Tutoriel JavaFx 3D

Documentation JavaFx : https://openjfx.io/
Documentation API JavaFx : https://openjfx.io/javadoc/16/
Pour aller plus loin : https://fxdocs.github.io/docs/html5/

1. Installation de JavaFX

Dans ce tutoriel nous allons apprendre à utiliser JavaFX, développé au sein du projet OpenJFX, pour réaliser des interfaces graphiques avec Java.

Vous êtes libre d'utiliser l'environnement de développement qui vous convient le mieux.
Nous vous conseillons cependant d'utiliser :

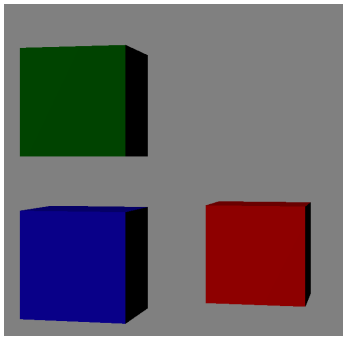
- soit **Eclipse** : https://www.eclipse.org/downloads/
- soit **IntelliJ** : https://www.jetbrains.com/fr-fr/idea/download/

Si vous choisissez une autre plateforme, l'installation de JavaFX ne sera pas abordée dans ce tutoriel.

Pour installer JavaFX sous Eclipse ou IntelliJ, vous trouverez un tutoriel détaillé écrit par Eugénie Brasier sur https://github.com/polytech-ihm-et3/HowToUseJavaFxWithIDE

2. Créer une application simple utilisant JavaFx

Dans cette section, nous allons créer un projet Java qui va utiliser JavaFX pour afficher 3 cubes de couleur (rouge, vert, bleu) à des positions différentes.



- Créer un nouveau projet Java simple (sans fichier FXML) comme vous l'avez fait lors du premier TP d'IHM. On le nommera *TutorielJavaFx3D* dans l'exemple.
- Vous pouvez commencer à coder une nouvelle application JavaFx. Pour faire une application simple, effectuer les opérations suivantes :
  - Créer un nouveau package, par exemple "*tutoriel*".
  - Créer une nouvelle classe dans le package "*tutoriel*" :
    - Donner un nom à la classe. Par exemple : "*CubeTest*".
    - Faire hériter cette classe de : *javafx.application.Application*
    - Ajouter la méthode "`public void start(Stage primaryStage)`" héritée de la classe `Application`.
    - Implémenter la méthode "`public static void main()`" pour qu'elle appelle la fonction `launch(…)` de la classe `Application`.
- Télécharger le fichier *CubeTest.java* disponible dans le répertoire de ressources du cours et le remplacer dans votre projet en utilisant l'explorateur de fichier.
- Sur la page suivante vous pouvez voir le code permettant de créer un cube de couleur bleu à la position (0, 0, 0) . Compléter le code de la classe *CubeTest.java* afin de créer un cube à l'initialisation de l'application.

```
//Create a Pane et graph scene root for the 3D content
Group root3D = new Group();
Pane pane3D = new Pane(root3D);

//Create cube shape
Box cube = new Box(1, 1, 1);

//Create Material
final PhongMaterial blueMaterial = new PhongMaterial();
blueMaterial.setDiffuseColor(Color.BLUE);
blueMaterial.setSpecularColor(Color.BLUE);
//Set it to the cube
cube.setMaterial(blueMaterial);

//Add the cube to this node
root3D.getChildren().add(cube);

//Add a camera group
PerspectiveCamera camera = new PerspectiveCamera(true);
Group cameraGroup = new Group(camera);
root3D.getChildren().add(cameraGroup);

//Rotate then move the camera
Rotate ry = new Rotate();
ry.setAxis(Rotate.Y_AXIS);
ry.setAngle(-15);

Translate tz = new Translate();
tz.setZ(-10);
tz.setY(-1);

cameraGroup.getTransforms().addAll(ry,tz);

// Add point light
PointLight light = new PointLight(Color.WHITE);
light.setTranslateX(-180);
light.setTranslateY(-90);
light.setTranslateZ(-120);
light.getScope().addAll(root3D);
root3D.getChildren().add(light);

// Create scene
Scene scene = new Scene(pane3D, 600, 600, true);
scene.setCamera(camera);
scene.setFill(Color.GREY);

//Add the scene to the stage and show it
primaryStage.setTitle("Cubes Test");
primaryStage.setScene(scene);
primaryStage.show();
```

- Exécuter l'application, une fenêtre devrait apparaître avec un cube bleu.
- 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 les *convenience methods*, par exemple :

```
redCube.setTranslateX(1.5);
```

3. Animer une géométrie

Pour animer les objets 3D, nous pourrions utiliser les animations vues en cours. Cependant, pour effectuer des opérations complexes sur les objets de la scène 3D de l'application, nous préférons utiliser ici un `AnimationTimer`.

Cet `AnimationTimer` doit implémenter une méthode `handle(…)` qui sera appelée à chaque *frame* et nous permettra donc d'effectuer des actions plus ou moins complexes sur les objets de la scène 3D à chaque pas de temps de notre simulation. Le nombre reçu en paramètre de la fonction `handle(…)` est le temps courant en nanoseconde.

Dans ce tutoriel, nous cherchons à donner une rotation à vitesse constante d'un cube de la scène 3D.

```
// Add an animation timer
final long startNanoTime = System.nanoTime();
new AnimationTimer() {
    public void handle(long currentNanoTime) {
        double t = (currentNanoTime - startNanoTime) / 1000000000.0;
        //Add your code here
    }
}.start();
```

- Ajouter à la méthode `start` de votre classe "*CubeTest*", la classe interne anonyme suivante, sur laquelle vous appellerez la méthode `start()` pour démarrer l'animation :

- Faire tourner l'un des cubes en appelant la méthode `setRotationAxis` pour définir l'axe de rotation puis `setRotate` pour appliquer une rotation autour de cet axe

```
greenCube.setRotationAxis(new Point3D(0,1,0));
greenCube.setRotate(rotationSpeed * t);
```

4. Contrôler la caméra

Pour visualiser plus facilement un objet particulier de la scène, nous vous fournissons une classe `CameraManager` permettant de gérer le déplacement de la caméra au clavier et à la souris. Cette classe permet à la caméra de tourner autour de l'objet sélectionné grâce aux entrées clavier/souris.

- Télécharger le fichier *CameraManager.java* depuis le répertoire de ressources du cours.
- Ajouter ce fichier dans le même répertoire que votre fichier "*CubeTest.java*" (probablement dans le répertoire *src/tutoriel* de votre projet).
- Le code modifiant la position initiale de la caméra doit être mis en commentaire :

```
//      Group cameraGroup = new Group(camera);
//      root.getChildren().add(cameraGroup);

//      //Rotate then move the camera
//      Rotate ry = new Rotate();
//      ry.setAxis(Rotate.Y_AXIS);
//      ry.setAngle(-15);
//
//      Translate tz = new Translate();
//        tz.setZ(-10);
//        tz.setY(-1);
//
//      cameraGroup.getTransforms().addAll(ry,tz);
```

- Instancier la classe `CameraManager` dans la méthode `start` de votre classe "*CubeTest*". Vous devez passer en paramètre du constructeur la `Camera` à contrôler, la `Scene` contenant la caméra (et qui reçoit aussi les entrées clavier/souris), ainsi que le `Group` contenant le ou les objets autour desquels nous voulons que la caméra tourne :

```
// Build camera manager
new CameraManager(camera, scene, root);
```

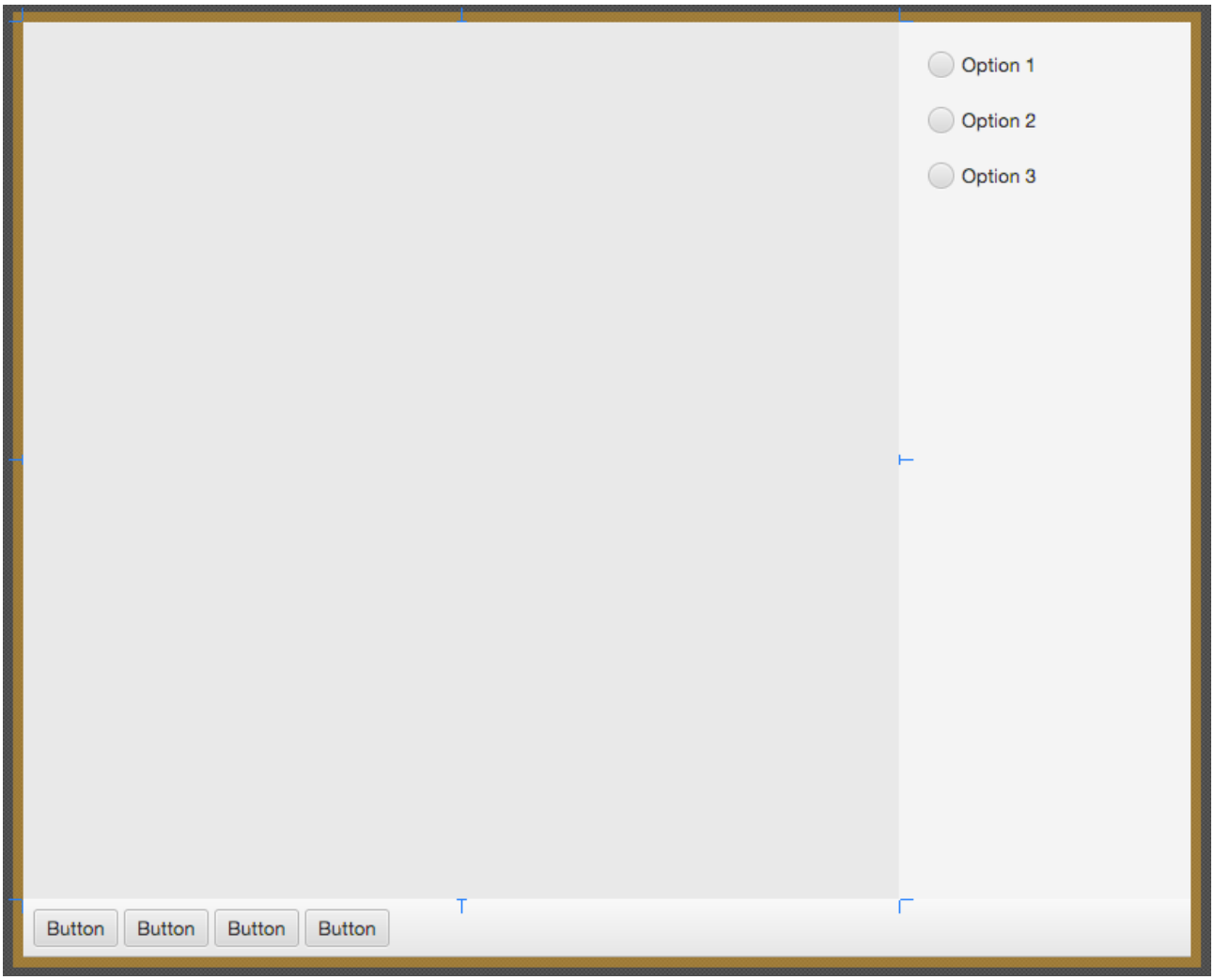
- Regarder le code de la classe `CameraManager` et essayer de comprendre quelles actions clavier/souris permettent de déplacer la caméra. Tester le déplacement de la caméra en lançant l'application.
- Regarder les constantes de la classe `CameraManager` et essayer de comprendre à quoi elles servent. Modifier les différents paramètres de déplacement de cette caméra pour lui donner le comportement que vous souhaitez.

```
private static final double CAMERA_INITIAL_DISTANCE = -5;
private static final double CAMERA_INITIAL_X_ANGLE = 0.0;
private static final double CAMERA_INITIAL_Y_ANGLE = 0.0;
private static final double CAMERA_NEAR_CLIP = 0.1;
private static final double CAMERA_FAR_CLIP = 10000.0;
private static final double CONTROL_MULTIPLIER = 0.1;
private static final double SHIFT_MULTIPLIER = 10.0;
private static final double MOUSE_SPEED = 0.05;
private static final double ROTATION_SPEED = 2.0;
private static final double TRACK_SPEED = 0.6;
```

5. Intégrer la vue 3D dans une interface 2D de JavaFx

Dans cette partie, nous souhaitons maintenant intégrer la scène que nous avons réalisée dans une application utilisant des widget 2D JavaFx créée avec le *SceneBuilder*. Pour cela, il faut intégrer la scène 3D dans une `SubScene` au lieu de la `Scene` principale de la fenêtre. Il faudra ensuite intégrer cette `SubScene` dans un "*layout pane*" (un `Pane` par exemple) que vous aurez créé et auquel vous aurez donné un identifiant dans le *SceneBuilder*.

- Créer un fichier *FXML* dans le repertoire *src/tutoriel* (par exemple *gui2D.fxml*).



- Ouvrir ce fichier avec le *SceneBuilder* et créer une interface ressemblant à l'interface suivante :

- La zone du haut à gauche doit être un `Pane` simple de 600 par 600 pixels.
- Dans l'interface de *SceneBuilder*, sélectionner ce `Pane`. Dans l'*inspector* de droite, choisir le sous-menu *Code* et donner une valeur au *fx:id* (par exemple, `pane3D`).

```java
public class CubeGUI extends Application {

    @Override
    public void start(Stage primaryStage) {
        try {
            Parent content = FXMLLoader.load(getClass().getResource("gui2D.fxml"));
            primaryStage.setTitle("Cube 3D in GUI 2D");
            primaryStage.setScene(new Scene(content));
            primaryStage.show();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }


    public static void main(String[] args) {
        launch(args);
    }
}
```

- Créer une nouvelle application JavaFx en créant une nouvelle classe (par exemple, *CubeGUI*). Cette classe chargera le fichier *gui2D.fxml* en utilisant le code vu en cours et en TP. Voici un exemple :

```java
public class Controller implements Initializable {

    @FXML
    private Pane pane3D;

    @Override
    public void initialize(URL location, ResourceBundle resources) {

        // Put the code of the 3D scene here !

    }
}
```

- Créer une nouvelle classe *Controller* qui permettra de faire le lien avec cette interface 2D. Créer dans cette classe un attribut `pane3D` de type `Pane` annoter avec `@FXML` afin de faire le lien avec le `Pane` que vous avez créé dans l'interface 2D avec le SceneBuilder.

- Mettre le code correspondant à la scène 3D dans le fonction `initialize(…)` du *Controller*.

```java
// Create the subscene
SubScene subscene = new SubScene(root3D, 600, 600, true, SceneAntialiasing.BALANCED);
subscene.setCamera(camera);
subscene.setFill(Color.GREY);
pane3D.getChildren().addAll(subscene);
```

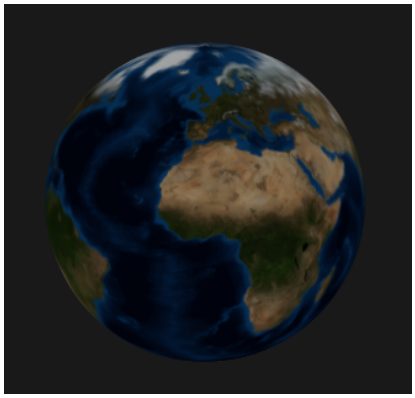
- Remplacer dans ce code la `Scene` par une `SubScene` dans laquelle vous ajouterez le graphe de la scène 3D (`root3D` dans notre précédent code) et à laquelle vous ajouterez aussi la camera. Intégrer cette `SubScene` dans le `pane3D` prenant du *FXML*.

```
<BorderPane prefHeight="640.0" prefWidth="800.0" xmlns="http://javafx.com/javafx/8.0.111" xmlns:fx="http://
javafx.com/fxml/1" fx:controller="tutoriel.Controller">
   <bottom>
```

- Faire le lien entre le *FXML* et le *Controller* en ajoutant `fx:controller="tutoriel.Controller"` au noeud de niveau de le plus haut dans le *FXML*.

6. Charger une géométrie

Nous souhaitons maintenant créer une nouvelle application JavaFx 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.



ll sera d'abord nécessaire d'inclure une librairie externe pour charger les géométrie au format WaveFront OBJ dans la scène 3D :

- Téléchargez et décompresser le fichier *jimObjModelImporterJFX.zip* depuis le répertoire de ressource du cours.
- Dans le répertoire du projet *TutorielJavaFx3D* (où il y a déjà les répertoires *src* et *bin* (**Eclipse**) ou *src* et *out (***IntelliJ***)*), créer un répertoire pour les dépendances "*deps*".
- Copier le fichier *jimObjModelImporterJFX.jar* et le coller dans le répertoire "*deps*" de votre projet.
- Ajouter ce fichier .jar aux librairies de votre projet :
  - Avec **Eclipse**:
    - Faire un clic droit sur le nom de votre projet et choisir "`Refresh`": *deps* apparait.
    - Faire un clic droit sur le nom de votre projet 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 le *.jar* présent dans *deps* et cliquer sur `OK`.
  - Avec **IntelliJ**:
    - Cliquer dans la fenêtre **IntelliJ**, le repertoire *deps* devrait apparaître.
    - Dérouler le repertoire *deps* pour voir son contenu et faire un clic droit sur le fichier *jimObjModelImporterJFX.jar* le nom de votre projet et choisir "`Add as Library…`". Laisser les paramètres proposés (`Project Library` et votre projet) et cliquer sur `OK`.

- Télécharger l'archive *Earth.zip* depuis le répertoire de ressource du cours. Cette archive contient un fichier *.obj* qui est la géométrie 3D, un fichier *.mtl* qui contient la définition des matériaux et une texture .png qui sera plaquée sur la géométrie.
- Décompresser l'archive *Earth.zip* dans le répertoire *src/tutoriel*.
- Créer une nouvelle application JavaFx (voir partie 1). On pourra appeler cette nouvelle classe "*EarthTest*".

- Télécharger le fichier *EarthTest.java* depuis le lien *#tutoriel / ressources* sur le discord du cours et le remplacer dans votre projet.

- Ajouter le code suivant à la fonction `start` afin de charger la géométrie contenue dans l'archive :

```
// Load geometry
ObjModelImporter objImporter = new ObjModelImporter();
try {
    URL modelUrl = this.getClass().getResource("Earth/earth.obj");
    objImporter.read(modelUrl);
} catch (ImportException e) {
    // handle exception
    System.out.println(e.getMessage());
}
MeshView[] meshViews = objImporter.getImport();
Group earth = new Group(meshViews);
```

- Ajouter ensuite le `Group` *earth* au graphe de la scène 3D.

```
scene.setFill(Color.gray(0.2));
```

- Vous pouvez aussi ajuster la couleur de l'arrière plan si vous le souhaitez avec la ligne de code suivante :

NB1 : pour la suite, il peut être intéressant de placer les géométries dans des **`Group`** afin de pouvoir les manipuler plus facilement ou pour pouvoir déplacer plusieurs objets en même temps (plusieurs objets dans le même `Group`).

7. 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 cet alignement :

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

- La fonction suivante permet d'effectuer la conversion des coordonnées GPS (latitude / longitude) en coordonnées 3D pour une sphère de rayon *radius* (dans cet exemple le rayon est de 1) et centrée en (0, 0, 0). Elle vous est fournie dans le fichier *EarthTest.java* :

```
public static Point3D geoCoordTo3dCoord(float lat, float lon, float radius) {
    float lat_cor = lat + TEXTURE_LAT_OFFSET;
    float lon_cor = lon + TEXTURE_LON_OFFSET;
    return new Point3D(
            -java.lang.Math.sin(java.lang.Math.toRadians(lon_cor))
                    * java.lang.Math.cos(java.lang.Math.toRadians(lat_cor))*radius,
            -java.lang.Math.sin(java.lang.Math.toRadians(lat_cor))*radius,
            java.lang.Math.cos(java.lang.Math.toRadians(lon_cor))
                    * java.lang.Math.cos(java.lang.Math.toRadians(lat_cor))*radius);
}
```

- Créer une fonction `displayTown(Group parent, String name, 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(0.01);
```

- Cette fonction devra placer la sphère dans un `Group` qui sera translaté à la bonne position et qui portera comme identifiant le nom de la ville (grâce à la méthode `setId(String id)`). Ce `Group` sera lui-même placé dans le `Group parent` passé en paramètre de la fonction.

- 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 centré en (0, 0, 0), il sera intéressant de placer la géométrie du globe et les géométries des villes dans le même `Group`. Cela pourra permettre de déplacer le globe et les villes en même temps en changeant la position du `Group`, 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).

8. Afficher une zone de *x* par *x* degrés

Le projet vous demandera d'afficher des couleurs sur des zones définies par une position (latitude et longitude) et une largeur/hauteur définie elle aussi en degré. Ces zones auront donc des formes différentes suivant qu'elles sont proches de l'équateur (≈rectangles) ou des pôles (≈triangles).

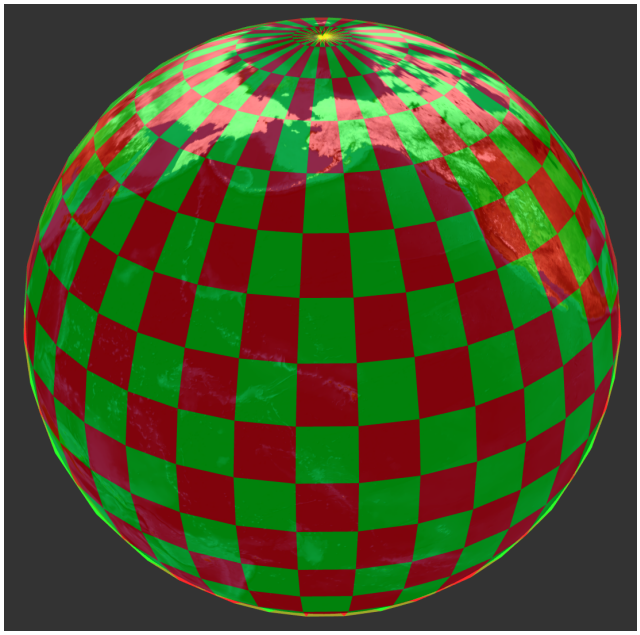
Pour cela vous pouvez utiliser la fonction `AddQuadrilateral` fournie ci-dessous. Cette fonction ajoute un quadrilatère constitué de deux triangles au groupe passé en paramètre.

```
private void AddQuadrilateral(Group parent, Point3D topRight, Point3D bottomRight,  Point3D bottomLeft,
        Point3D topLeft, PhongMaterial material) {

    final TriangleMesh triangleMesh = new TriangleMesh();

    final float[] points = {
            (float)topRight.getX(), (float)topRight.getY(), (float)topRight.getZ(),
            (float)topLeft.getX(), (float)topLeft.getY(), (float)topLeft.getZ(),
            (float)bottomLeft.getX(), (float)bottomLeft.getY(), (float)bottomLeft.getZ(),
            (float)bottomRight.getX(), (float)bottomRight.getY(), (float)bottomRight.getZ()
        };
    final float[] texCoords = {
            1, 1,
            1, 0,
            0, 1,
            0, 0
        };

    final int[] faces = {
            0, 1,  1, 0, 2, 2,
            0, 1, 2, 2, 3, 3
        };

         points:
           1      0
            -------    texture:
            |    /|  1,1(0)  1,0 (1)
            |   / |    -------
            |  /  |    |     |
            | /   |    |     |
            |/    |    -------
            -------  0,1(2)  0,0 (3)
           2      3


    triangleMesh.getPoints().setAll(points);
    triangleMesh.getTexCoords().setAll(texCoords);
    triangleMesh.getFaces().setAll(faces);

    final MeshView meshView = new MeshView(triangleMesh);
    meshView.setMaterial(material);
    parent.getChildren().addAll(meshView);
}
```

Vous pouvez, pour vous exercer, afficher des zone de 2° par 2° à la position des villes du chapitre précédent. Vous devrez sans doute les afficher un peu au dessus du globe terrestre, pour que les géométries n'entrent

pas en collision et que la zone soit bien affichée. Afin de continuer à voir la texture du globe, pensez à utiliser un matériau translucide.

Vous pourrez ensuite recouvrir le globe de zones de couleurs différentes comme dans l'image ci-dessous. Les latitudes vont de -90 à 90 et les longitudes de -180 à 180.



9. Utilisation de la souris pour indiquer des positions

Pour sélectionner des positions sur la surface du globe, nous allons utiliser la souris comme pointeur 3D. Pour cela, nous allons utiliser les fonctions de raycast de JavaFx : ces fonctions vont permettre de “tirer un rayon” depuis la position de la souris vers la sphère dans la vue 3D et ainsi obtenir la position de l’intersection de ce rayon avec la surface de la sphère.

Dans un premier temps, nous allons récupérer l’évènement qui est généré par un clic souris afin de déclencher le raycast. Pour éviter les conflits avec le clic souris permettant de contrôler la caméra, nous allons également demander à l’utilisateur d’appuyer en même temps sur la touche Alt.

- Ajouter le code suivant dans la fonction initialize :

```
subscene.addEventHandler(MouseEvent.ANY,  event -> {
    if (event.getEventType() == MouseEvent.MOUSE_PRESSED && event.isAltDown()) {

        PickResult pickResult = event.getPickResult();
        Point3D spaceCoord = pickResult.getIntersectedPoint();

        // ...

    }
});
```

La variable `spaceCoord` contient les coordonnées du point de la surface de la sphère désigné par la souris.

- Compléter ce code pour afficher une sphère sur le globe à la position de la souris lors d’un alt + clic

10. Conversion de coordonnées géographiques en geohash

Le geohash est un système de codage qui permet de diviser la surface terrestre en hash.
Son fonctionnement est expliqué à l'adresse suivante : https://borntocode.fr/java-comprendre-et-utiliser-lalgorithme-geohash/

Vous trouverez à la même adresse un lien vers un répertoire ou un algorithme de conversion latitude - longitude vers geohash.

- Ajoutez dans votre projet les fichiers java contenant l'algorithme de conversion

- Récupérez le geohash d'une zone du globe depuis la position du curseur :
  1. récupérez une position sur la surface du globe avec la souris (voir section précédente)
  2. transformez cette position en latitude / longitude en utilisant le code suivant :

Pour la valeur de `TEXTURE_OFFSET`, vous pouvez utiliser `1,01`.

```
public static Point2D SpaceCoordToGeoCoord(Point3D p) {

    float lat = (float) (Math.asin(-p.getY() / TEXTURE_OFFSET)
                    * (180 / Math.PI) - TEXTURE_LAT_OFFSET);
    float lon;

    if (p.getZ() < 0) {
            lon = 180 - (float) (Math.asin(-p.getX() / (TEXTURE_OFFSET
                    * Math.cos((Math.PI / 180)
             * (lat + TEXTURE_LAT_OFFSET)))) * 180 / Math.PI + TEXTURE_LON_OFFSET);
    } else {
            lon = (float) (Math.asin(-p.getX() / (TEXTURE_OFFSET * Math.cos((Math.PI / 180)
                        * (lat + TEXTURE_LAT_OFFSET)))) * 180 / Math.PI - TEXTURE_LON_OFFSET);
    }

    return new Point2D(lat, lon);
}
```

  3. transformez cette latitude et longitude en geohash en utilisant le code suivant :

```
Location loc = new Location("selectedGeoHash", latCursor, lonCursor);
System.out.println(GeoHashHelper.getGeohash(loc));
```

- affichez dans la console le geohash de l'endroit sélectionné avec la souris

11. Traitement de données en format JSON

JSON est un format de données utilisé dans de nombreux langages de programmation de par sa facilité de lecture et d'écriture par un humain, et de sa facilité d'analyse et de génération pour une machine. Sa syntaxe est expliquée en détail sur https://www.json.org/json-fr.html :

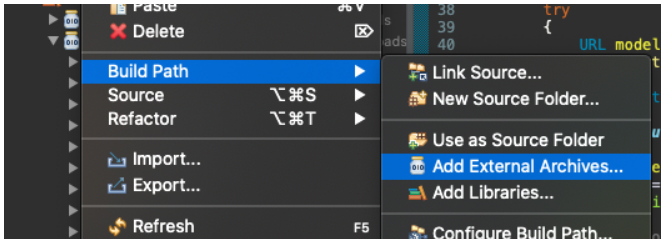
"Un **objet** [...] est un ensemble de couples nom/valeur non ordonnés. Un objet commence par une accolade gauche et se termine par une accolade droite. Chaque nom est suivi de deux-points et les couples nom/valeur sont séparés par une virgule."

"Un **tableau** est une collection de valeurs ordonnées. Un tableau commence par un crochet gauche et se termine par un crochet droit. Les valeurs sont séparées par une virgule."

"Une **valeur** peut être soit une chaîne de caractères entre guillemets, soit un nombre, soit true ou false ou null, soit un objet soit un tableau. Ces structures peuvent être imbriquées."

En Java, il existe plusieurs librairies permettant de générer ou lire du JSON. Nous allons dans ce tutoriel utiliser le package `org.json`.

- téléchargez depuis les ressources du tutoriel le fichier jar *json-20210307.jar* et ajouter le à votre projet



Dans ce package, chaque objet est représenté par un JSONObject et chaque tableau par un JSONArray. Pour obtenir une valeur dans un objet, on utilisera par exemple les fonctions :

- ➔ `getJSONObject(...)`
- ➔ `getJSONArray(...)`
- ➔ `getString(...)`
- ➔ `getInt(...)`
- ➔ `getFloat(...)`
- ➔ etc…

Les données que nous allons utiliser contiennent des informations sur des articles wikipedia. Elles sont stockées dans le fichier *data.json*, disponible dans les ressources du tutoriel sur le discord du cours.

- Ajoutez le fichier *data.json* à la racine de votre projet en utilisant l'explorateur de fichier. Il doit se situer dans le même répertoire que les répertoires *bin* et *src*.

- pour charger le contenu de ce fichier dans un objet JSONObject, nous allons utiliser les fonctions suivantes :

```
private static String readAll(Reader rd) throws IOException {
    StringBuilder sb = new StringBuilder();
    int cp;
    while ((cp = rd.read()) != -1) {
      sb.append((char) cp);
    }
    return sb.toString();
}
```

```
try (Reader reader = new FileReader("data.json")) {

    BufferedReader rd = new BufferedReader(reader);
    String jsonText = readAll(rd);
    JSONObject jsonRoot = new JSONObject(jsonText);

    // ...

} catch (IOException e) {
    e.printStackTrace();
}
```

Le contenu du fichier *data.json* est désormais chargé dans l'objet `jsonRoot`.
Nous allons maintenant nous intéresser à comment accéder au contenu de cet objet.

- ajoutez les lignes de codes suivantes pour parser l'objet `jsonRoot` :

```
JSONArray resultatRecherche = jsonRoot.getJSONObject("query").getJSONArray("search");
JSONObject article = resultatRecherche.getJSONObject(0);
System.out.println(article.getString("title"));
System.out.println(article.getString("snippet"));
```

Ce code permet d'afficher dans la console le titre et une courte description du premier article contenu dans *test.json*. Il procède pour cela de la façon suivante :

1. récupération de l'objet "*query*" depuis la racine
2. récupération du tableau "*search*" dans l'objet "*query*"
3. récupération du premier objet contenu dans le tableau "*search*"
4. récupération et affichage de la valeur "*title*"
5. récupération et affichage de la valeur "*snippet*"

- affichez dans la console de résultat de ce code
- affichez dans la console le nombre de mots de cet article
- affichez dans la console le titre du second article contenu dans *data.json*

12. Récupération de données JSON via une API web

De multiples API web proposent des données au format JSON. Nous avons vu comment exploiter des données contenus dans un fichier JSON local, nous allons maintenant voir comment obtenir ces données depuis un serveur distant via une API web.

- Ajoutez la fonction suivantes dans votre code :

```
public static JSONObject readJsonFromUrl(String url)
{
    String json = "";

    HttpClient client = HttpClient.newBuilder()
            .version(Version.HTTP_1_1)
            .followRedirects(Redirect.NORMAL)
            .connectTimeout(Duration.ofSeconds(20))
            .build();

    HttpRequest request = HttpRequest.newBuilder()
            .uri(URI.create(url))
            .timeout(Duration.ofMinutes(2))
            .header("Content-Type", "application/json")
        .GET()
        .build();

    try {
        json = client.sendAsync(request, BodyHandlers.ofString())
                .thenApply(HttpResponse::body).get(10,TimeUnit.SECONDS);
    } catch (Exception e) {
        e.printStackTrace();
    }

   return new JSONObject(json);
}
```

Vous pouvez désormais récupérer un JSONObject depuis l'URL d'une API Web, en renseignant le paramètre `url` de la fonction `readJsonFromUrl`.

Nous allons tester ces fonctionnalités avec l'API de wikipedia. La documentation de cette API est disponible à : https://www.mediawiki.org/wiki/API:Tutorial, et vous pouvez expérimenter les différents paramètres de l'url à : https://www.mediawiki.org/wiki/Special:ApiSandbox

- Affichez dans la console le tout début de l'article anglais de wikipedia sur les baleines

13. Git

Dans cette partie nous allons utiliser Git pour synchroniser le code d'un projet entre plusieurs développeurs.

Git est un outil open source et gratuit qui permet la gestion de versions. Il est très utile pour garder une trace des modifications apportées à un fichier contenant du code, et facilite la modification d'un même code par plusieurs personnes. Pour plus d'informations sur Git, vous pouvez visiter : https://git-scm.com/.

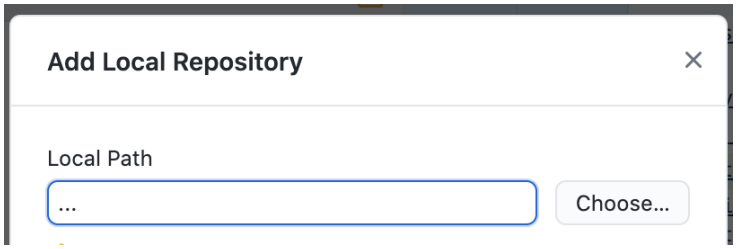
Création d'un répertoire synchronisé avec GitHub

Afin de stocker du code en ligne pour le rendre accessible à d'autres personnes, pour pouvoir le sauvegarder et garder une trace des différentes modifications, nous allons utiliser GitHub : https://github.com/. Il s'agit d'un service web d'hébergement et de gestion de développement de logiciels, basé sur Git.

- Créez un compte sur GitHub (si vous n'en possédez pas déjà un)

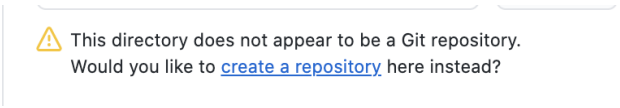
GitHub propose également une interface graphique pour git, GitHub desktop, que nous allons utiliser dans ce tutoriel. Vous êtes bien entendu libre d'utiliser un autre logiciel ou d'interagir avec Git en ligne de commande si vous le souhaitez.

- Téléchargez, installez et lancer GitHub Desktop https://desktop.github.com/
- Renseignez votre compte GitHub
- Cliquez sur Fichier, Add Local Repository
- Renseignez le répertoire racine de votre projet contenant les fichiers java

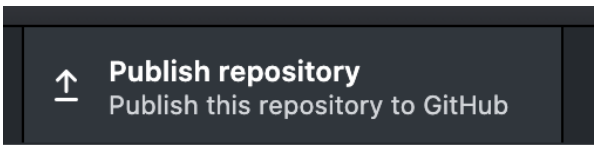


Le répertoire n'est pas un répertoire connu de Git : un message d'avertissement doit apparaître et le bouton Add Repository est inactif. Pour remédier à cela :

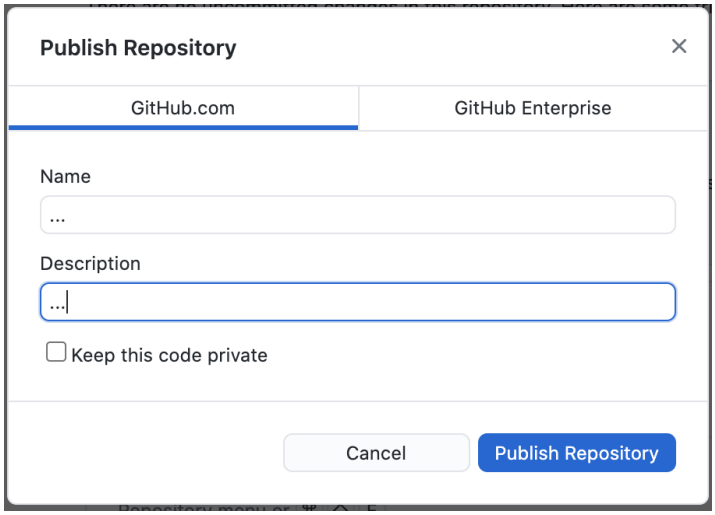
- Cliquez sur le lien pour créer un répertoire. Vérifier que le répertoire à créer est bien celui que vous avez renseigné dans l'étape précédente



- Appuyez sur le bouton Publish



- Vérifiez que l'onglet GitHub.com est bien sélectionné, que le nom et la description de votre projet sont bien remplies et que le code est public (*Keep this code private* décoché) puis appuyez sur Publish Repository



Le répertoire que vous avez renseigné est désormais synchronisé avec GitHub. Si vous vous connectez sur https://github.com/ avec votre compte vous pouvez voir les fichiers que vous venez d'ajouter.

Synchronisation des modifications

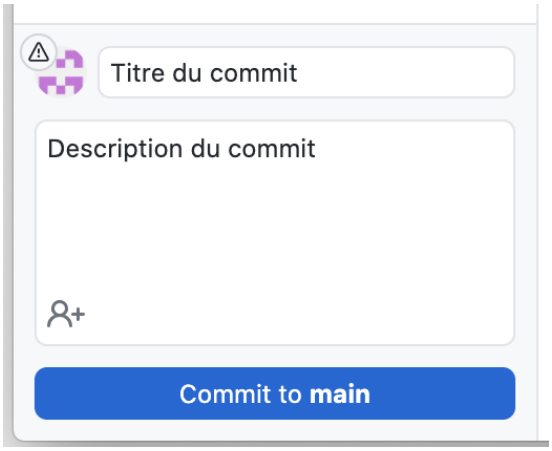
- sur votre ordinateur, modifiez un des fichiers que vous avez renseigné

GitHub desktop doit afficher les modifications que vous venez d'effectuer. Cependant, ces modifications ne sont effectives que sur votre ordinateur. Pour synchroniser ces modifications avec github, il va falloir réaliser deux étapes :

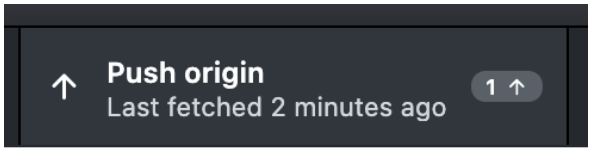
- un commit
- un push

Un commit contient des informations sur les lignes de code modifiées, l'utilisateur qui réalise le commit ainsi qu'une description donnée par l'utilisateur. Un commit est fait localement : pour qu'il soit propagé sur le répertoire distant il faut réaliser un push.

- en bas à gauche de l'interface renseigner un titre et une description et valider un commit



- effectuer un push en appuyant sur le bouton en haut à droite

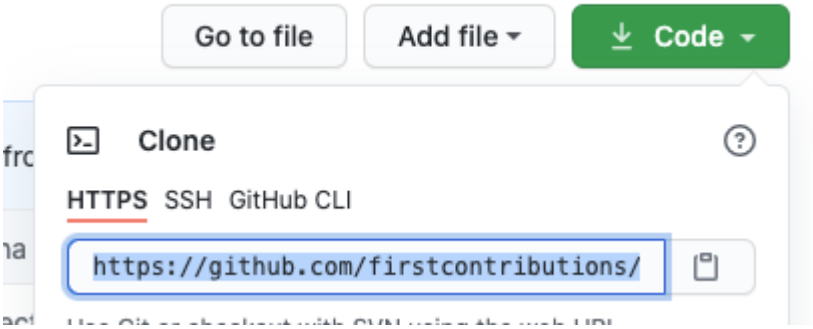


Les modifications sont désormais effectives sur le répertoire distant.

Partager un répertoire synchronisé avec GitHub

Vous pouvez partager votre projet à une autre personne en lui envoyant le lien vers votre répertoire github. Pour cela :

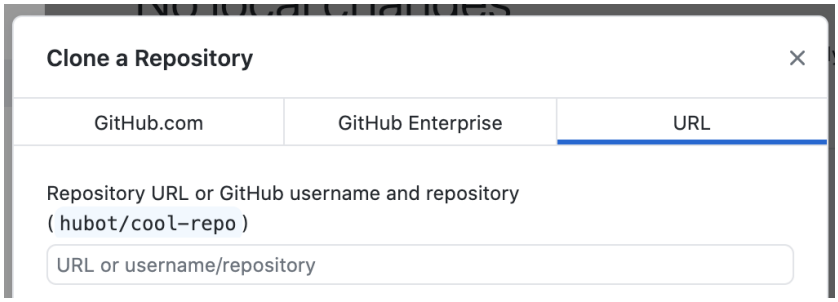
- Allez sur la page de votre projet sur GitHub
- Cliquez sur le bouton vert Code et copiez le lien https



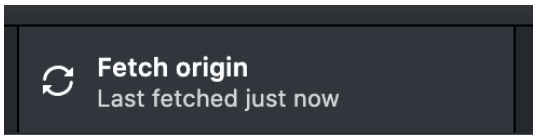
- Envoyez le lien à une autre personne

Lorsque vous obtenez un lien vers un répertoire GitHub :

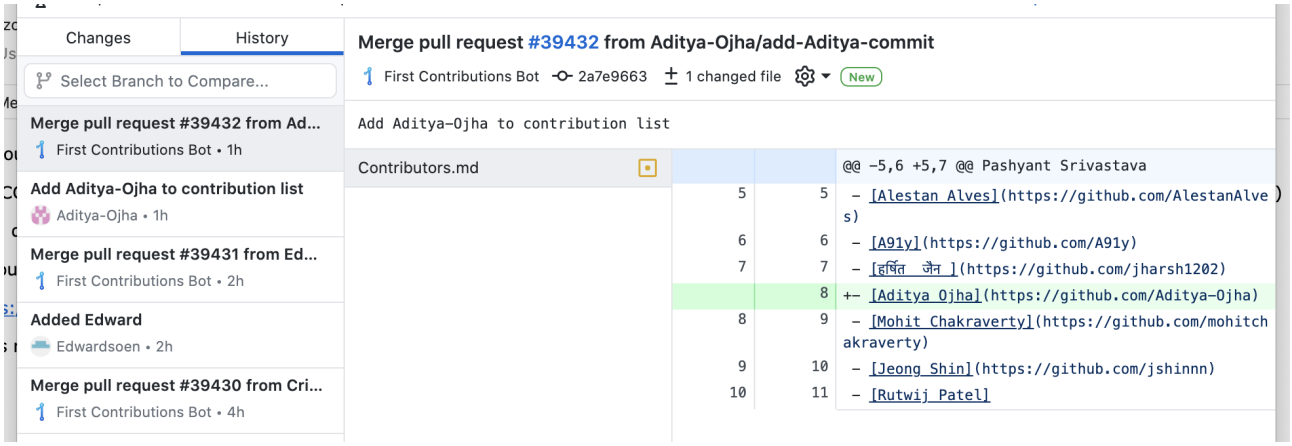
- Dans GitHub Desktop cliquez sur Fichier, Clone Repository...
- Renseigner le lien dans l'onglet URL, et l'emplacement local ou vous souhaitez télécharger le projet



- Cliquez sur fetch origin



Le code est désormais présent localement sur votre machine. Vous avez également accès à l'historique des commits du projet en cliquant sur History :



Travailler à plusieurs avec un répertoire synchronisé

Git est un outil puissant pour synchroniser les modifications de code effectuées par plusieurs personnes sur des machines différentes.

Lorsque l'une des personnes souhaite apporter des modification au code d'un répertoire synchroniser, elle doit :

1. Effectuer un *fetch*
2. Réaliser ses modifications
3. Réaliser un *commit*
4. Effectuer un *push*

L'ensemble de ces étapes permet de

1. récupérer d'éventuelles modifications du code non synchronisé sur la machine locale
2. enregistrer une ou plusieurs modifications
3. rendre accessible ces modifications

14. Contacts

Groupe 1 : Arthur FAGES – arthur.fages@universite-paris-saclay.fr
Groupe 2 : Tifanie BOUCHARA - tifanie.bouchara@universite-paris-saclay.fr