Tutoriel JMonkeyEngine

http://jmonkeyengine.org/

Documentation : https://jmonkeyengine.github.io/wiki/

1. JMonkeyEngine dans Eclipse

Même si JMonkeyEngine propose son propre environnement de développement (IDE), celui-ci est très accès sur le développement de jeux vidéos et il n'est pas *cross-platform*. Nous allons donc utiliser l'environnement de développement **Eclipse** afin de pouvoir effectuer plus facilement des développements externes à JMonkeyEngine (partie applicative, interface 2D Java Swing, etc.). Nous allons donc devoir importer les librairies JMonkeyEngine dans **Eclipse**.

- Créer un nouveau projet dans **Eclipse** (`File > New > Java Project`). On le nommera *TutorielJME* dans l'exemple.
- Télécharger les librairies de JMonkeyEngine à l'adresse suivante : https://github.com/jMonkeyEngine/jmonkeyengine/releases/download/v3.1.0-stable/jME3.1-stable.zip
- Afin que le projet **Eclipse** ne soit pas lié à des dépendances externes (*self-contained project*) et que vous puissiez ainsi partager plus facilement le projet pour le travail en groupe, nous inclurons les librairies directement dans le projet **Eclipse**.
  - Décompresser l'archive de JMonkeyEngine (*jME3.1-stable.zip*).
  - **Chez vous**, vous pourrez exécuter le fichier *jMonkeyEngine3.jar p*our tester les démonstrations de JMonkeyEngine.
  - Dans le répertoire du projet *TutorielJME* (où il y a déjà les répertoires *src* et *bin*), créer un répertoire pour les dépendances "*deps*".
  - Copier tous les *.jar* présents dans le répertoire *lib* de l'archive et coller les dans le répertoire *deps* de votre projet.
  - Faire un clique droit sur le nom de votre projet dans **Eclipse** et choisir `"Refresh"`: *deps* apparait dans **Eclipse**.
  - Faire un clique droit sur le nom de votre projet dans **Eclipse** et choisir `"Build Path > Configure Build Path"`. Aller dans l'onglet « `Librairies"` et cliquer ensuite sur `"Add JARs"`, aller dans le répertoire de votre projet, puis dans le répertoire *deps*. Sélectionner tous les *.jar* présents dans *deps* et cliquer sur `OK`.

- Afin de faciliter le développement, vous pouvez également intégrer la documentation `javadoc` de JMonkeyEngine dans **Eclipse**.
  - Faire un clique droit sur le nom de votre projet dans **Eclipse** et choisir `"Build Path > Configure Build Path"`. Aller dans l'onglet `"Librairies"` et localiser *jme3-core.jar*.
  - Cliquer sur la flèche devant *jme3-core.jar*, sélectionner `"Javadoc Location"`, et cliquer sur `"Edit..."`.
  - Cliquer sur `Browse` et choisir le répertoire *javadoc* de l'archive *.zip* que vous avez décompressé à l'étape précédente. Cliquer deux fois sur `OK`.

Si vous utilisez d'autres composants qui ne sont pas dans le package principal de JMonkeyEngine (*jme3-core.jar*), vous devrez répéter l'opération pour les autres *.jar* concernés.

- Vous pouvez commencer à coder une nouvelle application JMonkeyEngine. Pour faire une application simple, effectuer les opérations suivantes :
  - Créer un nouveau package (`File > New > New Package`). L'appeler par exemple "*tutoriel*".
  - Créer une nouvelle classe (`File > New > Class`) :
    - Sélectionner le package "*tutoriel*".
    - Donner un nom à la classe. Par exemple : "*CubesTest*".
    - Choisir comme `Superclass` : *com.jme3.app.SimpleApplication*
    - S'assurer que la checkbox `"public static void main()"` est cochée.
    - S'assurer que la checkbox `"Inheriting Abstract Methods"` est cochée.
    - Cliquer sur `"Finish"`.

2. Créer une application simple

Pour commencer, on va créer une application toute simple qui affiche 3 cubes de couleur (rouge, vert, bleu) à des positions différentes.

- Voici le code permettant de créer un cube de couleur bleu à la position $(0, 0, 0)$:

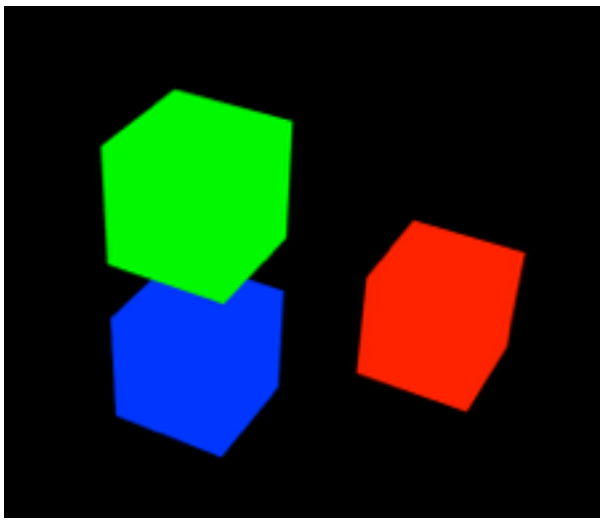
```
Box b = new Box(1, 1, 1); // create cube shape
Geometry geom = new Geometry("Box", b);  // create cube geometry from the shape
Material mat = new Material(assetManager,
  "Common/MatDefs/Misc/Unshaded.j3md");  // create a simple material
mat.setColor("Color", ColorRGBA.Blue);   // set color of material to blue
geom.setMaterial(mat);                   // set the cube's material
rootNode.attachChild(geom);              // make the cube appear in the scene
```

- Ajouter ce code dans la fonction `simpleInitApp` afin de créer le cube à l'initialisation de l'application.
- Ajouter le code suivant à la fonction `"public static void main()"` pour lancer l'application :

```
CubesTest app = new CubesTest();
app.start();
```

- Exécuter l'application, une fenêtre devrait apparaître avec un cube bleu (appuyer sur `ESC` pour quitter).
- Ajouter un cube vert et un cube rouge de telle façon que :
  - le cube vert soit au-dessus du cube bleu,
  - le cube rouge soit à droite et en retrait par rapport au cube bleu.

  Pour déplacer les cubes, vous pouvez appliquer des transformations à la géométrie en utilisant par exemple : `geom.setLocalTranslation(0.0f, 0.0f, 0.0f);`



3. Animer une géometrie

Pour permettre d'agir sur les objets de la scène 3D d'une application dérivant de `SimpleApplication`, JMonkeyEngine permet de redéfinir une méthode `SimpleUpdate` qui sera appelée à chaque *frame* (c'est-à-dire à chaque pas de temps de l'exécution). Cette fonction reçoit en paramètre le temps écoulé depuis la dernière *frame*.

- Ajouter l'appel à la méthode `SimpleUpdate` dans votre classe "*CubesTest*" :

```
@Override
public void simpleUpdate(float tpf) {
}
```

- Faire tourner l'un des cubes en appelant la méthode `rotate` sur sa géométrie (il faudra modifier le code précédent afin de pouvoir accéder à la variable correspondant à la géométrie du cube hors de la méthode `simpleInitApp`) :

```
public Spatial rotate(float xAngle,
        float yAngle,
        float zAngle)
```

4. Contrôler la caméra

Pour visualiser plus facilement un objet particulier de la scène, JMonkeyEngine offre une caméra qui à la capacité de suivre un objet particulier de la scène. Voir sur la page suivante le code qui permet de désactiver la caméra par défaut et de créer une caméra qui suit un objet particulier de la scène (le cube bleu dans cet exemple).

```
/*-------------- Camera settings ----------------*/
flyCam.setEnabled(false);// Disable the default flyby cam

// Enable a chase cam for a target (typically the player).
ChaseCamera chaseCam = new ChaseCamera(cam, geom, inputManager);

chaseCam.setDragToRotate(true);   // activate the windowed input behaviour

// Parameterize the camera motion
chaseCam.setInvertVerticalAxis(true);
chaseCam.setRotationSpeed(10.0f);
chaseCam.setMinVerticalRotation((float) -(Math.PI/2 - 0.0001f));
chaseCam.setMaxVerticalRotation((float) Math.PI/2);
chaseCam.setMinDistance(7.5f);
chaseCam.setMaxDistance(30.0f);
```

Ce code doit être ajouté dans la fonction `simpleInitApp()`. Vous pourrez modifier les différents paramètres de déplacement cette caméra pour lui donner le comportement que vous souhaitez.

5. Configurer l'application

Pour éviter d'avoir la fenêtre de paramétrage de l'application qui apparait à chaque fois que l'on lance l'application, vous pouvez définir une fois pour toute vos propres paramètres et les appliquer à l'application. Vous pouvez aussi désactiver la fenêtre de paramétrage qui apparait au lancement en ajoutant ce code à la fonction `"public static void main()"` :

```
AppSettings settings = new AppSettings(true);
settings.setResolution(1280,800);               // Resolution of the windows
settings.setSamples(8);                         // Anti-aliasing parameter

CubesTest app = new CubesTest();
app.setSettings(settings);                      // Add the settings to the application
app.setShowSettings(false);                     // Disable the configuration windows
app.start();
```

Pour ne pas surcharger votre ordinateur, vous pouvez aussi limiter la fréquence d'exécution du programme (dans tous les cas, il est inutile d'avoir un programme qui tourne plus vite que la fréquence de rafraichissement de votre écran qui est généralement de 60Hz). Vous pouvez donc ajouter le paramètre de réglage suivant :

```
settings.setFrameRate(60);
```

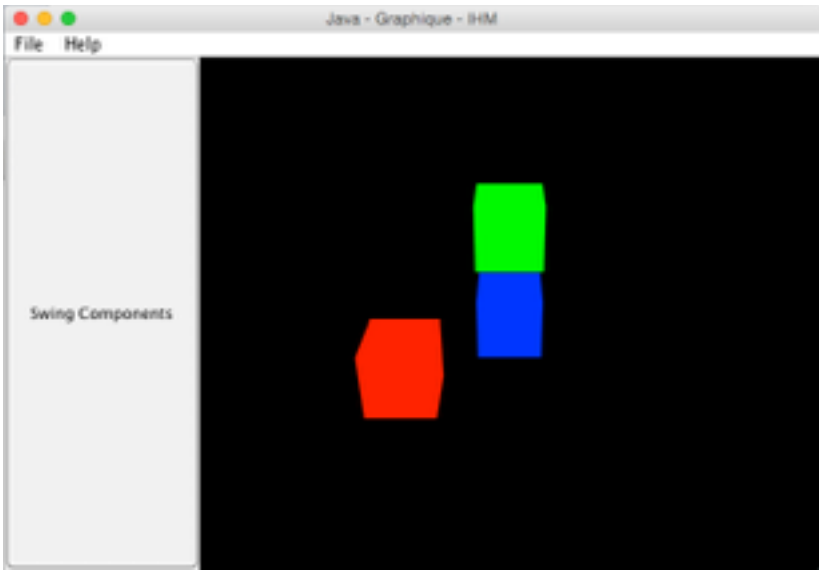
Pour éviter les déchirements verticaux de l'image, vous pouvez activer la synchronisation verticale même si c'est plus coûteux en terme de performance :

```
settings.setVSync(true);
```

Enfin, vous pouvez également cacher les statiques et l'affichage des FPS (*frame per second*) :

```
app.setDisplayStatView(false);
app.setDisplayFps(false);
```

6. Intégrer l'application JME comme un *canvas* dans JAVA Swing



Nous souhaitons maintenant intégrer l'application, que nous avons réalisé, comme un canvas dans une fenêtre JAVA Swing. Pour cela, au lieu d'appeler la fonction `app.start()` pour lancer l'application, nous appellerons la fonction `app.createCanvas()` qui créera un canvas que nous pourrons ensuite placer dans la fenêtre comme un composant JAVA Swing.

- Télécharger le fichier *WindowedTest.java*
  (sur https://www.lri.fr/~cfleury/teaching/et3-info/ProjetJavaIHM-2016/).
  Ce fichier contient le code pour créer une JFrame Swing avec quelques éléments de test et propose une structure de base pour intégrer une application JMonkeyEngine dans cette JFrame.
- Ajouter le fichier *WindowedTest.java* à votre projet **Eclipse**.
- Regarder ce fichier pour en comprendre la structure.
- Modifier le fichier en s'aidant des commentaires au niveau des `TODO` (au nombre de 6 dans le fichier) afin de pouvoir intégrer votre application *CubesTest* dans la JFrame.
- Parmi les options à ajouter à l'application (TODO n°4), il faut ajouter l'option suivante afin que l'application ne se mette pas en pause lorsque le *focus* n'est plus sur la fenêtre :

```
canvasApplication.setPauseOnLostFocus(false);
```

7. Charger une géométrie

Nous souhaitons maintenant créer une nouvelle application JMonkeyEngine avec une géométrie plus complexe que des simples cubes. Pour cela, nous voulons charger un modèle 3D d'une sphère avec une texture plaquée dessus.

- Télécharger l'archive *earth.zip* sur https://www.lri.fr/~cfleury/teaching/et3-info/ProjetJavaIHM-2016/. Cette archive contient un fichier *.mesh.xml* qui est la géométrie 3D, un fichier *.mtl* qui contient la définition des matériaux et plusieurs textures qui seront plaquées sur la géométrie. Il faut laisser cette archive compressée.

- Créer une nouvelle application JMonkeyEngine (voir partie n°1, dernière partie). On pourra appeler cette nouvelle classe "*EarthTest*".
- Déplacer l'archive *earth.zip* dans le répertoire du projet *TutorielJME* (où il y a les répertoires *src* et *bin*).
- Ajouter le code suivant à la fonction `simpleInitApp` afin de charger la géométrie contenue dans l'archive :

```
// Load a model
assetManager.registerLocator("earth.zip", ZipLocator.class);
Spatial earth_geom = assetManager.loadModel("earth/Sphere.mesh.xml");
Node earth_node = new Node("earth");
earth_node.attachChild(earth_geom);
earth_node.setLocalScale(5.0f);
rootNode.attachChild(earth_node);
```

- Vous devez ajouter une lumière dans la scène afin de voir la géométrie, en utilisant le code suivant :

```
// You must add a light to make the model visible
DirectionalLight directionalLight = new DirectionalLight(new Vector3f(-2, -10, 1));
directionalLight.setColor(ColorRGBA.White.mult(1.3f));
rootNode.addLight(directionalLight);
```

- Vous pouvez aussi changer la couleur de l'arrière plan si vous le souhaitez en ajouter la ligne de code suivante dans la fonction `simpleInitApp` :

```
// change the color of the background
viewPort.setBackgroundColor(new ColorRGBA(0.1f, 0.1f, 0.1f, 1.0f));
```

- Bien entendu, vous pouvez également modifier la façon de contrôler la caméra comme dans l'application *CubesTest* afin de pouvoir plus facilement tourner autour de l'objet.

NB1 : pour la suite, il peut être intéressant de placer les géométries dans des **`Node`** afin de pouvoir y accéder plus facilement (par leur nom dans le graphe de scène) ou pour pouvoir déplacer plusieurs objets en même temps (plusieurs objets dans le même `Node`). Nous vous invitons à regarder une explication du graphe de scène de JMonkeyEngine : https://jmonkeyengine.github.io/wiki/jme3/the_scene_graph.html.

8. Tracer une ligne dans le monde 3D

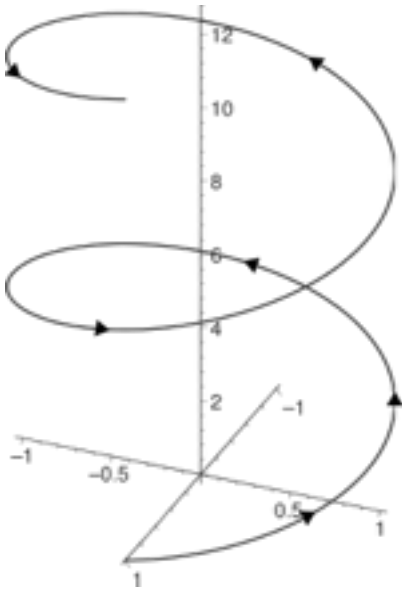
Pour tracer une ligne dans le monde 3D, une solution est de dessiner plusieurs petites lignes (`Line`) consécutives. Le constructeur de `Line` prend deux points en paramètres : le premier point définit le début de la ligne, le deuxième point définit la fin de la ligne. En mettant bout à bout des petites morceaux de lignes, on peut créer une ligne plus grande qui peut ainsi être courbe.

- Ajouter le code suivant à la fonction `simpleInitApp` pour créer une ligne simple à partir de deux points 3D. Vous devrez lui ajouter aussi un `material` comme nous l'avons fait pour les cubes dans la première application.

```
Node LinesNode = new Node("LinesNode");
Vector3f oldVect = new Vector3f(1, 0, 0);
Vector3f newVect = new Vector3f(-1, 1, 0);

// Draw one line
Line line = new Line(oldVect, newVect);
Geometry lineGeo = new Geometry("lineGeo", line);
Material mat = new Material(assetManager, "Common/MatDefs/Misc/Unshaded.j3md");
mat.getAdditionalRenderState().setLineWidth(4.0f); // set the width of the line
mat.setColor("Color", ColorRGBA.Green);
LinesNode.setMaterial(mat);
LinesNode.attachChild(lineGeo);

rootNode.attachChild(LinesNode);
```

- Nous voulons ensuite dessiner une hélice. Il faut donc créer un ensemble de petites lignes dont les points de début et de fin sont placés de façon à dessiner une hélice (par la suite, vous pourrez si vous le voulez modifier les positions de ces points afin de dessiner quelque chose d'autre). Voici une boucle `for` qui permet de créer une suite de points sur une hélice :

```
// Draw Helix
Vector3f oldVect = new Vector3f(1, 0, 0);
for (int i=0; i<100; i++) {
    float t = i / 5.0f;
    Vector3f newVect = new Vector3f(FastMath.cos(t), t/5.0f, FastMath.sin(t));

    // Draw one line
    // ...

    oldVect = newVect;
}
```



9. Convertir une latitude - longitude en coordonnées 3D sur une sphère

Pour placer des villes sur le globe, nous allons avoir besoin de convertir leurs coordonnées GPS (latitude / longitude) en coordonnées 3D sur la sphère. Nous allons vous aider à effectuer les calculs pour effectuer cette conversion. Nous allons par exemple essayer de placer les villes suivantes sur le globe :

- Brest : 48.447911 / -4.418539
- Marseille : 43.435555 / 5.213611
- New York : 40.639751 / -73.778925
- Cape Town : -33.964806 / 18.601667
- Istanbul : 40.976922 / 28.814606
- Reykjavik : 64.13, / -21.940556
- Singapore : 1.350189 / 103.994433
- Seoul : 37.469075 / 126.450517

- Premièrement, il se peut que la texture que nous affichons sur le globe ne soit pas exactement alignée avec les coordonnés GPS. Nous allons donc introduire deux constantes pour corriger cette alignement si nécessaire :

```
private static final float TEXTURE_LAT_OFFSET = -0.2f;
private static final float TEXTURE_LON_OFFSET = 2.8f;
```

- Voici ensuite le code pour effectuer la conversion des coordonnées GPS (latitude / longitude) en coordonnées 3D pour une sphère de rayon 1 et centrée en (0, 0, 0) :

```
private static Vector3f geoCoordTo3dCoord(float lat, float lon) {
    float lat_cor = lat + TEXTURE_LAT_OFFSET;
    float lon_cor = lon + TEXTURE_LON_OFFSET;
    return new Vector3f(- FastMath.sin(lon_cor * FastMath.DEG_TO_RAD)
                        * FastMath.cos(lat_cor * FastMath.DEG_TO_RAD),
                          FastMath.sin(lat_cor * FastMath.DEG_TO_RAD),
                        - FastMath.cos(lon_cor * FastMath.DEG_TO_RAD)
                        * FastMath.cos(lat_cor * FastMath.DEG_TO_RAD));
}
```

- Créer une fonction `displayTown(float latitude, float longitude)` qui affiche un point sur la carte à l'endroit de la ville (en fonction des coordonnées GPS qui seront passés en paramètre de la fonction). Pour afficher la ville, vous pourrez dessiner une sphère en utilisant la géométrie suivante :

```
Sphere sphere = new Sphere(16, 8, 0.005f);
Geometry town = new Geometry("Town", sphere);
```

- Tester la fonction `displayTown` en affichant en même temps les différentes villes dont les latitudes et longitudes vous sont données ci-dessus.

NB : la formule de conversion étant conçue pour un globe de rayon 1 et centré en (0, 0, 0), il sera intéressant de placer la géométrie du globe et les géométries des villes dans le même `Node`. Cela pourra permettre de déplacer le globe et les villes en même temps en changeant la position du `Node`, ainsi que de changer la taille du globe en changeant l'échelle du `Node` (ce qui permettra de conserver la relation entre le globe et les représentations des villes malgré le changement de taille).

10. Contacts

Groupe 1 : Cédric FLEURY - cedric.fleury@lri.fr
Groupe 2 : Olivier GLADIN - olivier.gladin@inria.fr