Salmon Hippolyte
Lisee Jérôme
3ETI

Projet de TSI: Carl & Carlos Revolution

Sommaire:

I. Introduction

Dans le cadre de l'apprentissage d'OpenGL dans le module TSI, nous avons pu réaliser un jeu vidéo minimaliste avec python et OpenGL. Notre jeu se nomme 'Carl & Carlos Revolution'. Il reprend le principe du jeu Dance Dance Revolution et le croise avec un jeu de plateforme: on contrôle donc Carl et Carlos, deux dinosaures sur une plateforme d'herbe: des flèches arrivent des 4 points cardinaux en direction de la plateforme. Le but pour les dinosaures est de se placer sur le point cardinal correspondant afin de marquer des points.

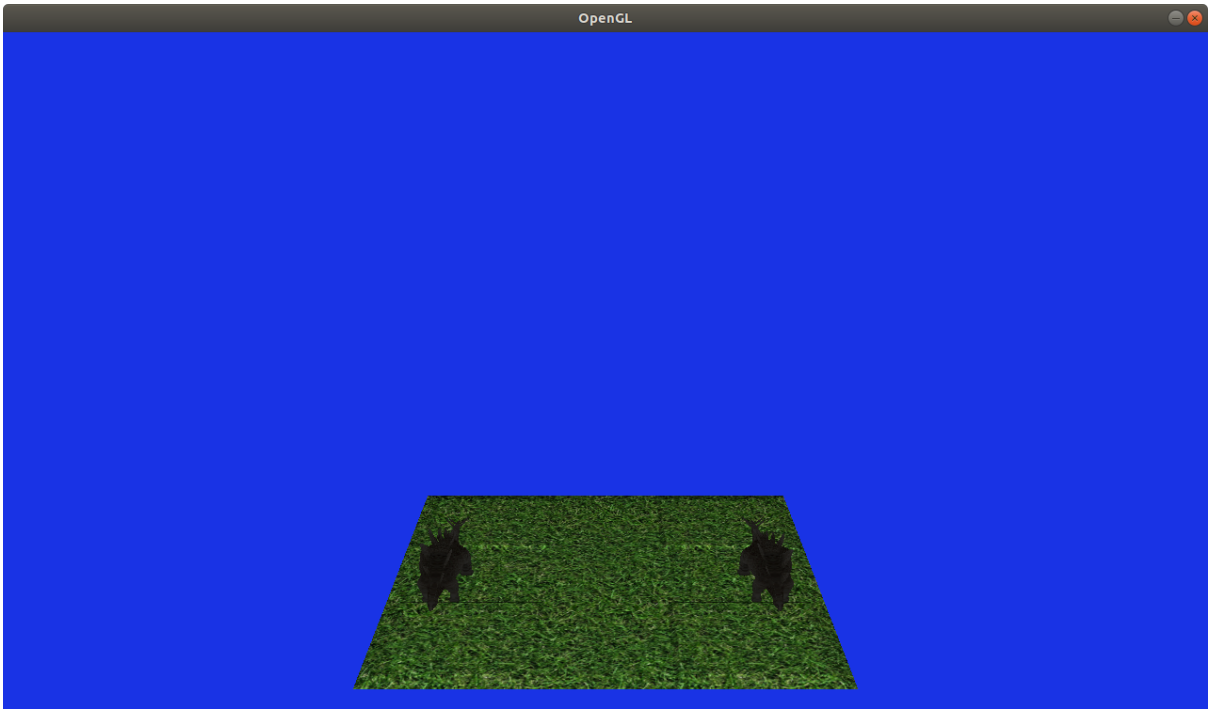

Figure 1: Image du jeu (sans les flèches)

II. Modules utilisés

Pour réaliser ce projet, nous avons utilisé plusieurs modules:

- glfw : Module de la gestion de la fenêtre, du contexte graphique et des événements (temps, appui des touches…)
- pyrr et numpy : module utiles pour des matrices et des tableaux, utiles pour les déplacements et rotations des objets
- pyopenGL : module permettant d'utiliser openGL sous Python
- sounddevice & soundfile : gestion de l'audio
- random : sert à générer des nombres aléatoires
- time : sert à pouvoir utiliser le passage du temps
- viewerGL : permet un interface graphique
- PIL : sert à pouvoir utiliser des images

III. Structure du code

Voici les fichiers dont est composé le programme :

fichiers shader:

- arrow.obj : contient les vertices et les fragments du mesh de la flèche.
- stegosaurus.obj : idem pour le stegosaurus.
- level.obj : idem pour la plateforme.
- shader.frag : sert à la gestion des fragments des objets.
- shader.vert: sert à la gestion des vertex des objets.
- gui.frag: sert à la gestion des fragments du texte.
- gui.vert: sert à la gestion des vertex du texte.

fichiers python:

- main.py : initialise les objets, la caméra, la musique, le texte et lance la boucle de jeu avec viewerGL. C'est le seul fichier à lancer pour faire tourner le programme.

- viewerGL.py : définit la classe viewerGL qui contient toute la boucle de jeu et ses paramètres, gère les intéractions entre les objets comme les collisions et la gravité, le lien entre l'utilisateur et le jeu via les touches, la disparition du texte et les mouvements de caméra.

- cpe3d.py : contient les classes d'Object, de texte, d'Object3D, de caméra, de Transformation3D (déplacement d'objets). Elles servent à procurer aux objets des fonctions fondamentales comme draw pour apparaître à l'écran.

- glutil.py : ajoute des fonctions à OpenGl pour utiliser des fichiers, des textures et des shaders. Notamment la fonction qui génère les program_ids de nos objets et qui traduit les fichiers shader de l'ascii vers le python.

- mesh.py : classe qui établit le lien entre les matrices de shaders et le gpu. Elle utilise des arrays pour load les objets dans le gpu.

- arrows.py : classe pour la classe des flèches, inutilisée autre que pour la texture et le load initial (lorsqu'on voulait utiliser des fonctions propres à la classe, on avait une erreur selon laquelle la fonction n'existait pas dans la classe mère Object3D que nous n'avons pas pu résoudre).

- level.py : classe pour la plateforme, inutilisée autre que pour la texture et le load initial. Possède une fonction qui retourne ses dimensions dans l'optique de limiter les mouvements des stégosaures mais elle n'a pas pu être implémentée (même problème).

- players.py : classe pour les deux stegosaurus jouables, inutilisée autre que pour la texture et le load initial. Possède une tentative de fonction de dimensions non conclusives qui aurait servi à rendre sa distance à la plateforme par collision dynamique vis à vis de la taille du stégosaure(même problème).

fichiers autres :

- fontB.jpg : lettres utilisées pour écrire le texte.
- grass.jpg : texture d'herbe utilisée pour la plateforme et la flèche.
- stegosaurus.jpg : texture de carl et carlos.
- music.wav : contient la musique du jeu.
- README.md : README contenant les liens à la github classroom.
- fichiers .mtl : artéfacts d'affichage.
- fichiers .pyc : artéfacts génériques.

IV. Fonctionnalités

- Les deux stégosaures peuvent être déplacés indépendamment avec les touches ZQSD et les flèches directionnelles.
- Les deux peuvent sauter indépendamment avec les touches V et spacebar. Ils obtiennent un vecteur vitesse verticale qui s'estompe progressivement avec le vecteur gravité opposé qui devient de plus en plus conséquent. Le vecteur gravité commence à augmenter seulement après le décollage du stégosaure, qui est donc initialisé avec une translation dans update_key() avant de procurer la vitesse verticale dans key_callback().
- Les stégosaures ont de la collision entre eux et avec la plateforme. Entre eux des orbes sont centrées sur leurs centres de gravité et lorsqu'elles se touchent, les stégosaures sont replacés à la limite de l'orbe de l'autre stégosaure. Pour la plateforme, le contact est fait avec le paramètre y de la plateforme et l'orb du stégosaure. Pour une raison inconnue, l'un d'entre eux est ancré au sol, il ne bougera pas si l'autre essaye de le pousser. Par contre, il peut pousser l'autre.
- La flèche spawn aléatoirement dans une des positions cardinales et se déplace (pas encore forcément vers la plateforme).
- Les touches ne sont accessibles qu'à la disparition du titre d'accueil.
- La caméra peut être tournée sur plusieurs axes via les touches IJKL.
- Les stégosaures subissent la gravité.

V. Conclusion et pistes à améliorer

Nous avons eu beaucoup de mal à utiliser des fonctions de classes et étions souvent bloqués dans nos avancements par les confusions system entre Object3D viewerGL et nos propres classes. Lorsque l'on voulait appeler une fonction d'une classe spécifique, nous obtenions une erreur expliquant que la classe mère Object3D n'avait pas cette fonction, bien que notre objet de sous classe oui.
De ce fait, plusieurs fonctions de classe sont créées mais pas utilisées.

Pour améliorer le jeu, il faudrait faire en sorte que les flèches avancent dans la direction de la plateforme et qu'à la collision, le jeu estime si le placement d'au moins un stégosaure est suffisamment proche sur l'axe x ou z (dépendamment de la direction d'arrivée de la flèche) et fait disparaître la flèche ou perdre le joueur.

Il faudrait aussi ajouter des obstacles traversant la plateforme qui désactivent les touches momentanément pour valoriser le saut.

Enfin, un système de score sur la longueur de la survie du joueur ou le nombre de flèches validées serait une bonne addition.