Unity3D - Débutant
Ajouter des ennemis et des projectiles

Credits & Licence

- This training
  - Author : Sébastien Yriarte, Tech'N'Smile
  - Google Slide Template : Slides Carnival
  - Photo : Unsplash

- Licence d'utilisation :
  - Creative Commons Attribution & Share Alike

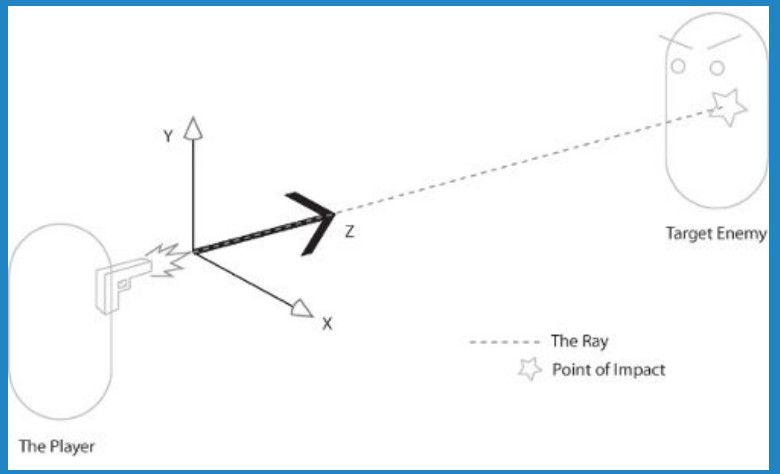
Sommaire

- Tirer en utilisant des raycasts
- Définir des cibles interactives
- Déplacer les ennemis
- Instancier des objets préfabriqués
- Tirer en instanciant des projectiles

1.

Tirer en utilisant des raycasts

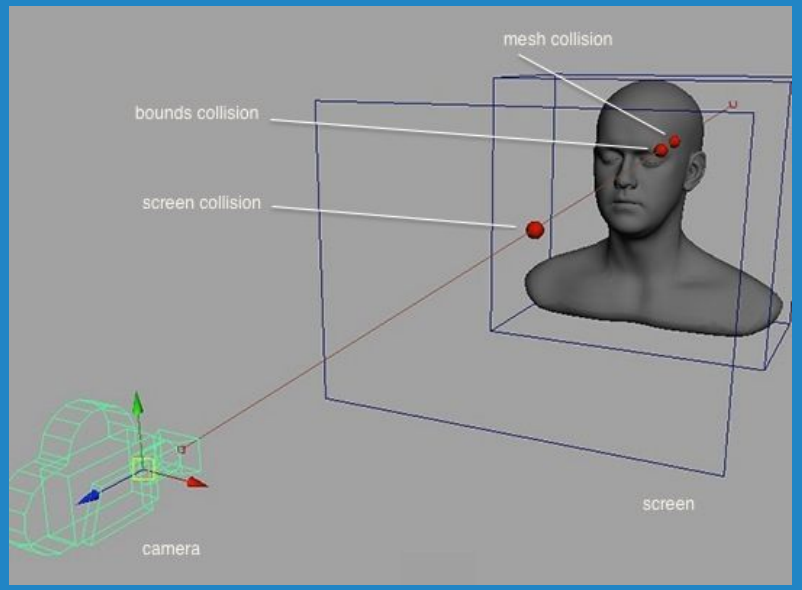
RayCasting & MousePicking
Tirer en utilisant des Raycasts
Instancier un objet sur le point d'impact
Créer un Heads Up Display (HUD)



RayCasting

Le raycasting crée un rayon et renvoie les objets sur le trajet et leurs points d'intersections.

https://en.wikipedia.org/wiki/Ray_casting



MousePicking

Le mousePicking est un raycasting spécifique renvoyant les objets en intersection avec un rayon partant du centre de l'écran

Unity fournit cette implémentation via la méthode **ScreenPointToRay()**

Exercice : Tirer en utilisant des raycasts

- Créer un nouveau script **RayCaster.cs**
- L'attacher à la caméra
- Récupérer l'instance de la caméra

```
6
7       private Camera camera;
8
9       // Use this for initialization
10      void Start () {
11
12          camera = GetComponent<Camera>();
13      }
14
```

GetComponent<T>

La méthode **GetComponent<T>** permet de récupérer un script attaché à un objet par son type. C'est une méthode générique dans le sens qu'elle peut récupérer le type d'objet spécifié entre les symboles "**<**" et "**>**"

Exercice : Tirer en utilisant des raycasts

- Modifier la méthode **Update()** pour créer un rayon sur l'appui de la souris

```
15      // Update is called once per frame
16      void Update () {
17          if (Input.GetMouseButtonDown(0)) {
18              Vector3 point = new Vector3(
19                  camera.pixelWidth / 2,
20                  camera.pixelHeight / 2, 0);
21
22              Ray ray = camera.ScreenPointToRay(point);
23
24              RaycastHit hit;
25              if (Physics.Raycast(ray, out hit)) {
26                  Debug.Log("Hit " + hit.point);
27              }
```

- Tester le jeu et observer la console une fois le clic de la souris utilisé

Exercice : Instancier un objet sur le point d'impact

- Modifier la méthode **Update()** pour exécuter une fonction gérant l'impact grâce aux coordonnées

```
17              if (Physics.Raycast(ray, out hit)) {
18                  StartCoroutine(CubeIndicator(hit.point));
19              }
```

- Créer la méthode **CubeIndicator(Vector3)**

```
23      private IEnumerator CubeIndicator(Vector3 pos) {
24          GameObject cube = GameObject.CreatePrimitive(
25              PrimitiveType.Cube);
26          cube.transform.position = pos;
27          yield return new WaitForSeconds(3);
28          Destroy(cube);
29      }
```

StartCoroutine

StartCoroutine gère l'exécution de commandes de manière asynchrone en insérant des temporisations grâce au mot clé **Yield**. Le type de retour d'une fonction appelée par StartCoroutine est **IEnumerator**

Exercice : Créer un Heads Up Display (HUD)

- Créer la méthode **OnGUI()** pour créer la mire de visée du FPS

```
void OnGUI() {
    int size = 40;
    float posX = (camera.pixelWidth / 2) - (size / 4);
    float posY = (camera.pixelHeight / 2) - (size / 2);
    GUI.Label(new Rect(posX, posY, size, size), "+");
}
```

Note : **OnGUI()** utilise l'ancien système de dessin pour gérer les interfaces utilisateurs et superposer des informations sur l'écran (score, mire de visée, inventaire …)

2.

Définir des cibles interactives

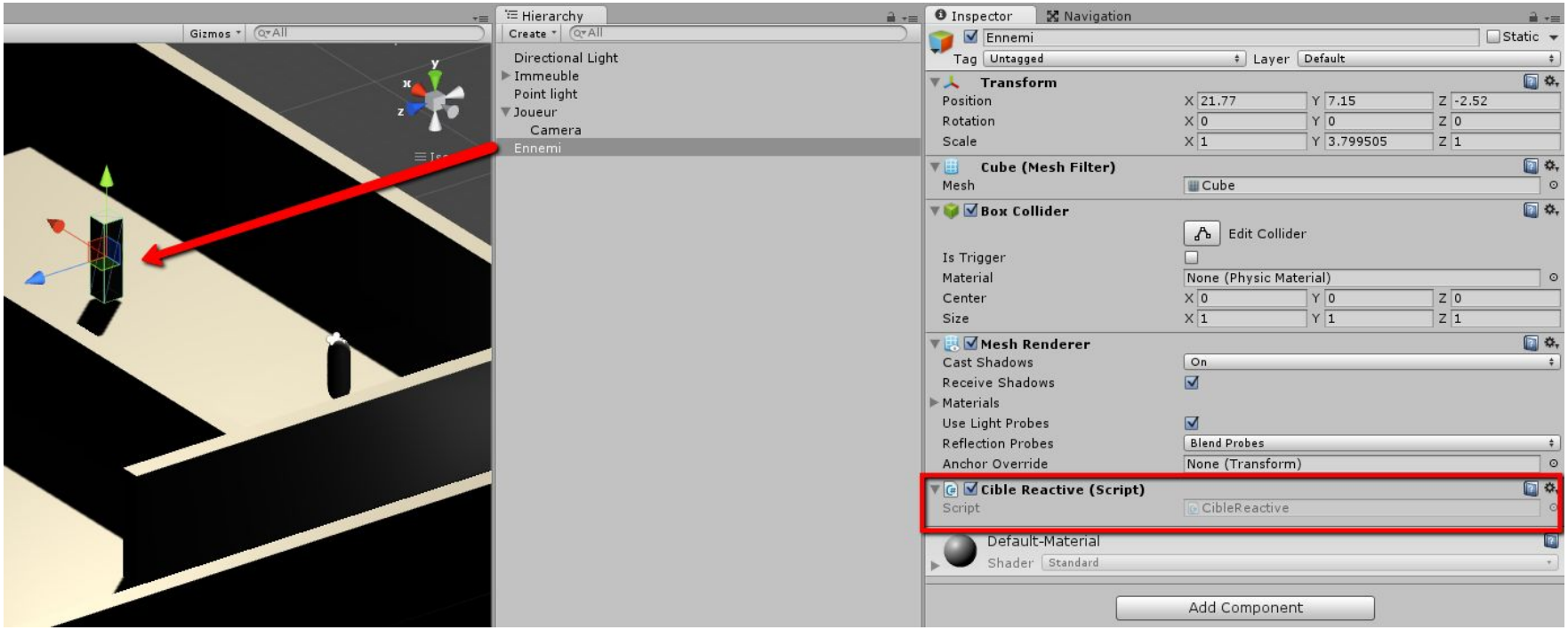
Créer une cible
La méthode Destroy(gameObject)
Présentation des étendues de visibilité

Définir des cibles interactives

- Créer un cube et le placer dans l'espace du joueur
- Donner lui une hauteur correspondante à celle du joueur
- Nommer l'objet **Ennemi**
- Déclarer la classe **CibleReactive.cs** en créant un nouveau script C#

Définir des cibles interactives

- Attacher le script **CibleReactive.cs** à l'objet **Ennemi**



Définir des cibles interactives

- Effectuer la détection de cible en modifiant la méthode **Update()** du script **RayCaster.cs**

```
if (Input.GetMouseButtonDown(0)) {
    ...

    Ray ray = camera.ScreenPointToRay(point);
    RaycastHit hit;
    if (Physics.Raycast(ray, out hit)) {
        GameObject hitObject = hit.transform.gameObject;
        CibleReactive cible =
            hitObject.GetComponent<CibleReactive>();
        if (cible == null) {
            StartCoroutine(CubeIndicator(hit.point));
        } else {
            Debug.Log("Cible Touchée " + hit.point);
            cible.JeSuisTouche();
        }
    }
}
```

Définir des cibles interactives

- Il est possible de récupérer l'objet de jeu lié à la position lié à l'impact

```
GameObject hitObject = hit.transform.gameObject;
```

- Il est ensuite possible de récupérer un composant **CibleReactive** attaché à cet objet de jeu

```
CibleReactive cible =
    hitObject.GetComponent<CibleReactive>();
```

Définir des cibles interactives

- Notifier la cible qu'elle est touchée

```
if (cible == null) {
    StartCoroutine(CubeIndicator(hit.point));
} else {
    Debug.Log("Cible Touchée " + hit.point);
    cible.JeSuisTouche();
}
```

Définir des cibles interactives

- Dans le script **CibleReactive.cs**, implémenter la méthode **JeSuisTouche()** et **JeMeurt()** pour permettre à la cible de réagir en cas d'impact

```
17      internal void JeSuisTouche() {
18          StartCoroutine(JeMeurt());
19      }
20
21      private IEnumerator JeMeurt() {
22          transform.Rotate(-90, 0, 0);
23          yield return new WaitForSeconds(2f);
24          Destroy(gameObject);
25      }
```

Destroy

Lorsqu'un élément dans une scène n'est plus utile, il est de bonne pratique de le détruire.

Pour nettoyer la scène, il faut passer l'objet de jeu (**GameObject**) en paramètre de la méthode Destroy. L'utilisation de la référence **this** ne détruira pas l'objet

Internal, Public, Protected, Private

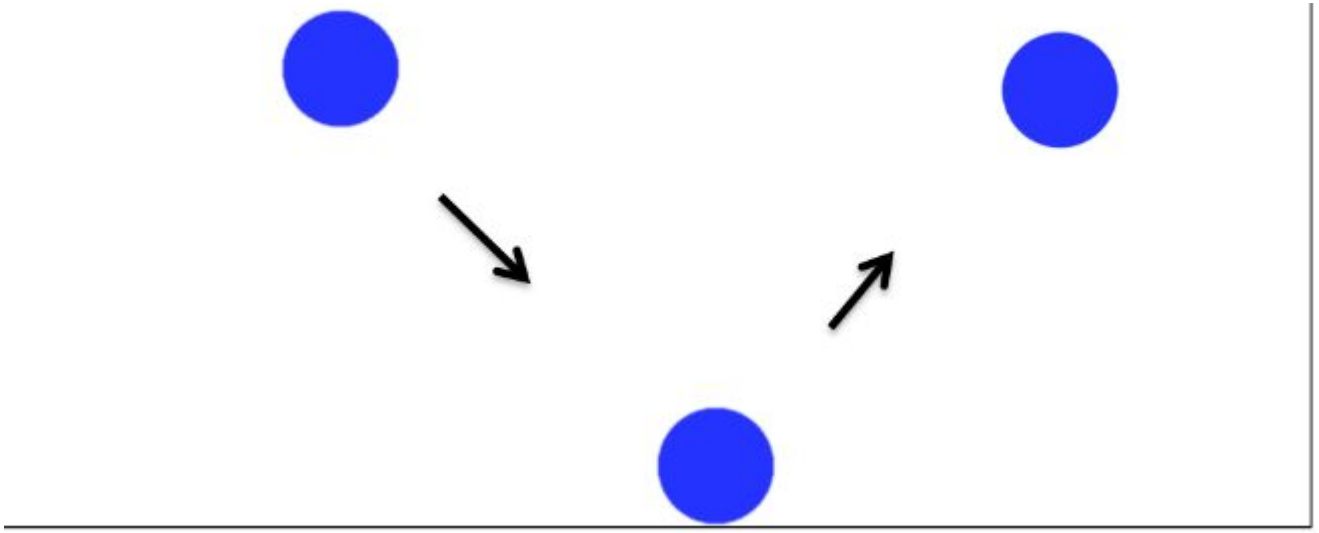
Les instances de classes interagissent via l'appel de méthode. Les méthodes étiquetées publiques sont utilisables par toutes les classes. Les méthodes privées ne peuvent être utilisées qu'à l'intérieur de la classe. Dans le doute, utiliser des méthodes publiques.

3.
Déplacer les ennemis

Déplacement via rebonds
Voir les objets via un RayCast
Gérer l'état de l'ennemi

Déplacer les ennemis

- Implémentation de déplacement en rebond
  - #1 L'ennemi avance vers l’avant
  - #2 Si il se trouve en face d’un obstacle il change de direction
  - #3 Retour à l'étape #1



Déplacer les ennemis
Voir les objets via un RayCast

- Créer un script C# **IntelligenceErrante** et l'affecter à l'objet de jeu **Ennemi**

- Déclarer les variables **vitesse** et **distanceObstacle** paramétrant le déplacement

```
13
14      public float vitesse = 3.0f;
15      public float distanceObstacle = 5.0f;
16
```

Déplacer les ennemis
Voir les objets via un RayCast

- Implémenter la méthode **Update()**

```
// Update is called once per frame
void Update () {
    if (!EstEnVie) return;

    transform.Translate(0, 0, vitesse * Time.deltaTime);
    Ray ray = new Ray(transform.position, transform.forward);
    RaycastHit hit;
    if (Physics.SphereCast(ray, 0.75f, out hit)){
        if (hit.distance < distanceObstacle) {
            // Evite le mur
            float angle = Random.Range(-110, 110);
            transform.Rotate(0, angle, 0);
        }
    }
}
```

- Lancer le jeu

Déplacer les ennemis
Voir les objets via un RayCast

```
 9      void Update() {
10          transform.Translate(0, 0, speed * Time.deltaTime);
```

- L'ennemi avance en continu

```
13          if (Physics.SphereCast(ray, 0.75f, out hit)) {
14              if (hit.distance < obstacleRange) {
15                  float angle = Random.Range(-110, 110);
16                  transform.Rotate(0, angle, 0);
17              }
```

- Un **SphereCast** permet de créer un rayon d'impact plus large

- La cible effectue une rotation lorsqu'elle se trouve trop proche d'un obstacle

Déplacer les ennemis
Gérer l'état de l'ennemi

- Dans le script **IntelligenceErrante.cs**, déclarer une variable pour gérer l'état interne de l'ennemi

```
8
9       public bool EnVie = true;
10
```

- Conditionner le déplacement en testant l'état interne

```
11      void Update() {
12          if (EnVie) {
13              transform.Translate(0, 0, speed * Time.deltaTime);
14          }
15
```

Déplacer les ennemis
Gérer l'état de l'ennemi

- Dans le script **CibleReactive.cs**, modifier l'état de l'ennemi à l'impact

```
public void IHaveBeenHit() {
    IntellingenceErrante ie = GetComponent<IntellingenceErrante>();
    ie.EnVie = false;

    StartCoroutine(IAmDying());
}
```

- Lancer le jeu

- Vérifier le changement d'état de l'ennemi

4.
Instancier des objets préfabriqués

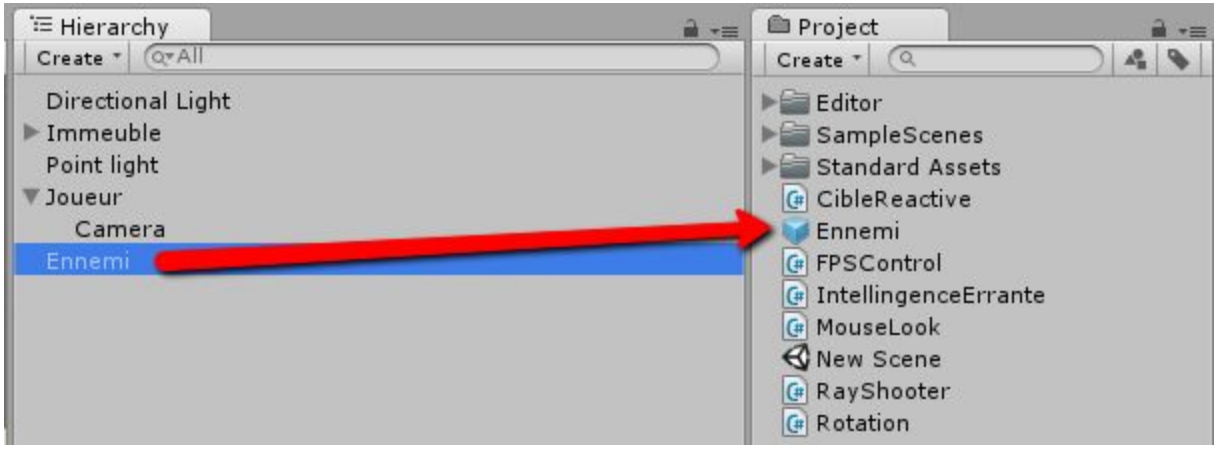
Qu’est ce qu’un prefab ?
Créer un Prefab
Modifier un Prefab
Le Controlleur de Scène
L’attribut SerializeField
La méthode Instanciate

Instancier des objets préfabriqués

- Un prefab est une définition d'un objet de jeu préconfiguré (script, paramètres, textures, …)
- Chaque instance d'un même prefab est similaire et partage les mêmes propriétés lors de leur instanciation dans la scène
  Ex : des impacts, des projectiles, des monstres
- Des prefabs peuvent être instanciés dynamiquement par code

Instancier des objets préfabriqués
Exercice : Créer un Prefab

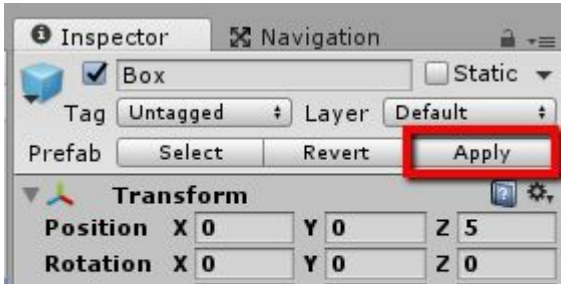
- Sélectionner un objet depuis la vue Hierarchy
- Glisser l'objet dans la vue projet à la racine ou dans un répertoire dédié



- L'objet initial dans la vue Hierarchy passe en bleu indiquant que sa définition est contenue dans un prefab.

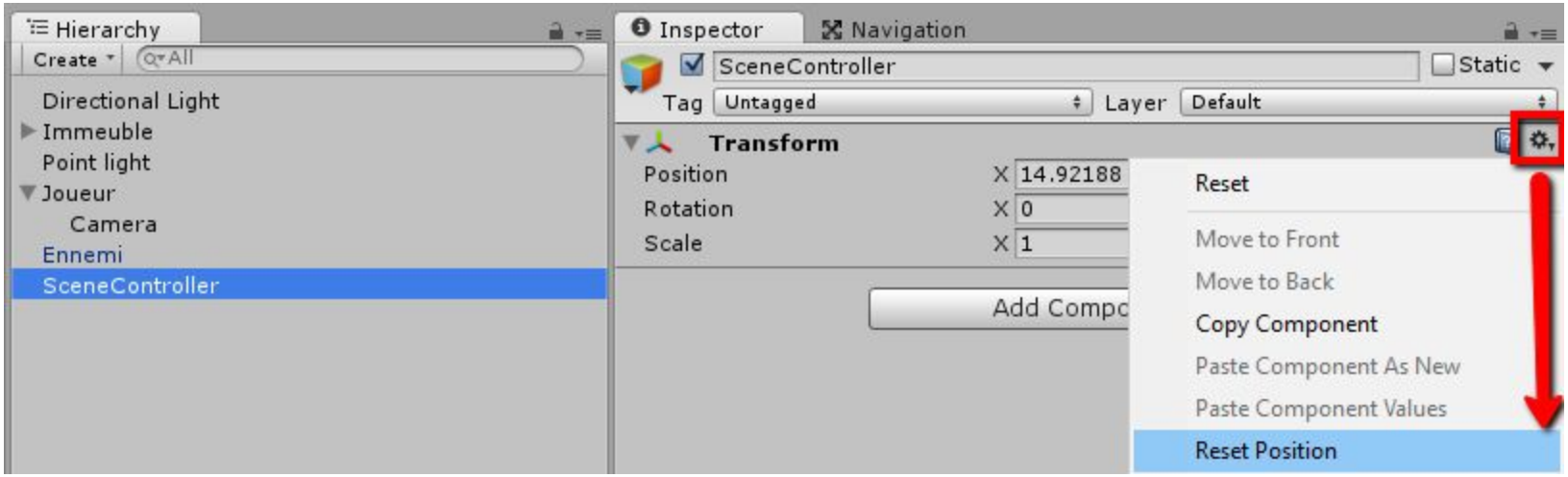
Instancier des objets préfabriqués
Modifier un Prefab

- Glisser le prefab de la vue Project dans la vue Scene
- Modifier les propriétés du prefab
- Appliquer les modifications
- Supprimer le Prefab de la vue Scene



Instancier des objets préfabriqués
Le Controleur de Scène

- Supprimer l'instance du Prefab ennemi de la scène
- Créer un objet de jeu vide, nommé **Controleur**, et le placer aux coordonnées (0,0,0) en effectuant une mise à zéro de sa position



Instancier des objets préfabriqués
Le Controleur de Scène

- Créer le script C# **ControleurScene.cs**, l'attacher à l'objet **Controleur**

- Déclarer un champ **prefabEnnemi** décoré par l'attribut **SerializeField** pour le rendre accessible depuis l'éditeur Unity malgré son statut private

```
5
6       [SerializeField]
7       private GameObject prefabEnnemi;
8
```

SerializeField

Cet attribut permet à l'éditeur Unity d'exposer un champ/propriété dans le composant script tout en masquant le champ aux autres objets via la visibilité **private**

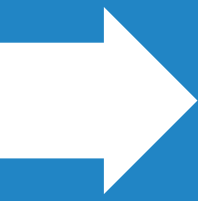
Instancier des objets préfabriqués
Le Controleur de Scène

- Déclarer un champ **ennemie** pour stocker l'instance en cours de l'ennemi

```
8
9       private GameObject ennemie;
10
```

- Compléter la méthode **Update** pour instancier un ennemi lorsque ce dernier n'est pas vivant

```
11      void Update() {
12          if (ennemie == null) {
13              ennemie = Instantiate(prefabEnnemi) as GameObject;
14              ennemie.transform.position = new Vector3(0, 1, 0);
15              float angle = Random.Range(0, 360);
16              ennemie.transform.Rotate(0, angle, 0);
17          }
18      }
```

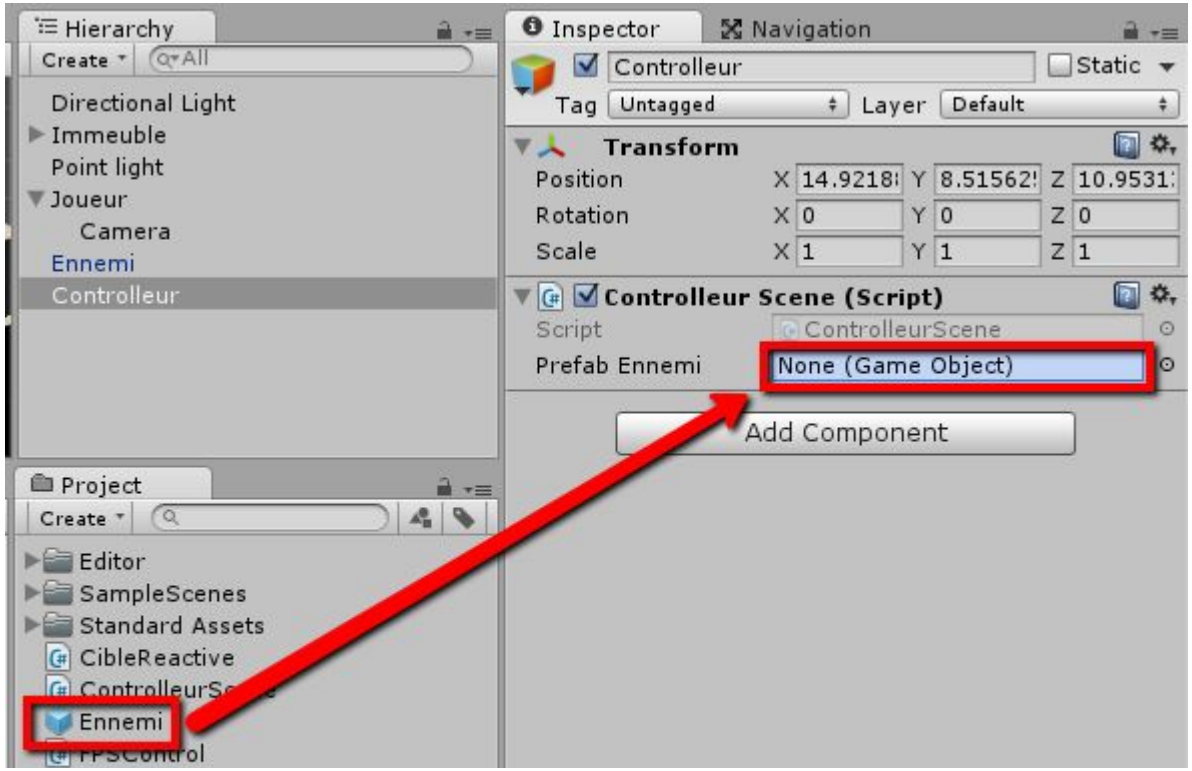


Instantiate

La méthode **Instantiate(prefab)** permet de créer/instancier un objet depuis un prefab dynamiquement

Instancier des objets préfabriqués
Le Controleur de Scene

- Sélectionner l'objet **Controleur** puis déposer le prefab ennemi dans le paramètre **ennemiPrefab** du script **ControleurScene.cs**



5.
Tirer en instanciant des projectiles

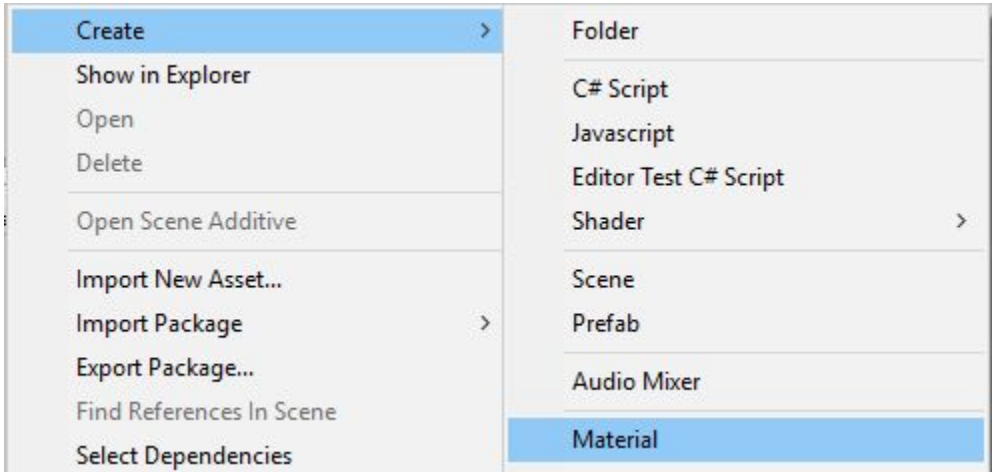
Créer le projectile
Détecter le joueur et lui tirer dessus
Animer le projectile
Affecter des dommages

Tirer en instanciant des projectiles
Créer le projectile

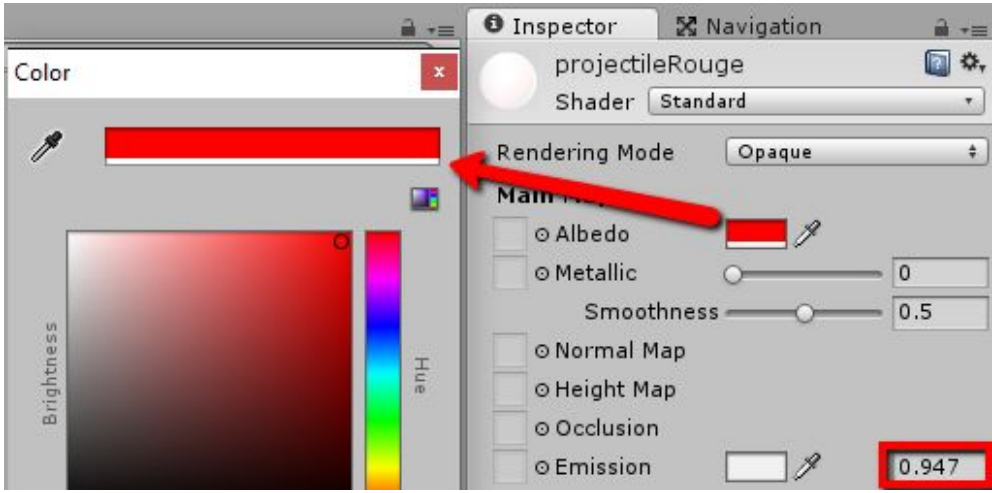
- Créer un objet de jeu basé sur une primitive (cube, sphere, capsule) pour représenter le projectile
- Nommer l'objet projectile
- Créer le script C# **Projectile.cs**
- Depuis la vue projet, créer un matériau et nommer le **projectileRouge**

Tirer en instanciant des projectiles
Créer le projectile

- Utiliser le contrôle de sélection de couleur de la propriété **Albedo** pour attacher une couleur au matériau



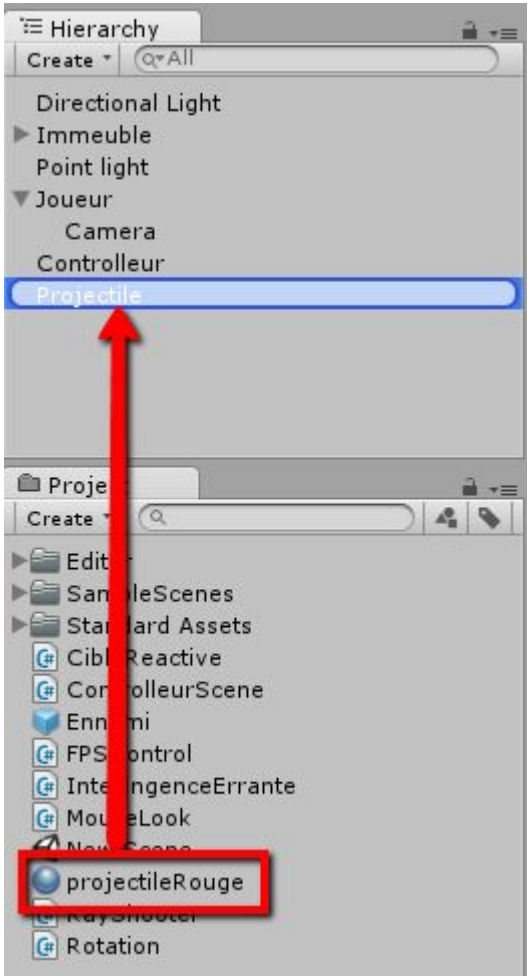
- Modifier la propriété **Emission** pour faire briller le projectile



Tirer en instanciant des projectiles
Créer le projectile

- Glisser le matériau depuis la vue Project vers l'objet **Projectile** de la vue Hierarchy pour affecter le matériau

- Alternativement vous pouvez glisser le matériau vers l'objet **Projectile** de la scène



Tirer en instanciant des projectiles
Créer le prefab projectile

- Glisser l'objet **Projectile** depuis la vue Hierarchy vers la vue Project pour créer un prefab du projectile

- Supprimer le projectile de la scène

Tirer en instanciant des projectiles
Détecter le joueur et lui tirer dessus

- Créer un script C# **Joueur.cs** et l'attacher à l'objet **Joueur** de la vue Scene

- Augmenter le code du script **IntelligenceErrante.cs** pour préparer l'instanciation des projectiles

```
10
11      [SerializeField]
12      private GameObject prefabProjectile;
13
14      private GameObject projectile;
15
```

Tirer en instanciant des projectiles
Détecter le joueur et lui tirer dessus

- Augmenter la méthode **Update** pour préparer l'instanciation des projectiles

```
if (Physics.SphereCast(ray, 0.75f, out hit)){
    if (hit.transform.gameObject.GetComponent<Joueur>()) {
        // Tir sur le joueur
        if (projectile == null) {
            projectile = Instantiate(prefabProjectile);
            projectile.transform.position =
                transform.TransformPoint(
                    Vector3.forward * 1.5f);
            projectile.transform.rotation = transform.rotation;
        }
    } else {
        if (hit.distance < distanceObstacle) {
            // Evite le mur
            float angle = Random.Range(-110, 110);
            transform.Rotate(0, angle, 0);
        }
    }
}
```

Tirer en instanciant des projectiles
Détecter le joueur et lui tirer dessus

- Sélectionner le prefab **Ennemi**

- Affecter le prefab Projectile à la propriété **Prefab Projectile** du script **IntelligenceErrante.cs**

- Lancer le jeu

Tirer en instanciant des projectiles
Animer le projectile

- Modifier le script **Projectile.cs**
  - Déclarer les champs permettant de gérer la vitesse et les dégâts associés

```
5
6       public float vitesse = 10f;
7       public int degat = 1;
8
```

  - Ajouter un comportement de déplacement

```
// Update is called once per frame
void Update () {
    transform.Translate(0, 0, vitesse * Time.deltaTime);
}
```

Tirer en instanciant des projectiles
Animer le projectile

- Modifier le script **Projectile.cs**
  - Coder la détection avec un joueur ou d'autres objets

```
12
13      void OnTriggerEnter(Collider other) {
14          Joueur joueur = other.GetComponent<Joueur>();
15          if (joueur != null) {
16              Debug.Log("Joueur touché");
17          }
18          Destroy(this.gameObject);
19      }
20
```

OnTriggerEnter

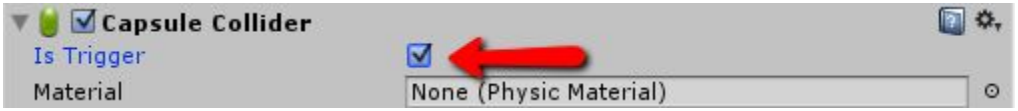
La méthode **OnTriggerEnter()** est automatiquement appelée lorsque l'objet rentre en contact avec un autre objet.

Tirer en instanciant des projectiles
Animer le projectile

- Modifier le script **Projectile.cs**
  - Coder la détection avec un joueur ou d'autres objets

```
12
13      void OnTriggerEnter(Collider other) {
14          Joueur joueur = other.GetComponent<Joueur>();
15          if (joueur != null) {
16              Debug.Log("Joueur touché");
17          }
18          Destroy(this.gameObject);
19      }
20
```

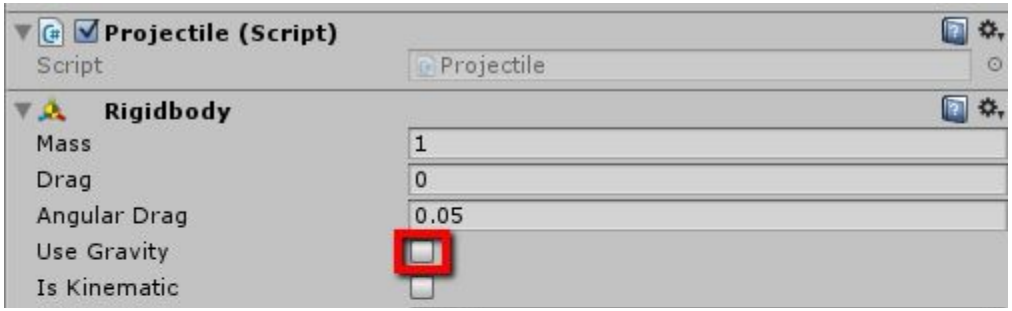
- Activer l'envoi de notification depuis le composant **Capsule collider** du projectile



Tirer en instanciant des projectiles
Animer le projectile

- Ajouter un composant **RigidBody** pour que le projectile soit géré par le moteur physique d'Unity

- Désactiver la gestion de la gravité pour que les projectiles aient une trajectoire rectiligne



Tirer en instanciant des projectiles
Affecter des dommages

- Dans le script **Joueur.cs**
  - Déclarer une variable représentant la santé du joueur

```
5
6       private int sante;
7
```

  - Déclarer la méthode **Touche(int)** pour impacter les dommages au joueur et qui sera utilisé comme point d'entrée pour les projectiles

```
11
12      public void Touche(int dommage) {
13          sante -= dommage;
14          Debug.Log("Sante : " + sante);
15      }
16
```

Tirer en instanciant des projectiles
Affecter des dommages

- Dans le script **Projectile.cs**
  - Augmenter la méthode **Update()** pour notifier la classe Joueur de la collision avec un projectile

```
13      void OnTriggerEnter(Collider other) {
14          Joueur joueur = other.GetComponent<Joueur>();
15
16          if (joueur != null) {
17              Debug.Log("Joueur touché");
18              joueur.Touche(damage);
19          }
20          Destroy(this.gameObject);
21      }
```

- Lancer le jeu