Moteur de jeux

Rapport de projet

Benjamin Serva
Louis Jean
Evan Combot

Master 1 IMAGINE
Université de Montpellier

https://github.com/louis-jean0/HAI819I-moteurjeux

15 mai 2024

Table des matières

1 Introduction

Dans le cadre de l'unité d'enseignement Moteur de jeux (`HAI819I`), nous avons eu l'opportunité de développer un moteur de jeu. Ce document vise à présenter les fonctionnalités de notre moteur, et du jeu support que nous avons créé.

Pour lancer le jeu, veuillez lancer ces commandes, après avoir cloné le dépot.

```
mkdir build
cd build
cmake ..
make -j
./main
```

2 Structure du moteur de jeu

Au départ, ne sachant pas vraiment vers quoi s'orienter, nous avons choisi de faire un moteur vraiment général, qui pourrait réaliser la plupart des jeux que nous voudrions. Nous avons donc commencé à bien séparer le travail et à encapsuler toutes les différentes logiques dans beaucoup de classes différentes, pour avoir un code facile à maintenir et modulaire.

Voici donc le diagramme UML de notre moteur de jeu. Il comporte 22 classes.

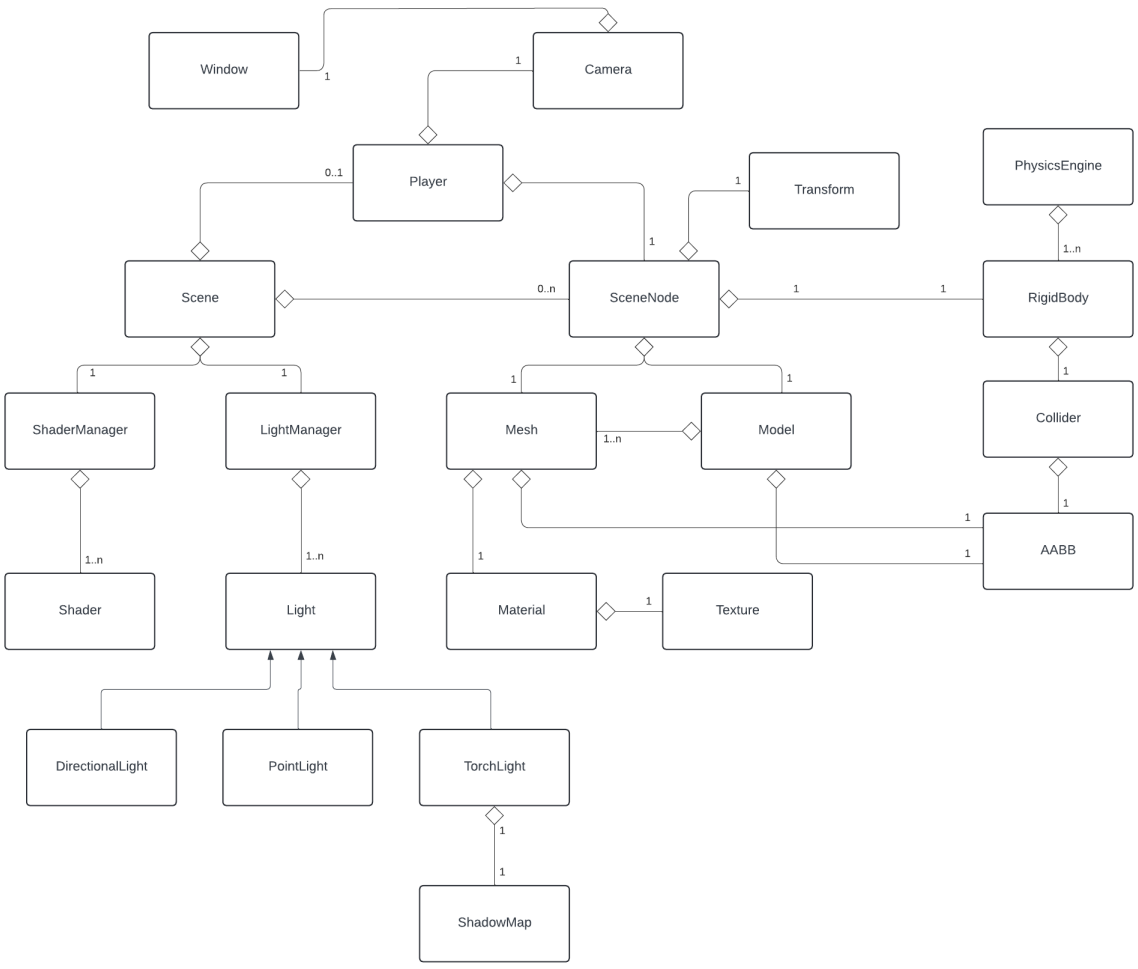

Figure 1: Diagramme UML du moteur

2.1 Librairies utilisées

Pour mener à bien notre projet, nous avons utilisé les librairies `GLFW`, `glad`, `glm`, `ImGui`, `Assimp` et `miniaudio`.

3 Le jeu

Nous avons fini par délibérer sur le type de jeu que nous voulions créer, pour nous mettre d'accord sur un platformer 3D. Très largement inspiré du jeu **BETON BRUTAL**, c'est ainsi que **Ciment Doux** est né. Le concept est simple : grimper une tour généreuse en pièges et difficultés en tout genre, et cela le plus vite possible.

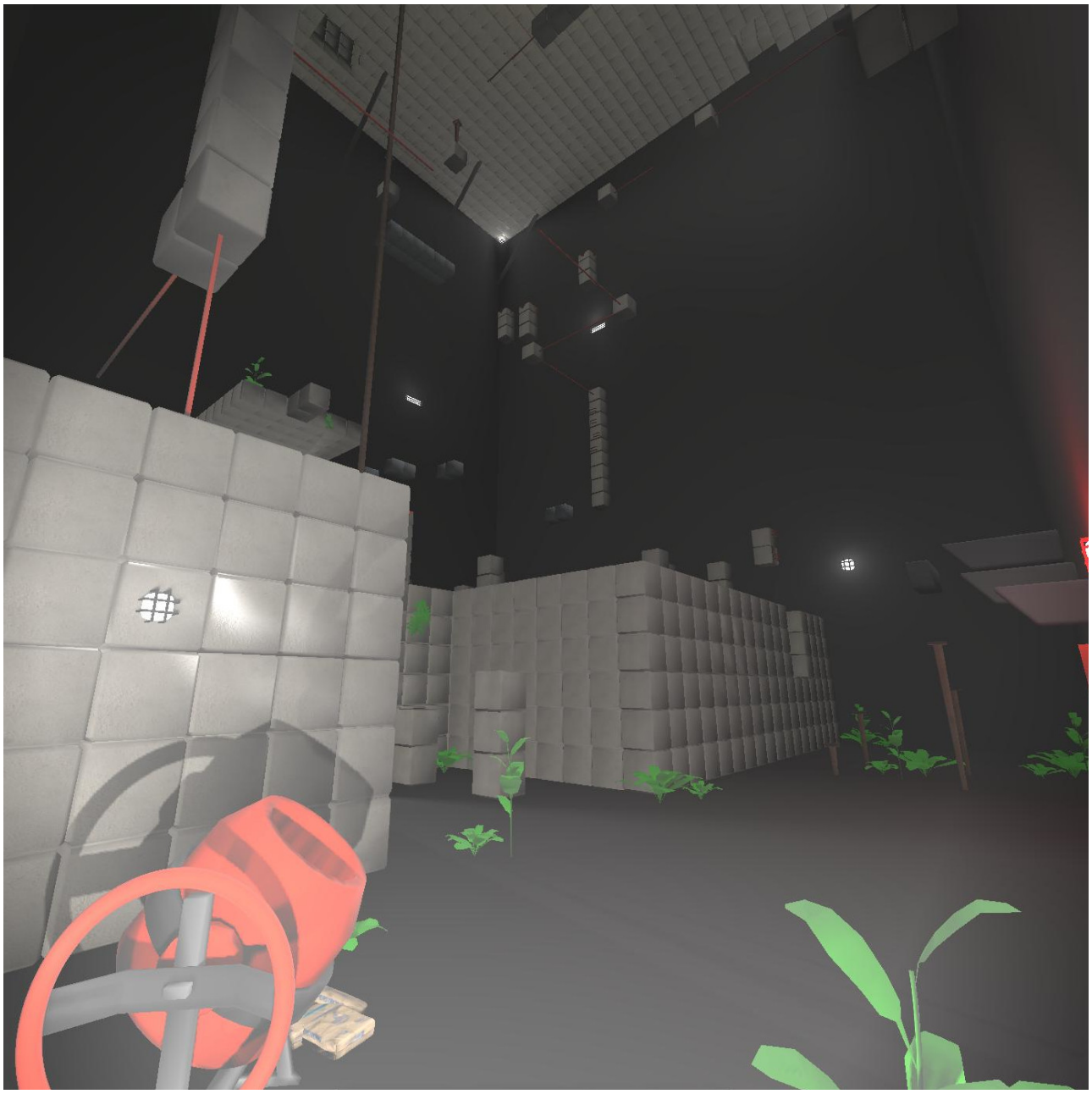

Figure 2: Aperçu du jeu

4 Moteur physique

Dans un premier temps, nous nous sommes concentrés sur l'implémentation d'un moteur physique robuste, pour détecter les collisions entre notre joueur et les objets de l'environnement.

4.1 Détection des collisions

4.1.1 Axis-Aligned Bounding Boxes (AABB)

Après avoir écrit un loader d'objet conjointement avec Assimp, nous avons pu générer automatiquement les AABB des maillages importés.

À partir de là, en utilisant le Separating Axis Theorem (SAT), nous avons pu déterminer les collisions AABB-AABB.

Nous utilisons une détection de collision discrète, à pas de temps fixé.

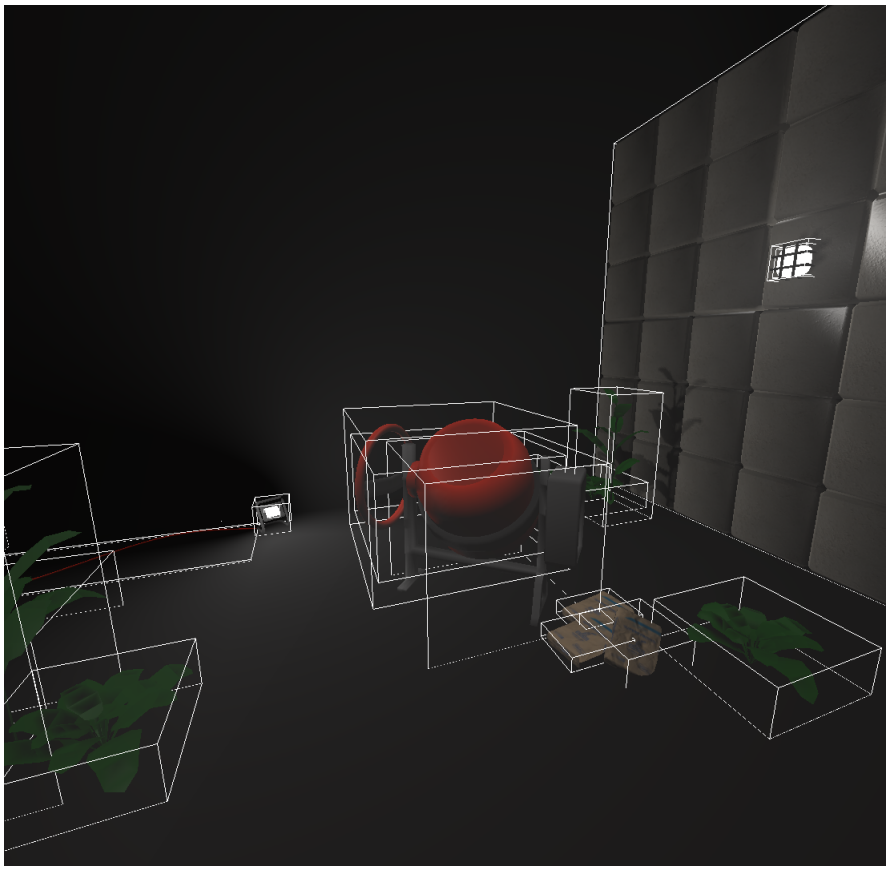

Figure 3: Aperçu des AABB

4.2 Résolution des collisions

Pour résoudre une collision AABB-AABB détectée, on calcule la profondeur de collision avec l'objet en question et on applique cette distance + un offset de sécurité, le tout multiplié par la normale au point de collision, comme translation à l'objet receveur. Comme seul notre joueur est en mouvement, nous n'appliquons une correction qu'à lui seul.

4.3 Intégration d'Euler

Pour calculer les forces qui s'appliquent sur un objet au cours du temps et ainsi déterminer son accélération, sa vitesse et ultimement sa position à un instant t, nous avons utilisé l'intégration d'Euler, avec un pas de temps fixé.

4.4 Différents types de collisions

Grâce au moteur physique, nous avons pu détecter et résoudre plusieurs types de collisions AABB-AABB, en fonction de la nature des objets rencontrés par le joueur.

4.4.1 Classique

Collision classique, le joueur est repoussé de manière à être collé à l'objet, et il est freiné par le coefficient de friction de cet objet.

4.4.2 Rebond

Pour le rebond, nous avons défini un coefficient de restitution de l'énergie. Si le joueur rencontre un objet donc le coefficient de restitution vaut 1, alors 100% de sa vitesse sera réattribuée en direction inverse de la normale de collision. Nous avons utilisé ce mécanisme pour créer des trampolines dans notre jeu.

4.4.3 Glace

Ici, c'est un coefficient de friction qui entre en jeu. Si l'on touche de la glace, alors ce coefficient devient négatif, augmentant ainsi la vitesse du joueur sans jamais le freiner (sensation de glisse).

4.4.4 Échelle

Quand le joueur se trouve sur une échelle, on désactive la gravité et on change les contrôles pour qu'il puisse monter.

4.4.5 Plateforme en mouvement

Lorsque l'on détecte que la plateforme sur laquelle on se trouve est en mouvement, on met tout simplement le joueur en tant que fils de la plateforme dans le graphe de scène et on l'enlève dès qu'il n'y a plus collision.

4.4.6 Plante

Si on détecte une plante, on décide de skip la collision pour ne pas rentrer en contact avec (élément de décor).

5 Environnement

Une fois les bases de la physique posées, il nous fallait un terrain de jeu. Pour cela, nous avons fait appel à l'aide d'une connaissance à l'aise avec la modélisation 3D sur Blender, à savoir Jonas Ronteix. Il nous a fait gagner un temps précieux en réalisant les assets de notre jeu. Grâce à lui, nous avons pu nous concentrer pleinement sur le level design de la carte. Nous pensons que cette expérience a été enrichissante et nous a appris à collaborer avec des artistes, chose que nous ferons peut-être souvent dans nos futurs métiers.

5.1 Map

La map est donc une tour de 50m de haut, que voici.

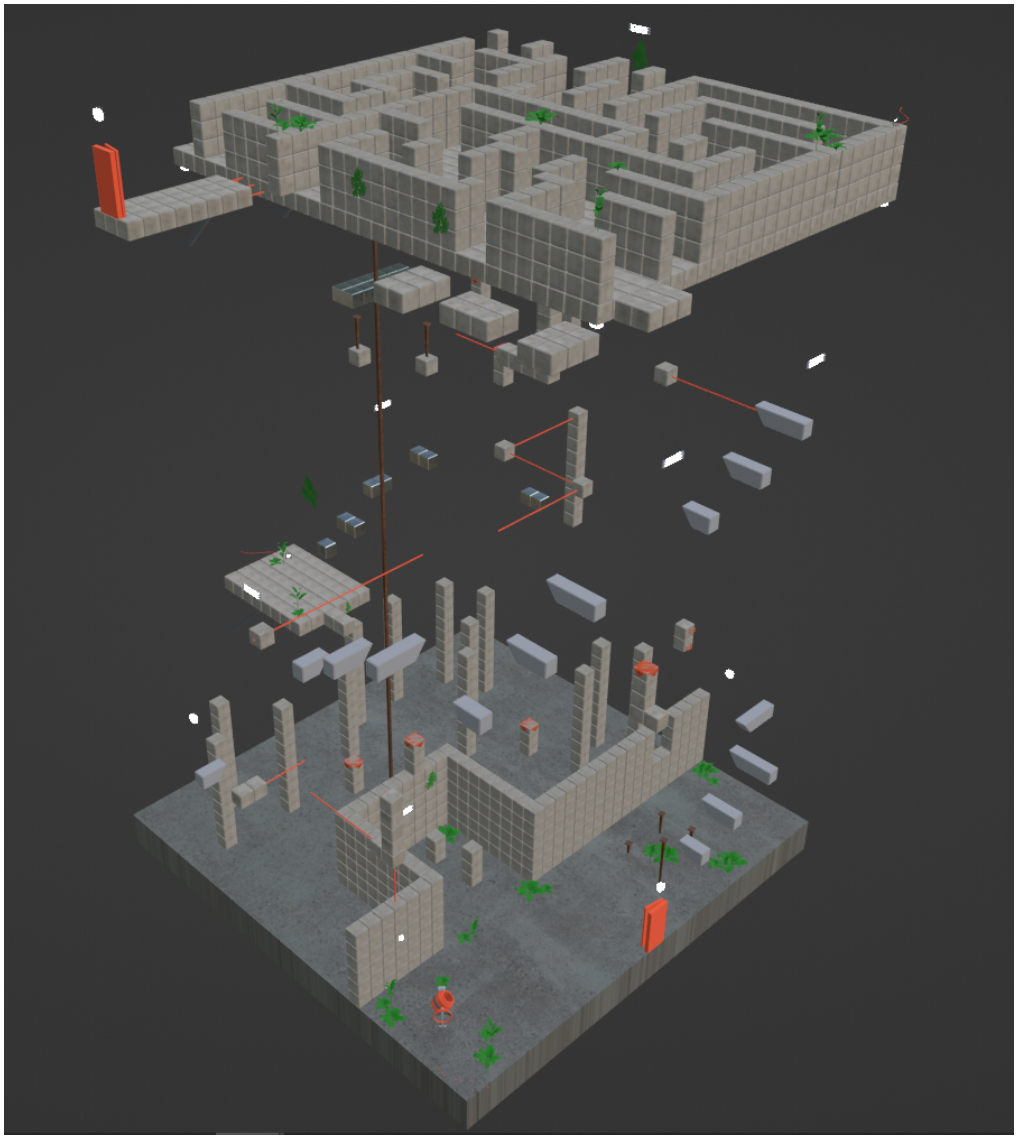

Figure 4: Aperçu du jeu

Elle contient des trampolines, des échelles, des blocs de glace, des blocs de béton, des plantes, et un labyrinthe pour terminer.

5.2 Lumières

Pour ajouter du charme visuel et une ambiance, nous avons implémenté l'illumination de Phong et le shadow mapping. Pour ce faire, nous avons écrit une classe abstraite `Light`, dont trois types de lumières héritent.

5.2.1 Directional light

La plus classique des lumières, tous les rayons sont parallèles, et elle éclaire toute la scène. Utilisée pour mettre une luminosité faible constante dans la tour.

5.2.2 Point light

Ce type de lumière possède une position et d'autres attributs qui lui permettent d'émettre dans toutes les directions à la fois. Elle est atténuée en fonction de la distance.

5.2.3 Spotlight (torch light)

Cette lumière prend une position et aussi une direction. Elle est similaire à la point light, mais elle émet uniquement dans un cône (que l'on peut définir). Nous l'avons utilisé pour créer la lampe-torche du joueur.

6 Menu et interface

Afin de rendre la vie du joueur plus agréable, nous avons intégré un menu pause, qui permet de régler certains paramètres comme le FOV ou le volume, de voir les crédits, de quitter ou de reprendre le jeu.

Aussi, challenge oblige, nous avons mis un minuteur en haut à droite de l'écran, et un altimètre en haut à gauche.

7 Audio

Dans le but de rendre plus plaisant l'expérience du joueur, nous avons ajouté du son. Nous avons utilisé la bibliothèque MiniAudio. Différents sons sont ainsi déclenchés lorsque le joueur déclenche un évènement comme par exemple lorsqu'il arrive sur la page des crédits, qu'il arrive au sommet du niveau ou bien tout simplement lorsque le joueur parcourt le niveau. Le joueur à également la possibilité de contrôler le volume du son dans les paramètres du jeu.

8 Comment jouer

Voici les contrôles de notre jeu :

- `Z,Q,S,D ou W,A,S,D` : se déplacer (varie selon le mode choisi dans le menu)
- `Mouvements de la souris` : bouger la vue
- `Espace` : sauter
- `Shift gauche` : courir
- `Clic gauche` : changer la puissance de la lampe-torche du joueur
- `Clic droit` : allumer/éteindre la lampe-torche du joueur
- `R` : recommencer la partie
- `Échap` : ouvrir le menu

9 Améliorations possibles

- Ajout d'un shake suite à une grosse chute
- Amélioration de la collision avec les échelles
- Évolution de la map
- Amélioration de l'environnement visuel

10 Conclusion

Nous avons pris **beaucoup** de plaisir à réaliser ce jeu. Nous nous y sommes pris assez tard il faut l'avouer, et nous avons l'impression d'avoir fait une game jam de deux semaines entières, mais si c'était à refaire, nul doute que nous le referions. L'expérience acquise sera très précieuse pour la suite de nos études.

Merci pour le temps et l'attention que vous avez consacrés à la lecture de ce compte-rendu.