Examen final – DDJV

Remettre à la fin dans le chemin indiqué au tableau.

XX points

### Règles

* Documentation Ursina
* Code qu’on a fait ensemble
* www.notes-de-cours.com
* Musique
* Modèle LLM (ChatGPT, Claude, Gemini, etc) Oui, c’est permis !
* Internet autre que ce qui est mentionné plus haut
* Voisin
* Téléphone

# Démo

Vous allez faire un petit **simulateur de collision en** **2D** comme illustré ci-haut. Vous simulerez dans l’axe XY de Ursina. L’axe des X est aligné avec l’axe des X, le Y avec les Y.

# Code de démarrage

Récupérez le main\_.py et examen\_util.py sur le site web. Renommez le fichier main par *main\_<votre\_nom>.py.*

*examen\_util.py*ne doit pas être modifié. Toutes les valeurs peuvent être utilisées directement dans le code (jamais besoin de les passer en paramètre).

# Question 1 : Configuration du débuggeur

Organisez-vous pour que lorsque la touche F5 est appuyée, le fichier *main.py* est systématiquement démarré. Copié l’entièreté du fichier de configuration du débuggeur comme réponse à la question 1.

# Modélisation

## Murs

Le mur gauche a été construit pour vous correctement dans la classe *BounceSimulatorXY*. Inspirez-vous de ce code pour construire le mur droit et le sol. Tous doivent être bleus et être éclairés. Tous les modèles doivent être parentés à la classe *BounceSimulatorXY*.

## Question 2

Pourquoi on utilise la multiplication par 0.5 plutôt d’une division par 2 ici ?

demi\_hauteur = hauteur \* 0.5

demi\_epaisseurxy = epaisseur\_xy \* 0.5

## Balle

La balle de la classe Balle est une sphère rouge dont le rayon est spécifié dans les constantes. Elle doit être parenté à la classe Balle.

Ça ressemble à ceci.

A red dot on a gray background

Description automatically generated

# Simulation sans gravité

# A red circle on a gray background Description automatically generated

## Mouvement linéaire

Faites-bouger dans l’espace 2D la balle en ligne droite avec une vitesse constante (celle passée en paramètre). Utilisez ces variables pour entreposer la vitesse et la position de la simulation.

self.\_sim\_velocity = Vec2(0,0)

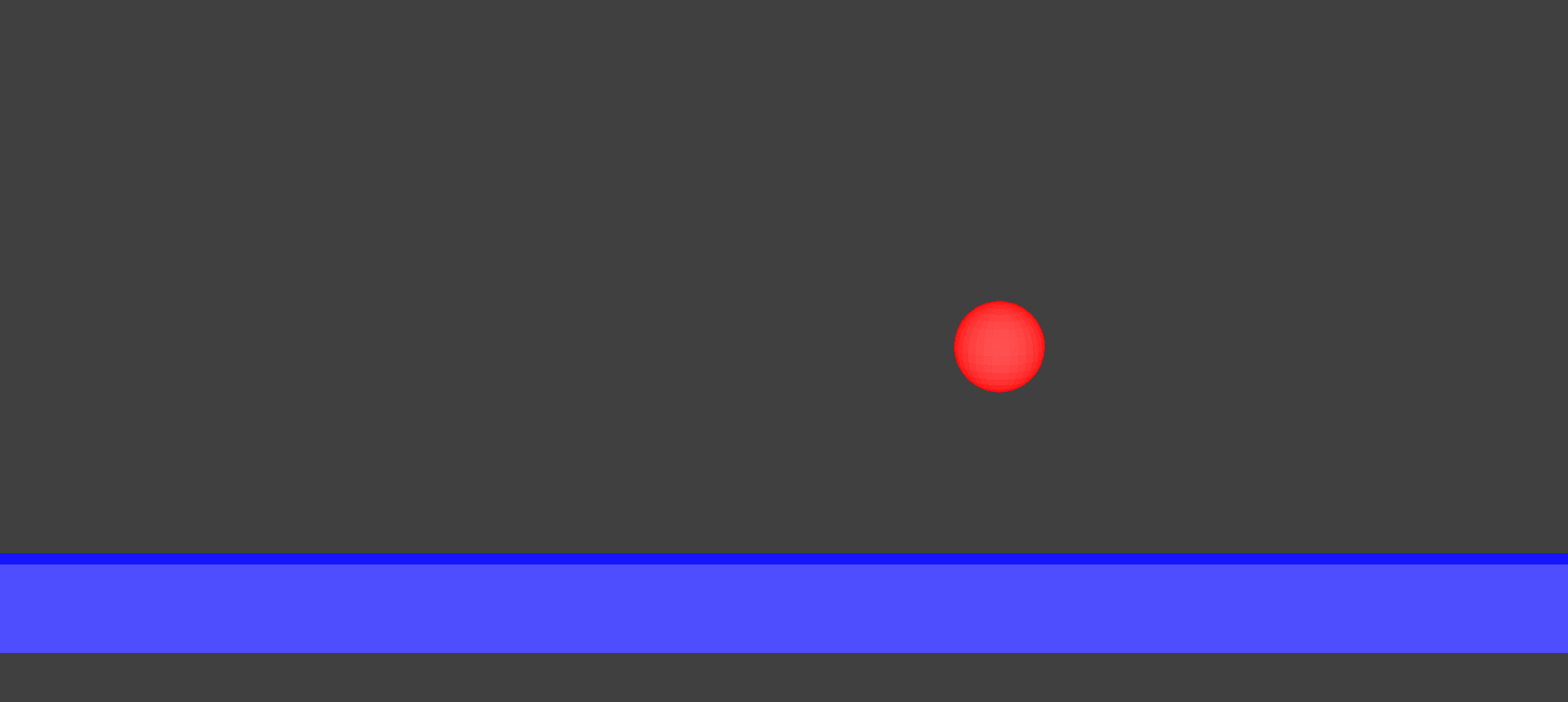
self.\_sim\_position = Vec2(0,0)

Sur une autre ligne, ajustez la position 3D de la Balle.

## Question 3

Expliquez en commentaire en analysant les unités pourquoi on doit multiplier la vitesse par time.dt pour obtenir le delta\_position.

# Simulation avec gravité



NOUVEAUTÉ ! Maintenant, en plus de la position qui change à chaque update, la vitesse changera également ! Nous appellerons delta\_vitesse le changement de vitesse. Ainsi…

delta\_vitesse = ACCELERATION\_2D \* time.dt

self.\_sim\_velocity += delta\_vitesse

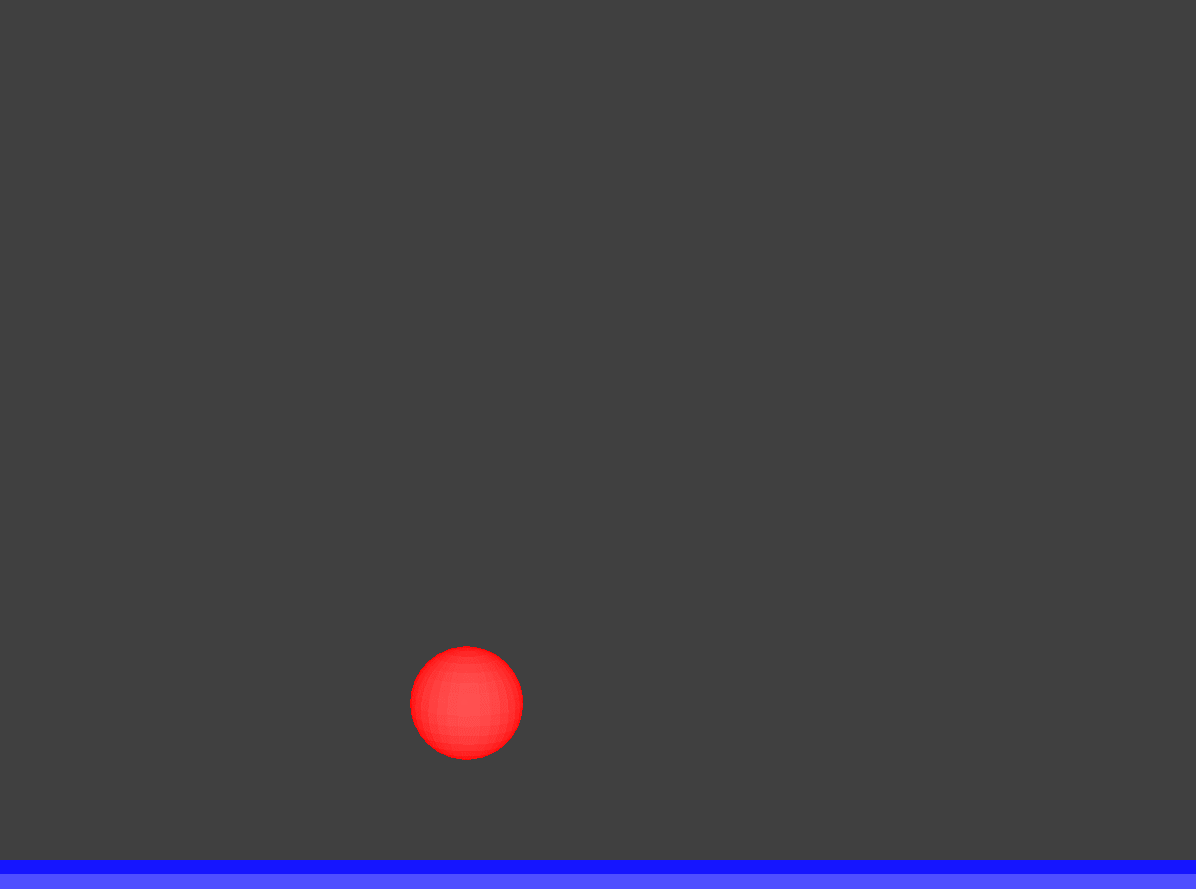
puis poursuivez la simulation avec le calcul de delta\_position et le changement de position.

# Rebond sur la boite

## Enregistrement de balles

Faites que l’enregistrement de balles de la classe *BounceSimulatorXY* fonctionne (utilisez une liste).

## Rebond au sol



1. Implémentez la méthode Balle.bounce\_y qui renversera la composante Y de la vitesse.
2. À partir de l’update de la classe *BounceSimulatorXY*, faites que lorsque les balles touchent au sol, elle rebondit en verticalement. Faites-vous un dessin de la balle et du sol pour comprendre le test à faire.

## Rebond aux murs

Implémentez la méthode Balle.bounce\_x et trouvez une solution pour que les balles rebondissent sur le mur gauche et le mur droit.

# Clignotement du sol/des murs

A red ball on a gray background

Description automatically generated

Lorsqu’un mur est atteint par une balle, celui-ci doit clignoter une fois. Le code de réaction doit se trouver à l’extérieur dans la fonction.

## Callable

1. Implémentez et nommez correctement la méthode BounceSimulatorXY.RENOMMER\_LA\_METHODE. Cette méthode prend un *callable* qui sera appelé lors d’une collision avec un mur.
2. Lors d’une collision avec un mur dans dans BounceSimulatorXY, faites que la callable est appelé avec la valeur en paramètre correcte de l’enum ECollisionType.
3. Créez une méthode globale qui sera appelée lors de la collision avec les murs.
4. Pourriez-vous initialiser le *callable* avec une valeur par défaut (une méthode qui en fait rien) afin que rien ne plante si personne ne l’utilise ?

## Clignotement

Dans la méthode globale créée à l’étape 3, faites que le mur touché clignote pour 0.1 seconde. Utilisez une séquence jumelée à Entity.animate\_color.

# Rebond entre les balles

## Paires de balles

1. Dans BounceSimulatorXY.update, un test doit être fait sur toutes les paires de balles possibles.
2. Pour toutes les paires de balles du système, vérifier si elles se touchent. Faites un dessin pour trouver le test à faire.

### Question 4

Expliquez-en 2-3 lignes maximum quel problème cet algorithme peut causer si un nombre important de balles est présent. Donnez un exemple de solution possible pour contourner ce problème.

## Rebondissement entre les balles

1. Implémentez Ball.bounce. La balle garde sa vitesse (la longueur) mais se dirigera dans la nouvelle direction passé en paramètre.
2. Quand deux balles se touches, calculez le delta entre la position des deux balles et normalisez. Ceci est la direction du rebond. Appelez Ball.bounce sur une des balles avec la direction et sur l’autre, la direction négative.

# Remise

Fichier main\_VOTRE\_NOM.py sur le U :