Goals and main task(s): Import the Bullet physics engine and create colliders for collision detection.

Tasks: Imported Bullet physics engine

Reflections: I decided to use the Bullet physics engine because it has a lot of features and is written in native code. I compiled it using CMake and imported it into my framework.

Tasks: Created colliders

Reflections: I created data structures called colliders that are used for detecting collisions. Every collider has an update and apply central impulse function. A collider also contains a shape that defines its collision “box”. There are four types of colliders: Base Collider, Box Collider, Capsule Collider, and Sphere Collider. The code for the abstract base collider is shown below.

Below: The code for the Base Collider class

#include "BaseCollider.h"

#include "../Engine.h"

#include "../Transform.h"

#include <glm/gtc/type\_ptr.hpp>

#include <glm/gtx/matrix\_decompose.hpp>

BaseCollider::BaseCollider(btCollisionShape \*shape, float mass)

{

m\_shape = shape;

bool isDynamic = (mass != 0.f);

btVector3 localInertia(0, 0, 0);

if (isDynamic)

m\_shape->calculateLocalInertia(mass, localInertia);

m\_myMotionState = new btDefaultMotionState(btTransform(btQuaternion(0, 0, 0, 1), btVector3(0, 0, 0)));

btRigidBody::btRigidBodyConstructionInfo rbInfo(mass, m\_myMotionState, m\_shape, localInertia);

m\_body = new btRigidBody(rbInfo);

}

BaseCollider::~BaseCollider(void)

{

delete m\_myMotionState;

delete m\_body;

delete m\_shape;

}

void BaseCollider::update(Input \*input, std::chrono::microseconds delta)

{

if (m\_body && !m\_body->isStaticObject() && m\_body->getMotionState())

{

btTransform trans;

m\_body->getMotionState()->getWorldTransform(trans);

glm::mat4 test;

trans.getOpenGLMatrix(glm::value\_ptr(test));

glm::vec3 scale;

glm::quat rotation;

glm::vec3 translation;

glm::vec3 skew;

glm::vec4 perspective;

glm::decompose(test, scale, rotation, translation, skew, perspective);

m\_parentEntity->getTransform().setPosition(translation);

m\_parentEntity->getTransform().setRotation(glm::quat(rotation.w, -rotation.x, -rotation.y, -rotation.z));

}

};

void BaseCollider::registerWithEngine(Engine \*engine)

{

auto pos = m\_parentEntity->getPosition();

auto rot = m\_parentEntity->getTransform().getRotation();

m\_body->setWorldTransform(btTransform(btQuaternion(-rot.x, -rot.y, -rot.z, rot.w), btVector3(pos.x, pos.y, pos.z)));

m\_body->setUserPointer((void \*)m\_parentEntity);

engine->getPhysicsManager()->registerCollider(m\_body);

}

void BaseCollider::deregisterFromEngine(Engine \*engine)

{

engine->getPhysicsManager()->deregisterCollider(m\_body);

}

void BaseCollider::applyCentralImpulse(const glm::vec3 &impulse)

{

m\_body->applyCentralImpulse(btVector3(impulse.x, impulse.y, impulse.z));

}