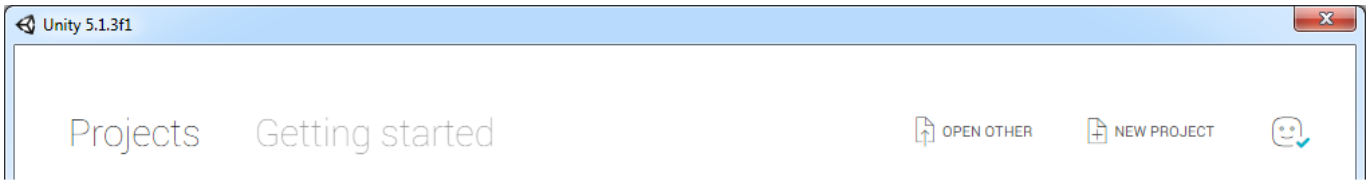
TD1 : Fondamentaux Unity

Rémy Frenoy, Florian Jeanne, Yann Soullard

Unity est un logiciel et un moteur de jeu multi-plateforme (PC, Mac, Linux, mobiles, tablettes, consoles de jeux et applications web). Il est principalement utilisé dans le domaine des jeux-vidéos, mais également en réalité virtuelle.

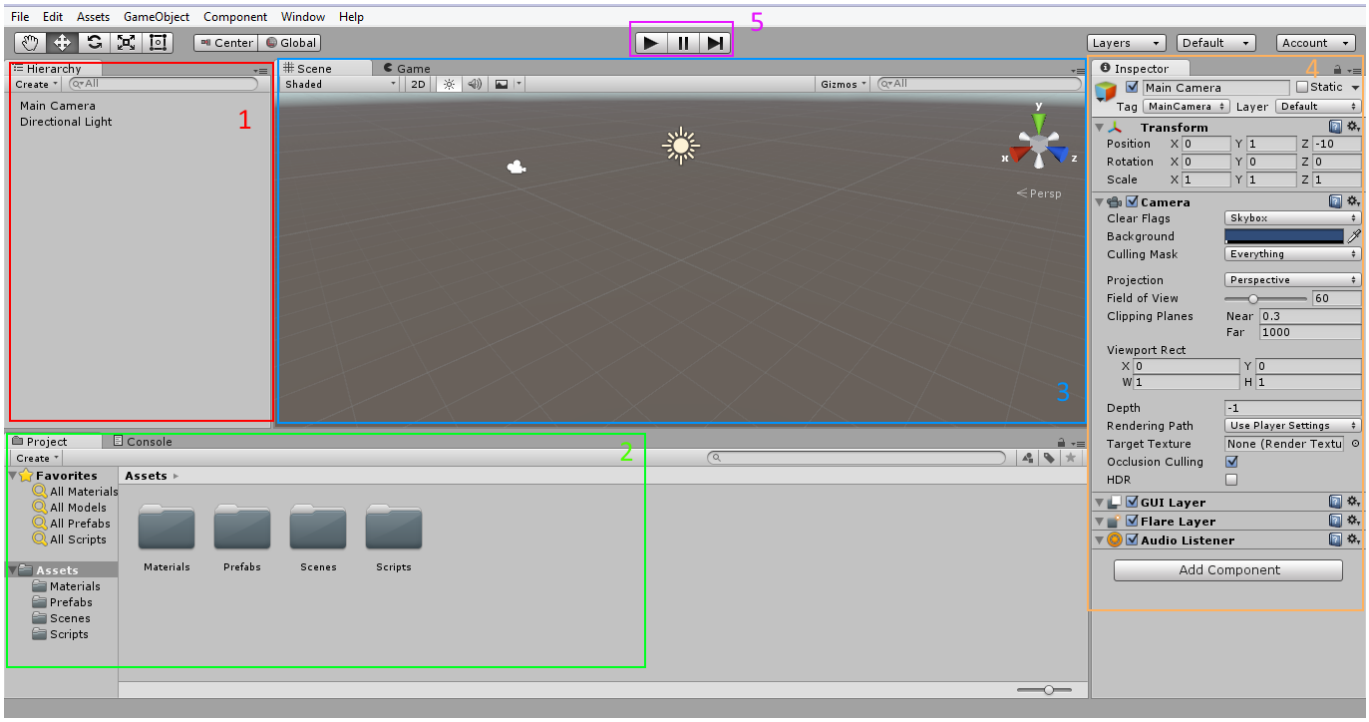
Création d'un projet

Lors du lancement de Unity, celui-ci vous propose d'ouvrir un projet existant, ou d'en créer un nouveau. Choisissez donc New Project, et nommez-le "TD1_nombinome1-nombinome2".



Il est conseillé de regrouper l'ensemble de vos TDs dans un dossier "RV01" sur votre disque Z afin de pouvoir accéder à vos sources facilement lors des TDs suivants.

Interface graphique de Unity



1 (rouge) : **Hierarchy** – Hiérarchie des éléments disposés sur la scène.

2 (vert) : **Project** – Ensemble des ressources du projet.

3 (bleu) : **Scene** – Scène d'édition, permet d'ajouter, modifier, supprimer des éléments de l'environnement.

3 bis (bleu) **Game** – Affiche l'application une fois celle-ci lancée.

4 (orange) : **Inspector** – Affiche les propriétés et les options de configuration de l'objet sélectionné.

5 (mauve) : **Control bar** – Permet de lancer, suspendre et arrêter l'application en cours.

Sauvegarder la scène

Lors de la création d'un nouveau projet, Unity crée une première scène. Dans la fenêtre *project*, créez un nouveau dossier "Scènes". Sauvegardez-y votre scène : File > Save Scene et nommez-la "Main".

La quantité de ressources présentes dans un projet Unity peut rapidement être imposante. Ces ressources sont de plus très variées (scènes, scripts, matériaux, ...). Nous nous attacherons donc à organiser rigoureusement nos ressources dans des dossiers correspondants à leur nature (les scènes dans le dossier "Scènes", les scripts dans le dossier "Scripts"... enfin vous saisissez l'idée quoi). Toutes les ressources Unity sont considérées comme des "assets", ainsi tous les dossiers que vous créerez seront à placer dans le dossier "Assets" que Unity crée automatiquement à la création d'un projet.

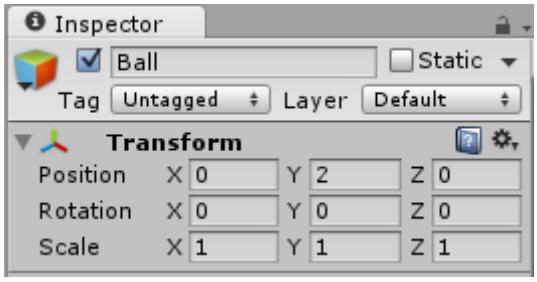
Créer un objet

Nous sommes maintenant familiarisés avec l'interface. Cependant, notre scène est vide... Ou presque. Comme nous pouvons le voir dans la fenêtre *hierarchy*, il y a deux éléments : *Main Camera* et *Directional Light*, qui sont une caméra par défaut et une lumière d'ambiance.

Nous allons commencer par créer des objets simples, comme une balle et un sol par exemple (dans Unity, un objet est appelé *GameObject*). Pour cela, il vous suffit d'aller dans le menu puis dans **GameObject > 3D Object > Plane**, pour le sol. Renommez-le en "ground" dans la fenêtre *hierarchy* (clic droit sur "plane", puis *rename*). Faites la même chose pour la balle que l'on nommera "Ball" : **GameObject > 3D Object > Sphere**.

Néanmoins, notre balle est coincée dans notre plan. Il nous faut donc la déplacer. Sélectionnez la balle avec la souris à partir de la scène ou tout simplement dans la fenêtre *hierarchy*. Puis, dans la fenêtre *inspector*, localisez la partie *Transform*. C'est ici que nous allons pouvoir modifier la position, la taille et l'orientation de nos objets. Tous les *GameObjects* possède un *component transform*. Dans Unity, l'axe par défaut pour la hauteur

est l'axe y. Il nous faut donc modifier la valeur de la position comme ceci, avec une valeur de 2 par exemple :



Désormais, notre balle est au dessus du sol

Déplacer un objet

Maintenant que notre scène possède des objets, il serait intéressant de pouvoir les déplacer. Pour cela, nous allons devoir ajouter une nouvelle propriété à notre balle (ou *component*) afin de lui ajouter de la physique. Sélectionnez donc la balle puis ajoutez-lui un *rigidbody* : **Component > Physics > Rigidbody**. Un nouveau *component* est alors ajouté à notre balle dans l'*inspector*.

Lancez la scène grâce au bouton *Play*. Miracle, la balle tombe !

Cependant, nous ne pouvons toujours pas la déplacer. Pour cela, nous allons utiliser des scripts. Unity supporte différents langages de programmation : le Javascript, le C# et le Boo (un langage propre à Unity). Durant les TDs, vous pourrez utiliser le langage qui vous plaît le plus. Néanmoins, le code que nous serons amenés à vous fournir sera en C#.

Avant de créer notre premier script et dans un souci de propreté, nous allons créer un dossier "Scripts" dans notre projet où seront stockés tous nos scripts. Pour cela, faites un clic droit dans la fenêtre **Project > Create > Folder**. Maintenant, faites un **clic droit dans ce dossier > Create > C# Script** (ou allez dans **Assets > Create > C# Script**). Un fichier apparaît dans l'explorateur du projet. Nommez-le "BallController". Ce script va nous

servir à contrôler le déplacement de notre balle. Il faut donc l'associer au *GameObject* "Ball". Un simple glisser-déposer du script sur la balle suffit. Vous pouvez également le faire depuis l'*inspector* avec le bouton *Add Component* (Scripts > Ball Control). Le script apparaît maintenant comme *component*.

Unity comprend un environnement de développement appelé MonoDevelop. Pour éditer notre script, double-cliquez sur lui et MonoDevelop s'ouvrira alors automatiquement. Il comprend par défaut deux méthodes : Start() et Update().

- Start() contient les instructions qui seront exécutées au chargement du script, à sa première exécution. Elle n'est donc appelée qu'une seule fois ;
- Update() est appelée à chaque frame, et contient donc les instructions qui seront exécutées en boucle.

Pour pouvoir déplacer la balle, nous allons donc modifier Update().

Le script agit comme un *component* de la balle, il a donc accès aux autres composants de celle-ci ; c'est notamment pour cela que nous le lions à la balle.

Définissez une nouvelle méthode, que l'on nommera KeyboardMovements(). Cette méthode a pour objectif de détecter les touches du clavier qui sont utilisées et d'appliquer une force sur notre balle dans la direction souhaitée. Pour cela, nous allons tout d'abord créer une variable publique "speed" avec la valeur 2 par exemple.

En C# :

```
public int speed = 2;
```

De plus, comme nous l'avons vu précédemment, la physique de la balle est gérée par son *rigidbody*. Il nous faut donc appeler la méthode *AddForce()* de celui-ci pour appliquer une force (voir http://docs.unity3d.com/ScriptReference/Rigidbody.AddForce.html). Nous allons donc commencer par récupérer le *rigidbody* de la balle dans notre script et l'associer à une variable, pour pouvoir par la suite appliquer nos AddForce(). Pour cela, nous utilisons la

méthode *GetComponent<>()* (voir la doc Unity) dans la méthode Start(), et nous récupérons le résultat dans une variable privée de type Rigidbody, créée au préalable. Nous avons maintenant tous les éléments pour définir KeyboardMovements() qui sera appelée dans notre Update() :

- Input.GetKey(String) pour déterminer quelle touche est pressée ;
- AddForce(Vector3) appliquée à un rigidbody pour appliquer une force. Comme indiqué dans la documentation Unity, le paramètre est un Vector3. Il existe des Vector3 de base qui sont préétablis, comme Vector3.left qui correspond au vecteur (-1, 0, 0).
- Nous multiplierons ce Vector3 par la variable speed que nous avons créée.

```
using UnityEngine;
using System.Collections;

public class BallController : MonoBehaviour {
    public int speed = 2;
    private Rigidbody rb;

    void Start()
    {
        rb = GetComponent<Rigidbody>();
    }
    void Update()
    {
        KeyboardMovements();
    }

    void KeyboardMovements()
    {
        if(Input.GetKey("right"))
        {
            rb.AddForce(Vector3.right * speed);
        }
        else if(Input.GetKey("left"))
        {
            rb.AddForce(Vector3.left * speed);
```

```
        }
        else if(Input.GetKey("up"))
        {
            rb.AddForce(Vector3.forward * speed);
        }
        else if(Input.GetKey("down"))
        {
            rb.AddForce(Vector3.back * speed);
        }
    }
}
```

Une fois la méthode terminée, n'oubliez pas de l'appeler dans Update(). Sauvegardez, puis lancez l'application. Les scripts sont automatiquement compilés par Unity. De plus, s'il y a des erreurs dans vos scripts, elles apparaîtront dans la fenêtre *Console*. Maintenant, essayez de déplacer votre balle en évitant qu'elle ne tombe du plan !

Placer la caméra

Unity place automatiquement une caméra sur la scène. Vous pouvez sélectionner cette caméra en choisissant *Main Camera* dans la fenêtre *hierarchy*. Une popup *Camera Preview* s'affiche alors et vous indique la vue depuis la dite caméra. Vous remarquerez, en scrutant l'*inspector* que la caméra est positionnée en (0, 1, -10). Déplacez la caméra de telle sorte que vous puissiez voir le plan et la sphère dans *Camera Preview*.

Lancez l'application. Déplacer la sphère afin qu'elle sorte du champ de la caméra. On ne la voit plus (forcément…). Une question existentielle vous vient alors : comment faire pour que la caméra suive la sphère ?

Mieux placer la caméra

La hiérarchie des objets permet de définir un objet comme étant enfant d'un autre. Glissez-déposez *Main Camera* sur *Ball* dans la fenêtre *hierarchy*, la caméra doit désormais être

“enfant” de la sphère.

La caméra est “enfant” de la sphère

Lorsqu’un objet *child* est enfant d’un objet *parent*, sa position devient relative à la position du parent. C’est à dire que le référentiel de l’objet enfant n’est plus le centre de la scène mais le centre de son objet “parent”. Relancez l’application. La position initiale de la caméra reste inchangée. Déplacez la sphère. La caméra se déplace désormais relativement au déplacement de la sphère. Ça vous plaît ? Mieux vaut avoir l’estomac bien accroché…

La méthode consistant à placer la caméra comme étant l’enfant d’un objet que l’on souhaite suivre peut être tout à fait pertinente, dans le cas d’un personnage à la première/troisième personne par exemple. En vue à la première personne, disposer la caméra comme “enfant” permet par exemple de suivre les déplacements et les mouvements de la tête. Attention toutefois, comme ici, aux déplacements de l’objet “parent”. Lorsque celui-ci est amené à faire de nombreuses rotations (typiquement le cas d’une sphère !), mieux vaut utiliser une autre méthode.

Correctement placer la caméra

On souhaite que la caméra suive les translations de la sphère dans l’espace, mais pas les rotations. La caméra ne peut donc pas avoir comme référentiel celui de la balle. Dans la hiérarchie, faites donc en sorte que la caméra ne soit plus enfant de la sphère.

La caméra n’est plus

“enfant” de la sphère

On se retrouve dans la configuration initiale, avec une caméra fixe. Pour modifier cela, ajoutons un script *CameraController*. Ouvrez le script dans MonoDevelop. Dans ce script, on souhaite que tout changement de position (c’est à dire une translation dans l’espace) d’un GameObject modifie de la même manière la position de la caméra. Nous avons dont besoin de deux variables :

- Une variable que l’on nommera *baseObject*, de type GameObject et qui correspondra à l’objet que l’on souhaite suivre.
- Une variable que l’on nommera *offset*, de type Vector3 et qui correspond au vecteur représentant l’écart entre la position initiale de la caméra et la position initiale de l’objet à suivre.

L’esprit de ce script est donc de repositionner la caméra afin que l’offset soit constant.

Il suffit donc de calculer l’offset au chargement de la scène (donc dans la méthode Start) de la façon suivante :

```
offset = transform.position - baseObject.transform.position;
```

A chaque frame (donc dans la méthode Update), on modifie la position de la caméra comme étant la position de l’objet à suivre plus l’offset.

```
transform.position = baseObject.transform.position + offset;
```

On obtient alors le code suivant :

```
using UnityEngine;
```

```
using System.Collections;

public class CameraController : MonoBehaviour {

    public GameObject baseObject; // l'objet à suivre

    private Vector3 offset; // L'offset initial

    void Start ()
    {
        offset = transform.position - baseObject.transform.position;
    }

    void Update ()
    {
        transform.position = baseObject.transform.position + offset;
    }
}
```

Ajoutez le script à votre objet caméra, et lancez l'application. Une erreur de compilation sauvage apparaît !!!

Oups !!!

En effet, nous avons déclaré une variable baseObject dans le script, mais Unity n'a aucune idée de quel objet il s'agit. Et c'est donc à nous de lui donner cette information. Sélectionnez la caméra dans la fenêtre *hierarchy* afin que ses propriétés apparaissent dans la fenêtre *inspector*. Dans le cadre Camera Controller, vous voyez votre script, et vous voyez également la variable Base Object qui est pour l'instant à None. D'où l'erreur de compilation !

La variable pointe vers None

Depuis la fenêtre *hierarchy*, glissez-déposez la balle vers la valeur de BaseObject dans la fenêtre *inspector*. Vous obtenez ceci :

Etudiant utilise l'attaque "affectation de variable". C'est très efficace !!

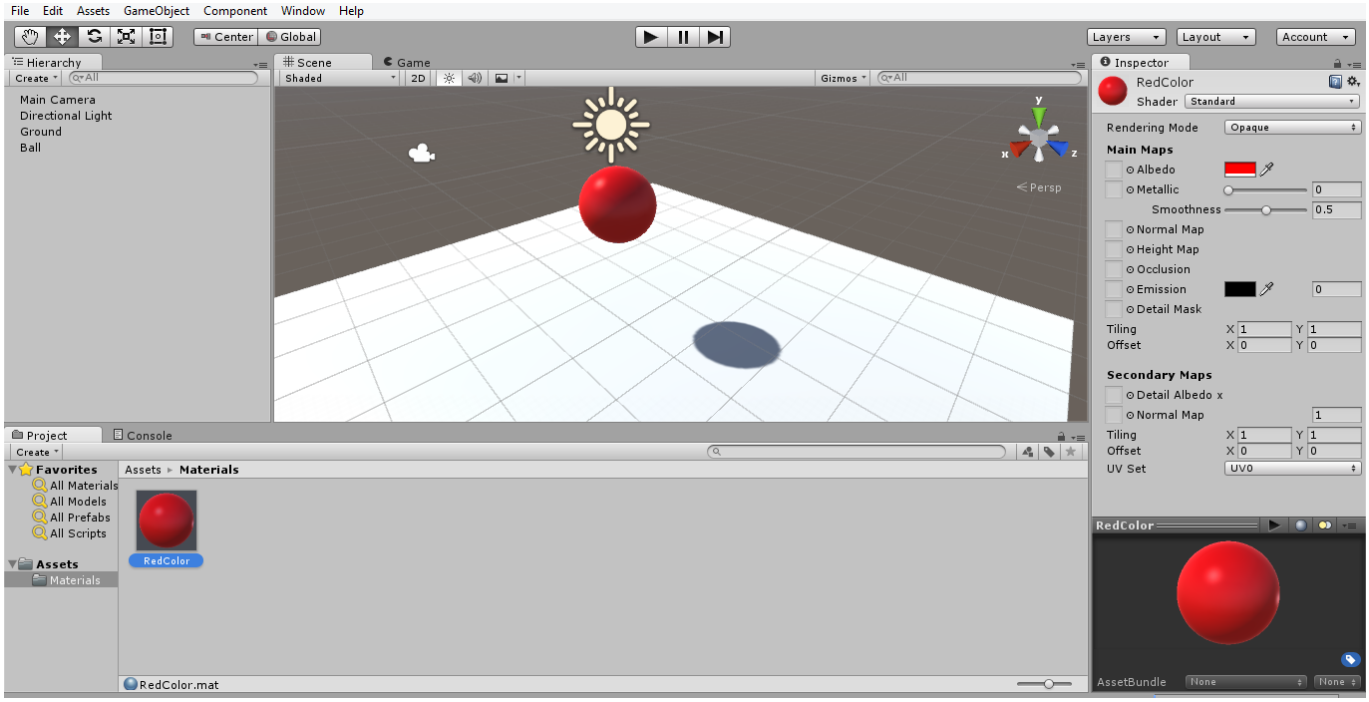
Lancez l'application, et déplacez la balle. Vous remarquez que la caméra suit les déplacements de la balle, sans en subir les rotations. Parfait !

C'est beau, c'est… texturé

Cette scène est un peu terne. Ajoutons-y un petit peu de couleur. Pour modifier l'apparence d'un objet, celui-ci doit avoir un *material*.

Commençons par créer un dossier "Materials" dans la fenêtre *Project*, qui permettra de stocker l'ensemble des *materials* utilisés dans un projet. Double-cliquez sur le dossier nouvellement créé, il est vide… (sans blague !)

Pour créer un *material*, faites un **clic droit > Create > Material**. Renommez-le "RedColor". Pour donner une couleur rouge à votre *material*, modifier la propriété "Albedo" dans la fenêtre *Inspector* et attribuez-lui la couleur rouge. Pour appliquer un *material* sur un objet, il suffit de glisser-déposer le *material* depuis la fenêtre *Project* vers l'objet voulu sur la scène.

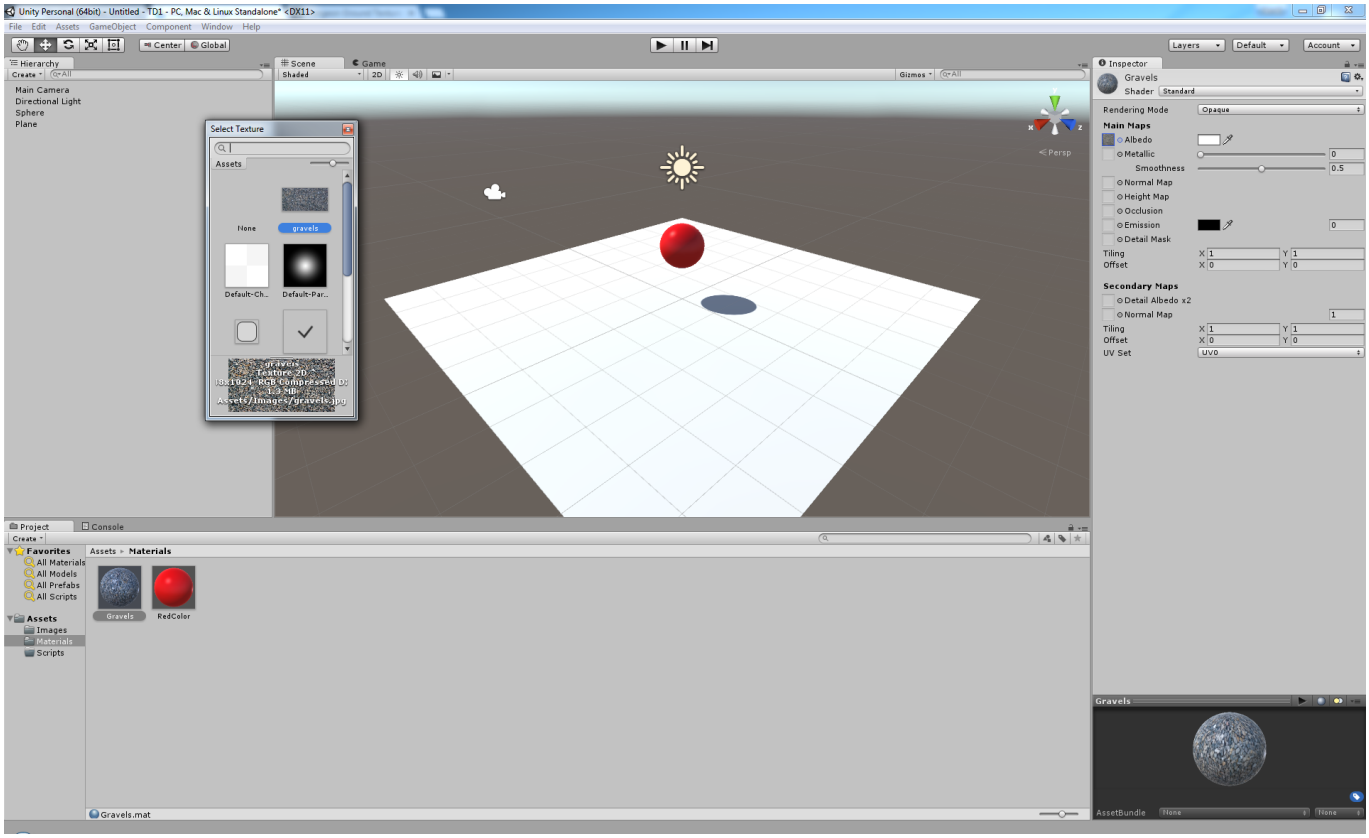


Que c'est beau !!!!

Il est également possible d'utiliser des images plutôt que de simples couleurs. Créer un dossier "Images" dans le dossier "Assets". Copiez-y la ressource "gravels.jpg" que vous pouvez télécharger ici :

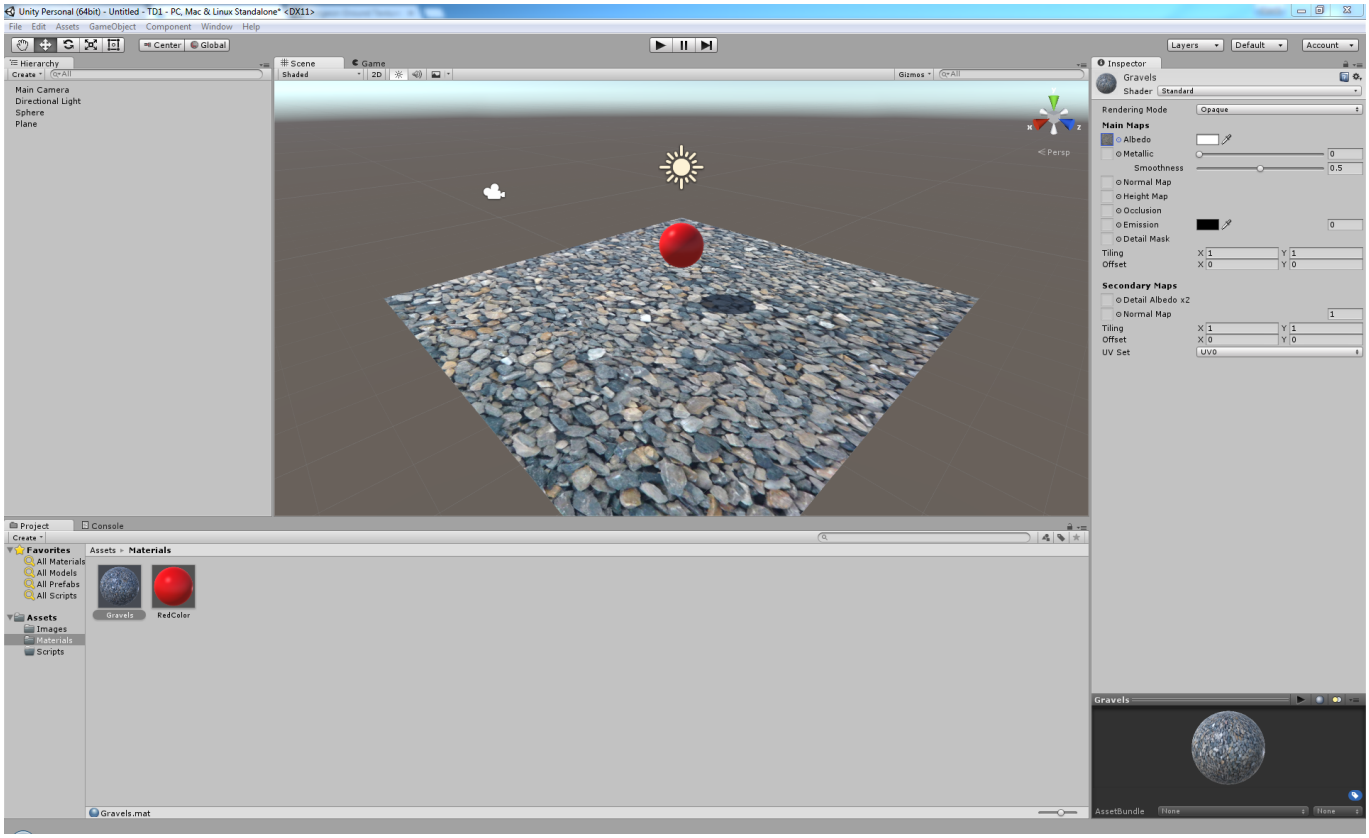
Cliquez pour télécharger

Créez maintenant un nouveau material "Gravels". Sélectionnez le material nouvellement créé afin qu'il s'affiche dans la fenêtre *inspector.* Cliquez sur le petit cercle juste à gauche de l'option "Albedo" et choisissez l'image gravels dans la fenêtre "Select Texture" qui vient de s'afficher.



La fenêtre de sélection de texture

Comme nous l'avons fait précédemment pour la sphère, glissez-déposez le nouveau material sur le plan de la scène.



Une telle beauté m'émeut…

Export

Notre projet terminé, nous souhaitons l'exporter (si votre TD n'est pas terminé, exportez-le quand même, c'est facile mais important). Pour cela, faites **Assets > Export Package**. Vérifiez bien que tout est coché, puis confirmez. Vous obtenez un fichier portant l'extension *unitypackage*. Vous pouvez désormais importer dans n'importe quel projet (nous le ferons lors du TD2) les assets (dont la scène) que vous venez d'exporter.

Pour aller plus loin

Les lumières

Comme son nom l'indique, la lumière placée automatiquement lors de la création d'une scène est de type directionnelle. Vous pourrez trouver de nombreux autres types de lumière (Point Light, Spot Light, Area Light) dans GameObject >Light. Vous pouvez également modifier les propriétés de ces lumières dans la fenêtre *hierarchy*, une fois la lumière sélectionnée.

L'Asset Store

Unity possède un asset store qui permet de télécharger des contenus gratuits ou payants afin de les intégrer à un projet. Vous pouvez ouvrir l'asset store depuis Window > Asset Store. Les contenus sont classés par catégories (3D Models, Textures and Materials, ...). Lorsque vous trouvez un contenu qui vous plait, il vous suffit de le télécharger pour que Unity l'intègre dans votre projet en cours. Vous pouvez par exemple modifier la texture du sol en utilisant une texture de l'asset store.

Parmi les contenus, vous trouverez notamment des fichiers portant les extensions **fbx** et **obj**. FBX et OBJ font parti des principaux format de fichiers 3D. Des logiciels de modélisation tels que Blender ou 3ds Max permettent de créer de tels fichiers.