1

Introduction à Unity

Du moteur temps réel à votre première scène interactive

Étienne Peillard
IMT Atlantique

NOTES DE COURS

Introduction au logiciel Unity et aux principes d'une application interactive. Public : étudiants à l'aise en informatique, débutants avec Unity. Proposition : 45 à 60 minutes, avec une courte démonstration et les questions. Le cours présente les concepts avant leur mise en pratique.

Les objectifs du cours

Expliquer

Relier entrées, simulation et rendu.

Se repérer

Distinguer ressources, scène et vues de travail.

Construire

Associer un comportement aux bons composants.

> Fil rouge : un cube que l'on voit, qui tombe, puis avec lequel on interagit.

NOTES DE COURS

À la fin, les étudiants doivent pouvoir expliquer pourquoi un cube visible ne tombe pas forcément, où se trouvent ses réglages et comment tester leur modification. Ce cours pose un modèle mental pour comprendre l'éditeur et le moteur.

Quatre repères pour commencer

1

Comprendre le moteur

Entrées · simulation · rendu

2

Lire l’éditeur

Projet · scènes · fenêtres

3

Composer une scène

GameObjects · composants · prefabs

4

Développer et tester

Scripts · diagnostic · build

NOTES DE COURS

Présenter le parcours et annoncer le même exemple de cube dans les quatre parties. Les cartes permettent de rejoindre chaque partie.

Diapositive 4 / 32

Diapositive en ligne ↗

Partie 1 sur 4

1

Comprendre le moteur

NOTES DE COURS

Cette partie développe les repères nécessaires pour comprendre et utiliser Unity.

Diapositive 5 / 32

Diapositive en ligne ↗

1. Comprendre le moteur

Un moteur orchestre une application

Entrées et logique

Lire une action et décider de ses effets.

Simulation

Faire évoluer objets, collisions et animations.

Image et son

Restituer l’état du monde à l’utilisateur.

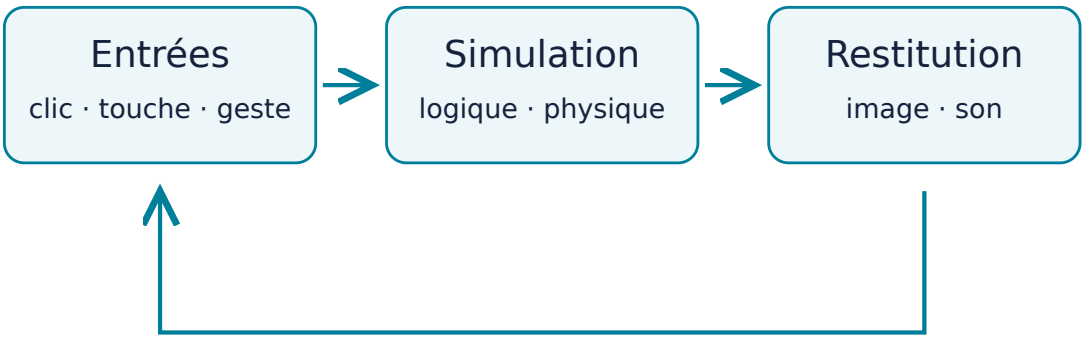
> Unity réunit un éditeur de création et un moteur d’exécution.

NOTES DE COURS

Un moteur de jeu fournit des services réutilisables pour construire des applications interactives, y compris des simulations ou des expériences de RV. Le moteur coordonne ces services ; les règles de votre application restent à définir. L’IA peut contribuer aux décisions et à la navigation, mais toute logique d’interaction n’est pas de l’IA.

Une boucle entre action et perception

1. **Agir**
   Un clic, une touche, un geste.
2. **Mettre à jour**
   La logique transforme l’état du monde.
3. **Restituer**
   Une nouvelle image et un retour sonore.



L’utilisateur perçoit le résultat et agit à nouveau.

Schéma de principe

> Temps réel : produire une réponse assez vite pour permettre l’interaction.

NOTES DE COURS

Exemple : un clic demande de lancer un cube ; le script applique une impulsion ; la simulation calcule son déplacement et les contacts ; le rendu montre le résultat. L’utilisateur observe puis agit à nouveau. Les entrées, la physique, le rendu et l’audio ne partagent pas nécessairement la même cadence.

Des services qui travaillent ensemble

Service	Rôle	Exemple du cube
Rendu	Produire l’image depuis une caméra	Afficher sa surface et son ombre
Physique	Simuler mouvement et contacts	Le faire tomber sur le sol
Audio	Jouer et spatialiser les sons	Entendre un impact
Entrées	Interpréter les actions	Déclencher le lancer

NOTES DE COURS

Le rendu prépare des données côté CPU et utilise le GPU via des API graphiques. La physique emploie des modèles et des approximations : un objet visuellement détaillé peut avoir un collider simple. Ces services n’impliquent pas que toutes les fonctionnalités soient actives automatiquement sur tous les objets.

Pourquoi Unity dans ce parcours ?

Un éditeur visuel

Assembler une scène et inspecter ses objets.

Des scripts C#

Ajouter les règles propres à l’application.

Des cibles variées

Construire un exécutable adapté à une plateforme.

> D’abord une scène 3D sur écran ; la RV ajoutera suivi, affichage et interactions spécifiques.

NOTES DE COURS

Unity permet d’aborder conjointement la création de scènes et la programmation de comportements. Son intérêt pédagogique est le passage rapide entre édition, programmation et test. Le choix d’un moteur dépend aussi de la cible, des compétences, des dépendances et

des conditions de licence. Le multiplateforme ne supprime pas les adaptations ni les tests sur le matériel cible.

Diapositive 9 / 32

Diapositive en ligne ↗

Partie 2 sur 4

2
Lire l’éditeur

NOTES DE COURS

Cette partie développe les repères nécessaires pour comprendre et utiliser Unity.

Diapositive 10 / 32

Diapositive en ligne ↗

2. Lire l’éditeur

Trois outils, trois moments

Unity Hub

Gérer les versions et ouvrir les projets.

Unity Editor

Créer, configurer et tester l’application.

Application construite

Exécuter le résultat sur la machine cible.

NOTES DE COURS

Les exemples sont présentés avec Unity 6.3 LTS et le pipeline Universal Render Pipeline (URP). Un projet doit être ouvert avec une version et des packages compatibles. Le Hub n’est pas l’éditeur. Le projet est le dossier de travail ; le build est le résultat construit pour une plateforme. Les modules nécessaires à cette plateforme doivent être disponibles.

Diapositive 11 / 32 Diapositive en ligne ↗

2. Lire l’éditeur

Projet, asset, scène, instance

Notion	Ce qu’elle désigne	Exemple
Projet	Ressources et réglages de l’application	Le dossier d’une application
Asset	Ressource enregistrée dans le projet	Matériau, script, scène, modèle 3D
Scène	Ensemble organisé de GameObjects	UnityBasics
Instance	Un exemplaire utilisé dans une scène	Un cube placé dans la scène

> Project montre les ressources ; Hierarchy montre les objets de la scène ouverte.

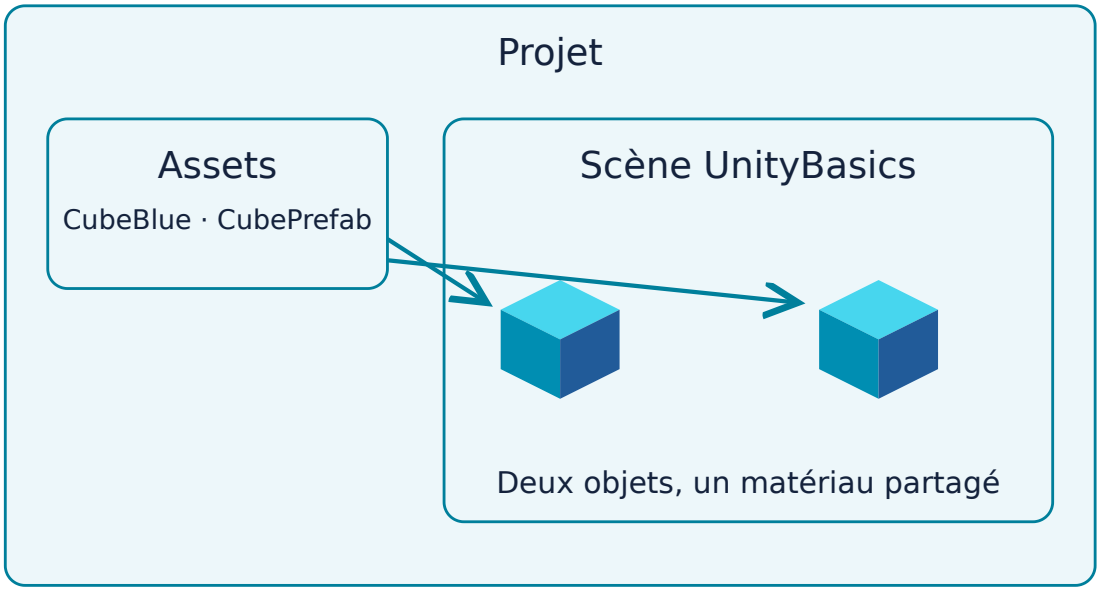
NOTES DE COURS

Une scène est elle-même un asset. Le dossier Assets contient les ressources ; Packages et ProjectSettings participent aussi au projet. Déplacer une instance ne déplace pas ses voisines. Modifier un matériau partagé peut en revanche changer tous les objets qui l’utilisent.

Diapositive 12 / 32 Diapositive en ligne ↗

2. Lire l’éditeur

Un matériau partagé, deux objets distincts

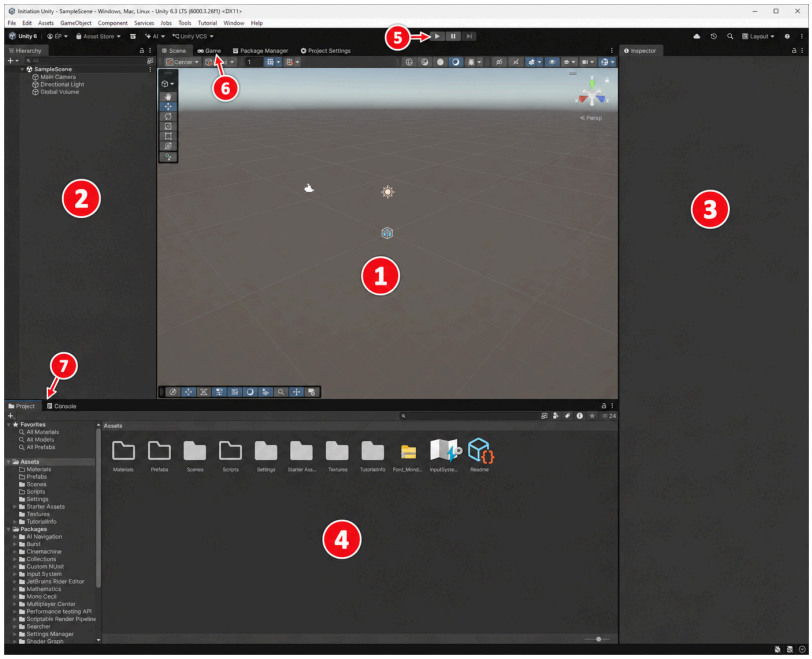


Déplacer un cube ne déplace pas l’autre ; modifier leur matériau partagé change les deux.

NOTES DE COURS

Faire distinguer l’asset de matériau, le prefab et les instances dans la scène. Les flèches indiquent une référence à une ressource, pas une relation parent-enfant.

Lire l’espace de travail



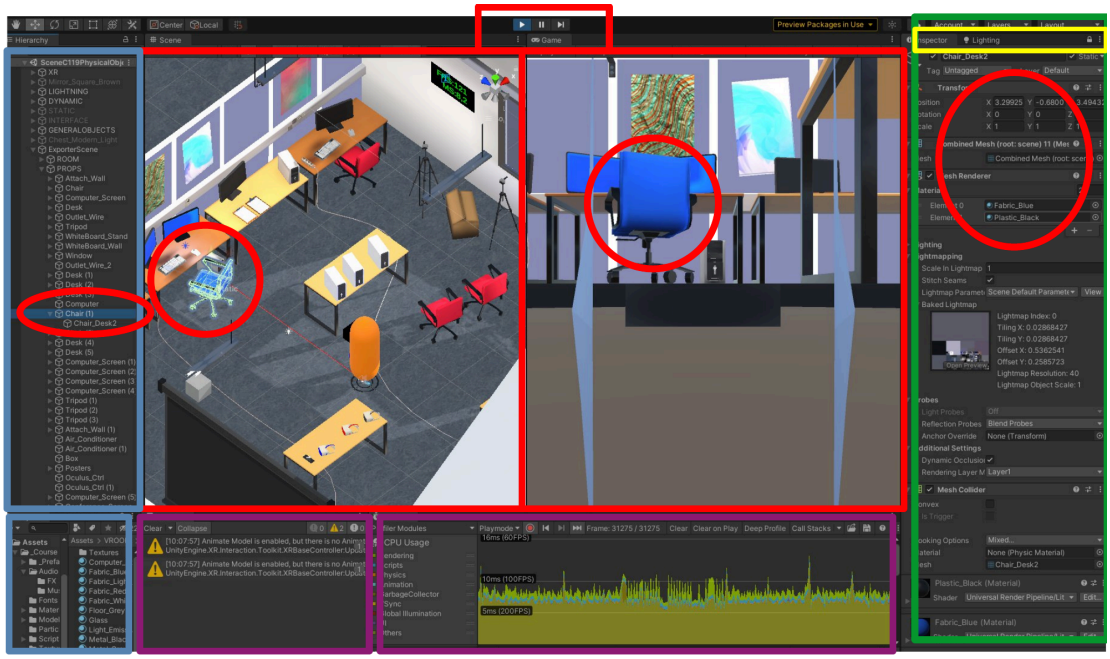
Unity 6.3 — retrouver les fenêtres par leur nom ; leur disposition est personnalisable.

NOTES DE COURS

Démonstration : sélectionner le même cube dans Hierarchy puis dans Scene et observer Inspector. Montrer ensuite l’asset de matériau dans Project. Faire verbaliser la différence entre la ressource et l’objet de scène. Repérer Scene, Game, Hierarchy, Inspector, Project et les commandes Play en haut de l’éditeur.

2. Lire l’éditeur

Un même objet, plusieurs points de vue



Support initial (2023) — suivre la chaise de Hierarchy à Inspector.

NOTES DE COURS

La capture du support initial montre une chaise : son nom est sélectionné dans Hierarchy, ses contours apparaissent dans Scene, son rendu est visible dans Game et ses composants dans Inspector. La vue Scene et la caméra Game n’ont pas le même point de vue. Les cadres colorés proviennent du support initial. Les fonctions de ces fenêtres se retrouvent dans Unity 6.3.

2. Lire l’éditeur

Les fenêtres à retrouver

Scene / Game

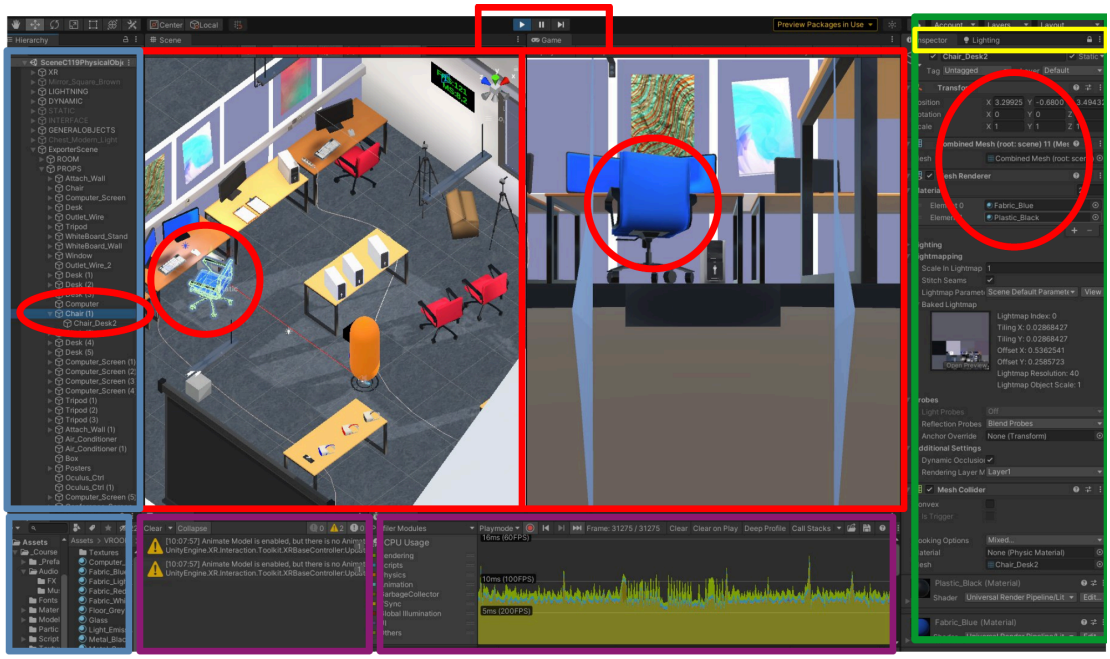
Construire le monde / voir depuis la caméra.

Hierarchy / Inspector

Sélectionner l’objet / régler ses composants.

Project / Console

Trouver une ressource / lire les erreurs.



Support de 2023 : mêmes rôles, disposition personnalisable.

NOTES DE COURS

Scene est une vue de travail indépendante de la caméra de jeu. Déplacer le point de vue dans Scene ne déplace pas automatiquement la caméra de l’application. L’Inspector affiche la sélection courante, qui peut être un GameObject ou un asset. La Console sert au diagnostic, le Profiler à mesurer les coûts d’exécution.

2. Lire l’éditeur

Construire, tester, arrêter

Edit

Configurer les objets puis enregistrer la scène.

Play

Observer le comportement et tester les entrées.

Stop

Revenir à l’édition et vérifier ce qui est conservé.

> Arrêter Play avant de modifier durablement la scène, puis sauvegarder.

NOTES DE COURS

Les modifications des objets de la scène effectuées en Play sont normalement annulées à l’arrêt. Cela ne constitue pas une protection de tous les fichiers : des changements d’assets ou des écritures de code peuvent persister. Démonstration : déplacer un cube en Play, arrêter

et constater son retour. Cliquer dans Game pour lui donner le focus avant de tester les entrées.

3
Composer une scène

NOTES DE COURS

Cette partie développe les repères nécessaires pour comprendre et utiliser Unity.

Un GameObject se construit par composition

GameObject

Le support : un nom, un état actif, des composants.

Transform

Position, rotation et échelle dans la hiérarchie.

Autres composants

Géométrie, affichage, collisions ou comportement.

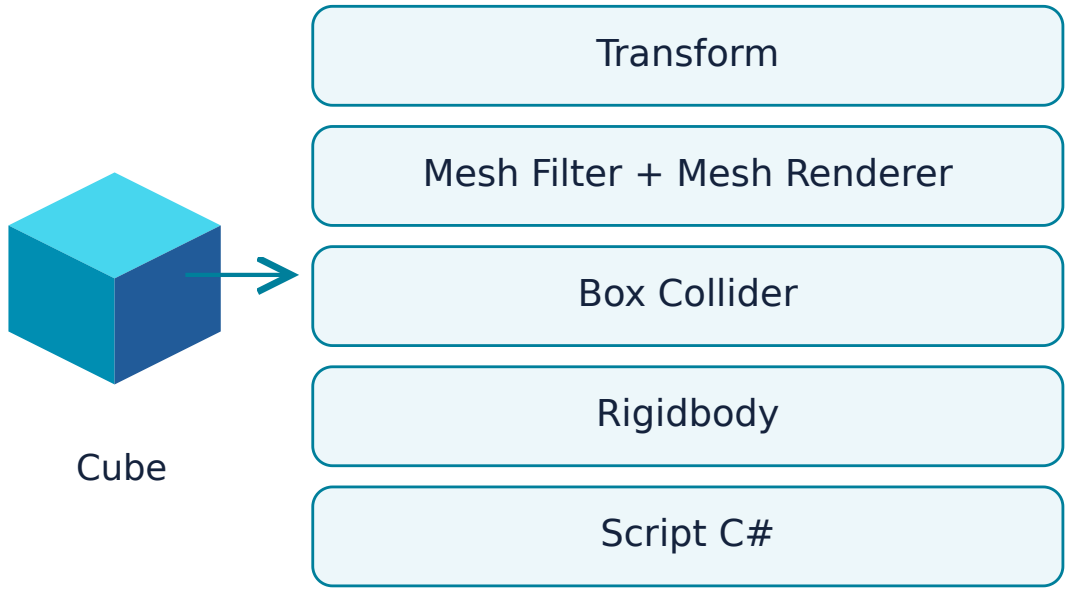


Schéma de principe

> Question utile : quel composant fournit la fonction que je cherche ?

NOTES DE COURS

Une caméra, une lumière et un cube sont des GameObjects avec des composants différents. Un GameObject vide n’a pas nécessairement de représentation visible. Dans une scène 3D, Transform décrit la relation avec le parent ; les propriétés locales et globales ne sont pas interchangeables. Les objets d’interface utilisent RectTransform, une spécialisation de Transform.

Positionner dans un repère

Position

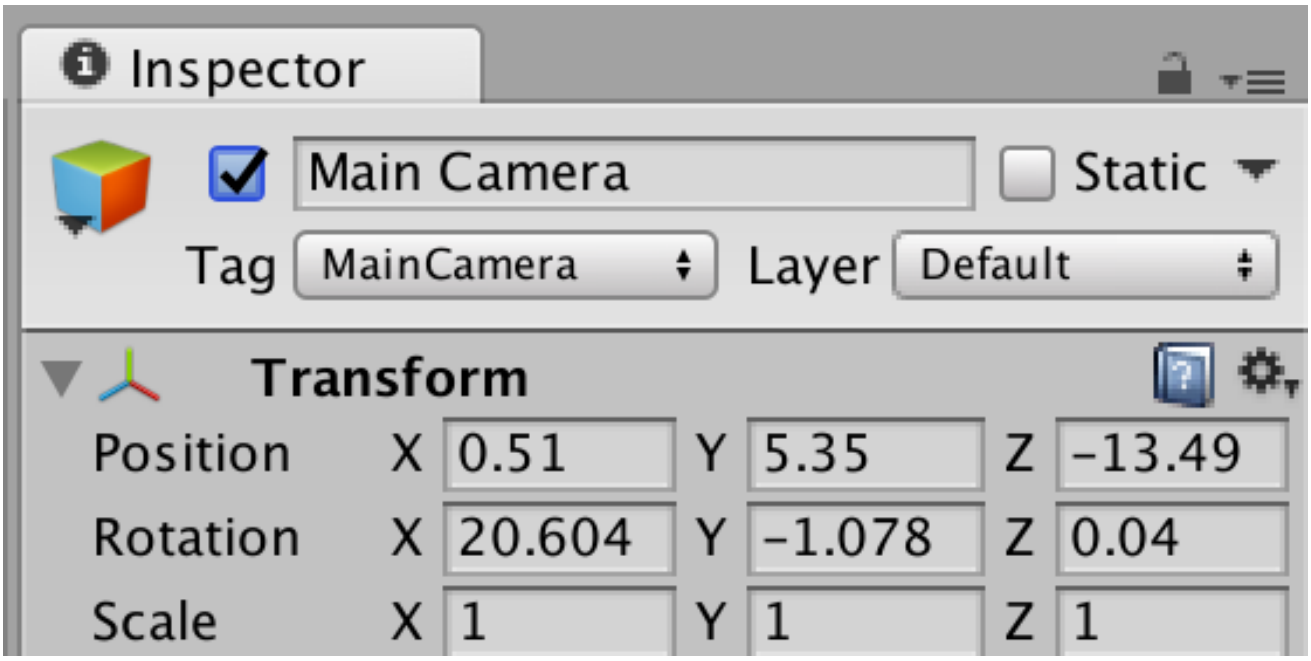
Situer un objet par rapport à son parent.

Rotation

Orienter ses axes et sa direction.

Échelle

Modifier ses dimensions selon les axes.



Transform d’une caméra — position, rotation et échelle (interface historique).

> Déplacer la vue Scene, déplacer un objet et déplacer la caméra sont trois actions distinctes.

NOTES DE COURS

Dans la convention habituelle d’Unity : X horizontal, Y vers le haut, Z vers l’avant. Dans une scène physique à échelle réaliste, on utilise généralement une unité comme environ un mètre. Une position locale (0, 0, 0) place l’objet à l’origine de son parent, pas nécessairement à l’origine du monde. Déplacer un parent entraîne ses enfants. Local/Global change le repère du manipulateur, pas la structure de la hiérarchie.

3. Composer une scène

Un cube visible n’est pas encore un cube physique

Composant	Fonction
Mesh Filter	Référence la géométrie du cube
Mesh Renderer	Affiche cette géométrie avec un matériau
Box Collider	Décrit une forme pour les contacts
Rigidbody	Permet de simuler le mouvement physique

> Afficher, détecter les contacts et simuler le mouvement sont des fonctions séparées.

NOTES DE COURS

Un cube 3D créé depuis le menu comporte déjà un Transform, un Mesh Filter, un Mesh Renderer et un Box Collider. Il faut ajouter un Rigidbody pour le faire tomber avec la gravité. Pour simuler une chute : Use Gravity activé, Is Kinematic désactivé ; le sol possède un collider et reste fixe sans Rigidbody. Un renderer ne détecte pas les collisions.

Géométrie, matériau, shader, texture

Mesh

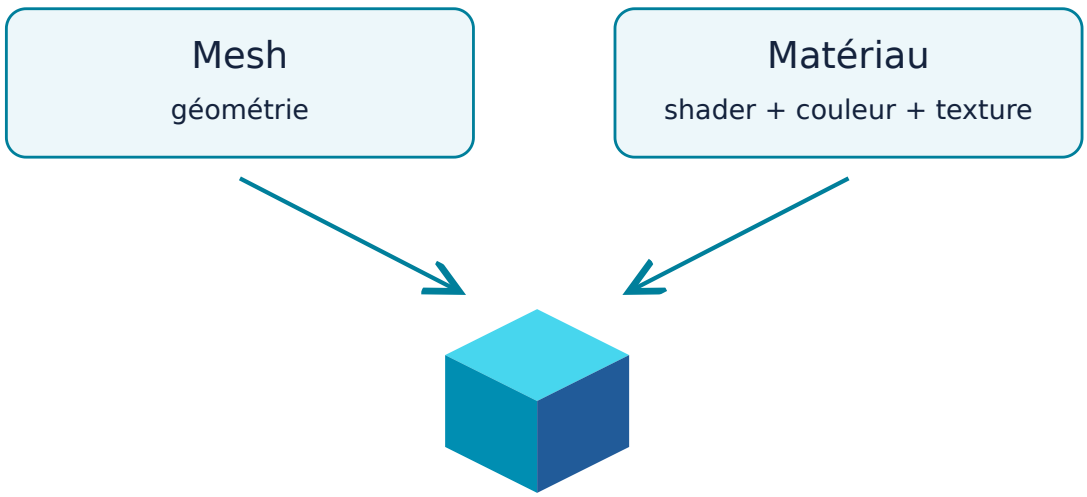
La forme : sommets et triangles.

Matériau

Des réglages d’apparence partagés entre objets.

Shader et textures

Un calcul de surface et les images qui l’alimentent.



Le Renderer combine forme et apparence.

Schéma de principe

> Choisir des matériaux compatibles avec le pipeline URP du projet.

NOTES DE COURS

Le Renderer utilise le matériau ; le matériau sélectionne un shader et fournit ses paramètres, dont éventuellement des textures. Par exemple, CubeBlue utilise URP Lit avec une couleur, et GroundGrid ajoute une texture dans Base Map. Une texture n’est donc pas un matériau. La lumière et la caméra influencent aussi l’image.

Un prefab, plusieurs instances

Le modèle

Un asset enregistre un objet, ses composants et ses enfants.

Les instances

Plusieurs exemplaires gardent un lien avec ce modèle.

Les overrides

Une propriété peut être modifiée sur un seul exemplaire.

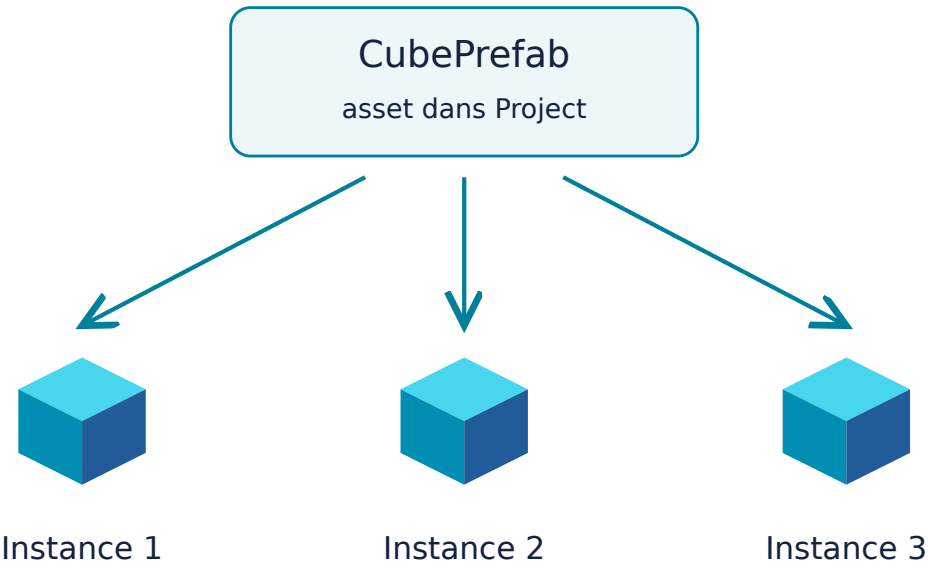


Schéma de principe

> Partager un matériau n’est pas la même chose que partager un prefab.

NOTES DE COURS

Fil rouge : douze cubes peuvent être des instances du même CubePrefab et partager CubeBlue. Modifier le prefab se propage aux instances sauf pour les propriétés surchargées. Apply reporte une modification au modèle ; Revert rétablit la valeur du modèle sur l’instance.

Vérifier ce que l’on applique avant Apply All. La position des cubes dans la scène reste propre à chaque instance.

Diapositive 23 / 32

Diapositive en ligne ↗

Partie 4 sur 4

4

Développer et tester

NOTES DE COURS

Cette partie développe les repères nécessaires pour comprendre et utiliser Unity.

Diapositive 24 / 32

Diapositive en ligne ↗

4. Développer et tester

Du fichier C# au comportement

1. **Écrire**
   Une classe dérive de MonoBehaviour.
2. **Compiler**
   Unity recharge le code enregistré.
3. **Attacher**
   Le script devient un composant sur un GameObject.

> Le script décrit le comportement ; chaque composant porte ses propres réglages.

NOTES DE COURS

Exemple : RotateCube.cs contient la classe RotateCube. Enregistrer dans l’éditeur de code puis revenir dans Unity. Corriger les erreurs de compilation avant d’essayer d’attacher le script. Un fichier présent dans Project ne suffit pas pour exécuter un comportement : il faut une instance du composant sur un GameObject actif et un composant activé pour recevoir Update.

Diapositive 25 / 32

Diapositive en ligne ↗

4. Développer et tester

Lire un premier composant

RotateCube.cs

```
using UnityEngine;

public class RotateCube : MonoBehaviour
{
[SerializeField] private float speed = 45f;

void Update()
{
transform.Rotate(0f, speed * Time.deltaTime, 0f);
}
}
```

RotateCube.cs — sur un objet de démonstration sans simulation physique.

> 45 degrés/seconde × durée de l'image = angle à appliquer sur cette image.

NOTES DE COURS

Lire de haut en bas : UnityEngine fournit les types Unity ; MonoBehaviour permet à Unity d'appeler les fonctions d'événement ; SerializeField rend le champ privé réglable dans Inspector ; Update est appelé à chaque image pour un composant actif ; transform désigne le Transform de l'objet porteur. speed est une vitesse angulaire en degrés par seconde. Time.deltaTime est la durée écoulée depuis l'image précédente : vitesse multipliée par durée donne un angle. Ce code illustre une animation de Transform ; ne pas l'utiliser pour piloter directement un Rigidbody dynamique.

Diapositive 26 / 32

Diapositive en ligne ↗

4. Développer et tester

Tous les calculs n'ont pas la même cadence

Start

Initialiser avant le premier Update du composant.

Update

Mettre à jour la logique liée aux images affichées.

FixedUpdate

Préparer les actions liées au pas de simulation physique.

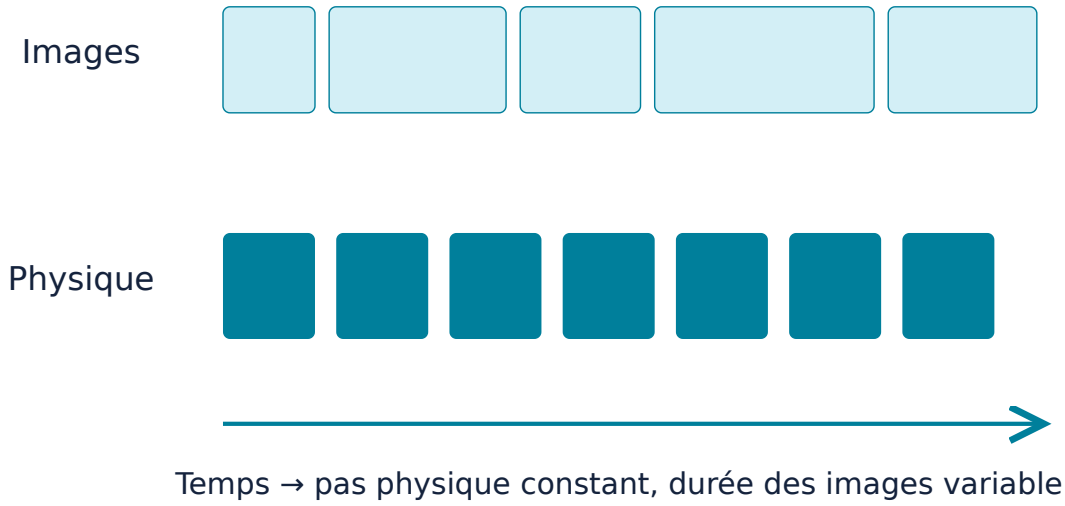


Schéma de principe

> Distinguer une vitesse, une variation par image et une force physique.

NOTES DE COURS

Unity appelle ces fonctions ; il ne faut pas écrire sa propre boucle infinie dans Update. En simulation physique automatique, FixedUpdate peut être appelé zéro, une ou plusieurs fois pour une image rendue, selon le temps accumulé. Le pas physique n'est donc pas la fréquence d'affichage. Pour les forces sur un Rigidbody, utiliser les API physiques au moment adapté ; ne pas multiplier systématiquement toutes les API par deltaTime.

Relier le script à la scène

Champ sérialisé

Exposer un paramètre ou une référence dans Inspector.

Référence assignée

Glisser l'objet ou le composant attendu dans le champ.

Composant recherché

Récupérer un composant existant avec GetComponent.

> Avant d'accuser le calcul, vérifier les références et les composants présents.

NOTES DE COURS

Exemple : un lanceur référence le prefab à instancier et la caméra qui donne la direction. GetComponent<Rigidbody>() cherche un composant sur le même objet ; il ne le crée pas et peut retourner null. Une NullReferenceException peut venir d'un champ non assigné ou d'un composant absent. Camera.main recherche une caméra activée portant le tag MainCamera : renommer un objet ne change pas son tag.

Importer une ressource sans perdre le fil

Package Manager

Gérer les fonctionnalités et leurs dépendances.

Assets importés

Examiner modèles, matériaux, prefabs et scripts.

Compatibilité

Vérifier version de Unity, pipeline et système d'entrées.

> Importer → vérifier → tester une scène minimale, avant d'intégrer au projet.

NOTES DE COURS

Un projet Universal 3D utilise URP et peut gérer les entrées avec le package Input System. Les ressources ou tutoriels fondés sur l'ancien UnityEngine.Input ne sont pas directement interchangeables avec ce choix. Un matériau rose peut signaler un shader absent ou incompatible ; vérifier Console et shader avant de conclure. Tester une ressource complexe dans un projet séparé peut aider à comprendre ses dépendances.

Diagnostiquer avec une hypothèse

Symptôme	Premier contrôle
Le cube traverse le sol	Colliders présents et configurations physiques
Le script ne fait rien	Compilation, composant attaché et activé
Une référence est null	Champ Inspector et objet/composant recherché

Symptôme	Premier contrôle
Les entrées ne répondent pas	Focus Game, Input System et configuration

> Observer → formuler une hypothèse → modifier une chose → vérifier.

NOTES DE COURS

Commencer par la première erreur pertinente de Console, lire son message et sa pile d’appels, puis reproduire sur une scène minimale. Un symptôme a plusieurs causes possibles : le tableau donne un premier contrôle, pas un diagnostic certain. Modifier une chose, tester et comparer à la prédiction. Le Profiler aide à mesurer un problème de performance après l’avoir reproduit.

Tester aussi en dehors de l’éditeur

1. **Configurer**
   Choisir la cible dans Build Profiles.
2. **Construire**
   Inclure les scènes nécessaires au démarrage.
3. **Exécuter**
   Tester le résultat sur la machine cible.

> Valider tôt un petit build, puis avancer par ajouts vérifiables.

NOTES DE COURS

Dans Unity 6, File > Build Profiles sert à préparer la construction. Vérifier la plateforme, les modules installés et la liste des scènes. Le succès de Play ne prouve pas que le build fonctionnera : les chemins de fichiers, les entrées et les performances peuvent différer. Livrer les fichiers produits ensemble, pas uniquement le fichier exécutable si celui-ci dépend de dossiers voisins.

4. Développer et tester

De la scène à l’application interactive

Assembler

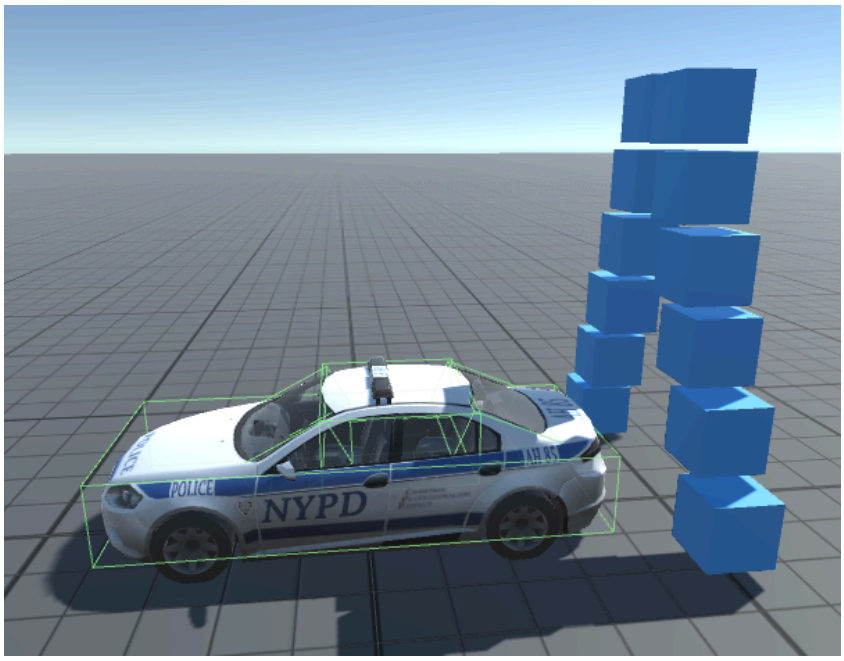
Sol, cubes, matériau partagé et prefab.

Faire agir

Gravité, rotation, contrôleur et lancer d’objets.

Vérifier

Console, comportement attendu et exécutable.



Une même scène combine géométrie, matériaux, physique et comportements.

> Identifier les objets, les composants et les effets attendus pour concevoir une interaction.

NOTES DE COURS

Une application interactive combine des objets, une apparence et des comportements. Commencer par une scène minimale, puis ajouter les fonctionnalités progressivement. Conserver des noms explicites, des dossiers organisés et des sauvegardes régulières. Vérifier le résultat en Play et dans une application construite.

4. Développer et tester

Retrouver une réponse fiable

Manuel Unity

Comprendre un concept et les réglages de l’éditeur.

API de script

Vérifier une propriété, une méthode et ses unités.

Version du projet

Vérifier la version de Unity et celle des packages installés.

Manuel Unity 6.3 ↗ API Unity 6.3 ↗ Cycle des scripts ↗

NOTES DE COURS

Support restructuré à partir de Unity_2023.pdf (43 pages : moteur temps réel, interface, programmation et démarche de conception) et IntroMoteur_FIL.pdf (10 pages : scène et sous-systèmes du moteur). Les répétitions de sommaire, les consignes historiques de rendu et les listes de technologies sans lien direct avec les concepts présentés ont été écartées. La présentation corrige l’assimilation de toute logique à de l’IA et distingue les cadences physique et graphique. Documentation officielle consultée le 26 octobre 2026. Les numéros des parties sont indépendants du parcours des cours de RV.