Members:
1. Megan Florence Sophia C. Sadio
2. Yzabelle Anne A. Montuerto

Github Link:
- https://github.com/SADIO-M/MP-GDPHYSX

Physics Implementation Summary

Class : Vector	
Custom vector class for the physics engine.	
Public Functions	
void Vector()	Default constructor for vectors.
void Vector(const float _x, const float _y, const float _z)	Constructor which assigns the x, y, and z values of the vector.
float magnitude()	Returns the magnitude of the vector as a float value.
Vector direction()	Returns the direction of the vector as a unit vector.
component(const Vector RHS)	Returns the component product of the vector and the vector received, using the received vector as the right-hand-side of the calculation.
float dot(const Vector RHS)	Returns the dot product of the vector and the vector received, using the received vector as the right-hand-side of the calculation.
Vector cross(const Vector RHS)	Returns the cross product of the vector and the vector received, using the received vector as the right-hand-side of the calculation.
Operators	
Vector operator+ (const Vector RHS)	Adds the vector to the vector received *RHS* and returns the result. Does not alter the original vector.
Vector operator- (const Vector RHS)	Subtracts the vector received *RHS* from the vector and returns the result. Does not alter the original vector.

Vector operator* (const float scalar)	Multiplies the vector to the received float *scalar* and returns the result. Does not alter the original vector.
void operator+= (const Vector RHS)	Adds the vector to the vector received *RHS*. Directly modifies the original vector.
void operator-= (const Vector RHS)	Subtracts the vector received *RHS* from the original vector. Directly modifies the original vector.
void operator*= (const float scalar)	Multiplies the vector to the received float *scalar*. Directly modifies the original vector.

Class : Particle	
The engine's particle object.	
Public Variables	
float mass	The particle's mass. Set to 1 by default.
Vector position	The particle's current position in world space. Does not have an initial value.
Vector velocity	The particle's current velocity. Does not have an initial value.
Vector acceleration	The particle's current acceleration. Does not have an initial value.
float lifespan	The particle's lifespan. Defines how long the particle exists. Set to 5 (seconds) by default.
float radius	The particle's radius. Set to 50 by default.
float restitution	How bouncy or rigid a particle is. It goes from 0 (rigid) to 1 (bouncy).
Public Functions	
void update(float time)	Handles the updating of the particle's

	position, velocity, and lifespan. Also resets the forces applied to the particle.
void addForce(Vector force)	Applies the received Vector *force* to the particle.
void resetForce()	Resets the particle's acceleration and accumulated force.
void Destroy()	Sets the particle's status to destroyed.
bool IsDestroyed()	Returns the particle's status.
void operator*= (const float scalar)	Multiplies the vector to the received float *scalar*. Directly modifies the original vector.
Protected Variables	
float damping	A drag force applied to the particle. Set to 0.9 by default.
bool isDestroyed	The status of the particle.
Vector accumulatedForce	The force accumulated by the particle before the physics update.
Protected Functions	
void updatePosition(float time)	Called in **update(float time)**, updates the particle's position with the appropriate calculations.
void updateVelocity(float time)	Called in **update(float time)**, updates the particle's velocity with the appropriate calculations.

Class : PhysicsWorld	
Handles physics operations.	
Public Variables	
ForceRegistry forceRegistry	A **ForceRegistry** instance. Contains all forces being applied to any object in the PhysicsWorld.

list<Particle*> Particles	List of all particles being handled by PhysicsWorld.
Public Functions	
void addParticle(Particle* toAdd)	Adds the received Particle* *toAdd* to the list of all particles (*Particles*).
void update(float time)	Handles the updating for all particles and forces.
list<Particle*>* getParticleList()	Returns *Particles*.
Particle* getParticleAtIndex(int index)	Returns a specific particle from *Particles* at the specified *index*.

Class : ForceRegistry	
A registry for managing particles and the forces that affect it.	
Public Functions	
void add(Particle* particle, ForceGenerator* generator)	Adds a force to the registry.
void remove(Particle* particle, ForceGenerator* generator)	Removes a force from the registry.
void clear()	Clears all forces.
void updateForces(float time)	Handles the updating/application of all forces in the registry.
Protected Variables	
struct ParticleForceRegistry { **Particle* particle; ForceGenerator* generator;** }	A struct containing a pointer to a particular particle and a **ForceGenerator** with which to apply a force to the object.
list<ParticleForceRegistry> Registry	A list of **ParticleForceRegistry**'s. The registry proper of the ForceRegistry

---

Class : ForceGenerator	
A base class for force generation.	
Public Functions	
virtual void updateForce(Particle* p, float time)	A function for updating/applying the force to the particle which can be overridden by classes inheriting this class.

Class : GravityForceGenerator : public ForceGenerator	
A force generator for Gravity which is a child of ***ForceGenerator****.*	
Public Functions	
GravityForceGenerator(const Vector grav)	The class constructor which receives a vector *grav* representing the force of gravity to be used.
virtual void updateForce(Particle* p, float time) override	A function for updating/applying gravity to the particle.

Class : DragForceGenerator : public ForceGenerator	
A force generator for drag forces which is a child of ***ForceGenerator****.*	
Public Functions	
DragForceGenerator()	The default constructor for this class..
DragForceGenerator(float newK1, float newK2)	A constructor for this class which accepts new values for the friction coefficients. The default value for *K1* is 0.74f and the default value for *K2* is 0.57f. Use this constructor to simulate drag forces by specifying newK1 and newK2
virtual void updateForce(Particle* p,	A function for updating/applying drag to

float time) override	the particle.

Class : RenderParticle	
A class that holds both the particle and its corresponding model.	
Public Variables	
bool isDestroyed	The status of the render particle.
Particle* physicsParticle	The actual particle.
Model3D* objectToRender	The 3D model the particle is rendered with.
Vector color	The color of the particle. This will override any color inherently assigned to **objectToRender**.
Public Functions	
RenderParticle()	The class's default constructor. Note that **physicsParticle** and **objectToRender** have no values by default and must be assigned before the render particle can be used.
RenderParticle(Particle* particle, Model3D* obj)	A constructor which creates a white render particle. *Particle* is assigned to **physicsParticle** and *obj* is assigned to **objectToRender**.
RenderParticle(Particle* particle, Model3D* obj, Vector color)	A constructor which creates a render particle with the specified color. *Particle* is assigned to **physicsParticle** and *obj* is assigned to **objectToRender**.
void draw()	Draws the **objectToRender** and updates its position based on the position of the **physicsParticle**.

Class : RenderParticleFactory	
A class that creates render particles without the need for manual assignment of the model and particle.	
Private Variables	
default_random_engine generator	Source of randomization.
PhysicsWorld* physicsWorld	A pointer to the **physicsWorld** present in the game. Has no value by default.
float minRadius	The minimum radius used by the randomizer. Has a default value of 2.0.
float maxRadius	The maximum radius used by the randomizer. Has a default value of 10.0.
float minLifespan	The minimum lifespan used by the randomizer. Has a default value of 1.0.
float maxLifespan	The maximum lifespan used by the randomizer. Has a default value of 10.0.
Public Functions	
RenderParticleFactory(PhysicsWorld* physWorld)	The constructor for this class. Requires a pointer to the game's **PhysicsWorld**.
RenderParticle* create()	Returns a fully made render particle. Automatically randomizes the particle's size, color, and lifespan. The **Particle** is also added to **PhysicsWorld**.
float randomizeFloat(float min, float max)	Returns a random float value between the received *min* and *max* values.

Class : ParticleContact	
Takes care of contacts between particles (resolves their velocity and interpenetration)	
Public Variables	
Particle* particles[2]	The two particles handled by the contact
float restitution	A number between 0 and 1 representing

	the “bounciness” of a contact. The coefficient of its elastic recoil.
float depth	The depth of the particle (how much it overlaps). Ideally is at 0.
Vector contactNormal	Used to calculate for separating speed
float separatingSpeed	The separating speed of the particles
Public Functions	
void resolve(float time)	Resolve velocity and interpenetration
float getSeparatingSpeed()	Get the separating speed of the particles
Protected Functions	
void resolveVelocity(float time)	Resolve the velocity of the particle (when they bump, how do they move)
void resolveInterpenetration(float time)	Resolve interpenetration (when they overlap, how far should they go back)

Class : ContactResolver	
Responsible for resolving all contacts between particles	
Public Variables	
unsigned maxIterations	The max amount of iterations that can be made by the **resolveAllContacts()** function.
Public Functions	
ContactResolver(unsigned maxIter)	The class’s constructor which accepts a number to be assigned as its **maxIterations**.
void resolveAllContacts(vector<ParticleContact*> contacts, float time)	The algorithm used to resolve all the contacts in a given frame.
Protected Variables	

unsigned currentIterations	Keeps track of how many iterations the **resolveAllContacts()** function has gone through.

Class : ParticleLink	
For creating links between two particles or one particle and an anchor point	
Public Variables	
Particle* particles[2]	The two particles handled by the contact
Vector anchor	A vector containing the anchor point position (if the player links a particle to a point in space)
Line* lineLink	Used to store the "line" of the particle link (for rendering purposes). It is nullptr by default
Public Functions	
virtual ParticleContact* getContact()	This function should be overridden by its children Calculates particleContact and is used to manipulate link length
virtual void createLineLink()	Creates a line link and stores it in lineLink variable Gets the vertices of the connected particles or anchor point, then creates a render for that link
virtual void fixLineLink()	Fixes the line link by updating the vertices in the VBO, so that it follows the connected objects
Protected Functions	
float currentLength()	Calculates the current length of the link
void setAnchor(Vector anchor)	Sets the anchor point position

Class : Cable : public ParticleLink	
A class for creating cables that connects a particle and anchors it to a point in space	
Public Variables	
float cableLength	The length of the cable
float restitution	A number between 0 and 1 representing the “bounciness” of a contact. The coefficient of its elastic recoil.
Vector anchor	Similar to anchor in ParticleLink, also holds an anchor point for the cable.
Public Functions	
Cable()	Default constructor
Cable(float cableLength, Vector anchor)	Constructor to set length and anchor position
Cable(float cableLength, float restitution, Vector anchor)	Constructor to set length, anchor position, and custom restitution
ParticleContact* getContact() override	An override of ParticleLink’s getContact function. Calculates for and checks if the link’s current length is greater than the cable length, if it is, fix it.

Class Rod : public ParticleLink	
A class for creating rods that connect two particles together	
Public Variables	
float rodLength	The length of the rod
float restitution	A number between 0 and 1 representing the “bounciness” of a contact. The coefficient of its elastic recoil. However, since this is a rod, it is set to 0 by default.

Public Functions	
ParticleContact* getContact() override	An override of ParticleLink's getContact function. Calculates for and checks the rod's current length, to make sure it stays equal all throughout.

Class AnchoredSpring : public ForceGenerator	
A class for creating rods that connect two particles together	
Private Variables	
Vector anchorPoint	The position of the anchor point
float springConstant	A spring constant to also simulate bounciness or rigidness of the spring
float restLength	The rest length of the spring, or its default length
Public Functions	
AnchoredSpring(Vector position, float springConst, float restLen)	Default Constructor
void updateForce(Particle* particle, float time) override	An override of **ForceGenerator**'s **updateForce** function. It calculates for the force applied by the spring onto the particle to simulate the bounce. The difference here and **ParticleSpring** is that its an **anchorPoint**'s position, not another particle

---

Class ParticleSpring: public ForceGenerator
A class for creating a spring connected between two particles
Private Variables

Particle* otherParticle	The particle that this current particle is attached to
float springConstant	A spring constant to also simulate bounciness or rigidness of the spring
float restLength	The rest length of the spring, or its default length
Public Functions	
ParticleSpring(Particle* particle, float springConst, float restLen)	Default Constructor
void updateForce(Particle* particle, float time) override	An override of **ForceGenerator**'s **updateForce** function. It calculates for the force applied by the spring onto the particle to simulate the bounce. The difference here and **AnchorSpring** is that its **otherParticle**'s position, not an anchor point