Custom physics engine

Game: Carrom

CREATED By: SARTHak saxena

Contents

[Physics Engine Class Diagram 2](#_Toc34751155)

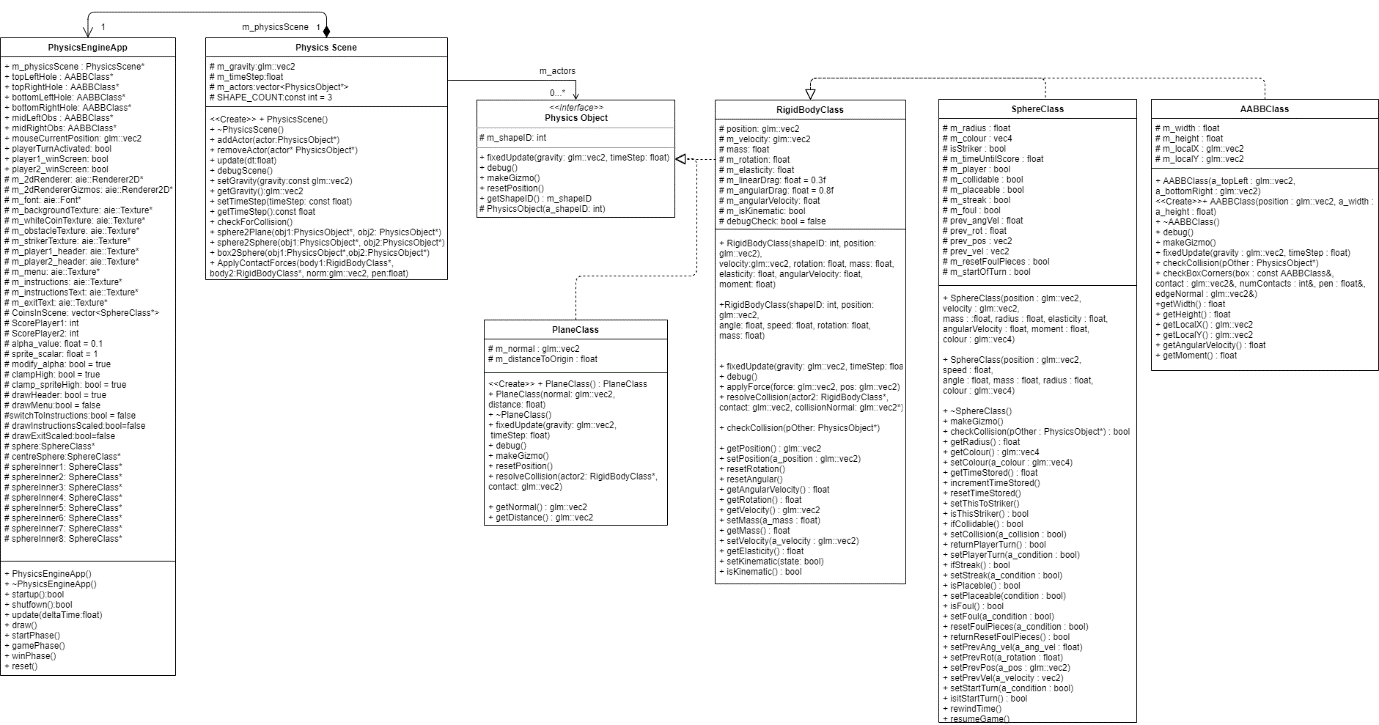
[Physics Class Systems 3](#_Toc34751156)

[Carrom - Game Overview 5](#_Toc34751157)

[Third Party Library Used 7](#_Toc34751158)

[Improvements For The Future 7](#_Toc34751159)

## Physics Engine Class Diagram



## Physics Class Systems

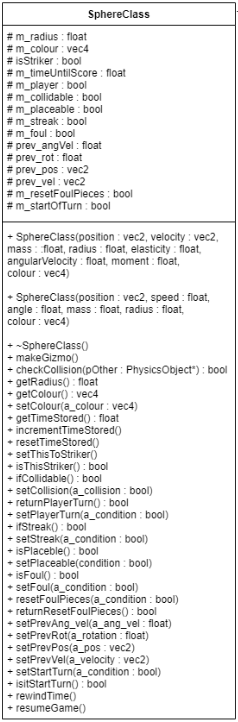
The physics being used factors in Dyanmic and Static Dynamic bodies in a kinematic scenario. The usage of the Physics system is displayed through the game ‘CARROM’, which uses three main properties AABB, Spheres, and Planes.   
  
These three classes inherit properties from the class PhysicsObject which keeps data of shape type, and Physics Scene class holds the collision required. Physics Objects creates an instance of a physics object, of either a Dynamic object, or a static object, and places it into the physics scene. 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

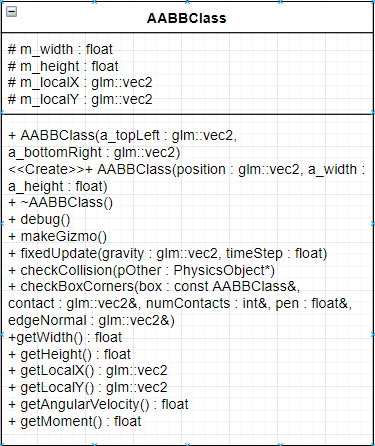


Figure 2. AABB Class Diagram

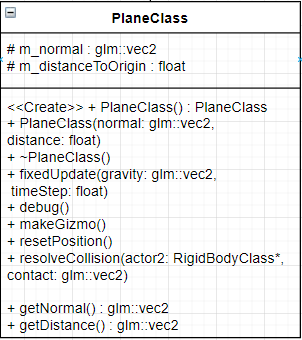


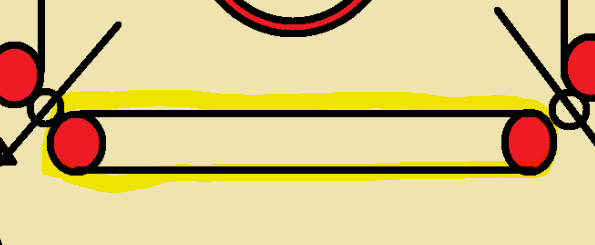
Figure 3. Plane Class Diagram

### Carrom - Game Overview

Game Flow-

Versus

The game includes two players taking turns trying to score more than the other player. The game starts off with all white coins in a fixed position in the centre with the striker being on the (refer to Game 1.) play area. The player can shoot the white coins and depending on whether the white coin goes in the corner pocket, that player can take their turn again. If the player does hit the striker and no white coin is hit the it will be the other player’s turn, and they get a chance to shoot.



Game 1. Play area highlighted

Rules of Carrom

* The player cannot shoot the Striker directly back or horizontal in any direction to hit a white coin. This is because any white coin on the play area where the striker is placed counts as a foul.
* The player cannot shoot a white coin and lose the striker in the process in the corner pocket as well. This will count as a foul and the play will be reset and move to the other player’s turn.
* If the player shoots a white coin on the play area where the Striker is placed, it will reset the play and change turns.

Controls

The controls for the keyboard are listed below:

|  |  |
| --- | --- |
| Key | Action |
| A | The striker will move to the left. |
| D | The striker will move to the right. |
| Escape | Opens the main menu. |

The controls for the mouse are listed below:

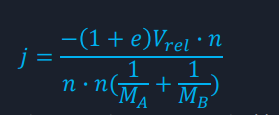
|  |  |
| --- | --- |
| Button | Action |
| (Hold) Right-Click | Displays (valid) green or (invalid) red to scale the power applied and direction to the Striker. |
| (ACTIVATED WHEN RIGHT CLICK HELD)  Left-Click | Finalizes the direction and power, so the physics engine can run the correct physics simulation. |

GAME OBJECTS USED IN CARROM EFFECTED BY PHYSICS ENGINE

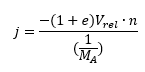
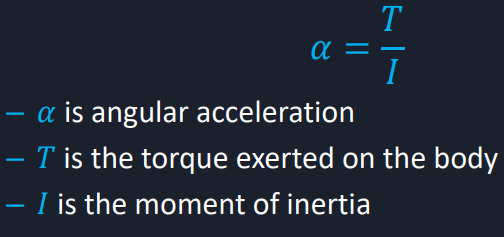
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Game Objects | Class | Properties | Functions used | Type |
| Sphere, centreSphere, sphereInner1, sphereInner2, sphereInner3, sphereInner4, sphereInner5, sphereInner6, sphereInner7, sphereInner8. | SphereClass | Mentioned in (Figure 1.) Sphere Class 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 Scene 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 functions, Resolve Collision and Apply Force. This then takes both physics objects, calculates the vector 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 determines 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

#### **Interactions Between AABB, Sphere, and Planes (continued…)**

In terms of the game

The interactions between AABB, Sphere, and Planes are depicted through the game simulated. The game is based upon three main collision checks which include sphere to plane, sphere to box, and sphere to sphere. As the ‘Player’ shoots the striker into the white coins, their force and angular acceleration will be determined to display how far and in which direction they will travel. The edges of the board will be an instance of the plane class which is a static object, which is why it is necessary for the sphere to plane collision check to be needed. For the game, there are two obstacles in the path of the ‘Player’ which are instances of the AABB Class. They are required to be static, to act as obstacles and therefore justifying the reason for the sphere to AABB collision check. As the game’s objective is to shoot the white coins in the corner pockets, the pockets are represented as AABB class instances that are trigger points. These trigger points act on the sphere to AABB collision check, which determines if the coin is colliding for more than x amount, from which they appear to ‘sink’. The game depends on sphere to sphere collision as ‘Player’ cannot shoot the striker in the corner pocket, as it would count as a foul and they will lose their turn.

## Third Party Library Used

Bootstrap   
IDE used – Visual Studio 2019

## Improvements For The Future

The Physics engine considers only primitive shapes such as rectangle and sphere. It needs to be able to function properly with the addition of complex shapes. The use of separating axis theorem will increase precision as it is fast and accurate and allows early exit when there is no collision between polygons and complex shapes.

##### Bibliography - REFEREnces

The Physics Classroom. “Newton's Third Law of Motion.” *The Physics Classroom*, 2020, www.physicsclassroom.com/class/newtlaws/Lesson-4/Newton-s-Third-Law.

Hecker, Chris. “Physics, Part 3: Collision Response.” *Physics, Part 3: Collision Response*, 1997, [www.chrishecker.com/images/e/e7/Gdmphys3.pdf](http://www.chrishecker.com/images/e/e7/Gdmphys3.pdf).

Shiu, Kah, et al. “Collision Detection Using the Separating Axis Theorem.” *Game Development Envato Tuts+*, 6 Aug. 2012, gamedevelopment.tutsplus.com/tutorials/collision-detection-using-the-separating-axis-theorem--gamedev-169.

“3D Collision Detection.” Edited by MDN Contributers, *MDN Web Docs*, 2019, developer.mozilla.org/en-US/docs/Games/Techniques/3D\_collision\_detection.

Dunn, F, 2011. 3D Math Primer for Graphics and Game Development, 2nd Edition. 2. A K Peters/CRC Press.

HyperPhysics. 2017. Torque and Equilibrium. [ONLINE] Available at: http://hyperphysics.phy-astr.gsu.edu/hbase/torq.html. [Accessed 21 June 2017].

HyperPhysics. 2017. Moment of Inertia. [ONLINE] Available at: http://hyperphysics.phy-astr.gsu.edu/hbase/mi.html. [Accessed 21 June 2017].

Kent. “Home.” *Studio Freya*, 15 Oct. 2019, studiofreya.com/3d-math-and-physics/sphere-vs-aabb-collision-detection-test/.