D-BOOKER éditions

I. VOTRE PREMIER JEU PC

CRÉEZ DES JEUX DE A À Z AVEC UNITY®

(version 5)

Anthony Cardinale

Créez des jeux de A à Z avec Unity

I. Votre premier jeu PC

par

Anthony Cardinale

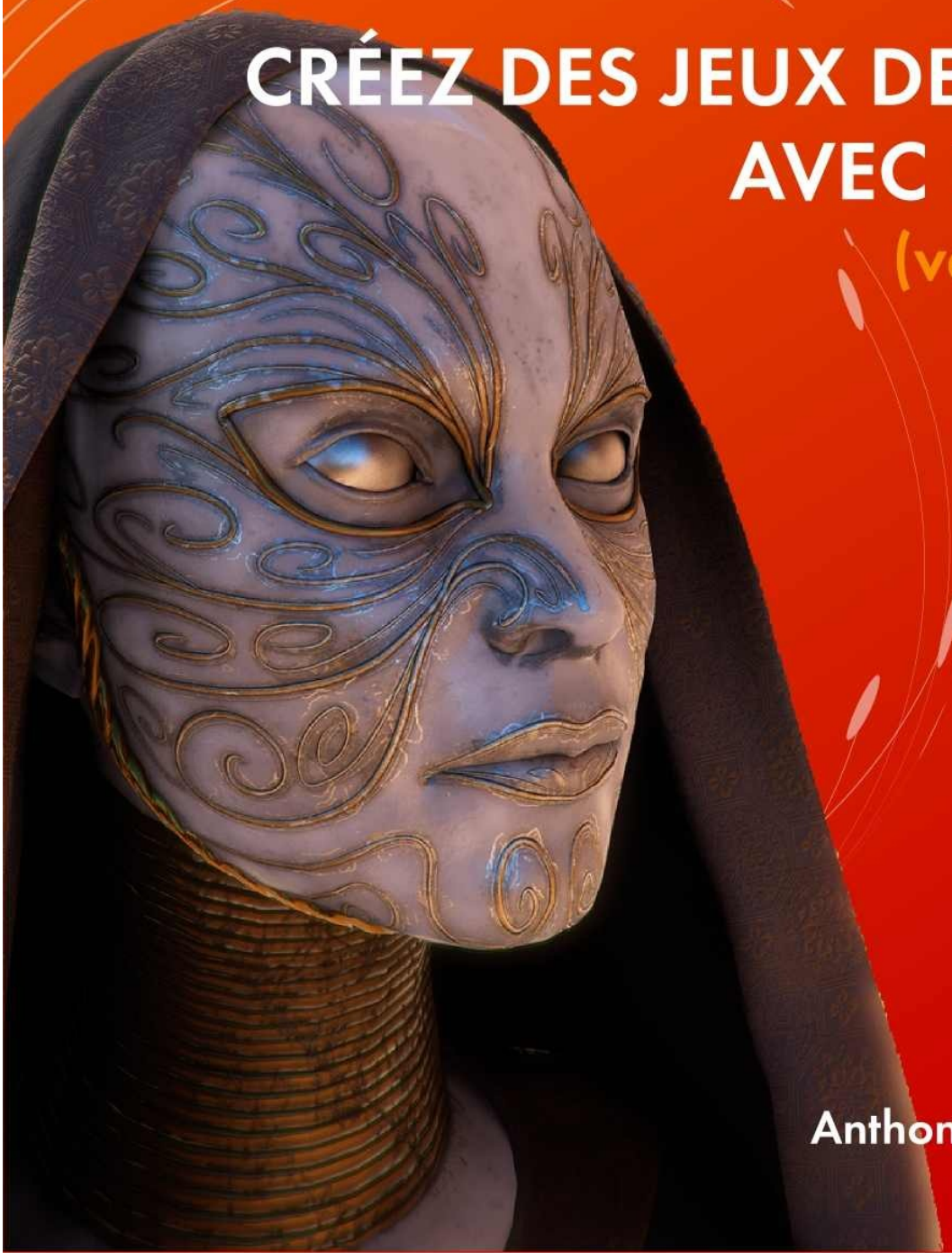
❶❷❸❹❺❻❼❽❾❿⓫⓬⓭⓮⓯

D-BOOKER éditions

I. VOTRE PREMIER JEU PC

CRÉEZ DES JEUX DE A À Z AVEC UNITY®

(version 5)



Anthony Cardinale

Aperçu général de l’ouvrage

Le livre que vous avez sous les yeux constitue un module autonome de l’ouvrage plus global, intitulé Créez des jeux de A à Z avec Unity. Chaque module vise à traiter un aspect de Unity. Le dessin ci-dessus vous donne un aperçu des thèmes traités dans l’un ou l’autre. Les chiffres indiquent le numéro de chapitre correspondant.

Créez des jeux de A à Z avec Unity - I. Votre premier jeu PC

par Anthony Cardinale

ISBN (EPUB) : 978-2-8227-0357-4

Publié par les Éditions D-BookeR, 5 rue Delouvain, 75019 Paris

www.d-booker.fr

contact@d-booker.fr

Ce module fait partie de l'ouvrage global intitulé : Créez des jeux de A à Z avec Unity

ISBN (HTML) : 978-2-8227-0342-0

Mise en page : générée sous Calenco avec des XSLT développées par la société NeoDoc (www.neodoc.biz)

Date de publication : 25/06/2015

Édition : 1

Version : 1

À propos de l’auteur

Anthony Cardinale

Développeur passionné de jeux vidéo, Anthony utilise Unity depuis 2008. À cette date, le logiciel commence tout juste à se faire connaître en France, et il compte parmi les premiers à l’adopter. Auteur de plus d’une trentaine de jeux publiés sur Google Play, l’App Store ou Windows Phone Store, il s’est peu à peu spécialisé dans les applications à destination des mobiles et tablettes, après avoir également développé pour le web et les PC.

En parallèle, Anthony maintient différents sites d’e-learning. Il a réalisé à ce jour plus de 200 heures de formation vidéo sur la création de jeux avec Unity, diffusées via son site internet Formation facile.

Préface

par Mathieu Muller

Ingénieur d'application chez Unity Technologies

Unity, au-delà d'être un outil complet de création de médias interactifs 2D et 3D multiplateformes, de sa communauté de plusieurs centaines de milliers d'utilisateurs ou de son Asset Store, me semble tout à fait remarquable à deux égards. Le premier est sa polyvalence, tant en genres qu'en types d'utilisateurs. Il suffit de consulter la galerie des contenus "made with unity" et de s'intéresser à leurs auteurs pour s'apercevoir d'une part que tous les styles s'y retrouvent, du jeu de plateforme 2D à l'œuvre d'art numérique interactive en passant par le configurateur industriel 3D, d'autre part, que leurs créateurs peuvent être jeunes étudiants ou bien vétérans de l'industrie du jeu vidéo, hommes ou femmes, artistes ou développeurs. Le second constat est encore plus surprenant : il arrive régulièrement qu'un passant interpelle un employé Unity qui porte un t-shirt de l'entreprise pour le remercier. Pour quelle raison ? Car cet outil lui a ouvert de nouvelles perspectives, permit de réaliser son rêve de faire son propre jeu ou bien de rendre ses journées professionnelles ou ses temps libres plus créatifs.

Mais aussi accessible soit-il, tout outil a besoin d'être maîtrisé. Tout comme la guitare ou le piano, les premières notes donneront vite satisfaction, mais seul le travail permettra de créer les plus riches compositions.

Unity propose un manuel complet, des tutoriaux, des webinars hebdomadaires, des forums alimentés par des centaines de milliers d'utilisateurs. Alors pourquoi un livre ? La première raison d'être de cet ouvrage est que l'essentiel de ce matériel n'est disponible qu'en anglais. La seconde et la plus romantique est que chacun a son média préféré d'apprentissage ou bien, pour aller plus loin encore, que le bon média au bon moment peut changer un bout de sa vie. Un livre, même technique, est bien plus qu'un manuel utilisateur, un tutoriel, un webinar ou un forum. C'est une histoire, un compagnon que l'on emporte dans le train ou sur un banc, qui nous remplit d'idées et d'émotions et les connecte à notre environnement lorsqu'on lève la tête du bouquin.

L'histoire c'est Anthony Cardinale qui la raconte. Anthony a consacré des milliers d'heures de sa vie à faire partager sa passion pour le jeu. Tournez cette page et suivez-le. Bienvenue dans ce nouveau monde, et bon voyage à la découverte et la maîtrise de l'infini des possibles des médias interactifs.

Paris, le 07 juin 2015

Avant-propos

Développer votre propre jeu est peut-être pour vous un rêve d'enfant. Il y a encore quelques années, créer un jeu était très compliqué voire presque impossible pour le grand public. Il fallait beaucoup d'argent et avoir des connaissances très poussées en programmation.

Unity a rendu le développement de jeux accessible au grand public. En effet, ce logiciel est un outil tout-en-un qui facilite grandement la conception de jeux et qui est très simple à prendre en main. Simple ne veut pas dire de mauvaise qualité, au contraire, Unity est utilisé par les plus grands studios (Disney, Ubisoft, Gameloft, EA, Blizzard…). Il a littéralement mis à mal la concurrence en devenant gratuit et en proposant un outil qui permet de créer un jeu exportable vers toutes les plateformes (PC, MAC, consoles, mobiles, tablettes, etc.). Adapté aussi bien aux gros studios qu'aux indépendants, ce logiciel est aujourd'hui incontournable lorsque l'on parle de création de jeux.

Dans ce livre, j'ai fait en sorte de couvrir une bonne partie de toutes les techniques de développement dont vous aurez besoin pour mener à bien vos projets. De la conception des niveaux à la programmation en passant par l'optimisation vous ferez un tour complet des fonctionnalités et techniques indispensables au développement de jeux avec Unity et le langage C#.

1. Qu’allez-vous apprendre dans ce livre ?

Ce livre s’adresse à ceux qui souhaitent découvrir la création de jeux avec un outil moderne et puissant. Il s’adresse aussi à tous les développeurs souhaitant migrer vers Unity. Nous allons développer un jeu d’aventure 3D à la première personne, ce qui va nous permettre de comprendre les concepts de la production de jeux 3D ainsi que la programmation avec le langage C#. Voici ce que nous allons voir au cours des onze chapitres de ce livre :

Chapitre 1 - L’environnement de développement

Avant de créer un jeu, il faut découvrir et prendre en main l’environnement de développement que nous allons utiliser. Dans ce chapitre, nous ferons un tour complet de l’interface de Unity3D.

Chapitre 2 - Les assets

Les assets sont les ressources de jeu (textures, modèles 3D, sons, scripts etc.). Dans ce chapitre, nous allons voir comment se procurer des ressources indispensables à la création de nos jeux 3D.

Chapitre 3 - Première scène 3D

Dans ce chapitre, nous allons apprendre à concevoir un monde 3D qui sera le décor de notre jeu vidéo.

Chapitre 4 - Les éléments préfabriqués

Les éléments préfabriqués (prefabs) sont des objets (par exemple un personnage) que vous pouvez créer afin de pouvoir les réutiliser par la suite.

Chapitre 5 - Premiers scripts C#

Dans ce chapitre, nous allons découvrir les bases de la programmation C# et de la création de scripts afin de rendre nos scènes interactives.

Chapitre 6 - Interagir avec l’environnement

Maintenant que nous savons créer des scripts, nous allons apprendre à créer des interactions (ramassage d’objets, augmentation du score, gestion des niveaux, etc.).

Chapitre 7 - Créer des effets spéciaux

Il est souvent nécessaire de créer des objets 3D pendant que le joueur est en train de jouer (apparition d’ennemis, apparition de coffre, etc.). Nous allons donc voir comment instancier des objets en jeu. On y apprendra également à ajouter du son.

Chapitre 8 - L’intelligence articifielle

L’intelligence artificielle (ou IA) est importante pour un jeu. Nous allons donc voir ensemble comment créer un script d’intelligence artificielle permettant à un personnage non joueur de nous suivre.

Chapitre 9 - L’interface utilisateur

Nous allons concevoir le menu principal de notre jeu afin que l’utilisateur puisse lancer votre application en cliquant sur le bouton Jouer.

Chapitre 10 - Optimiser son jeu

Avant de publier un jeu, il faut l’optimiser afin qu’il soit fluide et compatible avec tous les ordinateurs. C’est ce que nous allons apprendre à faire dans ce chapitre.

Chapitre 11 - Partager son jeu

Une fois le projet terminé, nous allons voir comment le compiler puis partager le jeu terminé afin de permettre aux joueurs d’y jouer.

2. Ce dont vous avez besoin

Pour suivre ce livre, vous avez besoin de télécharger et d'installer le logiciel Unity (http://unity3d.com) sur votre machine. Vous devez également télécharger les sources des exemples depuis mon site ou celui de l'éditeur. Pour toute remarque, suggestion, question, vous pouvez me contacter via mon site web http://anthony-cardinale.fr.

Il ne me reste plus qu'à vous souhaiter une bonne lecture !

3. Réglage de la largeur de l’écran

Vous trouverez de nombreux exemples de code, formatés dans une police à chasse fixe. Afin d’éviter des retours à la ligne inopportuns à l’intérieur d’une ligne de code, la longueur maximale des lignes de code a été fixée à 65 caractères, une valeur suffisamment basse pour être affichée sur la plupart des supports, tout en étant suffisante pour que le code puisse être correctement formaté.

Toutefois, il est possible que sur votre support la largeur maximale affichable soit inférieure à la limite fixée. Le paragraphe test ci-dessous permet de vérifier votre affichage. Il doit tenir sur deux lignes exactement :

```
00000000001111111111222222222233333333334444444445555555555566666
01234567890123456789012345678901234567890123456789012345678901234
```

Si ce n’est pas le cas, regardez si vous pouvez agrandir la taille de la fenêtre, diminuer la taille des marges ou diminuer la taille de la police d’affichage. Sur un téléphone portable, placez-le plutôt en mode paysage. Si vous n’y arrivez pas, ne vous inquiétez pas pour autant, la plupart des lignes de code sont inférieures à 65 caractères.

4. URL raccourcies

Dans un souci de lisibilité, et pour pouvoir les maintenir à jour, nous avons pris le parti de remplacer toutes les adresses internet par ce qu’on appelle des URL raccourcies. Une fois que vous avez accédé à la page cible, nous vous invitons à l’enregistrer avec un marque-page si vous souhaitez y revenir fréquemment. Vous disposerez alors du lien direct. Si celui-ci se périme, n’hésitez pas à repasser par l’URL raccourcie. Si cette dernière aussi échoue, vous pouvez nous le signaler !

1

L’environnement de développement

La popularité de Unity n’est pas seulement due à sa puissance mais aussi à son environnement de développement. En effet, Unity est un éditeur de type *WYSIWYG* qui va vous permettre de construire vos scènes 3D visuellement et de les tester à la volée. Nous allons donc découvrir ensemble son interface (qui est en anglais), interface que nous manipulerons ensuite tout au long des chapitres de ce livre.

Elle se décompose en cinq grandes parties, comme vous pouvez le voir sur la Figure 1.1.

Figure 1.1 : *L’interface principale de Unity*

❶ La scène

❷ La fenêtre de jeu

❸ L’inspector

❹ La fenêtre de projet

❺ La hiérarchie

L’interface est modulable, vous pouvez placer les fenêtres où vous le souhaitez et les redimensionner. Il est possible de changer rapidement sa disposition en sélectionnant un autre *layout* à l’aide du bouton layout qui se trouve en haut à droite du logiciel. Ici, j’ai choisi 2 by 3, ce qui me permet d’avoir sous les yeux les cinq fenêtres citées ci-dessus. Cette disposition est selon moi la plus adaptée pour avoir une vue d’ensemble et être plus

productif.

1. La scène

La scène ❶ est la zone de travail où vous allez construire votre monde 3D. C’est là que vous pourrez ajouter, positionner et personnaliser les objets 3D et les ressources. Il s’agit de la partie *level design*.

Lorsque vous aurez à ajouter un objet 3D, une texture, un son, un système de particules ou tout autre élément de jeu, il faudra passer par cette fenêtre scène.

Pour ajouter un objet 3D, glissez/déposez celui-ci de la fenêtre de projet vers la scène. Si vous n’en avez pas sous la main, créez un cube à l’aide du menu GameObject/Create Other/Cube. Il apparaîtra alors dans la scène.

Vous pouvez naviguer dans la scène à l’aide de votre souris. Un clic droit permet de regarder autour de vous, la molette de zoomer/dézoomer, un clic sur la molette de déplacer la vue et la touche Alt enfoncée associée à un clic de changer l’action réalisée par le clic.

2. La fenêtre de jeu

La fenêtre de jeu ❷ vous donne le rendu tel qu'il sera pour le joueur. Vous pouvez tester ici à la volée le jeu que vous avez développé. Une fois celui-ci terminé et compilé, il sera visuellement similaire à ce que vous avez sous les yeux lorsque vous le testez dans cette fenêtre de jeu.

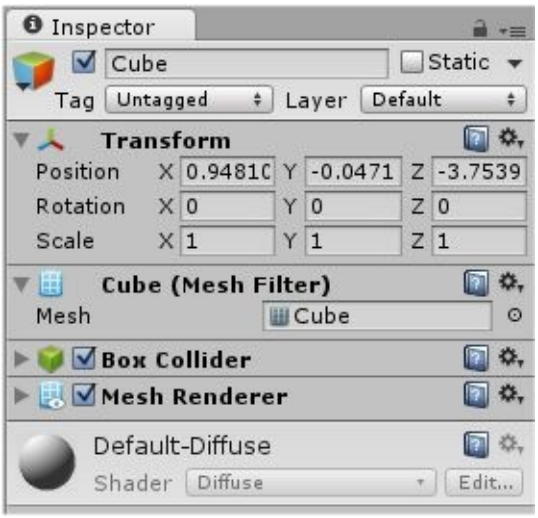
Pour ce faire, cliquez sur la petite icône play  qui se trouve en haut au centre de l'interface.

Le jeu se lance alors dans la fenêtre de jeu et vous pouvez y jouer comme si le produit était terminé. Vous pouvez ainsi tester immédiatement les modifications que vous avez apportées à votre application et détecter d'éventuels bugs. Pour quitter le jeu, vous devez de nouveau cliquer sur le bouton play.

3. L’inspector

L’inspector ❸ est également une zone très importante car c’est dans cette fenêtre que vous avez toutes les informations concernant l’objet sélectionné. Par exemple, si vous sélectionnez un cube créé à partir du menu principal de Unity (Game Object/Create Other/Cube), vous verrez tous les éléments qui le composent.

Figure 1.2 : *L’inspector*



Ici, vous constatez que le cube est composé d’un *Transform* (position, rotation et taille de l’objet), d’un *Mesh* (modèle 3D), d’un *Box Collider* (objet de collision) et d’un *Mesh renderer* (texture de l’objet). Vous pouvez ajouter, supprimer ou modifier des composants directement via l’inspector.

4. La fenêtre de projet

La fenêtre de projet ❹ contient tous les éléments que vous pouvez utiliser pour construire votre jeu. C’est là que vous y rassemblerez les objets 3D, les textures, les scripts, les sons, les vidéos, les particules ou tout autre type de ressources que vous prévoyez d’utiliser pour la création de votre jeu.

Vous pouvez ajouter des ressources à votre projet via la fonction d’importation d’assets ou simplement en faisant un glisser/déposer d’un élément de votre ordinateur vers la fenêtre de projet de Unity.

5. La hiérarchie

La hiérarchie ❺ est la fenêtre qui liste l’ensemble des éléments qui composent votre jeu. Lorsque vous utilisez un modèle 3D, par exemple un personnage, il apparaît dans la hiérarchie. Vous pouvez ainsi avoir sous les yeux la totalité des assets utilisés pour la construction de votre jeu. Cela peut permettre de sélectionner rapidement un objet ou simplement de faire une recherche parmis les assets utilisés.

6. Et après ?

Nous avons vu rapidement les fenêtres composant l'interface de Unity. Nous aurons l'occasion d'y revenir par la suite et de découvrir de nouvelles fenêtres. Il est important de naviguer aussi dans le menu principal du logiciel afin de repérer les fonctionnalités proposées par ce dernier comme par exemple la création de primitives 3D, l'importation de ressources, l'ajout de composants à un objet, etc.

Dans le chapitre suivant, nous allons voir comment se procurer des *assets* (ressources) qui vont nous permettre de concevoir un monde 3D pour notre jeu. Vous verrez qu'Unity nous fournit des ressources de base et qu'il existe des endroits où vous pouvez télécharger énormément de ressources.

Dans ce chapitre, nous avons fait connaissance avec l'interface du logiciel. Nous avons vu cinq parties importantes dont nous allons tout le temps nous servir :

- la scène – création du monde 3D ;
- la fenêtre de jeu – prévisualisation du jeu ;
- l'inspector – informations sur l'objet sélectionné ;
- la fenêtre de projet – les ressources importées ;
- la hiérarchie – les éléments utilisés dans la scène en cours.

2

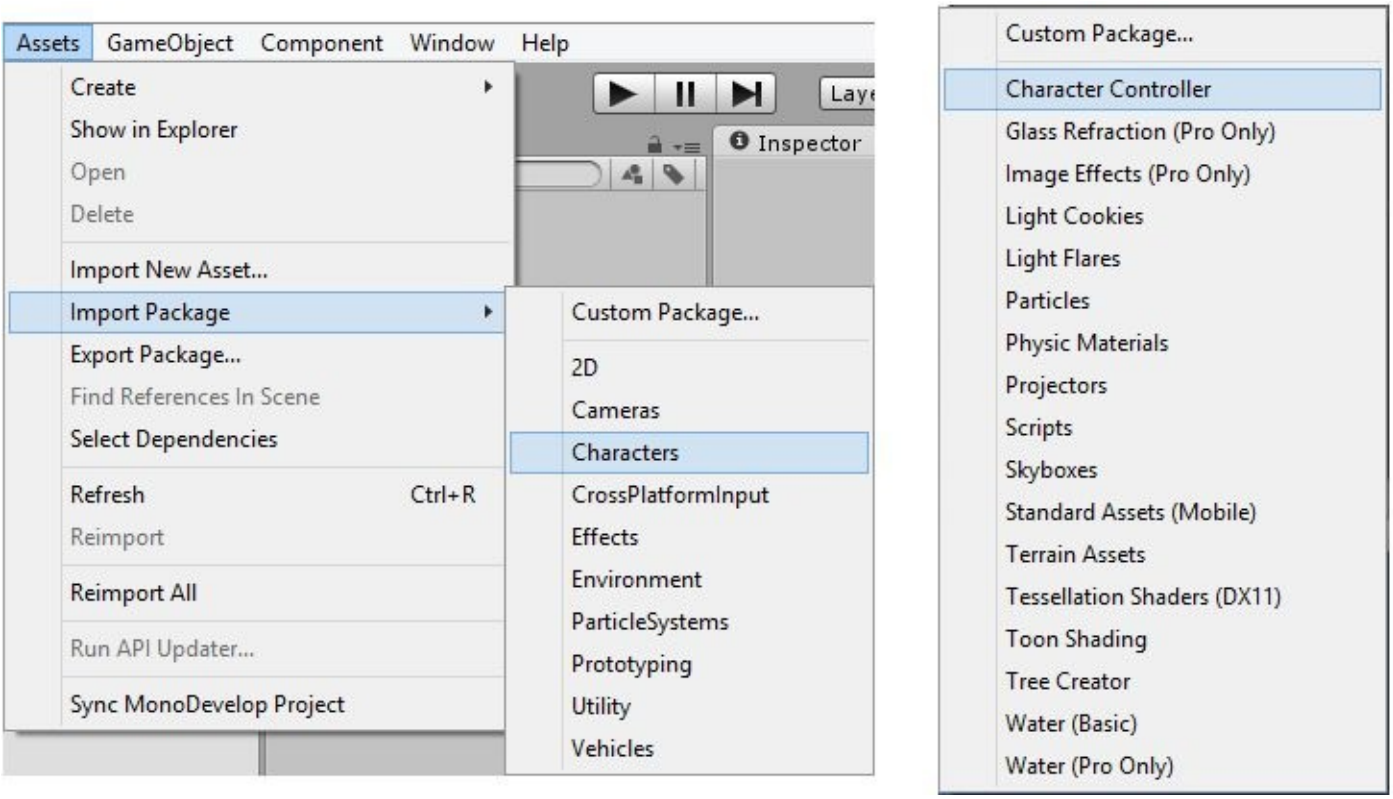
Les assets

On appelle *asset* des ressources graphiques ou des bouts de code prêts à être utilisés pour créer des jeux rapidement. La réalisation d’un jeu requiert beaucoup de ressources graphiques, comme des modèles 3D ou des textures, qu’il peut être difficile à réunir lorsqu’on débute. Les développeurs indépendants, souvent seuls, ont la possibilité de télécharger de nombreuses ressources sur le magasin en ligne de Unity. Grâce à cela, il est possible avec peu de moyens de réaliser des jeux vidéo seul ou en équipe réduite.

1. Les assets par défaut

Les développeurs de Unity ont pensé à nous et ont fourni avec le logiciel des ressources de base gratuites que nous pouvons utiliser pour concevoir nos jeux. Elles sont livrées sous forme de packs thématiques appelés les *standard assets*. Vous pouvez les importer lors de la création d'un nouveau projet ou en passant par le menu Assets/Import package et en choisissant le package de votre choix.

Figure 2.1 : *Importation de packages dans Unity (version 5 à gauche, 4.x à droite)*



Les ressources proposées depuis le logiciel ont été entièrement réorganisées avec l'arrivée de la version 5 en vue d'une optimisation pour les plateformes mobiles. Toutefois vous pouvez très bien continuer à utiliser les anciennes si vous préférez. C'est ce que nous allons faire ici, celles-ci convenant parfaitement au développement de jeux PC. Nous verrons les nouveaux packages dans le second module consacré au développement de jeux mobiles. Cela vous permettra de travailler avec différentes ressources afin de diversifier vos jeux.

***Note** > Afin d'utiliser les mêmes ressources que moi, je vous invite à télécharger un package sur mon site. Cela nous permettra de travailler sur une même base.*

Comme vous pouvez le voir, plusieurs packages sont proposés. Nous ne détaillerons ici que ceux dont nous aurons besoin pour le développement du jeu 3D qui nous servira de fil conducteur. Nous ne considérerons pas les packages *pro* qui ne fonctionnent que sur la version *pro* du logiciel.

- `Character Controller` est très intéressant car il propose deux personnages préconfigurés. Grâce à ce pack, vous pourrez aisément placer un personnage

contrôlable (au clavier et à la souris). Il contient un personnage en vue à la première personne (FPS) et un personnage en vue à la troisième personne.

- `Particles` contient, quant à lui, des systèmes de particules (feu, fumée, étincelles, poussière…) que vous allez pouvoir utiliser pour animer vos jeux et les rendre plus vivants. Vous pouvez par exemple créer un feu de camp pour votre scène 3D.
- `Scripts` contient quelques scripts comme par exemple un script permettant de faire en sorte que la caméra suive le joueur pendant ses déplacements.
- `Skyboxes` sont des boîtes texturées qui permettent de créer un ciel. Nous devons construire notre jeu à l'intérieur de cette boîte pour créer un ciel autour de notre monde.
- `Terrain assets` sont des modèles 3D permettant de créer un terrain 3D. Il s'agit de petites pierres, d'arbres ou d'herbes que nous allons pouvoir positionner sur le terrain pour le rendre plus réaliste.
- Enfin, `Water (Basic)` est un package contenant un modèle 3D représentant de l'eau. Cela va permettre de créer un océan par exemple.

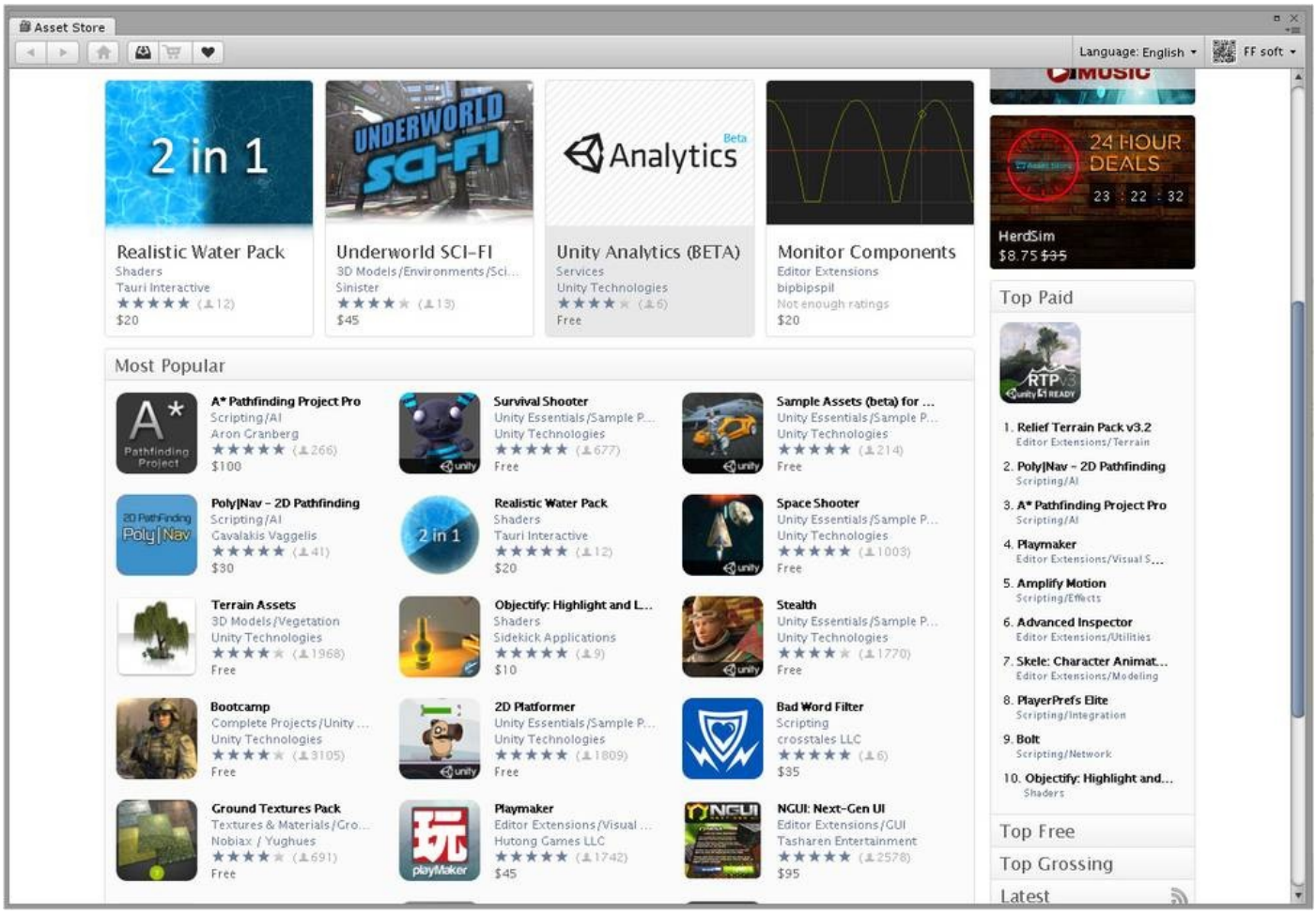
Importez ces ressources, nous allons en avoir besoin pour construire l'exemple de ce livre. Je vous invite également à explorer les autres packages pour vous familiariser avec leur contenu. Ces collections constituent une bonne base pour commencer la conception d'un jeu. Cependant, vous pouvez la compléter par les milliers de ressources disponibles sur l'Asset Store, qui est le magasin en ligne de Unity.

2. L’Asset Store

L’Asset Store est un magasin en ligne comparable à Google Play ou iTune, mais ici vous allez pouvoir télécharger des ressources (modèles 3D, textures, sons, scripts…) pour le développement de jeux. Beaucoup d’entre elles sont payantes mais vous en trouverez aussi de nombreuses gratuites qui vous seront très utiles pour enrichir vos scènes 3D.

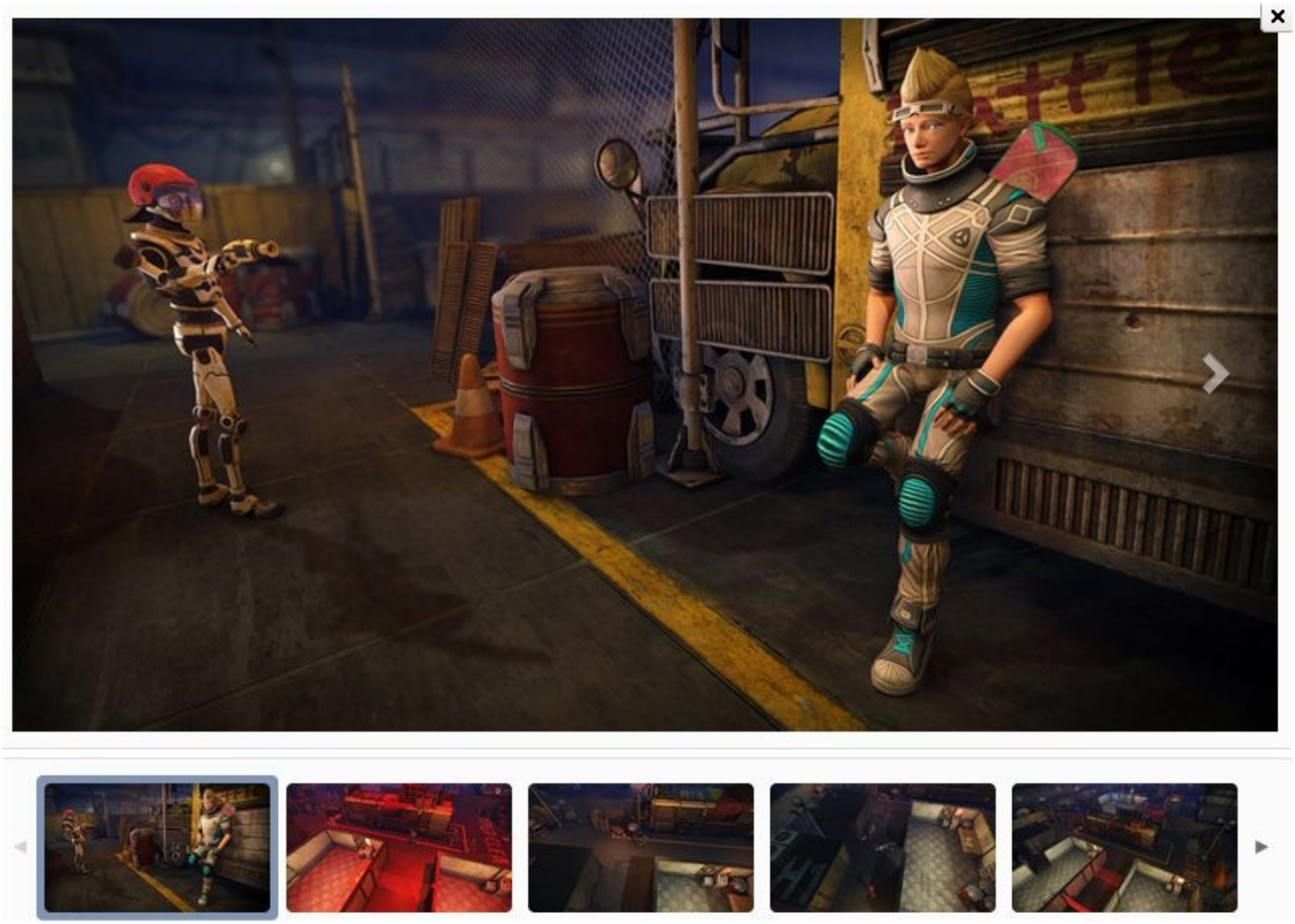
Pour accéder au magasin, sélectionnez le menu Window/Asset store.

Figure 2.2 : *L’Asset Store de Unity*



L’Asset Store est organisé en catégories (3D models, animation, audio, projects, extensions, particles, scripts, shader, textures…), ce qui vous permet d’effectuer des recherches de ressources dans les catégories de votre choix. Par exemple, pour télécharger des modèles d’arbres 3D, il faudra se rendre dans 3D Models, puis dans le sous-menu Environments.

Astuce *> Si vous cherchez des ressources de qualité gratuites, je vous invite à effectuer une recherche sur le mot clé “Unity Technologies”. Vous retrouverez les packages créés par les développeurs du logiciel. Ces ressources sont très nombreuses, gratuites et de grande qualité. Voici un exemple de projet que vous pouvez télécharger et utiliser librement :*



Afin que votre jeu (notre fil rouge) soit unique et entièrement personnalisé, je vous invite à revenir faire un tour sur le magasin quand nous aurons un peu plus avancé. Vous pourrez y télécharger des modèles 3D que vous utiliserez pour décorer votre application.

3. Importation d’assets

La troisième façon d’obtenir des ressources est tout simplement de les importer. Vous pouvez importer des images, des sons, des modèles 3D, des matériaux etc. Vous êtes libre (à condition d’en détenir les droits) d’importer des ressources achetées sur le net ou créées par un membre de votre équipe. Beaucoup de formats sont supportés par Unity, je vous invite à consulter la documentation pour retrouver la liste des extensions prises en compte.

Pour importer des ressources, vous pouvez directement les déposer dans votre projet ou cliquer sur Assets/Import new asset…. De même, vous pouvez exporter des ressources (Clic droit/Export package) via la fenêtre de projet, ainsi vous pourrez les réutiliser dans de futurs projets.

Dans le prochain chapitre, nous allons commencer la conception de notre petit jeu vidéo et nous occuper du level design (création du monde).

Note *> Sachez qu’Unity supporte de très nombreux formats. Google sera donc votre ami, vous aurez la possibilité d’effectuer des recherches afin de trouver des ressources 2D/3D et de les importer dans unity3D. Pensez à toujours vérifier que celles-ci sont libres et que vous pouvez les utiliser pour vos projets commerciaux ou non. Il existe de nombreux sites proposant des ressources gratuites et libres comme par exemple opengameart qui est l’un des plus complets.*

Dans ce chapitre, nous avons vu comment trouver et importer les ressources nécessaires à la création d’un jeu. Il y a trois voies possibles :

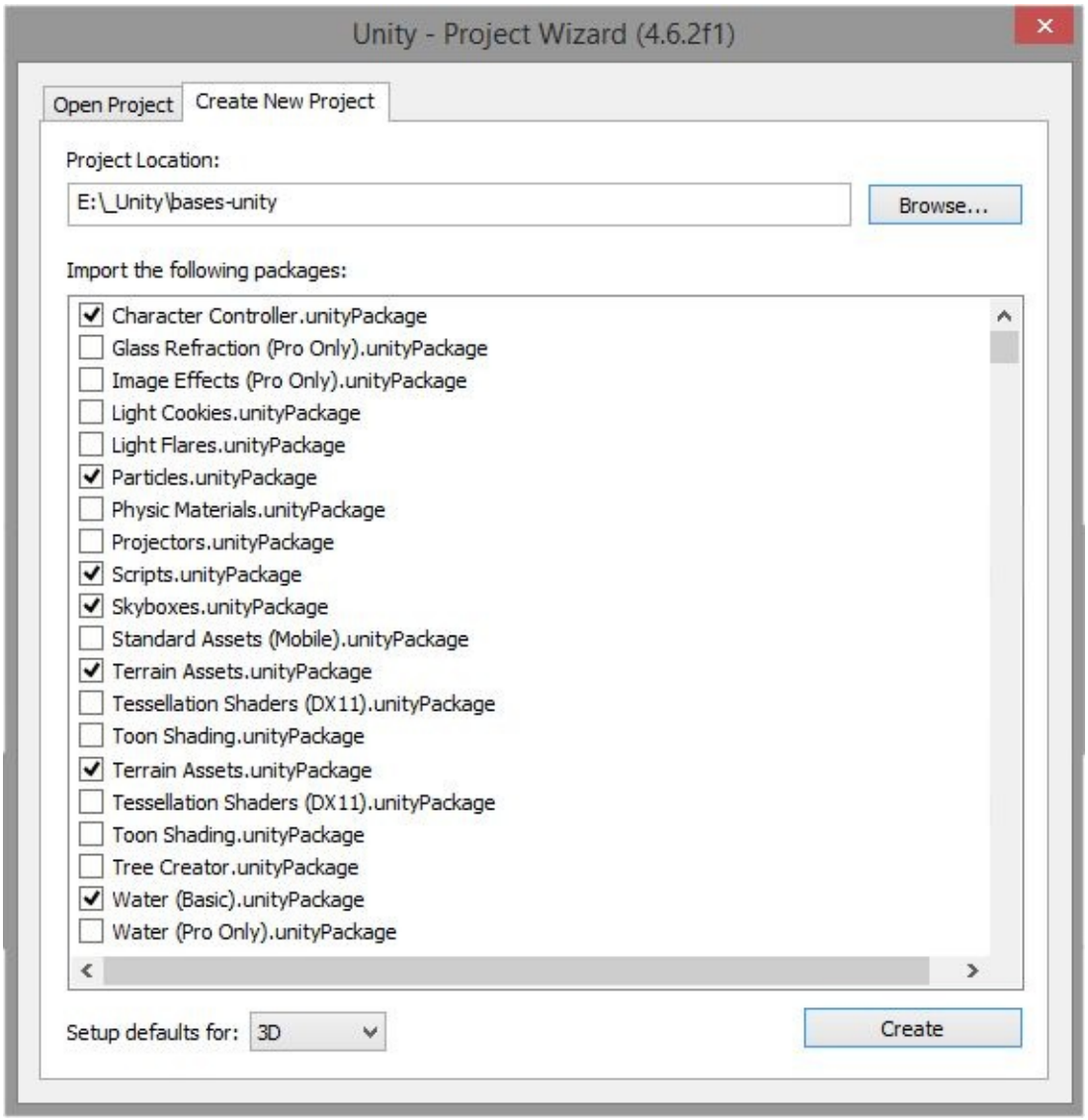
- utiliser les assets de base (*standard assets*) ;
- utiliser l’asset store ;
- utiliser ses propres ressources.

3

Première scène 3D

Afin de partir sur de bonnes bases, commençons par créer un nouveau projet et importer les ressources dont nous allons avoir besoin. Pour créer un nouveau projet, ouvrez le menu File/New project. Donnez un nom à votre projet et sélectionnez les ressources à importer. Je vais importer ici les ressources de base présentées au chapitre précédent (voir Figure 3.1).

Figure 3.1 : *Importation de packages lors de la création d'un nouveau projet*



Vous devez donc avoir un projet vide contenant un dossier avec les standard Assets (voir Figure 3.2).

Figure 3.2 : *Les standard assets*


Project
Create
Standard Assets
Character Controllers
Particles
Scripts
Skyboxes
Terrain Assets
Water (Basic)

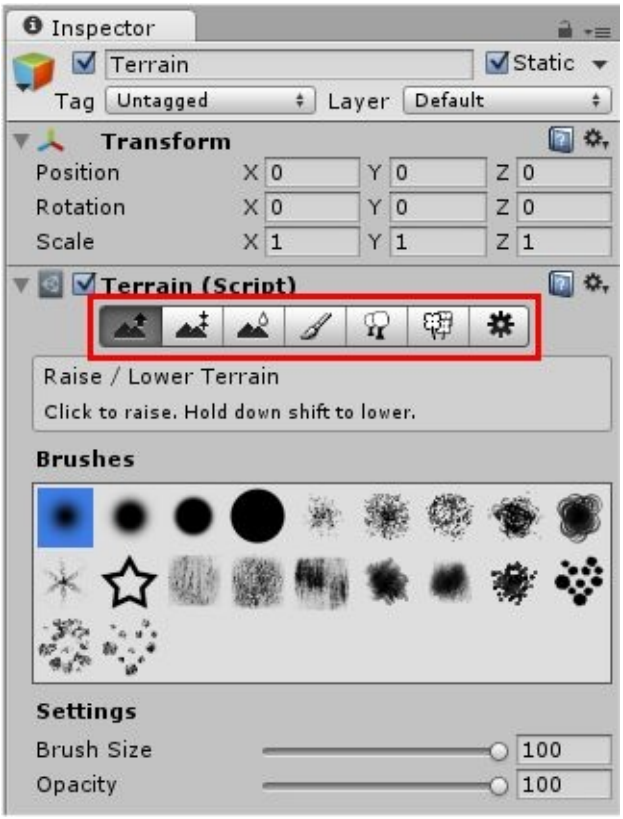
1. Création du terrain

Dans un premier temps, nous allons nous intéresser au terrain. Nous allons modéliser un petit monde 3D où nous pourrons nous déplacer. Pour créer un nouveau terrain, cliquez sur GameObject/3D Object/Terrain. Vous avez maintenant sous les yeux un terrain plat et gris. Nous allons voir dans un instant comment le modifier. Mais avant, afin de mieux distinguer la scène, ajoutons de la lumière en cliquant sur GameObject/Light/Directional light. Cette lumière agira comme un soleil.

1.1. Les outils de modélisation

Pour façonner notre monde, nous disposons d'outils de modélisation du terrain. Dans la fenêtre scène, cliquez sur le terrain afin d'afficher ses propriétés dans l'inspector. Sept outils sont disponibles :

Figure 3.3 : *Les outils de modélisation du terrain*



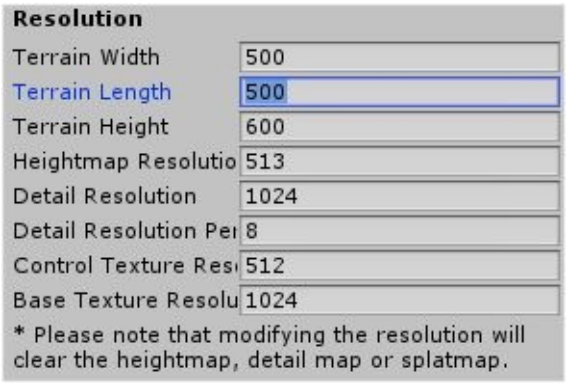
Le premier outil  permet de créer des montagnes en surélevant certaines parties du terrain. Le deuxième  permet aussi de modifier la hauteur du terrain mais en offrant la possibilité de préciser une hauteur manuellement, ainsi nous pouvons par exemple créer une montagne et creuser un trou. Le troisième  permet de lisser le terrain afin de le rendre plus *doux*. Le quatrième  va nous servir à texturer le sol. Nous pourrons ainsi ajouter une texture d'herbe sur l'ensemble de l'objet. Les deux outils suivants   permettent respectivement de placer des arbres, de l'herbe, de la végétation comme des

fleurs sur le sol. Enfin, le dernier bouton permet de modifier des paramètres comme la largeur ou la longueur du terrain.

1.2. Construisons notre monde !

Maintenant que nous connaissons les outils, passons à la pratique et voyons comment modéliser notre monde. Nous allons commencer par diminuer la taille du terrain en cliquant sur la dernière icône et en modifiant les valeurs de résolution :

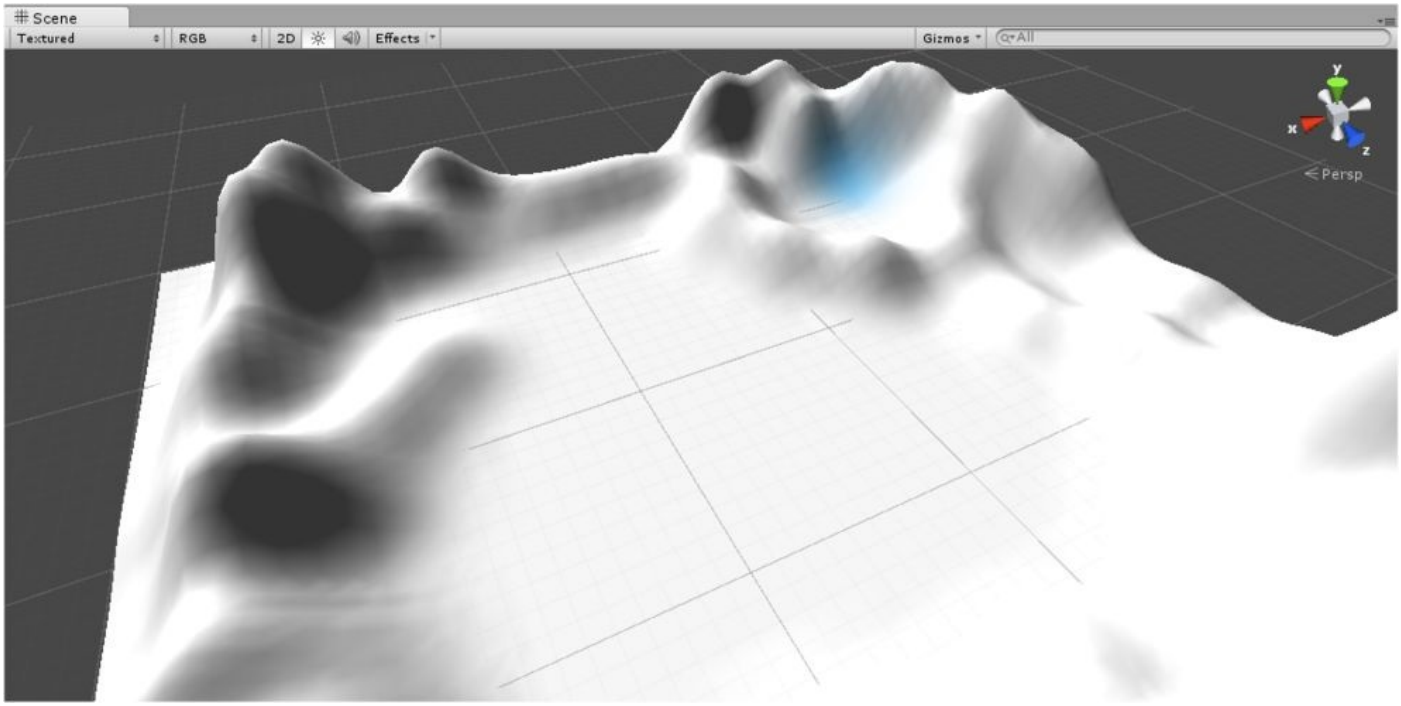
Figure 3.4 : *Modification de la résolution du terrain*



Mettez une valeur de 500 unités pour la largeur (width) et la hauteur (height) de votre terrain. Ceci est largement suffisant pour notre premier niveau. Les unités sont en quelque sorte des mètres pour vous donner un repère. Nous allons maintenant créer des montagnes tout autour de la carte. Pour cela, nous sélectionnons l'outil de hauteur et nous passons avec la souris, en maintenant le bouton enfoncé, sur les zones que nous souhaitons surélever (voir Figure 3.5).

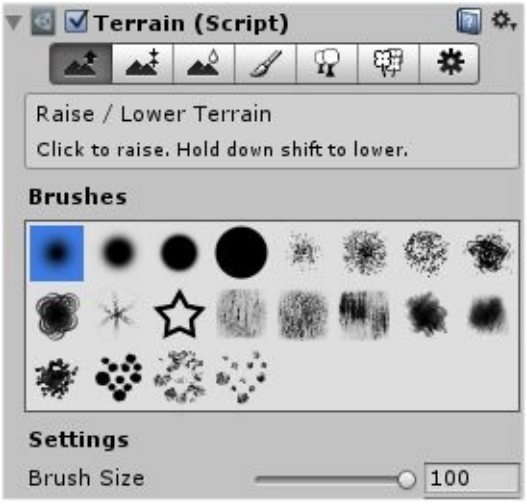
Note *> Afin de limiter les mouvements du joueur, vous devez créer une limite à votre monde. Ici ce seront les montagnes.*

Figure 3.5 : *Création des montagnes*



Vous pouvez utiliser les différents pinceaux et choisir la taille du pinceau dans les paramètres :

Figure 3.6 : *Les paramètres du pinceau*



Les différents pinceaux permettent de peindre avec des motifs variés pour rendre votre terrain irrégulier et donc plus réaliste.

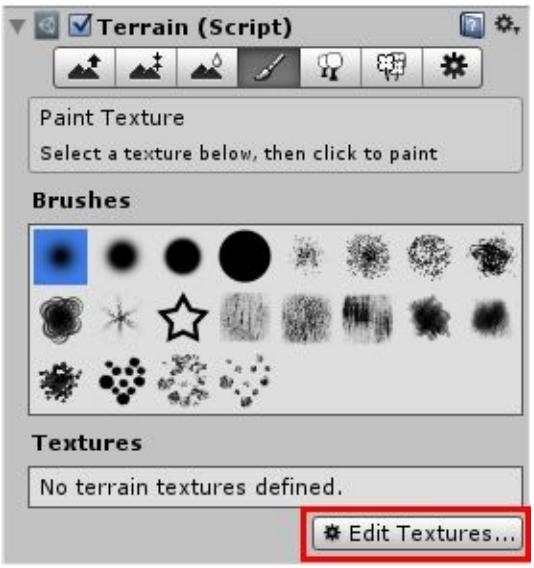
Je vous invite à modéliser votre terrain comme bon vous semble. Créez le décor que vous souhaitez présenter à vos joueurs. La règle à suivre est de bien fermer votre monde afin d’empêcher le joueur de sortir du décor.

Astuce > *Vous pouvez creuser ou créer des trous dans votre terrain en maintenant enfoncée la touche Maj pendant que vous peignez.*

Nous allons maintenant ajouter un peu de couleur à notre terrain grâce aux textures qui se trouvent dans les standard assets que nous avons importés. Vous pouvez utiliser toutes les textures qui se trouvent dans votre projet. Sélectionnez la quatrième icône en forme de

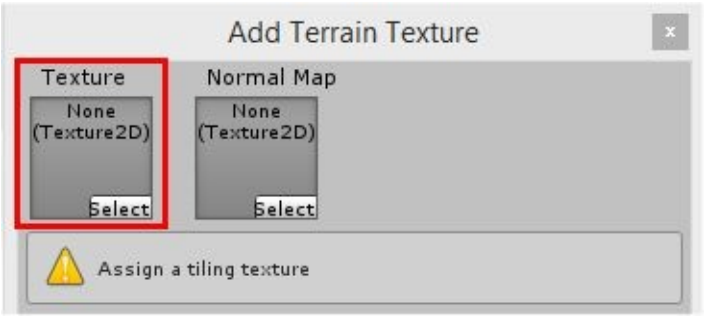
pinceau  et cliquez sur Edit Textures puis sur Add texture pour ajouter une texture.

Figure 3.7 : *Préparer l'ajout d'une texture*



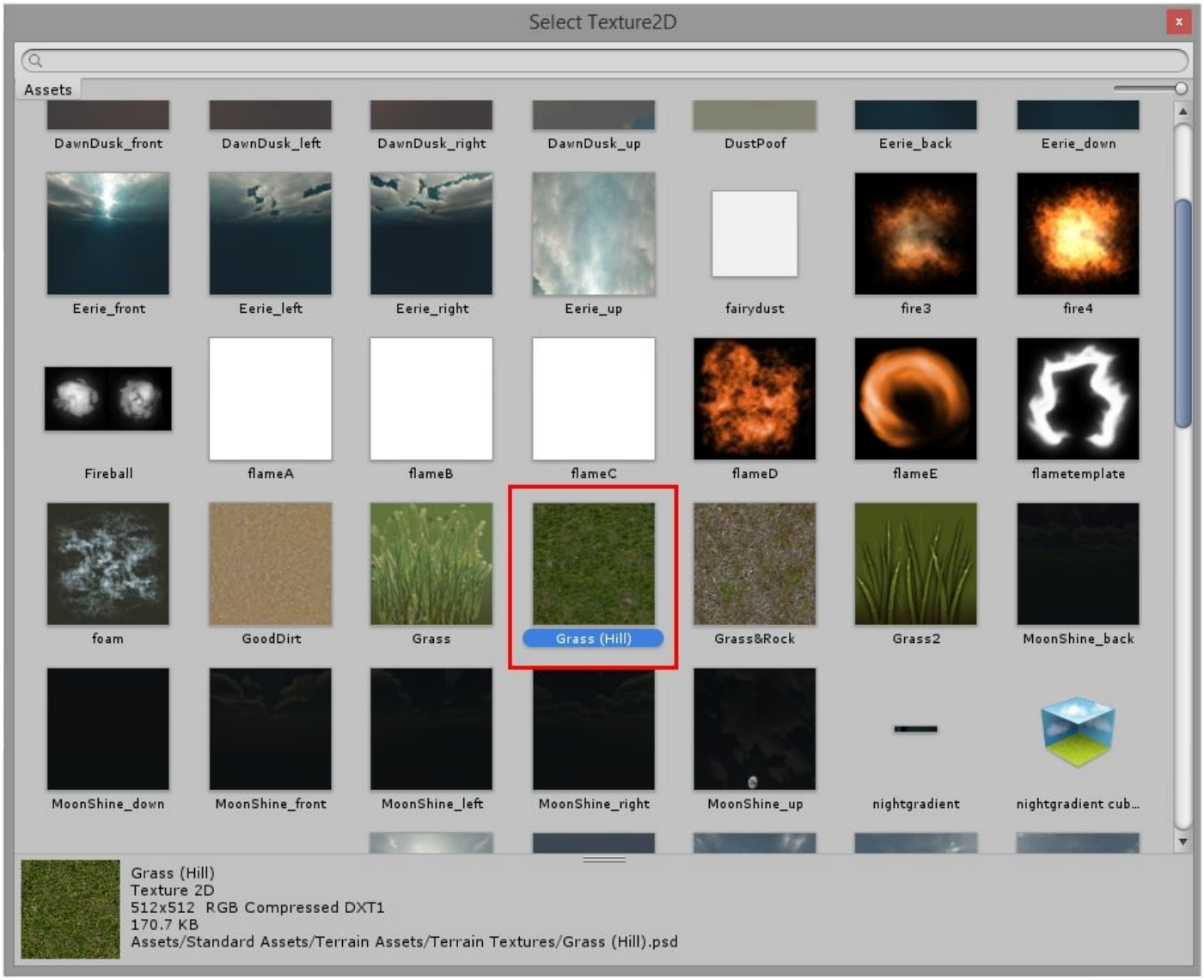
Une nouvelle zone de travail apparaît alors avec deux boîtes vides Texture et Normal Map.

Figure 3.8 : *Ajout d'une texture sur le terrain*



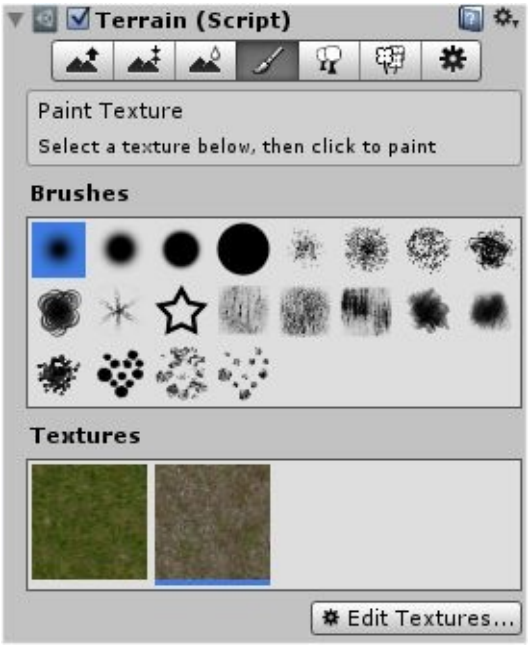
Cliquez sur celle nommée Texture et recherchez une texture d'herbe que vous souhaitez appliquer à votre terrain.

Figure 3.9 : *Sélection de la texture d'herbe*



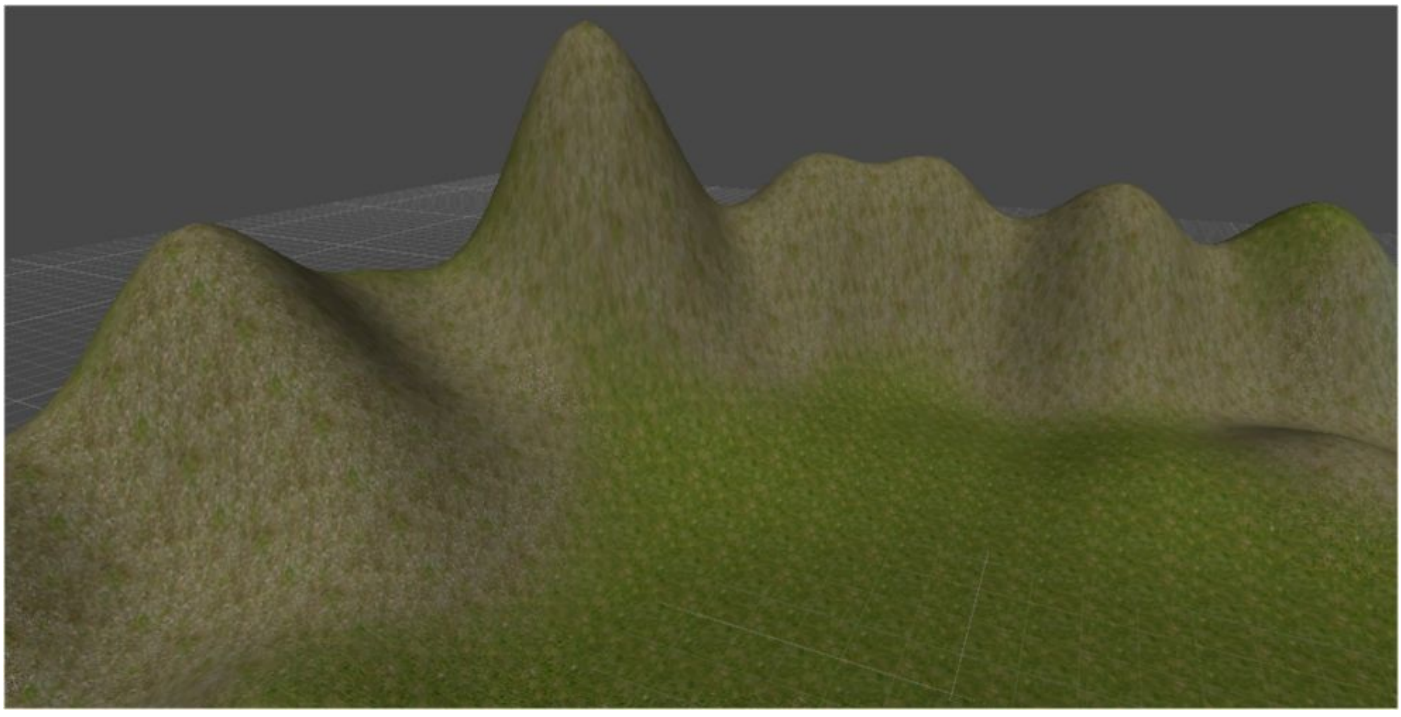
Validez votre choix en cliquant sur Add. Cette première texture s'applique à l'ensemble du terrain. Vous pouvez varier les textures en en appliquant localement à l'aide du pinceau. Nous allons ainsi choisir une autre texture pour peindre les montagnes. Répétez la procédure précédente (Edit textures/Add texture/Sélection d'une texture/Add). Cela ajoute une deuxième texture à votre palette :

Figure 3.10 : *Sélection de la texture de la montagne*



Cette fois, visuellement rien n’a changé. En revanche, si vous sélectionnez cette nouvelle texture, vous pouvez maintenant l’appliquer à l’aide du pinceau. Essayez par exemple de peindre les montagnes pour leur donner un aspect rocheux. La texture va s’appliquer par-dessus l’ancienne.

Figure 3.11 : *Texture des montagnes*

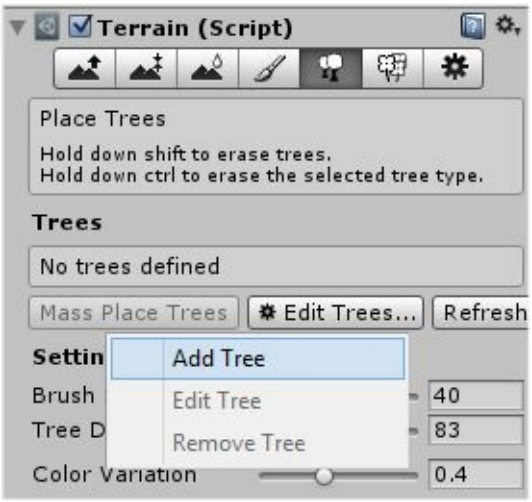


Si vous voulez diminuer l’opacité de cette nouvelle texture (afin de créer un mélange avec la texture d’herbe), diminuez la valeur du paramètre Opacity, par exemple à 50 %.

Comme pour les textures, vous allez ajouter des éléments comme des arbres, des roches ou des fleurs à votre palette que vous pourrez ensuite répandre à l’aide du pinceau sur votre terrain pour le rendre plus réaliste. Sélectionnez l’icône représentant les deux arbres dans l’inspector  et cliquez sur Edit Trees/Add tree (c’est la même procédure que pour

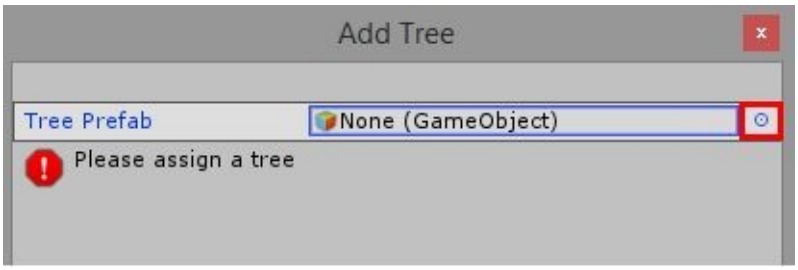
la texture) :

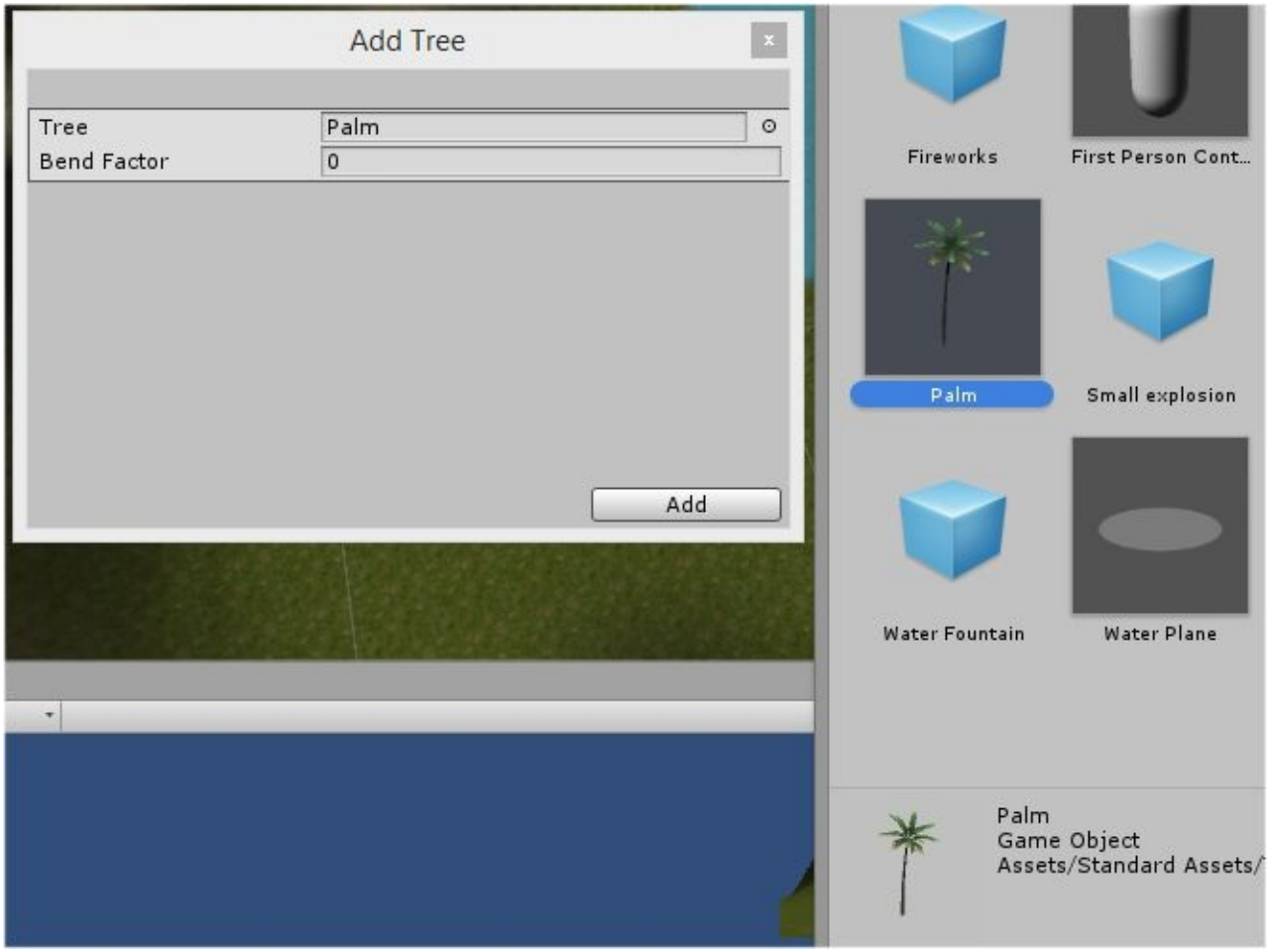
Figure 3.12 : *Ajout d'un arbre à notre palette*



Maintenant cliquez dans la "boîte" Tree afin de rechercher un modèle 3D d'arbre que vous souhaitez appliquer au terrain. Vous trouverez un palmier 3D avec les sources de base :

Figure 3.13 : *Sélection du palmier*

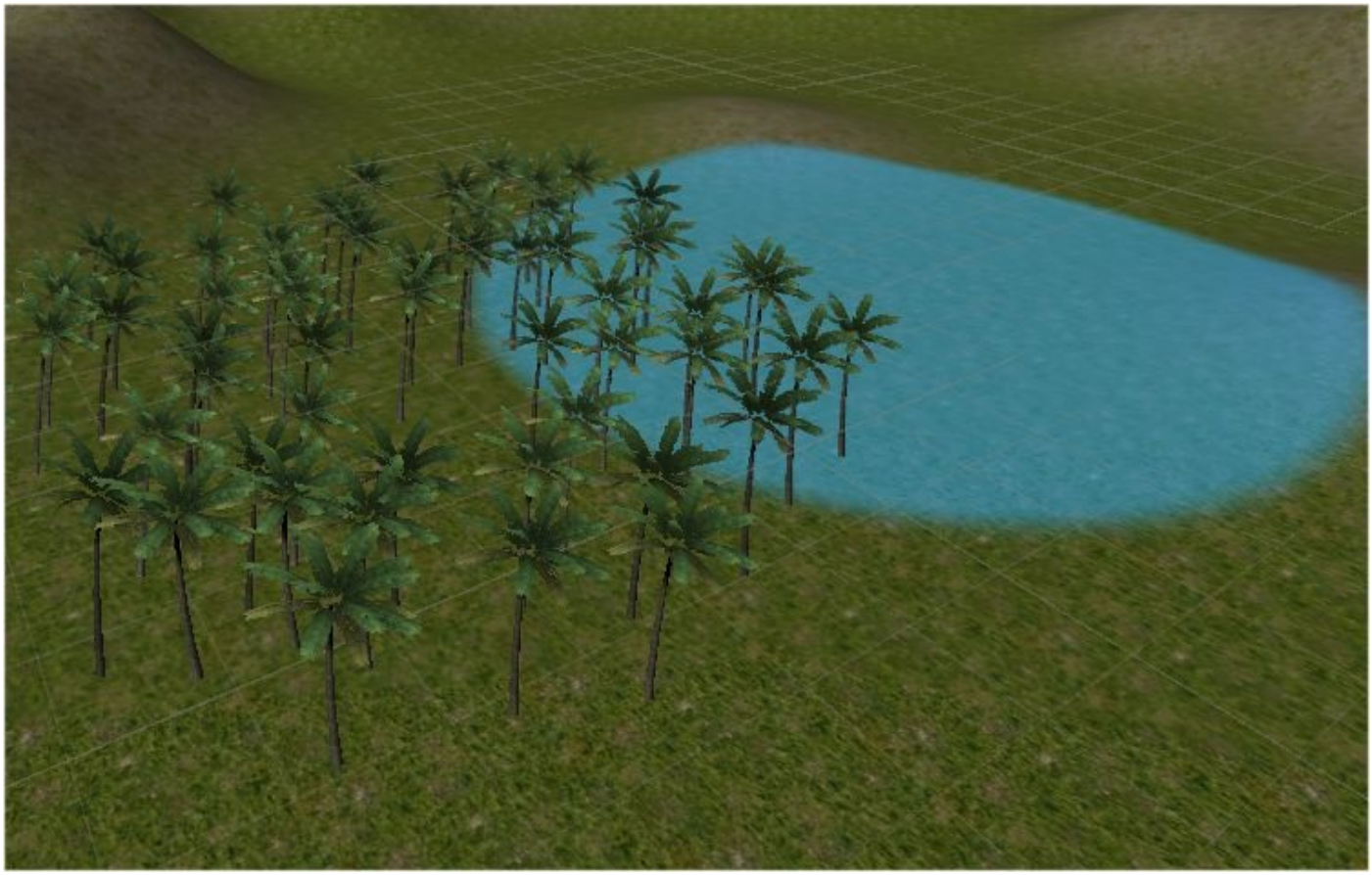




Vous pouvez également ajouter d’autres types d’éléments avec cet outil comme des roches ou d’autres modèles 3D.

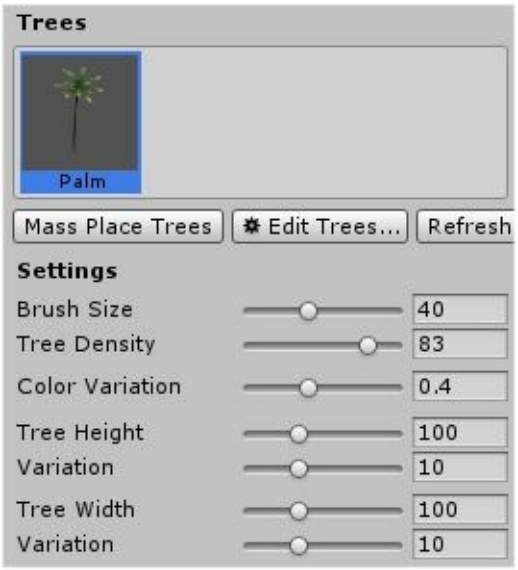
Un clic sur Add ajoute le palmier à votre palette. Vous pouvez créer une forêt en peignant sur le terrain, et les palmiers vont apparaître (voir Figure 3.14).

Figure 3.14 : *Création d’une forêt de palmiers*



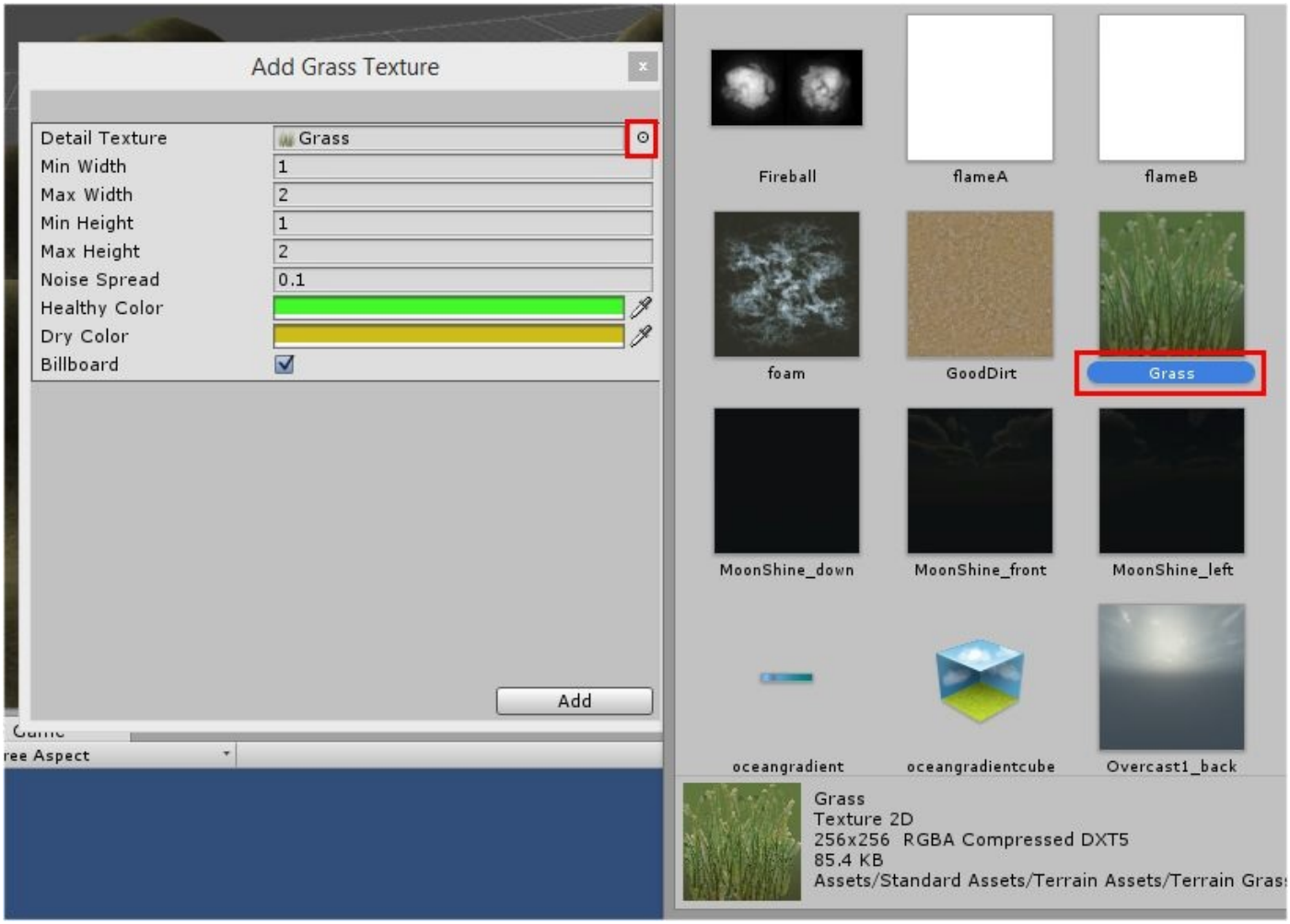
Vous pouvez augmenter ou diminuer la densité d'arbres en passant par la zone de configuration (voir Figure 3.15). Vous pouvez également modifier la taille de votre pinceau et définir un coefficient de variation pour vos arbres (ils n'auront pas la même taille) afin de les rendre plus réalistes :

Figure 3.15 : *Paramétrage des arbres*



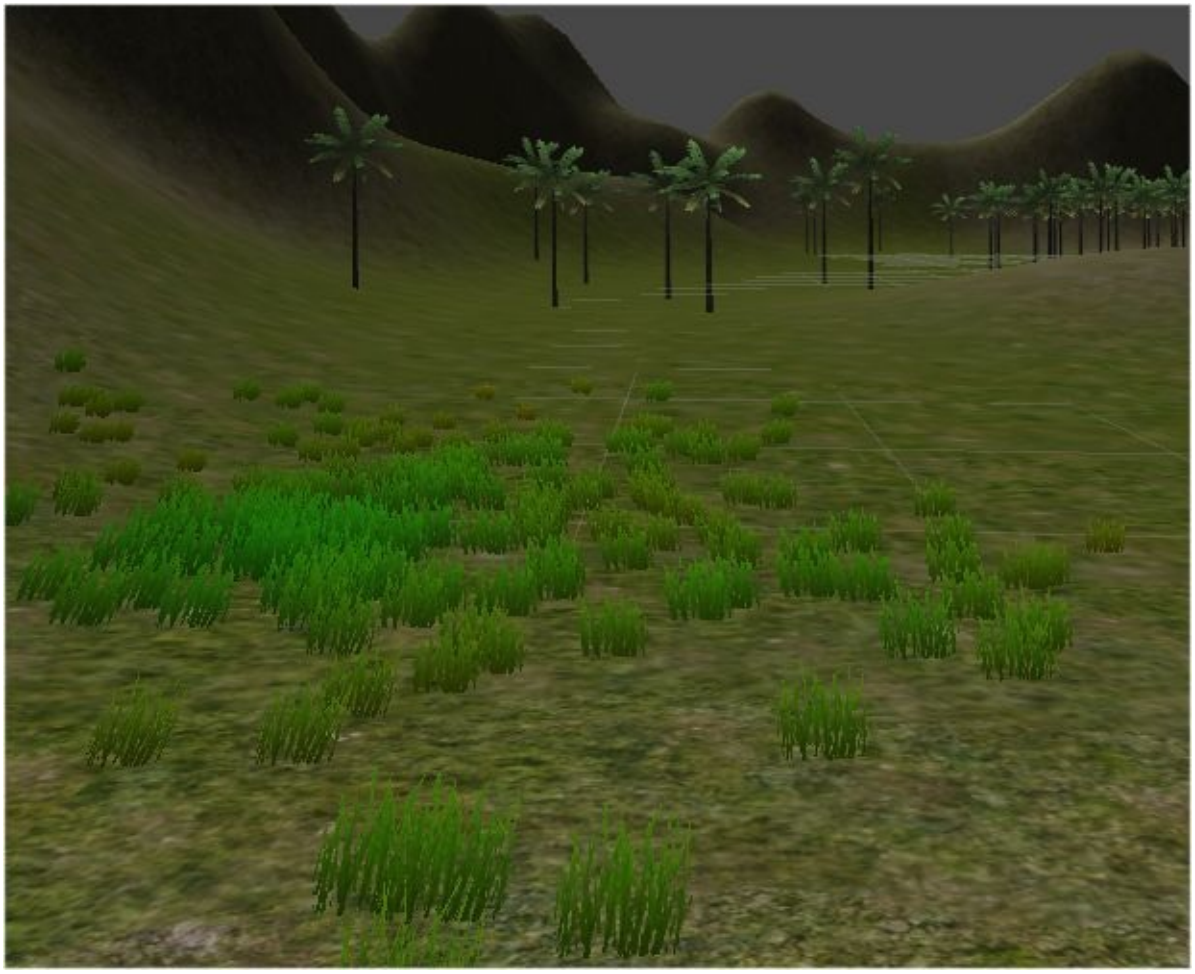
La procédure pour ajouter de l'herbe est presque la même. Sélectionnez l'outil permettant d'ajouter de la végétation , cliquez sur Edit details puis sur Add Grass Texture. Dans Detail Texture, sélectionnez la texture d'herbe que vous voulez ajouter à votre palette :

Figure 3.16 : *Ajout d'une texture d'herbe*



De la même façon que pour les arbres, vous pouvez peindre sur votre terrain pour ajouter ponctuellement de l'herbe.

Figure 3.17 : *Ajouter de l'herbe sur le terrain*



Vous pouvez aller encore plus loin en ajoutant par exemple des roches ou d'autres arbres grâce au package `terrain assets` que vous pouvez retrouver sur l'Asset Store si vous ne l'avez pas déjà importé. Vous devez soigner votre monde et le détailler sans le surcharger.

Encadré : Autre possibilité

Vous pouvez aussi créer un terrain dans un logiciel de modélisation 3D (Cinema4D, 3D studio max, Blender…) et importer votre terrain dans Unity. Cela peut vous permettre d'optimiser le modèle 3D afin de diminuer sa complexité (nombre de polygones).

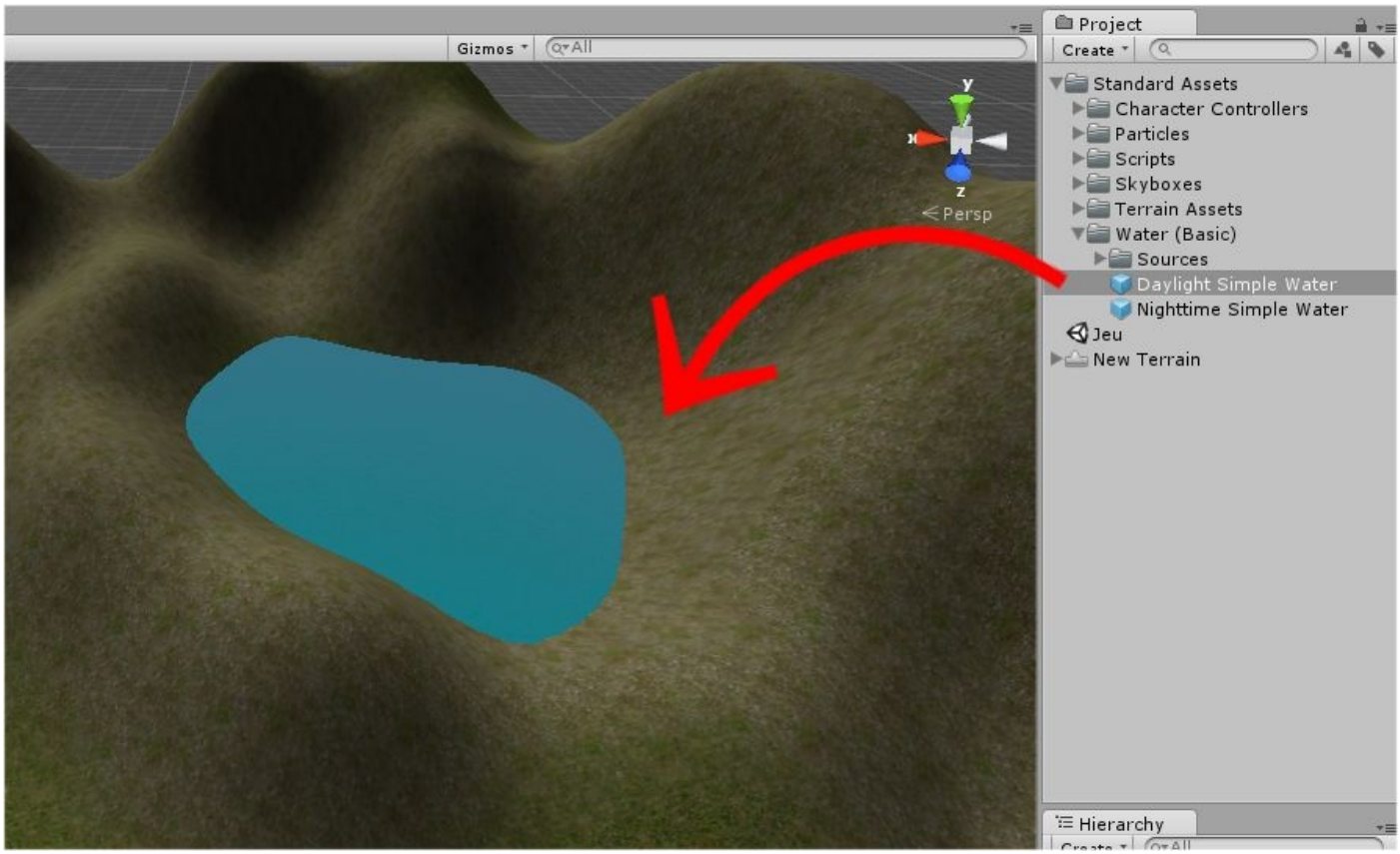
2. Enrichissement du terrain

Nous avons vu comment créer un décor avec l'outil de modélisation de terrain intégré à Unity. Nous allons maintenant améliorer l'ambiance générale de notre jeu en utilisant les ressources se trouvant dans les standard assets.

Nous voulons par exemple créer un lac. Pour cela, nous allons utiliser l'eau (Water) qui se trouve dans le dossier `Standard assets`. Dépliez le sous-dossier `Water (Basic)` et faites glisser le prefab (carré bleu) `Daylight Simple Water` sur votre scène. De l'eau apparaît alors.

Note *> Pour plus d'informations sur ce qu'est un prefab, se reporter au chapitre Les éléments préfabriqués.*

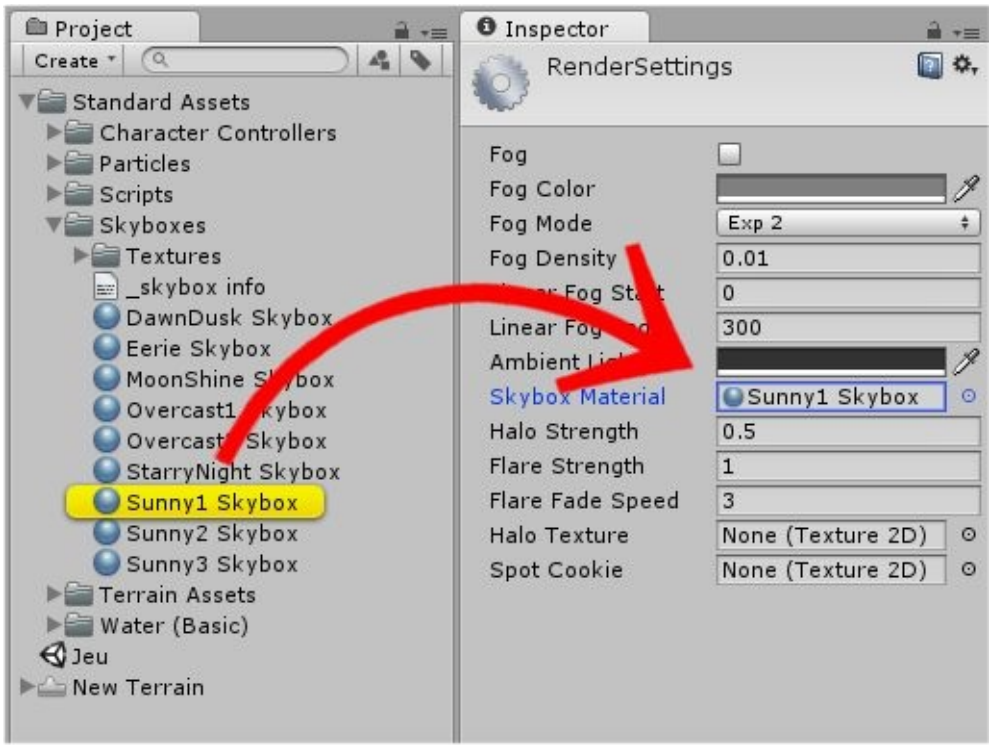
Figure 3.18 : *Création d'un lac*



Comme vous pouvez le constater sur le visuel, vous devez "entourer" le lac de montagnes afin de le délimiter pour ne pas avoir de l'eau qui "flotte" dans l'air. Vous pouvez ajouter de la végétation autour du lac et créer plusieurs points d'eau.

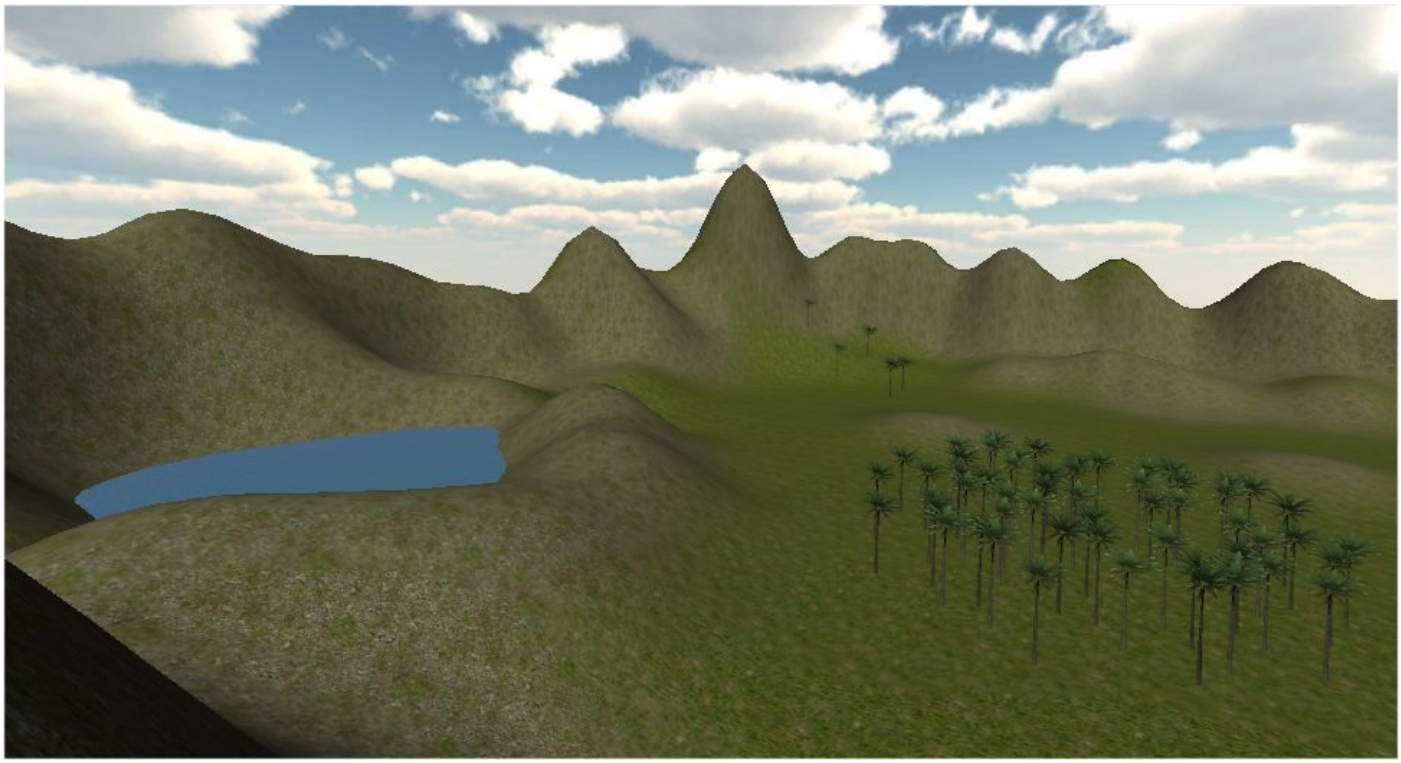
Nous allons maintenant nous occuper du ciel. Afin de créer un ciel réaliste, nous devons passer par le système de *skyboxes* (un ciel sous forme de boîte). Dans les ressources de base, vous trouverez un dossier nommé `Skyboxes` contenant neuf skyboxes différentes. Pour ajouter un ciel à votre projet, rendez-vous dans le menu Edit/Render Settings. Dans le menu qui s'ouvre dans l'inspector, vous trouverez une zone où vous pouvez ajouter un `Skybox material`. Faites un glisser/déposer de la skybox `Sunny1 Skybox` comme sur l'image ci-dessous :

Figure 3.19 : *Création d'un ciel*



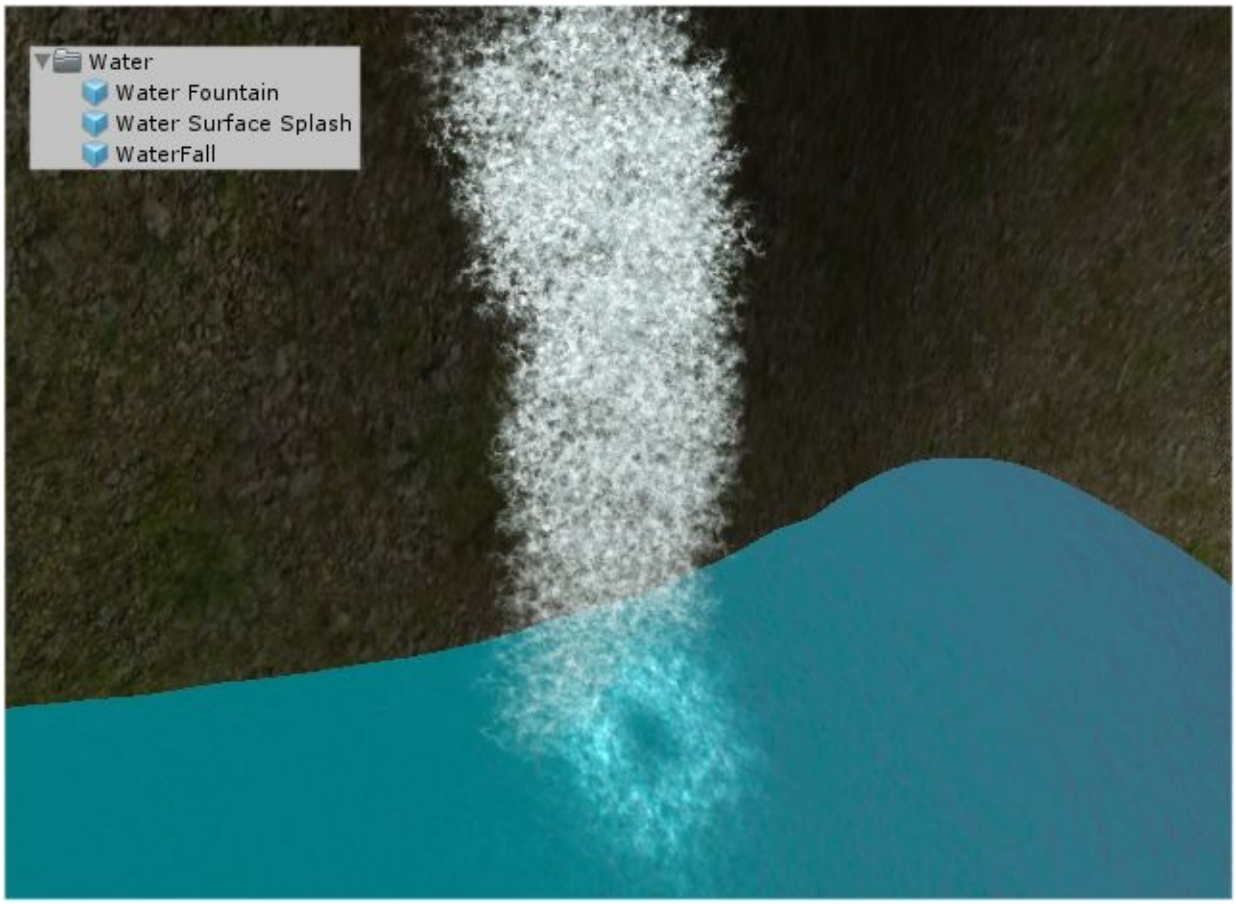
Comme vous pouvez le constater, notre monde 3D commence à prendre forme. Si vous regardez bien le rendu, vous pouvez voir que le ciel a été ajouté :

Figure 3.20 : *Aperçu de la scène 3D*



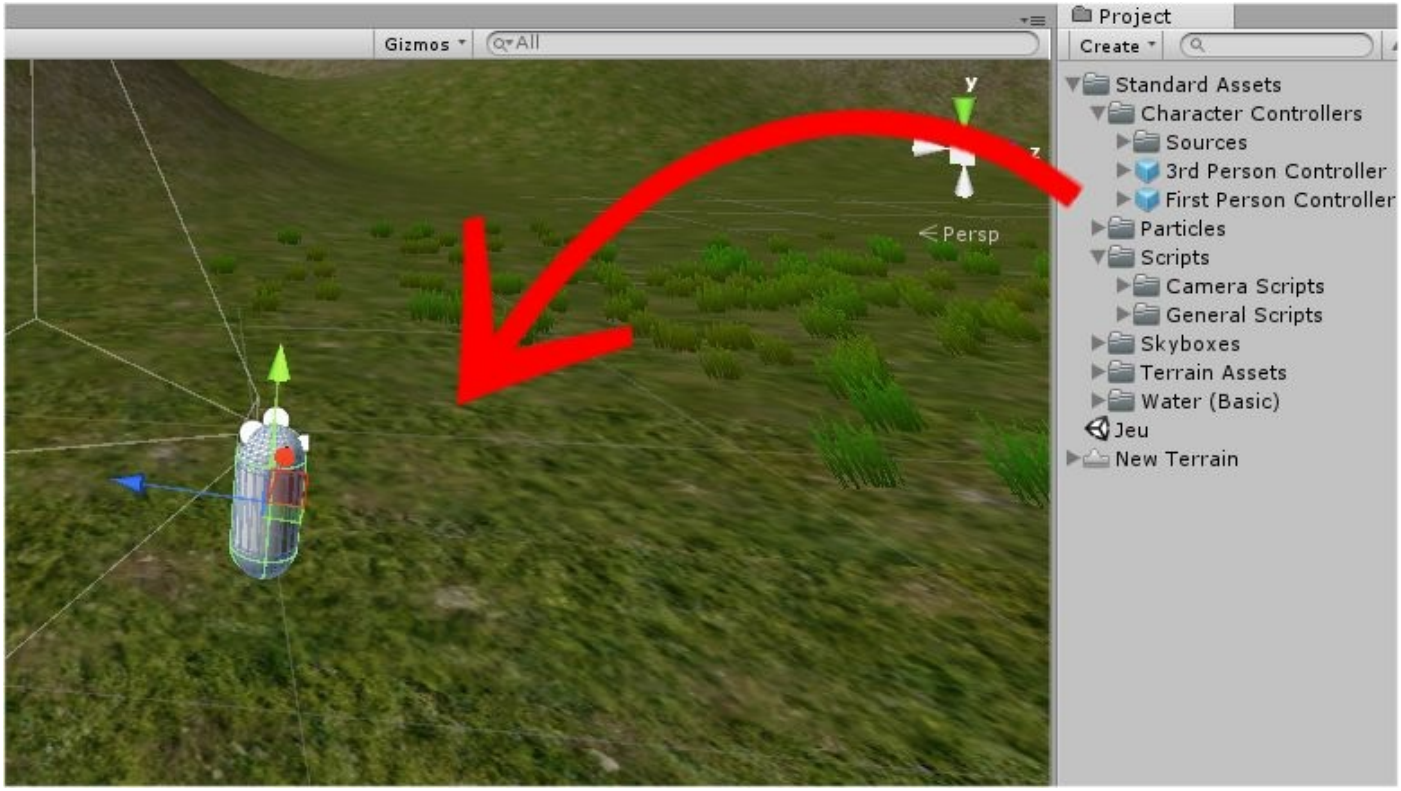
Vous pouvez ajouter d'autres éléments décoratifs comme par exemple des systèmes de particules. Vous en trouverez dans les ressources. Vous pouvez créer une cascade d'eau en utilisant les particules `WaterFall` et `Water Surface Splash`. Pour cela, faites encore une fois un glisser/déposer du prefab vers votre scène.

Figure 3.21 : *Création d'une cascade*



Maintenant que vous avez compris comment construire un monde 3D, je vous propose d'ajouter un personnage joueur contrôlable au clavier et à la souris afin de tester notre niveau. Nous choisirons un personnage en vue à la première personne (FPS). Allez chercher le prefab First Person Controller se trouvant dans le dossier `Character Controllers` des standard assets et déposez le prefab sur votre terrain.

Figure 3.22 : *Ajout d'un personnage jouable*



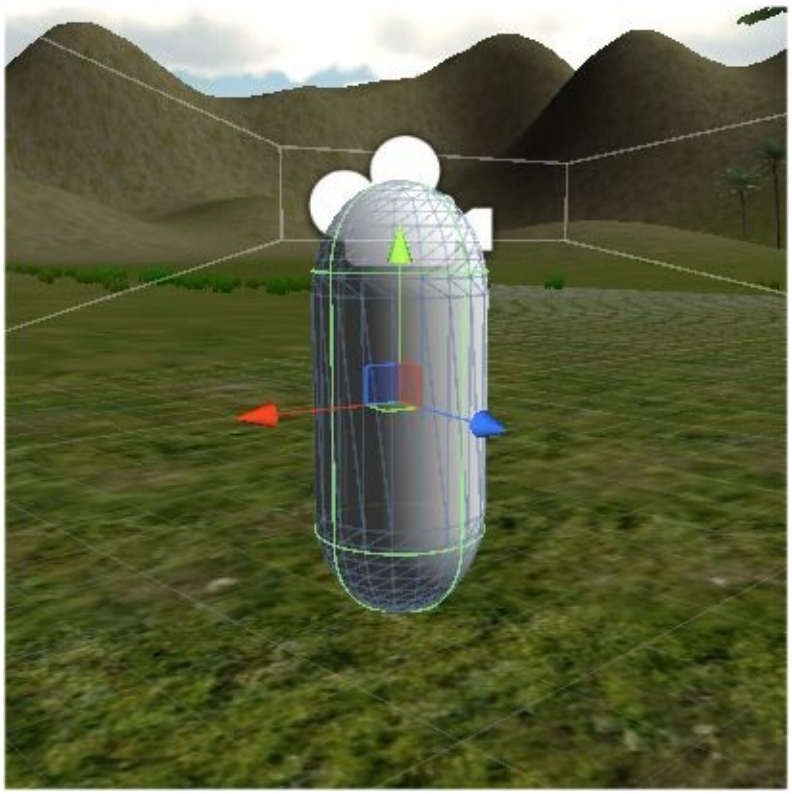
Note *> Le first person controller est une petite capsule grise que vous pouvez contrôler pour naviguer dans votre monde 3D. Il s'agit d'un personnage jouable et entièrement configuré que vous pouvez utiliser dans toutes vos scènes. La capsule grise vous permet simplement de voir l'objet dans la scène afin de le placer au bon endroit.*

Vous devez impérativement surélever la petite capsule afin de la positionner bien au-dessus du sol (la capsule ne doit pas toucher le sol sinon vous passerez au travers). Pour cela vous devez utiliser l'outil Transform sous la barre de menu :

Figure 3.23 : *Outil Transform*

Cela vous permet d'avoir accès aux flèches représentant les axes de l'objet lorsque vous cliquez sur celui-ci :

Figure 3.24 : *Les 3 axes (x, y et z)*



Dans notre cas de figure, nous devons utiliser la flèche verte (qui devient jaune lorsqu'on clique dessus) afin de "soulever" le modèle 3D. Si vous avez bien placé le joueur, vous pouvez le contrôler pour vous balader dans la scène. Pour cela, cliquez sur l'icône Play en haut de l'interface :

Figure 3.25 : *Lancement du jeu*

Vous pouvez découvrir le superbe monde que vous avez créé de vos propres mains ! Utilisez les flèches du clavier pour vous déplacer et la souris pour regarder autour de vous. Pour arrêter de jouer, cliquez de nouveau sur l'icône Play. Pensez à sauvegarder votre scène (File/Save Scene).

Dans le chapitre suivant, nous allons voir comment créer nos propres prefabs et ainsi découvrir comment nous allons pouvoir réutiliser plusieurs fois certaines ressources.

Dans ce chapitre, nous avons conçu notre premier monde 3D. Pour cela, nous avons créé un terrain et nous avons ajouté de nombreux éléments pour enrichir le décor. Maintenant vous savez comment :

- créer un terrain ;
- ajouter de la végétation ;
- ajouter des modèles 3D/particules ;
- créer un ciel ;

- ajouter un personnage jouable.

4

Les éléments préfabriqués

1. Comprendre les prefabs

Un jeu vidéo est un monde virtuel à part entière. Ce monde est composé d'éléments individuels comme par exemple des personnages, des arbres ou des maisons. Tous ces éléments sont créés par des personnes qui les assemblent pour constituer cet univers. Il faut bien comprendre que la modélisation d'un personnage, même secondaire, demande du temps et de l'argent. De plus, les modèles 3D doivent être stockés sur l'ordinateur du joueur. Il est impossible (humainement parlant et technologiquement parlant) de modéliser et de stocker chaque arbre d'une forêt par exemple. Vous avez certainement déjà joué à beaucoup de jeux dans lesquels les personnages étaient les mêmes (ou au mieux avec une texture différente). Si les concepteurs devaient modéliser chaque modèle 3D afin qu'il soit unique, la réalisation d'un jeu prendrait des années et il occuperait trop de place sur le disque. Les développeurs pratiquent donc la réutilisation de ressources. Comme vous pouvez le constater, la forêt de palmiers que nous avons créée dans le chapitre précédent n'est composée que d'un seul arbre. Nous pouvons tricher et changer le rendu d'un objet 3D en modifiant sa taille ou sa couleur afin de faire croire qu'il est différent. Cela nous fait gagner du temps et de la place en mémoire. C'est ce que nous allons apprendre à faire dans ce chapitre.

1.1. Qu'est-ce qu'un prefab ?

Un prefab est un objet constitué de plusieurs éléments. Par exemple, un personnage 3D constitué d'une texture, d'un objet de collision et d'un script de mouvement est un prefab car il s'agit d'un personnage préconfiguré prêt à être réutilisé ultérieurement. Prenons un autre exemple : une cabane en bois construite avec plusieurs cubes et plusieurs textures peut être stockée dans un prefab afin que nous puissions la réutiliser par la suite.

1.2. Comment créer un prefab ?

Avec Unity, la procédure est très simple. Vous devez dans un premier temps préparer un objet ; dans notre cas, nous allons créer une cabane en bois à partir de cubes. Puis, une fois l'objet créé, vous le glissez/déposez de la hiérarchie vers la fenêtre de projet. Nous allons faire la manipulation à la section suivante.

2. Créons un prefab de cabane

Pour commencer, nous devons créer une cabane en bois. Pour cela, nous avons besoin de planches en bois (cube + texture).

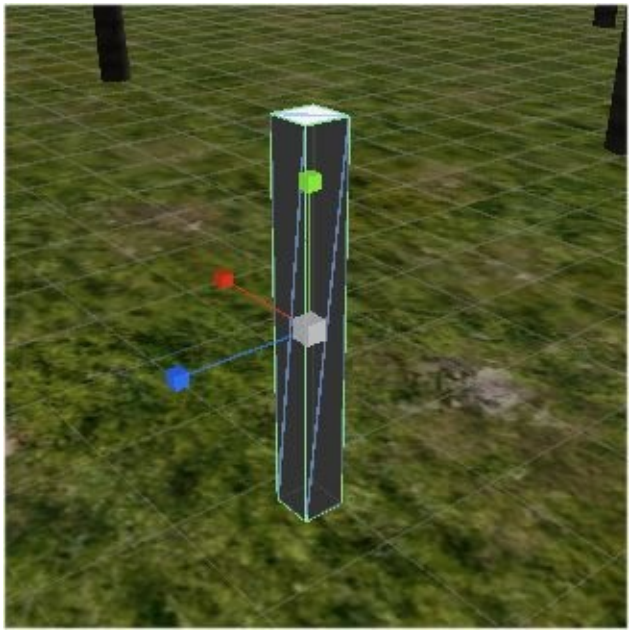
2.1. Du cube au pilier

Construisons les quatre pieds qui vont porter la cabane. Nous allons créer le premier à partir d'un cube, puis nous le dupliquerons trois fois. Pour créer le cube, allez dans le menu Game Object/Create Other/Cube. Le cube apparaît alors dans la scène. Dans la barre d'outils, sélectionnez l'outil Scale permettant la déformation d'un objet :

Figure 4.1 : *L'outil Scale*

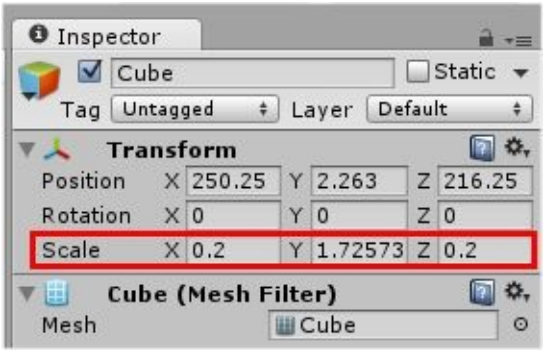
Maintenant vous pouvez déformer le cube afin de l'allonger et de diminuer son diamètre. Utilisez les trois axes pour effectuer la déformation :

Figure 4.2 : *Déformation d'un cube*



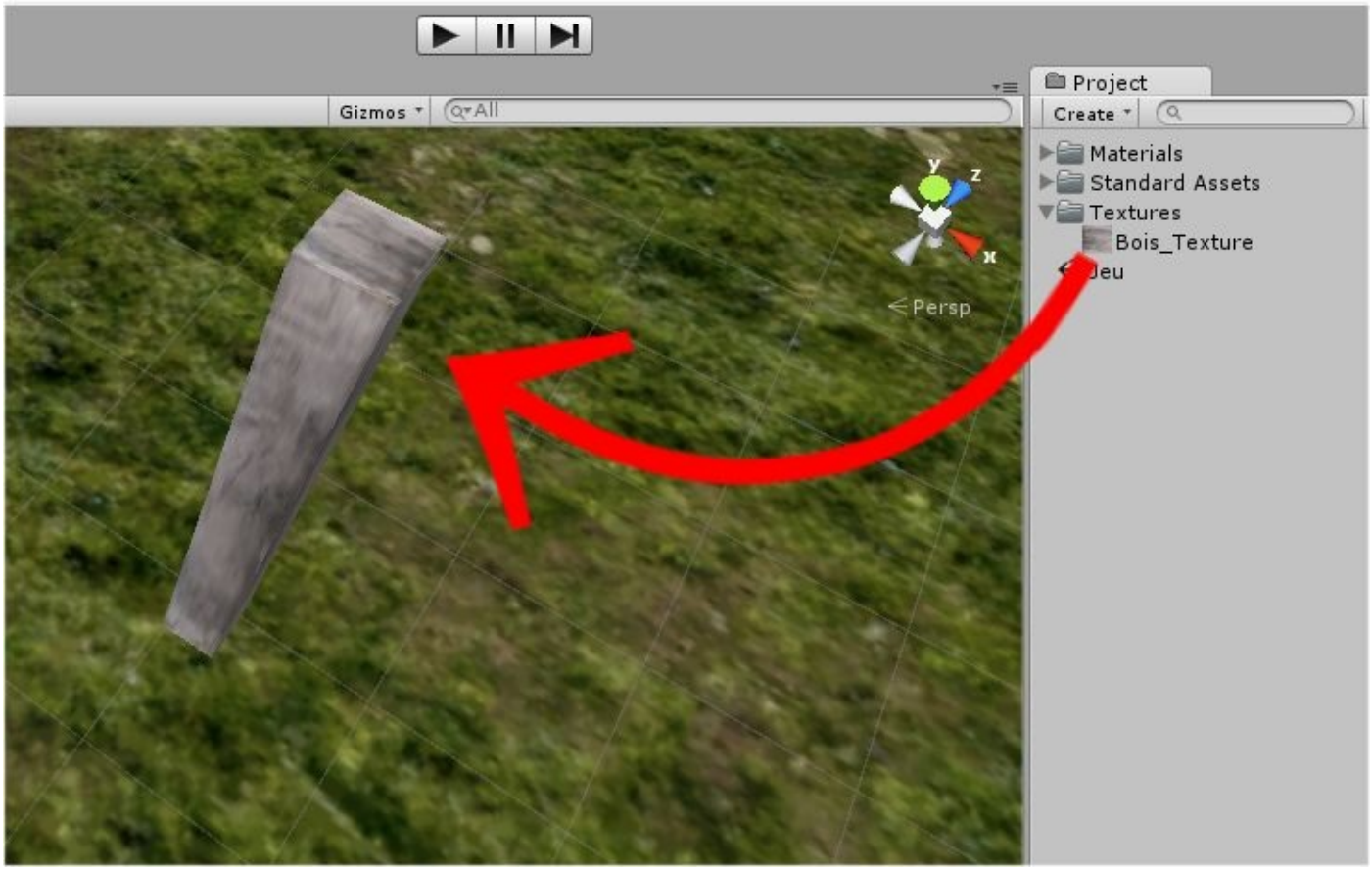
Le pilier prend forme. Sachez que vous pouvez aussi passer par l'inspector pour déformer votre cube avec plus de précision. Vous pouvez modifier manuellement les valeurs de la propriété Scale du cube dans la section Transform :

Figure 4.3 : *Déformation d'un cube via l'inspector*



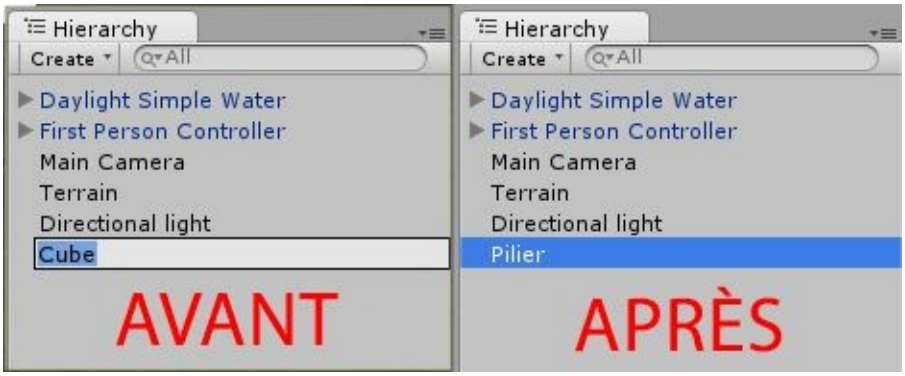
Nous allons maintenant ajouter une texture à notre pilier en bois. Une texture n'est ni plus ni moins qu'une photo représentant la surface d'un matériau qui, une fois appliquée à l'objet, lui en donne la consistance. Vous pouvez télécharger une texture de bois en tapant "wood texture" sur Google et en sélectionnant le visuel de votre choix. Pour un jeu commercial, vous devez faire très attention à n'utiliser que des textures libres de droits. Vous pouvez prendre n'importe quelle image ou récupérer celle que j'ai utilisée dans les sources téléchargeables de ce livre. Pour importer la texture dans Unity, vous devez la glisser/déposer dans la fenêtre de projet. Une fois la texture importée, vous pouvez l'appliquer à votre pilier en la faisant cette fois glisser de la fenêtre Projet sur l'objet dans la fenêtre Scène (voir Figure 4.4).

Figure 4.4 : *Ajout d'une texture au pilier*



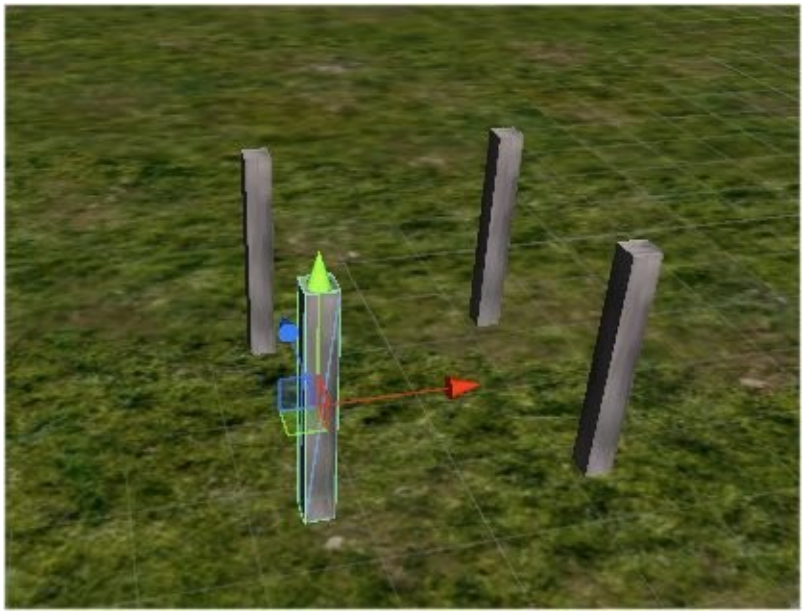
Vous allez maintenant organiser votre scène en nommant vos objets. Pour cela, cliquez sur l'élément Cube se trouvant dans la liste des objets de la hiérarchie et appuyez sur la touche F2 pour le renommer en *Pilier*.

Figure 4.5 : *Modification du nom d'un objet*



Nous avons besoin de quatre piliers pour construire notre cabane. Nous allons en créer un que nous allons ensuite dupliquer afin d'en obtenir trois copies supplémentaires. Pour dupliquer un objet, sélectionnez-le dans la scène en cliquant dessus, puis pressez les touches Ctrl+D. Vous pouvez aussi cliquer droit sur son nom dans la hiérarchie et sélectionner l'entrée Duplicate dans le menu contextuel. Répétez l'opération trois fois pour obtenir quatre piliers en bois. Utilisez l'outil Transform afin de déplacer les piliers pour les espacer.

Figure 4.6 : *Duplication du pilier*



2.2. Créer un objet Cabane

Nous allons une nouvelle fois devoir organiser notre projet. Comme vous pouvez le constater, vous avez quatre objets Piliers dans la hiérarchie. Si vous créez trois cabanes, vous aurez douze objets avec le même nom. Afin de ne pas surcharger votre espace de travail, vous allez les regrouper au sein d'un objet unique Cabane. Cliquez sur le menu GameObject/Create Empty : un objet vide nommé GameObject va alors apparaître dans la hiérarchie. Renommez-le *Cabane*, puis sélectionnez les quatre piliers et glissez/déposez-

les dedans.

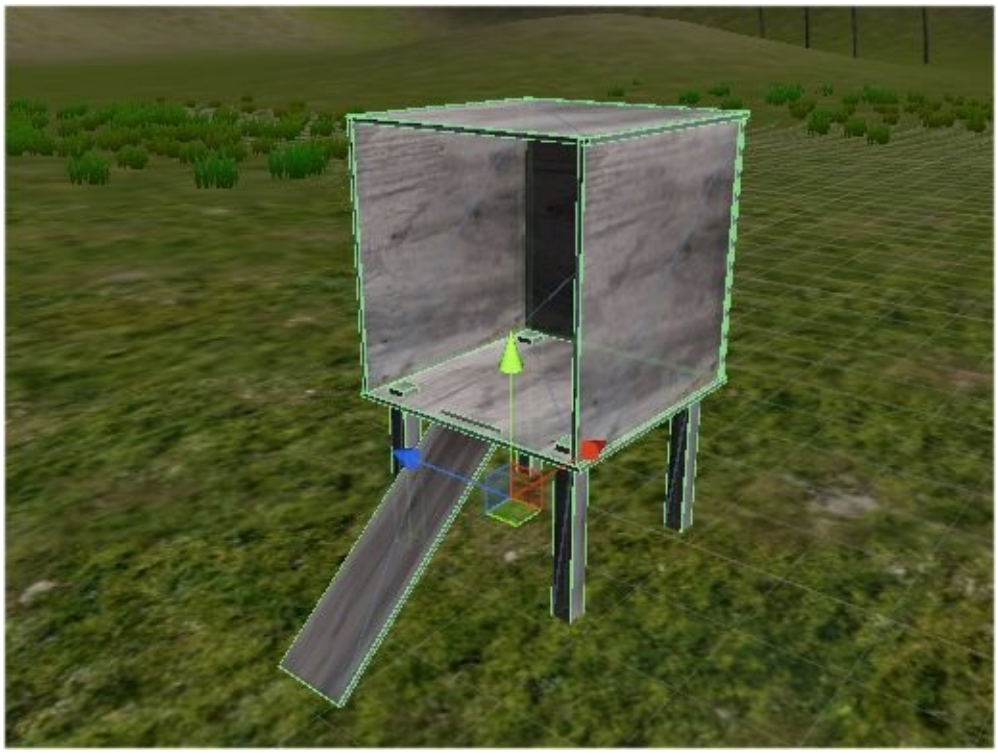
Figure 4.7 : *Création de l'objet Cabane*

Cela va nous permettre d'une part de mieux nous organiser et d'autre part de nous faciliter le travail lorsque nous créerons le prefab de notre cabane une fois celle-ci terminée.

Nous allons maintenant créer le plancher, les murs et le toit de la cabane. Vous allez partir d'un nouveau cube que vous agrandirez et aplatirez pour former une sorte de planche en bois. N'oubliez pas d'ajouter la texture. Une fois terminé, dupliquez-le cinq fois (trois murs, un toit, un plancher) et glissez les cubes dans l'objet Cabane créé précédemment.

Pensez à utiliser l'outil de rotation  pour faire tourner les cubes si besoin. Voici le résultat que j'ai obtenu :

Figure 4.8 : *La cabane complète*



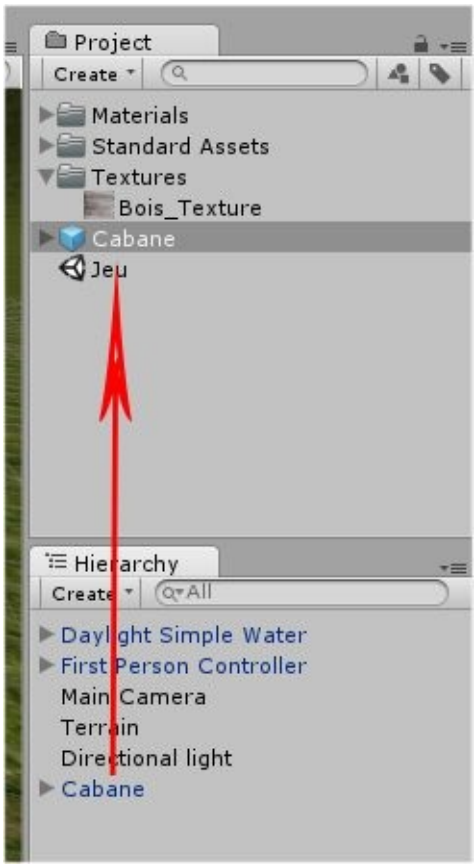
J'ai ajouté une planche permettant au joueur de pouvoir rentrer dans la cabane. Vous devriez obtenir une cabane similaire à la mienne mais vous êtes totalement libre de créer

la cabane de vos rêves (plus grande, plusieurs pièces, plusieurs étages, etc.). Vous avez réussi à créer votre premier objet 3D en partant de plusieurs cubes !

2.3. De l’objet au prefab

Maintenant que vous vous êtes donné du mal pour obtenir ce bel objet, il faudrait pouvoir le réutiliser plusieurs fois dans votre jeu. Vous allez donc créer un prefab de votre cabane. Rien de plus simple : glissez/déposez-le de la hiérarchie vers la fenêtre Projet. Un cube bleu devrait alors apparaître dans votre projet. Ce cube est votre cabane. Pour créer une nouvelle cabane, vous n’aurez plus qu’à glisser/déposer le prefab de la fenêtre de projet vers la scène !

Figure 4.9 : *Création du prefab*



Comme vous le voyez, vous pouvez facilement créer des prefabs afin de réutiliser vos créations. Testez celui-ci en créant plusieurs cabanes dans votre scène. Pour donner l’illusion qu’elles sont différentes, vous pouvez changer la taille, l’orientation et même la texture si vous le souhaitez. Vous pouvez également supprimer ou ajouter des planches pour en modifier la structure sans pour autant tout recommencer depuis le début.

Note > *Pensez à créer systématiquement des prefabs pour tous les objets que vous allez réutiliser dans votre jeu. Cela vous fera gagner du temps de développement et vous permettra de mieux vous organiser.*

2.4. Organiser ses prefabs en package

Vous pouvez rassembler vos prefabs sous forme de packages. Ces packages seront exportés sur votre ordinateur et vous pourrez à tout moment les importer dans Unity, par exemple dans un futur projet de jeu. Pour exporter un package, sélectionnez tous les éléments que vous souhaitez exporter dans la fenêtre de projet, cliquez droit et sélectionnez Export package dans le menu contextuel. Unity va lister tous les éléments à exporter ; cliquez sur Export… en bas de la fenêtre pour spécifier un nom et choisir l'emplacement où vous souhaitez enregistrer le package. Si dans un futur projet vous souhaitez importer ce package, ouvrez le menu Assets/Import package/Custom package et allez chercher votre package fraîchement créé.

Vous savez désormais créer et réutiliser des objets. Vous pouvez donc encore enrichir votre monde virtuel. Dans le prochain chapitre, les choses vont commencer à se corser car nous allons apprendre à programmer nos premiers scripts C# pour pouvoir interagir avec l'environnement et permettre de ramasser des objets. Avant de coder ce script, nous aborderons quelques concepts de base du langage.

Au cours de ce chapitre, vous avez vu comment créer des objets 3D et les réutiliser en passant par le système de prefabs de Unity. Vous savez donc :

- modéliser un objet à partir de primitives ;
- créer un prefab ;
- utiliser des prefabs ;
- exporter des packages.

***Note** > On appelle primitive une forme simple comme un cube, une sphère, etc.*

5

Premiers scripts C#

La programmation (création de scripts) est une étape indispensable lorsque vous créez un jeu. En effet, la plus grosse partie du travail (et aussi la plus complexe) est de programmer tous les événements qui doivent se produire au cours du jeu. Par exemple, le développeur doit créer un script permettant au joueur de ramasser les pièces d'or du niveau. De plus, il doit coder la fonction permettant d'afficher le nombre de pièces se trouvant dans l'inventaire du joueur. De façon plus générale, tout événement aussi simple soit-il (ouvrir une porte) nécessite de la programmation.

Trois langages de programmation sont compatibles avec Unity : le C#, le JavaScript et le Boo. Leur syntaxe est différente mais les noms des fonctions sont les mêmes. 85 % des développeurs utilisent le C#, 24 % le JavaScript et moins de 1 % le Boo. C'est pourquoi nous utiliserons le C#, qui est le plus populaire, le mieux documenté et doté d'une plus grande communauté.

Astuce *> Même si le C# est un peu plus complexe que le JS, il est fortement conseillé de l'apprendre en priorité. Le Boo va disparaître et ne sera plus disponible d'ici les prochaines versions de Unity et la documentation du logiciel ne proposera quasiment plus que des exemples en C#.*

Dans ce chapitre, nous allons voir comment écrire ses propres scripts C# et utiliser la documentation officielle pour aller plus loin.

Note *> Vous retrouverez tous les codes avec les sources téléchargeables du livre.*

1. Structure d’un script et variables

Reprenons l’exemple des pièces que le joueur doit ramasser. En langage naturel, nous pouvons dire : “Le joueur a 0 pièces d’or. Si le joueur ramasse une pièce, il en aura 0 + 1”. En langage informatique, nous écrirons la même chose tout en respectant la syntaxe du langage. Celle-ci est constituée de plusieurs éléments de base que nous allons découvrir au fur et à mesure. Pour commencer, nous allons avoir besoin d’une variable pour stocker le nombre de pièces d’or.

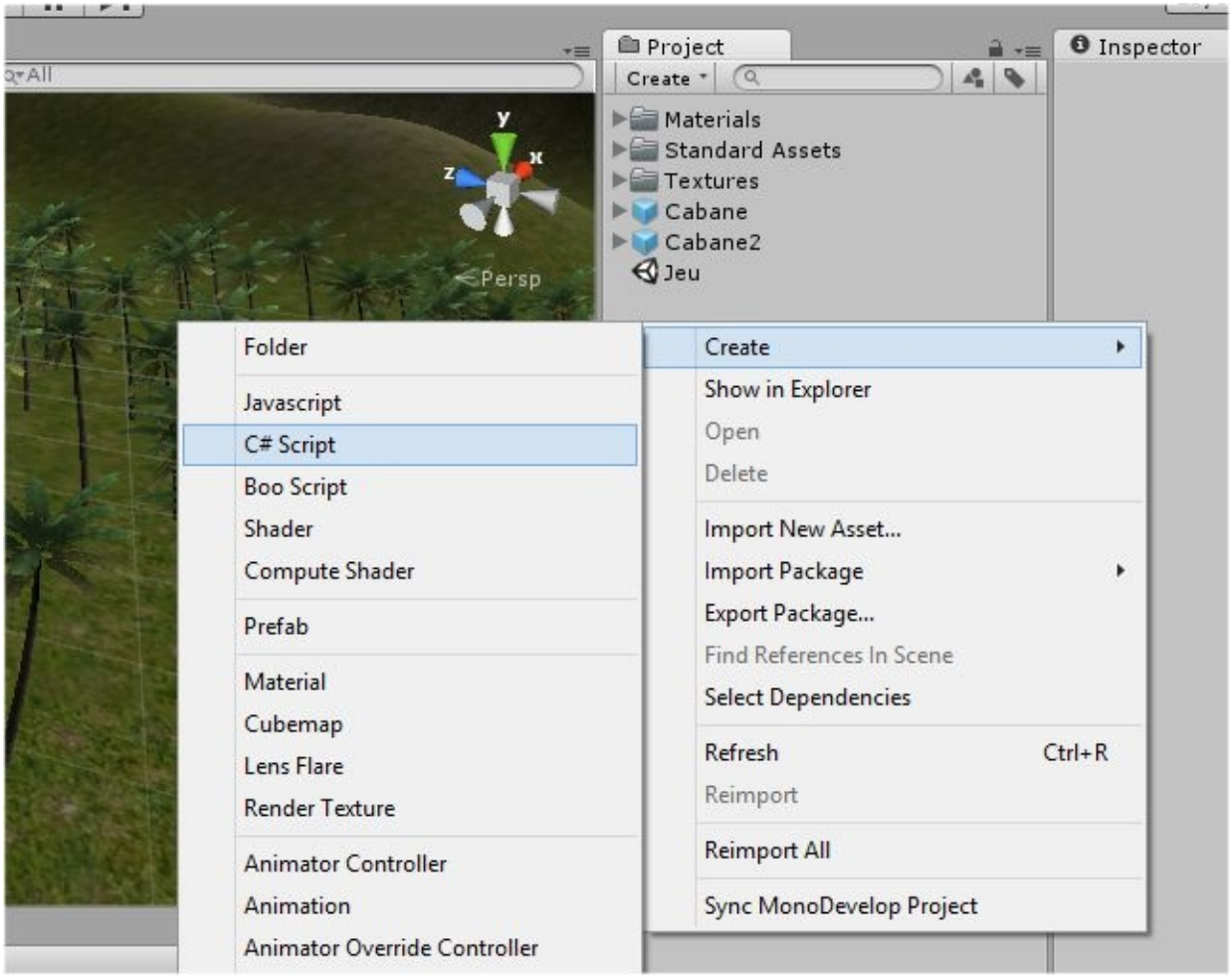
Une *variable* est un élément qui a un nom, un type et une valeur qui peut varier. Le but d’une variable est de stocker une valeur en mémoire afin de pouvoir la réutiliser par la suite. Vous pouvez lui donner le nom de votre choix à condition de ne pas utiliser un mot réservé par le langage. Dans notre exemple, je l’ai appelée `money`, mais vous pourriez tout aussi bien la nommer `NombreDePieces` ou `nbPieces` ou encore `pieces`. Enfin, une variable doit être typée, c’est-à-dire qu’on doit indiquer à l’avance la nature de sa valeur. Les principaux types sont :

- `int` : valeur entière (1, 8, 52, etc.) ;
- `float` : nombre à virgule (2.56, 23.876, 1.0, etc.) ;
- `bool` : variable booléenne (vrai ou faux) ;
- `string` : chaîne de caractères (“coucou”, “test”, “chaîne de caractères”, etc.).

Nous allons voir maintenant comment créer votre première variable.

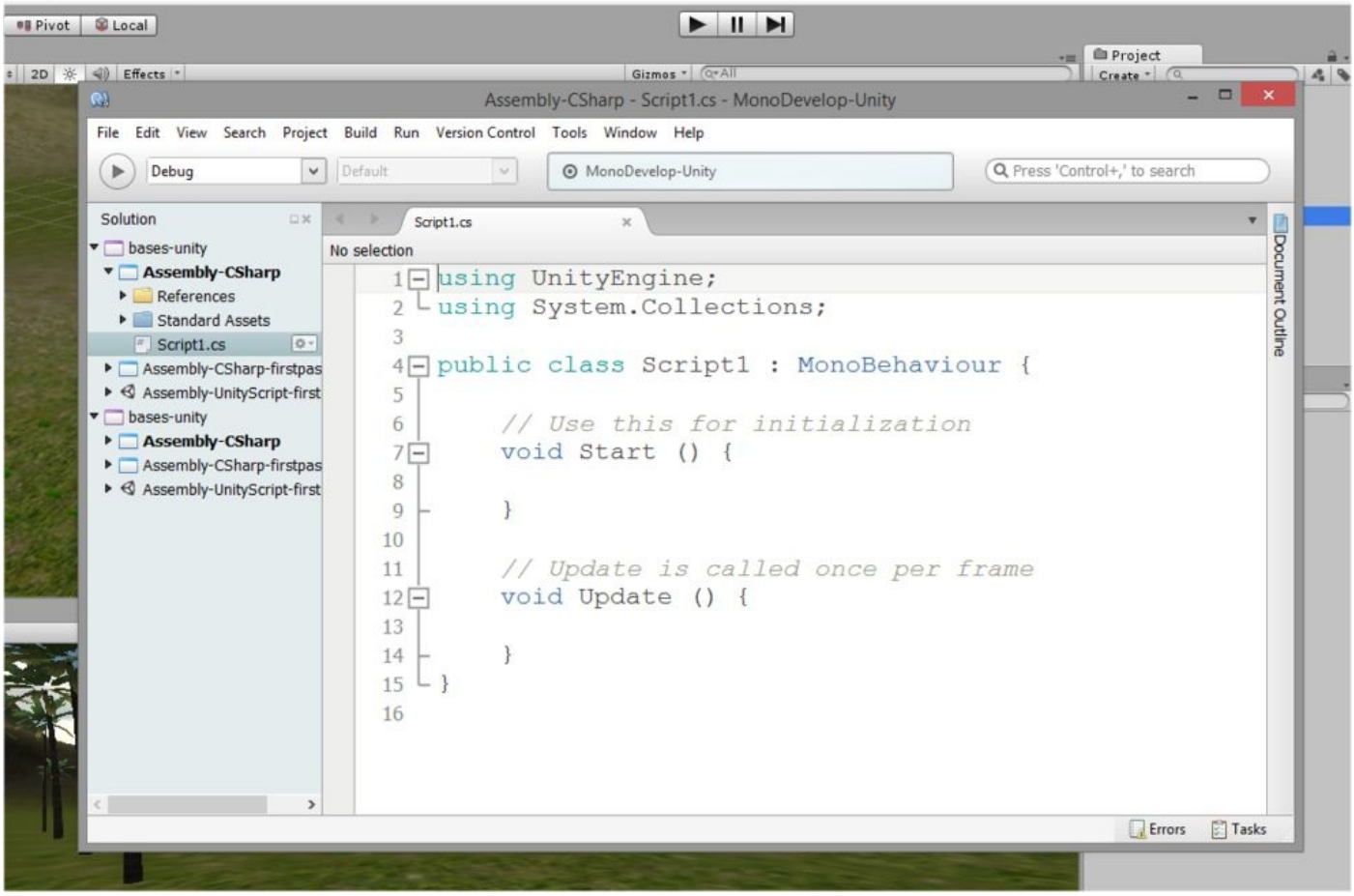
Avant toute chose, pour pouvoir commencer à programmer, vous devez créer un nouveau script. Pour cela, cliquez droit dans la fenêtre de projet et, dans le menu contextuel qui apparaît, sélectionnez Create/C# Script. Donnez-lui un nom, par exemple `Script1`, et appuyez sur la touche Entrée pour valider.

Figure 5.1 : *Création d’un script C#*



Votre nouveau script apparaît dans le projet et vous pouvez commencer à coder. Double-cliquez sur son nom pour l'ouvrir. La fenêtre de MonoDevelop, un logiciel installé avec Unity, va s'ouvrir et c'est là que vous allez écrire vos lignes de code.

Figure 5.2 : *MonoDevelop*



Comme vous pouvez le constater, le script n'est pas vide. Les `using` sur les premières lignes permettent d'importer des fonctions du langage. La fonction `Start`, déclarée de la façon suivante : `void Start(){}`, est la première fonction exécutée par le jeu et permet d'initialiser des variables par exemple. La fonction `Update` est une fonction qui s'exécute en boucle (60 fois par seconde si le jeu tourne en 60 images par seconde). Cela permet par exemple de gérer les animations ou le temps écoulé.

Vous devez déclarer les variables au tout début de votre classe (juste avant la fonction `Start`). Pour déclarer une variable, comme par exemple la variable `money` de type `int` avec une valeur de 0, vous devez écrire ceci :

```
int money = 0;
```

Vous déclarez le type (`int`), vous spécifiez le nom (`money`), vous attribuez une valeur (`= 0`) et vous terminez l'instruction par un point virgule (`;`).

Pour modifier la valeur d'une variable, vous pouvez utiliser n'importe quel opérateur mathématique (`+, -, *, /`). Faites l'essai et écrivez dans la fonction `Start` :

```
money = money + 5;
```

Cela va ajouter 5 à la valeur stockée dans votre variable.

Pour afficher le contenu de la variable, vous pouvez utiliser l'instruction `print` de la façon suivante :

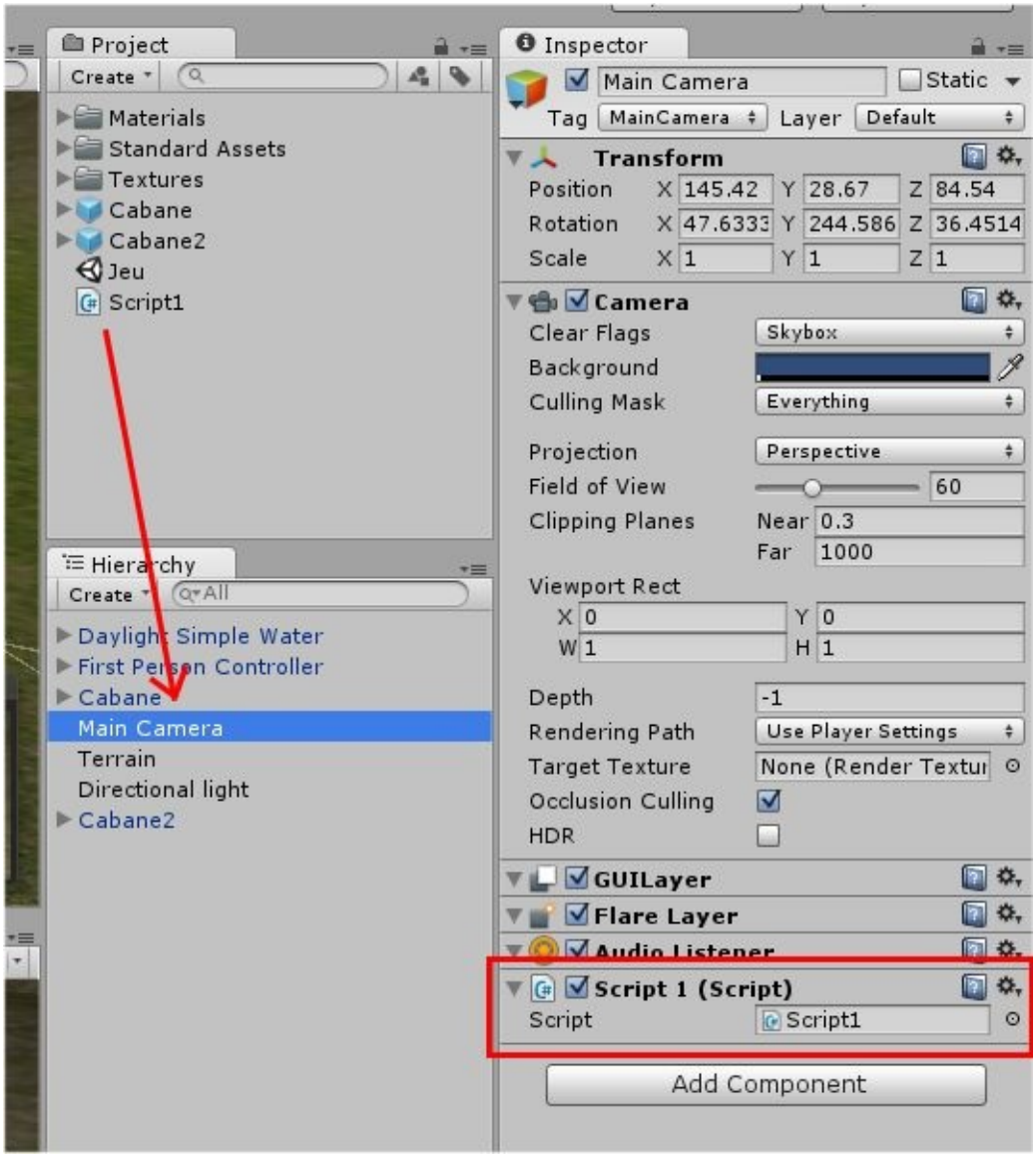
```
print(money); // Affiche la valeur de money dans la console
```

Le double slash suivi d'une phrase est un commentaire ignoré par Unity. Les commentaires vous permettent de commenter votre code pour mieux vous y retrouver. Vous devriez avoir le code suivant :

```
using UnityEngine;
using System.Collections;

public class Script1 : MonoBehaviour {

        int money = 0;

        // Use this for initialization
        void Start () {
                print (money);
                money = money + 5;
                print (money);
        }

        // Update is called once per frame
        void Update () {

        }
}
```

Pour tester votre script, vous devez l'enregistrer, retourner dans Unity et le faire glisser sur n'importe quel objet de votre scène, par exemple la caméra :

Figure 5.3 : *Utilisation d'un script*



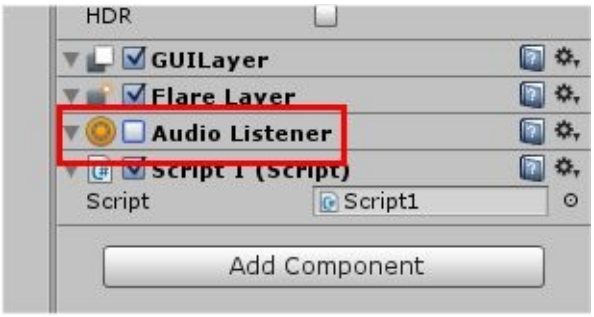
Après avoir ajouté le script à la caméra, il apparaît dans l'inspector. Vous pouvez lancer le jeu afin de voir si tout fonctionne. S'il n'y a pas d'erreur, vous devriez avoir ceci dans la console :

Figure 5.4 : *La console de debug*

Note *> Pour afficher la console, cliquez sur le petit encart tout en bas de l'interface (encadré rouge sur la Figure 5.4).*

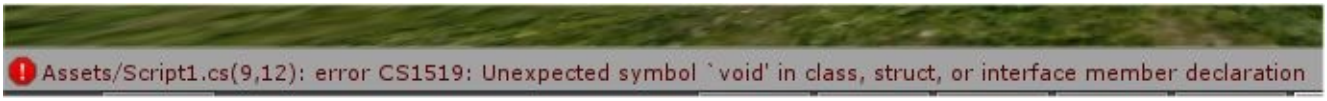
La console contient vos prints et les messages renvoyés par Unity. Vous pouvez voir les valeurs “0” et “5”. Cela correspond aux deux `print` qui affichent la valeur de `money`. Nous avons également le message suivant “There are 2 audio listeners in the scene…” qui signifie qu’il y a deux Audio listeners. L’Audio listener permet d’entendre les sons du jeu. Vous ne devez en avoir qu’un dans votre scène, en général sur la caméra représentant les yeux de votre personnage. Actuellement nous avons un Audio listener sur la caméra mais il y en a aussi un sur le First person controller. Décochez l’Audio listener de la Main Camera afin de supprimer ce message :

Figure 5.5 : *Audio listener*



Si vous avez une erreur dans votre code (oubli d’un `;`, mauvais nom de script ou de variable, etc.), vous obtiendrez un message d’erreur :

Figure 5.6 : *Message d’erreur*



Dans ce cas-là, le jeu ne se lance pas tant que l’erreur n’est pas corrigée.

2. Les fonctions

Les fonctions sont des blocs d'instructions. Un bloc est un ensemble d'instructions entouré par des accolades. Par exemple, ceci est une fonction :

```
void Start () {
        print (money);
        money = money + 5;
        print (money);
}
```

Une fonction est composée d'un type de retour (`void` = pas de retour). Si votre fonction doit renvoyer un nombre entier, elle aura pour type de retour `int`. Une fonction doit avoir un nom et éventuellement des paramètres. Les paramètres sont spécifiés entre parenthèses et séparés par des virgules. Nous allons voir deux exemples différents. La fonction suivante n'a pas de type de retour car elle ne renvoie rien, elle doit simplement afficher "Bonjour" dans la console :

```
void DireBonjour () {
        print ("Bonjour");
}
```

Pour appeler cette fonction, par exemple dans `Start`, vous devez écrire le nom de la fonction :

```
void Start () {
        DireBonjour (); // Affiche bonjour dans la console
}
```

Maintenant voici une fonction avec retour et paramètres :

```
int Ajouter (int nombre1, int nombre2) {
        return nombre1 + nombre2;
}
```

Ici la fonction `Ajouter` prend deux nombres en paramètres et les additionne pour retourner le résultat :

```
void Start () {
        print(Ajouter (4,6)); // Affiche 10 (la somme) dans la console
}
```

Les fonctions sont très souvent utilisées pour effectuer des actions répétitives (marcher, ramasser, attaquer, afficher, utiliser, etc). Il existe des fonctions prédéfinies que vous pouvez utiliser. Par exemple `OnTriggerEnter(Collider obj)`, qui permet de détecter lorsqu'un objet entre en collision avec un autre objet. C'est cette fonction que nous utiliserons pour savoir si le joueur a touché une pièce et s'il peut la ramasser.

3. Les conditions

Les conditions sont énormément utilisées en programmation. Elles permettent de tester si une chose a eu lieu ou non et d'agir en conséquence. Par exemple comment détecter que le joueur n'a plus de vie et qu'il a perdu ? Nous utilisons une condition : `SI (VIE == 0) {perdu = vrai;}`. Voilà à quoi ressemble une condition dans un pseudo code facile à comprendre. Maintenant en C#, c'est presque pareil :

```
int vie = 1;
void Start(){
        if(vie == 0){
                print("Perdu");
        }
}
```

Essayez ce script et regardez si le message "Perdu" apparaît dans la console. Il n'apparaît pas car la vie n'est pas égale à 0. Maintenant réessayez ce script en mettant 0 à la vie. Vous verrez que le message perdu sera affiché. Les conditions nous permettent d'exécuter un bout de code si une condition est respectée (Si le joueur a la clé alors la porte s'ouvre…). Nous pouvons créer plusieurs conditions en utilisant des `else if` (sinon si) et créer un bloc par défaut `else` :

```
int vie = 3;
void Start(){
        if(vie > 0){
                print("Vie plus grand que 0");
        }
        else if(vie < 0){
                print("Vie plus petit que 0");
        }
        else {
                print("Vie = 0");
        }
}
```

Vous allez très souvent être amené à utiliser les conditions pour le développement de vos jeux. Vous pouvez créer des conditions multiples afin de tester plusieurs variables en même temps. Par exemple, si vous voulez savoir si la vie est supérieure à 3 et si l'or est aussi supérieur à 3, vous pouvez procéder de la façon suivante :

```
int vie = 3;
int or = 3;

void Start(){
        if(vie > 3 && or > 3){ // Si vie > 3 et or > 3
        print("La condition retourne vrai");
        }
}
```

Vous noterez l'emploi de `&&`, qui signifie "Et aussi". Pour tester si `vie > 1` OU si `or > 1`, vous utiliserez `||`, qui signifie "Ou sinon" :

```
int vie = 3;
int or = 3;

void Start(){
        if(vie > 1 || or > 1){ // Si vie > 1 ou or > 1
        print("La condition retourne vrai");
        }
}
```

Vous pouvez imbriquer les conditions si vous avez besoin de faire des tests avancés.

4. Utilisation de la documentation

Unity dispose de centaines de fonctions. Il vous est impossible de toutes les connaître car il y a trop de commandes. Les développeurs ont toujours la documentation sous les yeux pour programmer. Je vous invite également à vous y reporter.

Pour savoir, par exemple, ce que vous pouvez faire sur un GameObject, entrez “gameobject” dans la barre de recherche et vous tomberez sur la page dédiée à la classe. Dans la liste des fonctions, vous trouverez `SetActive`, qui permet d’activer ou de désactiver un GameObject. Ainsi vous saurez que pour désactiver un GameObject il faut faire :

```
        void Start(){
        gameObject.SetActive(false);
}
```

Si vous voulez faire une recherche sur le web, je vous conseille d’écrire en anglais. Par exemple, pour trouver comment changer la couleur d’un objet je recherche “Unity change gameobject color” et je trouve immédiatement le résultat suivant :

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {
        public GameObject other;
        void Example() {
                other.renderer.material.color = Color.green;
        }
}
```

Enfin, n’hésitez pas à faire appel à la communauté pour obtenir de l’aide en programmation. En français, vous avez le forum Unity3d-France ou encore mes sites internet formation-facile et unity3d-dev. Il y aura toujours quelqu’un ou moi-même pour vous répondre.

Dans le chapitre suivant, nous allons mettre en pratique ce que nous avons vu afin de créer un script qui va nous permettre de ramasser des objets.

Dans ce chapitre, vous avez vu les bases de la programmation avec le langage C#. Vous savez ce qu’est :

- une variable ;
- une fonction ;
- une condition ;

et comment poursuivre seul votre apprentissage des fonctions à l’aide de la documentation.

6

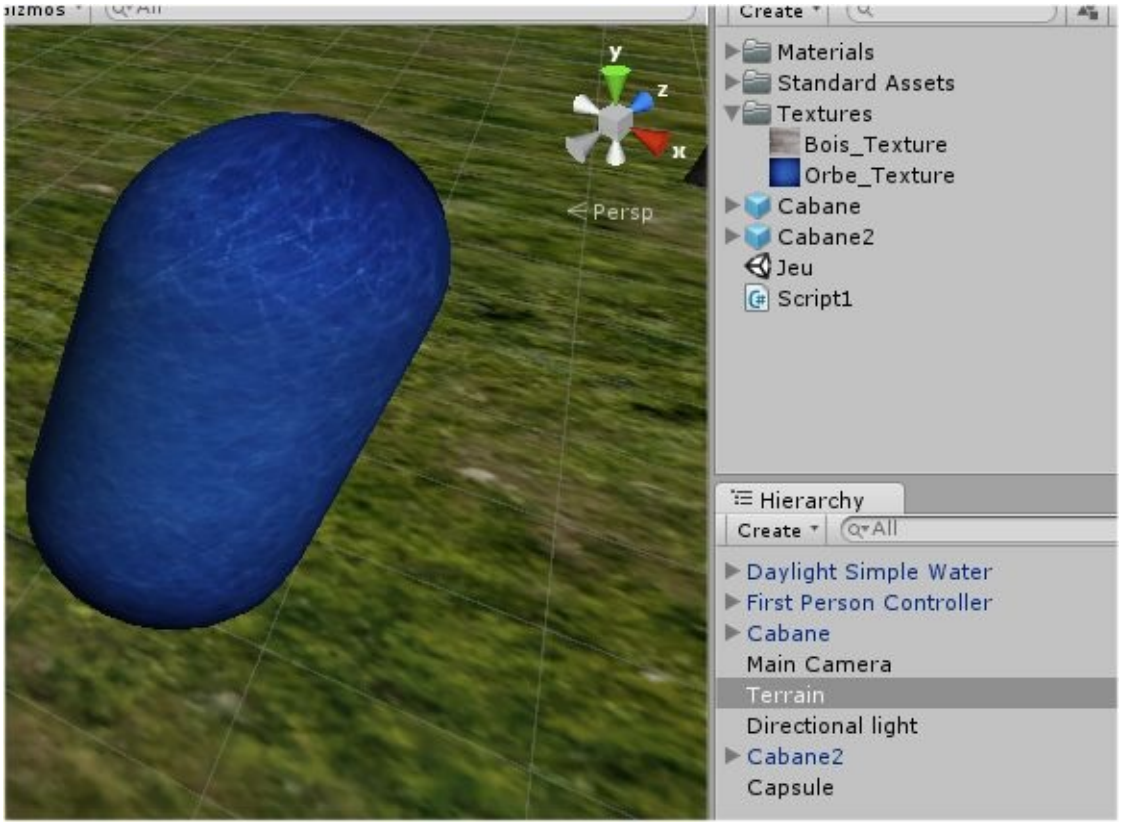
Interagir avec l’environnement

Maintenant que nous avons vu comment sont construits les scripts, nous allons en écrire un qui va permettre de ramasser des petites capsules (les pièces de monnaie de notre monde virtuel que nous appellerons des *orbes*).

1. Création d’une capsule

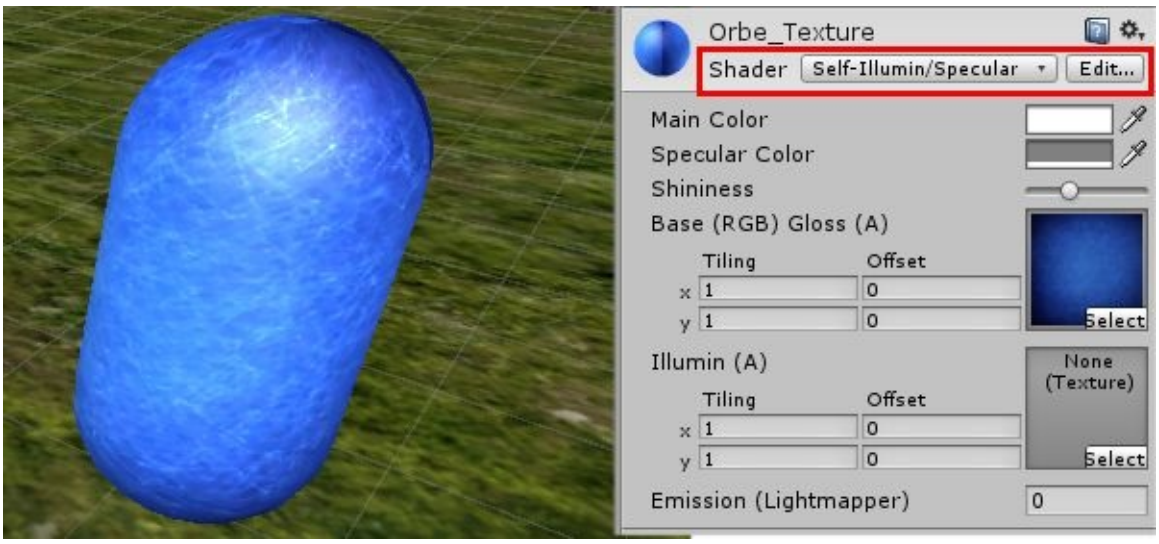
Nous allons créer une sorte de capsule qui représentera une orbe, la monnaie de notre monde virtuel. Pour cela, cliquez sur GameObject/3D Object/Capsule. Diminuez la taille de la capsule et ajoutez une texture. Pour ma part, je vais lui appliquer une texture bleue :

Figure 6.1 : *Création de l’orbe*



Modifiez le shader de la capsule en passant par l’inspector. Le shader permet de modifier l’aspect visuel de l’objet. Par exemple, pour le rendre lumineux et brillant, j’utilise le shader `Self-Illumin/Specular`.

Figure 6.2 : *Modification du shader*

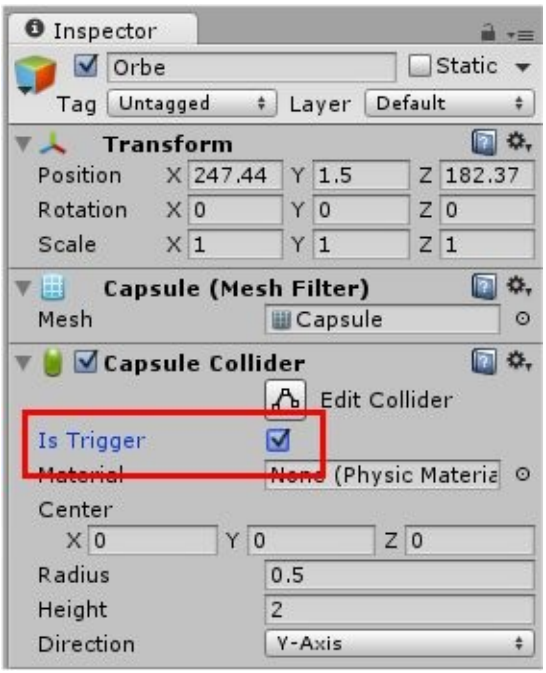


Renommez votre capsule avec F2 et donnez-lui le nom *Orbe*. C’est très important, car le script que nous allons écrire testera si le joueur a touché un objet ayant pour nom *Orbe*.

Toujours sur l’orbe, dans l’inspector, au niveau de l’objet de collision (Collider), cochez la case Is Trigger. Le fait de cocher cette case va permettre au joueur de passer à travers l’orbe.

Note *> Un collider est un composant permettant de rendre un objet solide pour empêcher le joueur de le traverser. Si l’option is trigger est activée, l’objet ne sera plus solide mais les collisions seront toujours détectées, cela permet par exemple de ramasser une pièce lorsque le joueur la touche.*

Figure 6.3 : *Modification du collider*



Vous pouvez créer un prefab de l’objet Orbe car nous allons le réutiliser plusieurs fois. Vous trouverez le mien dans les fichiers source du livre.

2. Création du script de collision

Nous allons maintenant créer le script de collision permettant de ramasser les orbes. Créez un nouveau script C# et appelez-le `Collision`. Ouvrez ce script afin de coder la fonctionnalité de ramassage. Supprimez les fonctions `Start` et `Update`. Créez une variable `nbOrbes` initialisée à 0. Cela va nous permettre de savoir combien d'orbes nous avons.

```
int nbOrbes = 0;
```

Pour détecter si le personnage a touché une orbe, nous utilisons la fonction `OnTriggerEnter`. Nous l'intégrons à notre script de la façon suivante :

```
void OnTriggerEnter(Collider obj)
{

}
```

`obj` représente l'objet touché. Vous pouvez créer une condition pour vérifier qu'il s'agit bien d'un objet Orbe ; son nom doit alors être égal à la chaîne `"Orbe"`. Pour récupérer le nom du GameObject vous devez écrire `obj.gameObject.name` (récupérer le nom du GameObject de `obj`). Cela donne :

```
void OnTriggerEnter(Collider obj)
{
	if(obj.gameObject.name == "Orbe")
	{
		// Si on touche une orbe…
	}
}
```

Consultez la documentation pour en savoir plus sur les fonctions de collision. Maintenant vous pouvez décrire l'événement qui doit se produire. Dans notre cas, nous voulons simplement faire disparaître l'orbe (fonction `Destroy`) et ajouter 1 au nombre d'orbes que le joueur possède. Voici le code complet :

```
using UnityEngine;
using System.Collections;

public class Collision : MonoBehaviour {

	int nbOrbes = 0;

	void OnTriggerEnter(Collider obj)
	{
		if(obj.gameObject.name == "Orbe")
		{
			nbOrbes += 1;
			Destroy(obj.gameObject);
		}
	}
}
```

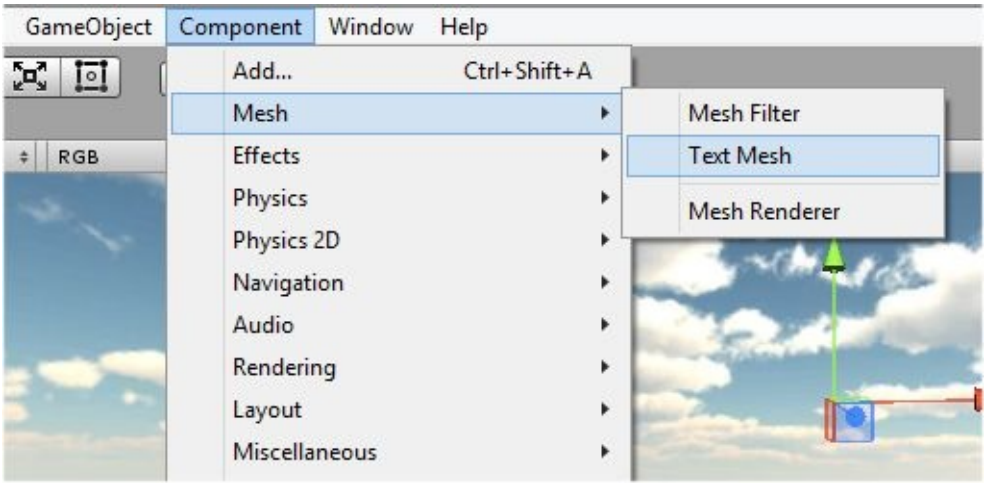
Pour tester le script, vous devez l’enregistrer et le placer sur votre First Person Controller. Vous pouvez ensuite lancer le jeu et essayer de rentrer en collision avec une orbe. L’orbe touchée disparaît.

3. Affichage du nombre d’orbes

Il serait intéressant d’afficher en direct le nombre d’orbes du joueur à l’écran. Pour cela nous allons utiliser un texte 3D accroché à la caméra du joueur afin que le texte soit toujours dans le champ de vision du joueur. Pour créer une zone de texte, nous commençons par créer un objet vide à l’aide du menu GameObject/Create Empty, que nous appellons Texte3D puis nous lui ajoutons un élément Texte 3D avec Component/Mesh/Text Mesh.

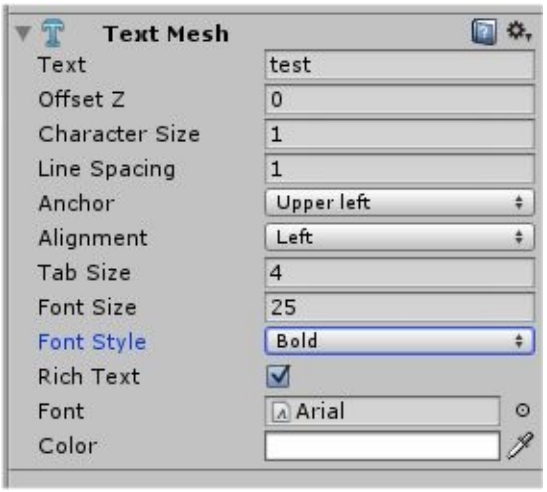
***Note** > Vous pouvez également, depuis la version 5 d’Unity, créer directement un Texte 3D en cliquant sur GameObject/3D Object/3D Text.*

***Figure 6.4 :** Création d’un Text Mesh*



Vous devez spécifier une taille (font size) une couleur (color) et une police (font) à votre texte. Voici mes réglages :

***Figure 6.5 :** Configuration du texte*



Nous allons maintenant placer le texte dans la caméra pour qu’il suive les mouvements de celle-ci. Il sera ainsi toujours dans le champ de vision du joueur. Supprimez la Main Camera de votre hiérarchie pour ne conserver que la caméra de votre personnage car l’autre caméra ne nous sera d’aucune utilité. Placez ensuite le texte 3D que vous venez de

créer dans l’objet Main Camera de votre First Person Controller.

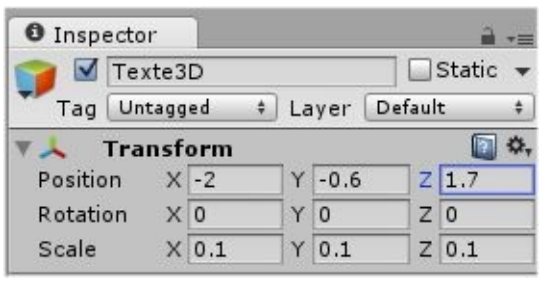
Figure 6.6 : *Association du texte à la caméra*

Modifiez sa taille, sa rotation et sa position afin de le placer devant la caméra à l’endroit que vous souhaitez. Pour vous aider, écrivez “test” dans la case Text du texte. Par exemple, sur la Figure 6.7, j’ai placé le texte en bas à gauche de mon écran.

Figure 6.7 : *Positionnement du texte*

Pour cela j’ai configuré l’objet Texte3D de la façon suivante :

Figure 6.8 : *Réglage du composant Transform*



Si vous lancez le jeu, vous constaterez que le texte reste toujours visible à l’écran car il est placé devant la caméra. Nous avons donc un texte que nous allons pouvoir utiliser pour

afficher le nombre d'orbes du joueur. Commencez par créer un nouveau script que vous appellerez `OrbeScript`. Comme précédemment, supprimez la fonction `Start`. Pour afficher le nombre d'orbes ramassées, nous allons utiliser la variable `nbOrbes` que nous avons créée dans le script `Collision`. Pour accéder à une variable se trouvant dans un autre script, il faut l'avoir déclarée comme *public static*. Pour cela, il suffit d'écrire `public static` devant sa déclaration. Dans notre cas, le script `Collision` doit donc avoir une variable déclarée comme ceci :

```
public static int nbOrbes = 0;
```

Pour accéder à une variable *static* d'un autre script, vous devez écrire `NomDuScript.NomDeVariable` ; dans notre cas, on écrira donc `Collision.nbOrbes` pour appeler la variable `nbOrbes`.

Revenons à notre script `OrbeScript`. Créez une nouvelle variable publique de type TextMesh :

```
public TextMesh texte;
```

Cette variable peut contenir un TextMesh et le fait de la mettre en public nous permet d'y accéder via l'inspector et de pouvoir faire glisser notre texte dans la variable (voir Figure 6.9). Dans la fonction `Update`, ajoutez l'instruction suivante :

```
texte.text = Collision.nbOrbes + " orbes";
```

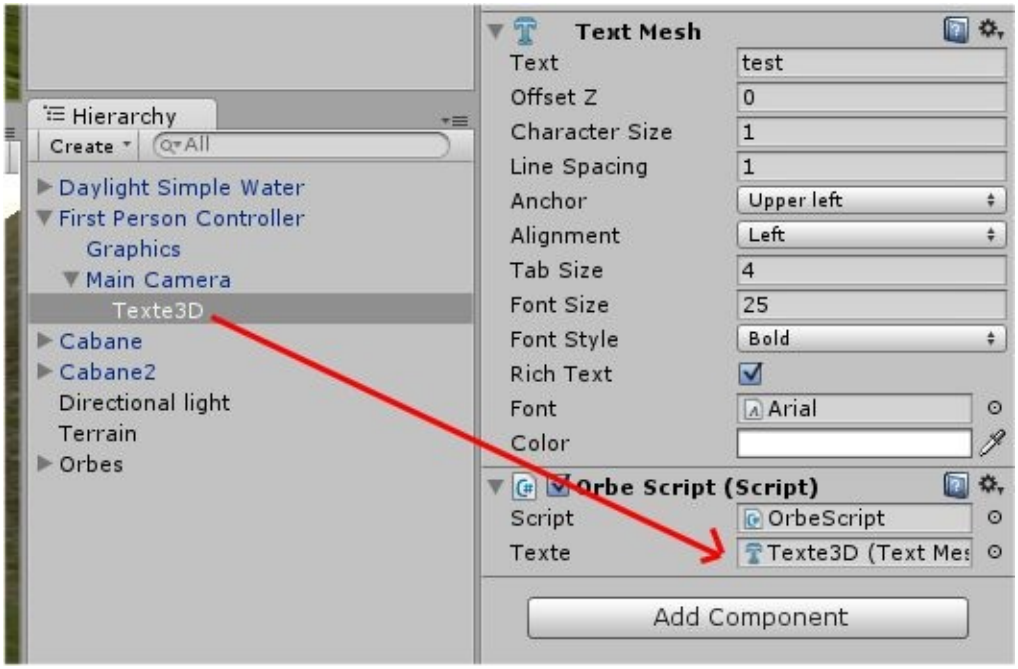
L'instruction `texte.text` permet d'accéder au texte de la variable appelée `texte` et de le modifier. Nous lui attribuons la valeur de la variable `nbOrbes` suivie de la chaîne de caractères `" orbes"`. Si le joueur possède trois orbes, il sera affiché "3 orbes" à l'écran. Ce qui nous donne le script suivant :

```
using UnityEngine;
using System.Collections;

public class OrbeScript : MonoBehaviour {

        public TextMesh texte;

        // Update is called once per frame
        void Update () {
                texte.text = Collision.nbOrbes + " orbes";
        }
}
```

La fonction `Update` permet de rafraîchir le texte et donc de toujours avoir la dernière valeur de `nbOrbes`.

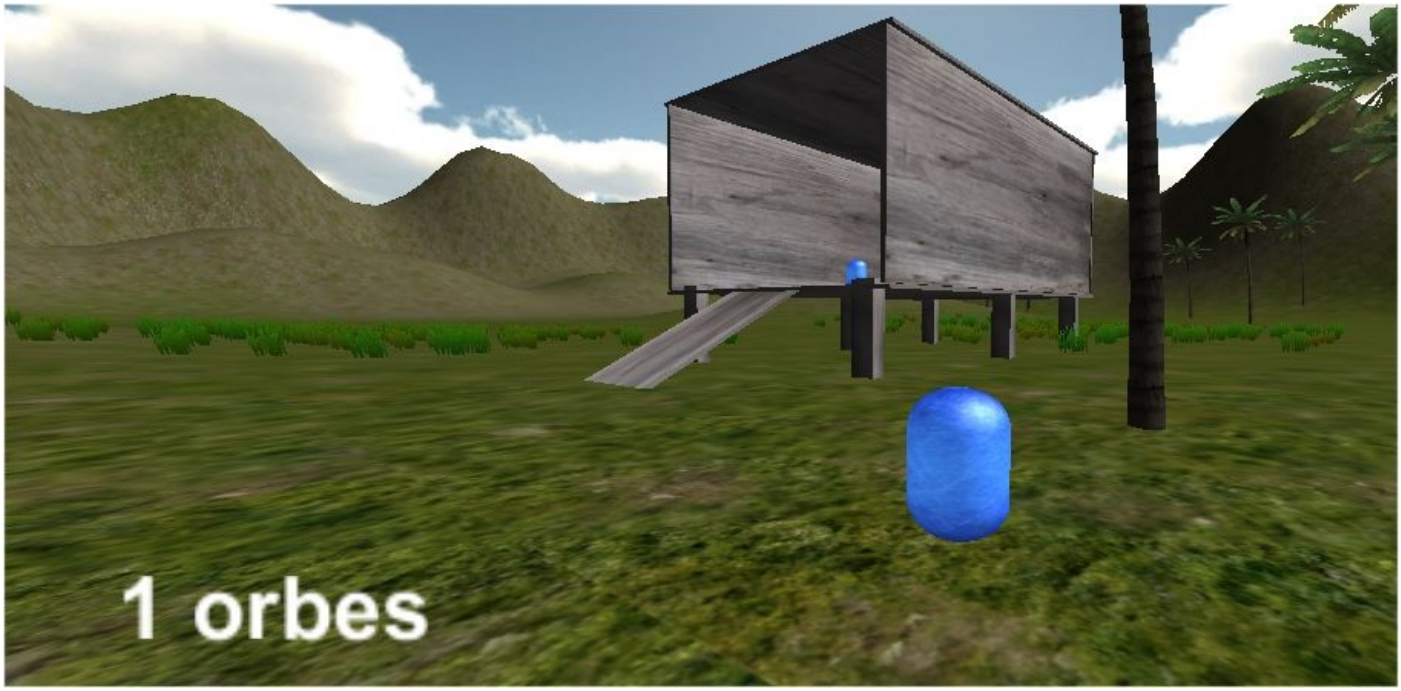
Enregistrez le script et placez-le sur votre objet texte (Texte3D). Ensuite, faites glisser celui-ci dans la variable `texte` du script dans l'inspector :

Figure 6.9 : *Association de l'objet texte avec la variable texte*



Cela nous permet d'associer le texte 3D à la variable `texte` du script. Ainsi c'est le texte 3D qui sera modifié par le script. Vous pouvez maintenant lancer le jeu et ramasser les orbes. Vous constaterez que le nombre d'orbes s'affiche bien à l'écran (voir Figure 6.10).

Figure 6.10 : *Affichage du nombre d'orbes*



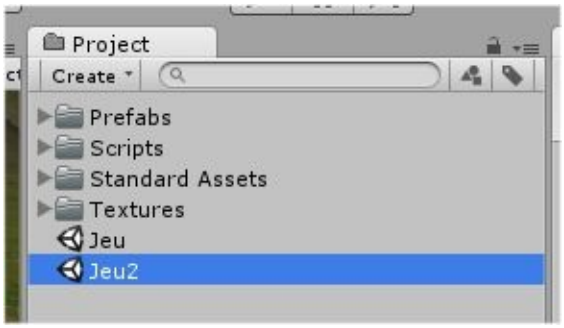
4. Changement de niveau

Votre script est désormais fonctionnel et le résultat est directement visible dans le jeu. Le joueur peut savoir combien d'orbes il a ramassées et s'il a atteint l'objectif du niveau (ramasser cinq orbes). Si l'objectif du niveau est atteint, vous pouvez charger le niveau suivant avec

```
Application.LoadLevel("NomDuNiveau");
```

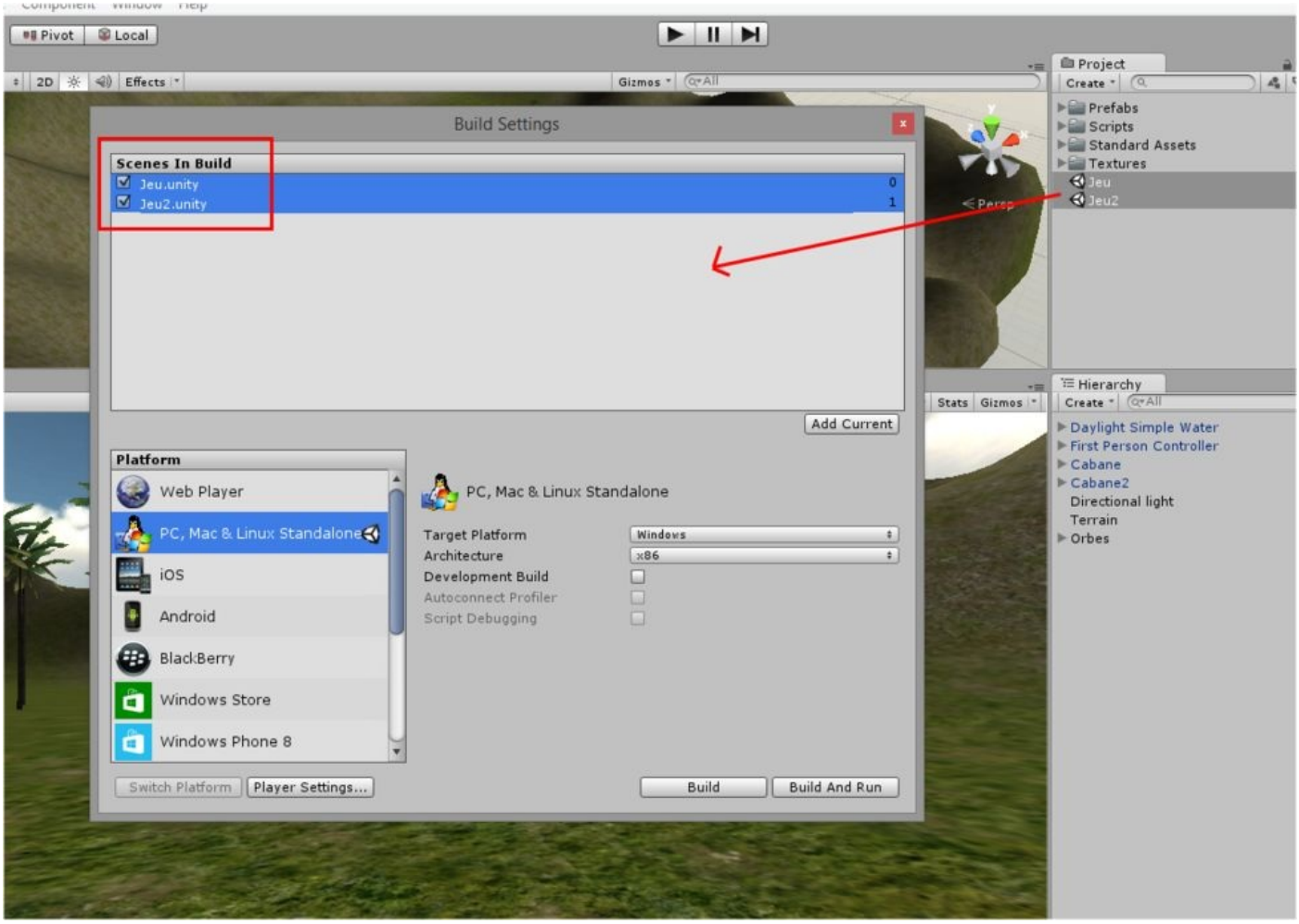
Pour pouvoir changer de scène (changer de monde ou de niveau), vous devez créer plusieurs scènes et les enregistrer. Dans notre exemple, la scène s'appelle `Jeu`. Vous pouvez la dupliquer et la renommer en `Jeu2` ou `Niveau2` afin de pouvoir tester si le changement de scène fonctionne. Pensez à modifier le second niveau pour pouvoir les distinguer.

Figure 6.11 : *Création d'une seconde scène*



Pour pouvoir charger une scène, vous devez l'avoir déclarée dans la liste des scènes composant votre jeu. Pour indiquer à Unity quelles sont vos scènes, cliquez sur File/Build Settings. Une nouvelle fenêtre s'ouvre et vous pouvez faire glisser vos scènes à l'intérieur de cette fenêtre :

Figure 6.12 : *Ajout des scènes*



Vous pouvez maintenant charger une scène se trouvant dans la liste. Par exemple, si le joueur a ramassé cinq orbes, nous chargeons la scène `Jeu2` comme ceci :

```
if(nbOrbes == 5){
        Application.LoadLevel("NomDuNiveau");
}
```

Nous ajoutons cette nouvelle condition à notre script `Collision`, ce qui donne :

```
using UnityEngine;
using System.Collections;

public class Collision : MonoBehaviour {

        public static int nbOrbes = 0;

        void OnTriggerEnter(Collider obj)
        {
                if(obj.gameObject.name == "Orbe")
```

❶ { nbOrbes += 1; Destroy(obj.gameObject); **if**(nbOrbes == 5){ ❷ Application.LoadLevel(***"Jeu2"***); ❸ } } } }

Lorsque le joueur ramasse l'orbe ❶, le script teste si le joueur en possède cinq ❷. Si tel est le cas, on passe au niveau suivant ❸. Testez le jeu pour voir si le changement de scène fonctionne.

***Note** > Je vous invite à consulter l'ensemble des fonctions de "Application" sur la documentation. Vous y trouverez des fonctions générales du type Charger une scène, Quitter le jeu, Récupérer le nom de la scène en cours, etc.*

Vous êtes maintenant en mesure de créer un objectif, de détecter lorsque l'objectif est atteint et vous savez comment changer de niveau. Dans le prochain chapitre, nous allons voir comment créer des effets spéciaux grâce à l'instanciation d'objets.

Dans ce chapitre, vous avez appris à créer un script de collision permettant le ramassage d'objets. Vous êtes capable de savoir en temps réel combien d'objets ont été ramassés et vous pouvez changer de scène si l'objectif fixé est atteint. Pour réaliser ces scripts nous avons utilisé :

- la fonction `OnTriggerEnter` qui gère la collision ;
- une variable `public static` pour y accéder à partir d'un autre script ;
- un TextMesh pour afficher un texte à l'écran ;
- la fonction `LoadLevel` pour changer de niveau.

7

Créer des effets spéciaux

1. Instanciation d’éléments

Un jeu vidéo doit être vivant et animé afin de toujours capter l’attention du joueur. Si l’environnement est trop statique, celui-ci risque de se lasser et son intérêt pour votre jeu va diminuer. De plus, vous devez impérativement faire comprendre au joueur que l’action qu’il a réalisée a fonctionné. Par exemple, lorsqu’il tire sur un ennemi, l’ennemi doit clignoter afin de bien montrer qu’il a subi des dégâts. Nous allons donc ajouter ce que l’on appelle des effets spéciaux.

Lorsque le joueur ramasse une capsule, celle-ci disparaît. Vous comprenez que vous avez ramassé la capsule, mais ça n’est peut-être pas clair pour tout le monde. Je vous propose donc de recourir à un système de particules pour créer une animation chaque fois que le joueur prend une capsule. Pour cela, vous devez *instancier* un système de particules au moment de la collision et à l’endroit de l’impact. Vous pouvez utiliser ce type d’effet dans n’importe quelle situation. Par exemple, si vous utilisez un arc pour tirer une flèche, vous pouvez placer un système de particules à l’endroit de l’impact pour créer un effet visuel.

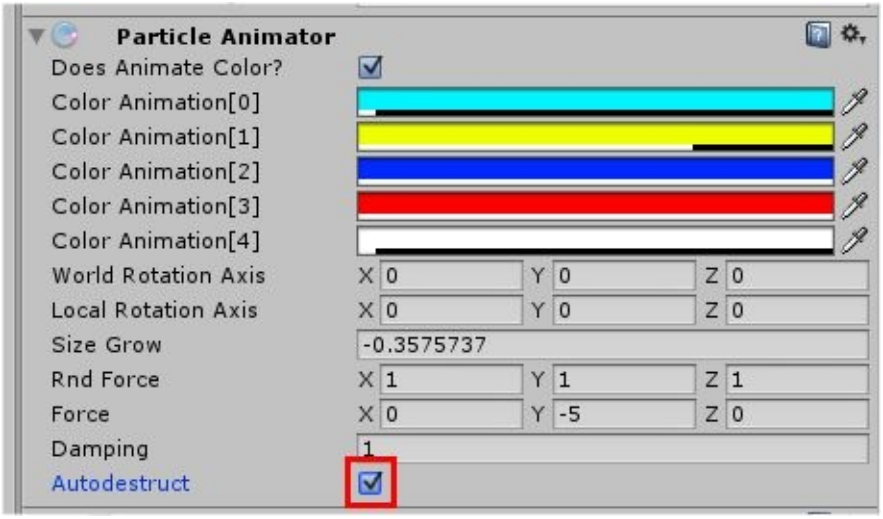
Afin de faire apparaître des objets ou des particules sur notre scène, nous allons utiliser la fonction `Instantiate`. Cette fonction permet de faire apparaître un nouvel objet en cours de jeu. Vous pouvez préciser quel objet doit être instancié et où il doit être créé. Cela vous permet de ne créer des objets que lorsque vous en avez besoin.

Pour ma part, je vais utiliser le système `Fireworks`, qui se trouve dans le package `particles` que nous avons importé lors de la création du projet (voir Figure 7.1), mais vous êtes libre de prendre le système de votre choix.

Figure 7.1 : *Système de particules Fireworks*

Par défaut, les particules se lancent en boucle. Pour ne jouer l’animation qu’une seule fois, vous devez modifier ce comportement dans ses propriétés. Cliquez sur le système pour afficher l’inspector et dans celui-ci cochez la case Autodestruct :

Figure 7.2 : *Configuration des particules*



Le système de particules est prêt à être instancié ! Nous pouvons passer à la partie code pour programmer l'événement souhaité. Nous allons ouvrir le script `Collision` qui gère le ramassage de la capsule et y ajouter notre effet. Dans la partie `OnTriggerEnter` du code, nous utiliserons la fonction `Instantiate` pour créer le système de particules pendant que le jeu est en train de tourner. Cette fonction prend trois paramètres :

- un GameObject : l'objet à instancier ;
- un vector3 : la position d'instanciation ;
- Quaternion : la rotation de l'objet instancié.

La position d'instanciation est en général égale à la position de l'objet ramassé ou à la position d'un autre objet de jeu avec lequel il y a eu une interaction, mais vous pouvez définir n'importe quelle position en créant votre propre `vector3` (`new Vector3(10, 10, 10)`) par exemple.

Voici comment procéder : vous créez une variable de type GameObject qui contiendra le système de particules ❶. Vous utilisez la fonction `instantiate` ❷. Cela permet d'instancier le système à la position de l'objet touché et dans la rotation d'origine. Vous devriez avoir le code suivant :

```
using UnityEngine;
using System.Collections;

public class Collision : MonoBehaviour {

        public static int nbOrbes = 0;
        public GameObject particule; // Le système de particules qui sera
instancié
```

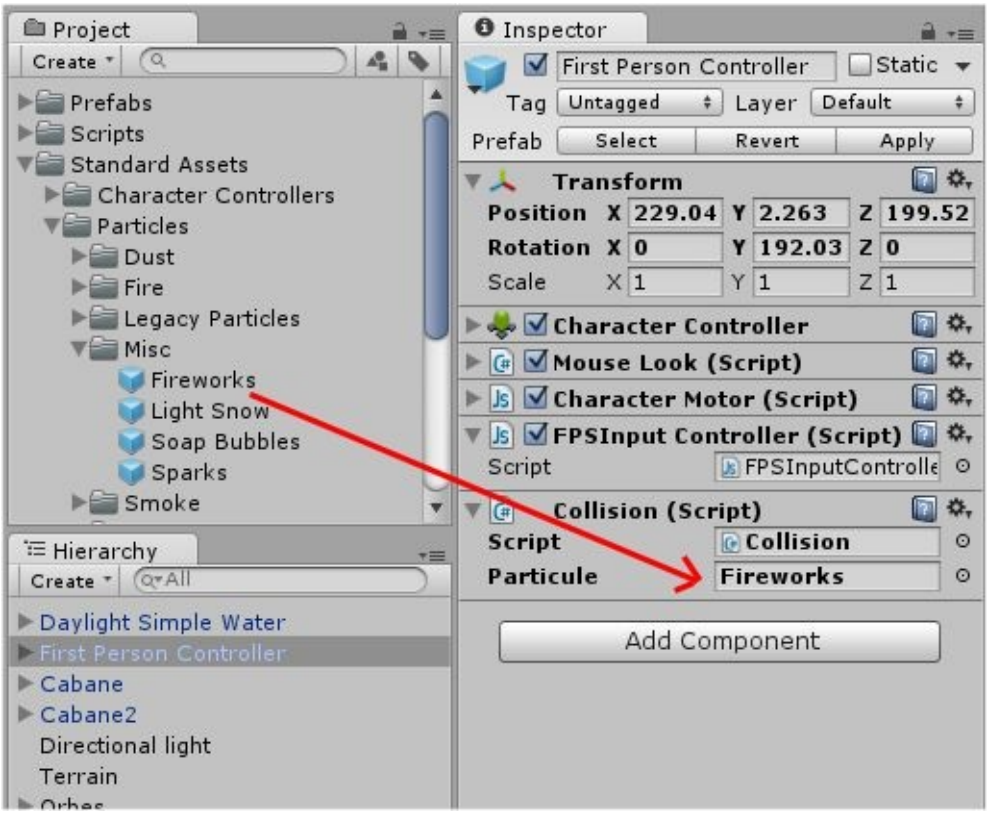
❶ **void** OnTriggerEnter(Collider obj) { **if**(obj.gameObject.name == ***"Orbe"***) { Instantiate(particule, obj.transform.position, Quaternion.identity); ❷ nbOrbes += 1; Destroy(obj.gameObject); **if**(nbOrbes == 5){ Application.LoadLevel(***"Jeu2"***); } } } }

Comme vous pouvez le voir, j'ai utilisé la position de l'objet touché (`obj.transform.position`) pour définir la position d'instanciation et j'ai utilisé

`Quaternion.identity` pour indiquer que l'objet doit être instancié avec sa rotation par défaut.

Lorsque vous procédez à une instanciation, pensez bien à instancier les particules avant de détruire l'objet touché car nous récupérons la position de l'objet touché. Pensez également à relier le système de particules choisi à la variable que nous avons créée dans le script `Collision`. Pour cela, cliquez sur le personnage qui porte le script de collision pour afficher ses propriétés dans l'inspector et faites glisser le système de particules dans la variable :

Figure 7.3 : *Attribution d'une valeur à la variable*



Désormais, le script va créer un système de particules à chaque fois que le joueur touchera une capsule. La puissance du script est de pouvoir être très facilement modifiable. Si vous souhaitez changer de particules, vous n'aurez qu'à faire glisser un autre système de particules dans la variable.

Vous pouvez instancier des objets dans de nombreux cas : pour faire apparaître des ennemis, créer un nouveau monde pendant le jeu par exemple lors d'un changement de zone, faire apparaître des objets ou des pièces lorsque le joueur élimine un ennemi, etc.

Encadré : Utilisation de la fonction instantiate

Vous pouvez utiliser `instantiate` dans de nombreux cas. Vous pouvez vous en servir pour faire apparaître des monstres ou du décor dans votre monde 3D. Cela permet de ne faire apparaître certains éléments du décor que quand cela est nécessaire et optimiser vos scènes. Par exemple, un jeu comme *Flappy bird* propose un monde infini. Bien évidemment, ce monde n'est pas réellement infini, il est simplement généré en continu.

2. Lancement d’un son

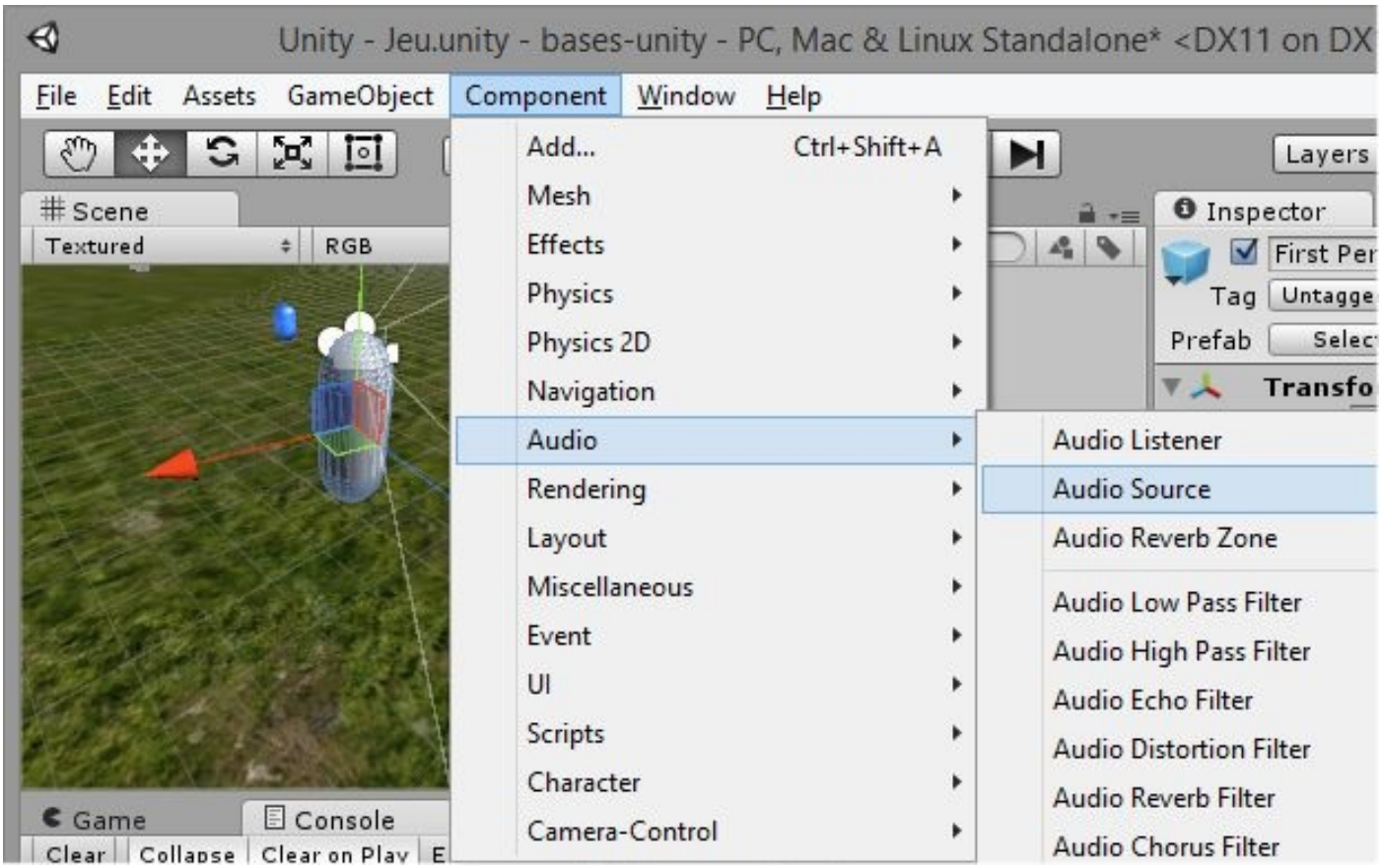
Un jeu vidéo contient des effets visuels mais aussi des effets de son. Le son est aussi important que le visuel. Il peut également signaler la réalisation d’une action. Par exemple, un son peut être émis chaque fois que le joueur ramasse une capsule, ainsi non seulement il y aura des particules mais aussi un bruit indiquant qu’une nouvelle capsule a été ramassée. Nous allons voir dans cette section comment ajouter du son.

Pour commencer, vous devez importer des sons dans Unity. Différents formats sont supportés : MP3, WAV, OGG, FLAC. Pour ma part, j’utilise souvent le site freesound qui propose énormément de sons gratuits. Les clips audio sont classés par catégories et vous pouvez effectuer une recherche, par exemple “pickup”, pour trouver un son de ramassage d’objets. Je vous conseille de prendre un son court de moins de 1 seconde. Pour télécharger un son, vous devez créer un compte sur le site. Importez ensuite votre son dans Unity en le glissant/déposant de votre ordinateur à la fenêtre du logiciel.

Attention *> Respectez les droits d’auteur des sons que vous utilisez.*

Pour pouvoir émettre un son, les objets doivent disposer d’un composant Audio Source. Vous allez donc en ajouter un au personnage joueur. Pour cela, cliquez sur le personnage et sélectionnez le menu Component/Audio/Audio Source.

Figure 7.4 : *Ajout d’un Audio source*



Le personnage peut maintenant produire un son. Pour pouvoir entendre un son, vous devez avoir un objet Audio listener dans votre scène. L’Audio listener correspond aux oreilles de

votre joueur, si vous n'en avez pas, vous n'entendez pas. Normalement vous avez un Audio listener automatiquement sur la Main Camera de votre First Person Controller. L'important est de s'assurer que la caméra puisse entendre vos sons. Les sons doivent être lancés à proximité de la caméra et non au fin fond de la scène.

Figure 7.5 : *Audio listener*

Passons maintenant à la partie code pour voir comment produire un son lors de la collision. Toujours dans le script `Collision`, nous allons ajouter une variable son de type AudioClip ❶. C'est dans cette variable que vous ferez glisser votre son. Pour jouer le son, vous devez utiliser la fonction `PlayOneShot` ❷. Ce qui nous donne le code suivant :

```
using UnityEngine;
using System.Collections;

public class Collision : MonoBehaviour {

        public static int nbOrbes = 0;
        public GameObject particule;
        public AudioClip son;
```

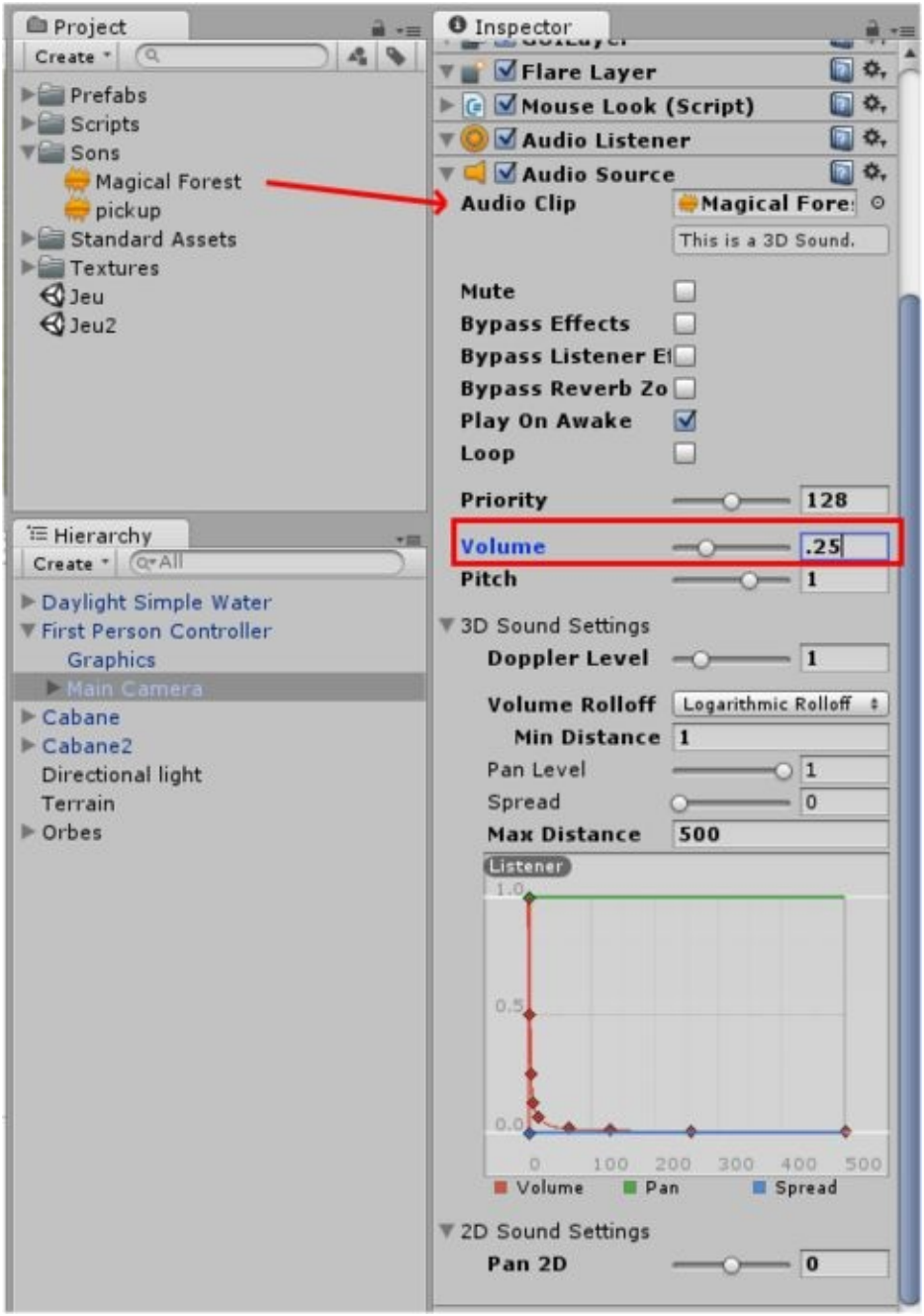
❶ **void** OnTriggerEnter(Collider obj) { **if**(obj.gameObject.name == ***"Orbe"***) { audio.PlayOneShot(son); ❷ Instantiate(particule, obj.transform.position, Quaternion.identity); nbOrbes += 1; Destroy(obj.gameObject); **if**(nbOrbes == 5){ Application.LoadLevel(***"Jeu2"***); } } } }

Vous pouvez jouer un son pour chaque événement (objectif atteint, victoire, défaite…). Vous pouvez également ajouter du bruitage en arrière-plan ou une musique. Dans ce cas, veillez à trouver un son qui boucle (*loop* en anglais) afin de pouvoir le jouer à l'infini.

Note *> Vous pouvez trouver des musiques ou sons d'ambiance sur l'Asset Store ou sur un site web comme opengameart qui propose de nombreuses ressources gratuites et libres de droits.*

Une fois votre musique choisie et importée, vous pouvez la lancer très facilement en la plaçant sur l'Audio source de la caméra se trouvant dans votre personnage (voir Figure 7.6). Chaque objet peut avoir un Audio source. Les musiques sont généralement placées sur la caméra. Cela permet de produire le son à l'endroit où se trouvent les "oreilles".

Figure 7.6 : *Ajout d'une musique*



Un son ajouté à l'Audio source se lancera automatiquement si la case Play On Awake de l'Audio source est cochée. Si vous voulez jouer le son en boucle, vous devez cocher la case Loop. Pensez à diminuer le volume en diminuant la valeur de la variable Volume afin de continuer à entendre les autres sons du jeu. Une musique trop forte masque tous les autres sons, alors utilisez un volume faible (de 0.1 à 0.4).

***Note** > Vous pouvez consulter la documentation pour avoir plus d'informations sur l'Audio source.*

Dans ce chapitre, vous avez découvert comment créer des effets visuels et des effets sonores afin de toujours capter l'attention du joueur. Vous savez donc comment :

- instancier un objet ;
- utiliser un système de particules ;

- jouer un son ;
- jouer une musique.

8

L’intelligence artificielle

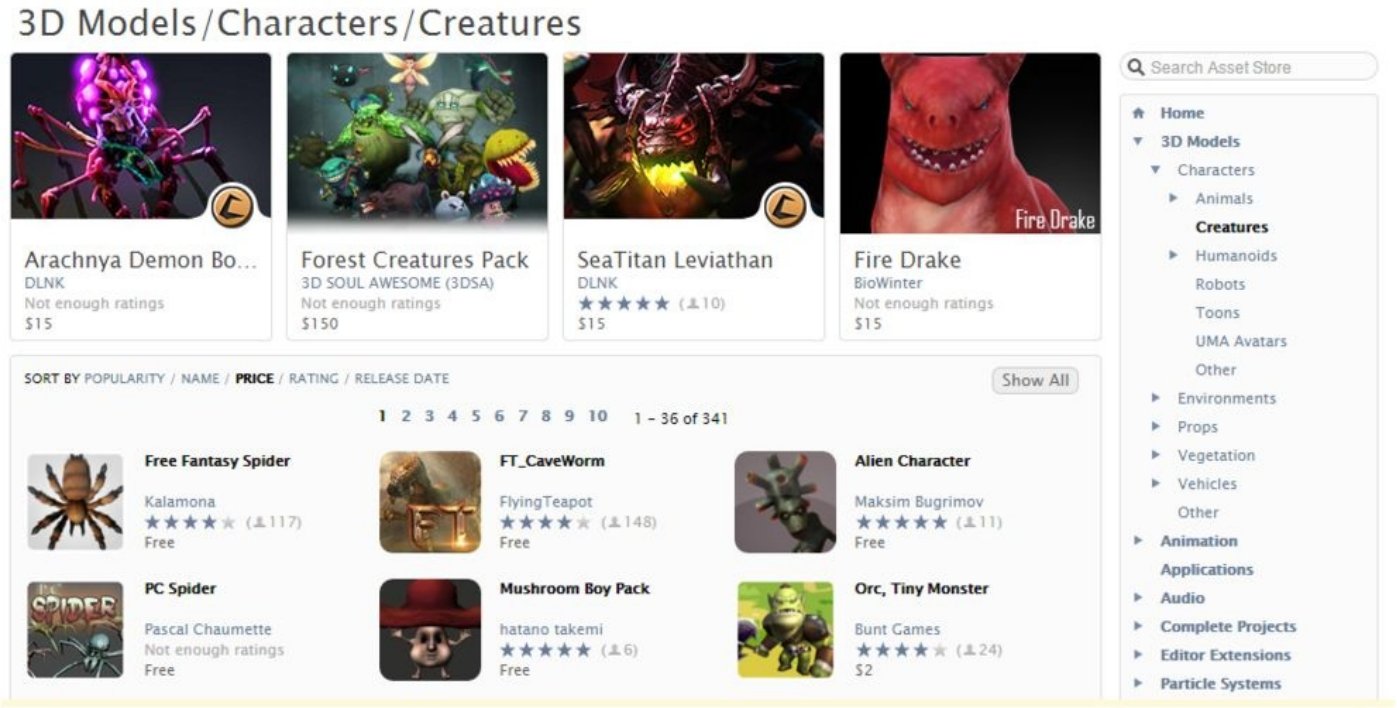
L’intelligence artificielle (ou IA) est présente dans la quasi-totalité des jeux vidéo. Autrefois l’IA était un peu mise à l’écart et les développeurs ne lui accordaient pas une grande importance. Aujourd’hui, elle occupe une place essentielle au point d’être devenue un réel argument de vente. L’intelligence artificielle est l’une des choses les plus complexes à coder dans un jeu vidéo. Il faudrait plusieurs ouvrages entiers pour vous présenter en détail sa programmation. Ce chapitre ne prétend donc pas faire de vous un expert en IA mais vous proposer une première approche afin de partir sur de bonnes bases.

Nous allons développer un algorithme qui va permettre de rendre un personnage non joueur presque autonome. Le but de cet exercice sera de créer une créature qui suivra le joueur si elle le voit. L’algorithme sera assez simple mais très intéressant, puisqu’il introduira les bases du repérage, de l’animation, des mouvements, etc.

1. Préparation de la créature animée

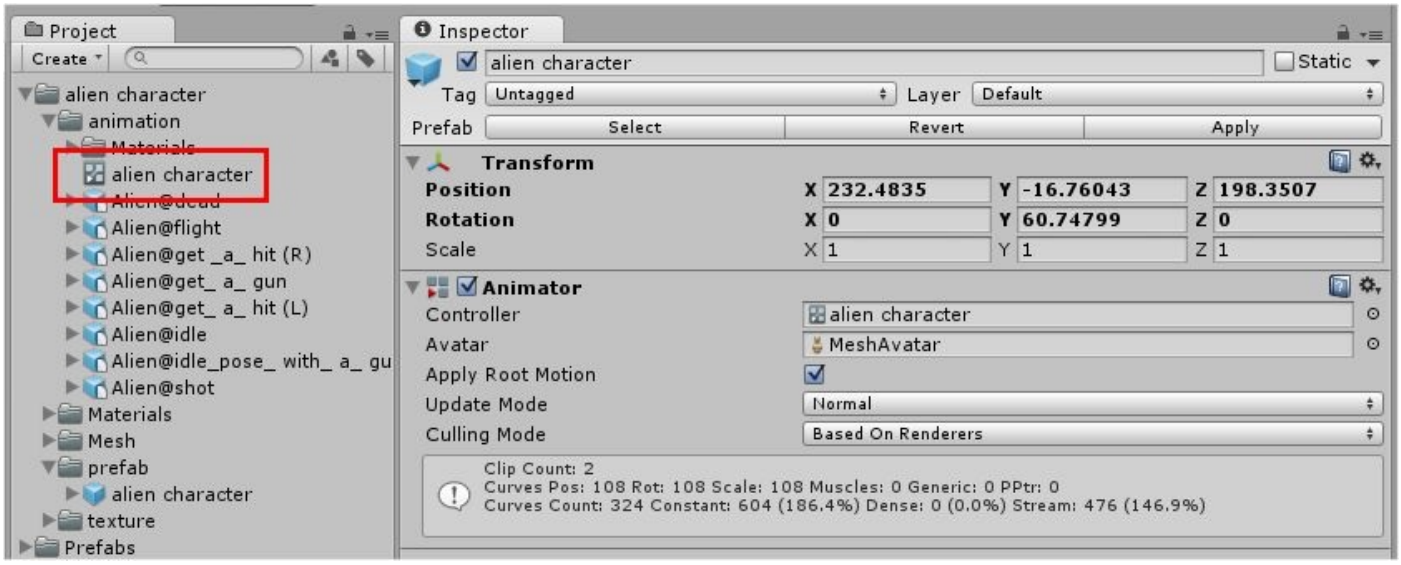
Avant de commencer, vous allez avoir besoin de télécharger une créature 3D animée pour pouvoir coder le script qui lui est associé. Vous pouvez en trouver quelques-unes gratuites sur l'Asset Store. Choisissez celle que vous voulez, un humain, une bête, un robot, cela n'a pas d'importance. L'essentiel est qu'elle soit animée et qu'elle possède les animations `idle` et `walk`. Vous trouverez cette information dans la description du modèle. Dans mon cas, je vais utiliser un alien que j'ai trouvé dans la rubrique 3D Models/Characters/Creatures.

Figure 8.1 : *Personnage non joueur*



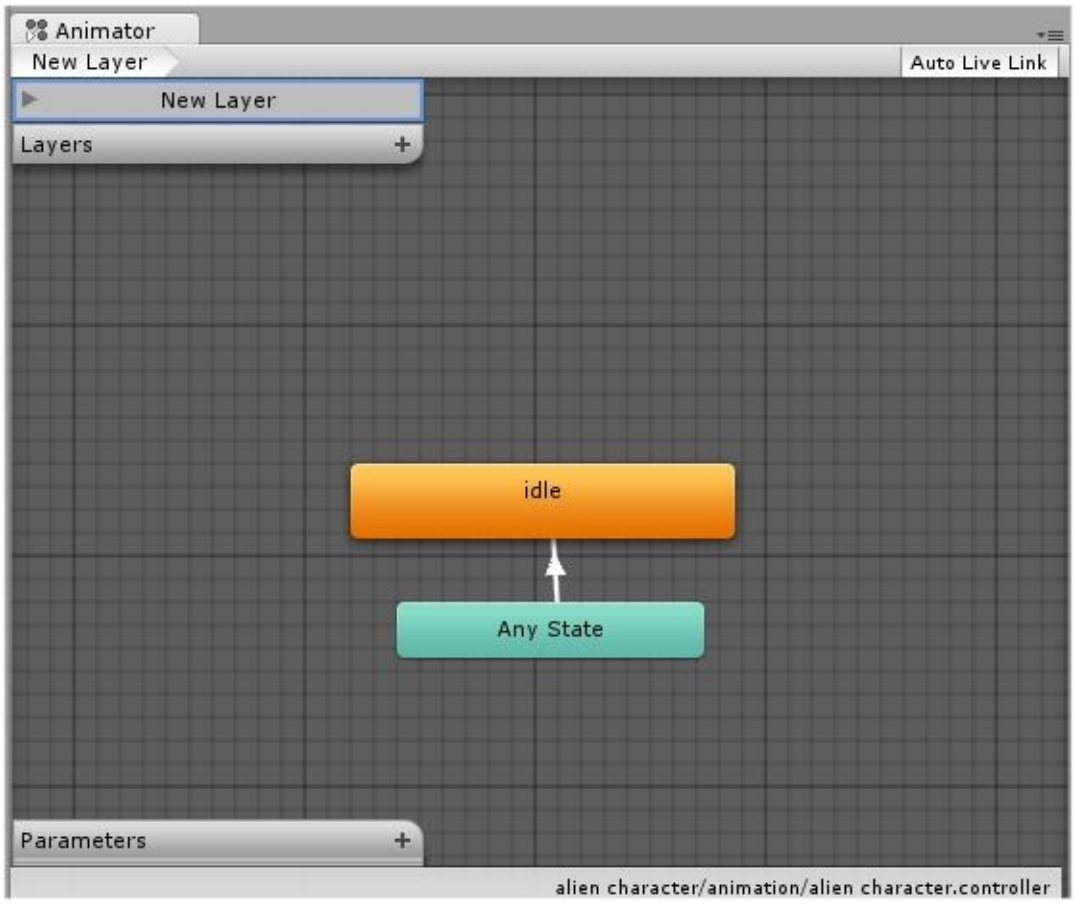
Cliquez simplement sur le bouton download pour importer le modèle dans Unity. Je vous recommande de passer par l'Asset Store pour le téléchargement de modèles 3D car ceux-ci sont déjà configurés. En général, vous y trouverez un prefab prêt à être utilisé, qui contient normalement un composant Animator permettant l'animation du modèle. Nous allons configurer ce composant afin de mettre en place deux animations. Une animation pour la marche et une autre pour lorsque la créature ne bouge pas. Double-cliquez sur le composant Animator de votre modèle (voir encadré sur la Figure 8.2) :

Figure 8.2 : *Modification de l'animator*



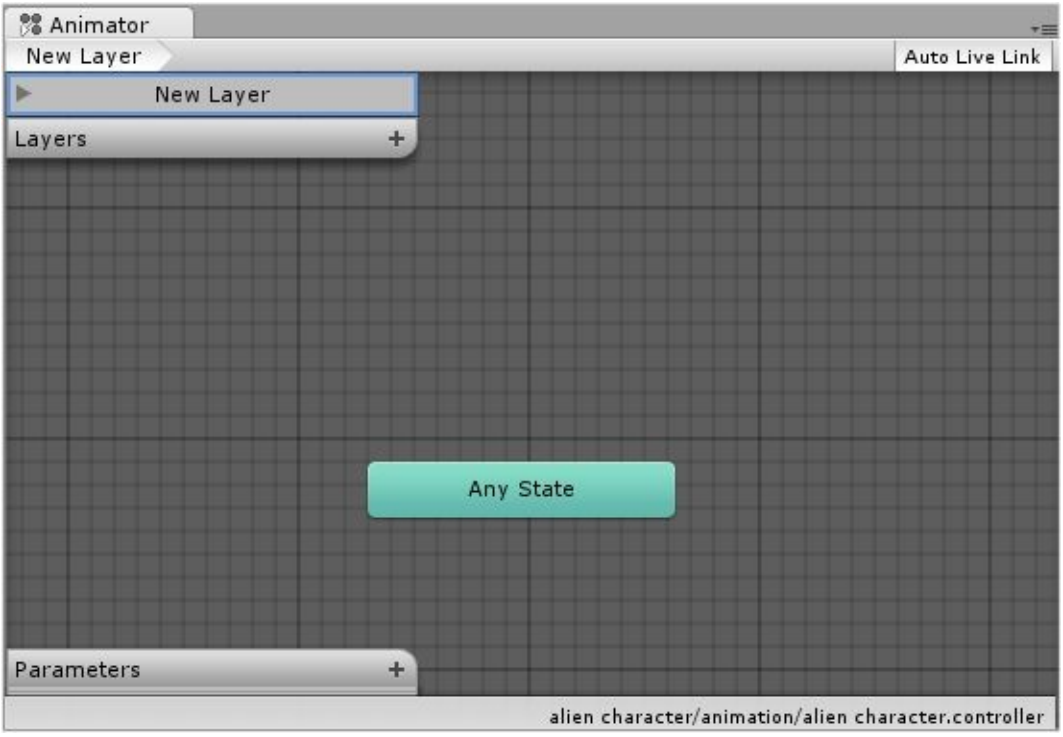
Cela ouvre une nouvelle fenêtre dans laquelle vous allez pouvoir configurer les animations de votre modèle.

Figure 8.3 : *La fenêtre Animator*



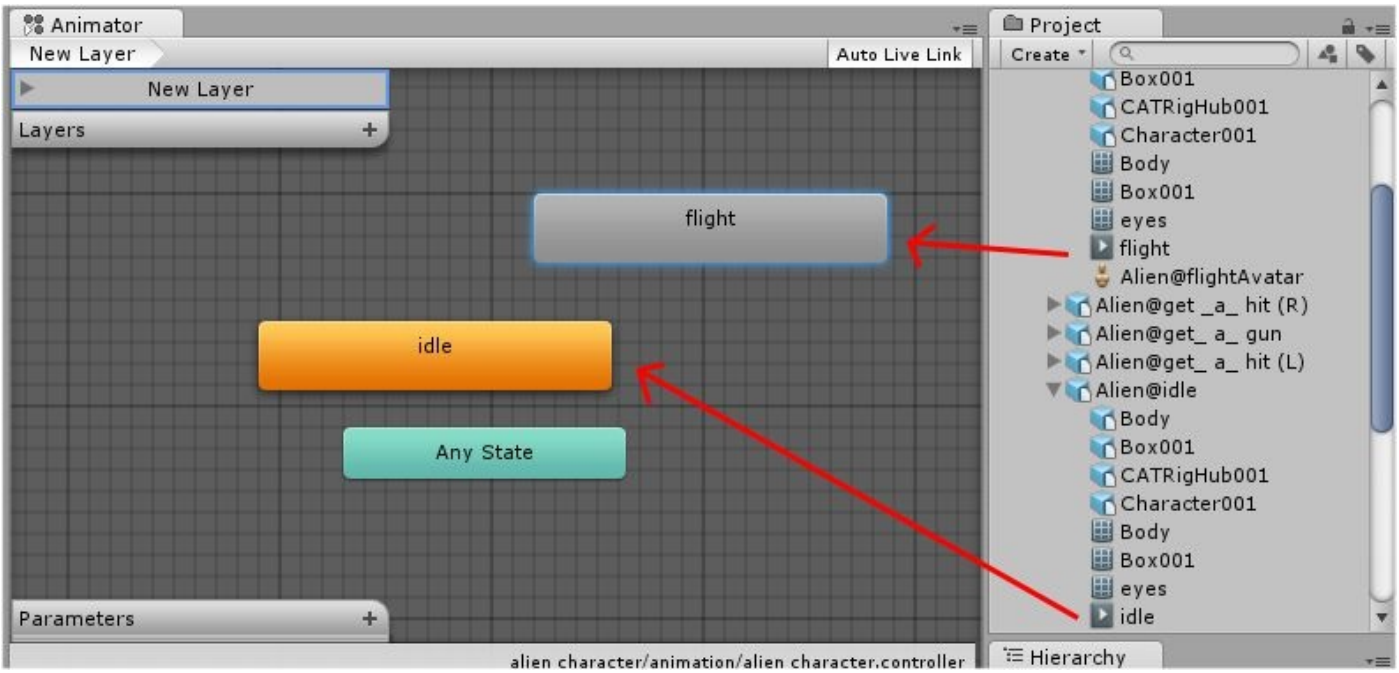
Supprimez toutes les animations (`idle`, `walk`, `jump`…) et tous les paramètres déjà présents. Pour cela, sélectionnez-les d’un clic et appuyez sur la touche Suppr. S’il y a des paramètres en bas à gauche de la fenêtre, supprimez-les également afin d’avoir une fenêtre épurée.

Figure 8.4 : *La fenêtre Animator épurée*



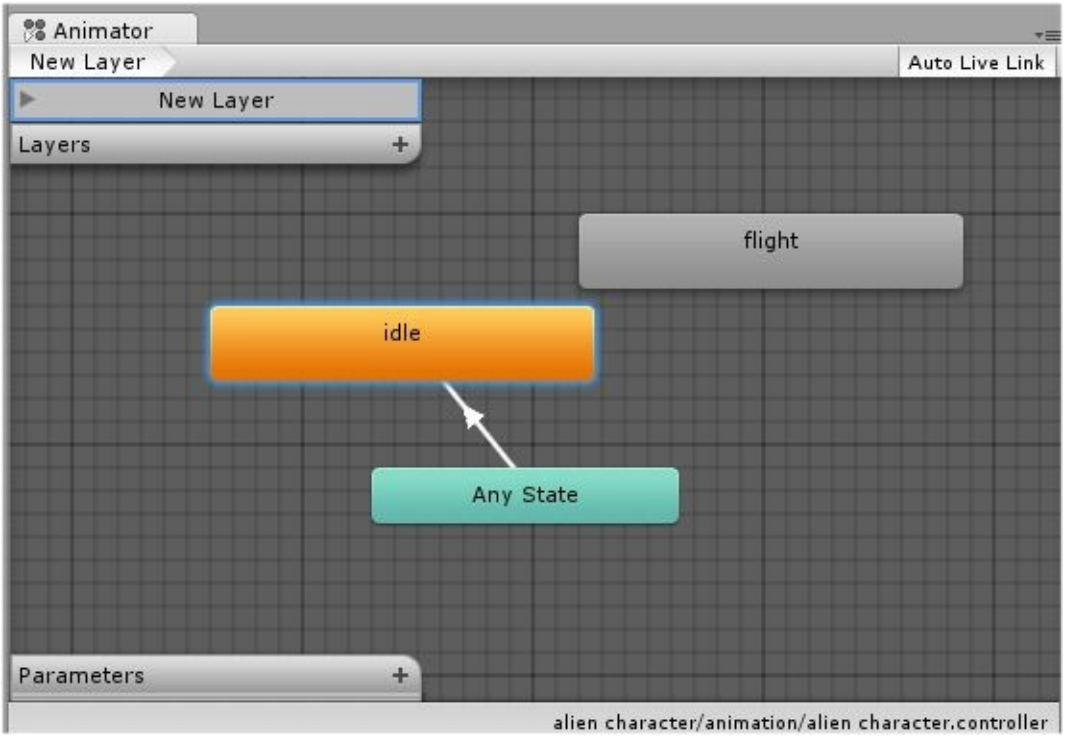
Maintenant vous pouvez ajouter les animations que vous souhaitez utiliser. Celles-ci se trouvent dans les sources du modèle que vous avez téléchargé. Dans notre cas, on choisira une animation `idle` (lorsque le personnage ne bouge pas) et une animation de déplacement (`walk`). Faites-les glisser dans la fenêtre Animator.

Figure 8.5 : *Ajout d'animations dans l'animator*



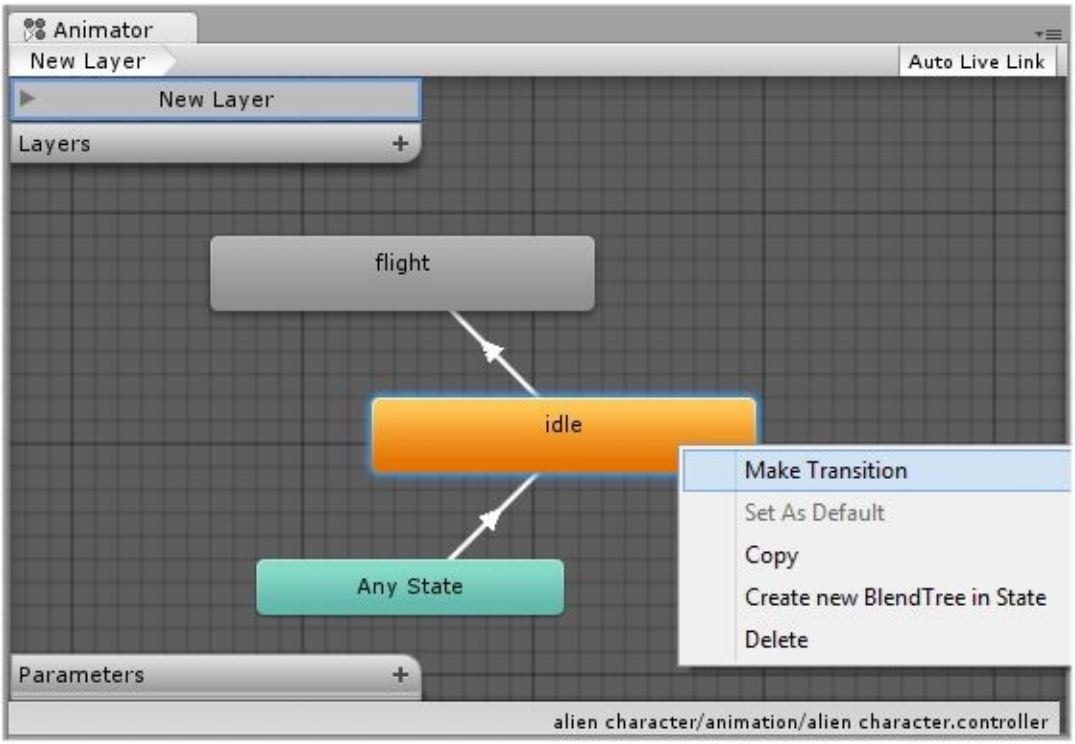
Nous pouvons maintenant créer une transition de Any State vers l'animation idle afin de jouer l'animation par défaut au lancement du jeu. Cliquez droit sur Any State puis sélectionnez Make Transition pour créer une flèche vers idle. Cela signifie qu'au lancement du jeu, l'animation `idle` sera initialisée par défaut (Any State).

Figure 8.6 : *Création d'une transition*



Maintenant votre animation se lancera automatiquement. En effet, l'Animator va s'occuper d'animer votre personnage 3D en utilisant les animations que vous avez déposées dans cette fenêtre. Nous allons créer une transition de idle vers la seconde animation (dans mon cas, il s'agit de flight mais dans le vôtre, il y a des chances qu'elle soit nommée walk). Procédez donc de la même façon :

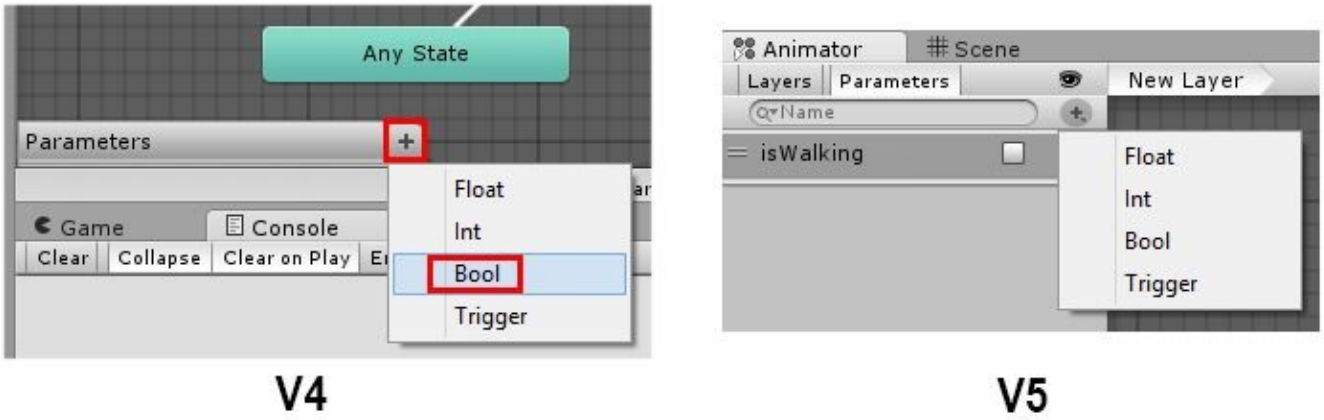
Figure 8.7 : *Création d'une autre transition*



Nous allons créer un paramètre qui sera utilisé pour effectuer le changement d'animation. Ce paramètre sera un booléen qui nous permettra de savoir si le personnage marche ou pas

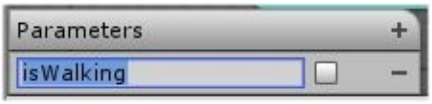
et ainsi d'indiquer à l'Animator que l'animation doit être changée. Cliquez sur le signe plus en haut à gauche de la fenêtre Animator à côté de l'onglet Layers et sélectionnez le type du paramètre dans le menu déroulant qui s'affiche ; dans notre cas, nous choisissons bool.

Figure 8.8 : *Création d'un paramètre*



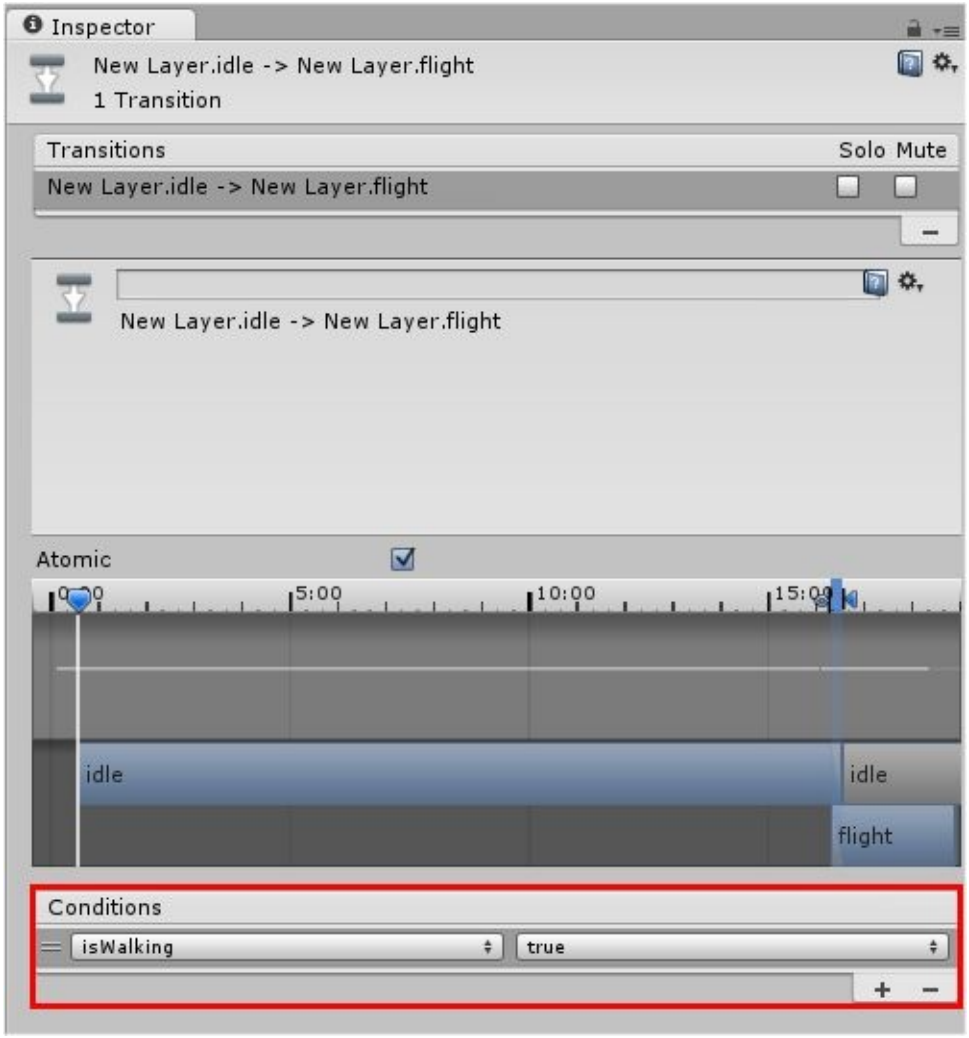
Nommez le paramètre *isWalking*.

Figure 8.9 : *Modification du paramètre*



Enfin, cliquez sur la flèche qui relie l'animation idle à la seconde animation afin de faire apparaître un nouveau menu dans l'inspector. Celui-ci permet de gérer les transitions grâce à des conditions. Dans la section Conditions, vous devriez avoir une condition de type Exit Time. Nous n'allons pas utiliser cette valeur par défaut mais plutôt notre paramètre `isWalking`. Modifiez cette condition afin de sélectionner la variable isWalking et définissez-la à true.

Figure 8.10 : *Modification de la transition*



Cette configuration permet de jouer la seconde animation si la variable isWalking vaut true. Nous pourrons modifier cette variable à l'aide du script et jouer, selon les circonstances, l'animation de notre choix.

La configuration du composant Animator est terminée. Nous sommes maintenant en mesure de coder le script de la créature.

2. Création du script d'IA

Pour écrire le script, nous allons procéder en trois étapes. Dans un premier temps, nous allons gérer la détection du joueur. Ensuite, nous verrons comment suivre le joueur. Enfin vous verrez comment animer le personnage via votre script.

2.1. Repérage du joueur

Commencez par créer un nouveau script C#, que vous appelerez `IA`. Nous aurons besoin de déclarer plusieurs variables dans ce script. Pour l'instant, déclarez les variables suivantes : `bool joueurVu = false;` (si le joueur est repéré, on passera à true) et `public GameObject joueur;` (l'objet joueur). Ces variables vont nous permettre de savoir si le joueur est repéré ou non.

Afin de rester sur des choses simples, nous allons uniquement tester la distance du joueur par rapport au personnage non joueur (PNJ) sans prendre en compte l'angle de vision. Si le joueur se trouve à moins de 10 mètres du PNJ, le PNJ commencera à suivre le joueur.

Nous allons commencer par associer le personnage joueur à la variable `joueur` que nous avons déclarée. Pour cela, nous utilisons la fonction `Find` de GameObject, qui prend en paramètre le nom de l'objet que vous voulez trouver. Dans la fonction `Start`, vous ajouterez donc `joueur = GameObject.Find("First Person Controller");`. La fonction `Find` vous permet d'associer un objet à une variable par script, sans passer par le glisser/déposer habituel. Cette fonction `Find` est parfois indispensable lorsque vous ne pouvez pas faire de glisser/déposer.

Nous allons maintenant créer une nouvelle fonction que nous appellerons `getDistance` et qui va nous permettre de calculer la distance entre le joueur et le PNJ. Nous utilisons pour cela la fonction `Distance` de Vector3. Voici ce que vous devez écrire :

```
float getDistance(){
        return Vector3.Distance(joueur.transform.position,
transform.position);
}
```

`getDistance` calcule la distance entre la position de `joueur` et la position de l'objet sur lequel le script est attaché et nous renvoie le résultat calculé.

Dans la fonction `Update`, nous pouvons appeler `getDistance` simplement en écrivant `getDistance();`. Nous créons ensuite une condition qui vérifie que le résultat de `getDistance` est inférieur à 5. Si tel est le cas, nous pouvons passer la valeur de `joueurVu` à `true` :

```
void Update () {
        if(getDistance() < 5)
        {
                joueurVu = true;
                print ("Joueur vu !");
```

```
        }
}
```

Vous devriez avoir le code suivant sous les yeux :

```
using UnityEngine;
using System.Collections;

public class IA : MonoBehaviour {

        bool joueurVu = false; // Le joueur est-il repéré ?
        public GameObject joueur; // Le joueur

        // Use this for initialization
        void Start () {
                /*On recherche le joueur avec son nom et la fonction Find*/
                joueur = GameObject.Find("First Person Controller");
        }

        // Update is called once per frame
        void Update () {
                if(getDistance() < 5)
                {
                        joueurVu = true;
                        print ("Joueur vu !");
                }
        }

                return Vector3.Distance(joueur.transform.position,
transform.position);
        float getDistance(){
        }
}
```

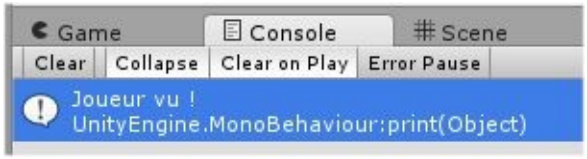
Ce code devrait fonctionner. Pour en être sûr, nous allons le tester.

Glissez-déposez votre créature sur la scène et ajoutez le script `IA` sur la créature également par glisser/déposer :

Figure 8.11 : *Ajout du PNJ*

Lancez le jeu et approchez-vous de la créature. Lorsque vous serez à moins de 5 mètres du PNJ, le message “Joueur vu !” devrait apparaître dans la console. Si tout fonctionne, vous pouvez passer à la gestion du déplacement du PNJ.

Figure 8.12 : *Lorsque le joueur est détecté*



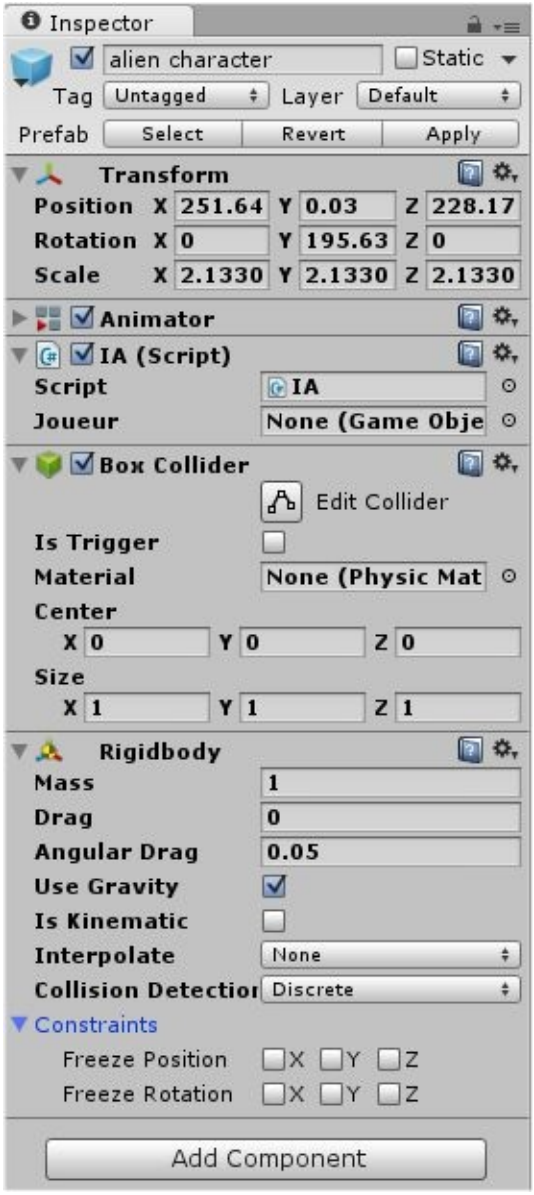
2.2. Programmation du mouvement

Avant de coder la fonction de déplacement, vous devez ajouter deux composants à votre PNJ :

- un Collider : pour gérer la collision du PNG ;
- un Rigidbody : pour que le PNJ soit affecté par la gravité.

Vous pouvez le faire en passant par les menus Component/physics/Rigidbody et Component/physics/Box Collider ou directement depuis l’inspector en cliquant sur Add component. La Figure 8.13 montre ce que doit contenir votre PNJ.

Figure 8.13 : *Paramètres du PNJ*



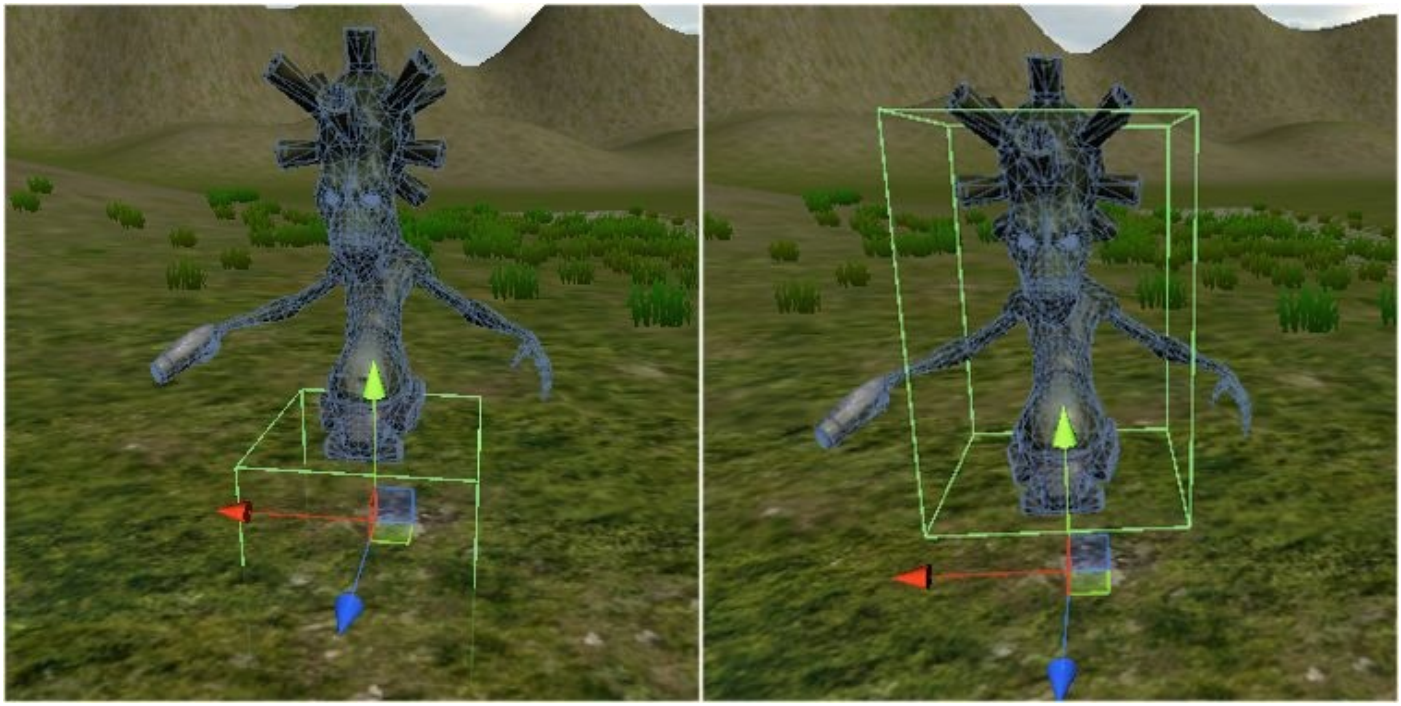
Vous allez devoir configurer le Rigidbody pour bloquer certains axes de rotation afin d'éviter que le personnage ne tombe sur le côté :

Figure 8.14 : *Contraintes sur la rotation*

Le Collider est utilisé pour détecter les collisions entre deux objets. Cela peut permettre de limiter les mouvements du personnage en le bloquant mais il peut aussi être utilisé pour la programmation des interactions comme le ramassage ou l'utilisation d'objets.

Vous devez modifier le Collider afin que le cube de collision soit autour du personnage. Pour cela, modifiez les valeurs de ses variables center et size. Voilà le résultat avant et après la modification des valeurs :

Figure 8.15 : *Modification du Collider*



Vous devez bien placer le Collider pour que le personnage ne passe pas à travers le sol. Testez-le pour vous assurez que le PNJ est affecté par la gravité et qu'il est bien bloqué par le sol grâce à son Collider.

Le Rigidbody est un composant très utilisé. Il permet de gérer la gravité mais aussi d'utiliser de nombreuses fonctions très importantes. Je vous invite à lire attentivement sa documentation. Il existe de nombreuses fonctions permettant d'appliquer des forces (`AddForce`, `AddExplosionForce`, `AddTorque`…). D'autre part, la variable magnitude du Rigidbody peut être utilisée pour programmer le mouvement d'un personnage. Dans notre cas, nous allons utiliser la fonction `MoveTowards` de Vector3 pour effectuer ce mouvement car cette fonction est plus simple à comprendre.

`MoveTowards` prend trois paramètres pour appliquer un mouvement :

- la position de l'objet à déplacer ;
- la position à atteindre ;
- la vitesse de mouvement.

Nous allons ajouter une variable `float vitesseMouvement = 1.0f;` dans notre code. Dans la condition `if(joueurVu)` nous ajoutons la fonction `MoveTowards` :

```
transform.position = Vector3.MoveTowards(
              transform.position,
              joueur.transform.position,
              Time.deltaTime * vitesseMouvement);
```

Cette fonction permet de faire en sorte que le PNJ se déplace en direction du joueur à la vitesse indiquée. `Time.deltaTime` est un bout de code très important qui va vous permettre de cadencer le mouvement afin qu'il soit le même sur tous les PC. En effet, un PC puissant fera tourner votre jeu à 200 ips (images par seconde), un PC peu puissant le

fera tourner à 50 ips. `Time.deltaTime` va contrôler l'application pour qu'elle tourne de la même façon sur tous les ordinateurs indépendamment de la puissance du processeur.

Nous allons utiliser la fonction `LookAt` qui va orienter le PNJ en direction du joueur : `transform.LookAt(joueur.transform.position);`. Voilà le code complet pour cette partie :

```
using UnityEngine;
using System.Collections;

public class IA : MonoBehaviour {

	bool joueurVu = false; // Le joueur est il repéré ?
	public GameObject joueur; // Le joueur
	public float vitesseMouvement = 1.0f; // Vitesse mouvement

	// Use this for initialization
	void Start () {
		/*On recherche le joueur avec son nom et la fonction Find*/
		joueur = GameObject.Find("First Person Controller");
	}

	// Update is called once per frame
	void Update () {
		if(getDistance() < 5) // Si distance < 5
		{
			joueurVu = true; // Joueur détecté
			print ("Joueur vu !");
		}
		if(joueurVu)
		{
			// On se déplace
			transform.position =
Vector3.MoveTowards(transform.position,

joueur.transform.position,

Time.deltaTime * vitesseMouvement);
			// On regarde le joueur
			transform.LookAt(joueur.transform.position);
		}
	}

	float getDistance(){ // Renvoie la distance entre joueur & PNJ
		return Vector3.Distance(joueur.transform.position,
transform.position);
	}
}
```

Vous pouvez tester le jeu et constater qu'une fois que vous êtes repéré, le personnage va vous suivre. Vous avez créé votre premier script d'intelligence artificielle. Le PNJ est capable de détecter le joueur et il peut le suivre. Pour le moment les animations ne changent pas ; nous allons configurer cela dans la prochaine section.

2.3. Gestion des animations

La dernière étape de cet exercice est d'animer le PNJ avec l'animation de marche lorsque celui-ci se déplace. Comme nous avons préparé notre modèle au préalable, nous n'aurons aucune difficulté à l'animer.

Nous allons commencer par connecter notre script à l'Animator qui contrôle les animations. Pour cela, créez une nouvelle variable de type Animator : `private Animator animator;`. Ensuite, dans la fonction `start`, ajoutez la ligne suivante : `animator = GetComponent<Animator>();`. `GetComponent` permet de récupérer un composant de l'objet attaché au script. Ici `GetComponent` nous permet de récupérer l'Animator et de l'associer à notre variable. Pour activer le paramètre `isWalking` de notre animator, nous procédons de la façon suivante : `animator.SetBool("isWalking", true);`. Et le tour est joué ! L'animation de marche se lancera automatiquement.

Vous pouvez bien sûr arrêter le mouvement du PNJ. Par exemple vous pouvez créer une condition testant si la distance entre le joueur et le PNJ est supérieure à 25. Si tel est le cas vous pouvez faire `joueurVu = false` et `animator.SetBool("isWalking", false);`. Ce qui nous donne le code suivant :

```
using UnityEngine;
using System.Collections;

public class IA : MonoBehaviour {

	bool joueurVu = false; // Le joueur est il repéré ?
	public GameObject joueur; // Le joueur
	public float vitesseMouvement = 1.0f; // Vitesse mouvement
	private Animator animator; // Référence à l'animator

	// Use this for initialization
	void Start () {
		/*On recherche le joueur avec son nom et la fonction Find*/
		joueur = GameObject.Find("First Person Controller");
		// Récupération de l'animator
		animator = GetComponent<Animator>();
	}

	// Update is called once per frame
	void Update () {
		if(getDistance() < 5) // Si distance < 5
		{
			joueurVu = true; // Joueur détecté
			print ("Joueur vu !");
			animator.SetBool("isWalking", true);
		}

		if(getDistance() > 25){
			animator.SetBool("isWalking", false);
			joueurVu = false;
			print ("Joueur perdu de vue !");
		}
```

```
                if(joueurVu)
                {
                        // On se déplace
                        transform.position =
Vector3.MoveTowards(transform.position,

joueur.transform.position,

Time.deltaTime * vitesseMouvement);
                        // On regarde le joueur
                        transform.LookAt(joueur.transform.position);
                }
        }

        float getDistance(){ // Renvoie la distance entre joueur & PNJ
                return Vector3.Distance(joueur.transform.position,
transform.position);
        }
}
```

Lorsque le joueur est détecté, le paramètre `isWalking` s'active et l'animation se lance. Si vous partez en courant et que vous vous trouver à plus de 25 mètres du PNJ, il arrête de vous suivre et l'animation de marche s'arrête. Vous pouvez tester et admirer la créature que vous avez programmé !

Nous avons vu des notions simples dans la création de cette intelligence artificielle. Ces notions fondamentales vont vous permettre de concevoir des IA beaucoup plus élaborées. Vous êtes capable de jouer une animation d'attaque si le PNJ est à 1 mètre du joueur, vous savez comment créer une variable `vie` qui peut être modifiée pendant un combat, vous pouvez améliorer le script que nous avons écrit ensemble si vous souhaitez aller encore plus loin.

Dans ce chapitre, vous avez vu comment concevoir un algorithme simple d'intelligence artificielle. Vous avez appris à créer un personnage non joueur et à programmer les éléments suivants :

- détecter un personnage ;
- suivre un personnage ;
- perdre de vue un personnage ;
- animer un personnage.

9

L’interface utilisateur

Pour rendre votre jeu plus professionnel et plus accessible aux joueurs, vous devez créer des menus (menu principal, menu pause, menu options, etc.). Dans ce chapitre, nous allons créer un menu principal qui permettra au joueur de lancer le jeu, de le quitter et, si besoin, de modifier certains paramètres si vous souhaitez donner cette possibilité au joueur. Dans un premier temps, nous allons voir comment créer un menu de A à Z, puis nous téléchargerons un package sur l’Asset Store afin de créer un menu futuriste. Enfin, nous créerons une fonction pour le chargement d’un niveau et une fonction pour quitter le jeu.

1. Création d’un menu avec des boutons

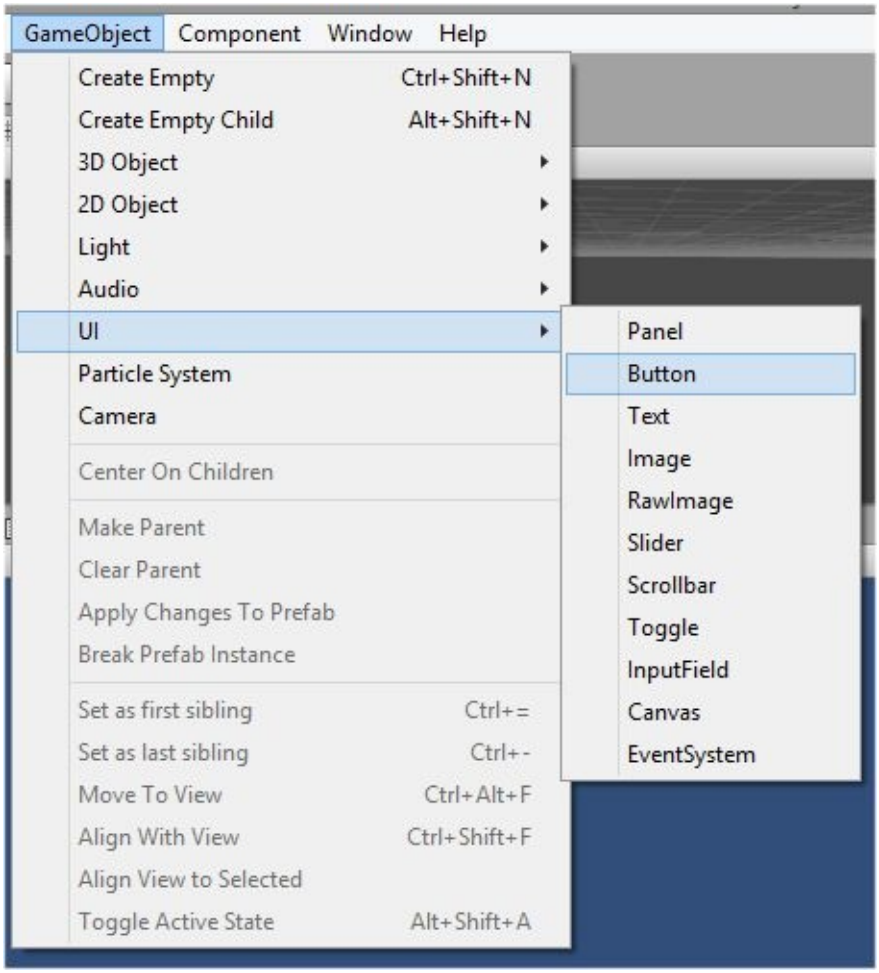
Nous allons créer, à l’aide du système d’UI de Unity, un menu très simple avec trois boutons, tel que présenté sur le schéma de la Figure 9.1.

***Note** > UI est l’abréviation de User Interface, soit interface utilisateur.*

***Figure 9.1 :** Schéma du menu*

Pour commencer, créez une nouvelle scène (Ctrl+N), que vous appellerez `MenuPrincipal`, et enregistrez-la (Ctrl+S). C’est là que vous allez construire l’interface. Ouvrez la scène de menu (vide). Depuis la version 4.6, Unity permet de créer très facilement des éléments d’interface via son menu GameObject/UI. La Figure 9.2 vous montre les différents éléments disponibles.

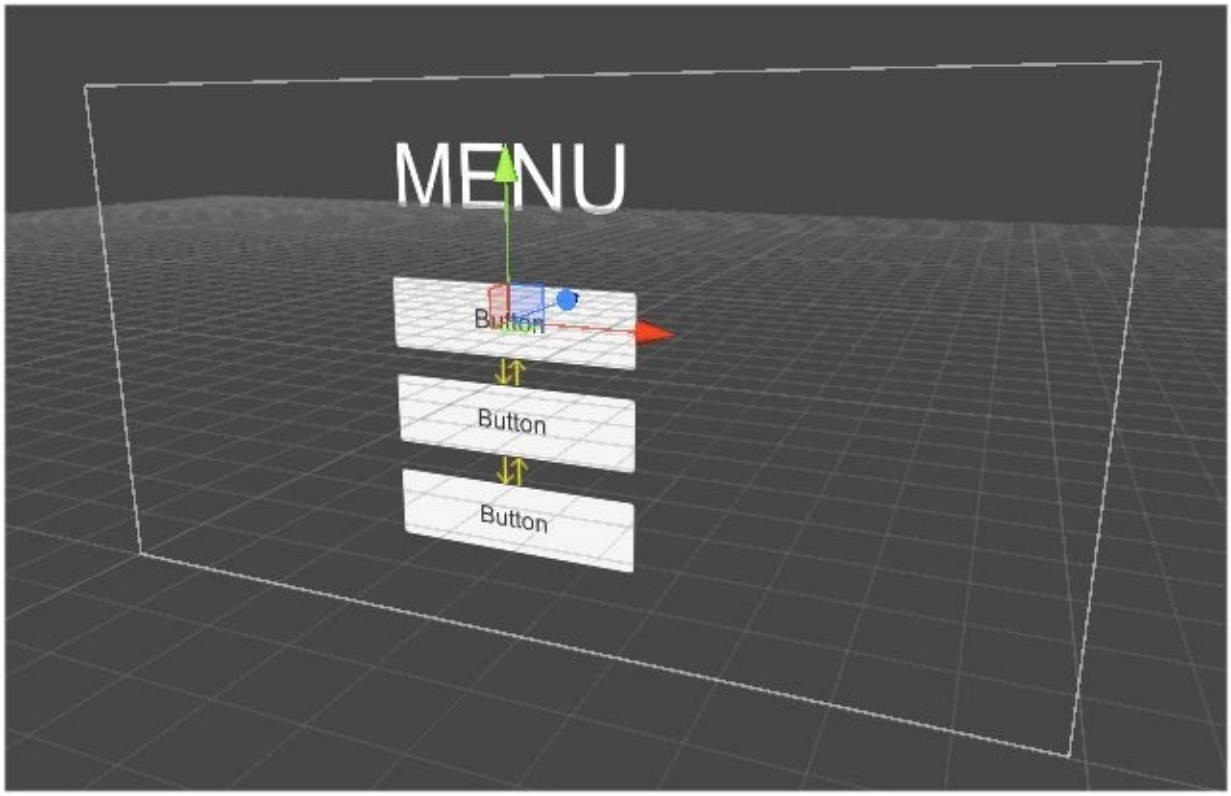
***Figure 9.2 :** Le système d’UI*



Vous pouvez ajouter du texte, des boutons, des zones de saisie (Input), des images, des barres de défilement (scrollbar), des cases à cocher (Toggle), etc.

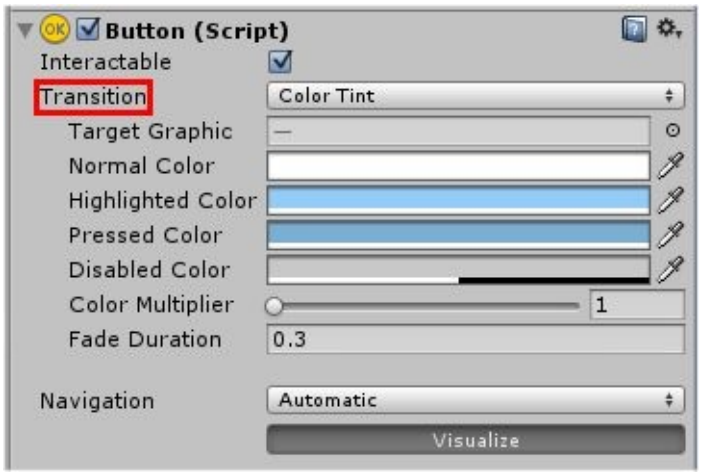
Vous allez créer un texte et trois boutons afin de reproduire le menu de la Figure 9.1. Redimensionnez vos éléments et positionnez-les comme vous le souhaitez. Vous pouvez changer la couleur, la taille, la police d’écriture, etc.

Figure 9.3 : *Conception du menu*



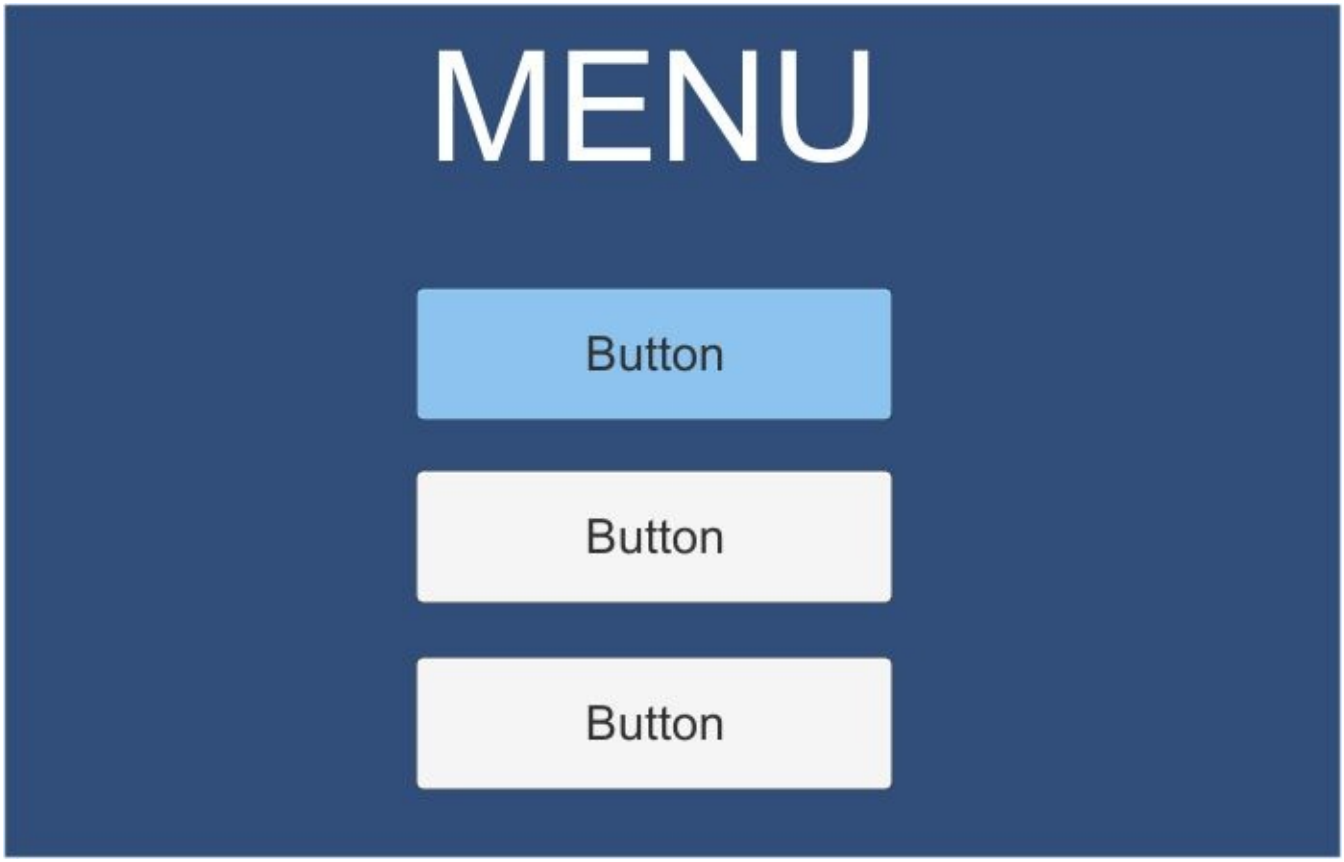
Lorsque vous cliquez sur un bouton, vous avez accès à certains paramètres dans l'inspector comme par exemple l'effet de transition. Vous pouvez par exemple provoquer un changement de couleur au survol de la souris en sélectionnant Color Tint.

Figure 9.4 : *Effet de transition sur le bouton*



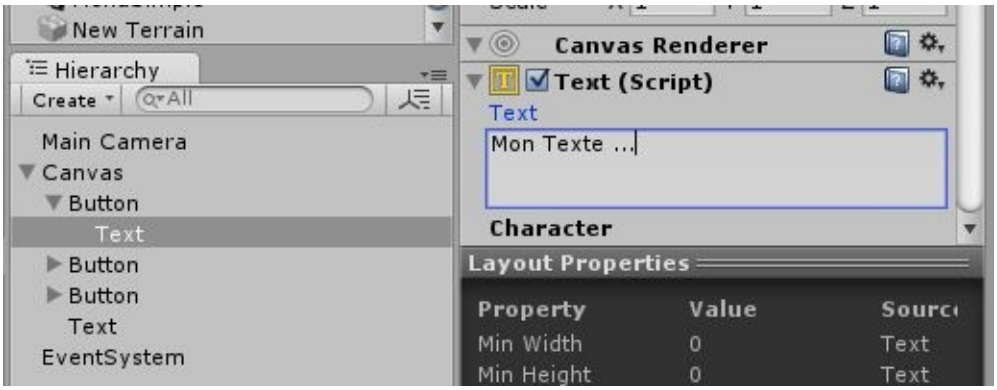
Ce qui nous donne :

Figure 9.5 : *Aperçu du menu principal*



Pour modifier le texte d’un bouton, vous devez modifier le texte se trouvant dans l’objet Button.

Figure 9.6 : *Modification du texte du bouton*



Vous pouvez rendre votre menu plus joli en ajoutant une image de fond et un système de particules pour animer la scène. Je vous invite à examiner le projet d’exemple téléchargé sur l’Asset Store à la section suivante pour voir comment construire un menu élaboré.

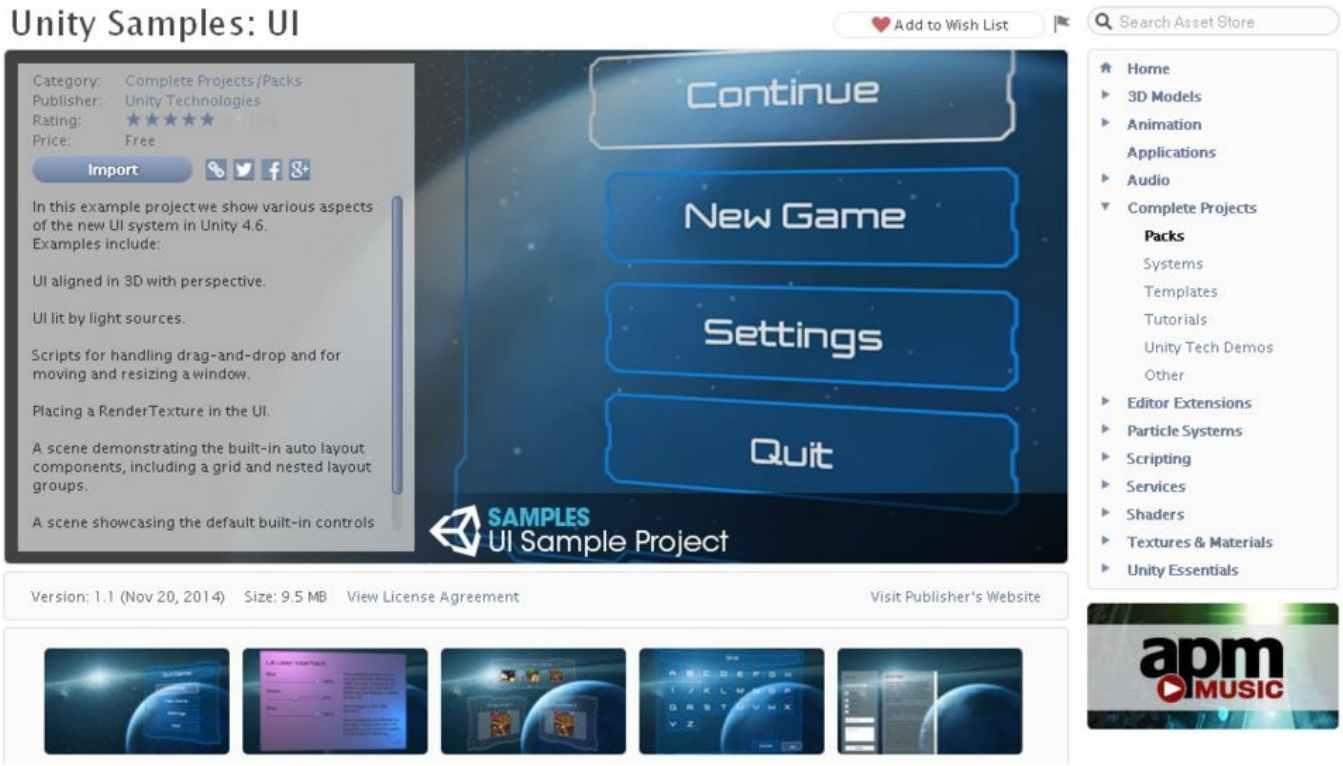
Le visuel de notre menu est terminé. Bien qu’il soit très simple, vous avez ce qu’il vous faut pour pouvoir jouer ou quitter le jeu. Maintenant que vous savez comment concevoir un menu par vous-même, vous allez apprendre à utiliser un menu déjà créé.

2. Utilisation d’un menu d’exemple

L’équipe de développeurs de Unity vous proposent de télécharger un menu moderne et futuriste gratuitement sur l’Asset Store. Il s’agit d’un projet d’exemple qui a pour but de vous montrer ce qu’il est possible de créer à partir des outils proposés par Unity. Nous allons donc télécharger ce projet et l’utiliser pour notre jeu.

Connectez-vous sur l’Asset Store et recherchez “Unity Samples : UI”.

Figure 9.7 : *Projet d’exemple UI*



Téléchargez et importez ce projet. Vous trouverez dans le dossier de nombreuses scènes très élaborées avec plusieurs types de menus. Nous choisirons la scène `Menu 3D` pour notre jeu. Voici à quoi ce menu ressemble :

Figure 9.8 : *Menu 3D*

Ce menu est en 3D et entièrement animé. Vous pouvez le modifier à votre gré et changer la disposition, la forme ou la couleur des éléments.

Nous allons maintenant passer à la partie programmation du menu. Vous pouvez indifféremment utiliser celui-ci ou celui que vous avez créé.

3. Programmation du menu

Nous allons maintenant créer le script C# qui va gérer les clics sur les boutons du menu. En fonction du bouton cliqué, celui-ci réagira d'une façon différente. Vous allez devoir créer deux fonctions : le chargement du jeu et la fermeture du jeu. Commencez par créer un nouveau script que vous pouvez appeler `Menu` et ouvrez-le pour l'éditer.

Vous pouvez supprimer les fonctions `Start` et `Update`, elles ne vous seront d'aucune utilité ici. Créez une nouvelle fonction publique que vous appellerez par exemple *`LoadLevel`* et qui prendra en paramètre le nom du niveau qui doit être chargé. Utilisez la fonction `Application.LoadLevel()` pour charger le niveau demandé. Voici à quoi ressemble la fonction :

```
public void LoadLevel(string lvl)
{
        Application.LoadLevel(lvl);
}
```

Le mot clé `string` permet de dire que l'argument `lvl` est une chaîne de caractères. En effet, `lvl` correspond au nom du niveau qui doit être ouvert.

Créez une seconde fonction publique *`ExitGame`* dans laquelle vous utiliserez la fonction `Application.Quit()`.

```
public void ExitGame(){
        Application.Quit();
}
```

Nos fonctions doivent impérativement être en "public" pour qu'elles puissent être appelées par le gestionnaire d'événements de Unity.

C'est tout ce dont vous allez avoir besoin pour votre menu principal. Voici le code complet de ce script :

```
using UnityEngine;
using System.Collections;

public class Menu : MonoBehaviour {

        public void LoadLevel(string lvl)
        {
                Application.LoadLevel(lvl);
        }

        public void ExitGame(){
                Application.Quit();
        }
}
```

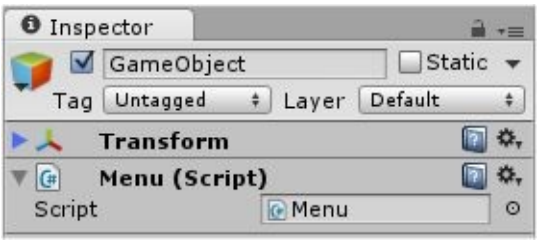
Pour pouvoir appeler une fonction lors d'un clic sur un bouton, sélectionnez le bouton que vous voulez paramétrer en cliquant dessus et regardez dans l'inspector du côté du gestionnaire des clics. Il se trouve dans la section Button (Script) et vous devez cliquer sur

le signe + pour ajouter votre événement :

Figure 9.9 : *Gestion du clic*

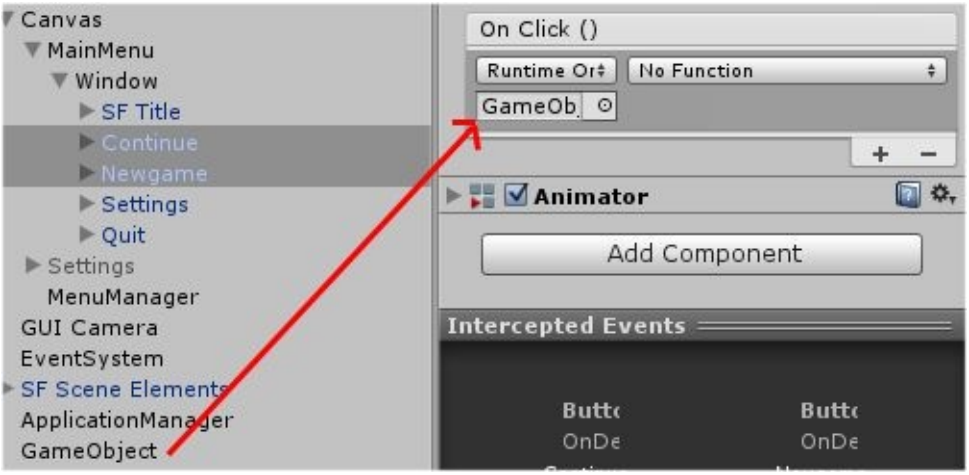
Si vous cliquez sur le signe +, vous aurez la possibilité de définir un événement (ou plusieurs) qui doit se produire lorsque le joueur clique sur le bouton. Pour accéder aux fonctions que nous avons créées, vous devez placer votre script sur un GameObject qui se trouve sur la scène. Je vous propose de créer un objet vide (afin de travailler plus proprement) GameObject/Create Empty et d'attribuer le script menu à cet objet.

Figure 9.10 : *Mise en place du script Menu*



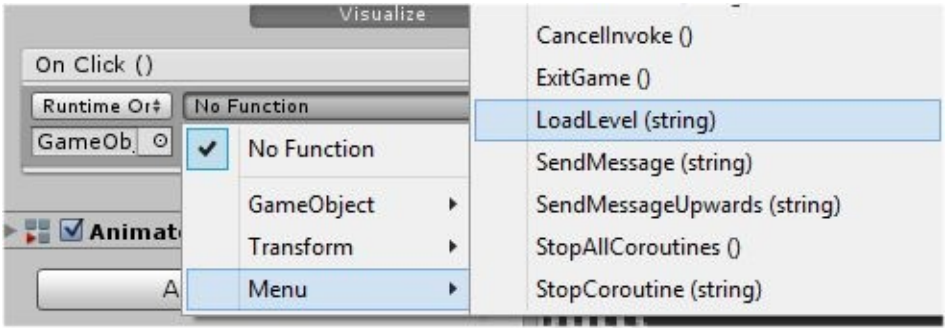
Maintenant vous pouvez faire glisser cet objet dans la variable GameObject du gestionnaire des clics (je rappelle que vous devez cliquer sur le signe + pour pouvoir ajouter votre GameObject) :

Figure 9.11 : *Ajout de l'objet à l'événement*



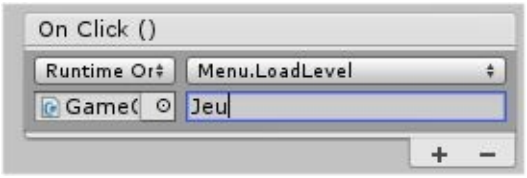
Une fois l'objet ajouté, vous avez accès à ses propriétés, ses scripts et aux fonctions publiques de ses scripts. Vous pouvez donc aller chercher la fonction `LoadLevel` en la sélectionnant dans la liste des fonctions du script :

Figure 9.12 : *Association de la fonction au clic*



Enfin, vous devez spécifier le paramètre de la fonction, ici il s'agit du nom du niveau que vous voulez charger :

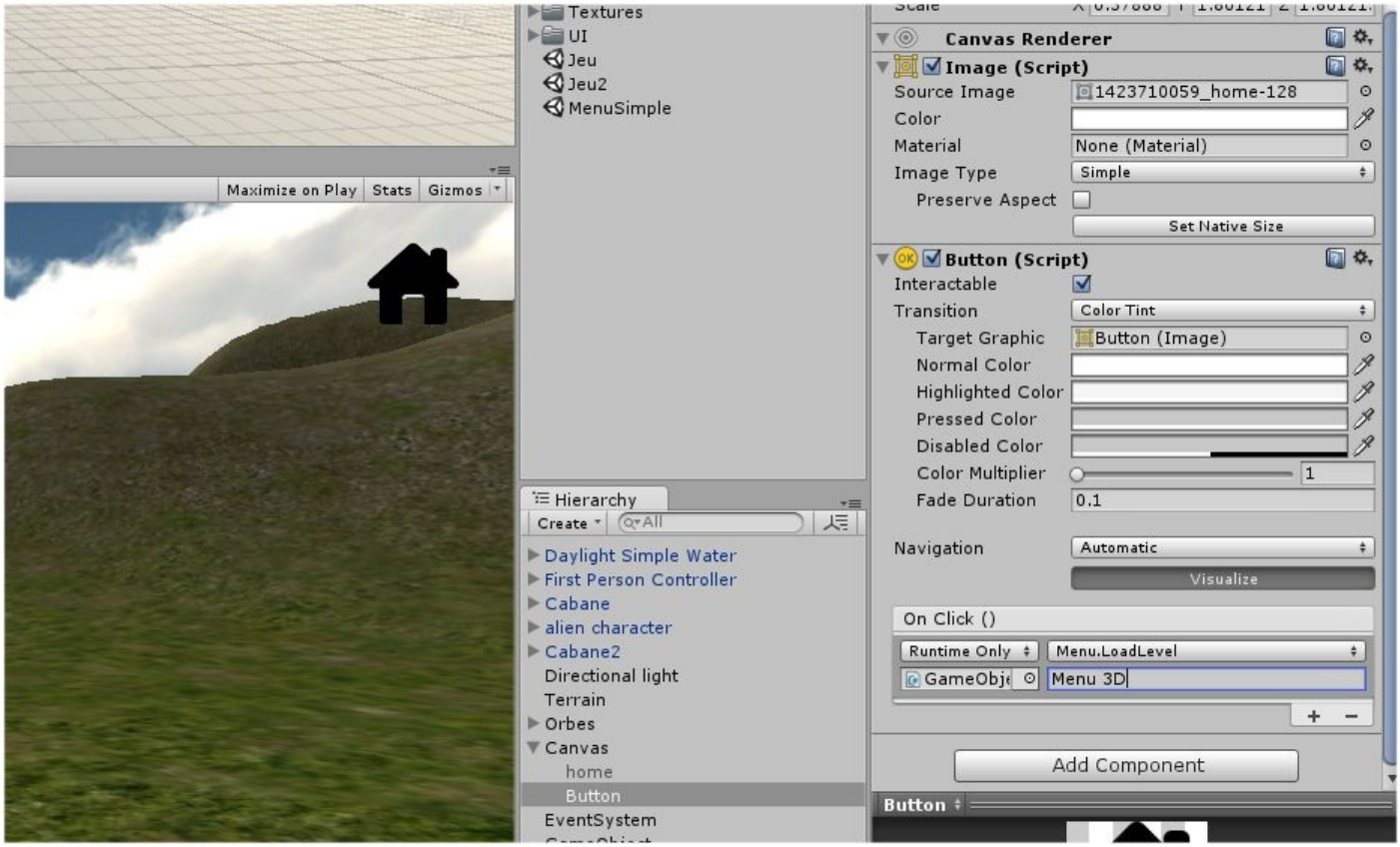
Figure 9.13 : *Ajout du paramètre de la fonction*



Maintenant, lorsque vous cliquerez sur le bouton, la scène se lancera. Vous pouvez tester et constater le bon fonctionnement de votre script. Procédez exactement de la même façon pour le bouton Quitter, qui doit appeler la fonction `ExitGame`.

Si vous voulez aller plus loin, vous pouvez ajouter à la scène du niveau un bouton Home en forme de maison qui permet au joueur de revenir au menu. Vous pouvez utiliser un bouton standard et mettre une image à la place du texte (voir Figure 9.14). Vous trouverez des icônes sur le site Iconfinder. Pour programmer le retour sur le menu principal, utilisez une nouvelle fois la fonction `Application.LoadLevel("Menu 3D");`.

Figure 9.14 : *Bouton Retour au menu*



Voilà pour la création de votre menu principal et de vos boutons d’interface. Nous avons vu les notions incontournables de la création de menus.

Dans ce chapitre, vous avez vu comment

- créer un menu utilisateur ;
- utiliser un menu préconfiguré ;
- programmer des événements ;
- gérer les clics sur un bouton.

10

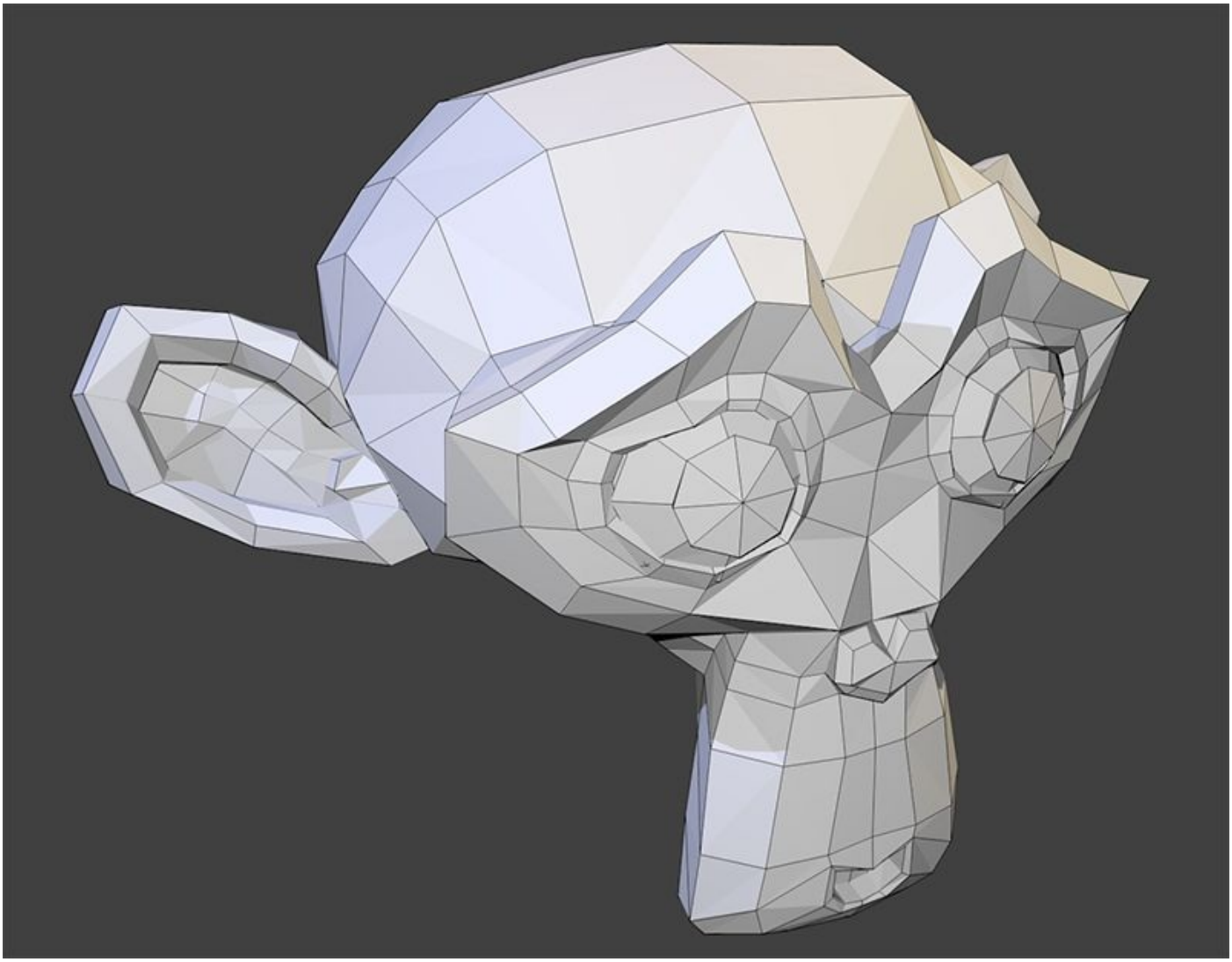
Optimiser son jeu

Pour rendre un jeu fluide sur tous les appareils, il existe plusieurs astuces à mettre en place dans Unity. Dans ce chapitre, vous allez découvrir les techniques essentielles de l’optimisation de vos scènes.

1. La caméra

La caméra permet au joueur de voir le monde virtuel. Plus elle voit de choses, plus la carte graphique (GPU) et le processeur (CPU) seront sollicités. Par exemple, un terrain 3D est un objet très complexe qui peut être composé de centaines de milliers de polygones. Le processeur et la carte graphique doivent calculer au moins 60 fois par seconde la position de chaque modèle 3D présents dans la scène. Cela représente des milliards de calculs chaque seconde. Un ordinateur peu puissant rencontrera des difficultés à effectuer autant de calculs en si peu de temps, et cela provoquera des ralentissements.

***Note** > Afin de bien comprendre ce que sont les polygones, voici un exemple de modèle 3D constitué de nombreux polygones qui constituent le visage. Chaque polygne permet de créer une partie du visage. Plus il y a de polygones, plus le modèle est détaillé. Dans le jeu vidéo, nous devons limiter le nombre de polygones pour conserver la fluidité.*



La meilleure façon d'optimiser un jeu est de diminuer le nombre d'éléments visibles. Nous pouvons par exemple diminuer la distance de vision et le champ de vision de la caméra. Bien sûr, vous devez être raisonnable dans votre choix. Dans notre jeu, le joueur est actuellement capable de voir jusqu'à 1000 mètres de distance, ce qui est un peu trop. 200 mètres suffiraient largement. Pour diminuer la distance de vision, cliquez sur la Main Camera se trouvant sur le First Person Controller et diminuez la valeur de la variable Far

qui représente la distance maximale de vision.

Figure 10.1 : *Modification de la distance de vision*

Maintenant le joueur peut voir jusqu'à 200 mètres au lieu de 1000. L'ordinateur n'aura plus à calculer le rendu des objets éloignés. Vous pouvez également réduire la valeur de Field of view pour diminuer le champ de vision. Ne diminuez pas trop cette valeur au risque d'obtenir un visuel médiocre et peu attractif. Vous devez optimiser votre jeu tout en gardant un bon visuel. Vous pouvez utiliser l'outil de statistiques de Unity pour voir comment tourne le jeu sur votre ordinateur. Il informe sur le nombre d'images par seconde, le nombre d'objets 3D à calculer par image et le nombre de polygones présents. Pour afficher les statistiques, cliquez sur l'onglet Stats de la fenêtre de jeu.

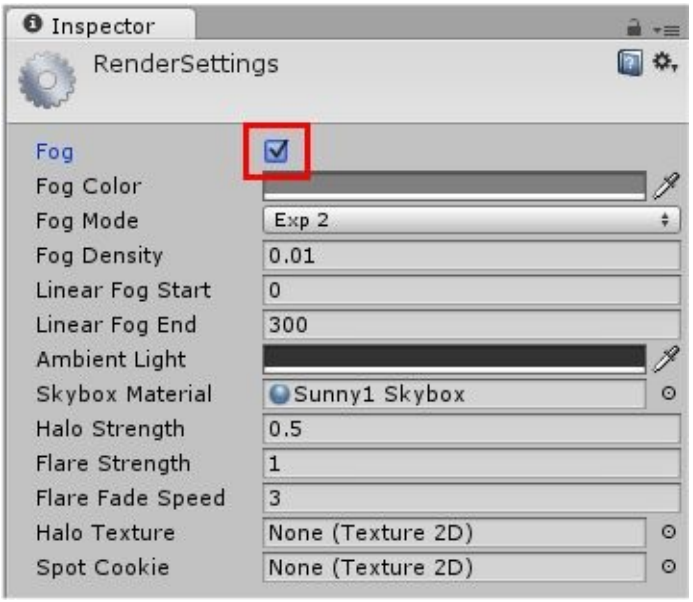
Figure 10.2 : *Onglet Statistiques*

2. Le brouillard

Le brouillard (fog) est utilisé dans la quasi-totalité des jeux professionnels. Il permet de créer un voile qui s'épaissit avec la distance pour masquer l'arrière-plan de la scène. Par exemple, nous avons configuré la caméra pour qu'elle ne voie pas à plus de 200 mètres. Nous pouvons donc configurer le brouillard de sorte à ce qu'il masque ce qui se situe à plus de 200 mètres : cela permettra de dissimuler le vide se trouvant au loin. Activez le brouillard en cochant la case Fog dans le menu Windows/Lighting.

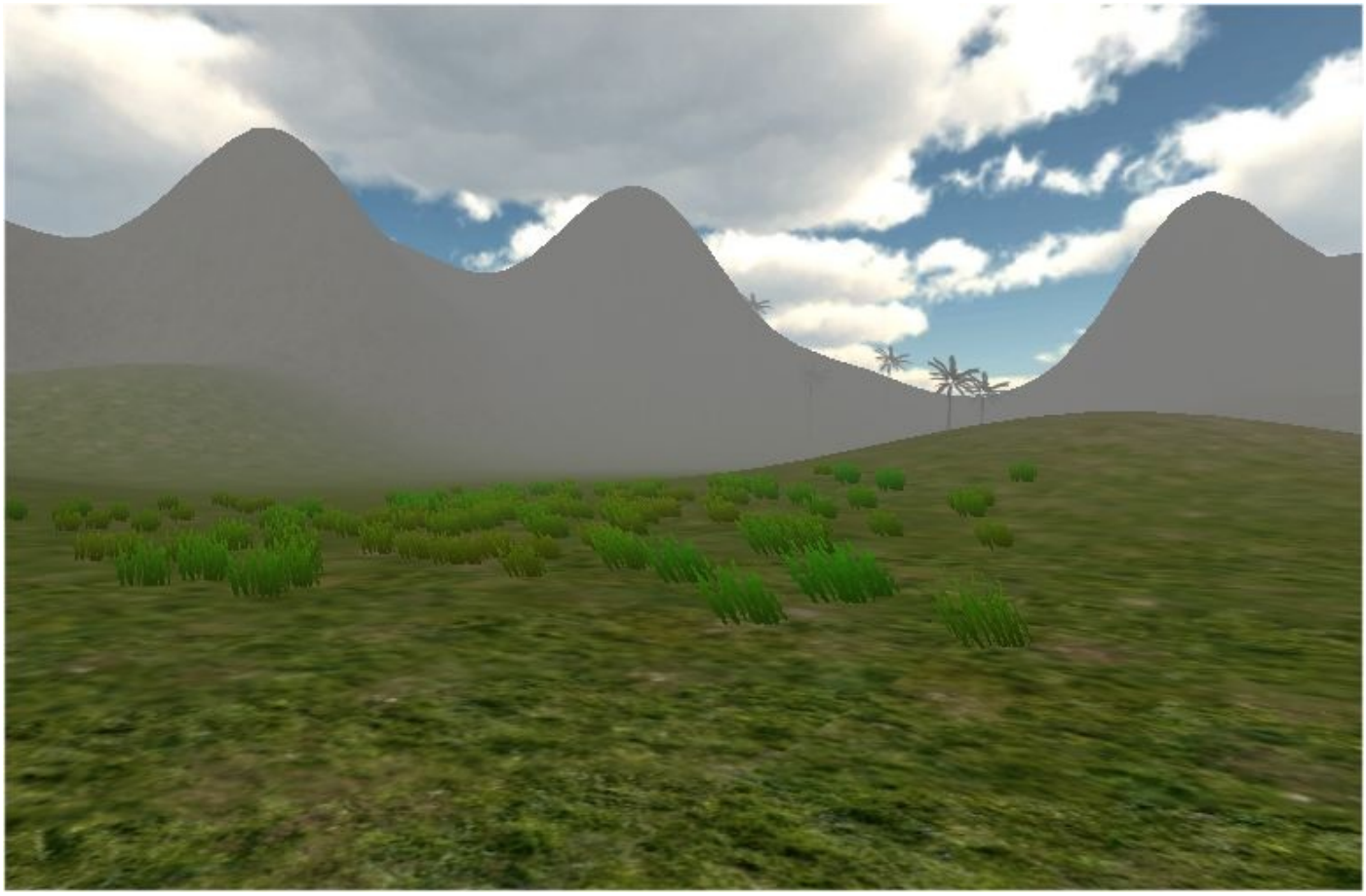
Figure 10.3 : *Activation du brouillard*

Attention *> Dans les anciennes versions de Unity, le brouillard peut être activé via le menu Edit/Render settings. Celui-ci a été remplacé par le Lighting dans la version 5.*



Les différentes valeurs vous permettent de définir la densité du brouillard ainsi que la distance à partir de laquelle celui-ci est visible. Dans mon cas, je vais laisser les valeurs par défaut et voici le résultat :

Figure 10.4 : *Brouillard*



3. Les ressources

L’optimisation d’un jeu passe également par l’optimisation des ressources que vous utilisez. Par exemple, n’utilisez pas des textures plus grandes que 512 × 512 pixels. Si votre texture est trop grande, réduisez sa taille avec un logiciel de retouche photo (GIMP, Inkscape, Photoshop, Pixelmator, etc.). Les modèles 3D que vous utilisez doivent être *low poly* ce qui signifie “peu de polygones”. Lorsque vous téléchargez des modèles 3D sur l’Asset Store, il est souvent précisé si ces modèles le sont ou pas. Plus vos ressources sont légères et simples, Plus elles seront faciles à calculer. Vous trouverez sur l’Asset Store des outils d’optimisation permettant d’améliorer les ressources de votre jeu.

4. Les lumières

Dans notre jeu, nous avons utilisé une *directional light* et c’est un bon choix car cette lumière se comporte comme un soleil. Cependant, si vous voulez davantage optimiser votre jeu, vous pouvez utiliser une *point light*, qui est moins gourmande en ressources mais qui n’éclaire qu’une petite zone de votre scène. La différence entre les deux lumières est la suivante : la *directional light* représente un soleil, qui éclaire la totalité de la scène de manière homogène, alors que la *point light* est un point lumineux qui éclaire la scène dans un rayon donné. Cette seconde lumière est plus “légère” mais n’est pas adaptée aux grandes scènes.

Si vous développez des jeux pour mobiles, vous serez dans l’obligation d’optimiser au maximum vos scènes car un téléphone est un appareil beaucoup moins puissant qu’un PC.

Dans ce chapitre, vous avez découvert comment optimiser vos scènes pour les rendre plus fluides et donc éviter les ralentissements. Vous avez appris à :

- diminuer la distance de vision de la caméra ;
- créer un brouillard pour masquer l’arrière-plan ;
- choisir des ressources optimisées ;
- utiliser un éclairage adapté.

11

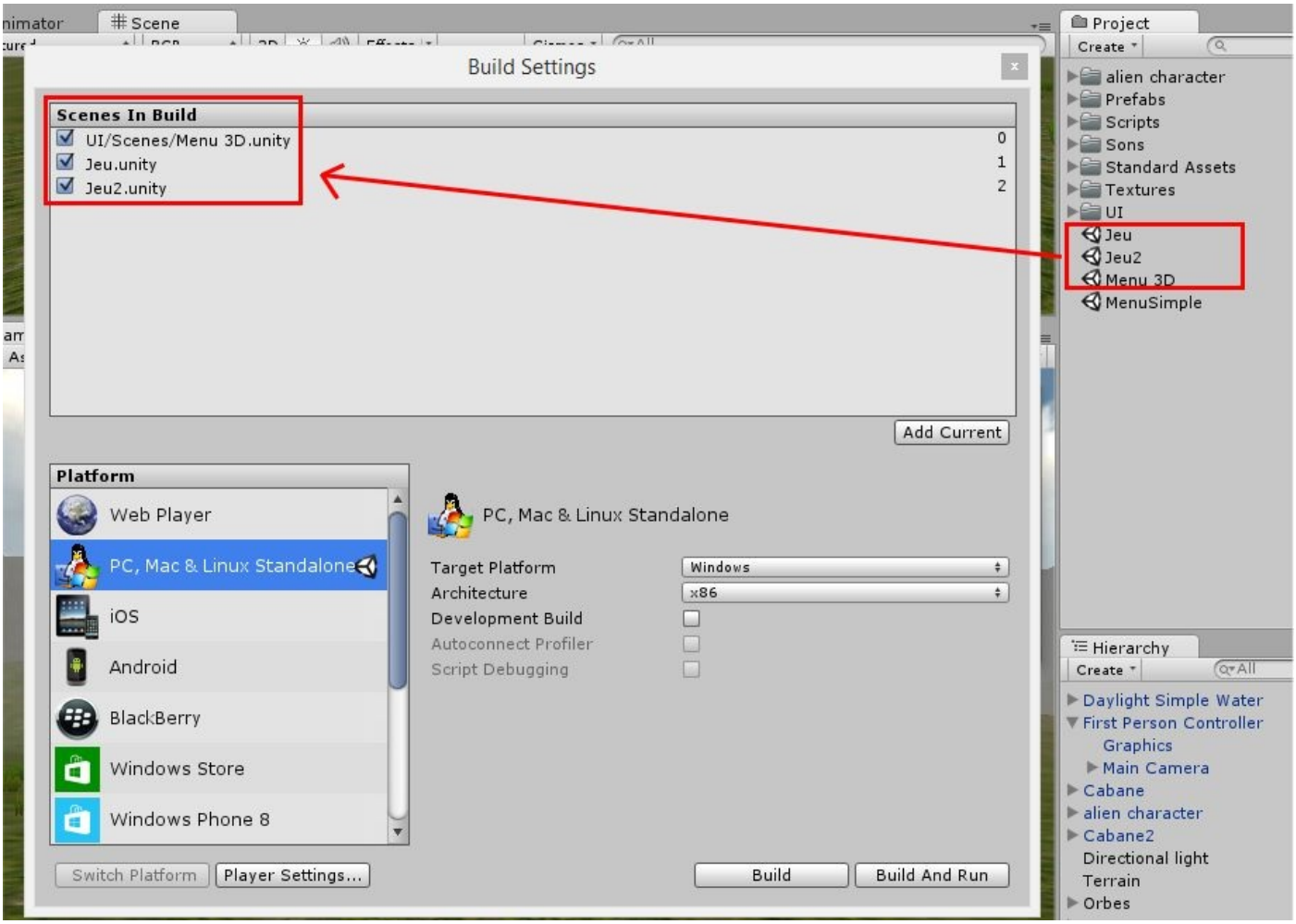
Partager son jeu

Votre jeu est terminé et vous souhaitez le partager avec vos amis ou le publier sur le web. Pour pouvoir le transmettre et permettre à d'autres d'y jouer, vous devez créer un (ou plusieurs) *exécutable(s)* à destination des plateformes que vous souhaitez cibler. Vous allez donc devoir compiler votre jeu. Dans ce chapitre, nous nous contenterons d'exporter la version Windows du jeu.

1. Préparation de la compilation

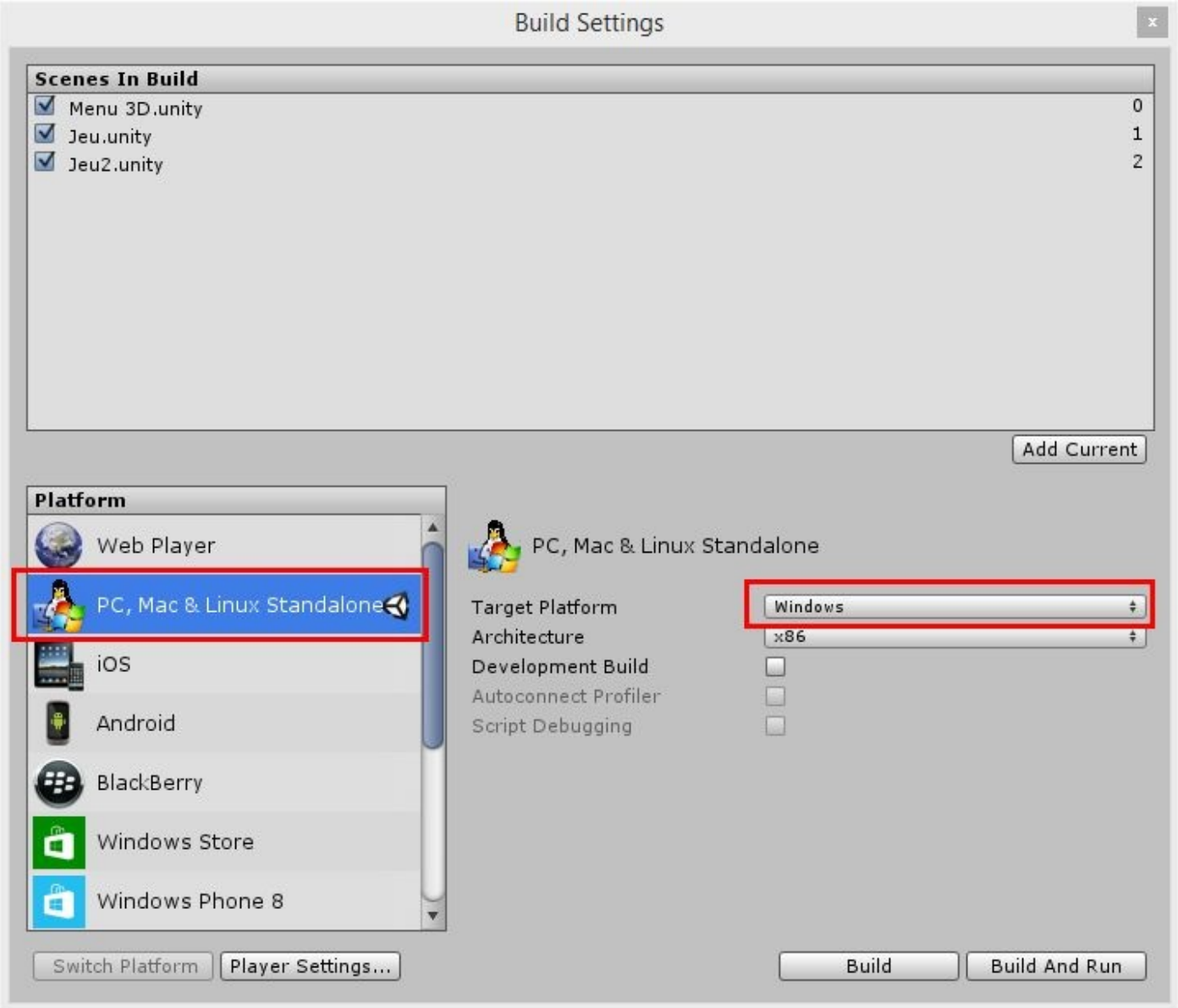
Pour commencer, vous devez spécifier les scènes qui doivent être inclues dans le projet via le menu File/Build settings. Dans notre cas, nous avons trois scènes : `Menu`, `Jeu` et `Jeu2`.

Figure 11.1 : *Ajout des scènes au build settings*



Une fois que vous avez choisi les scènes à compiler, vous devez sélectionner le type de plateforme pour lequel vous souhaitez compiler votre jeu. Nous allons choisir PC, MAC, Linux Standalone, qui va permettre de rendre le jeu jouable sur les ordinateurs de bureau. Dans la liste Target Platform, nous sélectionnons Windows avec comme architecture x86.

Figure 11.2 : *Sélection de la plateforme*

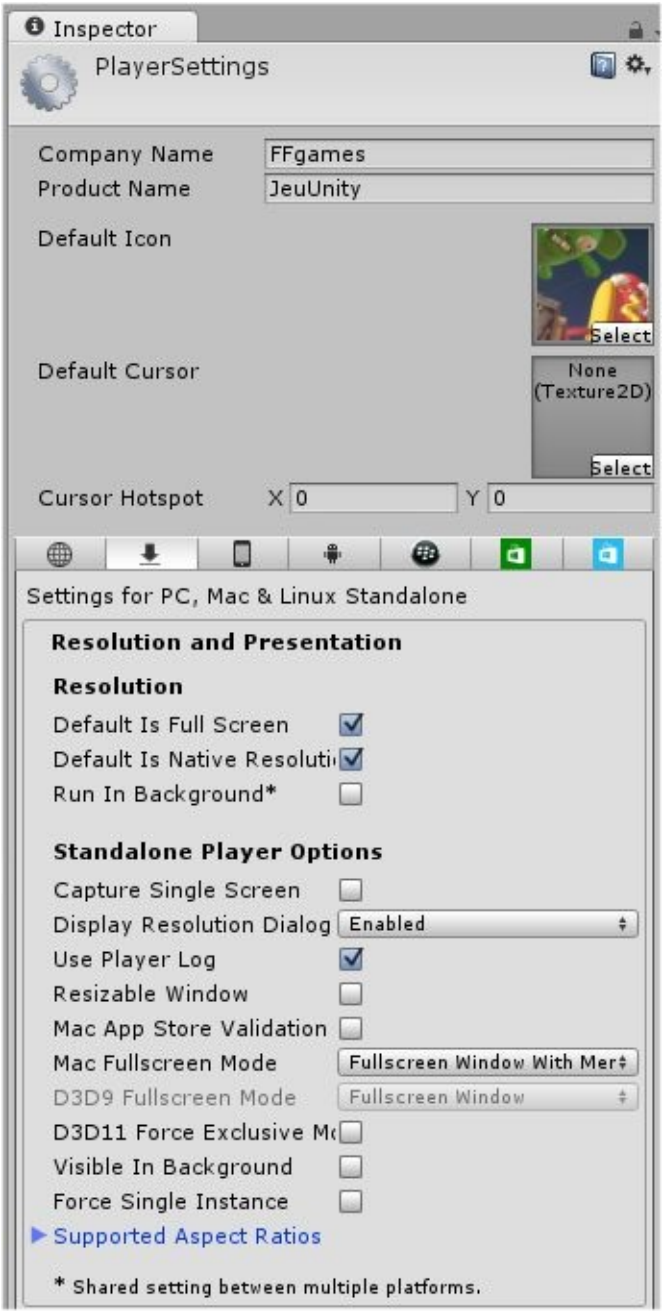


Si vous souhaitez changer de plateforme, par exemple cibler les navigateurs web (Web Player), sélectionnez dans la liste la nouvelle plateforme et cliquez sur le bouton Switch Platform pour qu'Unity s'adapte à celle-ci.

Attention *> Vous ne pouvez par compiler notre jeu d'exemple pour les téléphones et tablettes Android ou iOS car il ne serait pas compatible (pas de prise en charge de l'écran tactile et ressources non compatibles). De même pour les consoles de jeux, nous ne gérons pas les joysticks par exemple ; le développement n'est pas le même pour un jeu PC et un jeu console.*

Avant de lancer la compilation, nous allons modifier quelques paramètres (nom du développeur, nom du jeu, icône, etc.). Accédez à la fenêtre dédiée en cliquant sur le bouton Player settings. Ici vous allez pouvoir paramétrer votre jeu.

Figure 11.3 : *Paramétrage du jeu*

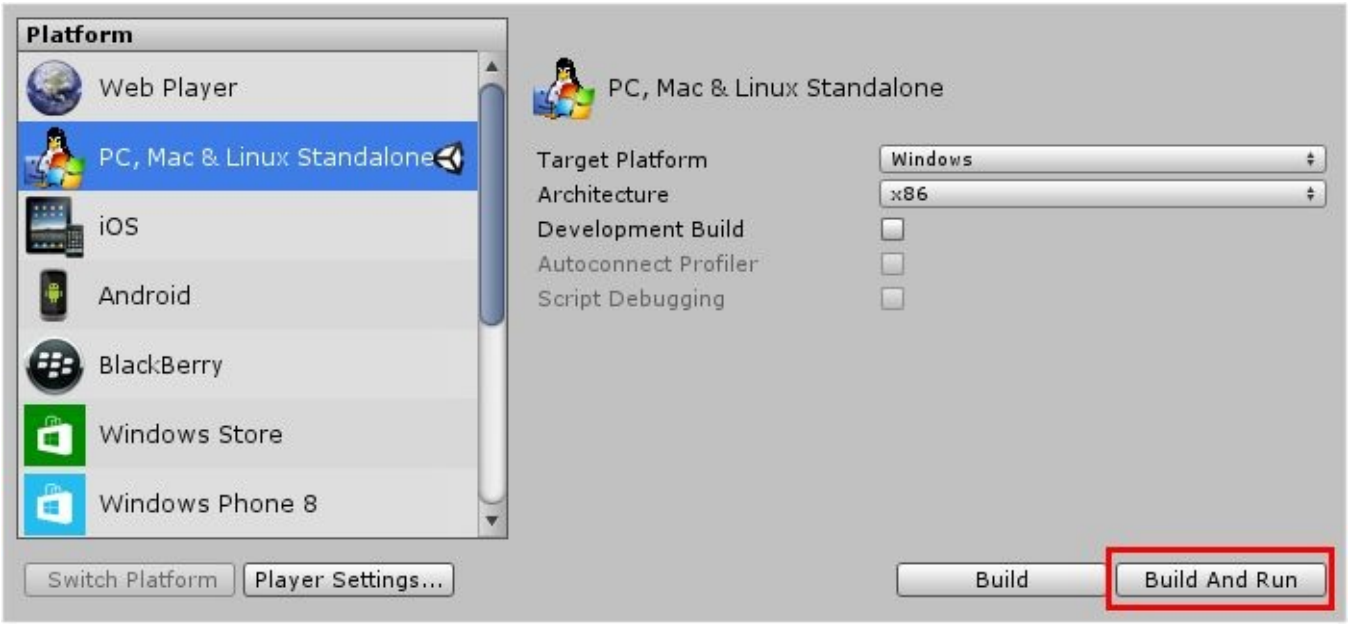


Note *> Dans ces mêmes paramètres, vous pouvez définir la résolution de votre jeu, définir si le mode plein écran doit être le mode par défaut, etc. Vous pouvez également demander à Unity de continuer à faire tourner le jeu même lorsque la fenêtre du jeu est en arrière plan. Je vous conseille de laisser les valeurs par défaut.*

2. Compilation et test

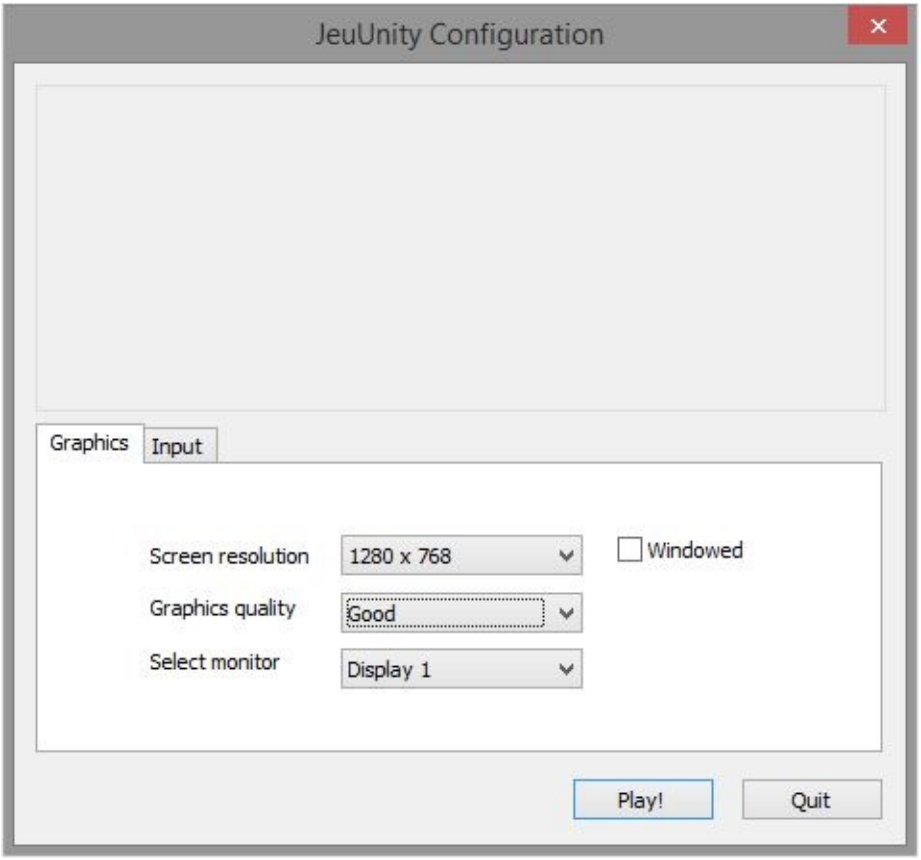
Lorsque vous avez terminé la configuration du projet, vous pouvez cliquer sur le bouton Build And Run pour compiler et tester votre jeu.

Figure 11.4 : *Build du jeu*



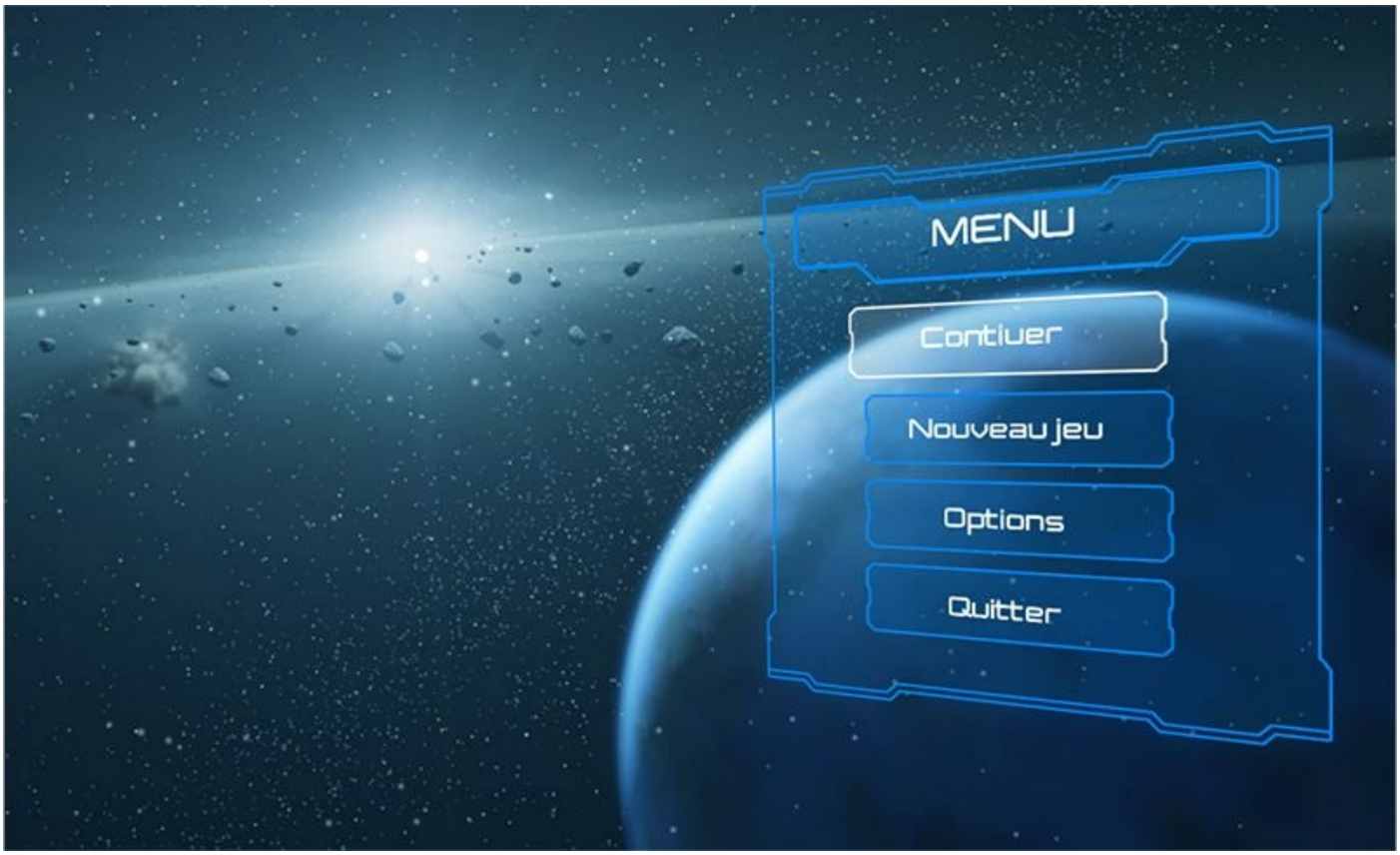
Vous devez spécifier un nom et un emplacement pour créer l'exécutable. Une fois fait, votre jeu compile et se lance. Une première fenêtre s'affiche, permettant au joueur de régler la qualité, la résolution et configurer les touches du jeu.

Figure 11.5 : *Lancement du jeu*



Lorsque vous validez, le jeu se lance.

Figure 11.6 : *Votre jeu*



Vous avez créé votre premier jeu avec Unity de A à Z ! Vous pouvez maintenant le

distribuer et continuer l’aventure. Pour le partager, vous devez transmettre le fichier exécutable et le dossier `Data`.

Dans ce chapitre, vous avez vu comment

- préparer un projet pour la compilation ;
- compiler un jeu pour PC ;
- partager votre jeu.

Pour aller plus loin

Dans ce livre, vous avez fait le tour des notions essentielles de la conception de jeux. Vous avez vu chaque étape, du level design au développement en passant par l'optimisation. Vous savez désormais comment concevoir un jeu avec Unity.

La prochaine étape pour vous est d'apprendre à concevoir des jeux pour les appareils mobiles afin de découvrir une autre façon de développer. En effet le développement d'un jeu mobile n'est pas le même que pour un jeu PC : les plateformes sont différentes donc vous devez adapter le gameplay et le design. C'est une aventure toute aussi intéressante qui vous attend si vous décidez d'apprendre à créer des jeux pour mobiles et tablettes !

Pour les plus courageux d'entre vous, sachez que vous pouvez également développer des jeux pour consoles de salon, oculus rift, lunettes de réalité virtuelle comme le Google CardBoard ou encore consoles portables. Les techniques de développement sont légèrement différentes en fonction de la plateforme pour laquelle vous programmez. Par exemple la Wii U dispose de deux écrans (la télévision et la manette à écran tactile). De plus les caractéristiques techniques de la console sont aussi à prendre en compte (puissance, manettes sans fil, reconnaissance de mouvements etc.).

Foire aux questions

1. Comment publier une version web de mon jeu ?

Lors de la compilation du projet, vous devez sélectionner Web Player. Unity va alors générer deux fichiers (html et unity3d) qu'il faudra placer sur votre serveur web. Cela permettra aux joueurs d'accéder à votre jeu à partir de leur navigateur en spécifiant l'adresse URL de votre site. Par exemple `http://unity3d.com/monjeu/NomDuJeu.html`.

2. Comment créer un personnage à la 3e personne ?

Si vous souhaitez que votre personnage joueur soit visible (vue à la 3e personne), optez pour Third person controller. Si vous souhaitez utiliser un personnage que vous avez créé vous-même, il suffira de placer la caméra derrière ou au-dessus de celui-ci.

3. Comment déclencher une fonction lors d’un clic souris ?

Vous pouvez détecter si l’utilisateur a cliqué sur un objet. Pour cela, vous devez utiliser la fonction `OnMouseUp(){//votre code;}`.

Pour plus d’informations, consultez la documentation sur les entrées clavier/souris.

4. Comment détecter si le joueur appuie sur une touche spécifique du clavier ?

Vous pouvez utiliser une condition telle que `if(Input.GetKeyDown("space"))` pour savoir si le joueur a appuyé sur la touche Espace.

Pour plus d'informations, consultez la documentation sur les entrées clavier/souris.

5. Mon jeu est lent, comment y remédier ?

Un jeu lent est un jeu mal optimisé. Concrètement cela signifie que le processeur fait trop de calculs. Vous devez diminuer le nombre d’éléments visibles à l’écran et utiliser des fonctions moins complexes. Par exemple, évitez d’utiliser trop souvent la fonction `Update` quand cela n’est pas nécessaire.

6. Mon personnage tombe en dessous du décor, pourquoi ?

Lorsqu’un objet passe sous le décor, c’est qu’il possède un composant Rigidbody et qu’il était en collision avec le décor avant le lancement du jeu. Une solution pour éviter ce genre de bug est de placer l’objet légèrement au dessus du sol. Cela peut aussi venir de l’absence de Collider sur le décor.

7. Mon son ne se lance pas !

Pour jouer un son, il faut que l’objet qui émet le son soit équipé d’un composant Audio source. Un Audio Listener doit également se trouver sur votre scène, on le place généralement sur la caméra principale.

8. Comment faire disparaître/apparaître un objet ?

Avec un script C#, vous pouvez activer ou désactiver n’importe quel GameObject. Pour ce faire, vous devez créer une variable de type GameObject et utiliser la fonction `SetActive()`. Par exemple :

```
monObjet.SetActive(false); // disparaître
monObjet.SetActive(true); // apparaître
```

Lexique

A

Asset

Ressource

Texture, son, bout de code ou modèle 3D prêt à être utilisé dans votre jeu.

B

Box Collider

Boîte de collision

Composant invisible permettant de détecter les collisions entre deux objets.

D

Draw calls

Nombre d’éléments affichés à l’écran simultanément.

E

Exécutable

Programme qui peut être exécuté sur un ordinateur.

F

First person controller

Personnage à la première personne

Personnage contrôlable (clavier/souris) en vue à la première personne.

Fonction

Sous-programme permettant d’exécuter des instructions.

G

GameObject

Objet de jeu

Tous les objets sont des GameObject (arbre, personnage, capsule…).

I

IA

Intelligence artificielle.

ips

Image(s) par seconde (Unité de mouvement).

L

Layout

Disposition

Sous Unity, l’option portant ce nom permet de configurer la disposition des fenêtres.

Level design

Façon d’assembler des éléments pour construire des niveaux pour notre jeu.

P

Prefab

Objet préconfiguré et réutilisable.

Primitive

Objet 3D simple comme un cube, une sphère ou un cylindre.

PNJ

Personnage non joueur.

R

RigidBody

Composant permettant à un objet de subir la gravité.

S

Shader

Nuanceur

Programme modifiant le rendu d'un objet. Permet de faire briller, rendre terne, amplifier ou atténuer la réflexion de l'objet ou de modifier son comportement vis à vis de la lumière.

T

Trigger

Déclencheur

Utilisé pour détecter un événement et déclencher un autre événement.

Text Mesh

Texte 3D

U

UI

User Interface : interface utilisateur

V

Variable

Espace de stockage permettant de conserver une valeur en mémoire.

W

WYSIWYG

What You See Is What You Get : ce que vous voyez est ce que vous avez

On désigne ainsi un éditeur visuel.

Index

A

B

C

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

V

Z