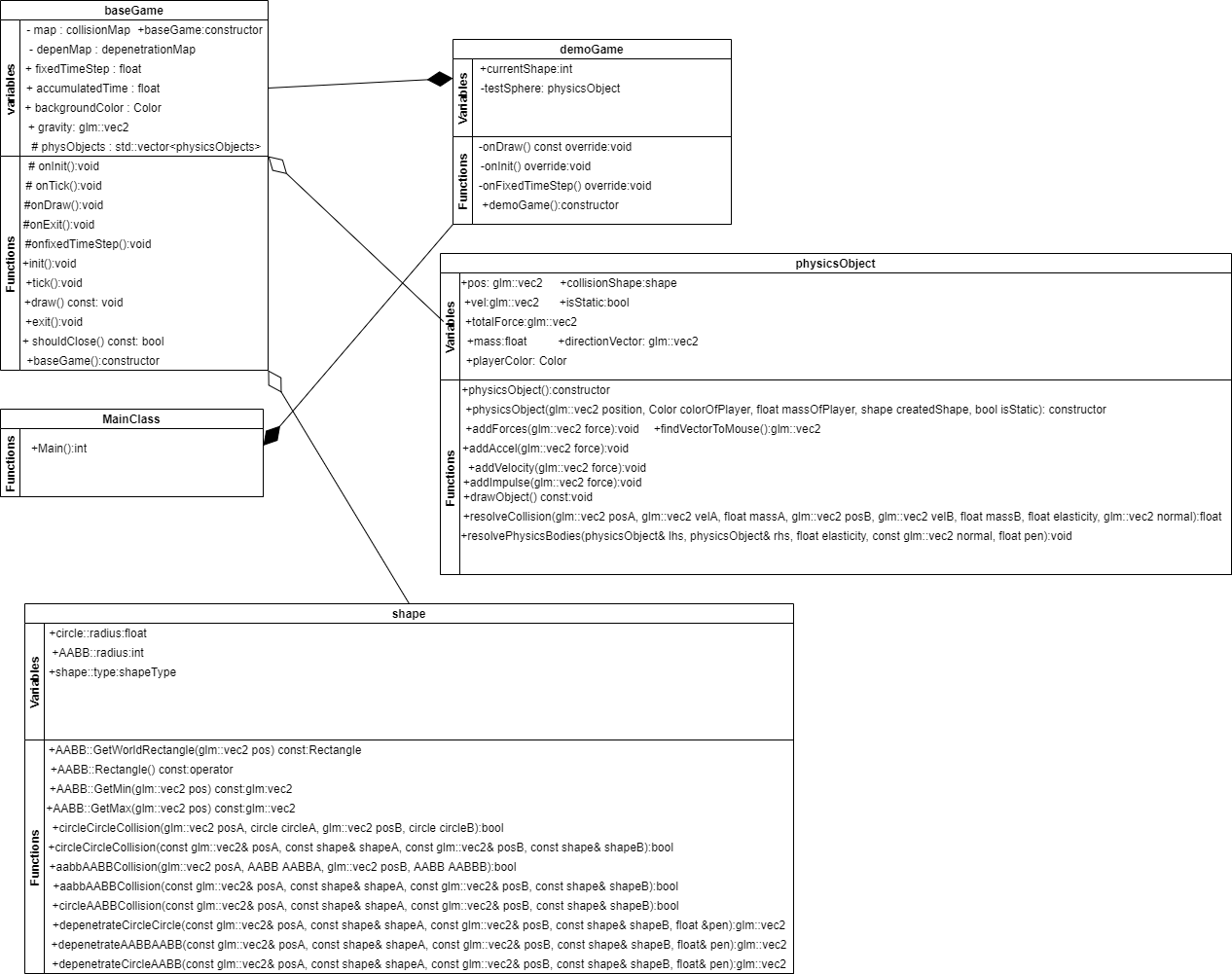
Custom Physics Simulation

Documentation for Joseph’s Physics simulation

# Demonstration Brief

My application allows the user to place circles and squares that are affected by gravity as well as a square that stays static even after colliding with other objects. My application also handles the various collisions between each shape combination provided. Finally, to help the user control this collision a bit more pressing the left mouse button also allows the user to pull objects to the mouse position by applying a force on them based on their distance from the mouse.

# Class Diagram



The above diagram was created using <draw.io> in “Class Diagram” mode. Elements for things like the “composition” and “aggregation” associations can be found in the element palette in the bottom-left corner of the interface.

# Research Material

# <https://www.geeksforgeeks.org/header-files-in-c-cpp-and-its-uses/>

# Third Party Libraries

## [GitHub - g-truc/glm: OpenGL Mathematics (GLM)](https://github.com/g-truc/glm)

[Release raylib v3.0.0 · raysan5/raylib · GitHub](https://github.com/raysan5/raylib/releases/tag/3.0.0)

# Public API

## baseGame

### game() - constructor

Initialize all variables to suitable defaults.

### float fixedTimeStep

The target amount of time between time steps as expressed in seconds. Evaluated by the shouldTickPhysics() function to determine if a physics update is due.

### void init()

Create our window and initializes our rendering context.

### void draw()

controls what we draw in the application and what we display on the screen for the user.

### void exit()

Called when closing the game and shutting down the program, add in methods for when the application closes into here.

bool shouldClose() const

This method will return true if the application is trying to close.

### float fixedTimestep

This variable is our fixed time step that we use to create an update to run our physics checks in.

### float accumulatedTime

This variable is our accumulated time which we use to tell when fixed update should start.

### Color backgroundColor

This is the color for our demo scene’s background.

### glm::vec2 gravity

This is our gravity variable for objects that use our physics body script.

## DemoGame

### demoGame() – constructor

Default constructor for the physics simulation.

### currentShape

Defines the shapes current type either a circle or AABB, used in the demo to switch between these types.

## Shape

### struct circle

a struct that defines the variables for a circle phyics object, this gives a radius for your circle collision methods, depenetration methods and constructor.

### struct AABB

a struct that defines the variables for an AABB phyics object, this gives an offset for creating the square for your collision methods, depenetration methods and constructor, it also provides a conversion method for AABB to rectangle and 2 methods that get the AABB’s minimum and maximum’s position in world space.

### enum class shapeType : uint8\_t

An 8 byte int that lets us know which type we’re using when a shape is made, helps us go between our collision and depenetration methods, when editing make sure to add in your new types here.

### struct shape

generic struct for the above 2 structs it has a constructor and also a union for each shape type.

### Collision(const glm::vec2& posA, const shape& shapeA, const glm::vec2& posB, const shape& shapeB)

The standard collision method parameters I included a collision method for each interaction between the provided shapes as well as a defined version of each method for that took in simply the given shape instead of a generic const shape reference. These are attached to the maps so when creating more structs expand this as needed.

### Depenetration(const glm::vec2& posA, const shape& shapeA, const glm::vec2& posB, const shape& shapeB, float &pen)

Same situation as the collision methods they are controlled by a map so when creating or adding new structs create a new map and method here with the above parameters.

## physicsObject

### glm::vec2 pos

a vector that holds the position of the physics object.

### glm::vec2 vel

A vector that holds the velocity of the physics object.

### Glm::vec2 totalForce

A vector that holds the total force applied to the current physics object.

### Float mass

Mass of the object currently just defaulted.

### Color playerColor

Intended to be used to differentiate from objects in the scene but at the moment is defaulted for all spawned shapes.

### Shape collisionShape

Holds the shapetype that will be made.

### Bool isStatic

This variable stops static objects from being affected by gravity and other forces in the scene.

### Glm::vec2 directionVector

This vector is used to store the direction the object has to move in to move towards the mouse.

### physicsObject(glm::vec2 position, Color colorOfPlayer, float massOfPlayer, shape createdShape, bool IsStatic)

A constructor that takes in all the variables needed for shapes in the current version.

### void onTickPhys(float delta)

The physics update for this application that checks and resolves collision.

### void addForces(glm::vec2 force)

Adds a force to the physics object.

### void addAccel(glm::vec2 force)

Adds an acceleration force to the object.

### void addVelocity(glm::vec2 force)

Adds a velocity force to the object.

### void addImpulse(glm::vec2 force)

Adds an impulse force to the object.

### void drawObject() const

The method to draw the object, update when more object shapes have been added.

### glm::vec2 findVectorToMouse()

A method that returns the vector towards the mouse position on screen.

### float resolveCollision(glm::vec2 posA, glm::vec2 velA, float massA, glm::vec2 posB, glm::vec2 velB, float massB, float elasticity, glm::vec2 normal)

This method resolves all the collision going on with this specific physics object.

### void resolvePhysicsBodies(physicsObject& lhs, physicsObject& rhs, float elasticity, const glm::vec2 normal, float pen)

This moves the physics bodies to their proper locations after all applicable forces have been applied to them.

Potential Future Improvements

For static objects the only spawnable one is an AABB, in the future I’d like to improve this to include circles.

There are 2 known bugs at the moment with collision;

circle AABB collision if both objects are moving fast enough, they will hit a normal high enough that the objects will cease to exist

AABB and static AABB collision is a bit jittery when they are colliding by the corners.

# Credits

Joseph Huskey © 2020