[Moteur de jeu] Projet

Adèle Imparato
Nicolas Luciani

24 mai 2023

Notre code peut être trouvé à cette adresse : cliquez ici.

Introduction

Dans ce rapport, on va brièvement décrire les parties essentielles de notre projet, c'est-à-dire :

- Le jeu
- Le pingouin
- Le terrain
- Les collisions
- La physique
- La caméra ressort
- Les obstacles
- L'interface Utilisateur

1 Le jeu

Notre jeu consiste en un pingouin qui glisse sur une piste enneigée. Dans ce jeu, la caméra suit le pingouin qui glisse en continu le long de la pente. L'utilisateur, lui, contrôle le pingouin à l'aide des flèches ←→, le but étant d'éviter les obstacles (i.e. boules de neige) pour que le pingouin arrive en bas de la piste le plus vite possible. L'utilisateur peut aussi appuyer sur les flèches ↑↓ pour faire accélérer ou décélérer le pingouin.

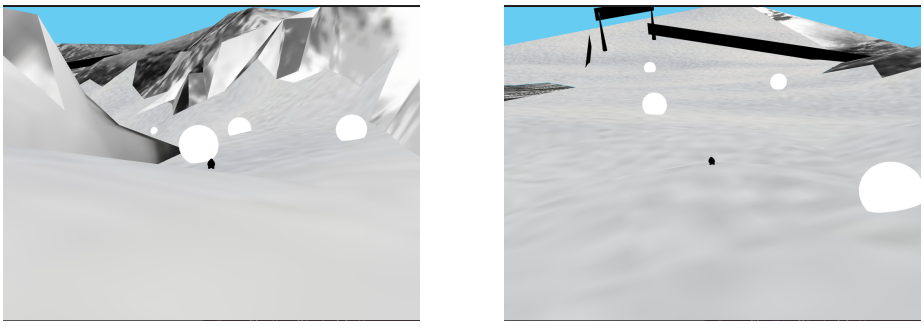
1.1 Comment démarrer ?

Pour jouer au jeu, après avoir compilé le projet grâce aux commandes suivantes :

1. `mkdir build`
2. `cd build`
3. `cmake ..`
4. `cd ..`
5. `cmake -build build`

On peut le lancer en entrant la commande suivante dans le répertoire principal :
`./build/launch-pengine.sh`

1.2 Aperçu du jeu



(a) (b)

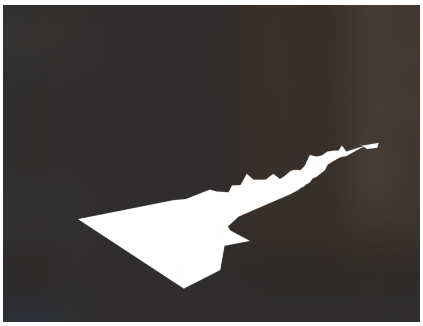
FIGURE 1 – Images *in-game*

2 Le pingouin

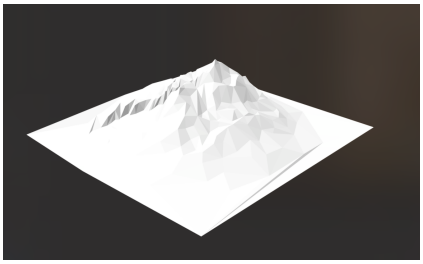
Le pingouin utilisé consiste en un maillage 3D trouvé cette adresse. Aucune texture de lui a été plaqué, il est simplement de couleur noir.

3 Le terrain

Le terrain a été généré sur Blender et consiste en un maillage 3D en forme de piste, et un autre en forme de montagne. Les deux maillages s'emboîtent mais le fait qu'ils soient codés comme deux maillages distincts permet de leur appliquer des textures différentes et de limiter les déplacements du pingouin à la piste. En effet, si le joueur décide d'utiliser la physique pour sortir du terrain, dès lors que le pingouin n'est plus dans boite englobante de la piste, il est ramené à sa position initiale, lui faisant perdre un temps considérable.



(a) Piste

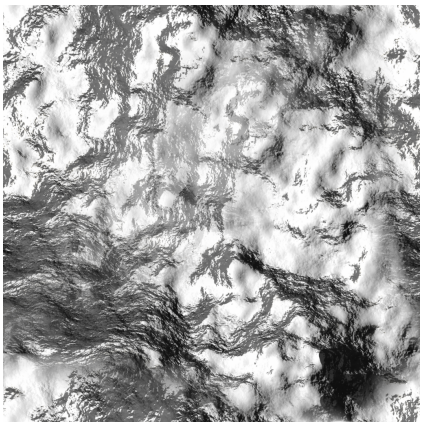


(b) Montagne

FIGURE 2

A la piste, on ajoute une texture de neige. Et aux montagnes, la texture snowrocks (fournie avec les TPs).

(a) Texture piste



(b) Texture montagne

FIGURE 3

Ce qui donne finalement le terrain que l'on voit à la Figure 1.

Note : On a défini le fond de la scène comme étant bleu pour donner un effet de ciel (puisque l'utilisation d'un quad avec une texture paysage rendait mal).

4 Collisions

4.1 Boîte englobante alignée sur les axes

La détection des collisions est implémentée dans une classe BoxCollider, représentant une boîte alignée sur les axes. Le calcul de cette boîte se fait au chargement du maillage. Toutes les collisions du pingouin sont gérées à l'aide de cette boîte. Nous avons en revanche prévu la possibilité

d'implémenter une boîte englobante orientée.
Nous avons implémenté les détections de collisions suivantes :

- AABB - Plan
- AABB - AABB
- AABB - Triangle
- AABB - Maillage

4.2 Détection des collisions

Hors-mis le calcul d'intersection entre deux boîtes, toutes les détections de collisions sont implémentées à l'aide du Separating Axis Theorem. En effet, comme nous ne disposions que d'objets convexes, les collisions sont détectées plutôt efficacement et en un temps raisonnable. Nos maillages ont une structures d'accélération implémentée (une Bounding Volume Hierarchy de type Octree) mais nous n'avons pas besoin de nous en servir pour avoir un rendu agréable.

4.3 Résolution des collisions

Tous nos calculs d'intersections entre nos objets retournent également une normale à l'intersection, nécessaire à la résolution. Nous avons fait le choix de résoudre nos collisions par le calcul de réponses impulsionelles. Grâce aux formules vues en cours, nous avons implémenté ces calculs dans le corps rigide. Nous avons paramétré ces méthodes pour pouvoir modifier les vitesses et les masses des corps en collision avec le pingouin ainsi que les coefficients de restitutions.

5 La physique

L'essentiel du système de physique est implémenté au coeur d'une classe représentant un corps rigide. Nous avons décidé d'utiliser une intégration d'Euler classique, cela semblait correspondre aux besoins de notre projet. En plus de pouvoir modifier la vitesse de ce corps (intéressant pour la résolution des collisions), on peut aussi y appliquer des forces, sous la forme de vecteurs.
La gravité terrestre y est appliquée à tout instant. Nous calculons donc aussi le poids de notre corps rigide grâce à sa masse initialisée à la création du corps.

6 Caméra ressort

Pour que la caméra suive le pingouin tout au long de la descente, nous avons implémentée une caméra accrochée à un ressort et liée au pingouin à travers le graphe de scène. La taille du ressort et sa résistance sont paramétrables.
Nous avons fait le choix de maintenir le pingouin dans le sens de son vecteur de vélocité, en appliquant les rotations nécessaires.

6.1 Transform

Une classe transform permet de calculer les matrices de transformation de tout objet de notre scène en respectant le graphe de scène. Cette classe implémente aussi les calculs des vecteurs *Forward, Up et Right* relatifs à l'espace local de l'objet et servant justement à appliquer les transformation nécessaires à la caméra. En effet, les vecteurs *Forward et Up* de la caméra sont égaux à ceux de sa cible.

7 Les obstacles

On rajoute à notre terrain des obstacles afin d'augmenter la difficulté. Ces obstacles sont des sphères blanches, correspondant à des boules de neige. Elles sont placées arbitrairement le long de la piste. Lorsque le pingouin entre en collision avec une de ces boules de neige, il rebondit dessus avant qu'elle ne disparaisse (comme si le pingouin avait explosé la boule de neige).

8 Interface Utilisateur

Une interface Utilisateur simpliste a été ajoutée au jeu. Plus précisément, un écran titre, un menu de pause, ainsi qu'un écran de fin. Comme GLFW ne permet pas de réaliser d'interface utilisateur à proprement parlé, nous avons fait le choix de représenter ces différents écrans comme des textures sur des plans face à la caméra. Ces rendus nécessitent des shaders différents de ceux utilisés pour le jeu.

FIGURE 4 – Écran Titre

FIGURE 5 – Écran de pause

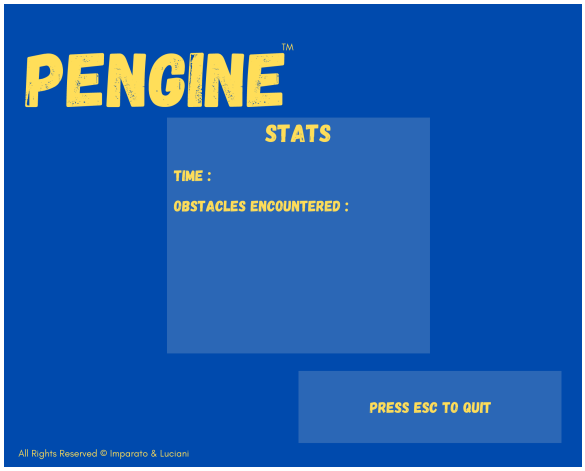


FIGURE 6 – Écran de fin

FIGURE 7 – Interface Utilisateur