POLYTECH PARIS-SUD

Option Réalité Virtuelle et Interaction (RVI)
Spécialité Informatique, 5ème année 2016 - 2017

Tutoriel : les bases d'Unity 3D

https://unity3d.com/

Documentation : https://docs.unity3d.com/
Tutoriels : https://unity3d.com/fr/learn/tutorials/

1. Lancer Unity 3D et créer un nouveau projet

Lancer le logiciel Unity 3D. Une fenêtre de dialogue apparaît, vous demandant de vous connecter avec un compte utilisateur. Vous avez donc deux options : soit vous vous créez un compte maintenant, soit vous choisissez l'option « `Work offline` ». Vous pourrez réaliser ce tutoriel sans problème en choisissant l'option « `Work offline` ». Cependant, si plus tard vous souhaitez récupérer des données de l'*asset store* Unity 3D (modèles 3D, scripts, etc.), vous aurez besoin d'un compte (il sera possible de créer et de vous connecter avec un compte plus tard même si vous choisissez l'option « `Work offline` » maintenant).

- Créer un nouveau projet Unity 3D en cliquant sur « `NEW` ». Choisir `3D`, ainsi que le nom et la destination de votre projet. Vous n'avez pas besoin de charger de packages prédéfinis pour le moment (vous pourrez le faire par la suite ci-besoin).
- Enregistrer la scène courante (extension *.unity*) en lui donnant un nom (`File > Save Scenes`).

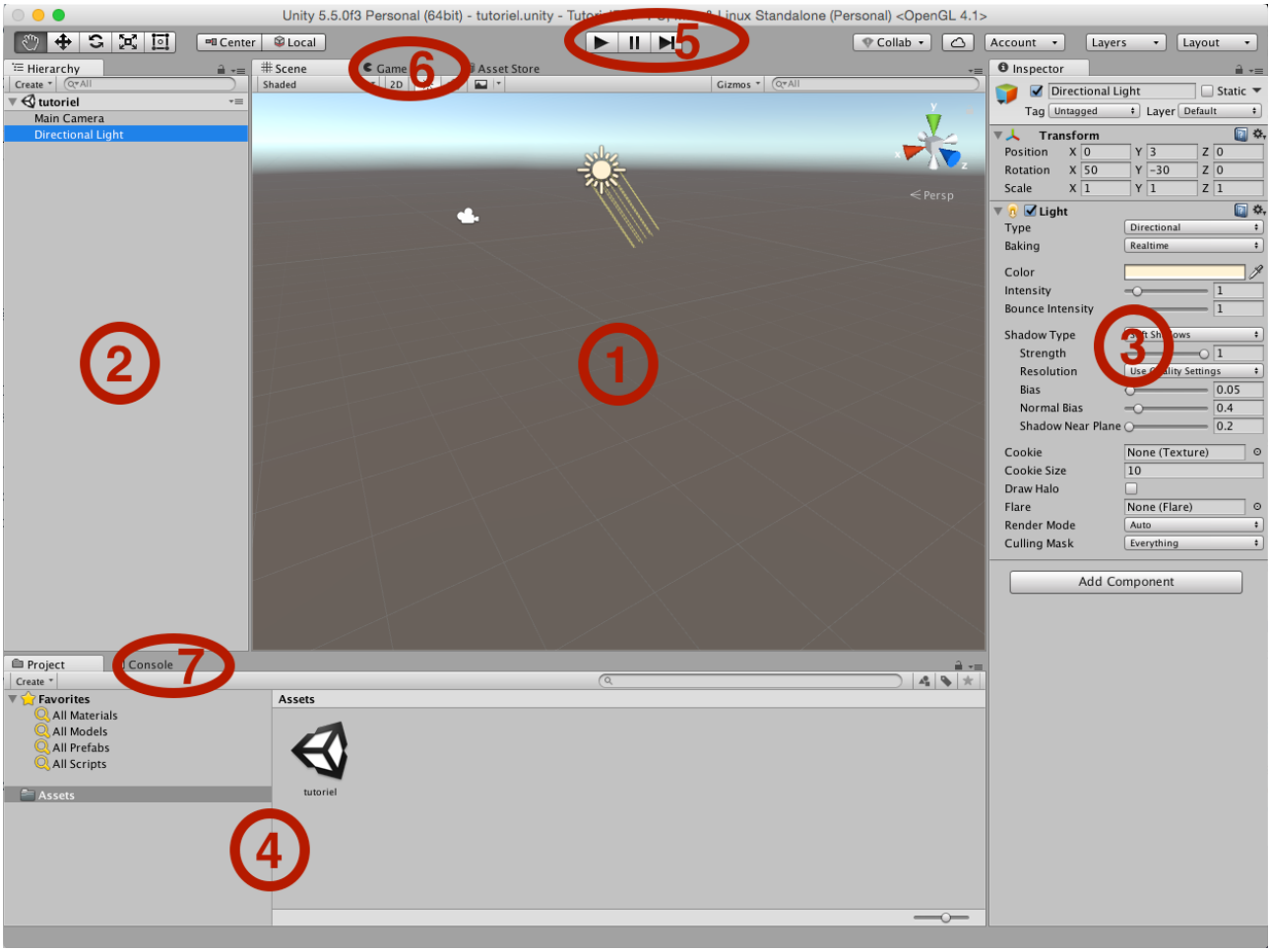
Sauvegarder votre travail : `projet` et `scenes`.

Unity 3D sauvegarde de façons différentes les informations à propos de votre projet. Cela veut donc dire que vous devez sauvegarde différemment votre travail selon les informations que vous avez modifiées :

- La `scene` correspond à la scène 3D contenant une hiérarchie d'objets (`GameObject`) ainsi que leur paramètres (position, etc.). Lors que vous effectuez une sauvegarde de la `scene`, le `projet` est également automatiquement sauvegardé. Plusieurs `scenes` peuvent être créées dans un même projet.
- Le `projet` permet de stocker les informations qui ne sont pas spécifiques aux `scenes`, comme par exemple les paramètres du projet, les scripts, les packages importés, les paramètres de « `Build` », etc.

2. L'éditeur d'Unity 3D

Lorsque vous créez un projet, vous obtenez par défaut dans la scène une caméra (`Main Camera`) et une lumière pour éclairer la scène 3D (`Directional Light`). La fenêtre de l'éditeur se présente alors de la façon suivante :



Cette fenêtre de l'éditeur est composée des éléments suivants :

1. L'éditeur graphique de la scène 3D permettant de manipuler directement des objets de la scène.
2. L'arborescence de la scène permettant de visualiser la hiérarchie des objets présents dans la scène.
3. L'inspecteur permettant de voir les différents scripts rattachés à l'objet sélectionné (par 1 ou 2) et de modifier leurs paramètres.
4. L'explorateur de projet permettant de voir les différents composants du projet et de les ajouter à la scène par un simple glisser-déposer. Pour l'instant, il y a seulement la scène dans le projet.
5. Les boutons permettant de lancer l'exécution de la scène 3D dans l'éditeur ce qui est très utile pour tester et debugger la scène (nous verrons plus tard comment créer un exécutable pour la scène).
6. La visualisation de la scène 3D à partir du point de vue de la caméra virtuelle. L'éditeur bascule automatiquement sur cette visualisation lorsque l'exécution de la scène est lancée.
7. La console permettant de voir les messages d'erreurs ou les logs lors de l'exécution de la scène.

Les interactions dans l'éditeur graphique peuvent se faire à partir des boutons suivants :

Voir https://docs.unity3d.com/Manual/SceneViewNavigation.html et https://docs.unity3d.com/Manual/PositioningGameObjects.html pour plus d'information.

3. Créer une scène simple

Pour commencer, nous allons créer une application toute simple qui affiche plusieurs cubes de couleur bleue à des positions différentes :

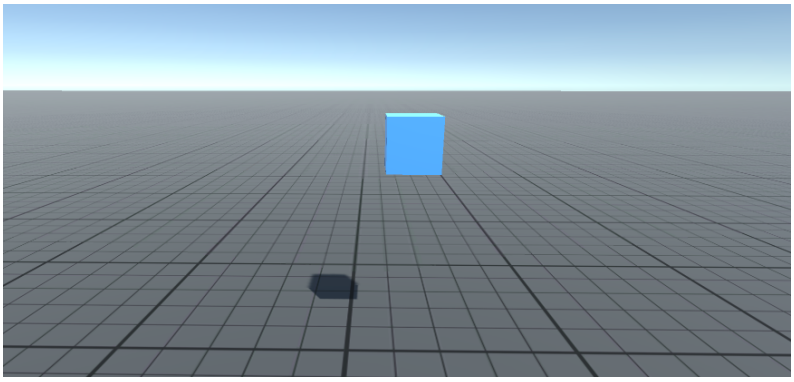
- Insérer un plan dans la scène pour créer un sol (`GameObject > 3D Object > Plane`).
  Au moyen de l'inspecteur (3), placer ce plan aux cordonnées (0, 0, 0) et vérifier qu'il est bien visible par la caméra maintenant. Agrandir ce plan en jouant sur l'échelle pour qu'il est une longueur et une largeur de 100 mètres et vérifier qu'il est bien visible à l'infini lorsque l'on prend la vue de la caméra.

- Ajouter un cube dans la scène (`GameObject > 3D Object > Cube`) et placer le pour qu'il soit visible par la caméra.

- Créer un `material` pour donner une couleur au cube (`Assets > Create > Material`). Un nouveau composant est créé dans le projet (4), donner lui un nom. Puis modifier sa couleur dans l'inspecteur pour obtenir le bleu de l'image suivante. Ensuite glisser-déposer le `material` à partir de l'explorateur de projet (4) sur le cube dans l'arborescence de la scène (2). Le cube devrait prendre la couleur bleue dans l'éditeur graphique (1).

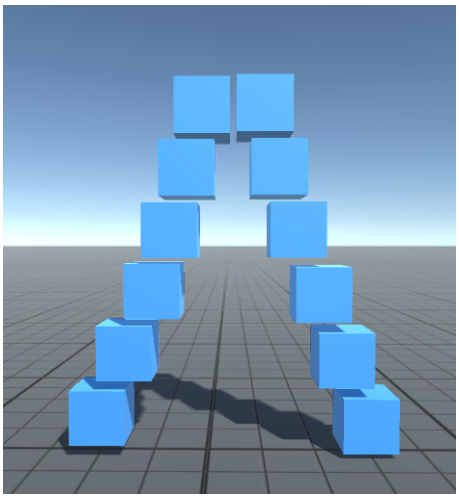
- Ajouter une texture sur le plan : télécharger la texture *grid_grey.png* sur le site du cours, aller dans (`Assets > Import New Asset…`) et choisir la texture. Glisser-déposer la texture à partir de l'explorateur de projet (4) sur le plan dans l'arborescence de la scène (2). Puis sélectionner le plan dans l'arborescence de la scène (2) et modifier les paramètres du `material` dans l'inspecteur (3) afin d'obtenir la même texture du sol que dans l'image ci-dessous. Il faudra en particulier jouer sur le nombre de fois que la texture est répétée (« *tiling* »).

- Déplacer le cube et la caméra afin d'obtenir une image similaire lorsque vous exécutez la scène :



- Ajouter un comportement physique au cube : sélectionner le cube et lui ajouter un `Rigidbody` (`Component > Physics > Rigidbody`). Observer ce qui se passe lorsque vous exécutez la scène.

- Glisser-déposer le cube depuis l'arborescence de la scène (2) vers l'explorateur de projet (4) : cela va créer un « `Prefab` » à partir du cube (cf. icône bleue) qui vous devrez renommer (`CubePrefab` par exemple). Puis, par l'opération inverse, insérer 11 autres cubes à partir de ce modèle dans la scène et déplacer les instances de ce cube afin d'obtenir la même disposition que dans l'image de droite.



- Exécuter la scène.

- Sélectionner le `CubePrefab` dans l'explorateur de projet (4) et modifier les paramètres de son `Rigidbody` afin de changer son comportement physique (en particulier, la masse « `Mass` » et la friction « `Drag` »). Exécuter la scène avec plusieurs paramètres

différents. Qu'observez-vous ?

4. Animer un objet de la scène

Des scripts en C# ou Javascript permettent de donner un comportement aux objects. Nous préférons les scripts en C# car ce langage est plus proche du Java que vous connaissez probablement mieux (typage, etc.). Nous allons donc créer un nouveau script C# permettant de faire tourner les cubes que nous avons crées :

- Créer un nouveau script (`Assets > Create > C# Script`) que vous nommerez `RotateCube`.
- L'éditer en double cliquant dessus (par défaut, il sera ouvert avec `Mono`).
- La fonction `Update()`, qui est héritée de la classe `MonoBehaviour`, est appelée à chaque pas de temps lors de l'exécution de la scène. Dans cette fonction `Update()`, effectuer une rotation de 3° selon l'axe vertical (Y). Pour cela, vous pourrez accéder à la position de l'objet auquel le script sera associé au travers de l'attribut `transform` (déjà défini dans la classe mère `MonoBehaviour`). Vous pourrez utiliser l'auto-complétion de `Mono` pour voir les fonctions que l'on peut appliquer à cet attribut `transform` et trouver la fonction qui permette d'effectuer la bonne rotation.
- Ajouter le script au `CubePrefab` par glisser-déposer sur l'icône du prefab dans l'explorateur de projet (4). Noter l'apparition du script en tant que composant dans l'inspecteur (3) : on peut le désactiver ou le supprimer. Noter également que toutes les instances de CubePrefab ont été modifiées par cet ajout.
- Exécuter la scène.
- Pour une plus généricité du script, ajouter un attribut public dans la classe `RotateCube` avant la fonction `Start()` : `public float speed = 3.0f;`
- Modifier le script en conséquence.
- Vous remarquerez l'apparition de cet attribut dans l'inspecteur (3), ce qui permet de modifier la valeur de la vitesse de rotation sans éditer le script à chaque fois. Noter que la valeur `3.0f` est la valeur par défaut lorsque l'on ajoute le script à un nouvel objet.
- Tester avec différentes valeurs.

5. Gestion des interactions

Unity 3D permet de capter les entrées de base comme les appuies sur les touches du clavier ou les actions de la souris. Cependant, afin de permettre une plus grand généricité et de ne pas coder en dur toutes les commandes dans les scripts, Unity 3D propose un système de gestionnaire d'`Input`. Pour plus d'information, voir la documentation accessible à l'adresse suivante :
https://docs.unity3d.com/Manual/ConventionalGameInput.html

Nous voulons donc maintenant modifier le script qui fait tourner les cubes pour que l'on puisse activer et désactiver la rotation. Par défaut, la rotation sera désactivée. Lorsque l'utilisateur appuiera sur la touche « r » (et la relâchera ensuite), les cubes se mettront à tourner. Puis lorsqu'il appuiera à nouveau sur la touche « r » (et la relâchera ensuite), les cubes s'arrêteront de tourner, et ainsi de suite : dès que la touche « r » est appuyée, les cubes changeront leur comportement. Pour cela :

- Aller dans le menu (`Edit > Project Setting > Input`), ajouter un nouvel axe en modifiant la taille `Size`. Changer son nom en « `ActivateRotation` » par exemple et définir le `Positive Button` comme la touche « `r` ».
- Pour récupérer l'événement « `ActivateRotation` » dans votre script, vous pourrez utiliser les fonctions suivantes qui retournent des booléens (cf. documentation avec l'auto-complétion de `Mono`) :

- `Input.GetButton("ActivateRotation")`
- `Input.GetButtonDown("ActivateRotation")`
- `Input.GetButtonUp("ActivateRotation")`

- Pour vérifier le bon fonctionnement de votre script, vous pouvez afficher dans la console de l'éditeur (7) un message chaque fois que la touche « r » est appuyée. Pour cela, vous pourrez utiliser la syntaxe suivante : `Debug.Log ("votre message");`

6. 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`.

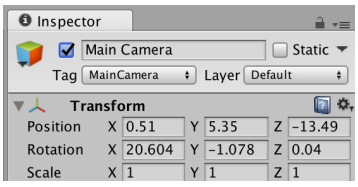
Pour plus d'information sur classe `MonoBehaviour` et ses fonctions, aller voir la documentation : https://docs.unity3d.com/ScriptReference/MonoBehaviour.html

7. 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.

8. 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 :

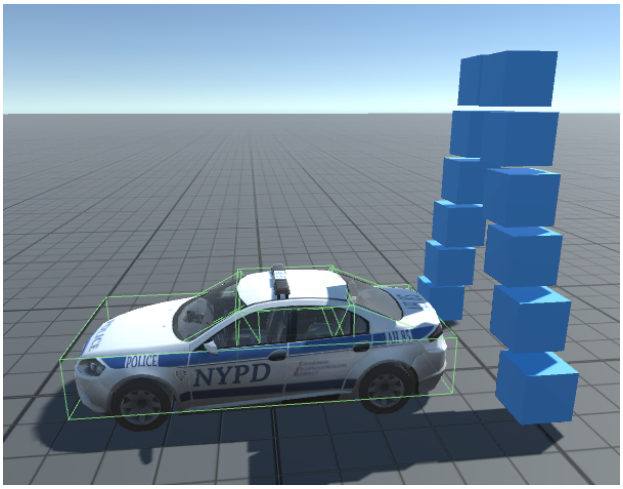
- Créer un nouveau script C# que vous nommerez `CreateOnClick`.
- Ajouter ce script au `FirstPersonCharacter` (l'object qui suit les mouvements de la souris) qui est le fils du `FPSController`. C'est aussi lui qui porte la caméra.
- Créer un attribut `public GameObject aCopier` qui vous permettra de 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.

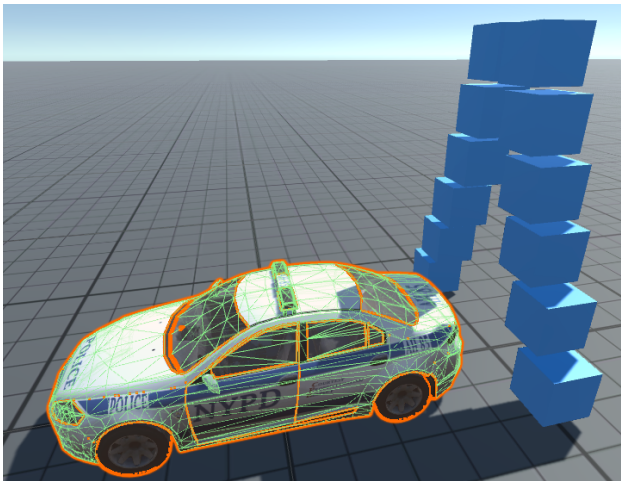
Noter que l'on peut aussi supprimer des objets de la scène grâce à la fonction `Destroy(`*GameObject obj*`)`.

9. 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.





10. 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.).

11. 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.

12. Contact

Cédric FLEURY - cedric.fleury@lri.fr