Sorbonne Université
Faculté des Sciences et Ingénierie

PSTL

4I508

Programmation Fonctionnelle Réactive en Clojure

Étudiants
Clément Busschaert
Antoine Meunier

Superviseur
Frédéric Peschanski

26 mai 2018

Table des matières

Chapitre 1

Le projet

Ce projet s'inscrit dans la continuité de précédents PSTLs. L'objectif principal est d'étendre les fonctionnalités actuelles vers une API réactive.

YAW (Yet Another World) est une bibliothèque Java, développée au fil de plusieurs PSTL, de manipulation et affichage d'objets 3D avec une surcouche Clojure. L'objectif de ce projet est de trouver un moyen d'offrir une interface de programmation interactive permettant de facilement développer des applications 3D de façon déclarative.

Chapitre 2

Remerciements

Chapitre 3

YAW : code existant

YAW est une bibliothèque Java/Clojure de rendu et manipulation 3D faisant principalement interface avec OpenGL.

OpenGL est une bibliothèque de rendu très bas-niveau, manipulant des points et des triangles, sans aucune notion d'objets ou de transformations telles que des rotations ou translations. YAW offre ces fonctionnalités de manipulation de plus haut niveau, tout en gardant le contrôle de l'utilisateur sur la géométrie des objets manipulés.

3.1 Interface Java

3.1.1 La classe `World`

L'interface publique Java de YAW est principalement accédée par la classe `World`.

Cette classe représente un fenétre sur un espace 3D et sa population. Le listing 3.1 liste quelques méthodes proposées par cette classe.

Un processus notable est celui d'ajout d'un objet 3D dans la scène. L'objet (dans le code *item*) est inséré dans le monde avec un nom, une position et une échelle, et un *mesh*. Ce *mesh* est aussi créé via une méthode de `World` avec les données bas-niveau de géométrie requises par OpenGL.

Cette classe propose un nombre de méthodes d'altération de la scène directement, à l'exception des lumières, qui nécessitent de récupèrer la `SceneLight` et d'interagir avec.

```
public class World implements Runnable {
  // ... implementation de Runnable omise, il s'agit de la gestion de la
  //     fenetre.

  public World(int pInitX, int pInitY, int pInitWidth, int pInitHeight);

  public Item createItem(String id, float[] pPosition, float pScale, Mesh
  pMesh);

  public Mesh createMesh(float[] pVertices, float[] pTextCoords, float[]
  pNormals, int[] pIndices, int pWeight, float[] rgb, String
  pTextureName);

  public Item createBoundingBox(String id, float[] pPosition, float pScale,
  float[] pLength);

  public boolean isInCollision(Item item1, Item item2);

  public void setSkybox(float pWidth, float pLength, float pHeight, float pR,
  float pG, float pB);
  public void removeSkybox();

  public SceneLight getSceneLight();

  public void addCamera(int pIndex, Camera pCamera);

  //... reste omis
}
```

Listing 3.1 – Interface incomplète de la classe World

3.1.2 Les lumières

```
public class SceneLight {
  public static final int MAX_POINTLIGHT = 5;
  public static final int MAX_SPOTLIGHT = 5;

  public SceneLight();

  public void removeAmbient();

  public void removeSun();

  public void setPointLight(PointLight pl, int pos);

  public void setSpotLight(SpotLight sl, int pos);

  public DirectionalLight getSun();
  public void setSun(DirectionalLight sun);

  public AmbientLight getAmbientLight();
  public void setAmbient(AmbientLight ambient);

  //... reste omis
}
```

Listing 3.2 – Interface incomplète de la classe SceneLight

La gestion des lumières se fait à travers une instance de la classe `SceneLight`, présentée partiellement en listing 3.2.

Il est possible d'avoir jusqu'à `MAX_POINTLIGHT` sources de lumières ponctuelles, et jusqu'à `MAX_SPOTLIGHT` projecteurs.

Contrairement à la création des objets dans `World`, la création des lumières se fait directement par constructeur.

3.2 Interface Clojure

L'interface Clojure de YAW est une interface quasiment traduite. Un fichier `world.clj` comporte un ensemble de fonctions appelant directement les méthodes de la classe `World`.

```
(defn start-universe! [& {:keys [width height x y]}] (...))

(defn create-mesh! [world & {:keys [vertices text-coord normals faces weight
  rgb texture-name]}] (...))

(defn create-item! [world & {:keys [id position scale mesh]}] (...))

(defn create-block! [world & {:keys [id position scale color texture]}] (...))

(defn create-bouding-box! [world & {:keys [id position length scale]}] (...))
(defn add-bounding-box! [item bounding-box] (...))
(defn check-collision! [world item1 item2] (...))

(defn rotate! [item & {:keys [x y z]} (...))
(defn translate! [item & {:keys [x y z]} (...))
```

Listing 3.3 – Interface initiale simplifiée de `world.clj`

Les fonctions clojure présentée en listing 3.3 ont été simplifiées par souci de brièveté : les arguments suivant les esperluettes sont optionels et ont des valeurs par défaut. Ces fonctions ont un point d'exclamation à la fin de leur nom, ce qui dans la communauté Clojure a été accpeté comme convention pour signifier qu'une fonction a des effets de bords sur la mémoire et n'est pas « thread-safe. »

L'interface est incomplète, manque la gestion des lumières, des caméras, et d'autres fonctionnalités diverses offertes par la bibliothèque Java.

Chapitre 4

Recherche

4.1 Inspirations initiales

4.1.1 React

4.1.2 Unity

4.2 Reagent

4.3 Re-frame

Chapitre 5

Fonctionnalités

5.1 Programmation fonctionnelle réactive

5.1.1 Objectif

5.1.2 Plan de réalisation

5.1.3 Problèmes

5.2 Moteur de jeu

5.2.1 Objectif

5.2.2 Plan de réalisation

5.2.3 Problèmes

5.3 Live Coding

5.3.1 Objectif

5.3.2 Plan de réalisation

5.3.3 Problèmes

Chapitre 6

Conclusion