Exercice – Réponse de collision

## Mise en place

* Continuez **ExerciceMouvement** du dernier cours.

### CollisionEvent – changement

Dans CollisionEvent, ajoutez la variable GameObject2D const \* \_MyObject et passez également l’object lui-même lors d’une collision.

### GameObject2D – changement

Ajoutez la fonction DrawDebug. Scene doit également l’avoir. Engine va l’appeler APRÈS Draw2D sur la scène seulement si le mode débug est activé.

## CollisionAnalysis+projectile – mise en place

Codez la classe CollisionAnalysis conjointement avec le projectile, votre premier cas d’usage.

Dans la classe Projectile…

1. Créer un membre \_CollisionAnalysis de type CollisionAnalysis.
2. Enregistrez la méthode CollisionAnalysis::Analyze de \_ CollisionAnalysis dans GameObject2D::OnCollisionEvent comme ceci

std::function<void(const CollisionEvent&)> methodToCall = [this](const CollisionEvent& InEvent) {\_CollisionAnalysis.Analyze(InEvent); };

OnCollisionEvent(methodToCall);

1. Créez une méthode protégée void Bounce(const CollisionAnalysis &) qui sera appelée via CollisionAnalysis::OnCollisionAnalysisEvent.

std::function<void(const CollisionAnalysis&)> methodToCallOnAnalyze = [this](const CollisionAnalysis& InAnalysis) {Bounce(InAnalysis); };

\_CollisionAnalysis.OnCollisionAnalysisEvent(methodToCallOnAnalyze);

1. Appelez CollisionAnalysis::Draw2D dans la nouvelle méthode DrawDebug.

## CollisionAnalysis - codage

Codez la classe CollisionAnalysis. Assurez-vous que les informations de débugs sont correctes avant de continuer ! **CONSEIL** : mettez vos masques de collision à 1 pour tous les objects afin qu’ils se touchent !

## Projectile – rebond

Dans la méthode méthode privée void Bounce(const CollisionAnalysis&), codez la logique pour faire rebondir votre projectile. Facile, n’est-ce pas ?

Limité le nombre de rebonds à 3 avant la destruction du projectile.

## Charactercontroller – contact continu

Implémenter le contact continu de votre *character controller* avec la classe CollisionAnalysis.