GDPHYSX PHASE 1 DOCUMENTATION

CLASSES AND PUBLIC FUNCTIONS OF THE ENGINE

Under Camera Folder

`MyCamera.h/cpp`

-The base class for the two cameras, their common attributes such as `z_far and z_near` will be stored here to avoid redundancy.

`OrthoCamera.h/cpp`

- This class provides the view and projection matrix of an orthographic camera.

`glm::mat4 OrthoCamera::giveView()`	This function returns the view matrix of orthographic camera
`glm::mat4 OrthoCamera::giveProjection()`	This function returns the projection matrix of orthographic camera

`PerspectiveCamera.h/cpp`

- This class provides the view and projection matrix of a perspective camera.

`glm::vec3 getPosition()`	Gets the value of the field position vector.
`glm::mat4 PerspectiveCamera::giveView(int type`	This function returns the view matrix of the perspective camera by using glm::lookAt() function.
`glm::mat4 PerspectiveCamera::giveProjection (float width, float height)`	This function returns the projection matrix of the perspective camera through the use of glm::perspective() function.

Under Model Folder

Model3D.h/cpp

- This class handles the attributes and functions that are needed by the model to be rendered.

void loadModel()	This function loads the model by storing its mesh indices and also calls the bindBuffers() function.
void drawModel()	This function manages the transformation and rendering of the model.
void setPosition(MyVector position)	Sets the field position vector according to the parameter position.
void setScale(glm::vec3 scale)	Sets the field scale vector according to the parameter scale.
void setColor(glm::vec4 color)	Sets the field color vector according to the parameter color.
void setShader(GLuint shaderProg)	Sets the shader program vector according to the parameter.
void setCameraProperties(glm::mat4 projection, glm::mat4 viewMatrix)	Sets the view matrix and the projection matrix respective to the parameters.

ModelManager.h/cpp

- This class is used for compiling all the registered models that are needed to render.

void AddModel(Model3D* toAdd);	This method adds the model from the parameter to the field list of models

Shader.h/cpp

- This class handles the procedure of setting up the shader for the engine

GLuint createShader(std::string fileVert, std::string fileFrag);	This function links the vertex and fragment shader to the shader program.
void loadVertAndFrag(std::string fileVert, std::string fileFrag);	This function loads vertex and fragment shader files.
void deleteShader();	This method is typically called after shaderProg has been set up

Under P6 Folder

DragForceGenerator.h/cpp

- This class is used for generating the drag force that will be applied to the particles.

void DragForceGenerator::UpdateForce(P6Particle * particle, float time)	This method is used for applying drag or to slow down a particle

ForceGenerator.h/cpp

- This class is for managing the types of forces that will be applied on the particles. The registration and deletion of forces are handled here.

GravityForceGenerator.h/cpp

- This class is used for generating the drag force that will be applied to the particles.

void GravityForceGenerator::UpdateForce(P6Parti	This function is used for applying gravity to the particle

cle* particle, float time)	

MyVector.h/cpp

- This class is used for generating the drag force that will be applied to the particles.

float getMagnitude()	This method is used for getting the magnitude of a vector.
MyVector normalize()	This method is used for applying normalization to a vector.
MyVector getDirection()	This method gets the direction of a vector.
MyVector operator+ (const MyVector v)	Operation for vector addition
void operator+= (const MyVector v)	Operation for vector addition
MyVector operator- (const MyVector v)	Operation for vector subtraction
void operator-= (const MyVector v)	Operation for vector subtraction
MyVector operator* (const float f)	Operation for scalar multiplication
void scalarMultiplication(const float f)	Operation for scalar multiplication
MyVector operator* (const MyVector v)	Applies Component product
void operator*=(const MyVector v)	Applies Component product
float scalarProduct(const MyVector v)	Applies Scalar product
MyVector getCrossProduct(const MyVector v)	Applies Vector product

P6Particle.h/cpp

- This class contains the properties of the particles such as position, velocity, acceleration, etc. This is also where the updates of the particles are done.

void UpdatePosition(float time)	This function is used for updating the position of the particle
void UpdateVelocity(float time)	This function is used for updating the velocity of the particle
void Update(float time)	This function manages the update of the properties of the particle. Properties such as position, velocity and forces associated with it.
void Destroy()	This function sets the flag of the particle on being destroyed.
bool IsDestroyed()	Returns a boolean whether a particle is destroyed or not.
void AddForce(MyVector force)	This function updates the accumulated force.
void ResetForce()	This function resets the force so that it won't apply force to particles unnecessarily

PhysicsWorld.h/cpp

- This class contains the list of particles generated inside the engine, the lifespan of said particles and the physics ongoing in each updated frame

void AddParticle(P6Particle* toAdd)	This method adds the parameter particle to the field list of particles.
void Update(float time)	This function handles the overall update on every particle added in the list.
void CheckLifespan(float	This function checks if the particle has exceeded their lifespan and calls Destroy() function if it did.

time)	
void UpdateParticleList()	This function checks all the particles registered if their isDestroyed flags is set to true, if so, remove them from the list.

RenderParticle.h/cpp -

- This class handles the association of the particles to the models by storing them through this class.

void draw()	This function is used for updating the position of the model based on the position of the particle, also used for calling the renderer of the object.

Utility.h/cpp -

- This class contains helper functions, particularly on generating random numbers

int getRandomNumber(int lowerBound, int upperBound)	This function is used for generating a random single number.
glm::vec3 getRandomVector(int lowerBound[], int upperBound[])	This function generates a random number thrice, respectively as x, y, z for the components of a vector to be returned.