BENLAMINE Mehdi – FUMEY David – TEIXEIRA-RICCI Maxime
Master 2 - IMAGINA

Projet Moteur de Jeu

GraviCube

Documentation

I. Introduction

Ce projet est l'expression d'une envie de créer un jeu dans sa globalité et de se confronter à des problèmes de conceptions que les moteurs de jeu (comme *Unity3D* et *Unreal Engine*) nous protège.

Il était intéressant de pouvoir concevoir une architecture par nous même, et de réaliser un jeu sans les contraintes de ces puissants outils.

Le projet GIT est disponible à cette adresse : https://github.com/maxime-teixeiraricci/ProjectMDJ

Dans ces quelques pages, nous allons aborder les différents aspects de notre projets : de notre conception du jeu à l'implémentation en passant par nos difficultés et les solutions que nous avons mis en place.

1. Inspiration

Illustration 1 - *Super Mario 3D World : Captain Toad Goes Forth*

Attacher à l'idée de réussir dans les temps à réaliser un projet relativement complets, touchant plusieurs aspects de la création d'un jeu vidéo moderne, nous avons donc décider de réaliser un prototype basé sur les niveaux du jeu « Super Mario 3D World » où le personnage de Captain Toad se retrouve dans des niveaux de taille réduite où le joueur doit contrôler le personnage afin de récupérer les cinq étoiles qui sont disséminés sur le terrain.

L'idée est donc de reprendre à notre compte cette mécanique de gameplay et de l'agrémenter de petite idée que nous avons.

2. Notre projet.

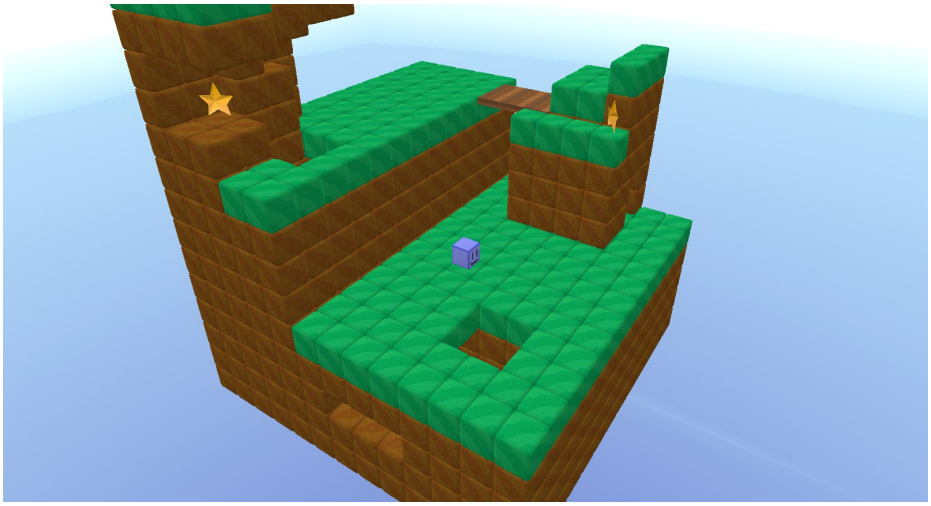

Illustration 2: GraviCube

A travers la dizaine de niveaux, le joueur devra être observateur et trouver les étoiles qui s'y cachent. Notre projet rajoute une mécanique de gameplay : l'inversion de la gravité. Nous avons pu utiliser notre imagination pour réaliser des niveaux plus ou moins complexes afin de mettre en avant les fonctionnalités de notre jeu.

II. Programmation

1. Game Loop

Le déroulement de la game loop se passe dans le fichier *mainwidget.ccp.* Ce fichier est le fichier principale de notre projet. Dans ce fichier, deux fonctions sont importantes. En effet, la fonction `void MainWidget::Update()` et la fonction `void MainWidget::paintGL()`.

La fonction `Update()` s'occupe de tout les calculs nécessaires pour le bon déroulement du jeu. Cette fonction se découpe en plusieurs phases :

- Le calcul de la position de la caméra en fonction de la position du joueur et des contrôles du joueur
- L 'exécution des composants de tous les GameObjects.
- Le test pour savoir si le joueur est en dehors des limites du jeu
- Le test pour savoir si le joueur à obtenue toutes les étoiles.

La fonction `paintGL()`, elle, s'occupe de l'affichage des GameObjects. Pour se faire, elle calcule la matrice MVP (Model View Projection) et l'envoie au shader. Cette fonction met à jour aussi les buffers pour l'affichage.

2. Collisions

Les GameObjects peuvent posséder un collider. Ce collider permettra à l'objet d'avoir une représentation physique. Pour l'instant, le seul composant collider disponible est un Box Collider. La classe *boxcollidercomponent.cpp* contient plusieurs méthodes permettant d'utiliser ce collider :

`bool BoxColliderComponent::Collide(BoxColliderComponent *collider) :`

Cette fonction, qui renvoie un booléen, permet de déterminer si deux colliders sont en collisions. Étant donnée que ces colliders sont des boîtes, les calculs sont simple et rapide.

`void BoxColliderComponent::Move(QVector3D moveVect):`

En utilisant la fonction `Collide`, cette fonction permet de déplacer le collider en fonction d'un vecteur `moveVect`. Cette fonction teste donc s'il est possible à ce collider de se déplacer sans entrer en collision avec un autre collider. S'il n'y a pas d'obstacle, le collider va déplacer le gameobject selon le vecteur souhaité.

Cependant, si un autre collider rentre en collision, alors on va diviser le vecteur de déplacement `moveVect` par deux, pour essayer de se rapprocher du gameobject. Cette méthode par dichotomie nous permet un gain de temps et de performance non négligeable. Si après 10 divisions successives, il persiste une collisions, on ne bouge pas le collider et l'objet reste à sa place.

`void BoxColliderComponent::Teleport(QVector3D pos):`

Cette simple fonction permet de téléporter à la position `pos` s'il n'y a pas de collisions à cette position.