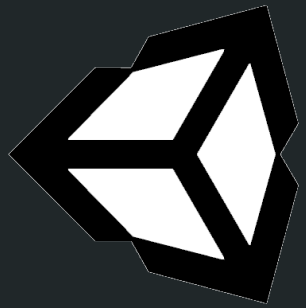
Introduction à Unity

Scripting



Un script : c’est quoi ça ?

- Un Script doit isoler une fonctionnalité du code par exemple : déplacement du joueur, gestion de la caméra, gestion des points de vie.
  Plus un script est petit meilleur sera le maintient du code.

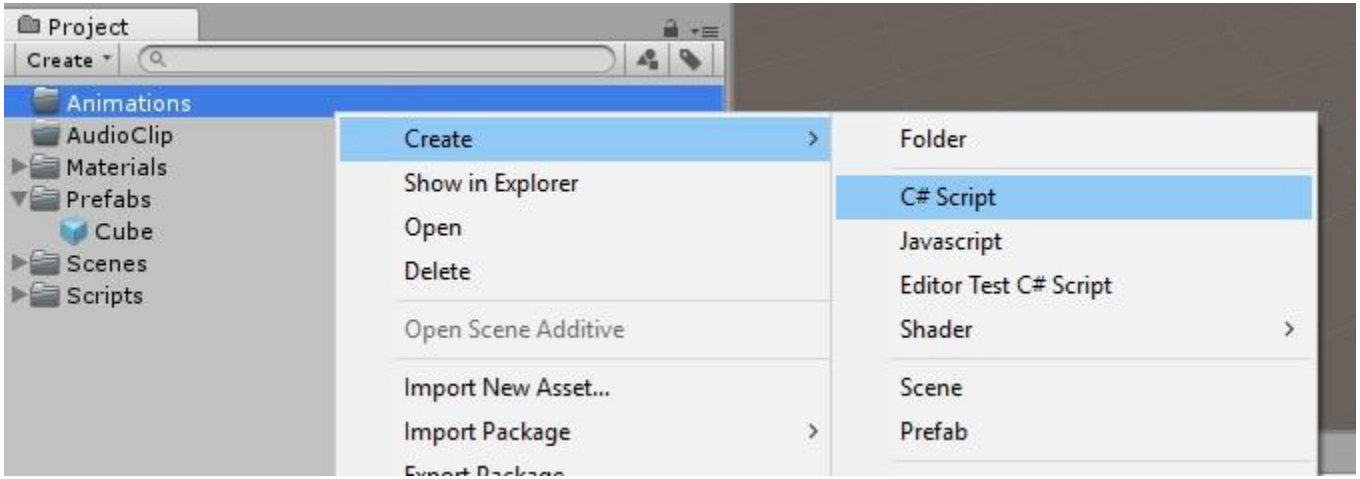
- Un script est un composant, il se glisse simplement sur le gameobject qui l’utilise

- Un script se doit donc d’être assez générique et ne pas référencer un objet par son nom absolue ou autre.

- MonoBehaviour : Un script héritant de la classe monobehaviour sera un composant attachable a un gameobject

Créer un nouveau Script

- Clic droit dans la fenêtre projet ➜ Create ➜ C# Script

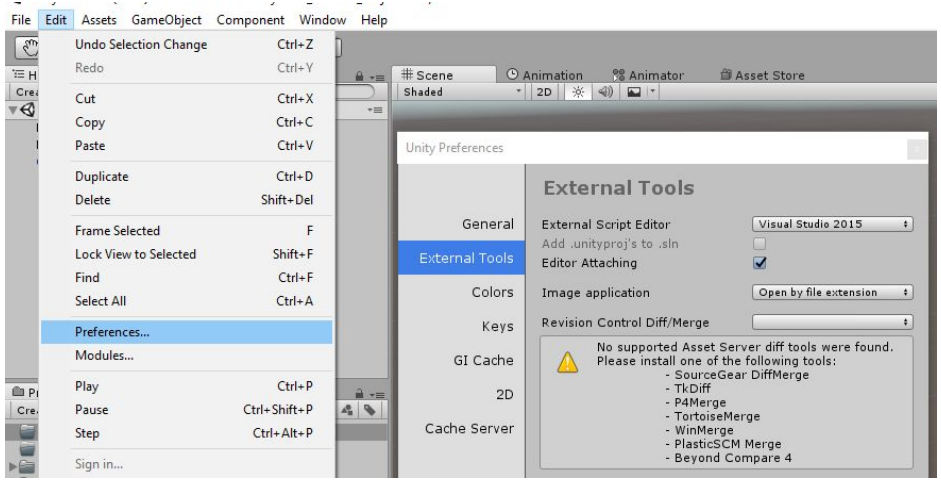


Optionnel : linker visual studio

Edit ➔ Preferences ➔ External tools

Si visual studio n'apparaît pas dans la liste faire browse et chercher devenv.exe

C:\Program Files (x86)\Microsoft Visual Studio 14.0\Common7\IDE\devenv.exe



Script : Variable SerializeField

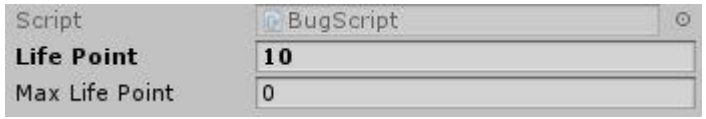
- Déclarer une variable privé et SerializeField ou une variable public si le type est compatible ➜ Créer un affichage automatique dans unity

- Permet de changer rapidement et visuellement la valeur des variables avant l'exécution ou pendant.

- Permet surtout de rendre les composants génériques ! Toujours utiliser ça !

```
[SerializeField]
private int LifePoint;

public int MaxLifePoint;
```



Script : Les types sérializable

- Toutes les classes héritant de UnityEngine.Object, par exemple GameObject, Component, MonoBehaviour, Texture2D, AnimationClip.

- Toutes les données de base de type int, string, float, bool.

- Vector2, Vector3, Vector4, Quaternion, Matrix4x4, Color, Rect, LayerMask.

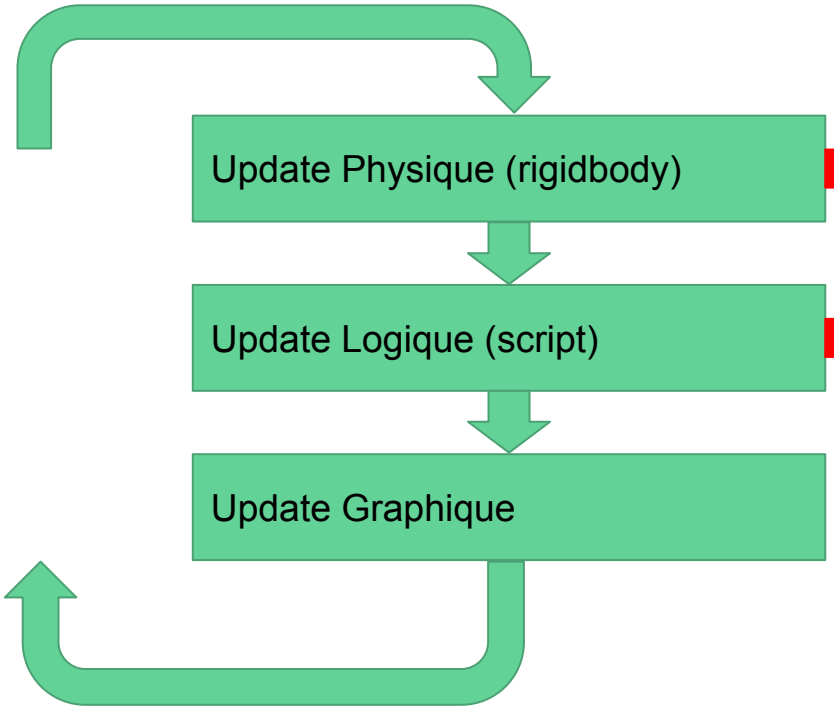
- Les tableaux de type serializable

- Les listes de types serializable

- Enums et/ou Structs

https://docs.unity3d.com/ScriptReference/SerializeField.html

Un moteur de jeu vidéo comment ça fonctionne ?



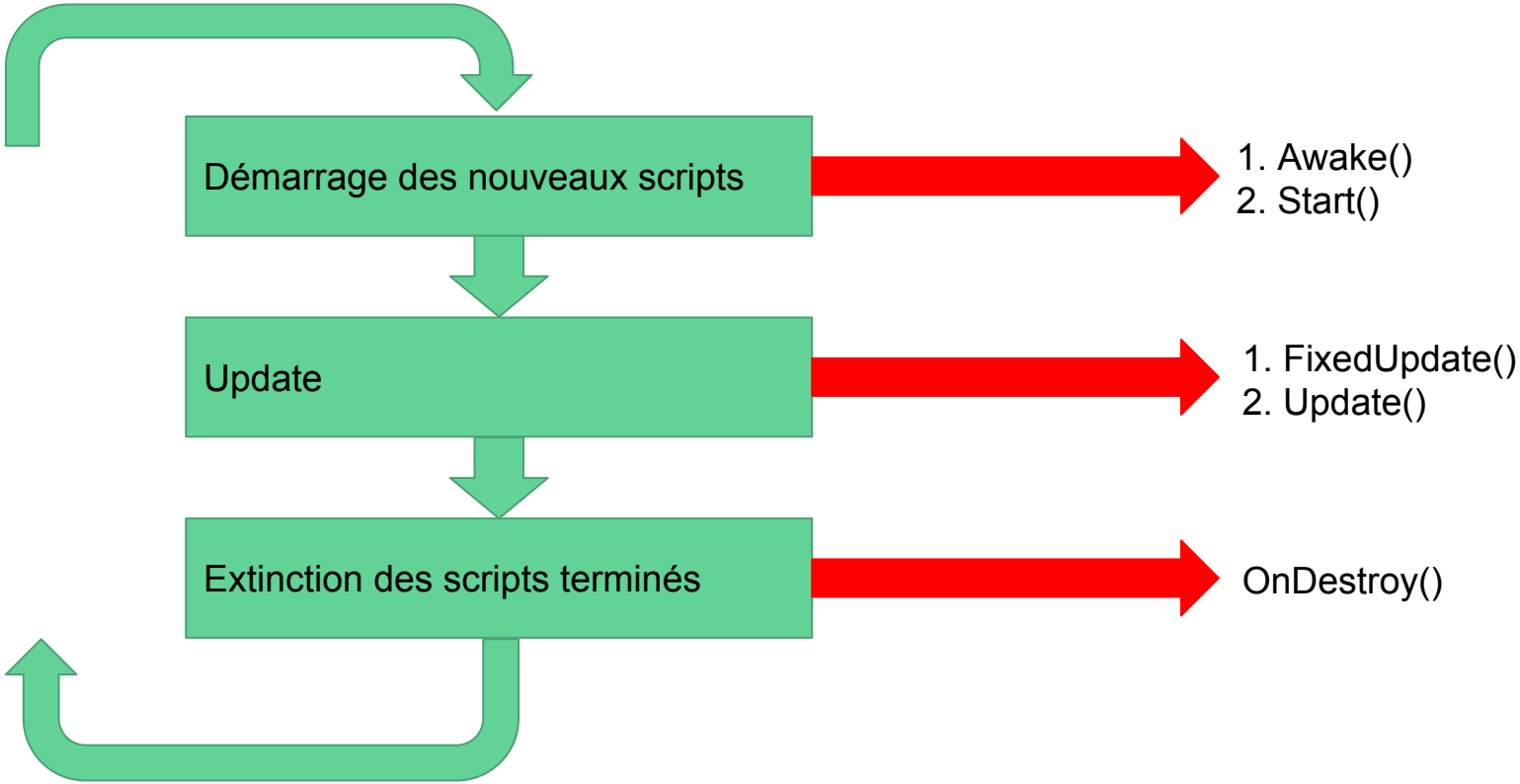
Qui se traduit dans
Unity par :

Fonction FixedUpdate

Fonction Update

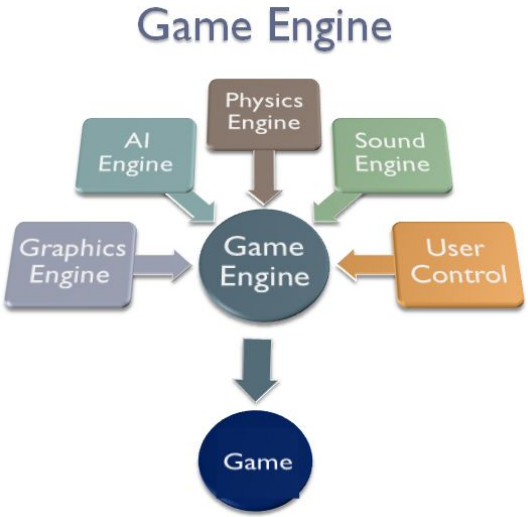
Attention pour des problèmes de performances l'update physics n'est pas forcément appelée toutes les frames.

Un moteur de jeu vidéo comment ça fonctionne ? (part2)



Optionnel : Fonctionnement Unity en détails

https://docs.unity3d.com/Manual/ExecutionOrder.html



Concrètement à quoi ça nous sert ?

- Lorsque vous allez démarrer votre jeu, les scripts vont se mettre en marche et donc la boucle va se lancer

- Ainsi vous allez devoir coder vos fonctions au bon endroit

- Par exemple pour le déplacement du joueur ceci doit être fait dans l'Update()

- Pour initialiser des variables dans le Start()

- Mais attention il ne faut pas se limiter à ces fonctions car des traitements coûteux dans l'update amène : DES LAGS

Script : Instantiate

- Instantiate permet de créer dans la scène un objet à partir d'un prefab ou d'un autre objet

- Si vous devez faire spawner X ennemie a un endroit il suffit d'appeler X fois Instantiate.

- Très pratique pour prototyper ou dans le cas d'un objet qui a besoin d'être instancié souvent tout au long de la vie du jeu cependant il faut faire attention :

Script : Instantiate c’est pas que du bonheur

- Instantiate est coûteux en ressources il va créer une allocation mémoire forte en copiant tous les éléments du gameobject pour en créer un nouveau

    - Un Instantiate a chaque frame dans l’Update tuera votre jeu !

    - De plus un instantiate ne permet pas de connaître l’objet au niveau scène facilement ➜ Ce qui vous oblige à utiliser Find !

    - Pour finir, si vous voulez récupérer un composant de cet objet il va falloir faire un GetComponent<>()

Les 3 fonctions diaboliques

1. FIND() : cette fonction est **EXTRÊMEMENT** coûteuse car elle va parcourir toutes l'arborescence des objets pour comparer leur noms !

2. GetComponent<>() : de même elle est coûteuse en temps de calcul et pas facilement généralisable (si vous récupérer un composant dans le script X vous devrez le reprendre via getcomponent() dans le script Y)

3. Instantiate() : Bien que pratique même si coûteuse, cette fonction entraine l'utilisation des deux autres précédente c'est pourquoi il vaut mieux l'éviter

Mais alors comment palier à tout ça ?

LA solution : l’object pooling

- Cette technique veut que tous les objets que vous allez utilisez soit créé au début du jeu (voir même avant avec Unity)

- Vos scripts devront avoir leur propre fonction de “mise en service” des objets

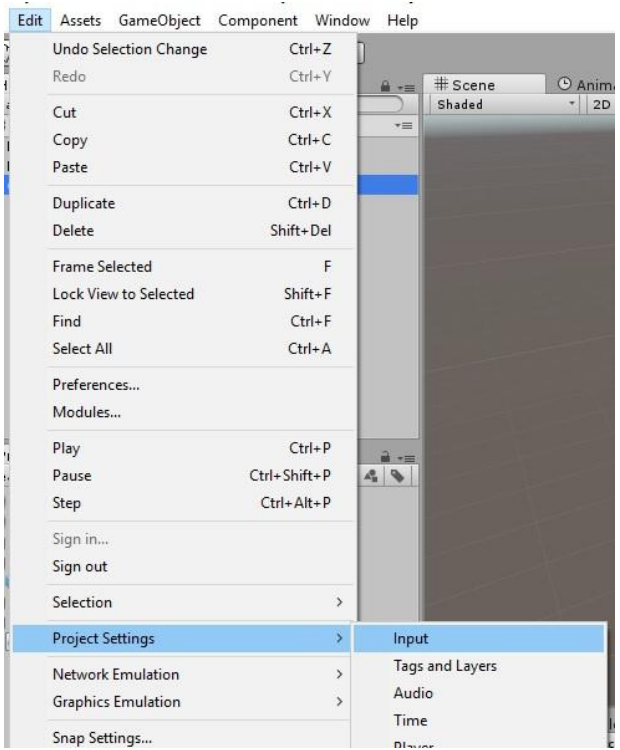
- Ainsi vous pouvez déjà pré-enregistré les composants ainsi que les objets

- De plus, vous ne faite que changer des valeurs qui sont déjà en mémoire RAM et peut être en cache !

Le joueur préférera toujours un chargement un peu long à des lags en jeux !

Les Inputs utilisateur

- A faire dans la fonction Update()
- La class s’appelle Input
- Si vous utiliser l’input manager
  - Input.GetButtonDown(“le nom de mon input”)
  - Input.GetButtonUp(“le nom de mon input”)
- Si vous n’utiliser pas l’input manager
  - Input.GetKeyDown(KeyCode.A)
  - Input.GetKeyUp(KeyCode.A)



Déplacement du joueur

- Ajouter un flaot (SerializedField) pour définir la vitesse

- La formule est :
  - La direction * la vitesse * le temps d'une frame

- Pourquoi le temps d'une frame ?
  Pour lisser le déplacement en fonction de la durée de la dernière frame (si la frame X est plus longue que la frame X+1 alors le joueur doit moins avancer lors de la frame X+1)

- La direction se trouve avec la classe Vector (exemple: Vector.forward pour avancer)

Maintenant : A TOI DE JOUER !