2023

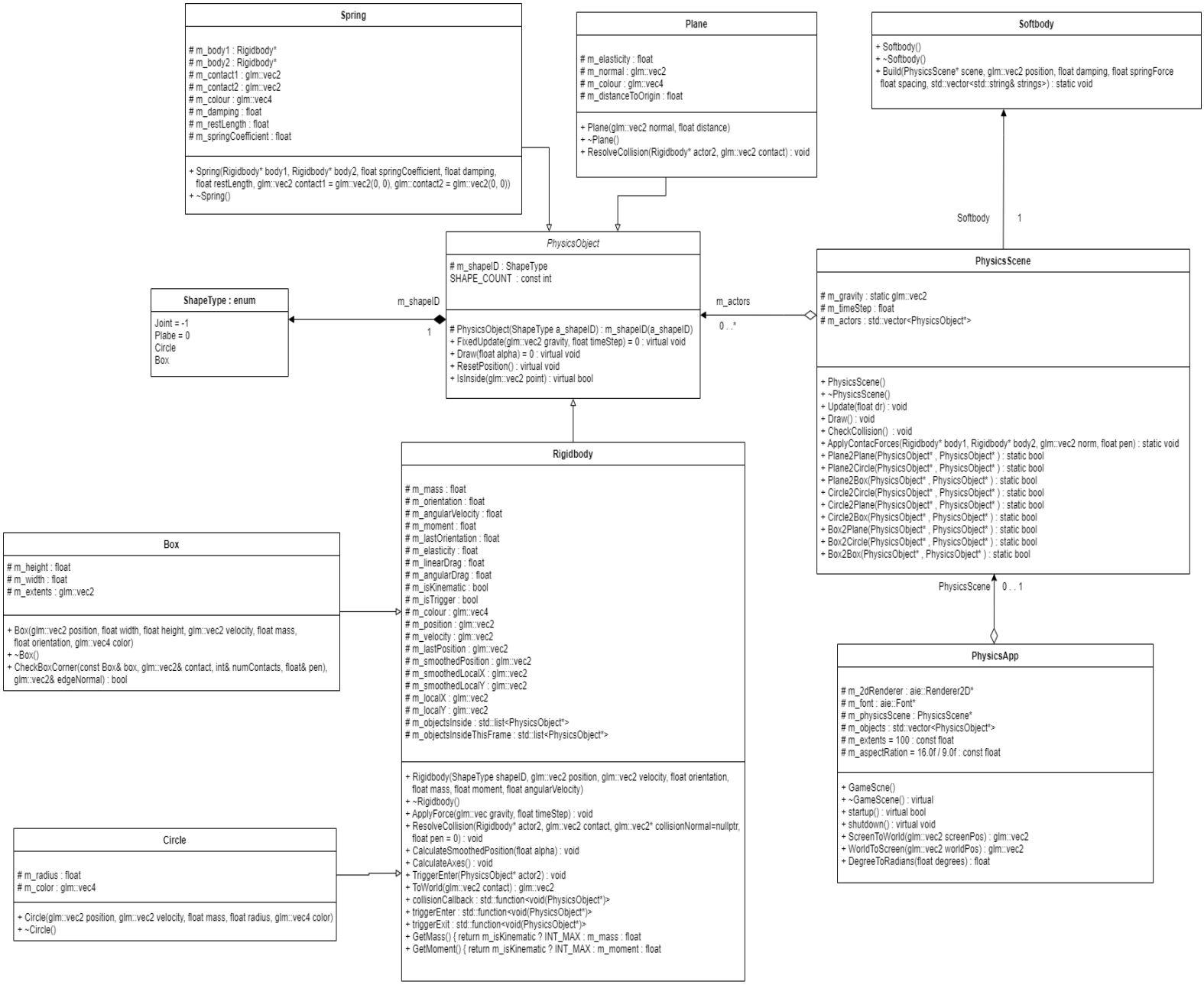
Custom Physics Documentation

8 BALL POOL

JACK AYLWARD

Contents

1.0 - Custom Physics Simulation Class Diagram



2.0 - Custom Physics Simulation Interactions

My custom physics simulation simulates different types of physics objects, such as springs, soft bodies, triggers, dynamic, and kinematic using external factors such as linear and angular drag, gravity, elasticity, and force. These types of objects can take the form of circles and boxes which inherits from a rigidbody class which allows the application of gravity, and planes which directly inherit from the main physics object class.

The collision is detected by using the positions of the objects and the radius (for circle) or the extents and corners (for box) to determine if they are going to overlap. On collision, objects will use an array of functions that find the function that uses those two objects. The correct function is found in the array by using the shapes ID, so when the ID is passed through the array it correctly returns the corresponding function. For example, if a circle and a box collide, it will use a “circle to box” function.

For dynamic object collisions, it will use factors such as elasticity, mass, and velocity to determine the correct force to be applied to the other object. Circle to circle collisions are calculated by checking to see if the distance between the two circles are less than the radius’ of both circles added together. Box to box collisions are calculated a similar way circle to circle is calculated but instead of a radius it uses the box extents and corners to determine if collides. Circle to box and box to circle collision is calculated also like the other collision checks but one of the collision objects will use and radius and the other will use extents and corners to determine if they collide.

For static object collisions, any object set to kinematic will remain static and not move due to object having infinite mass, acting like a wall or obstacle. For trigger collisions, the objects set to trigger will also remain static but will not apply any force to objects that pass through it but would rather be added to a list of objects inside. Plane to plane collisions return false so no math happens as planes are static and pass through each other, so they do not collide. Circle to plane and plane to circle compares the position and radius of the circle to the normal of the plane to determine possible collision. Box to plane and plane to box collision used the extents and corners of the box and the normal of the plane to calculate collision.

Soft bodies and springs utilise a rope like object that connects a series of boxes or circles, creating its own force when objects move further apart while connected. Planes are like a kinematic object, but object will not be able to be behind the plane as they will be forced outside of the plane, simulating ground.

3.0 - Custom Physics Simulation Potential Improvements

3.1 - Improvement #1

One improvement I could make to the physics simulation would be adding an angular motion and torque factor into the physics objects to get more accurate results in force application for the objects. This would be a good improvement as balls in real life have spin which heavily effect the direction and velocity it will travel when it lands. This will also allow to accurately simulate more scenarios such as a ball on a ramp. The ball going down a ramp should speed up due to the force applied from the torque which would exponentially increase the angular velocity of the ball as long as the ramp stays at a certain angle and gravity stays constant.

Variables such as torque and angular velocity could be used in the rigid body class so not just circles could use this and only objects that can be affected by force can use it. Adding surface friction is also a key factor into making angular velocity and torque work efficiently. The friction would be a variable applied on any physics object.

The value of the friction will heavily impact the torque and angular velocity of the physics object that collides. Friction will also help slow down objects that are moving on top of another object, such as player walking on ground and suddenly stopping.

For example, in 8 ball pool you could add spin to the cue ball as you can in real life. When the ball with spin collides with other objects, it will bounce not just in the reflected angle, but with curvature added to it. Additionally, you could create a working car.

3.2 - Improvement #2

Another improvement I could make to the physics simulation would be adding more additional shapes as currently it quite restricted on what you can use. Currently only boxes, circles, and planes are available in the simulation. I would add a polygon class which allows you to create chapes with a custom number of vertices in the shape, ranging from 3 to make a triangle, to enough vertices to essentially create a circle. This would allow shapes such as pentagons, hexagons, octagons, and decagons. The addition of these custom shapes opens a wide door for creativity and allows you to create unique physics simulations and seeing how they react to colliding with all other kinds of shapes.

This can be done by looping through the number of vertices to create a symmetrical shape with the desired number of vertices as each shape can just be multiple triangles together to simulate different shapes.

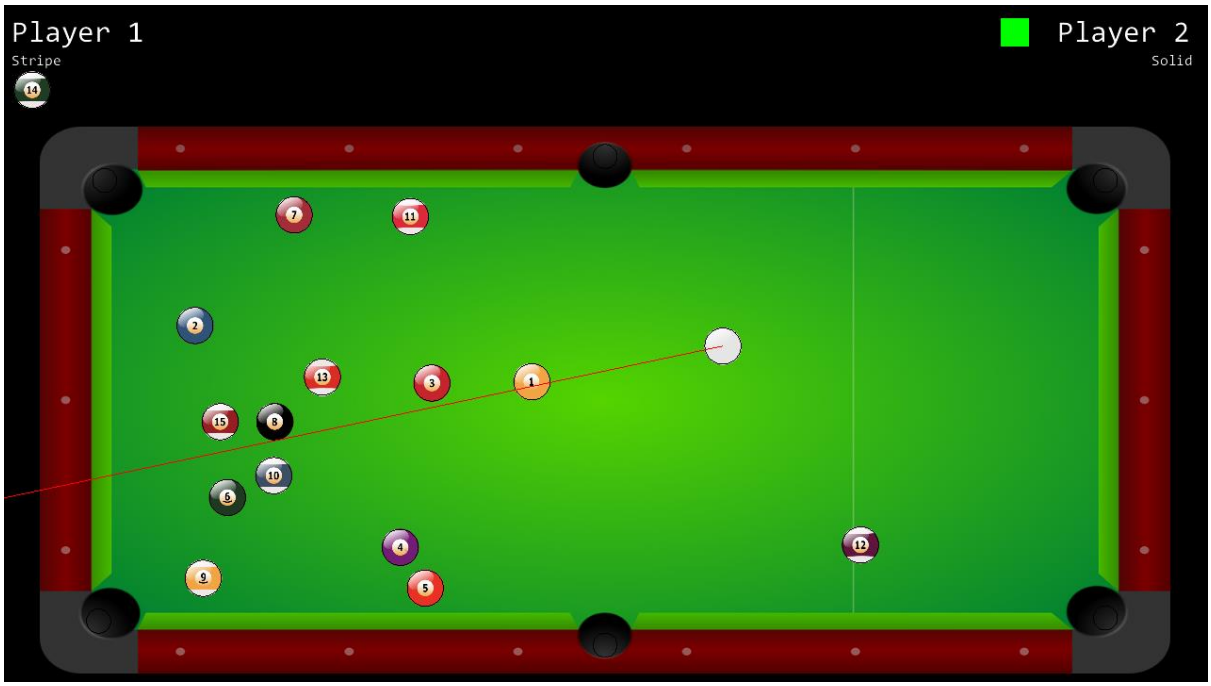
A good use for custom vertices could be to create a huge variety of obstacles in Pachinko as there are tons of different shapes you can use, making pachinko harder and utilize different types of collision. As more shapes are available you are also able create more advanced simulation has many different types of collision will be used.

New collision functions would have to be added to support all the new shapes created which might result in needing to remake the way the collision functions are selected as it might become a too messy if there are over 30 different collision functions.

4.0 - Visualised Game Using Your Custom Physics Simulation

I have chosen to visualize the application of my physics engine in the form of 8 ball pool. It utilises all different types of physics objects and shapes to simulate the real game. The pool balls are kinematic that all use a “pool ball” class inherited from the circle class, with the only difference is adding a boolean which tells the system if it has been sunk. The pockets are all kinematic triggers that check if the ball was a stripe, solid, black ball, or cue ball. The edges of the box are kinematic boxes that keep the cue ball within the table.

Each player alternate turns with a green indicator to tell you whose turn it is. Shooting the cue ball involves holding left click and moving the mouse to change directions depending on the angle the mouse is from the cue ball, and the power of the shot depending on the distance from the mouse to the ball which is visualised by a colour changing line. Triggers are place on each pocket so on trigger enter will run to check if what kind of ball triggered the pocket. When a stripe or solid is sunk, it is pushed back to its corresponding vector with stripes being pushed back to “sunk stripes”, and solids being pushed back into “sunk solids”. Then each ball in these vectors is set to kinematic and sent to the top of the screen under the corresponding player name to track progress. When the first ball of the game is sunk (other than black or white ball), depending on whose go it was, the player will be assigned the ball type they sunk in and the other player is assigned the other type. If the player sinks one of their corresponding balls, they will be granted an extra shot. When any of the vectors reach a size of 7, meaning the player has sunk all their corresponding balls, a boolean will be set to true meaning they can now sink the black ball and win. If the black ball is sunk prematurely, the other player will be granted the win. If the white ball is sunk, the turns will immediately switch, and the other player will be able to place the white ball anywhere on or before the line as well as being granted an extra shot.



5.0 - Third Party Libraries

No third-party libraries were used as the simulation only required what had been created in the bootstrap.

6.0 - References

The UML 2 class diagram (no date) *IBM developer*. IBM. Available at: https://developer.ibm.com/articles/the-class-diagram/ (Accessed: February 20, 2023).

Newton's laws of Motion (no date) *Encyclopedia Britannica*. Encyclopedia Britannica, inc. Available at: https://www.britannica.com/science/Newtons-laws-of-motion (Accessed: February 20, 2023).

Frictional forces: Static and Kinetic (2017) *YouTube*. Professor Dave Explains. Available at: https://www.youtube.com/watch?v=3EbUa5ZDybg&t=9s (Accessed: February 20, 2023).

Alrasheed, S. (1970) *Impulse, momentum, and collisions*, *SpringerLink*. Springer International Publishing. Available at: https://link.springer.com/chapter/10.1007/978-3-030-15195-9_5#Sec4 (Accessed: February 20, 2023).

Ch. 11 introduction - university physics volume 1 (no date) *OpenStax*. Available at: https://openstax.org/books/university-physics-volume-1/pages/11-introduction (Accessed: February 20, 2023).