# Récupérer un projet GitHub avec Fork

Pour récupérer un projet hébergé sur GitHub, nous utilisons un outil appelé Fork. Voici les étapes à suivre :

1. Télécharger Fork : C'est une application qui vous permet de gérer les projets GitHub plus facilement.

2. Cloner le projet :

- Ouvrez Fork.

- Allez dans "Files" et choisissez "Clone".

- Allez sur GitHub et trouvez le projet que l'on vous a donné. Cliquez sur le gros bouton vert "Code" et copiez le lien HTTPS.

- Collez ce lien dans Fork.

- Choisissez l'endroit où vous voulez sauvegarder le projet sur votre ordinateur.

- Une fois téléchargé, ouvrez le projet dans Unity.

# Installer Unity 2023.2.20f

Pour continuer, vous devez installer la version 2023.2.20f de Unity. Ensuite :

1. Ouvrez le projet que vous venez de télécharger avec Unity.

# Ouvrir la scène "SampleScene"

Dans Unity, une scène est comme un niveau ou une page de votre projet. Pour ouvrir la scène principale de ce projet :

1. Allez dans le dossier Assets, puis ouvrez le sous-dossier Scenes.

2. Double-cliquez sur le fichier SampleScene pour l'ouvrir.

# Ajouter un cube à la scène

Vous allez maintenant ajouter un objet simple, un cube, à votre scène :

1. Dans la fenêtre Hiérarchie (la liste des objets dans votre scène), faites un clic droit pour ouvrir un menu.

2. Sélectionnez 3D Object puis Cube. Le cube apparaît alors dans votre scène.

# Introduction aux branches GitHub

Les branches sur GitHub sont comme des versions différentes de votre projet. Elles vous permettent de travailler sur des modifications sans affecter le projet principal. Voici comment elles fonctionnent :

1. Ouvrir les branches : Dans Fork, vous verrez un menu à gauche avec "Branches" et "Remotes".

- Si c'est votre première fois, double-cliquez sur la branche dans "Remotes" pour l'importer sur votre ordinateur.

- Si elle est déjà disponible dans "Branches", il suffit de double-cliquer dessus pour la charger.

2. Travailler avec les branches : Si vous avez des modifications en cours mais que vous voulez changer de branche, vous avez plusieurs options :

- Annuler les changements.

- Garder les changements pour plus tard avec une fonctionnalité appelée Stash (on la verra plus en détail plus tard).

- Enregistrer les changements et les envoyer sur GitHub.

# Vérification du système de branches

Nous allons tester les branches en modifiant un objet dans la scène :

1. Ouvrez le projet en étant sur une nouvelle branche.

2. Changez le cube en une sphère.

3. Envoyez les changements sur GitHub (on appelle cela "push").

4. Amusez-vous à basculer entre la branche principale ("Main") et la nouvelle branche.

5. Vous verrez que juste en changeant de branche, les modifications dans la scène (cube/sphère) changent aussi.

# Création d'un script C#

Dans Unity, les scripts permettent de programmer le comportement des objets. Voici comment créer un script :

1. Dans le dossier Assets, faites un clic droit et sélectionnez Créer > C# Script.

2. Donnez un nom à votre fichier.

3. Double-cliquez sur le fichier pour l'ouvrir dans l'éditeur.

4. Analysons le script :

- Il y a un nom de classe, qui représente le script.

- Il y a aussi deux méthodes : Start (elle est appelée une fois quand l’objet est créé dans la scène) et Update (elle est appelée à chaque frame, c'est-à-dire plusieurs fois par seconde).

# Faire tourner un cube dans la scène

Vous pouvez faire tourner un objet dans Unity de deux manières :

1. Manuellement : Dans la scène, sélectionnez le cube, puis utilisez les gizmos arrondis (raccourci touche E) pour le faire tourner.

2. Par le code : Utilisez un script pour contrôler la rotation.

# Code pour faire tourner un cube au lancement du jeu

Nous avons essayé de faire tourner le cube de 45° sur l'axe X au début du jeu. Voici ce que nous avons appris :

1. La méthode `transform.rotation` dans Unity utilise un type de données appelé quaternion, qui nécessite 4 valeurs : x, y, z, w. Cela peut être complexe à utiliser.

2. Nous avons trouvé une autre méthode plus simple : **`transform.Rotate()`**, qui n'a besoin que des angles en x, y et z.

3. Par exemple, le code **transform.Rotate(45, 0, 0);** mettra le cube à 45° sur l'axe X dès que le jeu commence.

# Faire tourner le cube à chaque frame

Nous avons ensuite ajouté **transform.Rotate(45, 0, 0);** dans la méthode Update, ce qui a fait tourner le cube à chaque frame (image par seconde). Mais il y avait un problème : le cube tournait à des vitesses différentes selon les PC.

# Utilisation de `Time.deltaTime`

Pour corriger ce problème de vitesse, nous avons appris à utiliser `Time.deltaTime`. Cela permet de rendre la rotation du cube uniforme sur tous les PC. Voici le code corrigé :

**transform.Rotate(45 \* Time.deltaTime, 0, 0);**

Maintenant, le cube tourne beaucoup plus lentement, mais à la même vitesse sur tous les ordinateurs.

# Faire tourner le cube sur plusieurs axes

Nous avons ensuite modifié la rotation pour qu'elle se fasse sur plusieurs axes. Voici comment :

1. En utilisant

**transform.Rotate(45 \* Time.deltaTime, 20 \* Time.deltaTime, 30 \* Time.deltaTime);** , on a pu faire tourner le cube sur les axes X, Y, et Z.

2. Pour simplifier, nous avons utilisé un Vector3 :

**transform.Rotate(new Vector3(45, 20, 30) \* Time.deltaTime);**

Cela permet de faire tourner le cube uniformément sur tous les axes et sur tous les PC.