# Tutoriel 4: les bases d'Unity 3D


## 1. Programmation d'interaction simple avec la souris

Nous souhaitons créer un nouveau script permettant de pousser les cubes lorsque l'on clique dessus et de changer la couleur du cube change lorsque l'on passe la souris sur ce cube. Pour cela, nous allons utiliser les fonctions `OnMouse...` de la classe mère `MonoBehaviour` :

- `void OnMouseDown ()`
- `void OnMouseDrag ()`
- `void OnMouseEnter ()`
- `void OnMouseExit ()`
- `void OnMouseOver ()`
- `void OnMouseUp ()`

- Créer un nouveau script C# que vous nommerez `MouseInteraction`.
- Créer deux attributs privés de type `Rigidbody` et `Renderer` afin de pouvoir stocker des liens vers le `RigidBody` (pour la physique) et le `Renderer` de l'objet (pour pouvoir accéder au `material` et changer la couleur).
- Dans la fonction Start(), initialiser vos deux attributs en utilisant les fonctions : `GetComponent<Rigidbody>()` et `GetComponent<Renderer>()`
- Pour pousser le cube, vous pourrez appliquer une force sur son Rigidbody en utilisant la fonction `AddForce(`*Vector3D val_dir_force*`)`. L'attribut `Camera.main` permet d'accéder à la caméra principale et `Camera.main.transform.forward` donne le vecteur pointant vers l'avant de la caméra. Cependant, il faudra le multiplier pour obtenir une force suffisante.
- Pour changer la couleur, vous pourrez accéder à l'attribut `.material` sur le `Renderer` et à l'attribut `.color` sur le `material` : `rend.material.color`. Cela devrait vous permettre à la fois de stocker la couleur initiale de l'objet (pour la remettre une fois que la souris aura quitté le cube) et de changer cette couleur en utilisant une couleur prédéfinie comme : `Color.red`.

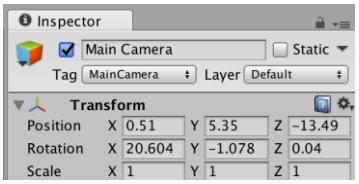
Pour plus d'information sur classe `MonoBehaviour` et ses fonctions, aller voir la documentation : https://docs.unity3d.com/ScriptReference/MonoBehaviour.html

## 2. Navigation en ré-utilisant des scripts d'interaction clavier/souris

Unity 3D propose plusieurs composants pour gérer différents types d'interaction de base. Dans cette partie, nous allons ajouter un composant pour pouvoir naviguer dans la scène à la façon d'un FPS :

- Importer le package `Characters` (`Assets > Import Package > Characters`).

- Ajouter le `prefab` nommé `FPSController` dans l'arborescence de la scène (2). Il possède sa propre `Main Camera` : désactiver la `Main Camera` utilisée précédemment en décochant la case dans l'inspecteur (3).




- Modifier la position du `FPSController` pour avoir une vue à la 1ère personne de la pile de cubes, en étant posé sur le plan.
- Exécuter la scène, tester les commandes (flèches, barre d'espace, souris) et écouter aussi le son produit.

## 3. Création dynamique d'objets

Nous voulons maintenant créer dynamiquement des objets dans la scène 3D. Pour cela, Unity 3D propose une fonction `Instantiate(`*GameObject obj*`)` qui permet d'instancier dynamiquement dans la scène une copie de l'objet passer en paramètre. L'objet passé en paramètre peut être un objet déjà présent dans la scène ou bien un `prefab` si l'on ne veut pas que l'objet soit déjà dans la scène.

Nous souhaitons faire en sorte que des cubes jaunes soit lancés dans la direction de la caméra lorsque l'utilisateur appuie sur le bouton droit de la souris :

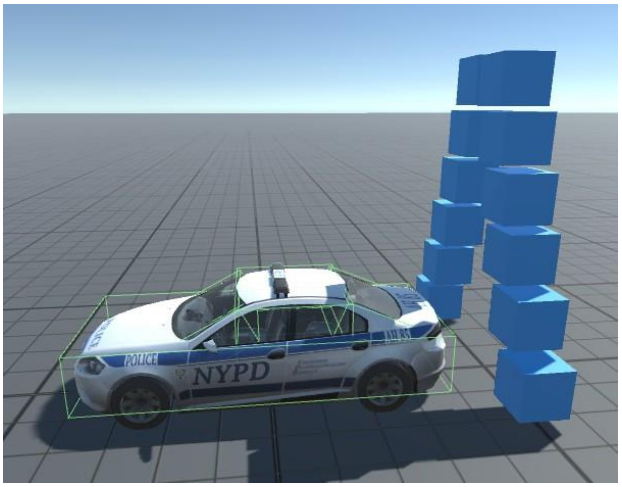
- Créer un nouveau script C# que vous nommerez `CreateOnClick`.
- Ajouter ce script au `FirstPersonCharacter` (l'object qui suit les mouvements de la souris) qui est le fils du `FPSController`. C'est aussi lui qui porte la caméra.
- Créer un attribut `public GameObject aCopier` qui vous permettra de stocker une référence sur l'objet à instancier. Il faudra initialiser cet attribut en faisant un glisser-déposer du `CubePrefab` dans le champs de l'attribut dans l'inspecteur (3) lorsque l'on a sélectionné le `FirstPersonCharacter`.
- Par défaut dans les `Input` du projet, le clique droit génère l'événement « `Fire2` » que vous pourrez capter avec les fonctions `getButton`… comme pour l'évènement « `ActivateRotation` ».
- A chaque clique droit :
  - Instancier un nouveau cube avec la fonction `Instantiate(`*GameObject obj*`)`,
  - Changer sa couleur,
  - Placer le devant le `FirstPersonCharacter`, sinon il sera éjecté dans n'importe quelle direction (collision avec le `FirstPersonCharacter`),
  - Appliquer lui une force pour le lancer en avant.

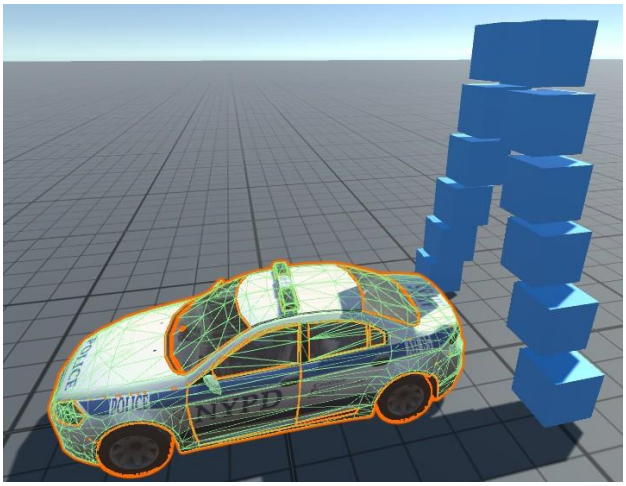
Noter que l'on peut aussi supprimer des objets de la scène grâce à la fonction `Destroy(`*GameObject obj*`)`.

## 4. Charger une géométrie

Nous souhaitons charger le modèle 3D d'une voiture avec des textures plaquées dessus :

- Télécharger l'archive *Ford_Mondeo.zip* sur le site du cours. Décompresser cette archive. Il contient un .obj décrivant le mesh, un .mtl décrivant les couleurs et les placements des textures, ainsi que les images de texture .tga.
- Effectuer un glisser-déposer de tout le dossier contenant les différents fichiers du modèle dans l'explorateur de projet (4). Le modèle devrait être chargé dans Unity 3D.
- Glisser-déposer le modèle dans l'arborescence de la scène (2) pour l'ajouter à la scène. Ajuster sa taille pour qu'il soit à la même échelle que les cubes et sa position pour qu'il se trouve près de la pile de cubes.
- Le chargement de nouveaux modèles 3D souffre souvent de petits problèmes de compatibilité dans toutes les librairies 3D. Observerer qu'ici, la transparence des vitres de la voiture n'a pas été conservée. Trouver le `material` qui correspond aux vitres et changer son `Rendering Mode`, ainsi que la composante alpha (`A`) de la couleur de la `Main Maps` afin de rendre les vitres transparentes.
- Essayer de lancer des cubes sur la voiture ou de marcher au travers. Qu'observez-vous ? Pourquoi ?
- Trouver une solution pour régler ce problème. Vous pourrez utiliser des `Box Colliders` que vous placerez vous même ou des `Mesh Colliders` que vous ajouterez aux parties adéquates de la voiture (cf. images sur la page suivante). La 2nd solution est plus précise, mais plus couteuse en temps de calcul.




## 5. Créer un exécutable pour la scène

Enfin, nous voulons créer un exécutable pour cette scène afin de pouvoir visualiser la scène en plein écran et éventuellement de la distribuer :

- Aller dans `File > Build Settings…`
- Sélectionner la cible `PC, Mac & Linux Standalone`. Observez qu'il y a d'autres cibles intéressantes comme par exemple WebGL (cf. l'exécutable de la scène du tutoriel sur le site du cours).
- Cliquer sur `Build`.
- Tester votre exécutable.

Vous pouvez ensuite cliquer sur `Player Settings…` en bas de la fenêtre `Build Settings…` et explorer les différents paramètres du *player* (fenêtre de dialogue au lancement, resolution, etc.).

## 6. Pour aller plus loin

Explorer les Assets, les GameObjects, les Components et la documentation pour ajouter des propriétés ou des animations à votre scène (i.e. particules, collisions...).

En particulier, vous pouvez charger le package `vehicles`, tester et essayer de comprendre comment fonctionnent les différents véhicules.

## 7. Contact

Cédric FLEURY - cedric.fleury@lri.fr