PHASE 3

Entrée #8 (14 Novembre)

État et avancement du projet

- Amélioration de l'architecture du moteur physique en gérant la boucle de jeu depuis le moteur (cela permet à l'utilisateur de seulement se concentrer sur la création de sa scène et plus sur la gestion de la physique aussi)
- Création d'une classe de GameObject permettant de faciliter la création d'objets dans la scène

Entrée #9 (22 Novembre)

État et avancement du projet

- Adaptation du moteur graphique pour gérer les matrices 34 et les rotations
- Ajout des rigidbodies et de leurs forces associées dans le moteur physique
- Scène d'exemple
- Quelques difficultés de compréhension concernant l'intégration, l'utilisation des matrices 34, et l'utilisation des coordonnées locales/globales ont été rencontrées, mais surmontées.
- Pour cette phase, on peut observer dans la scène deux cubes reliés par un ressort. Celui n'ayant pas de rotation est relié au ressort par son centre de masse, l'autre est relié par le dessus. On peut également instancier des cylindres subissant la gravité comme expliqué dans le Readme.md du projet.

PHASE 2

Entrée #4 (10 octobre 2022)

État et avancement du projet

- Amélioration des tests unitaires en utilisant les tests unitaires google tests pour simplifier l'exécution et la mise en oeuvre des tests unitaires

Entrée #5 (19 octobre)

État et avancement du projet

- Création de tests unitaires pour les matrices de transformation (scale, rotation, translation, projection, view et normal) à l'aide de la bibliothèque glm
- Implémentation des générateurs de forces

Entrée #6 (27 octobre)

État et avancement du projet

- Amélioration de l'architecture du projet en utilisant le modèle ECS (entité système composant) en s'inspirant grandement de ce qui a été fait ici : https://austinmorlan.com/posts/entity_component_system/
  Permet de gérer plus facilement les objets de la scène et de leur ajouter des propriétés facilement
- Création de différents systèmes pour le modèle ECS
  - Time pour gérer le temps
  - Physic pour mettre à jour les objets de la scène en fonction de la physique
  - InputsManager pour gérer les inputs de glfw
  - Render pour afficher la scène
  - Logic pour pouvoir ajouter des scripts aux objets de la scène (avec le Component LogicBehaviour)
- Implémentation des collisions
  - Résolveur de collisions
  - Classes de base (dont résolveur d'impulsions et résolveur d'interpénétrations)
  - Bâton
  - Câble
- Tests des générateurs de force

Entrée #7 (01 novembre)

État et avancement du projet

- Création du component Material pour pouvoir avoir un rendu graphique différent pour chaque objet de la scène
- Création du blob avec des collisions de type cables et du blob avec des collisions de type rod. Les particules dans les deux blob sont toutes reliées entre elles par des ressorts pour force.
- Ajout des générateurs de contact de type mur et contact de particule naïf
  - Un mur est constitué d'un centre, d'une épaisseur et d'une normale. On génère une interpénétration orientée en fonction de si la distance au centre de la particule est inférieure à la moitié de l'épaisseur.
  - L'idée derrière le contact de particules est de résoudre le contact avec la plus grande interpénétration, on itère sur chaque couple de la liste pour trouver ceux qui sont le plus pénétrés puis on crée le contact correspondant
- Test des collisions de type Bâton et Câble
- Tests des différentes forces, notamment les ressorts et corrections de bugs

PHASE 1

Entrée #1 (14 Septembre 2022)

État et avancement du projet

- Implémentation des classes Matrice et Vecteur
    - La classe Matrice utilise une taille générique mais ses fonctions spécifiques utiliseront la taille 3x3
    - Le header de Vecteur est complet
- Support de base OpenGL avec GLSW : Notre *main* peut ouvrir une simple fenêtre

TODO

- Finaliser l'implémentation des méthodes de la classe Vecteur
- Faire la classe particule
- Faire le programme de test
- Ajouter le support graphique des éléments

Entrée #2 (21 septembre 2022)

État et avancement du projet

- Implémentation d'une classe Object3D pour représenter les objets en 3D et de Sphere3D qui hérite de cette classe qui permet de représenter une sphère en 3D pour représenter la particule.
- Création d'une classe permettant de gérer l'affichage avec OpenGL.
- Création d'une classe Shader pour gérer les vertex et fragment shaders.
- Pour l'instant, on arrive à afficher une Sphère au centre de l'écran avec un shader basique gérant l'illumination par une source lumineuse.

TODO

- Améliorer la gestion de l'affichage des objets en 3D en créant un objet caméra et implémenter les calculs des matrices permettant de passer des coordonnées du monde aux coordonnées dans la fenêtre OpenGL

Entrée #3 (27 septembre 2022)

État et avancement du projet

- Création et test de la classe Matrix4D
- Partie graphique :
    - Terminer de faire les matrices view, transformation et projection pour les passer à glsl pour afficher les objets en 3d.
    - Amélioration de l'architecture du projet en créant une classe GameObject qui possède une classe Object3d représentant les points du GameObject, et qui peut posséder une classe Particule qui va gérer sa gravité.
    - Création de la classe scène et des classes light et camera pour représenter la scéne 3D.
    - Ajout de la gestion à openglmanager du fait que l'on peut afficher plusieurs gameobjects sur l'écran.
- Test de la classe Particule et implémentation des méthodes de mise à jour du mouvement avec prise en compte simple des frottement et du frame rate
- Actualisation du frame rate dans la boucle de jeu
- Possibilité de lancer plusieurs types de projectiles dans la boucle de jeu
    - Les projectiles suivent un mouvement parabolique simple et l'utilisateur peut les lancer en utilisant les touches numérotées du clavier
    - 3 projectiles simples : un "normal", un "léger" et un "lourd"