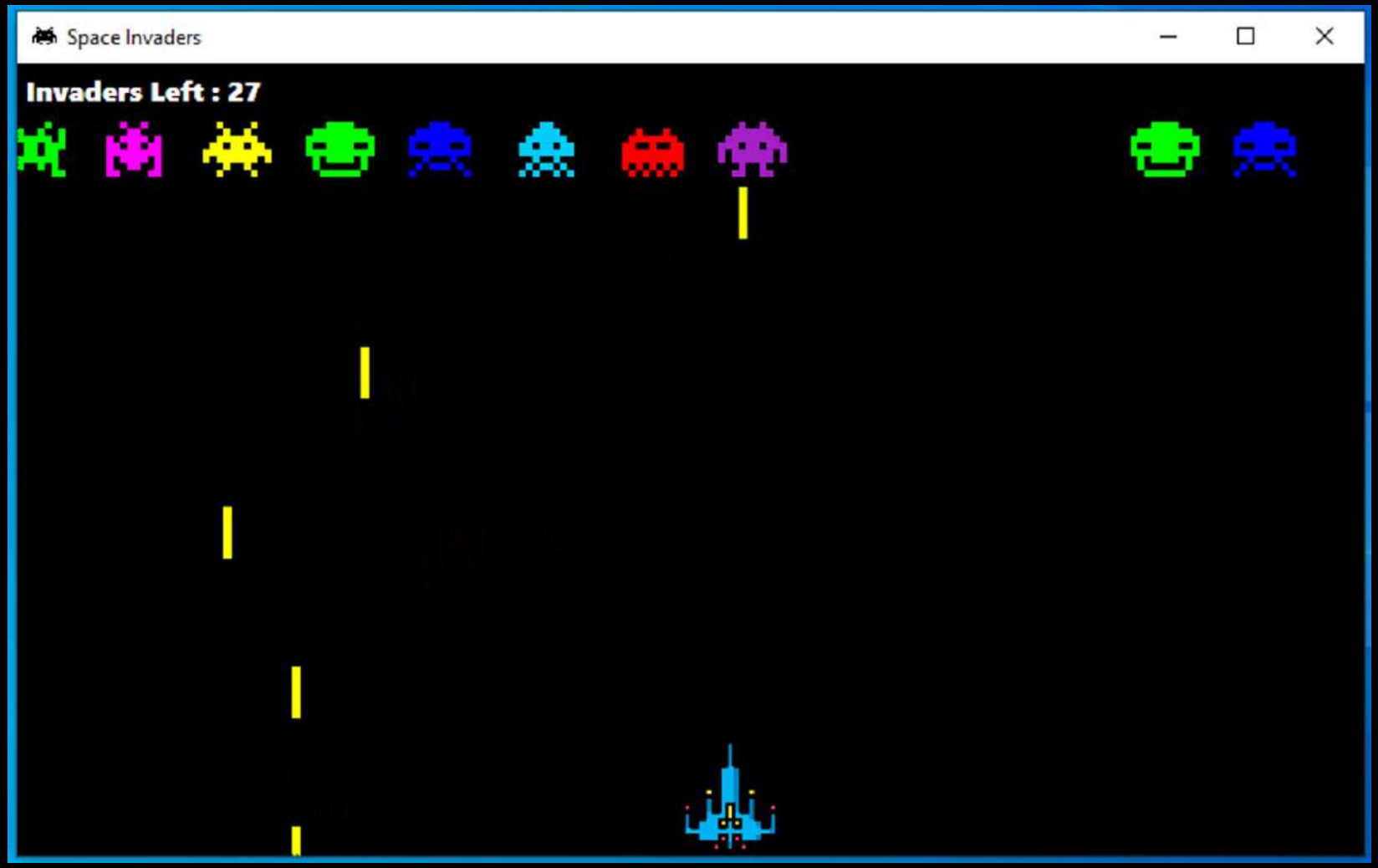

WPF
Créer des jeux vidéo

OBJECTIFS

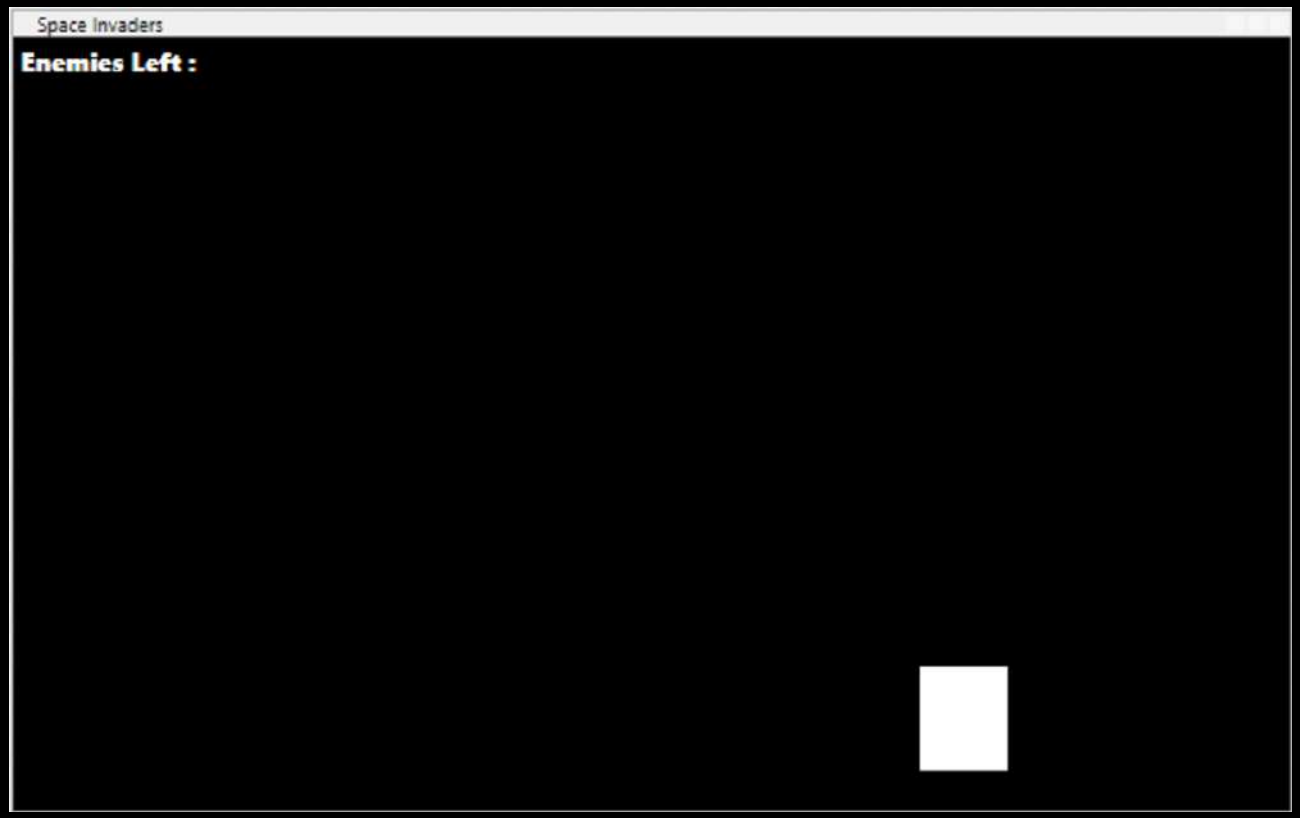
- Créer un jeu de style Space Invaders en WPF
- Utilisez un timer pour créer le moteur de jeu principal avec toute la logique et l’instantiation des objets
- Créer des méthodes pour améliorer l'efficacité du programme
- Contrôler le clavier pour se déplacer de gauche à droite et tirer sur les ennemis
- Déterminer comment gagner ou perdre la partie
- Changer la vitesse de l'ennemi pendant l’exécution du jeu
- Tirer sur le joueur avec des balles aléatoires

IHM : SPACE INVADER



REALISATION

- Création de l'IHM XAML



XAML - IHM

```
<Window x:Class="space_invaders.MainWindow"
…
        FocusManager.FocusedElement="{Binding ElementName=myCanvas}"
        Title="Space Invaders" Height="500" Width="800"
        WindowStartupLocation="CenterScreen" >

        <Canvas Name="myCanvas" Background="Black" Focusable="True"
                KeyDown="CanvasKeyIsDown" KeyUp="CanvasKeyIsUp">
            <Label Foreground="White" Name="enemiesLeft" FontSize="16" FontWeight="ExtraBold">
                Enemies Left :
            </Label>
            <Rectangle Name="player1" Fill="White" Height="65" Width="55"
                Canvas.Left= "370" Canvas.Top="394" />
        </Canvas>
</Window>
```

Focus automatique sur le canvas myCanvas au lancement de la fenêtre (permet de gérer immédiatement le clavier)

Canvas myCanvas focusable à true pour qu'elle soit toujours activée lorsque le jeu est chargé

Gestion des touches dans le canvas

Ce rectangle est l'objet joueur

BEHIND – METHODES

- Évenement touche enfoncée
- Évenement touche relâchée
- Création des ennemis
- Création des tirs ennemis
- Boucle de jeu (Timer)

```
private void Canvas_KeyIsDown(object sender, KeyEventArgs e)
{}
private void Canvas_KeyIsUp(object sender, KeyEventArgs e)
{}
private void MakeEnemies(int limit)
{}
private void EnemyBulletMaker(double x, double y)
{}
private void GameEngine(object sender, EventArgs e)
{}
```

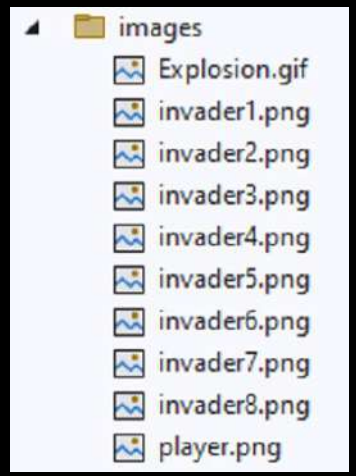
VARIABLES PRIVEES

```
// booléens pour aller à gauche et à droite
private bool goLeft, goRight = false ;
// liste des éléments rectangles
private List < Rectangle > itemsToRemove = new List < Rectangle >() ;
// entier nous permettant de charger les images des ennemis
private int enemyImages = 0 ;
// Timer du tir ennemi
private int bulletTimer ;
// limite de la fréquence du Timer du tir ennemi
private int bulletTimerLimit = 90 ;
// enregistre le nombre total d'ennemis
private int totalEnemies;
// crée une nouvelle instance de la classe dispatch timer
private DispatcherTimer dispatcherTimer = new DispatcherTimer () ;
// classe de pinceau d'image que nous utiliserons comme image du joueur appelée skin du joueur
private ImageBrush playerSkin = new ImageBrush () ;
// vitesse par défaut de l'ennemi
private int enemySpeed = 6 ;
// vitesse du joueur
private int playerSpeed = 10 ;
// vitesse du tir du joueur
private int bulletPlayerSpeed = 20 ;
// vitesse du tir ennemi
private int bulletEnemySpeed = 10 ;
```

liste contenant toutes les formes de rectangle du jeu, par exemple les tirs et les ennemis. On les supprimera lorsqu'elles ne seront plus nécessaires

CHARGEMENT INITIAL

```
InitializeComponent();
// configure le Timer et les événements
// lie le timer du répartiteur à un événement appelé moteur de jeu gameengine
dispatcherTimer.Tick += GameEngine;
// rafraissement toutes les 16 milliseconds
dispatcherTimer.Interval = TimeSpan.FromMilliseconds(16);
// lancement du timer
dispatcherTimer.Start();
// chargement de l'image du joueur
playerSkin.ImageSource = new BitmapImage(new
Uri(AppDomain.CurrentDomain.BaseDirectory +"images/player.png"));
// assignement de skin du joueur au rectangle associé
player1.Fill = playerSkin;
// appel à la méthode pour créer 30 ennemis
MakeEnemies(30);
```



GESTION CLAVIER

```
private void CanvasKeyIsDown(object sender, KeyEventArgs e)
{
// on gère les booléens gauche et droite en fonction de l'appui de la touche
      if (e.Key == Key.Left)
      {
            goLeft = true;
      }
      if (e.Key == Key.Right)
      {
            goRight = true;
      }
}
private void CanvasKeyIsUp(object sender, KeyEventArgs e)
{
// on gère les booléens gauche et droite en fonction du relâchement de la touche
      if (e.Key == Key.Left)
      {
            goLeft = false;
      }
      if (e.Key == Key.Right)
      {
            goRight = false;
      }
}
```

GESTION DU TIR DU JOUEUR

```
// test de l'appui sur la barre espace
if (e.Key == Key.Space)
{
    // on vide la liste des items
    itemsToRemove.Clear();
    // création un nouveau tir
    Rectangle newBullet = new Rectangle
    {
        Tag = "bulletPlayer", //permet de tagger les rectangles
        Height = 20,
        Width = 5,
        Fill = Brushes.White,
        Stroke = Brushes.Red
    };
    // on place le tir à l'endroit du joueur
    Canvas.SetTop(newBullet, Canvas.GetTop(player1) - newBullet.Height);
    Canvas.SetLeft(newBullet, Canvas.GetLeft(player1) + player1.Width / 2);
    // on place le tir dans le canvas
    myCanvas.Children.Add(newBullet);
}
```

RIPOSTE DES ENNEMIS

```
private void EnemyBulletMaker(double x, double y)
{
// création des tirs ennemies tirant vers l'objet joueur
// x et y position du tir

    Rectangle newEnemyBullet = new Rectangle
    {
        Tag =  "enemyBullet",
        Height = 40,
        Width = 15,
        Fill = Brushes.Yellow,
        Stroke = Brushes.Black,
        StrokeThickness = 5
    };
    Canvas.SetTop(newEnemyBullet, y);
    Canvas.SetLeft(newEnemyBullet, x);
    myCanvas.Children.Add(newEnemyBullet);
}
```

CREATION DES ENNEMIS

```
private void MakeEnemies(int limit)
{
      int left = 0;
      // on conserve le max d'ennemis
      totalEnemies = limit;
      for (int i = 0; i < limit; i++)
      {
            ImageBrush enemySkin = new ImageBrush();
            Rectangle newEnemy = new Rectangle
                  {
                  Tag = "enemy",
                  Height = 45,
                  Width = 45,
                  Fill = enemySkin,
                  };
            Canvas.SetTop(newEnemy, 30);
            Canvas.SetLeft(newEnemy, left);
            myCanvas.Children.Add(newEnemy);
            left -= 60;
            // incrémente les images des ennemis (max 8)
            enemyImages++;
            if (enemyImages > 8)
                  enemyImages = 1;

            enemySkin.ImageSource = new BitmapImage(new Uri(AppDomain.CurrentDomain.BaseDirectory +
            "images/invader"+ enemyImages +".gif"));
      }
}
```

GAME ENGINE– BOUCLE DE JEU

```
private void GameEngine(object sender, EventArgs e)
{
// création d'un rectangle joueur pour la détection de collision
Rect player = new Rect(Canvas.GetLeft(player1), Canvas.GetTop(player1),
player1.Width, player1.Height);
// complète l'affichage du nombre d'ennemis tués
enemiesLeft.Content = "Invaders Left : " + totalEnemies;
// déplacement à gauche et droite de vitessePlayer avec vérification des
limites de fenêtre gauche et droite
if (goLeft && Canvas.GetLeft(player1) > 0)
    {
    Canvas.SetLeft(player1, Canvas.GetLeft(player1) - playerSpeed);
    }
else if (goRight && Canvas.GetLeft(player1) + player1.Width <
Application.Current.MainWindow.Width)
    {
    Canvas.SetLeft(player1, Canvas.GetLeft(player1) + playerSpeed);
    }
```

GAME ENGINE– BOUCLE DE JEU

```
private void GameEngine(object sender, EventArgs e)
{
// suite
// décrémente de 2 du timer de 16 millisecondes du tir ennemi. Détermine donc la cadence de tir
bulletTimer -= 2;
// arrivé à 0 on place le tir ennemi en face du joueur
if (bulletTimer < 0)
{
        EnemyBulletMaker((Canvas.GetLeft(player1) + player1.Width/2 ), 10);
        // remise au max de la fréquence du tir ennemi.
        bulletTimer = bulletTimerLimit;
}
// on augmente la vitesse des ennemis quand ils sont inférieurs à 10
if (totalEnemies < 10)
    enemySpeed = 20;
```

GAME ENGINE– BOUCLE DE JEU

```
private void GameEngine(object sender, EventArgs e)
{ // suite
// parcours de la liste des rectangles d'objets du canvas
foreach (Rectangle x in myCanvas.Children.OfType<Rectangle>())
     {
     // traitement du rectangle de type tir joueur → déplacement
     if (x is Rectangle && (string)x.Tag == "bulletPlayer")
          {
          // si c'est un tir joueur on le déplace vers le haut
          Canvas.SetTop(x, Canvas.GetTop(x) - bulletPlayerSpeed);
          // création d'un tir joueur à base d'un rectangle Rect (nécessaire pour la collision)
          Rect bullet = new Rect(Canvas.GetLeft(x), Canvas.GetTop(x), x.Width, x.Height);
          // on vérifie que le tir a quitté le le haut du canvas (pas de collision avec un
          ennemie)
               if (Canvas.GetTop(x) < 10)
               {
               // si c'est le cas on l'ajoute à la liste des éléments à supprimer
               itemsToRemove.Add(x);
               }
```

GAME ENGINE– BOUCLE DE JEU

```
private void GameEngine(object sender, EventArgs e)
{ // suite
  // Traitement de la collision du tir
      foreach (var y in myCanvas.Children.OfType<Rectangle>())
      {
      // si le rectangle est un ennemi
      if (y is Rectangle && (string)y.Tag == "enemy")
            {
            // création d'un rectangle correspondant à l'ennemi
            Rect enemy = new Rect(Canvas.GetLeft(y), Canvas.GetTop(y), y.Width, y.Height);
            // on vérifie la collision
            // appel à la méthode IntersectsWith pour détecter la collision
            if (bullet.IntersectsWith(enemy))
            {
            // on ajoute l'ennemi de la liste à supprimer eton  décrémente le nombre d'ennemis
            itemsToRemove.Add(x);
            itemsToRemove.Add(y);
            totalEnemies -= 1;
            }
      }
  }
```

GAME ENGINE– BOUCLE DE JEU

```
private void GameEngine(object sender, EventArgs e)
{ // suite
     // retour dans la boucle de jeu. On va déplacer les ennemis
     // si on a un ennemi
     if (x is Rectangle && (string)x.Tag == "enemy")
     {
     // On le déplace vers la droite selon enemySpeed
     Canvas.SetLeft(x, Canvas.GetLeft(x) + enemySpeed);
     // si l'ennemi sort du canvas on le place complètement à gauche en le décalant vers le bas
     if ( Canvas.GetLeft(x) > Application.Current.MainWindow.Width + x.ActualWidth )
     {
          Canvas.SetLeft(x, -x.ActualWidth );
          Canvas.SetTop(x, Canvas.GetTop(x) + (x.Height + 10));
     }
     // vérification de la collision avec le joueur
     Rect enemy = new Rect(Canvas.GetLeft(x), Canvas.GetTop(x), x.Width, x.Height);
     if (player.IntersectsWith(enemy))
          {
          // collision avec le joueur et fin de la partie
          dispatcherTimer.Stop();
          MessageBox.Show("Perdu","Fin de partie",MessageBoxButton.OK, MessageBoxImage.Stop);
          }
     }
```

GAME ENGINE– BOUCLE DE JEU

```
private void GameEngine(object sender, EventArgs e)
{ // suite
// retour dans la boucle de jeu. On va vérifier les tirs ennemis
// test si tir ennemi
if (x is Rectangle && (string)x.Tag == "enemyBullet")
     {
     // déplacement vers le bas avec bulletEnemySpeed pixels
     Canvas.SetTop(x, Canvas.GetTop(x) + bulletEnemySpeed);
     // si le tir sort de l'écran on l'ajoute à la liste à supprimer
     if (Canvas.GetTop(x) > ActualHeight + x.ActualHeight )
          itemsToRemove.Add(x);
     // on conserve le tir
     Rect enemyBullets = new Rect(Canvas.GetLeft(x), Canvas.GetTop(x), x.Width, x.Height);
     // détection de collision entre le joueur et le tir ennemi
     if (enemyBullets.IntersectsWith(player))
          {
          // arrêt du timer et fin du jeu
          dispatcherTimer.Stop();
          MessageBox.Show("Perdu","Fin de partie", MessageBoxButton.OK, MessageBoxImage.Stop);
          }
     }
}
```

GAME ENGINE– BOUCLE DE JEU

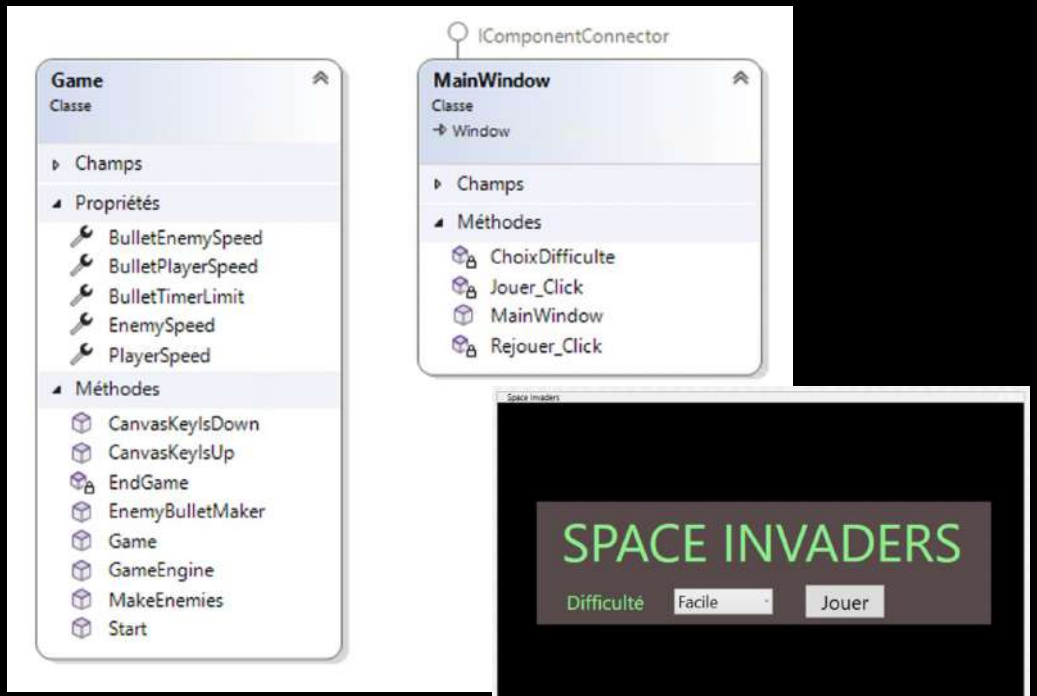
```
private void GameEngine(object sender, EventArgs e)
{ // suite
    // on vide la liste des objets à détruire (tirs et ennemis)
    foreach (Rectangle y in itemsToRemove)
    {
    // on les enlève du canvas
    myCanvas.Children.Remove(y);
    }
    // si le total d'ennemis est à zéro on gagne
    if (totalEnemies < 1)
    {
    // on arrête le timer et on affiche gagné
    dispatcherTimer.Stop();
    MessageBox.Show("Gagné !!","Fin de partie", MessageBoxButton.OK,
    MessageBoxImage.Exclamation);

    }
}
```

AMELIORATIONS

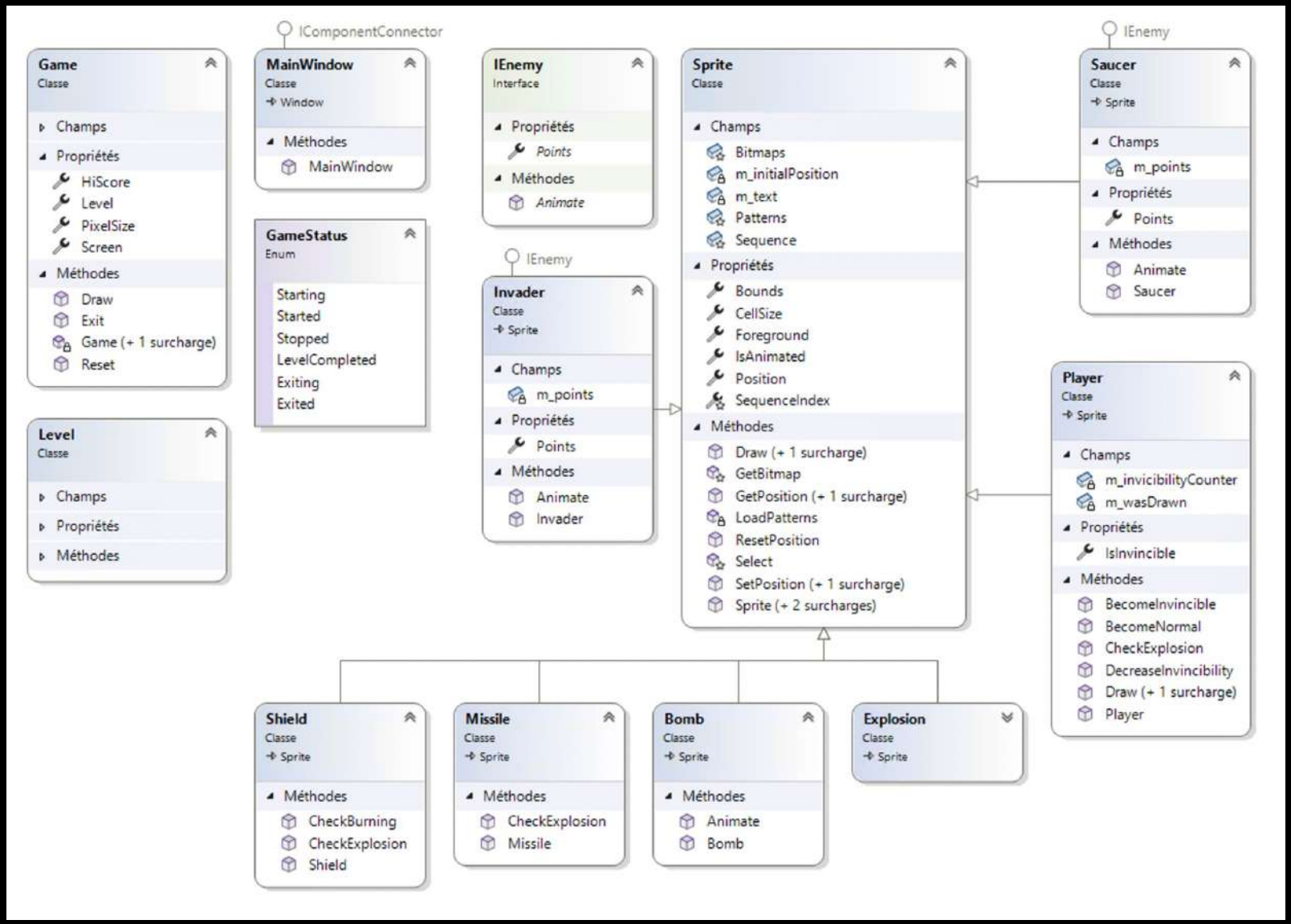
- Simplifier GameEngine (méthodes)
- Mettre en pause (simple avec 2 touches)
- Affichage dynamique dans le canvas (pause, perdu, gagné …)
- Animations, Explosions (Sprite animé ou Gif animé)
- Rejouer
- Highscores
- Bunkers
- …

- Classe Game
- MainWindow.cs
  le contrôleur de jeu
- XAML l'IHM



PROLONGEMENT

- Pour un clone parfait, l'architecture est plus complexe :



PROLONGEMENT