sert d'abstraction pour les contrôles des joueurs
 * Chaque contrôle (touches clavier, boutons souris, ...) correspond à une action réalisable dans le jeu
 * Le changement d'un contrôle (ex : touche D -> touche E) entraîne la modification d'une seule classe, Controls
 */
class Controls
{
    public const Keys JOUEUR1_UP = Keys.E;
    public const Keys JOUEUR1_DOWN = Keys.D;
    public const Keys JOUEUR2_UP = Keys.P;
    public const Keys JOUEUR2_DOWN = Keys.M;

    // Vérifie si le joueur passé en paramètre a effectué l'action "monter la raquette"
    public static Boolean CheckActionUp(int joueur)
    {
        Boolean checkActionUp = false;
        KeyboardState keyboard = Keyboard.GetState();
        switch (joueur)
        {
            case 1:
                checkActionUp = keyboard.IsKeyDown(JOUEUR1_UP);
                break;
            case 2:
                checkActionUp = keyboard.IsKeyDown(JOUEUR2_UP);
                break;
            default: break;
        }
```

XNA

- **Balle rebondissante**

Collision de la balle avec une raquette

Jusqu'à présent, la balle rebondit bien sur les bords de l'écran mais traverse les raquettes. Il faut gérer ce problème.

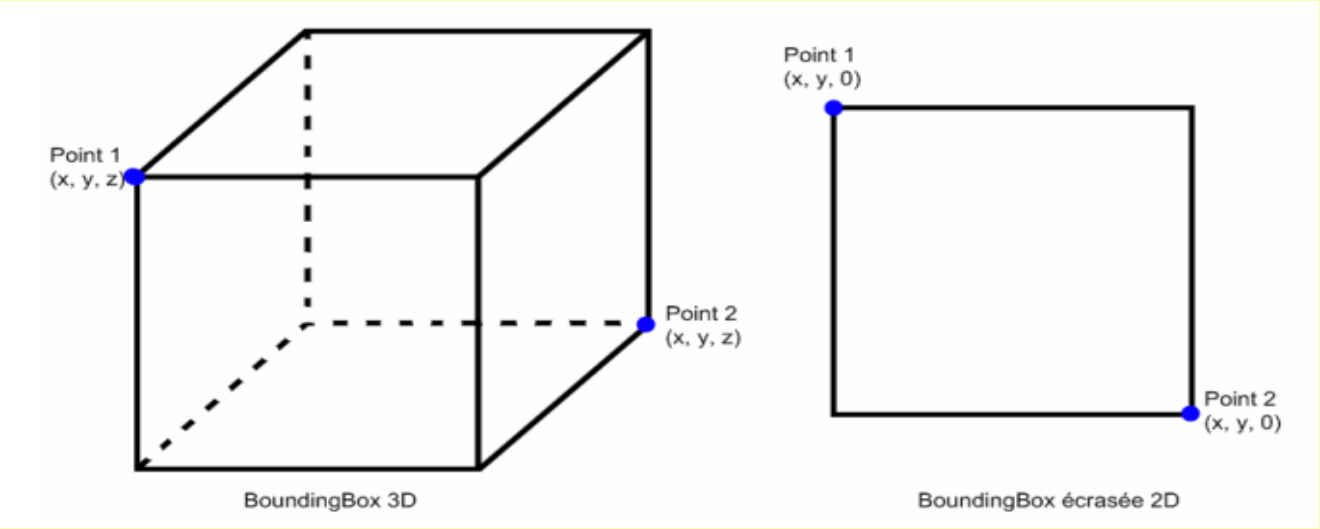
// source Microsoft.com Xna

Pour gérer les collisions entres deux volumes (et donc des rectangles), XNA propose des classes toutes faites pour gérer cela, **comme la classe BoundingBox** que nous allons utiliser. Ces boîtes sont normalement faites pour des collisions 3D, mais **en faisant abstraction de la composante sur l'axe Z,** on peut parfaitement gérer des collisions 2D.

XNA

- **Balle rebondissante**

Collision de la balle avec une raquette




```
// gestion des collisions
// boite pour la balle
BoundingBox balleBox = new BoundingBox(
        new Vector3(X1,Y1, 0),
         // on définit le premier point, avec la composante Z à 0 puisque
         //l'on fait de la 2D
        new Vector3(X2,Y2,, 0));
```

XNA

- **Balle rebondissante**

On définit une classe nommée Moteur2D

```
/*
 * Cette classe s'occupe de gérer tous les problèmes liés à un environnement 2D
 */
class Moteur2D
{
    // Positions relatives
    public const int CROISEMENT = -1; // Si les 2 objets se croisent
    public const int EN_DESSOUS = 0;
    public const int AU_DESSUS = 1;
    public const int A_DROITE = 2;
    public const int A_GAUCHE = 3;

    // Elements de l'environnement
    public const int MUR_BAS = 0;
    public const int MUR_HAUT = 1;

    // les parties gauche et droite sont considérées comme des "murs" même si la balle les traverse
    public const int MUR_DROIT = 2;
    public const int MUR_GAUCHE = 3;

    public static Boolean testCollision(Object obj1, BoundingBox obj2)
    {
        // On réalise le mvt dans une bounding box de test et on renvoie le résultat de l'intersection avec l'objet obj
        Vector3 upperLeftCorner = new Vector3(0, 0, 0);
        Vector3 bottomRightCorner = new Vector3(0, 0, 0);

        if (obj1 is Balle)
```

XNA

- **Balle rebondissante**

On définit une classe nommée Moteur2D

```
/// <summary>
/// Donne la position d'un objet par rapport à un autre
/// </summary>
/// <param name='obj1'>
///     int[] - un tableau contenant des informations sur l'objet à localiser :
///     position en X, position en Y, largeur, hauteur
/// </param>
/// <param name='obj2'>
///     int[] - un tableau de nombres décimaux contenant des informations sur l'objet de référence dans
///     cet ordre :
///     position en X, position en Y, largeur, hauteur
/// </param>
/// <returns>
///     Retourne un vecteur contenant la comparaison sur l'axe des x et celle sur l'axe des y
///     Les positions sont représentées par les constantes définies dans MoteurJeu suivantes :
///       - EN_DESSOUS
///       - AU_DESSOUS
///       - A_DROITE
///       - A_GAUCHE
/// </returns>
public static int[] getRelativePosition(float[] obj, float[] reference)
{
    int[] positionRel = { -1, -1 };

    // axe des X
    if (obj[0] >= reference[0] + reference[2])
        positionRel[0] = A_DROITE;
    else if (obj[0] + obj[2] <= reference[0])
        positionRel[0] = A_GAUCHE;
```

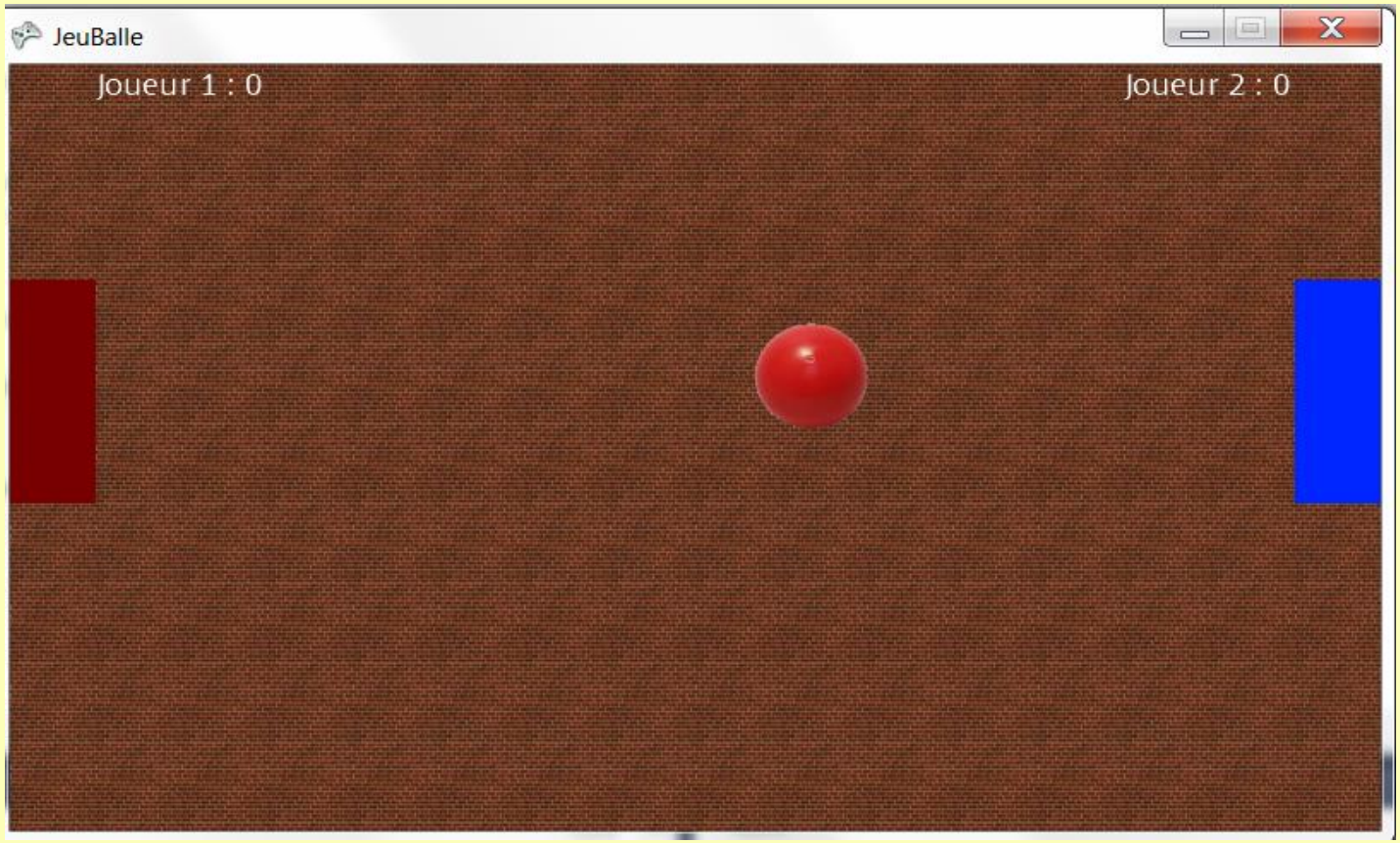
XNA

- **Balle rebondissante**

Collision de la balle avec une raquette

```
public static Boolean testCollision(Object obj1, BoundingBox obj2)
{
    // On réalise le mvt dans une bounding box de test et on renvoie le résultat de l'intersection avec l'objet obj
    Vector3 upperLeftCorner = new Vector3(0, 0, 0);
    Vector3 bottomRightCorner = new Vector3(0, 0, 0);

    if (obj1 is Balle)
    {
        Balle balle = (Balle)obj1;
        upperLeftCorner.X = balle.Uneballe.Position.X + balle.Uneballe.Vitesse.X;
        upperLeftCorner.Y = balle.Uneballe.Position.Y + balle.Uneballe.Vitesse.Y;
        bottomRightCorner.X = balle.Uneballe.Position.X + balle.Uneballe.Size.X + balle.Uneballe.Vitesse.X;
        bottomRightCorner.Y = balle.Uneballe.Position.Y + balle.Uneballe.Size.Y + balle.Uneballe.Vitesse.Y;
    }
    else if (obj1 is Raquette)
    {
        Raquette raquette = (Raquette)obj1;
        upperLeftCorner.X = raquette.Uneraquette.Position.X;
        upperLeftCorner.Y = raquette.Uneraquette.Position.Y + raquette.Uneraquette.Vitesse.Y;
        bottomRightCorner.X = raquette.Uneraquette.Position.X + raquette.Uneraquette.Size.X;
        bottomRightCorner.Y = raquette.Uneraquette.Position.Y + raquette.Uneraquette.Size.Y + raquette.Uneraquette.Vitesse.Y;
    }

    BoundingBox bbox_test = new BoundingBox(upperLeftCorner, bottomRightCorner);
```

XNA *balle rebondissante*



XNA

Pacman

XNA

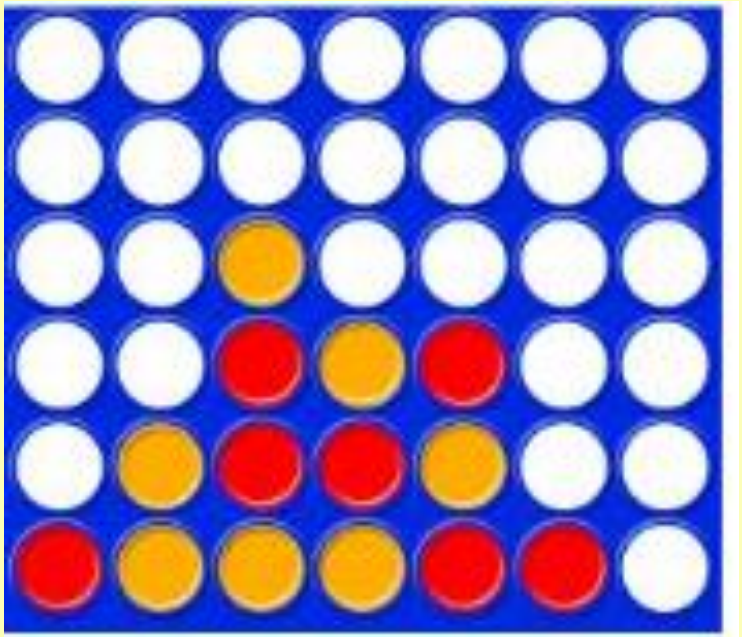
XNA

Pacman

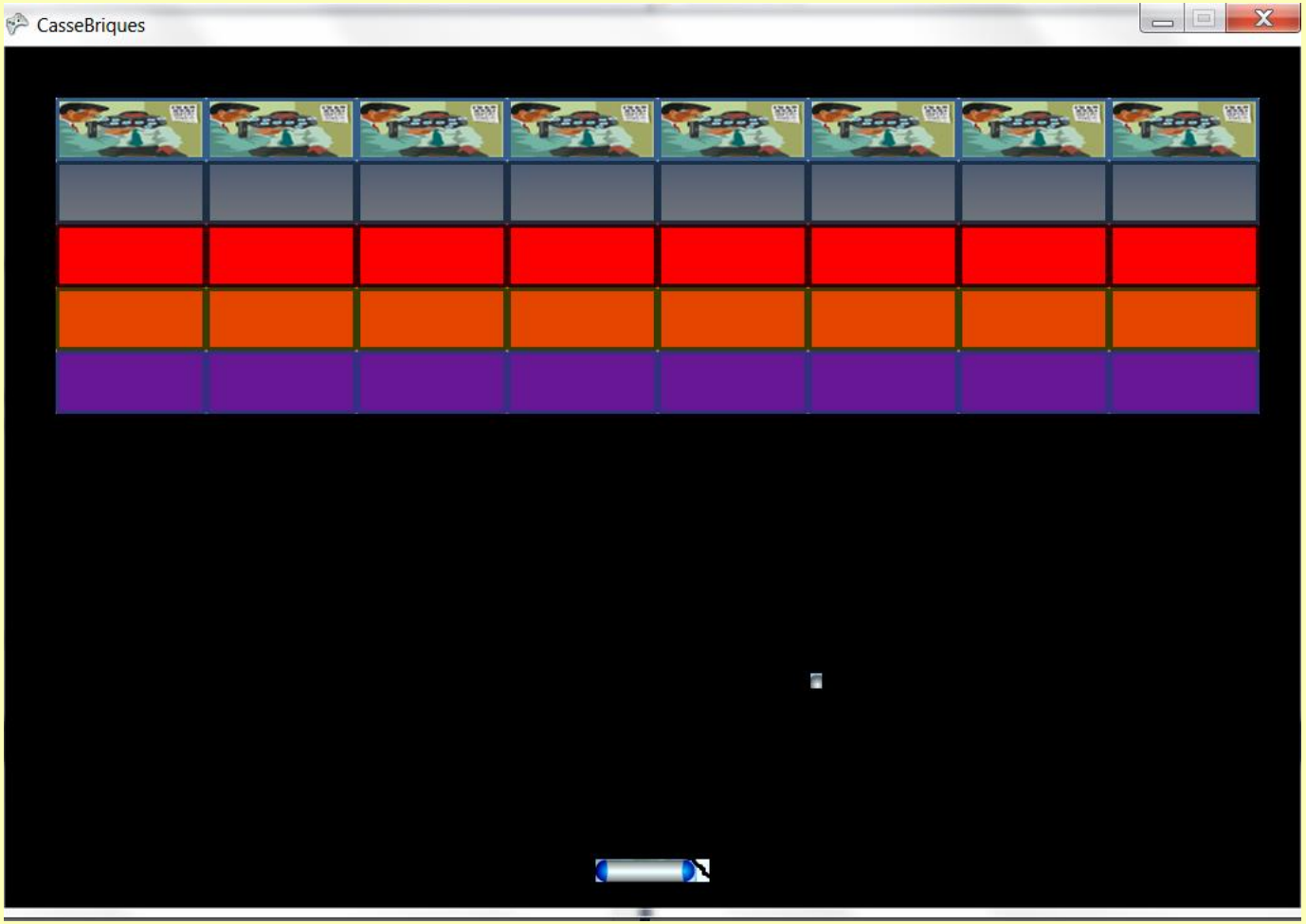
```
{
    map = new byte[VX, VY]{
        {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
        {0, 4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 4, 0},
        {0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0},
        {0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0},
        {0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0},
        {0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0},
        {0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0},
        {0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0},
        {0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0},
        {0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 3, 3, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 2, 2, 2, 2, 2, 2, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0},
        {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 2, 2, 2, 2, 2, 2, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1},
        {0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 2, 2, 2, 2, 2, 2, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0},
        {0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0},
        {0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0},
        {0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0},
        {0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0},
        {0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0},
        {0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0},
        {0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0},
        {0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0},
        {0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0},
        {0, 4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 4, 0},
        {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
    };
```

XNA

Puissance 4



XNA *Casse briques*



XNA

- **Pacman**

Développement de l'application

Le but de ce TP est vous initier à la programmation d'un jeu plus complexe qui nécessite :

- une analyse de conception
- l'écriture d'algorithmes

Ce TP s'adresse aux étudiants intéressés par la programmation d'un jeu video.