TP1 : Simulation

Par :
Perrot, Myriam – 111 083 767,
Li, Meiqi - 537 305 934,
Casanova, Angelina - 537 307 056,
Équipe 17

Réalisé dans le cadre du cours :
IFT-2103 – Programmation de jeu vidéo

Rapport présenté à :
Chéné François

Remis le :
11 octobre 2024

Environnement

- Espace euclidien : Oui
- Nombre de dimensions : 2
- Nature des dimensions : Continue
- Forme des dimensions : Plane
- Grandeur des dimensions : Finie

Boucle de jeu

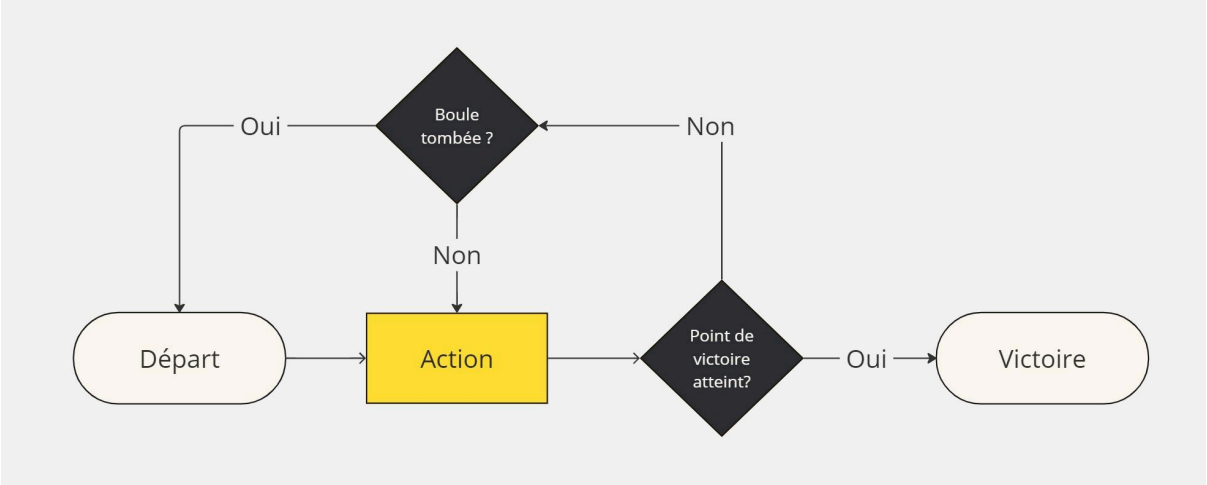

Figure 1 - Boucle de jeu

Le joueur commence au point d'apparition en haut à gauche de l'écran.

- Si le joueur atteint le point de victoire représenté par le bonhomme de fin à droite de l'écran, le joueur s'arrête de bouger, la partie est gagnée et un message de victoire apparaît.
- Si le joueur tombe d'une plateforme, il repart du point d'apparition.
- Tant qu'aucune de ces deux conditions n'est remplie, on met à jour la position du joueur.

Action

Sauter

- Précondition : Le cercle doit toucher le sol
- Effet : Le cercle ne touche plus le sol
- Loi(s) physiques utilisées : Inertie, force gravitationnelle

Déplacements horizontaux

- Précondition : Aucun obstacle présent vers la direction souhaitée
- Effet : Le cercle bouge vers la gauche ou la droite
- Loi(s) physiques utilisées : Friction dynamique, inertie, force gravitationnelle

Collisionneurs

- Cercle (Circle collider) : Joueur (cercle)
- Boîte : Plateformes, bonhomme de fin, obstacle

Optimisation de la détection de collision

Au départ, nous avons utilisé la fonction intégrée OverlapCircle de Unity pour détecter si le joueur touche le sol. Cependant, nous avons opté pour la méthode de détection par rayons. La fonction OverlapCircle parcourt tous les colliders, ce qui peut ralentir les performances. La détection par rayons se concentre uniquement sur une ligne, réduisant ainsi le coût de calcul. De même, pour la fonction CheckCollisions, nous avons choisi une approche basée sur la vérification des distances.

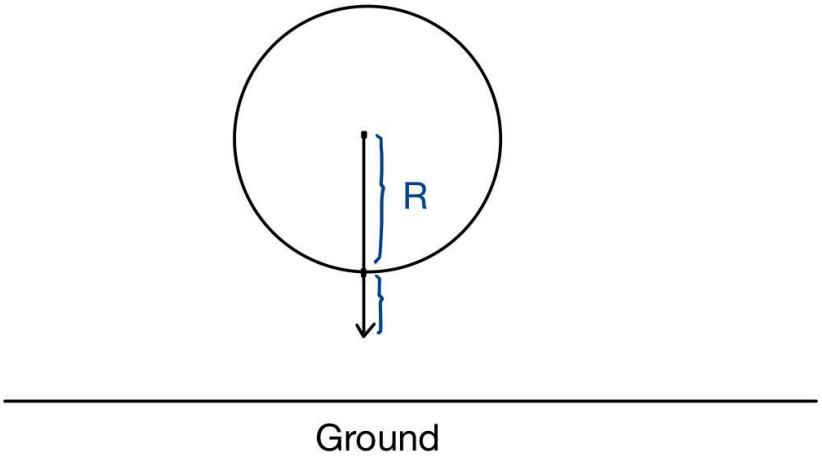

Figure 2 - Explication de la fonction IsGrounded()

Algorithme IsGrounded(p, b, r) : $\mathbb{R}$
Entrée:

- p : position du cercle
- b : limites du niveau (AABB)
- r : rayon du cercle

Sortie:

- Booléen (vrai si le cercle est au sol, sinon faux)

Pseudo-code:

```
Lancer une raycast vers le bas à partir de la position p
Si la raycast touche un collider au sol
    Retourner vrai, sinon faux
```

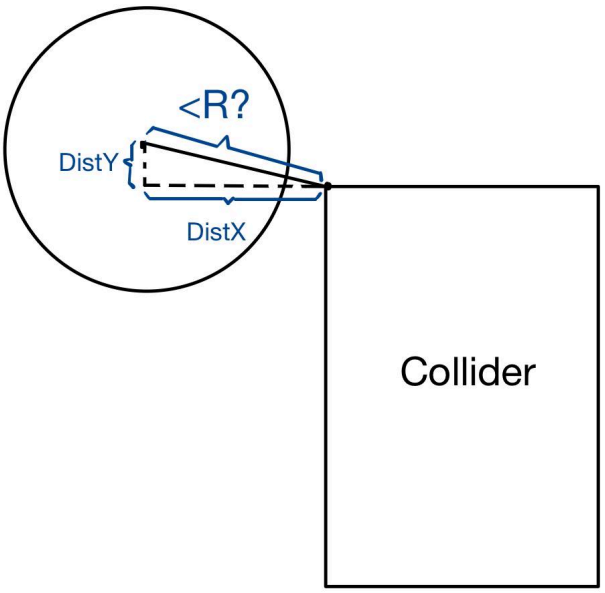

Figure 3 - Explication de la fonction CheckCollisions()

Algorithme CheckCollisions(p, b, r) : $\mathbb{R}$
Entrée:

- p : position du cercle
- b : limites des autres colliders
- r : rayon du cercle

Sortie:

- Booléen (vrai si le cercle touche un autre collider, sinon faux)

Pseudo-code:
Calculer le centre et la taille du rectangle
Trouver le point le plus proche du centre du cercle par rapport au rectangle
Calculer la distance du cercle au point le plus proche (en utilisant le théorème de Pythagore)
Si la distance est inférieure au rayon
Retourner vrai, sinon faux

Réactions aux collisions

- Collision entre le cercle et le bonhomme de fin : Fin du tour, un message de victoire apparaît, immobilisation.
- Collision entre le cercle et les plateformes ou l'obstacle : Immobilisation, glissement

Pour pouvoir effectuer ce TP, nous avons utilisé ChatGPT lorsque nous avons eu de la difficulté à implémenter les différentes formes de collisions ainsi que le frottement dynamique.