

MonoGame

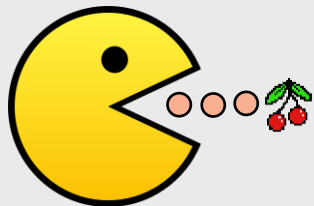
Les sprites

Développement de jeux
IFM25912

Pier-Luc Bonneville

Copyright © 2021

Basé sur le cours de Programmation de jeu (20884 IFM) conçu par Marco Lavoie
Copyright © 2010–2020



Licence d'exploitation

© Marco Lavoie, 2010-2020. Tous droits réservés.

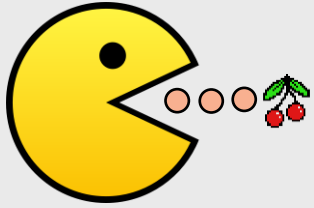
© Pier-Luc Bonneville, 2021. Tous droits réservés.

L'utilisation de ce matériel pédagogique (présentations, code source et autres) avec ou sans modifications, est permise en autant que les conditions suivantes soient respectées:

1. La diffusion du matériel doit se limiter à un intranet dont l'accès est limité aux étudiants inscrits à un cours exploitant le dit matériel. **IL EST STRICTEMENT INTERDIT DE DIFFUSER CE MATÉRIEL LIBREMENT SUR INTERNET.**
2. La redistribution des présentations contenues dans le matériel pédagogique est autorisée uniquement en format Acrobat PDF et sous restrictions stipulées à la condition #1. Le code source contenu dans le matériel pédagogique peut cependant être redistribué sous sa forme originale, en autant que la condition #1 soit également respectée.
3. Le matériel diffusé ou redistribué doit contenir intégralement la mention de droits d'auteur ci-dessus, la notice présente ainsi que la décharge ci-dessous.

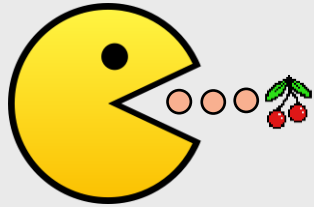
CE MATÉRIEL PÉDAGOGIQUE EST DISTRIBUÉ "TEL QUEL" PAR L'AUTEUR, SANS AUCUNE GARANTIE EXPLICITE OU IMPLICITE. L'AUTEUR NE PEUT EN AUCUNE CIRCONSTANCE ÊTRE TENU RESPONSABLE DE DOMMAGES DIRECTS, INDIRECTS, CIRCONSTANTIELS OU EXEMPLAIRES. TOUTE VIOLATION DE DROITS D'AUTEUR OCCASIONNÉ PAR L'UTILISATION DE CE MATÉRIEL PÉDAGOGIQUE EST PRIS EN CHARGE PAR L'UTILISATEUR DU DIT MATÉRIEL.

En utilisant ce matériel pédagogique, vous acceptez implicitement les conditions et la décharge exprimés ci-dessus.



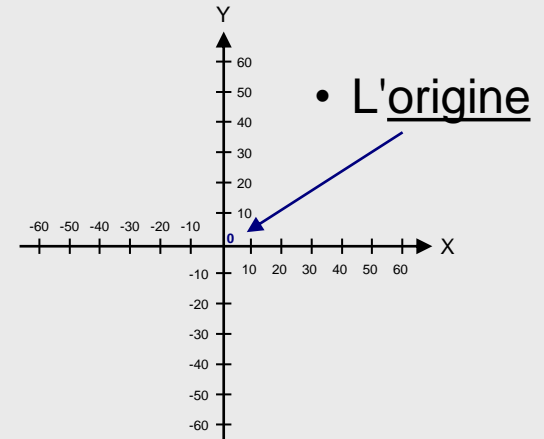
Introduction

- La plupart des jeux 2D sont basés sur les *sprites*
 - Images de type *bitmap* qu'on peut manipuler
 - Déplacer, modifier, redimensionner, multiplier, etc.
 - Offrent une qualité d'image difficile à atteindre autrement
- À l'opposé, les jeux 3D exploitent principalement des objets vectoriels
 - Qualité visuelle moindre, mais essentiel à la représentation de la "profondeur" du 3D

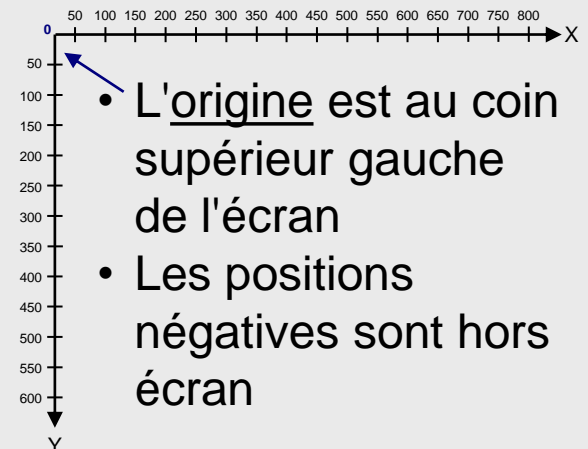


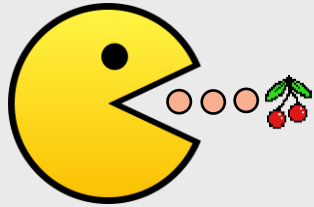
Systeme de coordonnees

- En géométrie, le plan cartésien représente un système de coordonnées 2D



- *MonoGame* exploite un système cartésien semblable, mais avec l'axe des Y inversé
 - L'unité de coordonnée correspond à un pixel d'écran

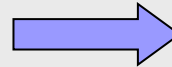
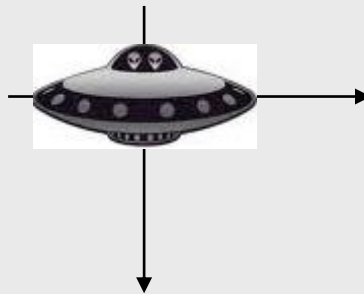


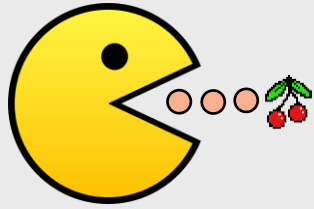


Système de coordonnées (suite)

- Même si l'origine est dans un coin de l'écran, un objet peut tout de même être positionné à des coordonnées négatives

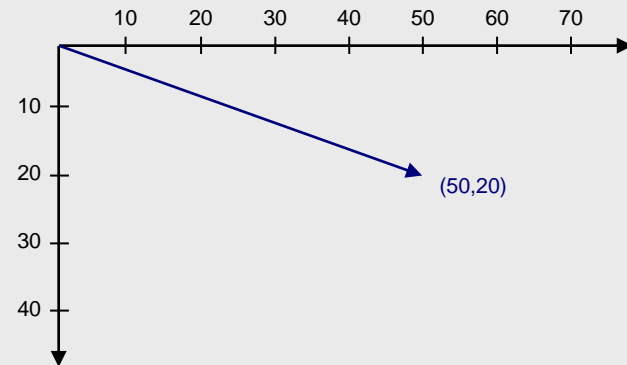
□ Exemple : Image centrée à l'origine (0,0)

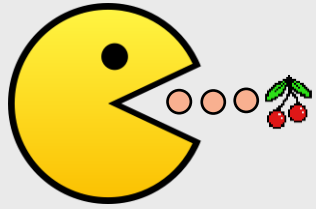




Les vecteurs

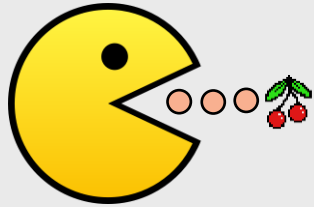
- Dans un plan cartésien 2D, une position est représentée par un point (x,y)
- Dans *MonoGame*, une position est représentée par un **vecteur**, dont les coordonnées sont l'extrémité d'une ligne originant de $(0,0)$
 - L'extrémité du vecteur est exploité pour représenter une position (i.e. un point)
 - L'autre point du vecteur est $(0,0)$ et est généralement ignoré





Les vecteurs (suite)

- Pourquoi exploiter des vecteurs plutôt que des points?
 - Car une classe **Vector** peut offrir toutes les fonctionnalités d'une classe **Point**
 - Puisque les vecteurs servent aussi à autres choses que la représentation de positions, alors autant les utiliser pour représenter des points
 - Moins de code source à entretenir
 - *MonoGame* permet de représenter des vecteurs en 2, 3 et même 4 dimensions (pour certaines manipulations mathématiques)



Classe `Vector2`

- La classe `Vector2` représente les vecteurs 2D

- ☐ Déclaration de variable

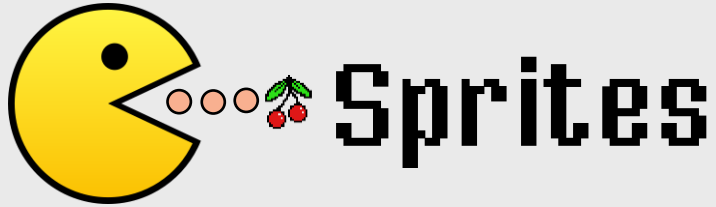
```
Vector2 positionVaisseau;
```

- ☐ Initialisation

```
positionVaisseau = new Vector2( 20.0f, 20.0f );
```

- ☐ Accès aux coordonnées

```
positionVaisseau.X = 50.0f;  
positionVaisseau.Y = 20.0f;
```

- Un sprite est une image 2D (i.e. *bitmap*)

- Exemples de sprites



- La plupart des jeux 2D exploitent des sprites

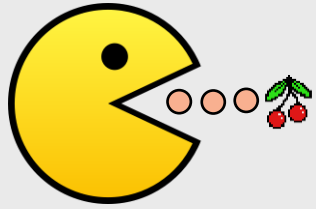
- Meilleure qualité visuelle

- Facilité de manipulation

- Translation

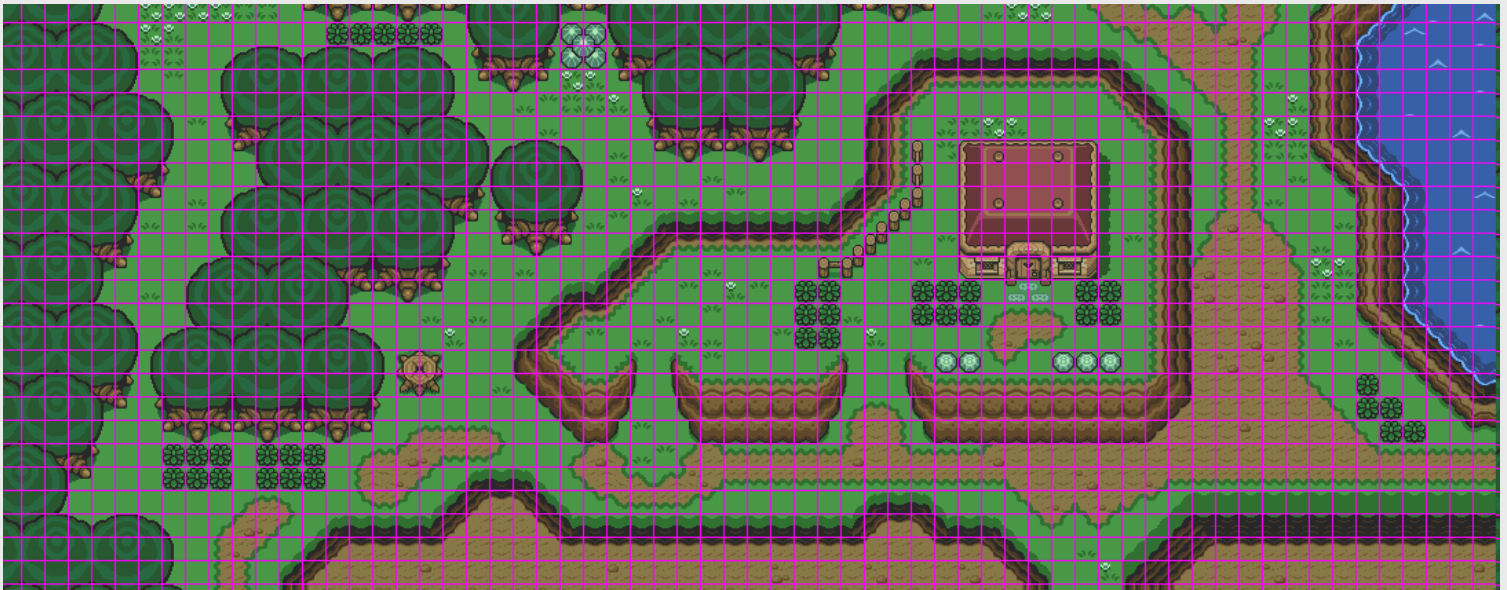
- Rotation

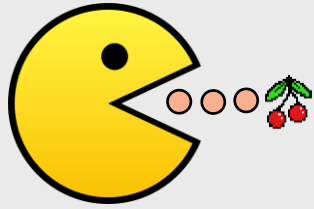
- Redimensionnement




Sprites (suite)

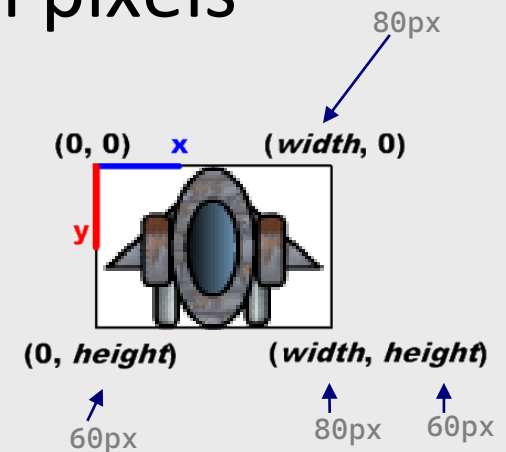
- Les tuiles d'arrière-plan de certains jeux 2D sont aussi stockées sous forme de sprites
 - Exemple : *Zelda*

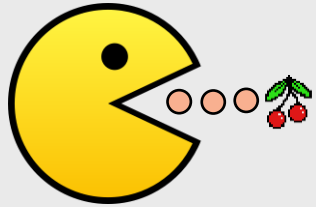




Sprites (suite)

- Dans *MonoGame*, les sprites sont appelés des *textures*
 - Exemple : classe `Texture2D`
- Taille d'un sprite = dimensions en pixels
 - Largeur x hauteur :  80 x 60
- Formats d'image supportés
 - BMP, JPG, PNG et TGA



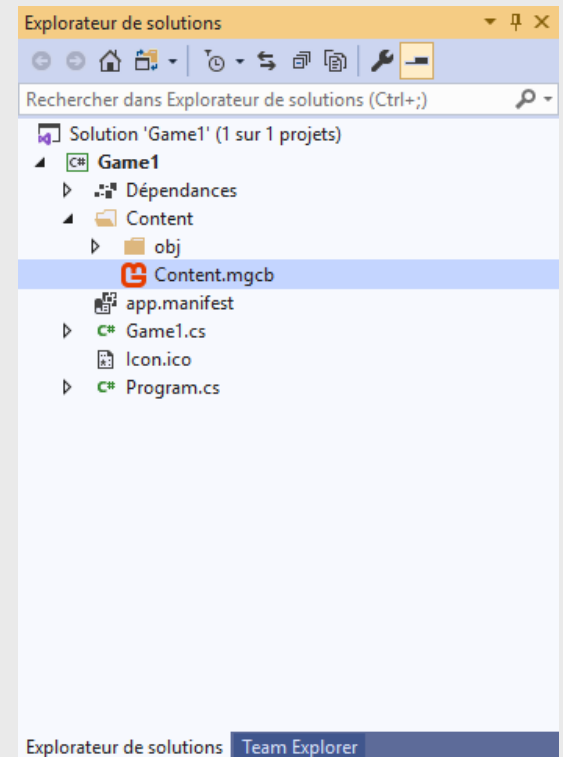


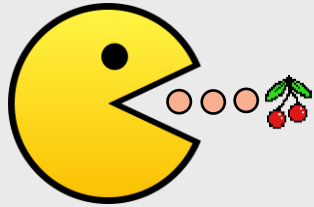
Ajouter un sprite au projet

- Via l'explorateur de solutions, sous la rubrique

Content

- ☐ Le fichier Content.mgcb est le « pipeline de contenu » du projet
- ☐ Ce fichier contient les ressources utilisées par le projet (textures, audio, polices, etc.)
- ☐ *MonoGame* fournit un éditeur pour ces fichiers : le *MonoGame Pipeline Tool*
 - Il est installé automatiquement avec *MonoGame*

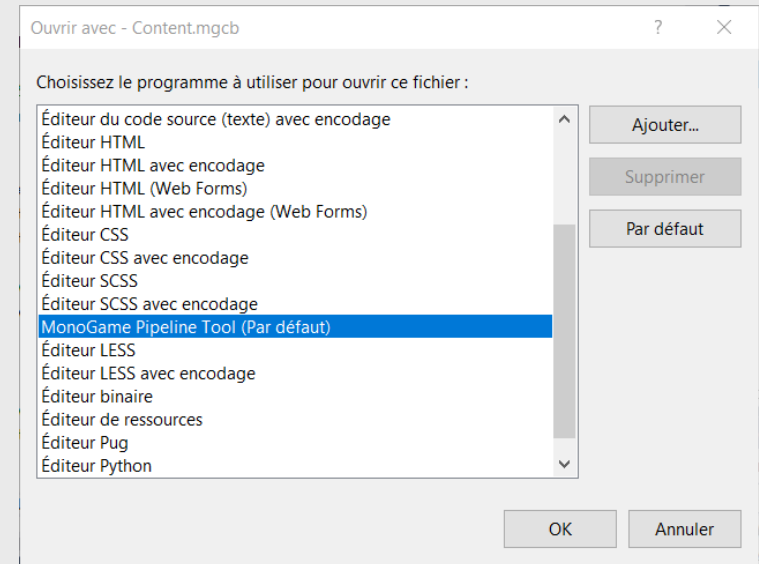


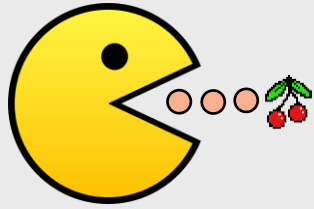


Éditer Content.mgcb

- La première fois, il faut associer les fichiers `.mgcb` au *MonoGame Pipeline Tool*

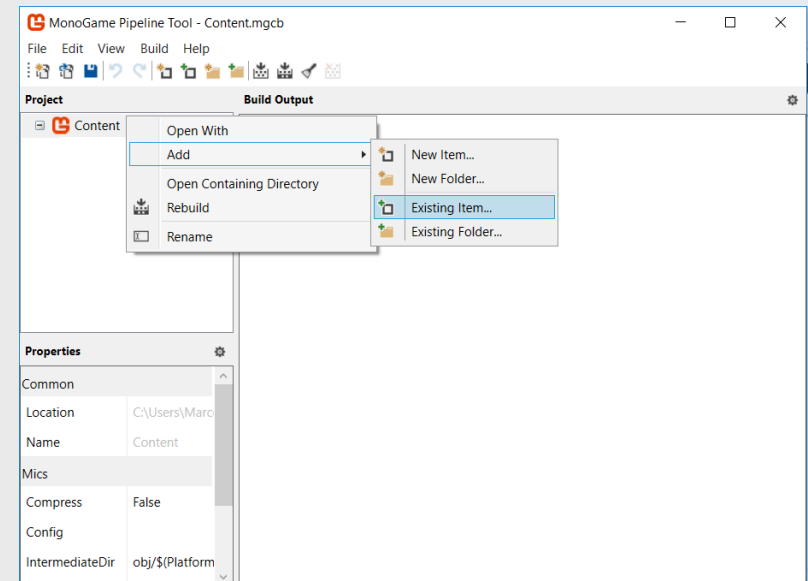
- Cliquez sur le fichier `Content.mgcb` avec le bouton droit de la souris et sélectionnez « Open with... »
- Sélectionnez *MonoGame Pipeline Tool*
- Cliquez le bouton *Par défaut* suivi du bouton *OK*
- La prochaine fois, il suffira de double-cliquer le fichier

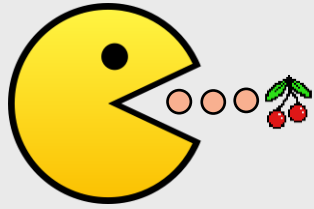




Éditer Content.mgcb

- Pour ajouter des sprites aux projet via le *MonoGame Pipeline Tool*
 1. Cliquez le bouton droit de la souris sur l'item *Content*
 2. Sélectionnez l'item de menu **Add** → **Existing item...**
 3. Sélectionnez les fichiers de texture à ajouter
 - Sélectionnez l'option **Copy the file to the directory** lorsqu'elle est offerte

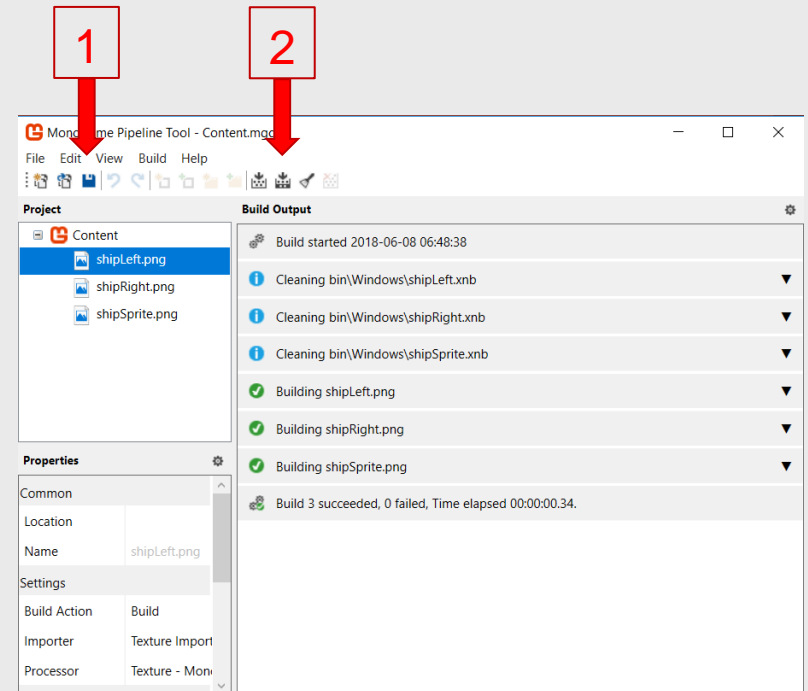


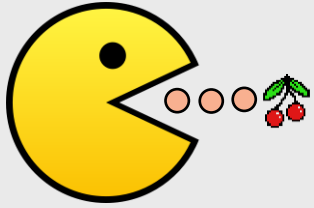


Éditer Content.mgcb

■ Une fois les fichiers de textures ajoutés

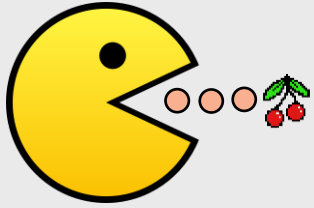
1. Sauvegardez les modifications au fichier pipeline
2. Recompilez le fichier pipeline
3. Vous pouvez ensuite fermer l'outil





Modifier les sprites du projet

- Il faudra ainsi utiliser le *MonoGame Pipeline Tool* pour
 - ☐ Ajouter des sprites au projet
 - ☐ Supprimer des sprites du projet
- Pour modifier un sprite existant
 1. Il faut modifier son fichier de texture à l'aide d'un outil d'édition d'images (p.ex. *Paint*, *Paint.NET* ou *Gimp*)
 2. Il faut remplacer l'image dans `Content.mgcb`



Charger un sprite

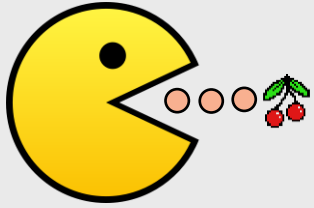
- Il faut premièrement créer un attribut membre **Texture2D** dans la classe pour le sprite

```
public class Game1 : Game {  
    GraphicsDeviceManager _graphics;  
    SpriteBatch _spriteBatch;  
  
    Texture2D vaisseauAvant;
```

- ... et charger l'image dans LoadContent()

- ☐ Le nom du fichier (sans l'extension) est exploité

```
protected override void LoadContent() {  
    _spriteBatch = new SpriteBatch(GraphicsDevice);  
  
    // TODO: use this.Content to load your game content here  
    vaisseauAvant = Content.Load<Texture2D>("shipSprite");  
}
```



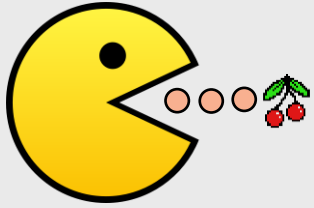
Positionner le sprite

- Créer une instance de **Vector2** pour conserver la position du sprite

```
public class Game1 : Game {  
    GraphicsDeviceManager _graphics;  
    SpriteBatch _spriteBatch;  
  
    Texture2D vaisseauAvant;  
    Vector2 vaisseauPosition;
```

- ... et initialiser sa position dans `Initialize()`

```
protected override void Initialize() {  
    // TODO: Add your initialization logic here  
    vaisseauPosition = new Vector2(100.0f, 100.0f);  
  
    base.Initialize();  
}
```

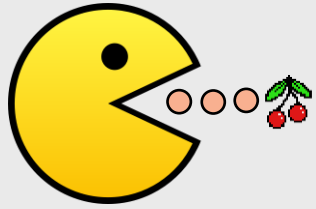


Afficher le sprite

- Exploiter l'objet `spriteBatch` dans la fonction membre `Draw()`

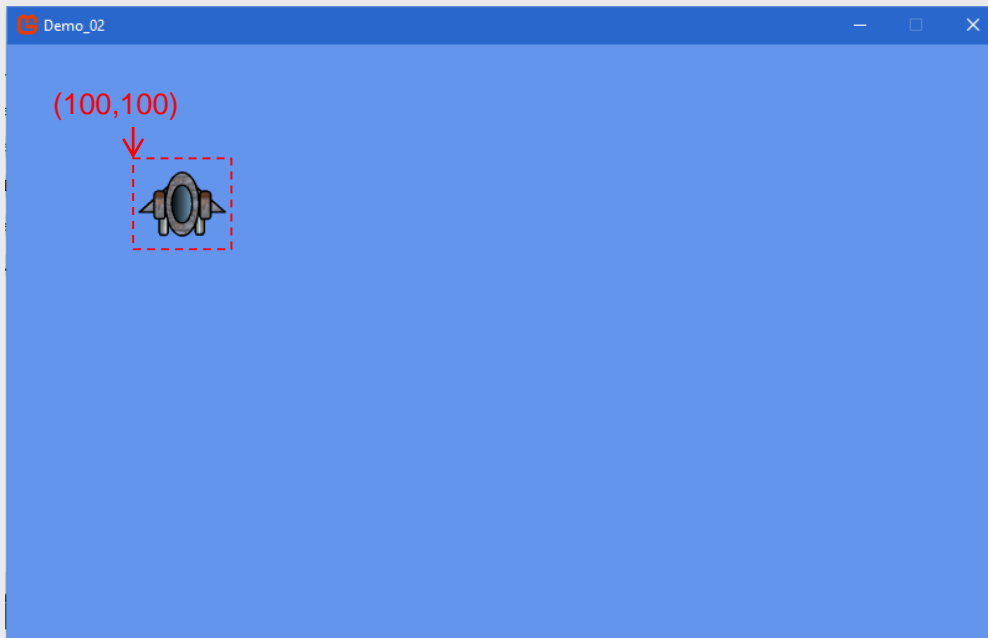
```
protected override void Draw(GameTime gameTime) {  
    GraphicsDevice.Clear(Color.CornflowerBlue);  
  
    // TODO: Add your drawing code here  
    _spriteBatch.Begin();  
    _spriteBatch.Draw(vaisseauAvant, vaisseauPosition, Color.White);  
    _spriteBatch.End();  
  
    base.Draw(gameTime);  
}
```

- ☐ L'objet `spriteBatch` est créé avec le projet
- ☐ La couleur `Color.White` sert à contrôler la *modulation* d'affichage (p.ex. teinte)
 - Le blanc désactive la modulation

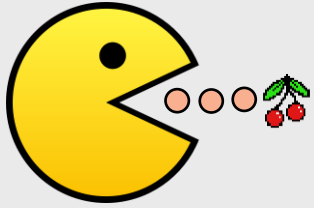


Afficher le sprite (suite)

- Et voilà le résultat



Notez que l'origine de l'image est positionnée à (100,100)



Qu'est de qu'un SpriteBatch ?

- Permet de regrouper les sprites

- ☐ Par fonctionnalités

- Vaisseaux, arrière-plan, textes, etc.

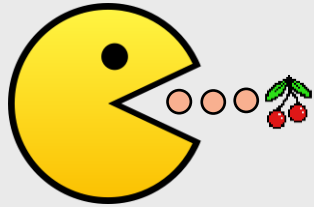
- ☐ Par caractéristiques

- Affichage transparent, superposition, etc.

- Accélère l'affichage

```
_spriteBatch.Begin();  
_spriteBatch.Draw(vaisseauAvant, vaisseauPosition, Color.White);  
_spriteBatch.End();
```

- ☐ Affichage dans un *bitmap* qui est par la suite affichée à l'écran (évite les clignotements)

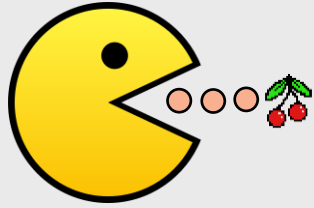


SpriteBatch.Draw()





■ Sept signatures, dont les deux principales

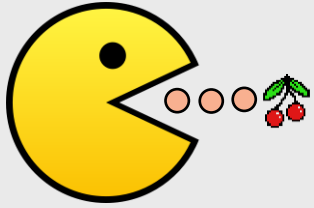
- `SpriteBatch.Draw(Texture2D sprite, Vector2 position, Color modulation)`
 - ← Sprite à afficher
 - ← Position à l'écran
 - ← Tinte optionnelle

- `SpriteBatch.Draw(Texture2D sprite, Vector2 position, Color modulation, float rotation, Vector2 origine, float echelle, SpriteEffects effets, float couche)`
 - ← Angle de rotation
 - ← Origine du sprite
 - ← Échelle du sprite
 - ← Effets (e.g. flip)
 - ← Profondeur (0.0f à 1.0f)



Exercice 2.1

- Reproduisez la démonstration précédente à l'aide des fichiers distribués par l'enseignant
 - ☐ Affichez le vaisseau tel que démontré
 - ☐ Supplément : affichez les autres sprites en fonctions des touches flèche pressées
 -  lorsque la flèche gauche  est pressée
 -  lorsque la flèche droite  est pressée
 - ☐ **Suggestion** : créez un nouvel attribut **Texture2D** pointant à l'instance à afficher selon la touche pressée

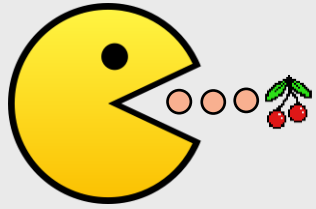


Ajouter un arrière-plan

- L'arrière-plan est aussi constitué d'un sprite
 - Ajouter l'image au projet (via le *Pipeline Tool*)
 - Créer une instance de `Texture2D` pour le sprite
 - Nouvel attribut de classe : `Texture2D arrierePlan;`
 - Dans `LoadContent()` :

```
arrierePlan = Content.Load<Texture2D>( "spaceBackground" );
```
 - Afficher le sprite à la position (0,0)
 - **Important** : l'afficher avant les autres sprites

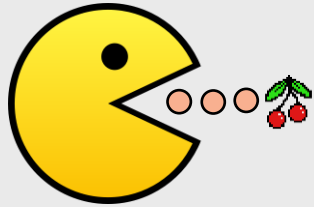
```
_spriteBatch.Begin();  
_spriteBatch.Draw(arrierePlan, position: Vector2.Zero);  
_spriteBatch.Draw(vaisseau, position: vaisseauPosition);  
_spriteBatch.End();
```

Ajouter un arrière-plan (suite)

- Voici le résultat :

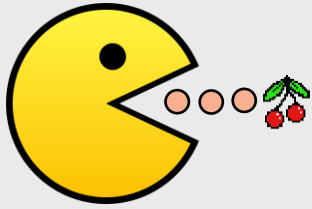




Déplacement latéral

- Pour déplacer latéralement le vaisseau
 - Modifier la position du vaisseau selon la touche flèche pressée dans `Update()`

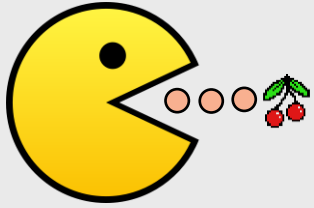
```
// Changer la texture du vaisseau selon sa direction de déplacement latéral
if (Keyboard.GetState().IsKeyDown(Keys.Left)) {
    vaisseau = vaisseauGauche;
    vaisseauPosition.X -= 5; // déplacer de 5 pixels vers la gauche
}
else if (Keyboard.GetState().IsKeyDown(Keys.Right)) {
    vaisseau = vaisseauDroite;
    vaisseauPosition.X += 5; // déplacer de 5 pixels vers la droite
}
else
    vaisseau = vaisseauAvant;
```
 - Est-ce que la vitesse de déplacement varie selon la puissance du matériel ? *Oui!*



Déplacement latéral (suite)

- Pour rendre la vitesse de déplacement indépendante du matériel
 - Exploiter le paramètre `gameTime`
 - Trouver une vitesse de déplacement adéquate ($0.5f$) par expérimentation

```
// Changer la texture du vaisseau selon sa direction de déplacement latéral
if (Keyboard.GetState().IsKeyDown(Keys.Left)) {
    vaisseau = vaisseauGauche;
    vaisseauPosition.X -= gameTime.ElapsedGameTime.Milliseconds * 0.5f; // déplacer vers ...
}
else if (Keyboard.GetState().IsKeyDown(Keys.Right)) {
    vaisseau = vaisseauDroite;
    vaisseauPosition.X += gameTime.ElapsedGameTime.Milliseconds * 0.5f; // déplacer vers ...
}
else
    vaisseau = vaisseauAvant;
```

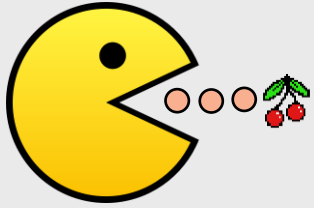


Garder le vaisseau à l'écran

- Éviter que le vaisseau soit déplacé latéralement hors de l'écran

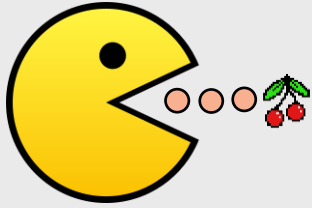
```
// Changer la texture du vaisseau selon sa direction de déplacement latéral
if (Keyboard.GetState().IsKeyDown(Keys.Left)) {
    vaisseau = vaisseauGauche;
    vaisseauPosition.X = System.Math.Max(vaisseauPosition.X -
                                         gameTime.ElapsedGameTime.Milliseconds * 0.5f, 0.0f); // déplacer
}
else if (Keyboard.GetState().IsKeyDown(Keys.Right)) {
    vaisseau = vaisseauDroite;
    vaisseauPosition.X = System.Math.Min(vaisseauPosition.X +
                                         gameTime.ElapsedGameTime.Milliseconds * 0.5f,
                                         graphics.GraphicsDevice.Viewport.Width - vaisseau.Width); // dép
}
else
    vaisseau = vaisseauAvant;
```

- Notez que la limite supérieure (i.e. droite) tient compte de la largeur du sprite



Exercice 2.2

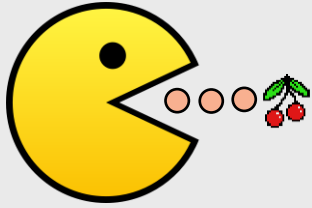
- Introduisez le mouvement vertical du vaisseau (i.e. avant et arrière)
 - ☐ Assurez-vous que le vaisseau ne quitte pas l'écran
- Ajustez les limites de déplacements latéraux jusqu'au centre du vaisseau (i.e. la moitié du vaisseau peut quitter l'écran)
- La position de départ du vaisseau doit être centrée horizontalement, au bas de l'écran à 9/10 de sa hauteur



Observations sur la solution de l'exercice 2.2

- Le fait que l'origine du sprite soit son coin supérieur droit complique le code source

```
// Changer la texture du vaisseau selon sa direction de déplacement latéral
if (Keyboard.GetState().IsKeyDown(Keys.Left)) {
    vaisseau = vaisseauGauche;
    vaisseauPosition.X = System.Math.Max(vaisseauPosition.X -
                                           gameTime.ElapsedGameTime.Milliseconds * 0.5f, -vaisseau.Width / 2f);
}
else if (Keyboard.GetState().IsKeyDown(Keys.Right)) {
    vaisseau = vaisseauDroite;
    vaisseauPosition.X = System.Math.Min(vaisseauPosition.X +
                                           gameTime.ElapsedGameTime.Milliseconds * 0.5f,
                                           graphics.GraphicsDevice.Viewport.Width - vaisseau.Width / 2f);
}
else
    vaisseau = vaisseauAvant;
```



Observations sur la solution de l'exercice 2.2 (suite)

- Logique simplifiée si l'origine du sprite est son centre

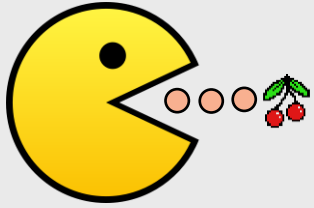
- ☐ L'origine est spécifiée dans Draw() :

```
_spriteBatch.Draw(vaisseau,  
    position: vaisseauPosition,  
    sourceRectangle: new Rectangle(0, 0, vaisseau.Width, vaisseau.Height),  
    Color.White,  
    rotation: 0,  
    origin: new Vector2(vaisseau.Width / 2f, vaisseau.Height / 2f),  
    scale: 1,  
    SpriteEffects.None,  
    layerDepth: 0);
```

- ☐ Refactoriser le code en conséquence, tel que

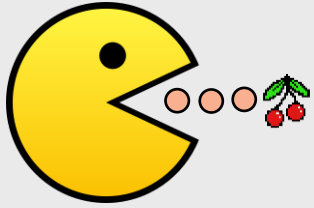
```
// Initialement, le vaisseau ne se déplace pas et est positionné centré au bas de l'écran  
vaisseau = vaisseauAvant;
```

```
vaisseauPosition.X = graphics.PreferredBackBufferWidth / 2f - vaisseau.Width / 2f;  
vaisseauPosition.Y = graphics.PreferredBackBufferHeight * 0.9f - vaisseau.Height / 2f;
```



Un peu de physique

- Notre vaisseau doit être soumis aux principes de physique suivants
 - **Accélération** : les déplacements doivent accélérer graduellement
 - **Inertie** : les décélérations doivent aussi être graduelles lorsque les touches de déplacement (c.à.d. les flèches) sont libérées



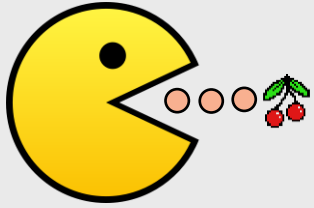
Un peu de physique (suite)

■ Refactorisation du code requise

- Attributs de classe pour vitesses latérale et frontale (car ces vitesses vont fluctuer)

```
public class MyGame : Game {  
    GraphicsDeviceManager _graphics;  
    SpriteBatch _spriteBatch;  
  
    float vitesseLaterale = 0.0f;           // vitesse lors des déplacements de côté  
    float vitesseFrontale = 0.0f;          // vitesse lors des déplacements avant/arrière
```

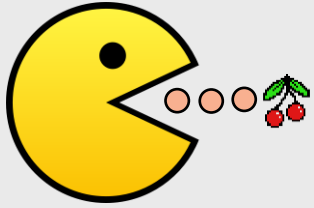
- Les vitesses seront continuellement appliquées à la position du vaisseau
 - Presser les touches flèches occasionnera une modification à la vitesse correspondante



Un peu de physique (suite)

■ Refactorisation de Update()

```
protected override void Update(GameTime gameTime) {  
    ...  
  
    // Changer la texture du vaisseau selon sa direction de déplacement latéral  
    if (Keyboard.GetState().IsKeyDown(Keys.Left) &&  
        !Keyboard.GetState().IsKeyDown(Keys.Right)) {  
        vaisseau = vaisseauGauche;  
        vitesseLaterale = -0.5f;        // appliquer une vitesse en X décrémentationale  
    }  
    else if (Keyboard.GetState().IsKeyDown(Keys.Right) &&  
        !Keyboard.GetState().IsKeyDown(Keys.Left)) {  
        vaisseau = vaisseauDroite;  
        vitesseLaterale = 0.5f;        // appliquer une vitesse en X incrémentale  
    }  
    else {  
        vaisseau = vaisseauAvant;  
        vitesseLaterale = 0f;        // aucun déplacement en X  
    }  
}
```



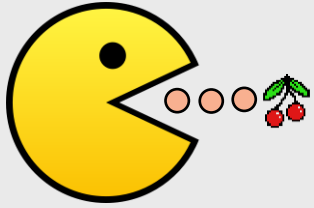
Un peu de physique (suite)

■ Refactorisation de Update() (suite)

```
// Changer la position verticale selon la touche pressée.
if (Keyboard.GetState().IsKeyDown(Keys.Up) && !(Keyboard.GetState().IsKeyDown(Keys.Down)))
    vitesseFrontale = -0.5f;        // appliquer une vitesse en Y décrémenteale
else if (Keyboard.GetState().IsKeyDown(Keys.Down) && !(Keyboard.GetState().IsKeyDown(Keys.Up)))
    vitesseFrontale = 0.5f;        // appliquer une vitesse en Y incrémementale
else
    vitesseFrontale = 0f;           // aucun déplacement en Y

// Déplacer le vaisseau en fonction des vitesses latérales et frontales
vaisseauPosition.X =
    MathHelper.Clamp(vaisseauPosition.X + gameTime.ElapsedGameTime.Milliseconds * vitesseLaterale,
        0f, graphics.GraphicsDevice.Viewport.Width);
vaisseauPosition.Y =
    MathHelper.Clamp(vaisseauPosition.Y + gameTime.ElapsedGameTime.Milliseconds * vitesseFrontale,
        vaisseau.Height / 2f, graphics.GraphicsDevice.Viewport.Height - vaisseau.Height / 2f);

base.Update(gameTime);
}
```

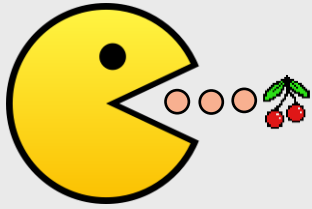


Accélération et décélération

■ Modifier graduellement la vitesse latérale dans Update()

```
// Changer la texture du vaisseau selon sa direction de déplacement latéral
if (Keyboard.GetState().IsKeyDown(Keys.Left) &&
    !Keyboard.GetState().IsKeyDown(Keys.Right)) {
    vaisseau = vaisseauGauche;
    vitesseLaterale = -0.5f; ← System.Math.Max(vitesseLaterale - 0.02f, -0.5f)
}
else if (Keyboard.GetState().IsKeyDown(Keys.Right) &&
    !Keyboard.GetState().IsKeyDown(Keys.Left)) {
    vaisseau = vaisseauDroite;
    vitesseLaterale = 0.5f; ← System.Math.Min(vitesseLaterale + 0.02f, 0.5f)
}
else {
    vaisseau = vaisseauAvant;
    vitesseLaterale = 0f; ←
}

if (vitesseLaterale <= -0.02f)
    vitesseLaterale += 0.02f;
else if (vitesseLaterale >= 0.02f)
    vitesseLaterale -= 0.02f;
else
    vitesseLaterale = 0.0f
```

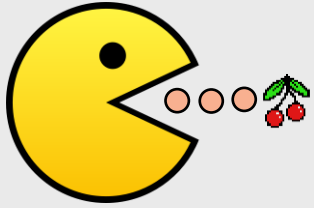


Accélération et décélération (suite)

- Modifier graduellement la vitesse frontale dans `Update()`

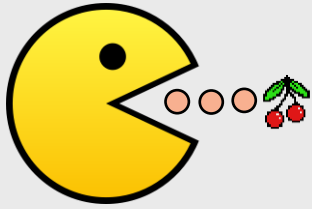
```
// Changer la position verticale selon la touche pressée.  
if (Keyboard.GetState().IsKeyDown(Keys.Up) && !(Keyboard.GetState().IsKeyDown(Keys.Down)))  
    vitesseFrontale = System.Math.Max(vitesseFrontale - 0.02f, -0.5f);  
else if (Keyboard.GetState().IsKeyDown(Keys.Down) && !(Keyboard.GetState().IsKeyDown(Keys.Up)))  
    vitesseFrontale = System.Math.Min(vitesseFrontale + 0.02f, 0.5f);  
else  
    if (System.Math.Abs(vitesseFrontale) >= 0.02f)  
        vitesseFrontale -= System.Math.Sign(vitesseFrontale) * 0.02f;  
    else  
        vitesseFrontale = 0.0f;
```

- À noter que l'utilisation de `Math.Sign()` est équivalente à la structure `if` correspondante de l'acétate précédente



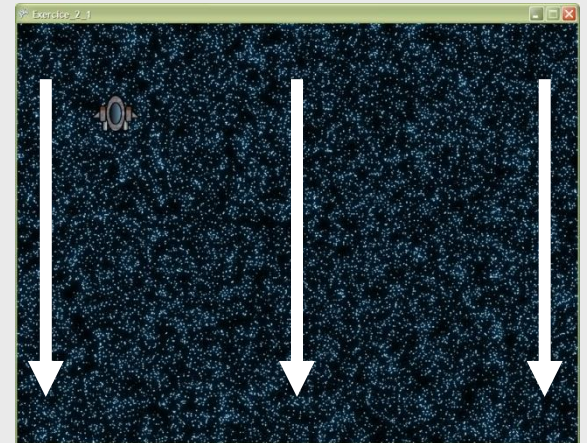
Exercice 2.3

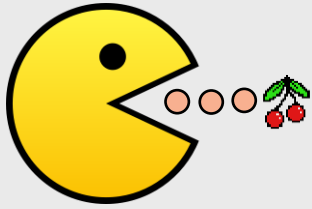
- Introduisez l'accélération et la décélération dans votre solution de l'exercice 2.2
 - Exploitez une constante (initialisée à $0.02f$) pour représenter le facteur d'accélération et une autre (initialisée à $0.5f$) pour la vitesse maximale



Défilement de l'arrière-plan

- Défilement vertical de l'arrière-plan afin d'émuler le déplacement continu du vaisseau
- Stratégie
 - Déplacer continuellement vers le bas le sprite d'arrière-plan
 - Réafficher le sprite au dessus afin de combler l'espace vide au-dessus du premier affichage
 - *L'image d'arrière-plan doit évidemment être assez grande pour remplir l'écran*





Défilement de l'arrière-plan (suite)

- Nouvel attribut de classe :

```
Vector2 arrierePlanPosition;  
float vitesseArrierePlan;
```

- Dans Initialize() :

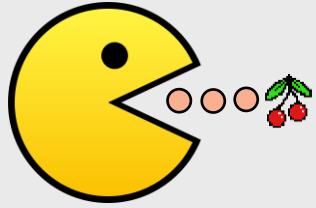
```
arrierePlanPosition = new Vector2(0f, 0f); // position initiale de l'arrière-plan  
vitesseArrierePlan = 0.15f // vitesse de défilement de l'arrière-plan
```

- Dans Update() :

```
// Mettre à jour l'arrière-plan  
arrierePlanPosition.Y += gameTime.ElapsedGameTime.Milliseconds * vitesseArrierePlan;  
if (arrierePlanPosition.Y > graphics.GraphicsDevice.Viewport.Height)  
    arrierePlanPosition.Y -= arrierePlan.Height;
```

- Dans Draw() :

```
// Afficher l'arrière-plan deux fois (un au-dessus de l'autre) afin de combler l'espace en  
// haut d'écran laissée vacante par l'image défilant vers le bas.  
_spriteBatch.Draw(arrierePlan,  
    position: new Vector2(arrierePlanPosition.X, arrierePlanPosition.Y - arrierePlan.Height),  
    Color.White);  
_spriteBatch.Draw(arrierePlan, position: arrierePlanPosition, Color.White);
```

Complexité de Update()

- La fonction membre Update() devient vite complexe avec l'ajout de multiples sprites

- Du code source pour la mise à jour de chaque sprite

- Exemple pour ci-contre pour deux sprites

- Il faut répartir ce code dans différentes classes

```
protected override void Update(GameTime gameTime)
{
    // Permettre de quitter le jeu via la manette ou le clavier.
    if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed || Keyboard.GetState().IsKeyDown(Keys.Escape))
        Exit();

    const float facteurAcceleration = 0.02f; // facteur d'accélération et de décélération
    const float vitesseMaximale = 0.5f; // vitesse latérale et frontale maximale
    const float vitesseArrierePlan = 0.2f; // vitesse de défilement de l'arrière-plan

    // Changer la texture du vaisseau selon sa direction de déplacement latéral, ainsi que sa position horizontale
    // selon la touche pressée et sa vitesse latérale tout en appliquant l'inertie si la vitesse de croisière n'est
    // pas atteinte.
    if (Keyboard.GetState().IsKeyDown(Keys.Left) && !Keyboard.GetState().IsKeyDown(Keys.Right))
    {
        vaisseau = vaisseauGauche;
        vitesselerale = System.Math.Max(vitesselerale - facteurAcceleration, -vitesseMaximale);
    }
    else if (Keyboard.GetState().IsKeyDown(Keys.Right) && !Keyboard.GetState().IsKeyDown(Keys.Left))
    {
        vaisseau = vaisseauDroite;
        vitesselerale = System.Math.Min(vitesselerale + facteurAcceleration, vitesseMaximale);
    }
    else
    {
        vaisseau = vaisseauAvant;

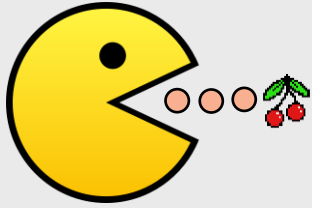
        // Réduire le déplacement en X selon la direction de celui-ci tout en appliquant l'inertie si la vitesse nulle
        // n'est pas atteinte.
        if (System.Math.Abs(vitesselerale) >= facteurAcceleration)
            vitesselerale -= System.Math.Sign(vitesselerale) * facteurAcceleration;
        else
            vitesselerale = 0.0f;
    }

    // Changer la position verticale selon la touche pressée et sa vitesse latérale tout en appliquant l'inertie si la
    // vitesse de croisière n'est pas atteinte.
    if (Keyboard.GetState().IsKeyDown(Keys.Up) && !Keyboard.GetState().IsKeyDown(Keys.Down))
        vitesseFrontale = System.Math.Max(vitesseFrontale - facteurAcceleration, -vitesseMaximale);
    else if (Keyboard.GetState().IsKeyDown(Keys.Down) && !Keyboard.GetState().IsKeyDown(Keys.Up))
        vitesseFrontale = System.Math.Min(vitesseFrontale + facteurAcceleration, vitesseMaximale);
    else
    {
        // Réduire le déplacement en Y selon la direction de celui-ci tout en appliquant l'inertie si la vitesse nulle
        // n'est pas atteinte.
        if (System.Math.Abs(vitesseFrontale) >= facteurAcceleration)
            vitesseFrontale -= System.Math.Sign(vitesseFrontale) * facteurAcceleration;
        else
            vitesseFrontale = 0.0f;
    }

    // Déplacer le vaisseau en fonction des vitesses latérales et frontales.
    vaisseauPosition.X = MathHelper.Clamp(vaisseauPosition.X + gameTime.ElapsedGameTime.Milliseconds * vitesselerale,
        0f, graphics.GraphicsDevice.Viewport.Width);
    vaisseauPosition.Y = MathHelper.Clamp(vaisseauPosition.Y + gameTime.ElapsedGameTime.Milliseconds * vitesseFrontale,
        vaisseau.Height / 2f, graphics.GraphicsDevice.Viewport.Height - vaisseau.Height / 2f);

    // Mettre à jour l'arrière-plan
    arrierePlanPosition.Y += gameTime.ElapsedGameTime.Milliseconds * vitesseArrierePlan;
    if (arrierePlanPosition.Y > graphics.GraphicsDevice.Viewport.Height)
        arrierePlanPosition.Y -= arrierePlan.Height;

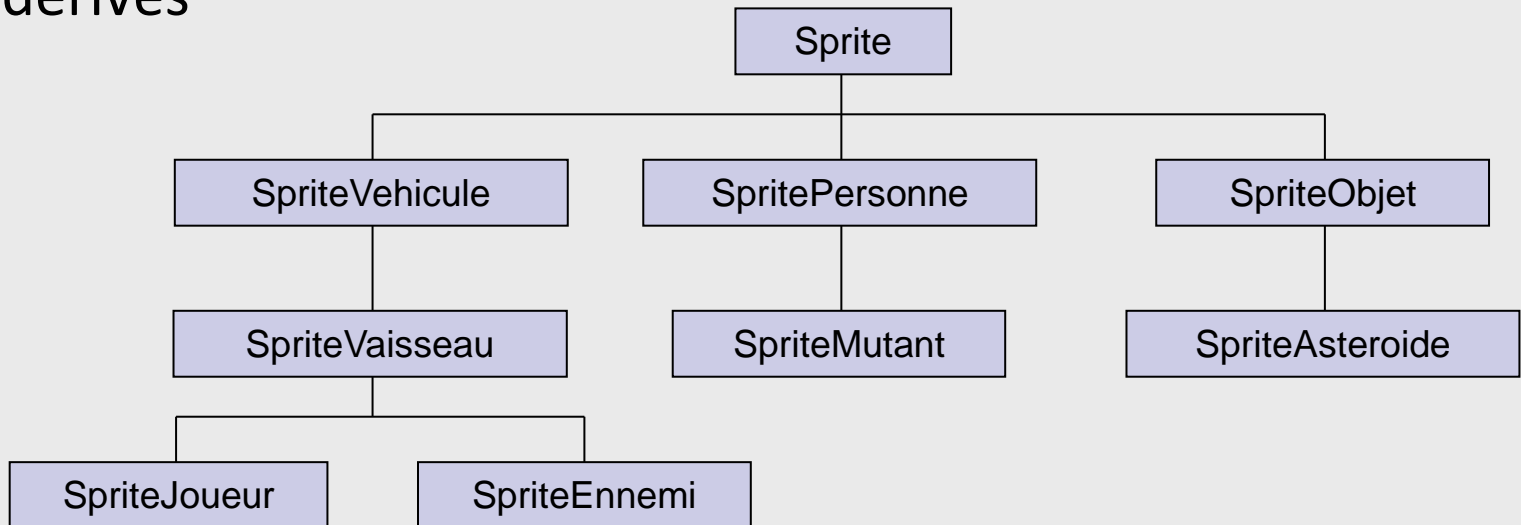
    base.Update(gameTime);
}
```

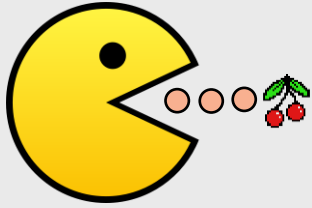


Complexité de Update()

Classe pour gérer les sprites

- Pour faciliter la programmation, nous implantons une classe de base facilitant la gestion d'un sprite
 - Les sprites spécialisés sont implantés dans des classes dérivés



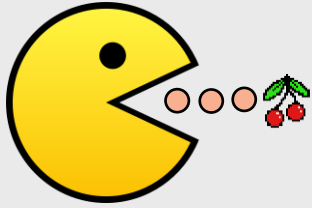


Complexité de Update()

Classe **Sprite**

■ Classe abstraite (i.e. non instanciable)

```
public abstract class Sprite {  
    // Propriété abstraite pour manipuler la texture du sprite. Doit être  
    // surchargée dans les classes dérivées afin de manipuler une Texture2D.  
    public abstract Texture2D Texture { get; }  
  
    public Vector2 Position { get; }          // position du sprite en 2D  
  
    // Accesseur surchargeable pour obtenir la largeur du sprite.  
    public virtual int Width {  
        get { return Texture.Width; }  
    }  
    public virtual int Width => Texture.Width;  
  
    // Accesseur surchargeable pour obtenir la hauteur du sprite.  
    public virtual int Height {  
        get { return Texture.Height; }  
    }  
    public virtual int Height => Texture.Height;
```



Complexité de Update()

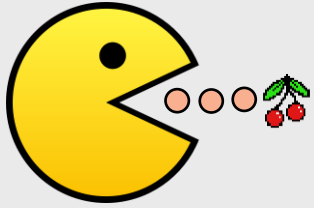
Classe `Sprite` (suite)

```
// Constructeur paramétré recevant la position du sprite.
public Sprite(int x, int y) : this((float)x, (float)y)
{}

// Constructeur paramétré recevant la position du sprite.
public Sprite(float x, float y) {
    this.Position = new Vector2(x, y);
}

// Fonction membre abstraite (doit être surchargée) mettant à jour le sprite.
public abstract void Update(GameTime gameTime, GraphicsDeviceManager graphics);

// Fonction membre à surcharger pour dessiner le sprite. Par défaut la
// texture est affichée centrée à sa position.
public virtual void Draw(SpriteBatch spriteBatch) {
    _spriteBatch.Draw(this.Texture,
        position: this.Position,
        rotation: 0.0f,
        sourceRectangle: new Rectangle(0, 0, this.Texture.Width, this.Texture.Height),
        origine: new Vector2(this.Width / 2f, this.Height / 2f)
        scale: 1.0f,
        SpriteEffects.None,
        layerDepth: 0.0f);
}
}
```



Complexité de Update()

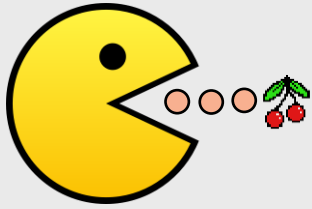
Classes dérivées de **Sprite**

■ Classe **ArrierePlan**

- ☐ Y intégrer les fonctionnalités associées à la gestion de l'arrière-plan

■ Classe **VaisseauJoueur**

- ☐ Y intégrer les attributs membres du vaisseau (vaisseauAvant, vaisseauGauche, vaisseauDroite, vitesseLaterale et vitesseFrontale) et les fonctionnalités associées à ces attributs membres



Complexité de Update()

Classe de jeu résultante

■ Classe de jeu exploitant nos nouvelles classes

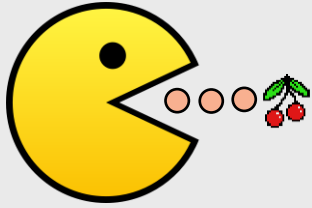
```
public class Game1 : Game {
    GraphicsDeviceManager _graphics;
    SpriteBatch _spriteBatch;

    VaisseauJoueur _joueur;           // sprite du joueur
    ArrierePlan _arrierePlan;         // arrière-plan du jeu

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

    protected override void Initialize() {
        _joueur = new VaisseauJoueur(0.0f, 0.0f); // créer le sprite du joueur
        _arrierePlan = new ArrierePlan(0.0f, 0.0f); // créer le sprite d'arrière-plan

        base.Initialize();
    }
}
```



Complexité de Update()

Classe de jeu résultante (suite)

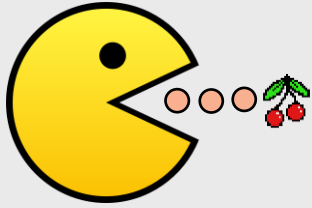
```
protected override void LoadContent() {
    _spriteBatch = new SpriteBatch(GraphicsDevice);

    // À FAIRE: Utiliser this.Content pour charger votre contenu ici.
    _joueur.LoadContent(this.Content, _graphics);
    _arrierePlan.LoadContent(this.Content, _graphics);
}

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

    // À FAIRE: Ajoutez votre logique de mise à jour ici.
    _joueur.Update(gameTime, graphics);
    _arrierePlan.Update(gameTime, graphics);

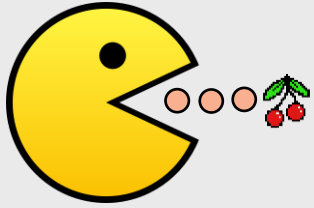
    base.Update(gameTime);
}
```



Complexité de Update()

Classe de jeu résultante (suite)

```
protected override void Draw(GameTime gameTime) {  
    GraphicsDevice.Clear(Color.CornflowerBlue);  
  
    // À FAIRE: Ajoutez votre code d'affichage ici.  
    _spriteBatch.Begin();  
  
    _arrierePlan.Draw(_spriteBatch);  
    _joueur.Draw(_spriteBatch);  
  
    _spriteBatch.End();  
  
    base.Draw(gameTime);  
}
```

La semaine prochaine

- Structure de deux autres types d'arrière-plan pour jeux 2D
 - Tuiles géographiques
 - Exemples : *Zelda*, *Ultima*
 - Défilement horizontal
 - Exemples: *Mario Bros*, jeux de combats



LUP