

# Tutoriel 1 : Prise en main de l'environnement Unity

---

Pour pouvoir réaliser le projet de l'UE, il est nécessaire de disposer et de maîtriser la plateforme de développement de jeu Unity. L'objectif de ce tutoriel est de vous familiariser avec cette plateforme avant de commencer le projet proprement dit.

## Contenu

1	Installation.....	1
2	Premier contact avec Unity .....	2
2.1	Créer un projet 2D .....	2
2.2	Présentation de l'interface.....	2
2.3	Parallèles avec le formalisme Entity-Component-System .....	3
2.3.1	GameObject.....	3
2.3.2	Component.....	3
2.3.3	FYFY .....	3
3	Premier projet en Unity.....	3
3.1.1	Création et initialisation du projet 2D .....	4
3.1.2	Créer un macrophage.....	4
3.1.3	Mettre en mouvement le macrophage .....	5
3.1.4	Créer un virus qui se déplace aléatoirement .....	6
3.1.5	Ajouter un décor.....	6
3.1.6	Traiter les interactions entre le macrophage et le virus .....	6
3.1.7	Générer dynamiquement d'autres virus .....	7

## 1 Installation

Dans le contexte des salles de TP, Unity a été pré-téléchargé et installé. Toutefois, pour réaliser le projet à venir les seules séances programmées ne suffiront pas et vous devrez fournir un travail personnel non négligeable. Par conséquent vous aurez très certainement besoin d'installer Unity sur votre ordinateur personnel : <https://unity3d.com/fr/get-unity/download>

*Lorsque vous lancez Unity pour la première fois, une fenêtre vous invite à vous identifier. Cette étape n'est pas obligatoire et vous pouvez cliquer sur le bouton **Work offline** si vous souhaitez passer cette étape.*

## 2 Premier contact avec Unity

Ce tutoriel présente les aspects d'Unity qui sont indispensables pour réaliser le projet. Ce document n'a cependant pas vocation à présenter l'ensemble des fonctionnalités du logiciel. N'hésitez donc pas à consulter le manuel officiel d'Unity : <https://docs.unity3d.com/Manual/>

### 2.1 Créer un projet 2D


Après avoir installé et lancé Unity, créez un nouveau projet de type **2D**.

**Note à propos des projets 2D :** Unity est une plateforme de développement de jeu 3D ; créer un projet 2D permet simplement de préconfigurer l'environnement au travail en 2D :


- Caméra positionnée en (0, 0, -10) orientée vers les Z positifs
- Caméra en projection orthographique (sans perspective)
- Impossibilité de faire tourner la caméra
- Vue d'édition alignée sur la caméra
- Déplacement planaire du point de vue en mode édition
- Importation de texture automatique sous forme de Sprite

### 2.2 Présentation de l'interface

Après avoir créé votre premier projet, l'environnement Unity se lance. Par défaut, l'interface d'Unity est composée de 5 vues principales (voir Figure 1) :

1. La vue de la Scène (**Scene**) : elle vous permet de positionner les différents objets du jeu (**GameObject** ou **GO**) dans l'environnement. Vous pouvez utiliser dans cette vue les outils de manipulation de GO  (déplacement, rotation, mise à l'échelle).
2. L'arbre des objets de la scène (**Hierarchy**) : il représente sous forme d'un arbre l'ensemble des GO actuellement positionnés dans la scène.
3. Le panneau de gestion du projet (**Project**) : il permet d'accéder à tous les Assets de votre projet. Un asset peut être une image, une scène, un son, une vidéo, un script ou un objet 3D. Lorsque vous importez des assets dans votre projet, ils apparaissent dans ce panneau.

**Attention :** ne confondez pas la **vue Scene** et un **asset de type scène**. Un asset de type scène est un fichier qui contient la description d'une scène (structuration, position et orientation des GO) tandis que la vue Scene permet de visualiser les assets de type scène.

4. **l'Inspector** : c'est un panneau contextuel qui présente les propriétés des GO ou assets sélectionnés et vous permet de les éditer.
5. La vue en jeu (**Game**) : elle permet de tester votre jeu et s'active automatiquement lorsque vous cliquez sur le bouton play . Pour sortir du mode **Game** et revenir au mode édition, il suffit de cliquer à nouveau sur le bouton play.

**Attention :** Unity vous offre la possibilité de modifier les GO en cours de test via **l'Inspector**. Sachez que toutes les modifications réalisées dans ce contexte seront **perdues** lorsque vous reviendrez en mode édition. Veillez donc bien à **quitter** le mode lecture avant de poursuivre le développement du jeu.

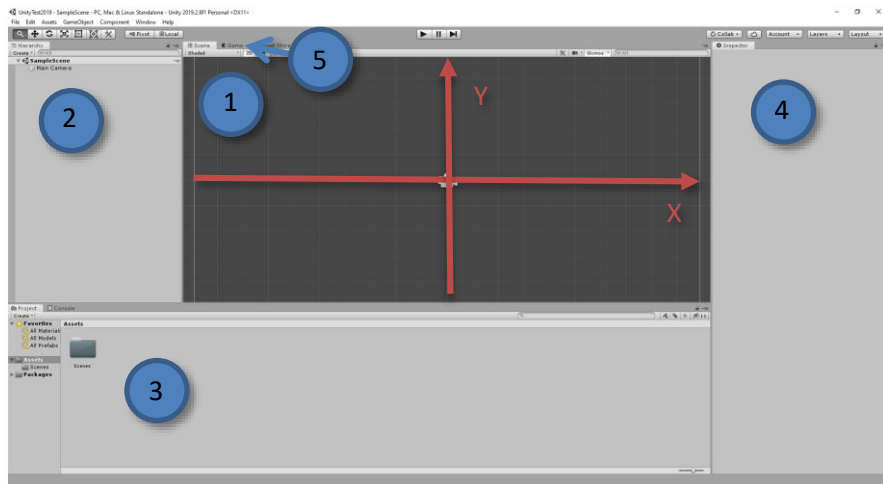


Figure 1

**Note :** toute l'interface d'Unity est configurable. Vous pouvez déplacer les différentes vues pour adapter votre environnement de travail à vos préférences. Le menu **Window** vous permet d'ajouter à votre environnement des vues non activées par défaut.

## 2.3 Parallèles avec le formalisme Entity-Component-System

Dans son usage « classique », Unity est une plateforme mettant en œuvre un mécanisme proche de l'Entity-Component-System (ECS). Dans le vocabulaire Unity nous parlons de **GameObjects** et de **Components** (ou Scripts). Il existe quelques différences que nous allons étudier ici.

### 2.3.1 GameObject

Le type d'objet le plus important dans un projet Unity est le **GameObject** (ou **GO**). Un **GO** est l'équivalent de l'**Entity** dans le formalisme Entity-Component-System, c'est un simple conteneur de composants. Créer un objet spécifique d'un jeu en Unity consiste donc à créer un **GO** et y associer les composants qui décrivent cet objet. Tout **GO** possède à minima un composant **Transform**.

### 2.3.2 Component

Dans la description d'un **GameObject**, les composants permettent de décrire les objets du jeu. Unity propose de nombreux types de composants préconstruits et laisse la possibilité aux développeurs de créer leurs propres composants sous la forme de **Scripts**.

Ici s'arrête le parallèle avec le formalisme Entity-Component-System car un script Unity est un objet contenant à la fois des données et de la logique, ce qui est en opposition avec le formalisme ECS pour lequel un composant est un objet ne contenant que des données.

### 2.3.3 FYFY

Compte tenu de ces observations, nous avons développé le module FYFY (Family For unitY) qui intègre dans Unity les concepts de **System** et de **Family** permettant de décrire la logique des composants. Ce premier tutoriel étant consacré à Unity « pur », nous reviendrons sur le framework FYFY dans un second tutoriel.

## 3 Premier projet en Unity

Dans cette section, vous allez aborder quelques concepts de base de l'outil **Unity** en créant un objet 2D représentant un macrophage, qu'il sera possible de déplacer grâce aux flèches directionnelles du clavier et qui sera capable de détruire des virus apparaissant aléatoirement dans la scène.

### 3.1.1 Création et initialisation du projet 2D

Lancez Unity et créez un nouveau projet en veillant à sélectionner l'option **2D**. Le répertoire **Assets** est situé à la racine du projet. La fenêtre **Project** de l'éditeur donne un accès direct à ce répertoire (Figure 2). Créez-y trois répertoires nommés **Scenes**, **Scripts** et **Textures**, qui contiendront respectivement les scènes Unity du projet, les fichiers de code C# et l'ensemble des images que vous allez utiliser au cours du projet. Vous pouvez également effectuer cette action dans l'explorateur de fichiers.

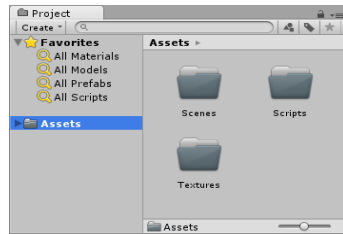


Figure 2

Dans le répertoire **Textures** nouvellement créé, copiez toutes les ressources images fournis avec cette documentation. Vérifiez pour chacune d'elles via l'**Inspector** que le type de Texture est bien **Sprite (2D and UI)**. Si tel n'est pas le cas c'est que **votre projet n'est pas de type 2D**, fermez le projet en cours, créez-en un nouveau en vérifiant bien que vous avez choisi un projet de type 2D (voir figure 3) et recommencé cette partie du tutoriel.

Enregistrez votre scène (pour l'instant vide) sous le nom **scene** au sein du répertoire **Scenes**.

A ce stade, le projet compile et affiche lors de son exécution une fenêtre vide à fond bleu (qui correspond au fond défini dans les paramètres de la caméra par défaut). Il faut maintenant peupler cette scène. Veillez à bien vérifier que vous n'êtes plus en mode lecture avant de passer à la suite du tutoriel.

### 3.1.2 Créer un macrophage

Pour créer un **GameObject** représentant un macrophage, il suffit de glisser-déposer l'image du macrophage (présente dans le répertoire **Textures**) de la fenêtre **Project** de l'éditeur vers la fenêtre **Scene** ou vers la fenêtre **Hierarchy**.

**Note :** cette action n'est possible que si son type est **Sprite (2D and UI)**. A noter également que c'est le type d'importation par défaut pour les projets de type 2D.

Votre scène contient maintenant un nouveau GameObject possédant automatiquement deux composants : un composant **Transform** et un composant **Sprite Renderer**.

Chaque objet de la scène possède obligatoirement un composant **Transform**. Il est utilisé pour stocker et manipuler la position, la rotation et la taille du GameObject (voir Figure 3).

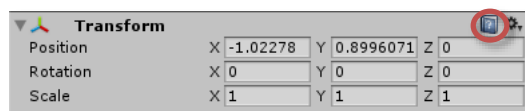


Figure 3

Quant au composant **Sprite Renderer**, il permet d'afficher une image sur une surface 2D, ici notre macrophage.

Pour plus de détails sur ces différents composants, vous pouvez accéder à la documentation Unity par l'intermédiaire des boutons d'aide de la fenêtre **Inspector** de l'éditeur, présents sur la première ligne de chacun des composants du **GameObject** (voir Figure 3).

A l'exécution, un macrophage apparaît dans la fenêtre.

Vous pouvez manipuler les paramètres de la caméra et la position du macrophage pour avoir une vision du jeu plus agréable.

### 3.1.3 Mettre en mouvement le macrophage

Vous allez ajouter un Script à votre macrophage afin de pouvoir le contrôler avec les touches directionnelles du clavier.

Commencez par créer un nouveau **Script C#** dans votre dossier **Scripts** et nommez-le **Move**. Double-cliquez sur le fichier **Move.cs** pour lancer l'environnement de développement et recopiez dans la fenêtre d'édition le code suivant (Figure 4) :

```
1 using UnityEngine;
2 using System.Collections;
3
4 // Indique à Unity que nous ne souhaitons pas que plusieurs
5 // composants Move puissent être associés à un même GameObject
6 [DisallowMultipleComponent]
7 public class Move : MonoBehaviour {
8     // Tout attribut public à un composant est automatiquement
9     // paramétrable dans l'Inspector (très utile pour paramétrer
10    // les composants par la suite)
11    public float speed = 2.5f;
12
13    // Update is called once per frame
14    void Update () {
15        Vector3 movement = Vector3.zero;
16
17        // Détermination du vecteur de déplacement en fonction de la touche pressée
18        if(Input.GetKey(KeyCode.LeftArrow) == true)
19            movement += Vector3.left; // Abréviation pour définir un Vector3(-1, 0, 0).
20        if(Input.GetKey(KeyCode.RightArrow) == true)
21            movement += Vector3.right; // Abréviation pour définir un Vector3(1, 0, 0).
22        if(Input.GetKey(KeyCode.UpArrow) == true)
23            movement += Vector3.up; // Abréviation pour définir un Vector3(0, 1, 0).
24        if(Input.GetKey(KeyCode.DownArrow) == true)
25            movement += Vector3.down; // Abréviation pour définir un Vector3(0, -1, 0).
26
27        // Mise à jour de la position du macrophage en fonction de sa position précédente,
28        // du vecteur de déplacement, de sa vitesse et du temps écoulé depuis la dernière frame.
29        transform.position += movement * speed * Time.deltaTime;
30    }
31 }
```

Figure 4

**Que faisons-nous ?** Ce script définit une classe **Move** qui implémente la fonction **Update** qui sera appelée automatiquement par **Unity** à chaque pas de simulation. Au sein de cette fonction, nous modifions le composant **Transform** du **GameObject** auquel le script est rattaché (ici le macrophage) en fonction des flèches directionnelles du clavier sur lesquelles on appuie.

Référez-vous à la documentation **Unity** pour vous éclairer sur les notions suivantes :

- Classe de base **MonoBehaviour**,
- Membre public visible par l'inspecteur,
- Attribut **DisallowMultipleComponent**.

**Important :** pour être exécuté, chaque script doit être ajouté en tant que composant à au moins un **GameObject**. Dans notre cas, sélectionnez le macrophage via la zone d'édition ou la fenêtre **Hierarchy**. Dans l'**Inspector**, cliquez sur le bouton **Add Component > Script >** puis sélectionnez le composant **Move**. Vous constatez dans l'**Inspector** que le composant **Move** a bien été ajouté au

macrophage. Notez également que l'attribut public **speed**, défini dans le code en mode **public**, est directement accessible et modifiable dans l'**Inspector**.

A l'exécution, vous pouvez maintenant déplacer le macrophage affiché à l'écran en pressant les touches directionnelles du clavier.

### 3.1.4 Créer un virus qui se déplace aléatoirement

A l'aide des sections précédentes, créez un **GameObject** représentant un virus qui se déplace vers des positions aléatoires dans la fenêtre. N'hésitez pas à consulter la documentation de la classe **Vector3**, vous y trouverez des fonctions intéressantes (en particulier la fonction **MoveTowards**).

### 3.1.5 Ajouter un décor

Pour créer un **GameObject** représentant le background, il suffit de glisser-déposer l'image du background (présente dans le répertoire **Textures**), de la fenêtre **Project** vers la fenêtre **Scene** ou **Hierarchy**.

Afin de garantir que les objets peuplant la scène ne soient pas masqués par l'image background (les objets étant par défaut tous affichés sur un même layer), modifiez l'ordre d'affichage du **GameObject** background en portant à -1 la valeur du paramètre **Order In Layer** de son composant **Sprite Renderer**. Cela a pour effet d'afficher d'abord le sol puis, au-dessus, tous les autres objets de la scène (le macrophage et le virus ont par défaut, pour **Order In Layer**, la valeur 0).

Le sol étant un objet immobile, en gardant sélectionné le **GameObject** background, cochez la case **Static** dans la fenêtre **Inspector**. Cela permettra à Unity d'alléger certains calculs durant la phase de jeu.

Reprenez les scripts de mouvement du macrophage et du virus afin de contraindre les déplacements dans la zone de jeu.

### 3.1.6 Traiter les interactions entre le macrophage et le virus

Nous allons maintenant chercher à détruire les virus à chaque fois qu'ils entrent en contact avec le macrophage.

Pour détecter les contacts entre deux objets, il est nécessaire que ces objets aient chacun un composant de type **Collider** (qui permet de générer une hitbox) et qu'au moins l'un des deux ait un composant de type **Rigidbody** (qui permet au **GameObject** associé d'être traité par le moteur physique).

Pour ce faire, ajoutez un composant **Rigidbody 2D** au **GameObject** du macrophage via l'**Inspector** : Add Component > Physics 2D > Rigidbody 2D. Si vous exécutez le jeu, vous vous rendrez compte que le macrophage est maintenant sensible à la gravité et tombe. Dans ce même **GameObject**, définissez dans le composant **Rigidbody 2D** le **Body Type** sur **Kinematic** pour désactiver ce comportement.

**Note** : les composants **Rigidbody** dont la propriété **Body Type** est définie sur **Kinematic** ne sont pas contrôlés par le moteur physique d'Unity. Il n'y a alors pas de prise en compte de forces telles que la gravité.

Ajoutez maintenant au **GameObject** du macrophage le composant **Circle Collider 2D** : Inspector > Add Component > Physics 2D > Circle Collider 2D.

Ajoutez également un composant **Circle Collider 2D** au **GameObject** du virus mais cochez-y le paramètre **Is Trigger**.

**Note** : définir le Collider du virus comme un Trigger permet d'économiser des ressources. Unity se chargera simplement de générer des messages indiquant la présence de collisions **sans faire appel au**

**moteur physique** pour résoudre les conséquences des collisions (points d'impact, forces d'impact...). Pour plus de détail consultez la documentation Unity des classes **Collider2D**, **Collision2D** et **Rigidbody2D**.

Nous allons maintenant ajouter un nouveau comportement au macrophage afin qu'il puisse « manger » un virus entrant en contact avec lui. Pour cela, créez dans le dossier **Scripts** un nouveau script et nommez-le **Eat**. Supprimez-y les méthodes **Start** et **Update** qui ne sont pas utiles dans ce composant, puis complétez le script par le code ci-dessous (Figure 5) :

```
1 using UnityEngine;
2 using System.Collections;
3
4 [DisallowMultipleComponent]
5 public class Eat : MonoBehaviour {
6     // Nous implémentons la méthode OnTriggerEnter2D car nous utilisons
7     // des Collider2D. Si nous avions utilisé des Collider classiques (3D)
8     // il aurait fallu implémenter la méthode OnTriggerEnter. Faites donc
9     // attention à implémenter la bonne méthode en fonction du bon type de
10    // Collider utilisé.
11    void OnTriggerEnter2D(Collider2D other){
12        if (other.gameObject.CompareTag ("Virus")) {
13            Destroy (other.gameObject);
14        }
15    }
16 }
```

Figure 5

**Que faisons-nous ?** Ce script définit une classe **Eat** qui implémente la fonction **OnTriggerEnter2D**. Cette fonction, qui va être associée au GameObject du macrophage, sera automatiquement exécutée par **Unity** lorsque ce GameObject entrera en collision avec un autre GameObject dont le **Collider2D** est défini comme **Is Trigger** (tel que le GameObject virus). Nous vérifions alors si le GameObject qui est entré en collision avec le macrophage est bien taggé comme « virus » et si tel est le cas, cet objet est détruit.

Ajoutez le composant **Eat** au GameObject du macrophage puis taggez le GameObject du virus en « **virus** » (à ajouter s'il n'est pas présent dans la liste des tags) : voir Figure 6.

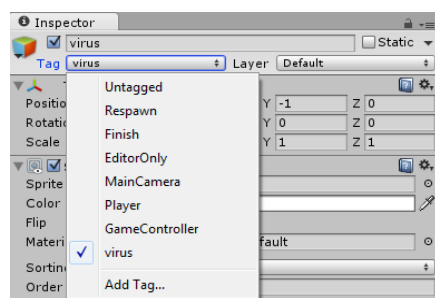


Figure 6

Maintenant, lors du jeu, le macrophage détruit les virus au contact.

### 3.1.7 Générer dynamiquement d'autres virus

Afin de générer à intervalle régulier des virus à des positions aléatoires, il est nécessaire de pouvoir créer des objets semblables au virus créé précédemment. Pour ce faire, il existe en **Unity** le concept des **prefabs**. Un **prefab** agit comme un template (un patron) d'objets avec lequel l'on peut générer autant d'instances que l'on veut, toutes semblables. Référez-vous à la documentation **Unity** pour plus de détail sur les **prefabs** : <https://docs.unity3d.com/Manual/Prefabs.html>.

Dans la fenêtre **Project** de l'éditeur, créez dans le répertoire **Assets** du projet un répertoire **Prefabs**. Glissez-déposez le GameObject virus de la fenêtre **Hierarchy** de l'éditeur vers ce répertoire **Prefabs**. Le Prefab virus est créé. Vous pouvez maintenant le supprimer de la **Hierarchy**.

Créez dans le dossier **Scripts** un nouveau script et nommez-le **VirusGenerator**. Complétez le script avec le code ci-dessous (Figure 7) :

```

1 using UnityEngine;
2 using System.Collections;
3
4 [DisallowMultipleComponent]
5 public class VirusGenerator : MonoBehaviour {
6     // Le prefab sera défini dans l'Inspector
7     public GameObject virusPrefab;
8
9     public float reloadTime = 2f;
10    public float reloadProgress = 0f;
11
12    void Update() {
13        reloadProgress += Time.deltaTime;
14        if(reloadProgress >= reloadTime) {
15            // Instanciation d'un nouveau virus
16            Instantiate<GameObject>(virusPrefab);
17            reloadProgress = 0;
18        }
19    }
20 }

```

Figure 7

**Que faisons-nous ?** Ce script définit une classe **VirusGenerator** qui implémente la fonction **Update** appelée automatiquement par **Unity** à chaque pas de simulation. Au sein de celle-ci, on instancie un nouveau virus à partir du Prefab **virus** toutes les 2 secondes.

Créez ensuite un GameObject vide appelé **virusFactory**. Pour ce faire, dans le menu **GameObject** choisissez l'option **Create Empty** (voir Figure 8). Ajoutez-lui le script **VirusGenerator** en tant que composant.

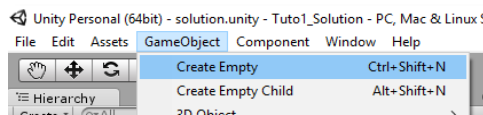


Figure 8

Il reste à paramétrer le composant **Virus Generator** pour lui indiquer quel est le prefab à utiliser pour instancier dynamiquement les GameObjects. Pour cela, glissez-déposez le prefab **virus** depuis le répertoire **Prefabs** de la fenêtre **Project** vers le paramètre **Virus Prefab** du composant **VirusGenerator** de l'objet **VirusFactory**, dans l'**Inspector** : voir Figure 9.

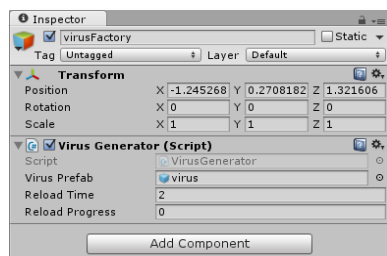


Figure 9

Modifiez ensuite le script **VirusGenerator** pour que la position de chaque nouveau virus instancié soit aléatoire.

**Vous pouvez maintenant déplacer le macrophage dans une scène sur laquelle apparaissent aléatoirement des virus.**