Custom physics engine

Game: Carrom

CREATED By: SARTHak saxena

Contents

[Physics engine class diagram 1](#_Toc34680947)

## Physics Engine Class Diagram

## Physics Class Systems

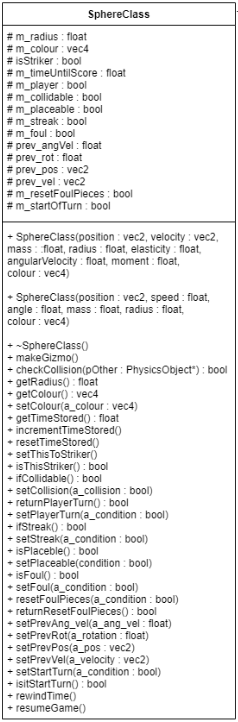
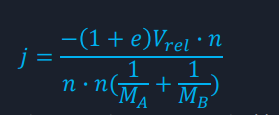
The physics being used factors in Dyanmic and Static Dynamic bodies in a kinematic scenario. The usage of the Physics system is displayed through the game ‘CARROM’, which uses three main properties AABB, Spheres, and Planes.   
  
These three classes inherit properties from the class PhysicsObject which keeps data of shape type, and the collision required. Physics Objects creates an instance of a physics object, of either a Dynamic object, or a static object. For Dynamic objects a Rigidbody class is created, holding velocity, position, rotation, moment, angle, speed, mass, angular velocity, elasticity, and shape id. The id lets the previously mentioned AABBClass and SphereClass hold shape id when inheriting properties from the RigidBodyClass. The PlaneClass is not a Dynamic Rigidbody, so it won’t inherit properties from Rigidbody, and rather be created through PhysicsObject only.   
  
Class Diagram of Sphere Class:  


Figure 1. Sphere Class Diagram

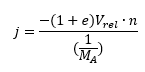
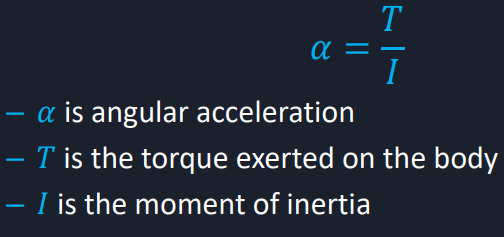
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Game Objects | Class | Properties | Functions used | Type |
| Sphere, centreSphere, sphereInner1, sphereInner2, sphereInner3, sphereInner4, sphereInner5, sphereInner6, sphereInner7, sphereInner8. | SphereClass | Mentioned in (Figure 1.) Sphere Class which inherits properties from Rigid body Class.  Instantiated through the first constructor in Sphere Class. | Sphere Class, which inherits functions from Rigid body Class as well. Refer to Figure1. | Dynamic |
| CoinsInScene | SphereClass | Mentioned in (Figure 1.) Sphere Class which inherits properties from Rigid body Class. | Sphere Class, which inherits functions from Rigid body Class as well. Refer to Figure1. | Vector of all Sphere Class Game Objects created. |
| topLeftHole, topRightHole, bottomLeftHole, bottomRightHole, midLeftObs, midRightObs | AABBClass | Mentioned in (Figure 2.) AABB Class which inherits properties from Rigid body Class. Though for game purposes it will act as a static object, eg. Wall. | AABB Class functions, which inherits functions from Rigid body Class as well. Refer to Figure 2. | Static |
| bottomPlane, topPlane,  rightPlane, leftPlane | Plane Class | Mentioned in (Figure 3.) Plane Class. | Plane Class Functions | Static |

#### **Interactions Between AABB, Sphere, and Planes**

In terms of Physics Engine

The interactions between AABB, Sphere, and Planes happen each fixed update (by passing through fixed Timestep), which determines if any interaction has been made. An interaction is checked through the Physics Object Class.cpp which takes nine types of collision checks. An interaction can be AABB to sphere, AABB to AABB, AABB to Plane, and the same for Spheres and Planes following the same format. The engine then runs a check through a fixed time step, to remove framerate dependant physics interaction issues. The collision checks if interaction has occurred by checking through all actors (physics objects) in scene, checking their shape id to determine the correct collision check to be started. Once the correct collision check has begun the variables get assigned accordingly and the correct physics forces, and properties get applied to both the physics objects through the Rigid body class function resolve collision and applyForce. This then takes both physics objects, calculates the vecor between their centres or the provided direction of force and normalizes it. Then to determine the total velocity of the contact points of the two objects for both linear and rotational, it gets the vector perpendicular to the collision normal. From which it determines the radius from axis to the application of force, by applying the perpendicular vector to the contact point minus the position of the actor. Velocity of the contact point will be calculated by the dot product of the velocity and the normal minus the radius from the axis to the application of force. Then determine the impulse magnitude (Equation 1.) applied to both the actors.

Equation 1. Calculating the impulse magnitude.

Before moving on to how the force gets applied, due to velocity being a factor in calculating the impulse magnitude, the equation gets altered to produce the force for an interaction with plane. This is due to planes, e.g. Walls acting as static objects, which won’t move after being resolved of collision. The planes will act as static objects meaning they effectively will have an infinite mass. Due to this the following equation changes to this (refer to Equation 2.), where the mass of only one of the actors needs to be present in the equation. Thus, force is applied relative to the collision normal of the interaction and passed through the apply force function in Rigid Body Class. Rigid body Class applies force about Newton’s third law, “For every action, there is an equal and opposite reaction”. Therefore, the impulse force applied to one object, the opposite of that must be applied to the other, the velocity increases by force divided by mass and vice versa with negative force applied for the second physics object. Torque is then calculated through the following equation (Equation 3.), from which the angular acceleration of the physics objects can be calculated. Determines the angle from the already known radius from the axis of rotation to the point of application of force. Applies the radius, force and angle to calculate the torque needed, and subs it into the equation for angular acceleration.   
Each update the engine will run a simulated friction and gravity onto the physic objects, reducing velocity, rotation speed, and setting its position each time step.

Equation 4. Angular Acceleration Formula

Equation 3. Torque Formula

Equation 2. Calculating the impulse magnitude For Planes

## Third Party Library Used

Bootstrap   
IDE used – Visual Studio 2019