MOCAH Team

TME 2 : Prise en main du framework FYFY

Contenu

1 Unity et 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.

1.1 GameObject

Le type d'objet le plus important dans un projet Unity et 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**.

1.2 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.

1.3 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 TME étant consacré à Unity « pur », nous reviendrons sur le framework FYFY dans un second TME.

2 Installation

2.1 Code source et documentation

Code source de FYFY : https://github.com/Mocahteam/FYFY

Documentation de l'API : https://webia.lip6.fr/~muratetm/docFYFY/

2.2 Etape 1 : le cœur de FYFY

Le cœur de FYFY est composé de deux bibliothèques (.dll) que vous devez intégrer dans votre projet Unity. Commencez donc par créer un projet 2D avec Unity. Dans votre dossier **Assets** créez un dossier **Libraries** et copiez-y les fichiers **FYFY.dll, FYFY.xml et FYFY_Inspector.dll** fournis avec ce TME. En fonction de la version de .NET utilisé par votre version d'Unity, intégrez le bon package. Pour vérifier la version de .NET utilisée par votre version d'Unity consultez les informations dans le menu Edit > Settings > Player.

Note : le fichier **FYFY.xml** n'est pas indispensable au fonctionnement de la bibliothèque mais contient la documentation, il permet l'affichage d'infos bulle contextuelles lors du développement.

Après intégration du fichier **FYFY_Inspector.dll**, vous remarquerez l'ajout du nouveau menu **FYFY** dans la barre de menu d'Unity comme illustré sur la Figure 1. Ce menu FYFY vous permettra de créer la boucle principale (Main Loop) indispensable à l'exécution des systèmes (nous étudierons cette procédure par la suite).

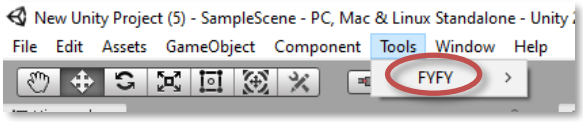

Figure 1

Vous devez configurer la bibliothèque **FYFY_Inspector.dll** pour qu'elle ne soit pas exportée dans le *build* final lorsque vous procéderez à la génération de votre jeu (cette dll contient du code qui ne doit être exécuté que dans le contexte de l'éditeur Unity). Pour ce faire sélectionnez le fichier **FYFY_Inspector.dll** situé dans votre dossier **Libraries** puis dans l'**Inspector** décochez la case **Any Plateform** et cochez la case **Editor** (voir Figure 2). Terminez en cliquant sur le bouton **Apply**.

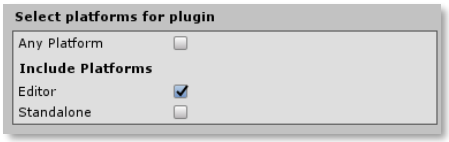

Figure 2

Note : cette première étape devra être réalisée **pour chaque nouveau projet**.

2.3 Etape 2 : les compléments

En complément de cette bibliothèque d'autres ressources vous sont proposées. Si vous travaillez sur votre ordinateur personnel, collez les fichiers contenus dans le dossier template fourni avec ce TME dans le dossier *C:\Programmes\Unity\Editor\Data\Resources\ScriptTemplates* (le chemin est à adapter en fonction de l'endroit où est installé Unity sur votre système).

L'intégration de ces fichiers ajoute, **après redémarrage d'Unity**, un sous-menu FYFY au menu **Assets > Create** (voir Figure 3). Ce sous-menu FYFY permet de créer des composants et des systèmes pré-formatés pour une utilisation avec FYFY.

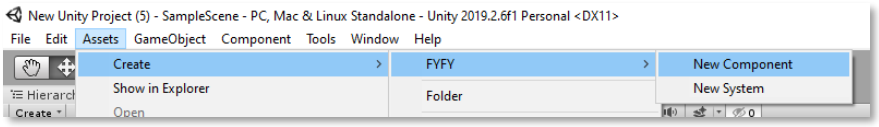

Figure 3

Note 1 : ces fichiers ont été préinstallés sur les ordinateurs de la salle machine. En revanche vous aurez à réaliser cette procédure sur votre ordinateur personnel.

Note 2 : cette seconde étape n'est à réaliser qu'une seule fois.

3 Premier projet avec FYFY

Dans cette section vous allez réaliser le même projet que dans le TME 1 Unity, mais cette fois en utilisant la bibliothèque FYFY. Cela vous permettra d'apprécier les différences entre l'approche Unity « pure » et l'approche Entity-Component-System de FYFY.

3.1 Initialisation d'un projet FYFY

Nous considérons que vous avez créé un **projet 2D** et que vous y avez intégré les bibliothèques **FYFY** et **FYFY_Inspector** tel qu'indiqué dans l'étape 1 de la section Installation. Si tel n'est pas le cas, faites-le.

Ajoutez à votre projet, dans le dossier **Assets**, les dossiers **Scenes**, **Components**, **Systems** et **Textures**. Ajoutez au dossier **Textures** les ressources images fournies avec ce TME et enregistrez votre scène sous le nom de votre choix dans le dossier **Scenes**.

Afin de terminer l'initialisation d'un projet **FYFY** vous devez créer la boucle principale qui vous permettra de gérer l'ordre d'exécution des systèmes et donc votre simulation. Via le menu **FYFY**, sélectionnez l'option **Create Main Loop**, comme illustré dans la Figure 4.

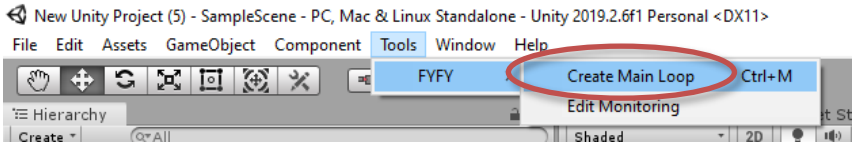

Figure 4

Vous remarquez dans la fenêtre **Hierarchy** l'ajout d'un nouveau GameObject nommé **Main_Loop**. Sélectionnez-le et observez sa structure via l'**Inspector**. Vous remarquez que, comme tout GameObject, il contient un composant Transform, mais il n'a **pas de représentation physique**. En effet, ce GameObject n'a pas vocation à être affiché dans la scène ; il vous servira notamment à paramétrer le contexte d'exécution de vos systèmes. Trois étapes de simulation vous sont proposées : **FixedUpdate**, **Update** et **LateUpdate** (voir Figure 5).

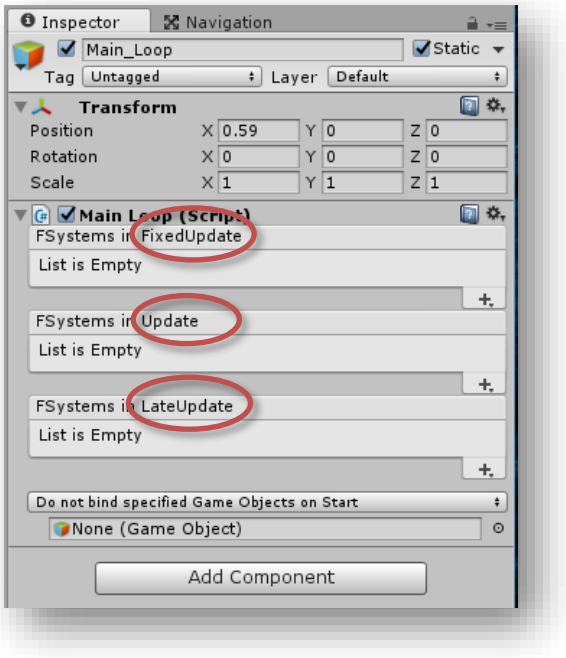

Figure 5

Ces trois étapes de simulation sont issues de la boucle de simulation d'Unity décrite à l'adresse suivante : https://docs.unity3d.com/uploads/Main/monobehaviour_flowchart.svg. Vous noterez dans le schéma qui y est présenté que :

- le **FixedUpdate** est exécuté dans le cycle **Physics**. Il est conseillé d'inscrire dans cette étape de simulation les systèmes travaillant sur la physique du jeu (gestion des collisions, applications de forces telles que la gravité…). **Important : plusieurs cycles d'appel au FixedUpdate peuvent être réalisés pour chaque frame**.
- l'**Update** est exécuté au début du cycle **Game logic** après la gestion des événements d'entrée. Il est conseillé d'inscrire dans cette étape de simulation les systèmes gérant les événements d'entrée (souris, clavier…) et les animations.
- le **LateUpdate** est également exécuté à la fin du cycle **Game logic**, mais les systèmes inscrits en **LateUpdate** vous garantissent d'être exécutés **après** les systèmes inscrits dans l'étape de simulation **Update** et donc après les étapes d'animation.

Par défaut, tous les GameObjects présents dans la fenêtre **Hierarchy** seront automatiquement traités par FYFY au démarrage du jeu et abonnés à leurs familles respectives. Si vous souhaitez contrôler les GameObjects traités par FYFY (voir Figure 6), vous pouvez :

- exclure un ensemble de GameObjects en sélectionnant **Do not bind specified Game Objects on Start** et en glissant les GameObjects à exclure depuis la fenêtre **Hierarchy** dans le champ prévu à cet effet ;
- inclure uniquement un ensemble de GameObjects en sélectionnant **Bind only specified Game Objects on Start** et en glissant les GameObjects à inclure depuis la fenêtre **Hierarchy** dans le champ prévu à cet effet ;

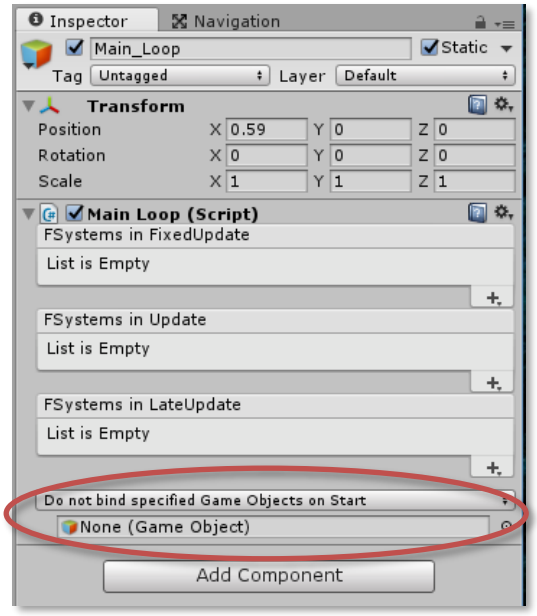

Figure 6

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 de ce TME.*

A noter que la vue du GameObject **Main_Loop** change dans l'**Inspector** lorsque vous lancez l'exécution de votre jeu (voir un exemple dans la Figure 7). Dans cette vue, vous ne pouvez plus abonner de nouveaux système aux différents contextes. En revanche quatre outils vous sont proposés :

- Le premier vous permet de gérer l'exécution de vos systèmes, chaque système peut être mis en pause de manière indépendante en cliquant sur les **interrupteurs vert/rouge** ;
- Le second (**FSystem profiler**) vous permet de visualiser le temps d'exécution de chaque contexte et ainsi identifier les systèmes gourmands en ressources en vue d'éventuelles optimisations ;
- Le troisième (**Bind tools**) vous permet au runtime d'ajouter ou retirer dynamiquement un Game Object à l'environnement FYFY.
- Le quatrième (**Families Inspector**) vous permet d'inspecter vos familles afin de connaître les GameObjects les peuplant. Très utile pour vérifier si les GameObjects présents dans les familles sont bien ceux attendus, sans quoi une révision de la déclaration des familles devra être réalisée dans les systèmes considérés pour améliorer le filtrage.

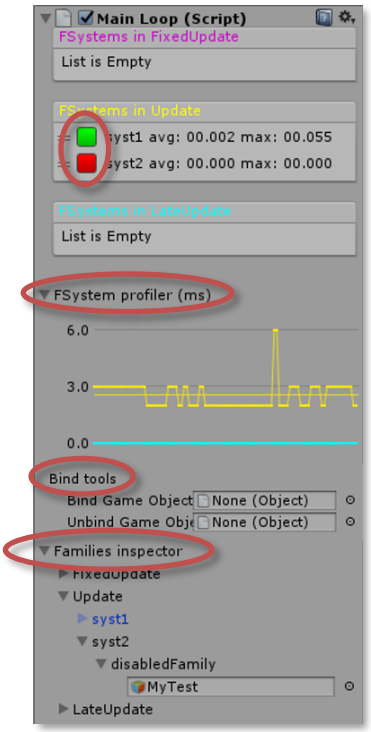

Figure 7

3.2 Créer et mettre en mouvement un macrophage

Créez un macrophage en faisant glisser 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**.

Dans le dossier **Components** créez un nouveau composant FYFY (**Assets > Create > FYFY > New Component**) et nommez le **Move**. Complétez ce composant avec le code de la Figure 8.

```
using UnityEngine;

public class Move : MonoBehaviour {
    // Advice: FYFY component aims to contain only public
    // members (according to Entity-Component-System paradigm).
    public float speed = 2.5f;
}
```

Figure 8

Dans le dossier **Systems** créez un nouveau système FYFY (**Assets > Create > FYFY > New System**) et nommez le **ControllableSystem**. Complétez ce système avec le code de la Figure 9.

Note 1 : Vous noterez que le template de création d'un système vous invite à implémenter quatre méthodes : **onStart**, **onPause**, **onResume** et **onProcess**.

Note 2 : Le paramètre **familiesUpdateCount** de la méthode **onProcess** vous permet de connaître le nombre de fois que les familles ont été mises à jour depuis le démarrage de la simulation.

```
using UnityEngine;
using FYFY;

0 références
public class ControllableSystem : FSystem {
    // Construction d'une famille incluant tous les GameObjects contenant le composant Move
    private Family f_controllable = FamilyManager.getFamily(new AllOfComponents(typeof(Move)));

    // Use to process your families.
    0 références
    protected override void onProcess(int familiesUpdateCount) {
        // Parcours des GameObjects inclus dans la famille
        foreach (GameObject go in f_controllable) {
            Vector3 movement = Vector3.zero;

            // Détermination du vecteur de déplacement en fonction de la touche préssée
            if(Input.GetKey(KeyCode.LeftArrow))
                movement += Vector3.left; // Abréviation pour définir un Vector3(-1, 0, 0).
            if(Input.GetKey(KeyCode.RightArrow))
                movement += Vector3.right; // Abréviation pour définir un Vector3(1, 0, 0).
            if(Input.GetKey(KeyCode.UpArrow))
                movement += Vector3.up; // Abréviation pour définir un Vector3(0, 1, 0).
            if(Input.GetKey(KeyCode.DownArrow))
                movement += Vector3.down; // Abréviation pour définir un Vector3(0, -1, 0).

            // Mise à jour de la position du macrophage en fonction de sa position précédente,
            // du vecteur de déplacement, de sa vitesse et du temps écoulé depuis la dernière frame.
            go.transform.position += movement * go.GetComponent<Move>().speed * Time.deltaTime;
        }
    }
}
```

Figure 9

Vous venez de créer votre premier composant et votre premier système compatibles avec **FYFY**.

Ajoutez le composant **Move** au macrophage comme un composant Unity classique (voir TME 1 Unity pour plus de détails) et enregistrez le système **ControllableSystem** dans l'étape de simulation **Update** du GO **Main_Loop** ; pour ce faire sélectionnez le GO **Main_Loop** dans la fenêtre **Hierarchy**, cliquez sur le bouton + de l'étape de simulation **Update** et sélectionnez dans la liste **ControllableSystem**.

Après avoir associé le système **ControllableSystem** à l'étape de simulation **Update**, vous devez obtenir le résultat illustré Figure 10.

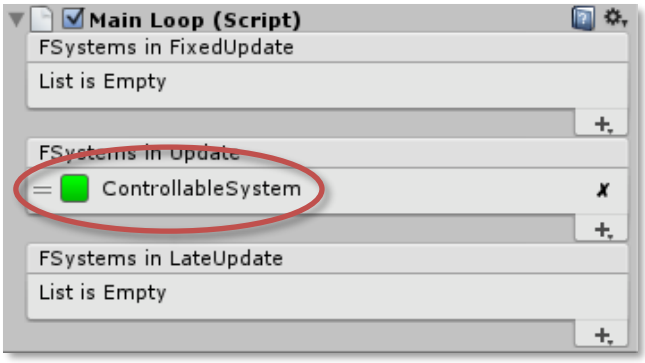

Figure 10

Note 1 : les systèmes actifs ont leur voyant en vert, les systèmes en pause ont leur voyant en rouge. Vous pouvez mettre en pause un système en cliquant sur le voyant vert. Si vous souhaitez réaliser un traitement particulier lors de la mise en pause ou du réveil d'un système, vous devez implémenter les méthodes **onPause** et **onResume** de ce système.

Note 2 : pour supprimer un système d'une étape de simulation, cliquez sur la croix située à sa droite.

Note 3 : un même système ne peut être associé qu'à une et une seule étape de simulation.

Note 4 : si plusieurs systèmes sont associés à une même étape de simulation, ils seront exécutés dans l'ordre d'affectation. Vous pouvez modifier l'ordre des systèmes au sein d'une même étape de simulation par drag and drop.

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

3.3 Modifier les attributs d'un système depuis l'Inspector

Comme vos systèmes n'héritent pas de MonoBehaviour, FYFY fournit un mécanisme de **wrappers** vous permettant de créer un pont entre l'**Inspector** de Unity et vos systèmes. Le dossier utilisé pour enregistrer les **wrappers** est défini dans la propriété **Wrappers directory** du **MainLoop** (voir Figure 11). Laissez cette propriété avec sa valeur par défaut.

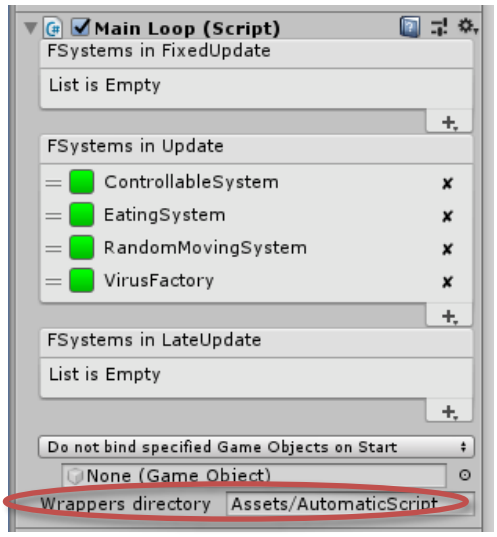

Figure 11

Pour chaque système ajouté à votre MainLoop, FYFY scanne son contenu et intègre à son **wrapper** les données compatibles avec l'**Inspector**. Pour expérimenter ce mécanisme nous allons modifier le système **ControllableSystem** comme illustré dans la Figure 12.

```
1  using UnityEngine;
2  using FYFY;
3
   0 références
4  public class ControllableSystem : FSystem {
5      // Construction d'une famille incluant tous les GameObjects contenant le composant Move
6      private Family f_controllable = FamilyManager.getFamily(new AllOfComponents(typeof(Move)));
7
8      public KeyCode left;
9      public KeyCode right;
10     public KeyCode up;
11     public KeyCode down;
12
13     // Use to process your families.
       0 références
14     protected override void onProcess(int familiesUpdateCount) {
15         // Parcours des GameObjects inclus dans la famille
16         foreach (GameObject go in f_controllable) {
17             Vector3 movement = Vector3.zero;
18
19             // Détermination du vecteur de déplacement en fonction de la touche préssée
20             if (Input.GetKey(left))
21                 movement += Vector3.left; // Abréviation pour définir un Vector3(-1, 0, 0).
22             if (Input.GetKey(right))
23                 movement += Vector3.right; // Abréviation pour définir un Vector3(1, 0, 0).
24             if (Input.GetKey(up))
25                 movement += Vector3.up; // Abréviation pour définir un Vector3(0, 1, 0).
26             if (Input.GetKey(down))
27                 movement += Vector3.down; // Abréviation pour définir un Vector3(0, -1, 0).
28
29             // Mise à jour de la position du macrophage en fonction de sa position précédente,
30             // du vecteur de déplacement, de sa vitesse et du temps écoulé depuis la dernière frame.
31             go.transform.position += movement * go.GetComponent<Move>().speed * Time.deltaTime;
32         }
33     }
34 }
```

Figure 12

Enregistrez votre système, revenez dans Unity et laisse-le compiler votre projet. Vous devriez maintenant pouvoir initialiser les attributs **left**, **right**, **up** et **down** de votre système via son **wrapper** automatiquement ajouté à votre **MainLoop** (voir Figure 13). Personnalisez votre système en indiquant dans l'Inspector de Unity les touches à utiliser pour contrôler votre macrophage.

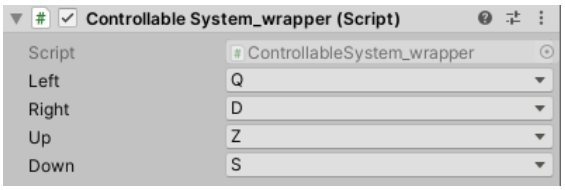

Figure 13

A l'exécution, vous pouvez maintenant déplacer le macrophage affiché à l'écran en pressant les touches que vous aurez renseigné.

Note 1 : ce mécanisme vous permet seulement d'initialiser les attributs de vos systèmes lors de leur création. Une fois les systèmes démarrés les données ne sont plus synchronisées avec les **wrappers**.

Note 2 : Pour qu'un attribut soit intégré au **wrapper** il doit être **public** et d'un **type compatible** (type primitif, string, Enum, type héritant de UnityEngine.Object, Vector2, Vector3, Vector4, Rect, Quaternion, Matrix4x4, Color, Color32, LayerMask, AnimationCurve, Gradient, RectOffset, GUIStyle UnityEngine.Object ou Vector3, un container de type Array ou List<T> contenant un type compatible)

3.4 Ajouter un décor

Voir le TME 1 Unity pour les détails.

3.5 Créer un virus qui se déplace aléatoirement dans la zone de jeu

Créez un virus en faisant glisser l'image du virus (présente dans le répertoire **Textures**) de la fenêtre **Project** de l'éditeur vers la fenêtre **Scene** ou vers la fenêtre **Hierarchy**.

En vous plaçant dans le dossier **Components**, créez un nouveau composant FYFY par **Assets > Create > FYFY > New Component**, et nommez-le **RandomTarget**. Complétez ce composant avec le code de la Figure 14.

```
1   using UnityEngine;
2
3   public class RandomTarget : MonoBehaviour {
4       // Advice: FYFY component aims to contain only public
5       // members (according to Entity-Component-System paradigm).
6
7       // Nous définissons la propriété "target" publique en accord avec
8       // le formalisme Entité-Composant-Système mais nous ne souhaitons
9       // pas que cette propriété soit modifiable dans l'Inspector.
10      // L'attribut [HideInInspector] assure ce mécanisme.
11      [HideInInspector]
12      public Vector3 target;
13  }
```

Figure 14

En vous plaçant dans le dossier **Systems**, créez un nouveau système FYFY (**Assets > Create > FYFY > New System**) et nommez le **RandomMovingSystem**.

Complétez ce système avec le code de la Figure 15.

Vous noterez que nous implémentons ici le constructeur de la classe **RandomMovingSystem** car nous souhaitons réaliser un traitement sur chaque GameObject initialement inscrit dans la famille et chaque nouveau GameObject entrant dans la famille. Nous parcourons donc la famille dans le constructeur pour initialiser les GameObjects présents au début de la simulation (ceux placés dans la scène en mode édition) et nous définissons un callback pour gérer ceux qui seront créés dynamiquement au cours de la simulation (anticipation sur la génération automatique de virus que nous traiterons à la fin de ce TME).

```
using UnityEngine;
using FYFY;

0 références
public class RandomMovingSystem : FSystem {
    // Construction d'une famille incluant tous les GameObjects
    // contenant le composant Move ET le composant RandomTarget
    private Family f_randomMoving = FamilyManager.getFamily(
        new AllOfComponents(typeof(Move), typeof(RandomTarget)));

    // Use to init system before the first onProcess call
    0 références
    protected override void onStart(){
        // Initialisation des GameObjects inclus dans la famille
        // au démarrage du système
        foreach (GameObject go in f_randomMoving)
            onGOEnter (go);

        // Définition d'une callback sur l'entrée d'un GameObject
        // dans la famille pour l'initialiser en conséquence
        f_randomMoving.addEntryCallback (onGOEnter);
    }

    2 références
    private void onGOEnter (GameObject go){
        go.GetComponent<RandomTarget>().target = go.transform.position;
    }

    // Use to process your families.
    0 références
    protected override void onProcess(int familiesUpdateCount) {
        // Parcours des GameObjects inclus dans la famille
        foreach (GameObject go in f_randomMoving) {
            // Récupération des composants du GameObject courant
            RandomTarget rt = go.GetComponent<RandomTarget> ();

            // Vérification si le GameObject est arrivé à destination
            if (rt.target.Equals (go.transform.position))
                // tirage aléatoire vers une nouvelle destination
                // dans la zone de jeu
                rt.target = new Vector3 ((Random.value - 0.5f) * 7,
                    (Random.value - 0.5f) * 5.2f);
            else
                // progression vers la destination
                go.transform.position = Vector3.MoveTowards (go.transform.position,
                    rt.target, go.GetComponent<Move>().speed * Time.deltaTime);
        }
    }
}
```

Figure 15

Pour terminer, ajoutez au GameObject **virus** les deux composants **Move** et **RandomTarget**, puis enregistrez le système **RandomMovingSystem** dans l'étape de simulation **Update** du GO **Main_Loop**.

Vous devez obtenir le même résultat que celui présenté sur la Figure 16.

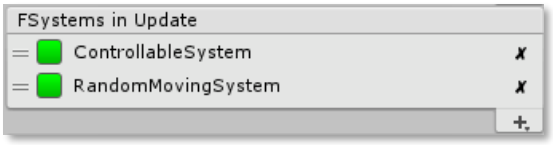

Figure 16

A l'exécution, vous constatez que le virus se déplace vers des positions aléatoires.

Cependant, si vous tentez de déplacer le macrophage à l'aide des touches du clavier, vous constatez que **vous perturbez aussi le déplacement du virus**. En effet le système **ControllableSystem** travaille sur la famille des GO contenant le composant **Move**, ce qui est le cas du macrophage, mais aussi du virus.

Pour corriger ce problème, nous allons modifier la famille du système **ControllableSystem** pour exclure les GO contenant le composant **RandomTarget** (voir Figure 17).

```
4   public class ControllableSystem : FSystem {
5       // Construction d'une famille incluant tous les GameObjects contenant
6       // le composant Move MAIS PAS le composant RandomTarget
7       private Family f_controllable = FamilyManager.getFamily(
8           new AllOfComponents(typeof(Move)),
9           new NoneOfComponents(typeof(RandomTarget)));
```

Figure 17

A l'exécution, vous constatez que le virus se déplace vers des positions aléatoires, mais sans être perturbé par la pression sur les touches directionnelles.

3.6 Traiter les interactions entre le macrophage et le virus

Comme pour le TME 1 Unity nous allons maintenant chercher à détruire les virus à chaque fois qu'ils entrent en contact avec le macrophage.

Comme le formalisme ECS interdit l'intégration de logique dans les composants, il n'est plus envisageable d'implémenter les méthodes **OnTriggerEnter2D** comme dans le TME 1 Unity.

Néanmoins il est intéressant de pouvoir utiliser le moteur physique d'Unity et son système de gestion des collisions.

Pour ce faire vous allez intégrer la bibliothèque **TriggerManager**. Copiez les fichiers **TriggerManager.dll** et **TriggerManager.xml** fournis avec ce TME dans le dossier **Libraries** de votre projet (contenu dans le dossier **Assets**). L'intégration de cette bibliothèque vous donne accès à deux nouveaux composants : **Trigger Sensitive 2D** et **Trigger Sensitive 3D**, auxquels on accède via **Add Component > Scripts > FYFY_plugins.TriggerManager**.

Ces deux composants vous permettent de rendre sensible un GameObject au moteur de collision d'Unity en respectant le formalisme ECS.

Lors d'une collision entre deux GameObjects, les GameObjects possédant un composant de la famille des **Trigger Sensitive** se verront automatiquement complétés d'un composant **Triggered2D** ou **Triggered3D**. Ces composants « éphémères » seront automatiquement retirés du GameObject lorsque les deux GameObjects ne seront plus en collision.

Note 1 : Les composants **Triggered2D** et **Triggered3D** fournissent un attribut **Targets** vous permettant de connaître la liste des GameObjects actuellement en collision avec le GameObject possédant un **Trigger Sensitive.**

Note 2 : Pour que les collisions soient traitées par le moteur physique d'Unity, les composants classiques (**Collider**, **RigidBody**…) sont toujours requis.

3.6.1 Initialisation des GameObjects pour permettre au moteur physique d'Unity de les gérer

Ajoutez un composant **Rigidbody 2D** et un composant **Circle Collider 2D** au GameObject du macrophage. Paramétrez le **Rigidbody 2D** en **Kinematic** pour désactiver l'application des forces sur ce GameObject (notamment la gravité).

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

3.6.2 Gestion des collisions avec le formalisme ECS

Rendez sensible le macrophage aux Triggers en lui ajoutant le composant **Trigger Sensitive 2D** via **Add Component > Scripts > FYFY_plugins.TriggerManager > Trigger Sensitive 2D**.

Dans le dossier **System,** créez un nouveau système FYFY (**Assets > Create > FYFY > New System**) et nommez le **EatingSystem**. Complétez ce système avec le code de la Figure 18.

Note : Pensez toujours à désabonner les GameObjects de Fyfy avant de les détruire sans quoi les familles seront peuplées de données inconsistantes.

```
using UnityEngine;
using FYFY;
using FYFY_plugins.TriggerManager; // Package contenant le composant Triggered2D

0 références
public class EatingSystem : FSystem {
    // Construction d'une famille incluant tous les GameObjects contenant
    // le composant Triggered2D (en collision)
    private Family f_triggeredGO = FamilyManager.getFamily(new AllOfComponents(typeof(Triggered2D)));

    // Use to process your families.
    0 références
    protected override void onProcess(int familiesUpdateCount) {
        // Parcours des GameObjects inclus dans la famille
        foreach (GameObject go in f_triggeredGO) {
            // Récupération des composants du GameObject courant
            Triggered2D t2d = go.GetComponent<Triggered2D> ();
            // Parcours des GameObjetcs actuellement en collision
            foreach (GameObject target in t2d.Targets) {
                // Désabonnement de la cible pour qu'elle ne soit plus gérée par Fyfy
                GameObjectManager.unbind (target);
                // Destruction du GameObject
                Object.Destroy (target);
            }
        }
    }
}
```

Figure 18

Terminez en enregistrant le système **EatingSystem** dans l'étape de simulation **Update** du GO **Main_Loop**.

Vérifiez que la position initiale du virus n'est pas la même que celle du macrophage. A l'exécution, vous constatez alors que le virus est détruit lorsque le macrophage entre en collision avec lui.

Qu'avons-nous fait ? Les deux GameObjects de la simulation (le macrophage et le virus) ont été paramétrés pour être pris en compte par le moteur physique d'Unity (par l'ajout des composants **RigidBody** et **Collider**).

Nous avons également ajouté le composant **Trigger Sensitive 2D** au macrophage pour le rendre sensible aux collisions en respectant le formalisme ECS.

Lorsque les deux GameObjects entrent en collision, la présence du composant **Trigger Sensitive 2D** génère un composant **Triggered2D** dans le macrophage pour notifier la collision. Il ne reste plus qu'à parcourir les objets en collision pour les supprimer.

3.7 Générer automatiquement d'autres virus

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** pour créer le Prefab virus.

Dans le dossier **Components**, créez un nouveau composant FYFY (**Assets > Create > FYFY > New Component**) et nommez le **Factory**. Complétez ce composant avec le code de la Figure 19.

```
1   using UnityEngine;
2
3   public class Factory : MonoBehaviour {
4       // Advice: FYFY component aims to contain only public
5       // members (according to Entity-Component-System paradigm).
6       public float reloadTime = 2f;
7       public float reloadProgress = 0f;
8
9       public GameObject prefab;
10  }
```

Figure 19

Dans la fenêtre **Hierarchy** créer un nouveau GameObject vide, nommez-le **VirusFactory** et ajoutez-y le composant **Factory** précédemment créé.

Dans le dossier **Systems**, créez un nouveau système FYFY (**Assets > Create > FYFY > New System**) et nommez le **VirusFactory**. Complétez ce système avec le code de la Figure 20.

```
using UnityEngine;
using FYFY;

0 références
public class VirusFactory : FSystem {
    // Construction d'une famille incluant tous les GameObjects contenant
    // le composant Factory
    private Family f_factory = FamilyManager.getFamily(new AllOfComponents(typeof(Factory)));

    // Use to process your families.
    0 références
    protected override void onProcess(int familiesUpdateCount) {
        foreach (GameObject factory_go in f_factory)
        {
            Factory factory = factory_go.GetComponent<Factory>();
            factory.reloadProgress += Time.deltaTime;
            if (factory.reloadProgress >= factory.reloadTime)
            {
                // Instanciation d'un nouveau virus
                GameObject go = Object.Instantiate<GameObject>(factory.prefab);
                // Abonnement de ce nouveau GameObject à Fyfy
                GameObjectManager.bind(go);
                // Positionnement aléatoire du virus
                go.transform.position = new Vector3((Random.value - 0.5f) * 7,
                    (Random.value - 0.5f) * 5.2f);
                factory.reloadProgress = 0;
            }
        }
    }
```

Figure 20

Terminez en enregistrant le système **VirusFactory** dans l'étape de simulation **Update** du GO **Main_Loop**.

A l'exécution, vous constatez qu'un nouveau virus apparaît toutes les 2 secondes.

3.8 Lié un élément d'UI avec un système

Nous allons maintenant créer un bouton permettant de faire apparaître un certain nombre de virus.

Ajoutez un bouton à votre scène comme indiqué sur la Figure 21.

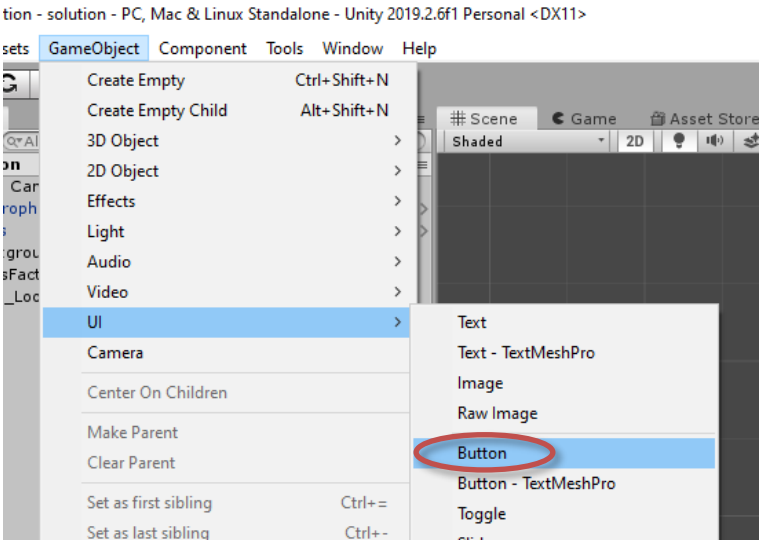

Figure 21

Observez la structure des GameObjects créé automatiquement par Unity. Le bouton a été ajouté à la scène à l'intérieur d'un **canevas** et le GameObject **EventSystem** a également été ajouté. Ne supprimez pas ce GameObject sans quoi les évènements de clic sur le bouton ne seront plus traités.

Modifiez le texte du bouton pour afficher le libellé : **Pop virus**.

Ajoutez une fonction dans le système **VirusFactory** qui prend en paramètre le nombre de virus à créer à des positions aléatoires. N'oubliez pas d'abonner les nouveaux virus créés afin qu'ils soient pris en compte par FYFY. La signature de la fonction doit respecter le format suivant :

```
40    public void popVirus(int amount)
41    {
42        // A vous de compléter...
43    }
```

Figure 22

Toute fonction définie dans un système comme **publique**, **void** et acceptant au maximum un paramètre de type **int**, **float**, **bool** ou héritant de **UnityEngine.Object** est automatiquement détectée par FYFY qui ajoute une callback dans le **wrappers** du système permettant d'y faire référence dans l'interface d'Unity.

Pour lier le bouton avec la fonction définie dans le système vous devez donc passer par le MainLoop. Sélectionnez le bouton, identifiez le composant **Button**, dans l'évènement **OnClick()** cliqué sur le « **+** », glisser le GameObject **Main_Loop** dans la cible de l'évènement, dans la liste déroulante **No function** sélectionnez le wrapper du système **VirusFactory** et sélectionnez la fonction **popVirus**. Il ne vous reste plus qu'à indiquer la valeur à passer en paramètre, 1 000 par exemple.

À l'exécution, à chaque fois que vous cliquez sur le bouton Pop virus, 1 000 nouveau virus doivent apparaitre dans la scène.

3.9 Pour aller plus loin

Améliorez le projet actuel pour ajouter une nouvelle mécanique de jeu, voici quelques propositions mais sentez-vous libre de développer votre propre idée :

- Comptabiliser le nombre de virus mangé par le macrophage et détruire le macrophage au bout d'un certain temps, ce seuil pourra être définit dans l'éditeur d'Unity. Le jeu consiste à manger le plus possible de virus dans le temps imparti.
- Ajoutez un système de température, plus il y a de virus plus la température augmente, plus la température augmente plus les virus se multiplient lentement. Si la température dépasse 45°C le joueur perd la partie.
- Définir les virus intelligents qui s'éloignent du macrophage quand il se rapproche trop d'eux mais qui se déplacent plus lentement que le macrophage.
- …