




Plan

2

- **Objectifs** 
- Premier code
- Chargement et affichage d'images
- Map navigable à base de Tile
- Contrôle d'une entité orientée plateforme
- Un '*EntityPlatform*' sur une '*MapTile*'


Objectifs

3

- Découvrir pas à pas les principaux modules
- Comprendre le fonctionnement de ce qui est utilisé
- Illustrations avec des exemples détaillés
- N'utiliser que ce qui est nécessaire
 - ▣ Savoir ce qu'il faut choisir
 - ▣ Avoir les bon reflexes
 - ▣ Eliminer le superflus
- Arriver à créer un petit jeu par la suite

Plan

4

- Objectifs
- **Premier code** 
- Chargement et affichage d'images
- Map navigable à base de Tile
- Contrôle d'une entité orientée plateforme
- Un '*EntityPlatform*' sur une '*MapTile*'

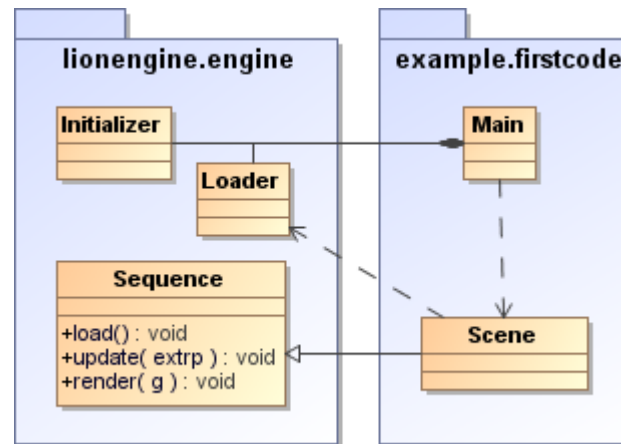
Premier code

5

- Préparation de l'environnement
 - ▣ Création d'un projet Java 7+ dans votre IDE favori
 - ▣ Ajout des librairies `LionEngine_*_x.y.z.jar` au projet
 - ▣ Ajout d'une classe Main de lancement
 - `public static void main(String args[])`
 - ▣ Ajout d'une classe Scene représentant la boucle du jeu
 - Héritage de `com.b3dgs.lionengine.engine.Sequence`
- Test et affichage d'un écran noir

Premier code

6



Premier code

7

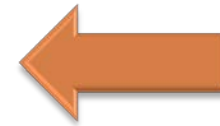


Byron 3D Games Studio - LionEngine v6.0.0

Plan

8

- Objectifs
- Premier code
- **Chargement et affichage d'images**
- Map navigable à base de Tile
- Contrôle d'une entité orientée plateforme
- Un '*EntityPlatform*' sur une '*MapTile*'



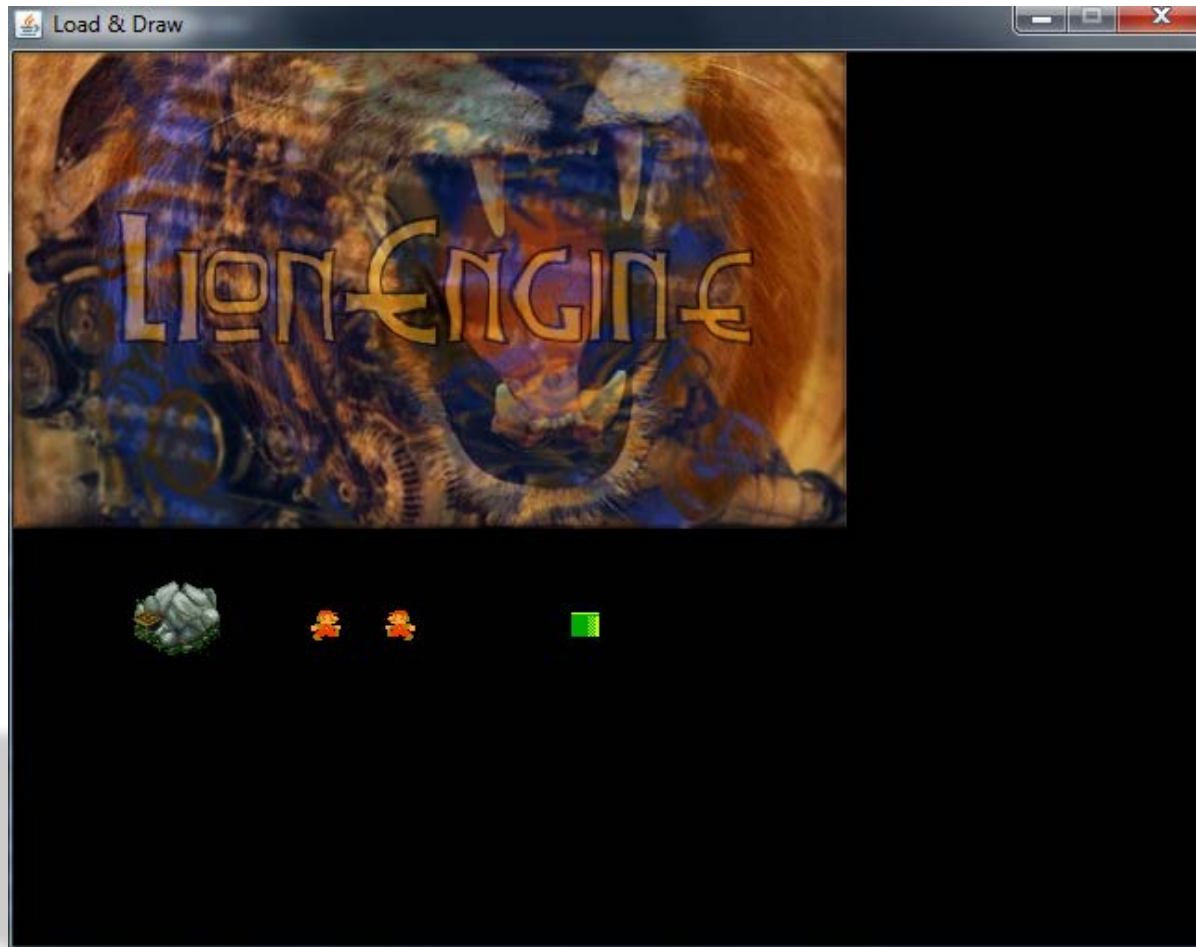
Chargement et affichage d'images

9

- Reprise de la base fonctionnelle précédente
- Utilisation des principaux types d'image
 - ▣ `Image`, `Sprite`, `SpriteAnimated`, `SpriteTiled`
- Chargement des ressources depuis un fichier
 - ▣ Appel à `Media.get(filename)`
- Préparation et affichage de la surface
 - ▣ Image simple, sprite, sprite animé, tile défilant

Chargement et affichage d'images

10



Chargement et affichage d'images

11

- Le chargement des medias est relatif à un dossier
 - ▣ Défini par: `Engine.start("Name", Version.create(1, 0, 0), "resources");`
 - Ici le dossier racine est « resources »
 - ▣ Il s'effectue à l'aide de `Media.get(name)`
 - La chaine retournée sera le chemin complet du fichier
 - Exemple: pour avoir « ../resources/img.png »
 - `String path = Media.get("img.png");`
 - Exemple: pour avoir « ../resources/snd/audio.wav »
 - `String path = Media.get("snd", "audio.wav");`

Chargement et affichage d'images

12

□ Chargement d'un sprite animé:

▣ `Drawable.loadSpriteAnimated(Media.get("animation.png"), 7, 1);`

- Il y a 7 frames horizontales, sur une ligne

□ Si il y avait eu une 2^{ème} ligne contenant des frames

▣ `Drawable.loadSpriteAnimated(Media.get("animation.png"), 7, 2);`

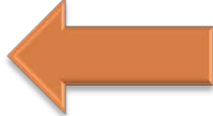
□ Chargement d'un tile:

▣ `Drawable.loadSpriteTiled(Media.get("tilesheet.png"), 16, 16);`

- La taille d'un tile dans ce cas est de 16px*16px

Plan

13

- Objectifs
- Premier code
- Chargement et affichage d'images
- **Map navigable à base de Tile** 
- Contrôle d'une entité orientée plateforme
- Un '*EntityPlatform*' sur une '*MapTile*'

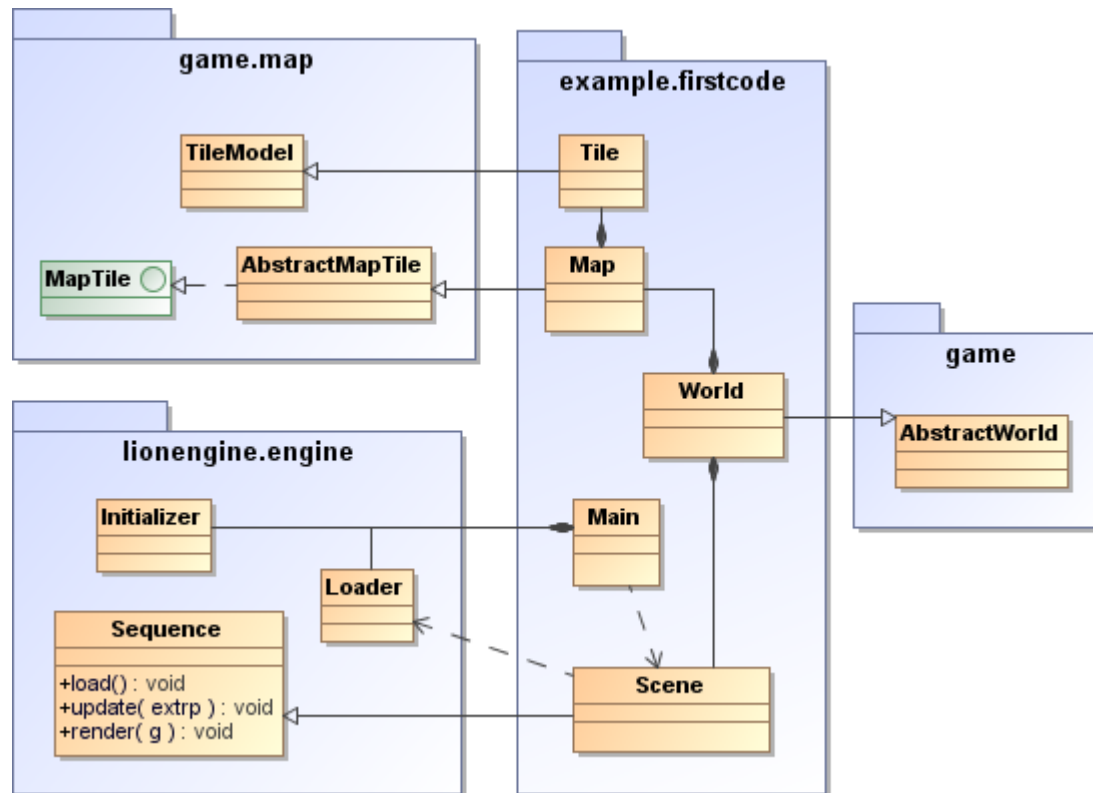
Map navigable à base de Tile

14

- Découverte de la première base abstraite '*World*'
 - ▣ Gérée par la '*scene*'
 - ▣ Utilisation du module « Platform »
- Implémentation de base d'une map à base de tile
 - ▣ Utilisation de '*TileGame*' et '*MapTilePlatform*'
- Utilisation d'une camera, gestion de ses mouvements
- Affichage de la map en utilisant la camera

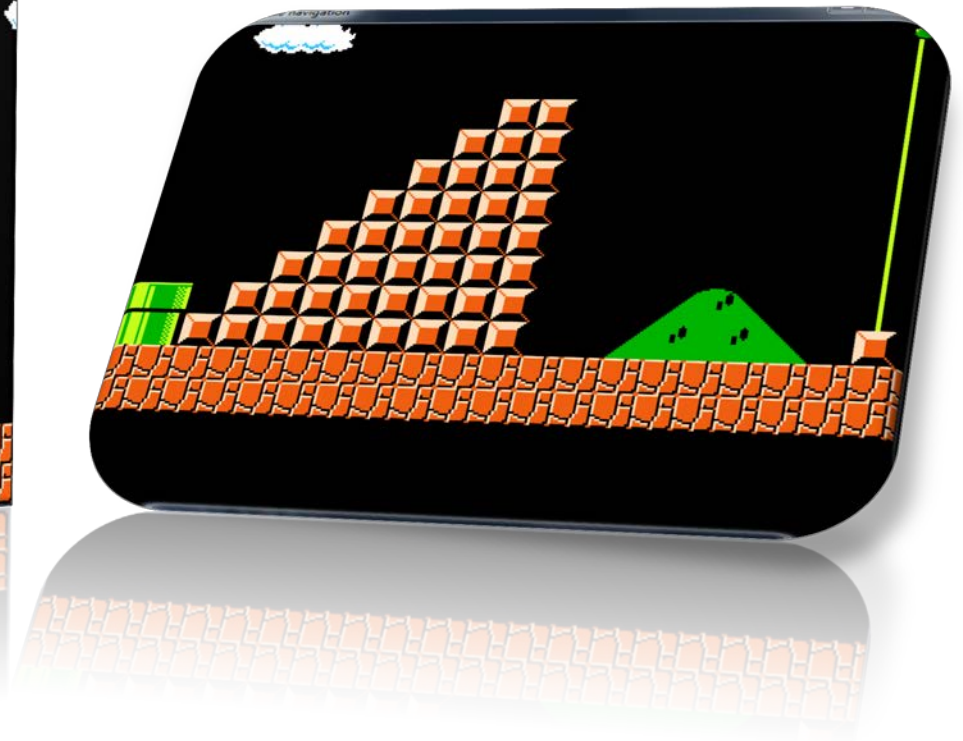
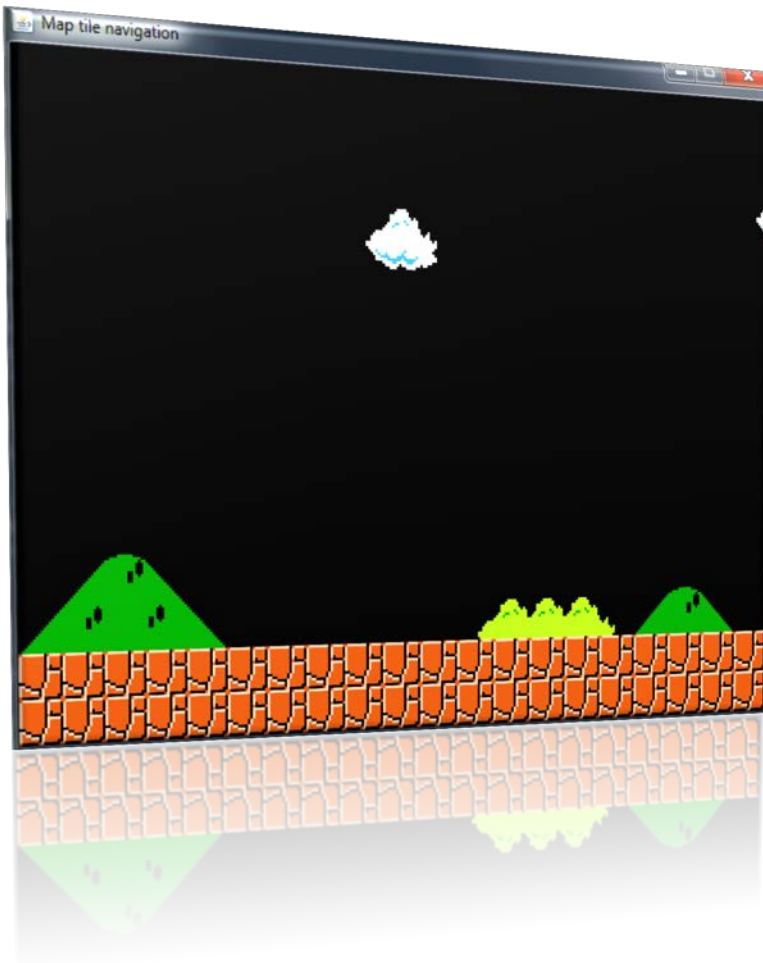
Map navigable à base de Tile

15



Map navigable à base de Tile

16



Byron 3D Games Studio - LionEngine v6.0.0

Map navigable à base de Tile

17

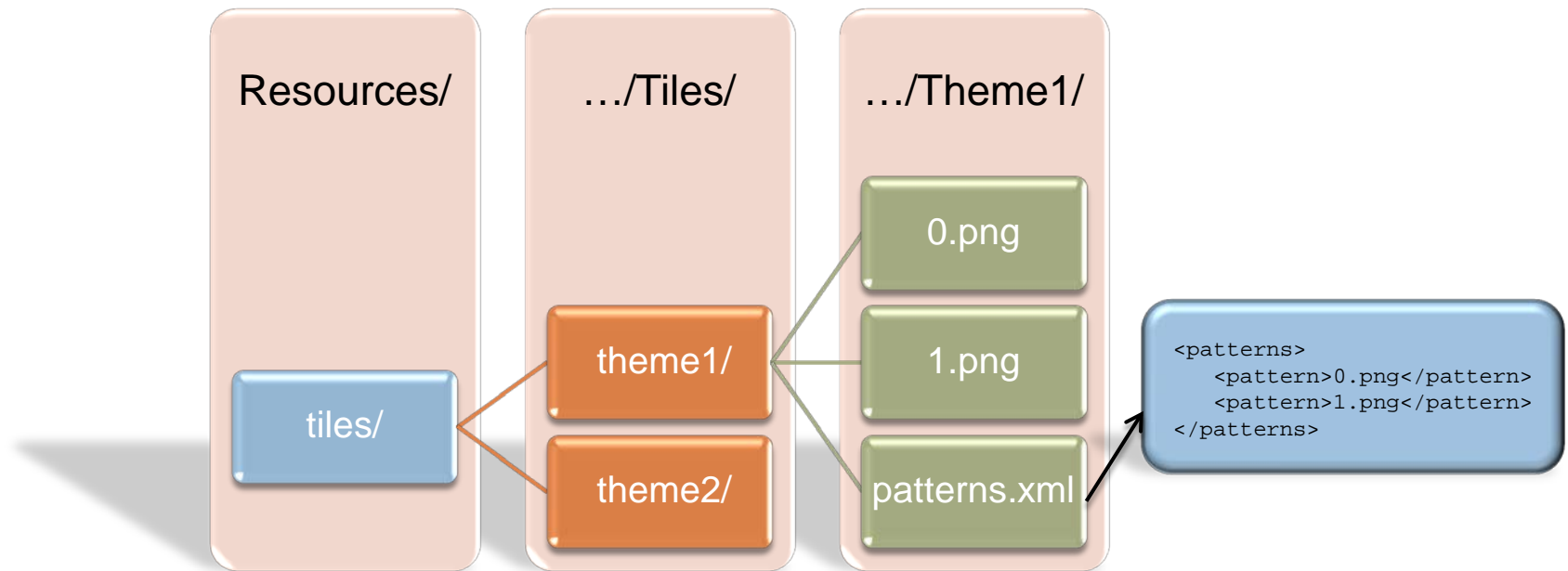
- La définition d'un '*World*' permet
 - ▣ D'avoir une boucle dédiée pour plusieurs types de jeux
 - ▣ La gestion de la sauvegarde/chargement facilement
 - ▣ De garder la séquence pour un rendu en aval/amont

- De base, une MapTile a besoin
 - ▣ De la taille d'un Tile
 - ▣ D'un dossier contenant le(s) tilesheet(s) + fichier d'index
 - Le fichier 'patterns.xml' permet de définir la liste des tilesheets à charger automatiquement

Map navigable à base de Tile

18

- L'architecture impose une structure de rangement
 - ▣ Un dossier, placé dans le dossier des ressources, doit contenir un dossier par thème, eux même contenant la liste des tilesheets avec le fichier 'count'
 - ▣ Sans ce fichier, toutes les images de type .png seront chargées



Map navigable à base de Tile

19

- La partie '*update*' de '*World*'
 - ▣ Gestion des états des entrées (clavier & souris)
 - ▣ Application du mouvement en conséquence
 - ▣ Mise à jour des vitesses de déplacement sur la camera

- La partie '*render*' de '*World*'
 - ▣ Affiche la map en fonction de la camera
 - ▣ La camera sera alors le point de vue sur la map

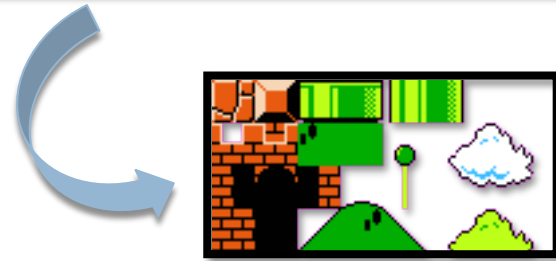
Map navigable à base de Tile

20

- Pour automatiser la génération d'une map
 - ▣ Utilisation d'un levelrip (image complète du niveau)
 - ▣ '*LevelRipConverter*' se charge de convertir le niveau
 - En utilisant le levelrip
 - A partir du dossier des tiles
 - Reconstitue le niveau en données de tile
 - Possibilité de sauvegarder le résultat dans un fichier
 - ▣ '*TileExtractor*' utilise le procédé inverse (extraie les tiles)
 - Enregistre dans une tilesheet chaque tile unique

21

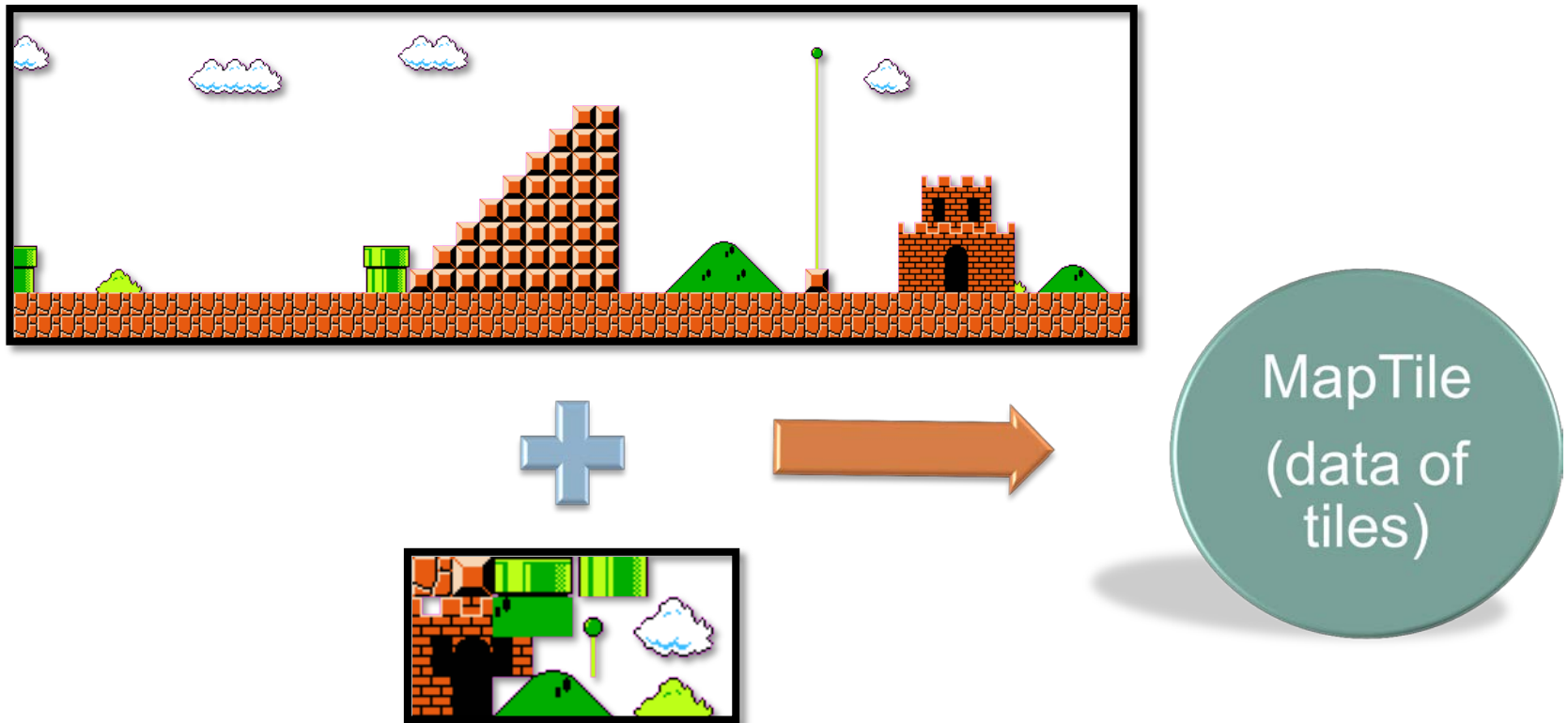
-
- A Super Mario Bros. level scene. The ground is made of brown brick blocks. On the left, there is a green pipe. To its right is a small green hill. Further right is a tall brick wall made of brown bricks. To the right of the wall is another green hill. Further right is a green flag on a pole. To the right of the flag is a brick tower with a black arched entrance. To the right of the tower is a small green hill. The sky is white with four blue clouds.



Map navigable à base de Tile

22

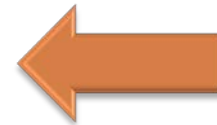
- *'LevelRipConverter'* en action:



Plan

23

- Objectifs
- Premier code
- Chargement et affichage d'images
- Map navigable à base de Tile
- **Contrôle d'entité orientée plateforme**
- Un '*EntityPlatform*' sur une '*MapTile*'



Contrôle d'entité orientée plateforme

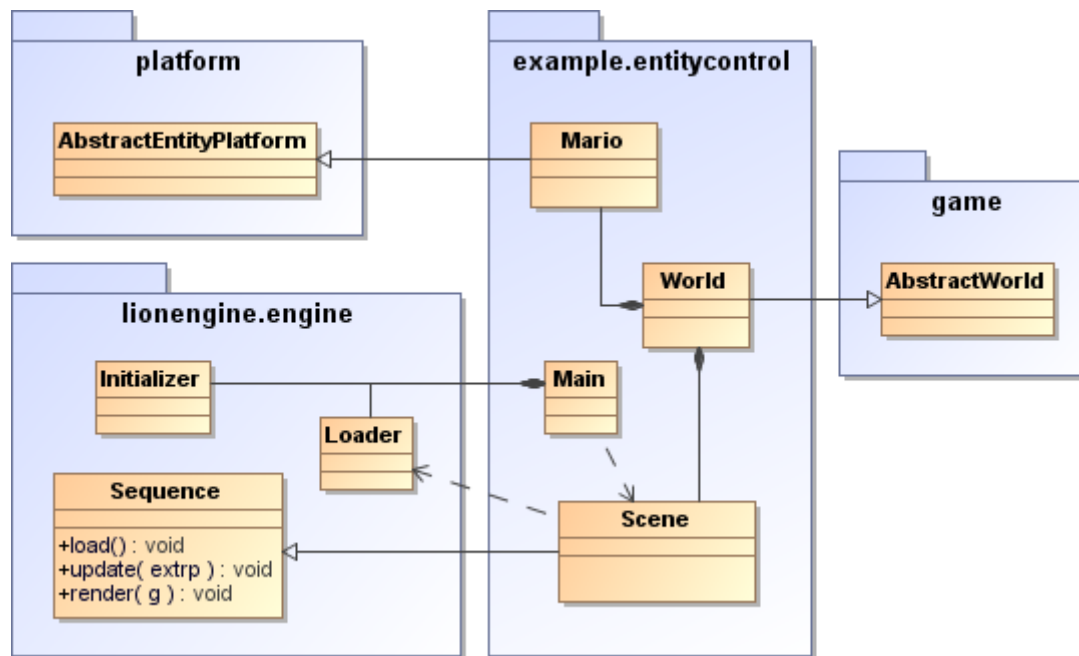
24

- Découverte de la base abstraite 'EntityPlatform'
 - ▣ Architecture incorporée pour faciliter le développement
 - ▣ Routine d'update découpée en plusieurs parties

- Implémentation d'un héros contrôlable au clavier
 - ▣ Gestion des touches
 - ▣ Manipulation de la vitesse de déplacement
 - ▣ Configuration de sa gravité
 - ▣ Mise à jour des animations en conséquence

Contrôle d'entité orientée plateforme

25



Contrôle d'entité orientée plateforme

26



Byron 3D Games Studio - LionEngine v6.0.0

Contrôle d'entité orientée plateforme

27

- Notre héro héritera de '*EntityPlatform*'
 - ▣ Il présente donc plusieurs caractéristiques, dont être
 - Un corps (potentiellement soumis à la gravité)
 - Localisable (peut donc se déplacer)
 - Sujet à la collision avec d'autres entités de ce type
 - Configurable depuis un fichier externe
 - Un animateur (capable de jouer une animation sur le sprite)
 - ▣ Pour l'instancier, il lui faudra un '*SetupGame*'
 - Contient la surface et le fichier de configuration

Contrôle d'entité orientée plateforme

28

- L'implémentation respectera l'architecture
 - ▣ `handleActions`; effectuer les actions suite aux inputs
 - ▣ `handleMovements`; application des actions
 - ▣ `handleCollisions`; vérifications des collisions
 - ▣ `handleAnimations`; met à jour l'animation de l'entité

- Ces fonctions sont appelées séquentiellement
 - ▣ Suivre ce découpage facilitera le développement

Contrôle d'entité orientée plateforme

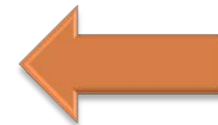
29

- Concernant les mouvements
 - ▣ La pesanteur terrestre est appliquée (si masse > 0)
 - ▣ Il est possible de stopper à tout moment la gravité
 - ▣ Il est possible de soumettre le corps à plusieurs forces
 - L'une gérant le déplacement horizontal
 - L'une gérant le saut
 - ...
 - ▣ Les forces sont totalement contrôlables
 - Effets de courbes incurvées
 - Peuvent s'additionner entre elles (progressivement ou non)

Plan

30

- Objectifs
- Premier code
- Chargement et affichage d'images
- Map navigable à base de Tile
- Contrôle d'entité orientée plateforme
- **Un '*EntityPlatform*' sur une '*MapTile*'**



Un 'EntityPlatform' sur une 'MapTile'

31

- Reprise des deux chapitres précédents
 - ▣ Combinaison de l'affichage de la map avec camera, et la gestion du joueur
 - La camera suivra désormais le joueur
 - Elle héritera maintenant de '*CameraPlatform*'
- Le rendu du joueur est effectué sur la camera
 - ▣ Ainsi la map et le joueur auront le même référentiel
 - ▣ La map défilera selon les mouvements du joueur
 - ▣ La camera ne dépassera pas les bordures fixées

Un '*EntityPlatform*' sur une '*MapTile*'

32

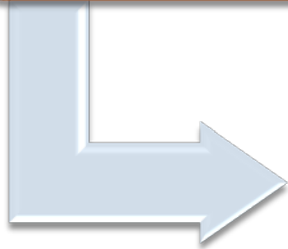
- La gestion des collisions est effectuée
 - ▣ En amont côté Tile
 - Récupère son type de collision depuis un fichier externe
 - Définit les points de collision en fonction d'une localisation
 - Dépend du type de collision du tile
 - ▣ En aval côté Entité
 - Teste quel est le premier tile traversé ayant une collision
 - Se place sur le point de collision du tile correspondant
 - Gère plusieurs collisions (sol, zones bloquantes...)

Un '*EntityPlatform*' sur une '*MapTile*'

33

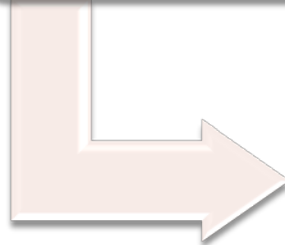
Check

- Recherche le premier tile ayant une collision suite à une intersection



Get

- Récupère le tile concerné, et recherche le point de collision associé



Apply

- L'entité se place sur le point de collision

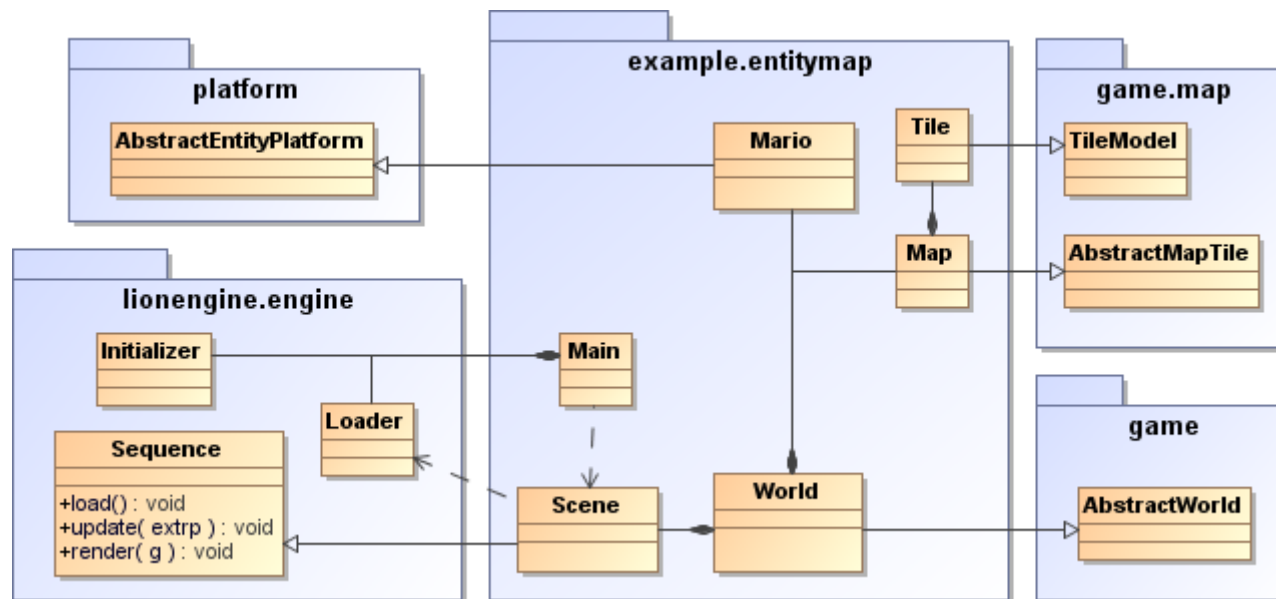
Un '*EntityPlatform*' sur une '*MapTile*'

34

- `getCollisionTile` permet de récupérer le tile '*raycasté*'
 - ▣ Référentiel de l'entité (modulo un offset ajustable)
 - ▣ Possibilité de filtrer le type de collision recherché
- Récupération du point de collision du tile
 - ▣ L'appliquer à l'entité si il en existe un
- L'entité a récupéré le tile touché, et sa collision
 - ▣ Le processus est maintenant complet

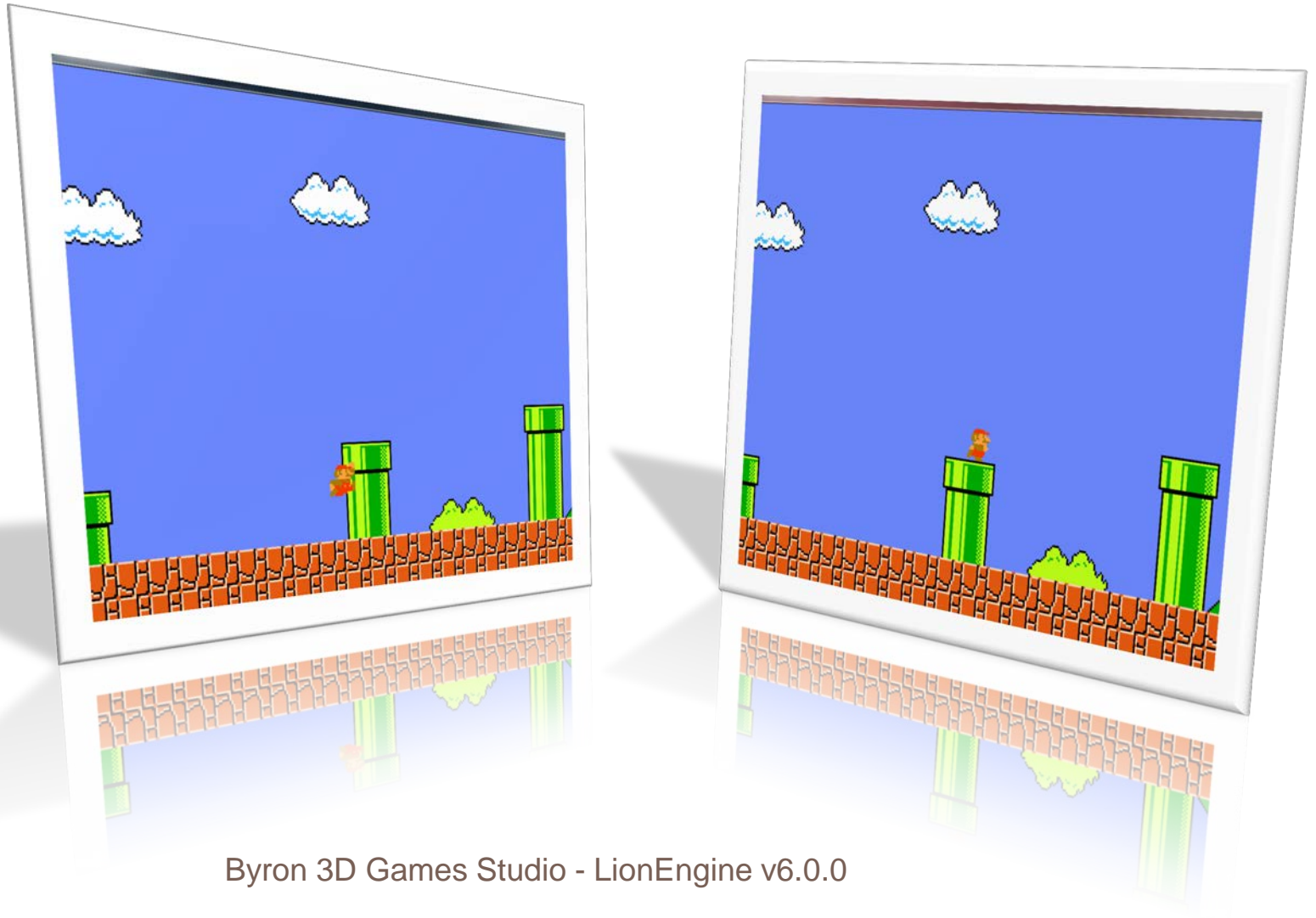
Un '*EntityPlatform*' sur une '*MapTile*'

35



Un '*EntityPlatform*' sur une '*MapTile*'

36



Byron 3D Games Studio - LionEngine v6.0.0

Un '*EntityPlatform*' sur une '*MapTile*'

37

- Pour ajouter une entité autonome
 - ▣ Créer une classe abstraite (vous servant de base)
 - ▣ Gérer les éléments standard à tout type d'entité
 - ▣ Créer une classe spécifique pour le héros contrôlable
 - ▣ Créer une classe spécifique pour l'ennemi

- Pour gérer de multiples entités
 - ▣ Créer un handler héritant de '*HandlerEntityPlatform*'
 - ▣ L'instancier dans '*World*', et gérer sa mise à jour

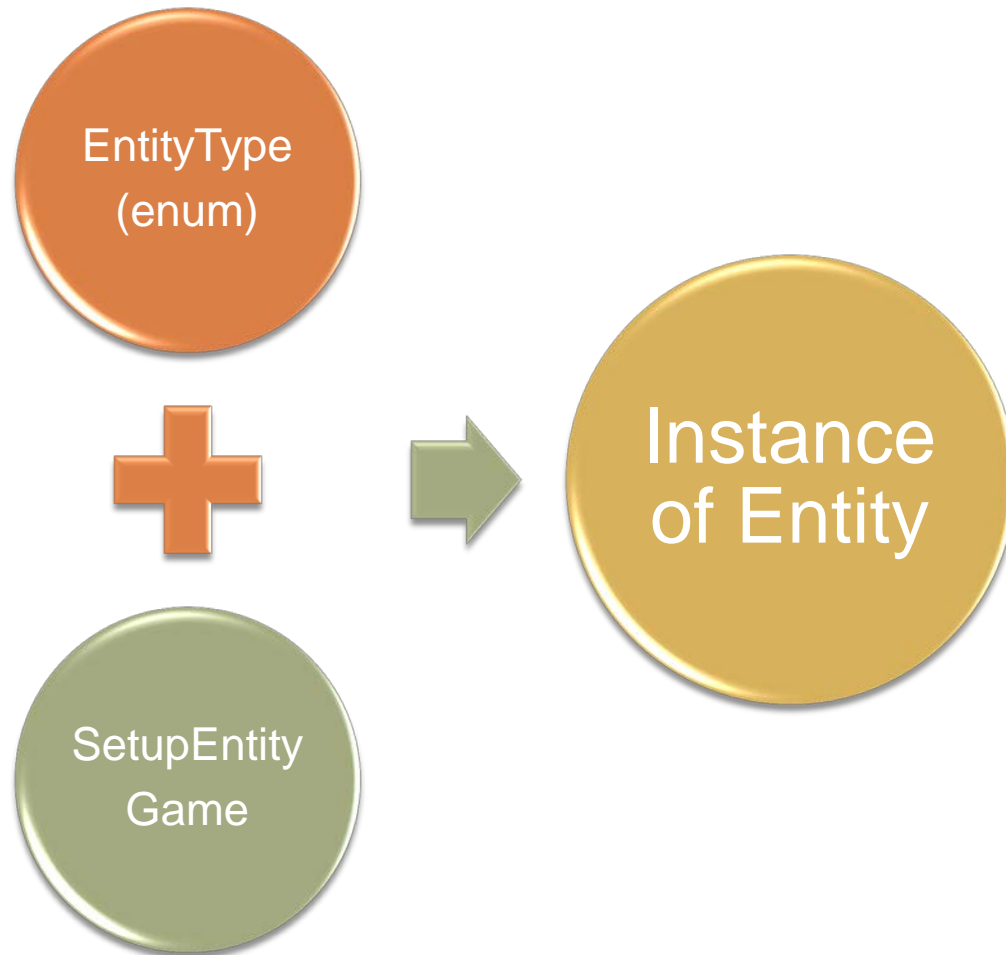
Un '*EntityPlatform*' sur une '*MapTile*'

38

- Il est conseillé de regrouper les instanciations en:
 - ▣ Faisant appel à un '*factory*' dédié
 - `FactoryEntityGame`
 - ▣ Définissant un '*enum*' contenant tous les types d'entités
- Ce dernier sera chargé de préparer les '*Setup*'
 - ▣ Définit une et unique fois les ressources des entités
 - ▣ Permet de créer des entités en partageant les ressources (graphiques et données externes)

Un '*EntityPlatform*' sur une '*MapTile*'


39




Un '*EntityPlatform*' sur une '*MapTile*'

40

Définition des types d'entités dans un *enum*



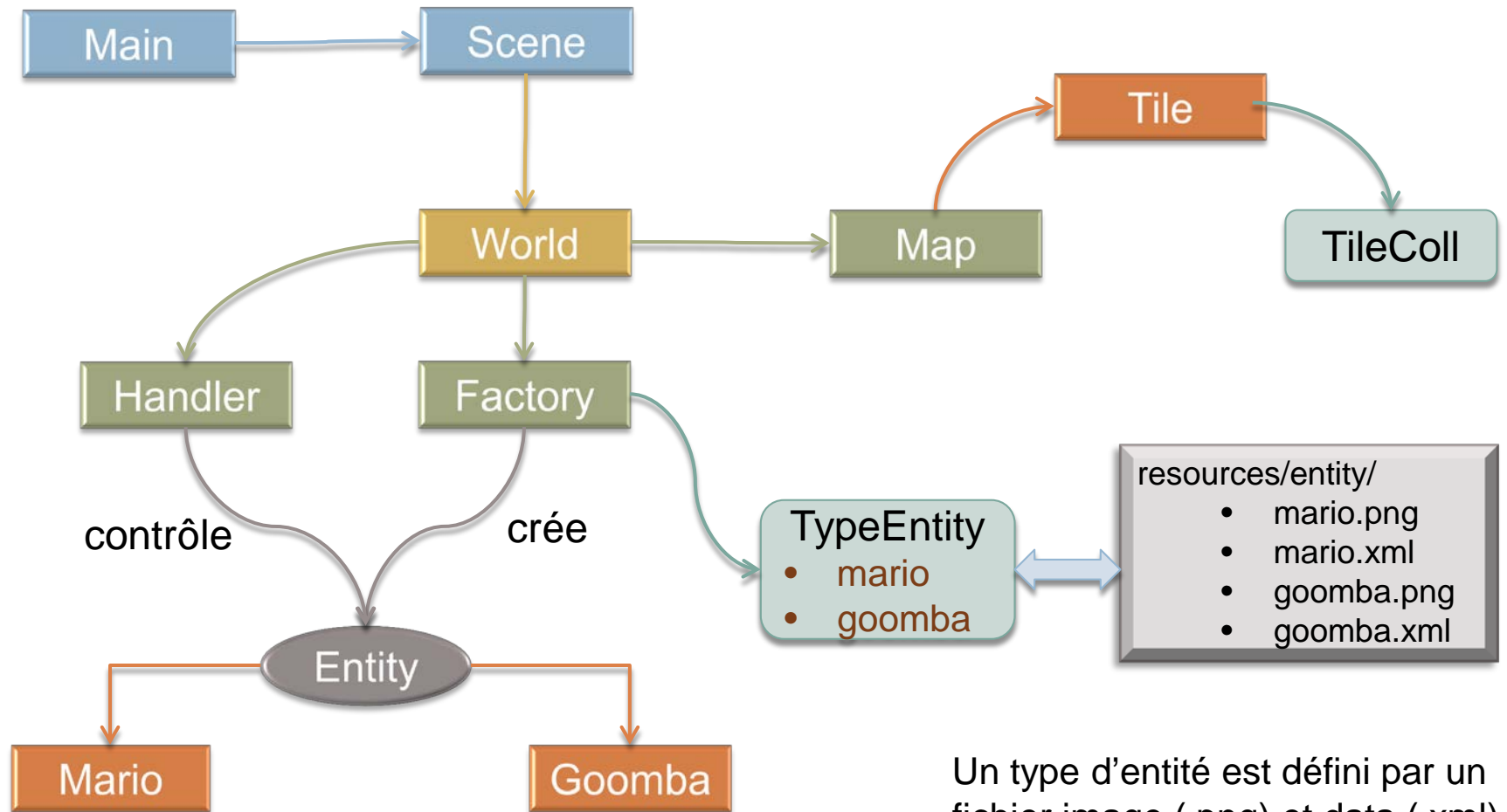
Chargement des ressources pour chaque entité (en correspondance avec son chemin dans un dossier), stocké dans le *Factory* (Setup associé à l'*enum*)



Le *Factory* fournit les instances des entités à partir de fonctions, ou de manière générique depuis leur type (*enum*), en récupérant le setup associé à l'*enum* et en créant l'instance

Un 'EntityPlatform' sur une 'MapTile'

41



Un type d'entité est défini par un fichier image (.png) et data (.xml)

Un 'EntityPlatform' sur une 'MapTile'

42

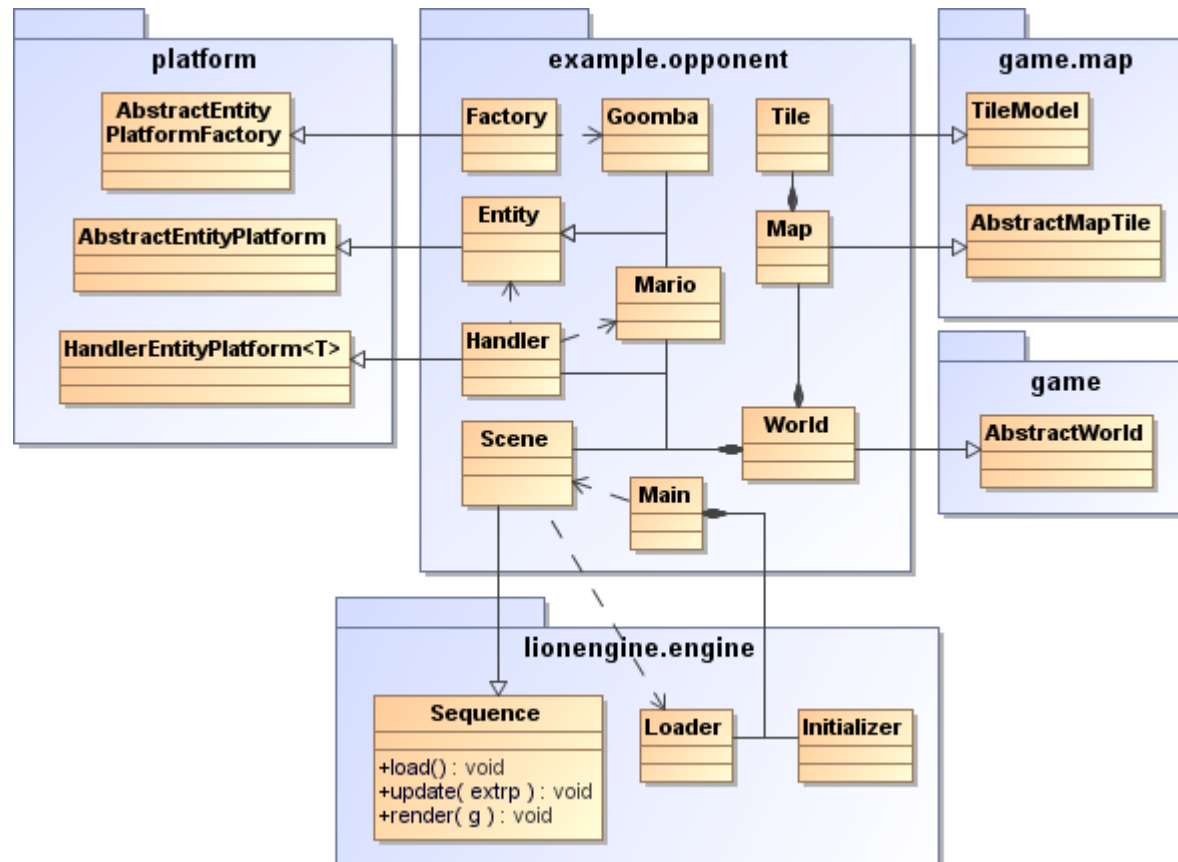
□ Configuration minimale de l'entité (entity.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<entity name="name" surface="surface.png">
    <frames horizontal="3" vertical="1"/>
    <size width="16" height="16"/>
</entity>
```

- Le fichier de configuration (xml) et la surface (png) doivent être dans le même dossier

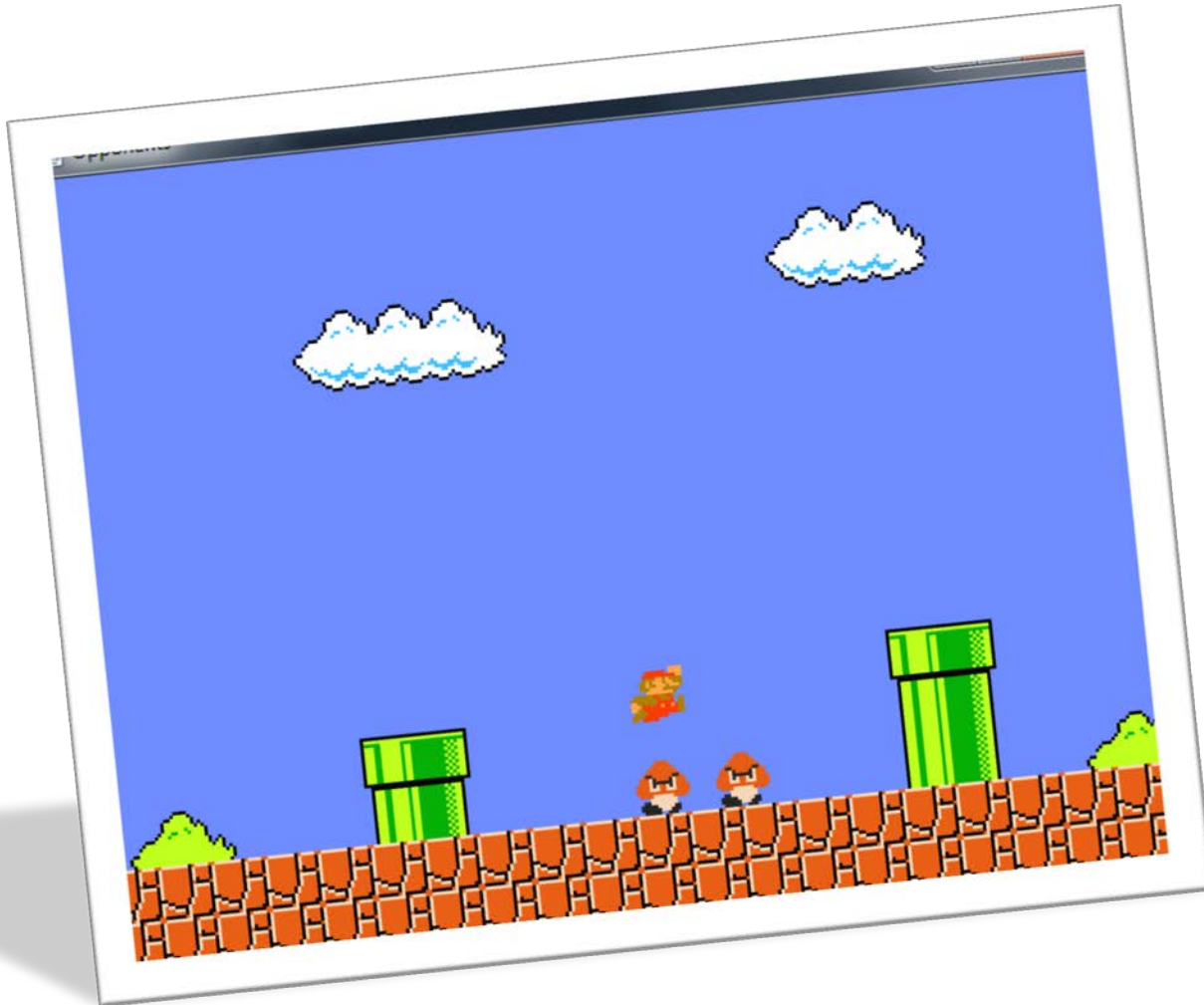
Un '*EntityPlatform*' sur une '*MapTile*'

43



Un '*EntityPlatform*' sur une '*MapTile*'

44



Byron 3D Games Studio - LionEngine v6.0.0