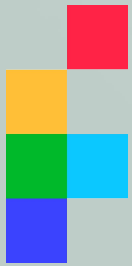
Initiation à la programmation de jeux vidéo avec AS3, Flixel et FlashDevelop

Licence Professionnelle Jeux Vidéo

Paris 13

chauvin.simon@gmail.com



2012 - 2013

Présentation

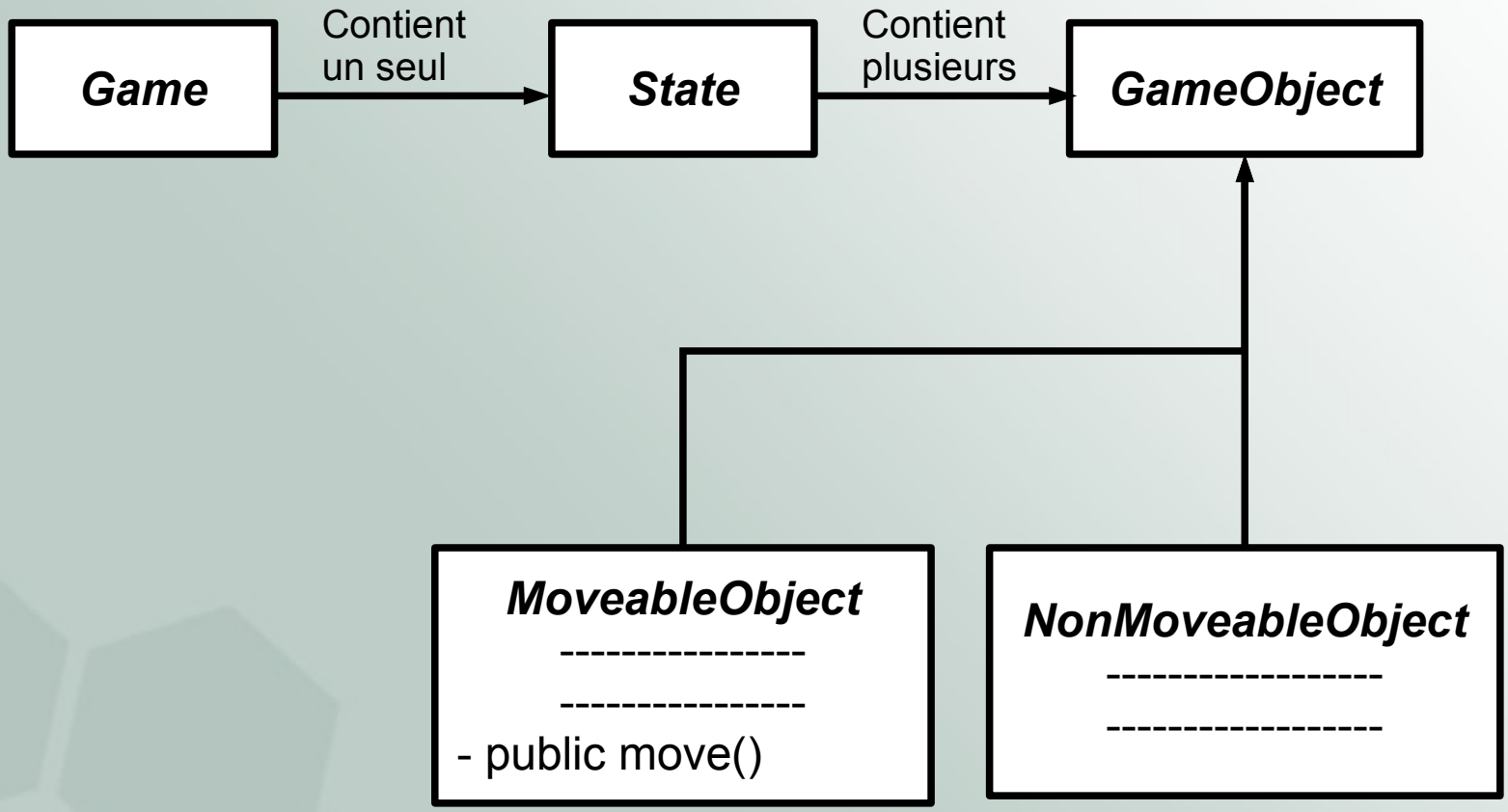
- Programmation d'un jeu vidéo :
  - *Principes, description*
- Fonctionnement de Flixel :
  - *Principes, architecture*
- Utilisation basique de Flixel :
  - *States, groups, sprites, collisions, scrolling*

Programmation d'un jeu vidéo

Programmation d'un jeu vidéo

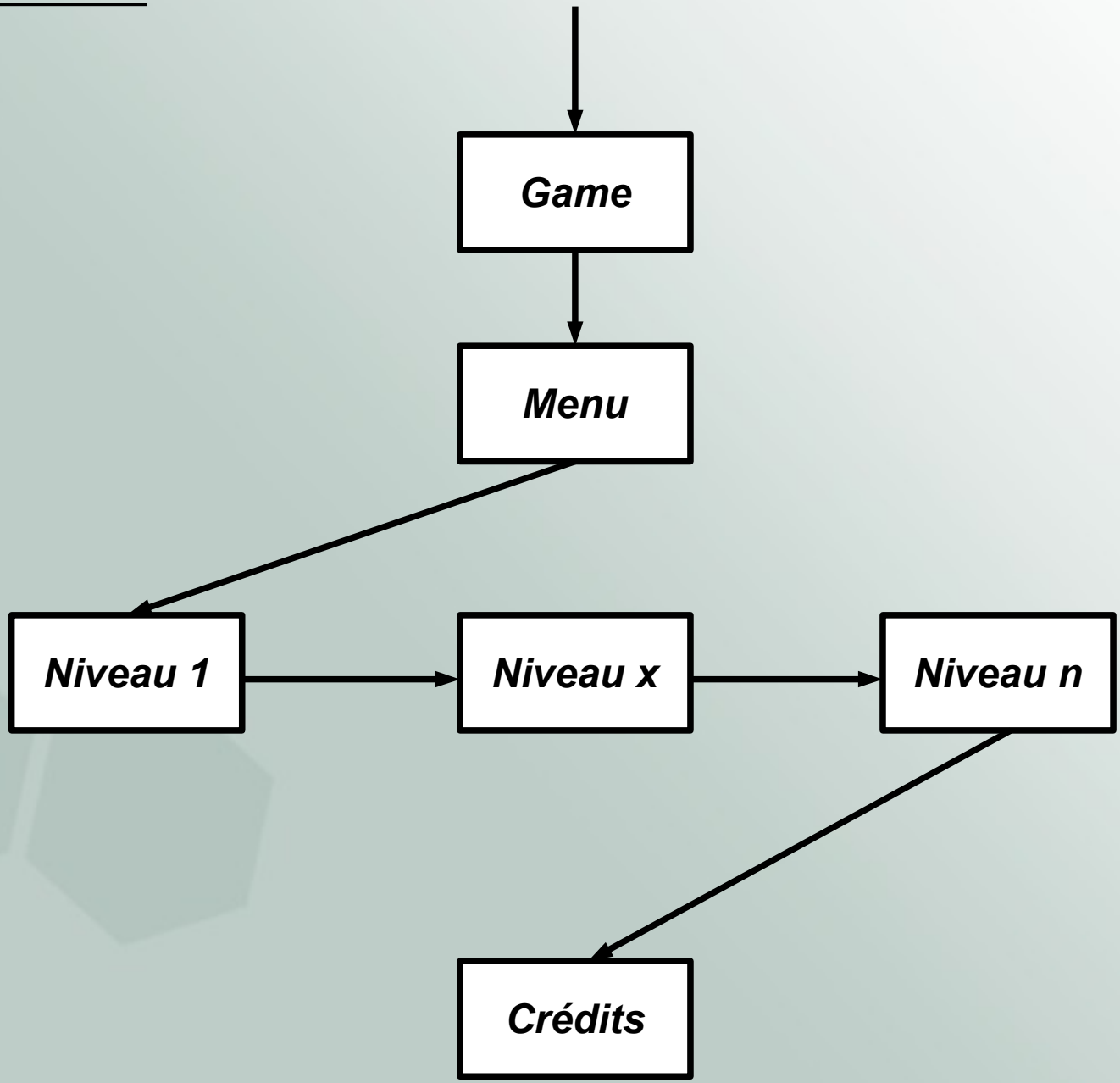
- La programmation gameplay :
  - Utilisation des **fonctions** fournis par le **moteur** pour créer la **structure** ainsi que les **règles** du jeu
  - Ce que nous allons faire en initiation et spécialisation AS3
- La programmation moteur :
  - **Architecture** du moteur
  - Rendu, réseau, sons, etc.
  - **Gestion** des sprites, animations, states, etc.
  - Ce que **Flixel** fera pour nous

- Architecture type d'un jeu :



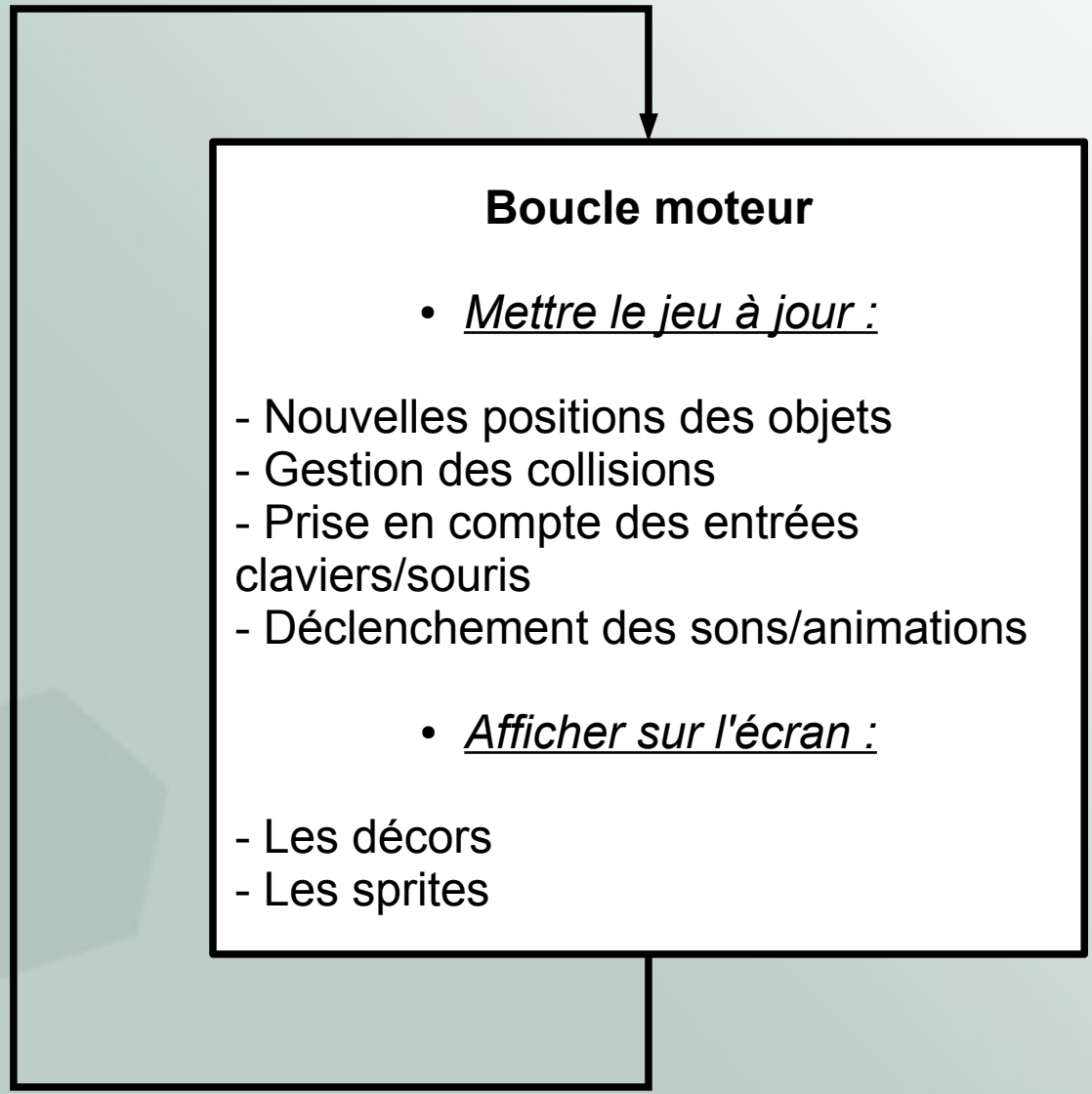
Programmation d'un jeu vidéo

- Flux :



Programmation d'un jeu vidéo

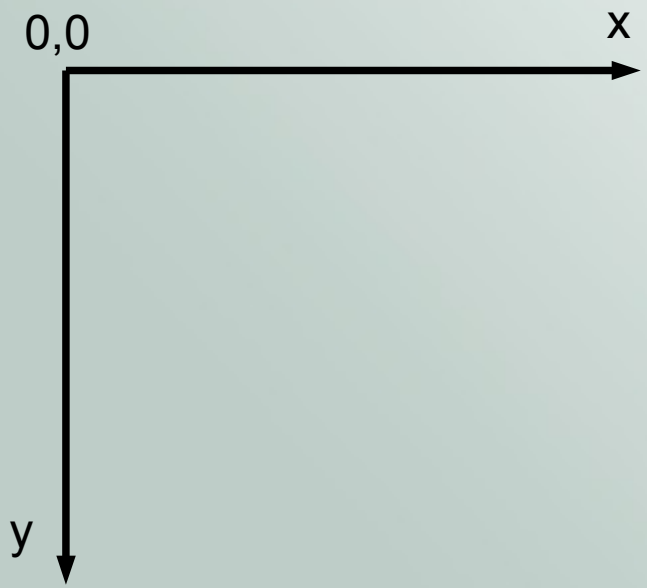
- Boucle moteur :



- Boucle moteur :
  - Le jeu **met à jour** (« update ») les éléments de jeu et **redessine** l'écran (« draw »)
  - Dans Tetris à chaque mise à jour le moteur **modifie** la position des blocs puis les **redessinent**
  - Dans un jeu à scrolling type SuperMario Bros à chaque frame le moteur **modifie** la position du joueur et **dessine** la zone de jeu visible
  - En Flash et flixel « draw » et « update » sont rassemblés en une seule « update »

- Frame rate :
  - Nombre de **parcours** de la boucle **moteur** par secondes
  - *Norme actuelle :* 30, 60 ou 100 frames par secondes (= 1/30, 1/60 ou 1/100 secondes entre deux frames)
  - Dépend du taux de rafraîchissement des entrées/sorties (écran, souris, clavier)
  - Frame rate **variable** en Flash et Flixel

- Repère :



- Origine en haut à gauche
- La position y d'un objet augmente à mesure qu'il descend

- Les assets :
  - *Image vectorielle :* importée d'un outil de dessin vectoriel ou directement créée avec Adobe Flash, elle est composée uniquement de primitives géométriques
  - *Image bitmap :* importée d'un outil de dessin, souvent des .png (gestion de la transparence), correspond à une suite de pixels
  - *Son :* importé d'un fichier .wav, .mp3, etc.
  - Flixel utilise des bitmaps
  - Flash utilise des .mp3 mais ceux-ci imposent un blanc au début et à la fin du morceau, nous verrons une technique permettant d'utiliser des .wav

- Les assets :
  - *Bitmap :*
    - Plus de mémoire
    - Plus performant
    - Adapté pour le jeu vidéo
  - *Vectoriel :*
    - Moins de mémoire
    - Moins performant
    - Adapté pour les sites web et la pub

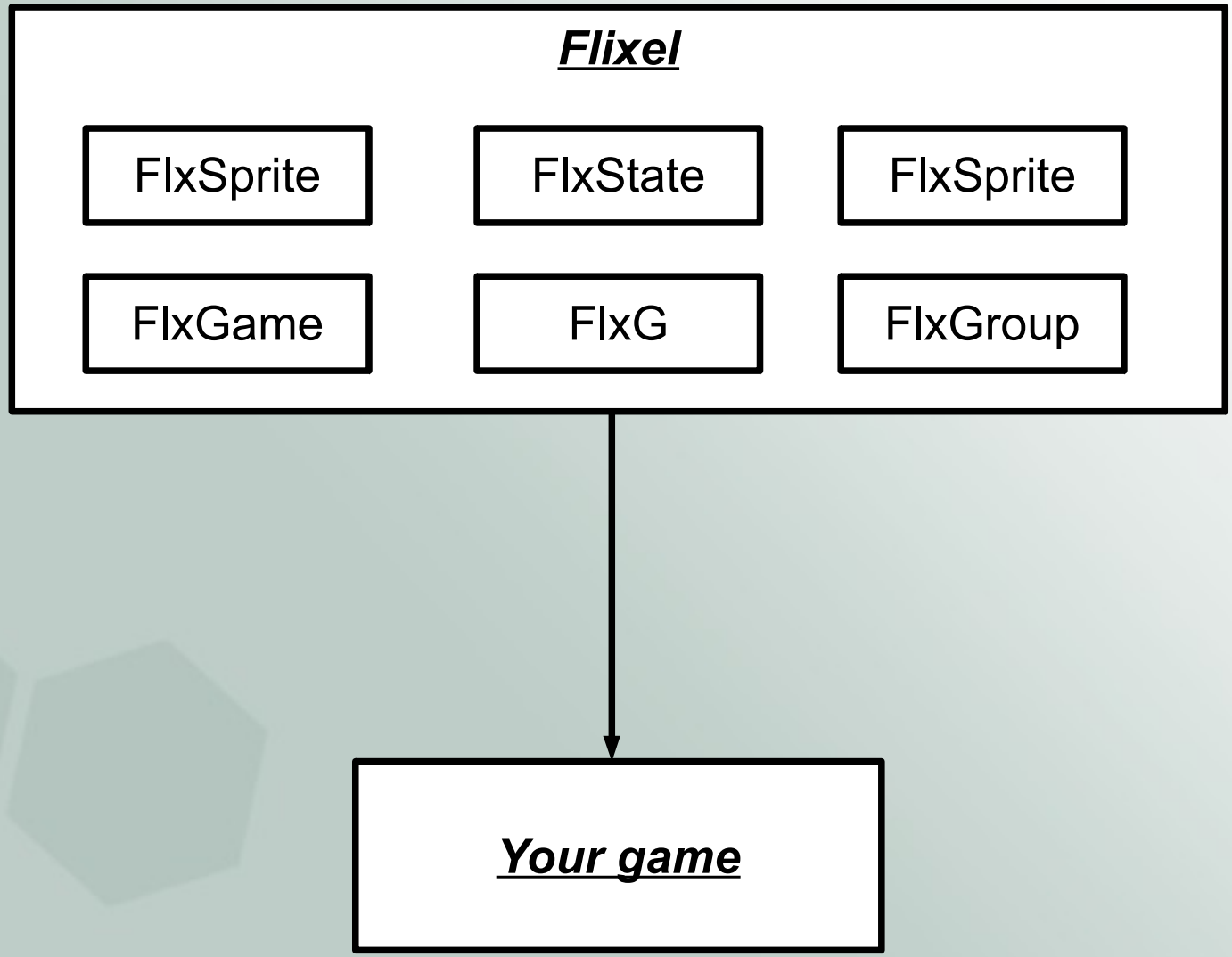
Fonctionnement de Flixel

Fonctionnement de Flixel

- Principes :
  - Propose un ensemble de **classes** dédiées à la création de **jeux vidéo** :
    - FlxSprite, FlxState, FlxGroup, FlxGame, FlxTileMap, etc.
  - Créer un jeu avec **Flixel** nécessite de créer des **classes** qui vont **hériter** de Flixel :
    - *Exemple :* une classe Ball dans un Breakout va hériter de FlxSprite
    - *Exemple :* la classe pour gérer le menu héritera de FlxState

Fonctionnement de Flixel

- Principes :

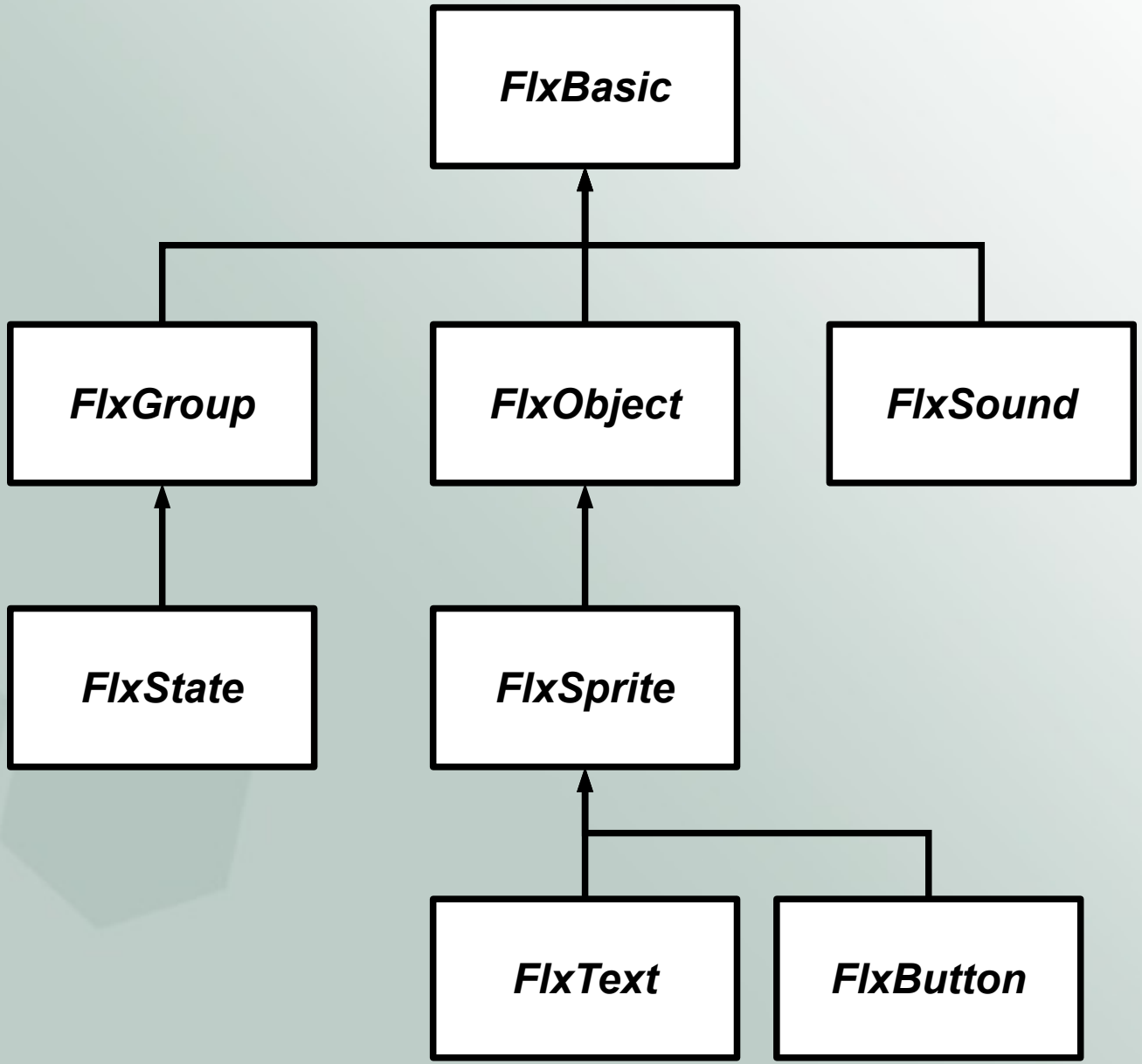


Fonctionnement de Flixel

- Principes :
  - Chaque classe de **Flixel** propose alors un certain nombre de **fonctions** :
    - *FlxSprite :*
      - Ajout d'animations
      - Physique (vitesse, friction, etc.)
      - Mise à jour à chaque frame
    - *FlxState :*
      - Ajout/suppression d'objets
      - Mise à jour à chaque frame

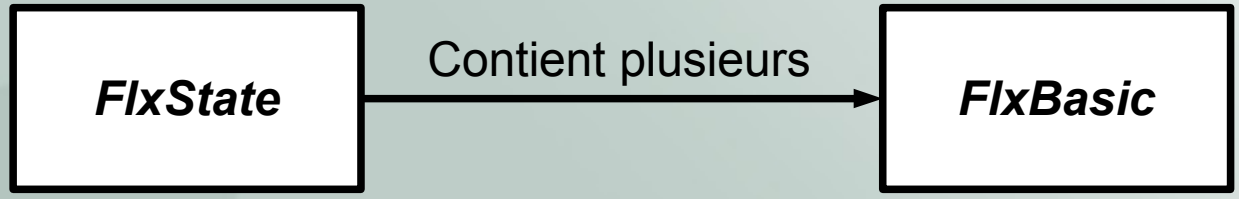
Fonctionnement de Flixel

- Principales classes de Flixel :



Fonctionnement de Flixel

- Fonctionnement :
  - « FlxState » représente un **état de jeu** (menu principal, monde principal, niveau 1, etc.)
  - Tous les objets que contient « FlxState » sont alors des objets de l'**état de jeu courant**
  - Tous objets **héritant** de « FlxBasic » (FlxSprite, FlxGroup, FlxObject, FlxButton, FlxText, etc.) peuvent être ajoutés à un « FlxState »



Fonctionnement de Flixel

- Fonctionnement:
  - A **chaque boucle moteur** « FlxState » appel toutes les méthodes « update » de chaque « FlxBasic » qu'il contient
  - Ainsi chaque objet du jeu possède sa **propre méthode de mise à jour** que l'on va pouvoir redéfinir et spécialiser

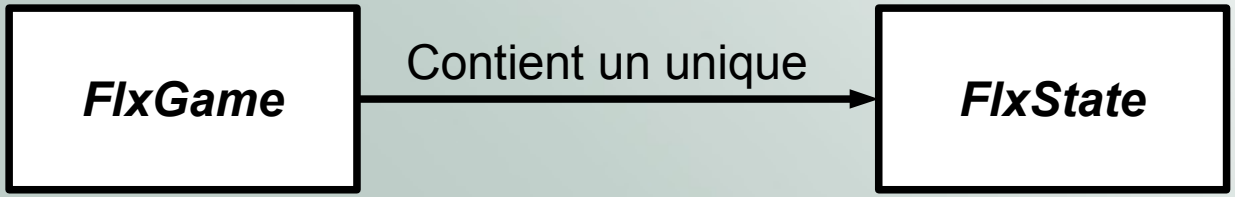
Utilisation basique de Flixel

Utilisation basique de Flixel

- Imports :
  - Chaque classe de Flixel **utilisée** doit être **importée** :
    - Utilisation de **l'opérateur** « import » et sélection du **chemin** de la classe
    - Doit être placé avant la déclaration de la classe
    - *Exemple :* `import org.flixel.FlxSprite;`

Utilisation basique de Flixel

- Point d'entrée de l'application :
  - Une classe **héritant** de « FlxGame »
  - Appel du **constructeur** de « FlxGame » en précisant le premier « FlxState » et la largeur et hauteur de la fenêtre du jeu
    - Utilisation du mot clé « super »



  - *Exemple :*

```
super(1024, 768, MenuState);
```

Utilisation basique de Flixel

- Point d'entrée de l'application :
  - *Exemple :*

```
public class Exemple extends FlxGame {

   public function Exemple(){
      super(800, 600, MenuState);
   }

}
```

- Création d'un état de jeu :
  - **Cœur de l'application**, c'est ici que nous ajoutons les objets et manipulons l'interface.
  - Une classe **héritant** de « FlxState »
  - Un « FlxState » pour le menu, chaque niveau, etc.
  - **Redéfinition** de la méthode « create » dans laquelle on **initialise** l'état de jeu :
    - Création des objets
    - Création de l'interface
    - Etc.
  - **Utilisation** de la méthode `add(gameObject)` pour **ajouter** un objet au jeu
    - Tout objet Flixel doit être ajouté de cette façon
    - Sinon rien ne sera affiché ni considéré

- Création d'un état de jeu :
  - *Exemple :*

```
public class MenuState extends FlxState {

    private var title:FlxText;

    override public function create():void {
        title = new FlxText(380, 300, 50,
        "Hello");
        title.setFormat(null, 16, 0x123fff);
        add(title);
    }
}
```

Utilisation basique de Flixel

- Mise à jour d'un état de jeu :
  - **Redéfinition** de la méthode « update » dans laquelle on **met à jour** l'état de jeu :
    - Mise à jour du score
    - Collisions entre objets
    - Mise à jour de l'interface
    - Etc.
  - **Utilisation** de l'opérateur « super » pour appeler la méthode « update » de FlxState
    - Met à jour les objets parents

Utilisation basique de Flixel

- Mise à jour d'un état de jeu :
  - *Exemple :*

```
override public function update():void {
   super.update();

   title.x = title.x + 10 ;
}
```

Utilisation basique de Flixel

- Création d'un Sprite :
  - **Tout objet** affiché à **l'écran** est un **sprite**
  - Création d'une classe **héritant** de « FlxSprite »
  - **Appel** du **constructeur** avec en arguments :
    - La position x
    - La position y
    - L'image elle même
  - **Utilisation** de la méthode `add(gameObject)` pour **ajouter** le sprite à l'état

Utilisation basique de Flixel

- Création d'un Sprite :
  - *Exemple :*

```
public class Avatar extends FlxSprite {
    [Embed(source = '../assets/pad.png')]
    protected var ImgPad:Class;

    public function Avatar() {
        super(100, 20, ImgPad);
    }
}
public class MenuState extends FlxState {
    private var avatar:Avatar;

    override public function create():void {
        avatar = new Avatar();
        add(avatar);
    }
}
```

Utilisation basique de Flixel

- Mise à jour d'un Sprite :
  - **Redéfinition** de la méthode « update » dans laquelle on **met à jour** le sprite :
    - Mise à jour de l'animation
    - Modifications de la vitesse
    - Etc.
  - Example :

```
override public function update():void {
    if (FlxG.keys.pressed("RIGHT")) {
        velocity.x = 10 ;
    }
}
```

Utilisation basique de Flixel

- Création de textes :
  - **Instancier** la classe « FlxText » permet de **créer** un **texte** d'une certaine taille, police, etc.
    - Fournir en paramètres :
      - Sa position « x » et « y »
      - La largeur allouée
      - Le texte à afficher
  - **Possibilité** d'utiliser la **méthode** « setFormat »
    - Avec en paramètres :
      - Une police (null si aucune)
      - Une taille de police
      - Une couleur sous la forme 0x000000

Utilisation basique de Flixel

- Création de textes :
  - *Exemple :*

```
var title:FlxText = new FlxText(150, 150, ''Welcome'');
title.setFormat(null, 16, 0x000aaa);
add(title);
```

Utilisation basique de Flixel

- Points et rectangles :
  - « FlxPoint » et « FlxRect » sont deux **classes** fournies par Flixel et permettant de **manipuler** des **points** et des **rectangles**
  - Cela permet de **créer** des **zones** particulières ou de définir des **points de spawns** par exemple
  - « FlxPoint » possède deux attributs « x » et « y »
  - « FlxRect » possède quatre attributs « x », « y », « width » et « height »

Utilisation basique de Flixel

- Points et rectangles :
  - *Exemple :*

```
//Créer une zone de 500 px par 500 px à la position 0,150
var safeArea:FlxRect = new FlxRect(0, 150, 500, 500);


//Créer un point au pixel 450 sur x et 200 sur y
var start:FlxPoint = new FlxPoint(450, 200);
```

Utilisation basique de Flixel

- Physique basique :
  - **Utilisation** des **attributs** présents dans « FlxObject » dont **hérite** « FlxSprite »
  - **Modifier** la **position** d'un **objet** se fait en modifiant ses **attributs** « x » et « y »
    - *Exemple :*

```
public class Avatar extends FlxSprite {
    [Embed(source = '../assets/pad.png')]
    protected var ImgPad:Class;

    public function Avatar() {
        super(100, 20, ImgPad);

        x = 500;
        y = 400;
    }
}
```

Utilisation basique de Flixel

- Physique basique :
  - Nous pouvons également modifier son **élasticité** :
    - Gère le rebond
    - 0 : pas de rebond, 1 : rebond sans perte de vitesse
  - Ou sa **vélocité** :
    - Gère la vitesse linéaire
    - Permet de modifier la vitesse en x ou y
    - En pixels par secondes
  - Accélération, mass, etc.

- Physique basique :
  - *Example :*

```
public class Avatar extends FlxSprite {
    [Embed(source = '../assets/pad.png')]
    protected var ImgPad:Class;

    public function Avatar() {
        super(100, 20, ImgPad);

        elasticity = 0.2;
        velocity.x = 15;
    }
}
```

Utilisation basique de Flixel

- Gestion du clavier :
  - **Utilisation** de la classe **static** « FlxG » (à importer donc)
  - **Appel** de la fonction « pressed(key:String) » de **l'objet** « keys » présent dans FlxG :
    - Fournir le nom de la touche
    - La fonction renvoie « true » si la touche est effectivement enfoncée, « false » sinon
  - **Appel** de la fonction « justReleased(key:String) » de **l'objet** « keys » présent dans FlxG :
    - Fournir le nom de la touche
    - La fonction renvoie « true » si la touche a été relâchée, « false » sinon

Utilisation basique de Flixel

- Gestion du clavier :
  - *Exemple :*

```
override public function update():void
{
   super.update();

   if (FlxG.keys.pressed("LEFT")) {
      player.x = player.x - 5 ;
   }
}
```

Utilisation basique de Flixel

- Gestion de la souris :
  - **Utilisation** de la classe static « FlxG » (à importer donc)
  - **Appel** de la fonction « pressed() » de **l'objet** « mouse » présent dans FlxG :
    - La fonction renvoi « true » si le bouton 1 de la souris est effectivement enfoncée, « false » sinon
  - **Appel** de la fonction « justReleased() » de **l'objet** « mouse » présent dans FlxG :
    - La fonction renvoie « true » si la touche a été relâchée, « false » sinon

- Gestion de la souris :
  - *Exemple :*

```
override public function update():void
{
   super.update();

   if (FlxG.mouse.pressed()) {
      if (player.isAlive()) {
         player.destroy();
      }
   }
}
```

Utilisation basique de Flixel

- Gestion du scrolling :
  - **Définition** de la **taille du monde** grâce à l'attribut « worldBounds » de la classe « FlxG »
    - Affectation d'un nouveau « FlxRect » à l'attribut « worldBounds »
  - **Utilisation** de **l'objet** « camera » présent dans la classe « FlxG »
  - **Appel** de la **fonction** « setBounds » pour définir la zone où la caméra peut aller
  - Appel de la fonction « follow » pour **spécifier** l'objet **responsable** du scrolling
  - L'attribut « scrollFactor » permet de définir à quel point un objet **subit** le **scrolling**. Utile pour créer le **HUD** (qui ne scroll pas) ou pour faire du **parallax**

Utilisation basique de Flixel

- Gestion du scrolling :
  - *Exemple :*

```
override public function create():void
{
    FlxG.worldBounds = new FlxRect(0, 0, 2048, 768) ;
    FlxG.camera.setBounds(0, 0, 2048, 768);
    FlxG.camera.follow(player);
}
```

- Gestion des groupes :
  - Création d'un **groupe** « FlxGroup », fonctionne de la même façon qu'un « FlxState »
  - Ajout du **groupe** dans l'état de jeu
  - **Ajout** des **objets** dans le groupe avec la méthode « add »
  - *Exemple :*

```
enemies = new FlxGroup();
add(enemies);
enemies.add(new Enemy(10, 50));
enemies.add(new Enemy(400, 150));
enemies.add(new Enemy(1000, 480));
enemies.add(new Enemy(800, 368));
enemies.add(new Enemy(750, 845));
```

- Gestion des groupes :
  - **Suppression** des **objets** dans le groupe avec la méthode « remove »
  - *Exemple :*

```
var enemy:Enemy = new Enemy(10, 50);
enemies.add(enemy);
enemies.add(new Enemy(750, 845));

enemies.remove(enemy);
```

  - **Suppression** de **tous les objets** du groupe avec la méthode « clear »
  - *Exemple :*

```
enemies.clear();
```

- Gestion des destructions/créations :
  - Il est **possible** de « **tuer** » et « **réanimer** » des **objets** avec les méthodes « kill » et « revive »
  - *Exemple :*

```
var bonus:FlxSprite = new
FlxSprite(50,50,ImgBonus);
add(bonus);


bonus.kill(); //L'objet disparaît et
n'est plus considéré par le jeu (mais
reste présent en mémoire)


bonus.revive(); //L'objet réapparaît et
fait de nouveau partie intégrante du jeu
```

- Gestion des destructions/créations :
  - A tout moment nous pouvons savoir combien d'objets sont présents dans le jeu
  - *Exemple :*

```
var bonus:FlxSprite = new FlxSprite(50,50,ImgBonus);
add(bonus);
bonus = new FlxSprite(150,150,ImgBonus);
add(bonus);

countLiving(); //Renvoie 2
countDead(); //Renvoie 0
bonus.kill();
countLiving(); //Renvoie 1
countDead(); //Renvoie 1
```

- Gestion des collisions :
  - **Utilisation** de la **méthode** « collide » présente dans la **classe** « FlxG »
    - **Appeler** cette méthode à chaque frame pour **effectuer** les **calculs de physique**
    - **Spécifier** les deux **groupes** ou **objets** qui doivent rentrer en collisions
  - **Utilisation** de la **méthode** « overlap » présente dans « FlxG »
    - N'effectue **aucuns calculs de physique**
    - **Spécifier** les deux **groupes** ou **objets** dont la collision doit être testée
    - **Renvoie** `true` si la collision à eu lieu, `false` sinon

Utilisation basique de Flixel

- Gestion des collisions :
  - *Exemple :*

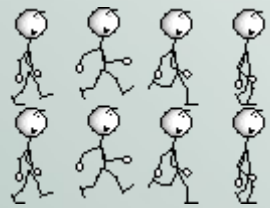
```
override public function update():void {
   super.update();

   FlxG.collide(paddle, ball);
   FlxG.collide(ball, walls);

   if (FlxG.overlap(ball,bonus) == true)
   {
      bonus.kill();
   }
}
```

Utilisation basique de Flixel

- Gestion des animations :
  - Pour les **animations** nous allons utiliser des **spritesheets** :
    - **Suite des frames** des animations rassemblées en une seule image
    - Toutes les **frames** font la même taille
    - Ainsi un **spritesheet** de 1000 pixels par 1000 pixels possédant 10 colonnes et 10 lignes de frames aura des frames de 10 pixels par 10 pixels
  - *Exemple :*
    - Animation de marche
    - Un spritesheet de 144x98
    - 2 lignes et 4 colonnes
    - 8 frames de 36x49



- Gestion des animations :
  - **Utilisation** de la **fonction** « loadGraphic » fournie par la classe « FlxSprite pour créer un sprite animé
    - Fournir le **nom** du spritesheet
    - Préciser qu'il s'agit d'une animation
    - Préciser s'il faut **générer les frames inverses** de l'animation (évite de faire l'animation pour marcher vers la gauche et la droite)
    - Préciser la **taille** d'une **frame** (et pas de l'image entière)

Utilisation basique de Flixel

- Gestion des animations :
  - **Utilisation** de la **fonction** « addAnimation » fournie par la classe « FlxSprite » pour ajouter une animation à un sprite animé
    - Fournir le nom de l'animation
    - Fournir la **suite des frames** composant l'animation
    - Préciser la **vitesse** de l'animation
    - Spécifier si l'animation **boucle** ou non
  - **Utilisation** de **l'attribut** « facing » pour préciser le **sens** des frames de l'animation
    - Peut prendre la constante LEFT ou RIGHT (fournie par « FlxSprite »)

- Gestion des animations :
  - *Exemple :*

```
public function Avatar() {
    super(200, 200);
    loadGraphic(ImgAvatar, true, true,
    36, 49);
    addAnimation("idle", [0], 10);
    addAnimation("walk", [0, 1, 2], 10,
    true);
    play("idle");
}
```

Utilisation basique de Flixel

- Gestion des animations :
  - *Exemple :*

```
override public function update():void {
    if (FlxG.keys.pressed("LEFT")) {
        facing = LEFT;
        play("walk");
        velocity.x = -100;
    }
}
```

Utilisation basique de Flixel

- *Flixel site web*
- Liste des classes et fonctions
- Ressources en cas de problèmes