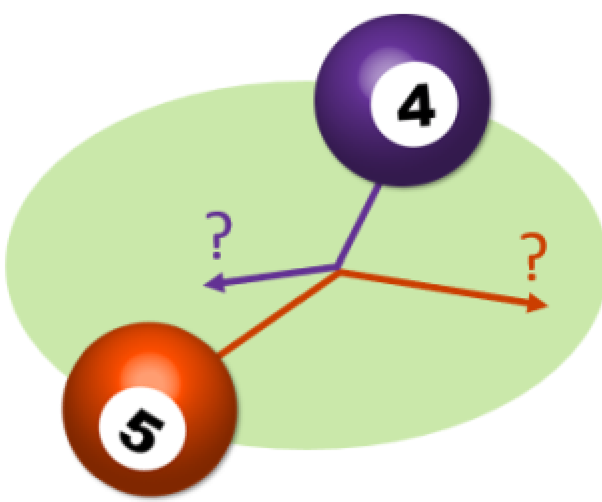
Elastic Collision in a Pool Game

Posted on February 7, 2023 | Posted in Computer Science, Computing Concepts, JavaScript, Solved Challenges

A range of video games use **elastic collision** formulas to predict the **change of velocity** of two objects when a collision occurs.



Elastic collision occurs when two objects are colliding and the total kinetic energy of the two objects remains the same. In reality, most collisions between two objects would result in some loss of energy (inelastic collision). However in some contexts, such as when two rigid billiard balls are colliding, this loss of energy is negligeable. In a video game of pool we

Discuss this challenge on reddit r/101Computing_Hub

Recent Posts

- Elastic Collision in a Pool Game
- The Monty Hall Problem
- Denary to Binary Conversion Algorithm
- Software Crosswords
- Short Path Algorithm Practice
- TCP/IP Stack: Network Layers and Protocols
- Snow Poem Algorithm

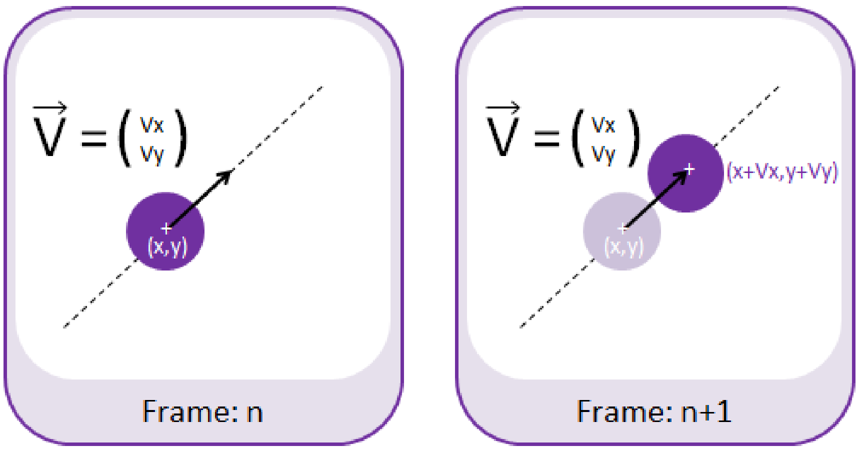
Weekly Challenge Ideas...

would therefore implement an algorithm based on an elastic collision.

Let's investigate how we can implement an elastic collision in a computer animation or a video game.

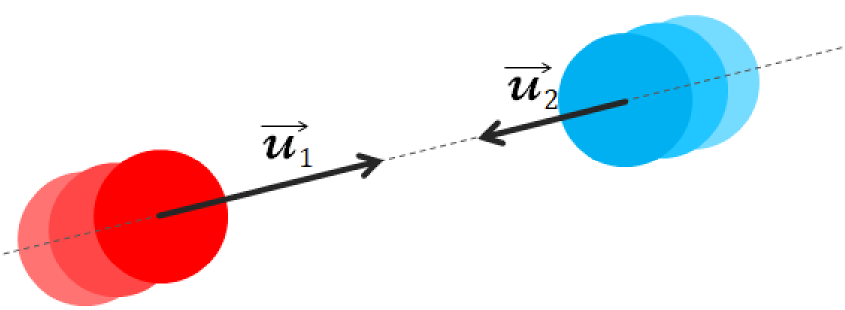
Velocity Vector

In a frame based game, most moving sprites would have a set of coordinates *(x,y)* to inidicate their position on the screen as well as a vaolcity vector (vx,vy) to indicate used to increment the *(x,y)* coordinates between two frames of the game and hence implement the movement of the sprite.



1 Dimensional Elastic Collision

Let's consider two perfectly elastic balls of masses *m1* and *m2* moving along the same straight line with velocities *u1* and *u2*.

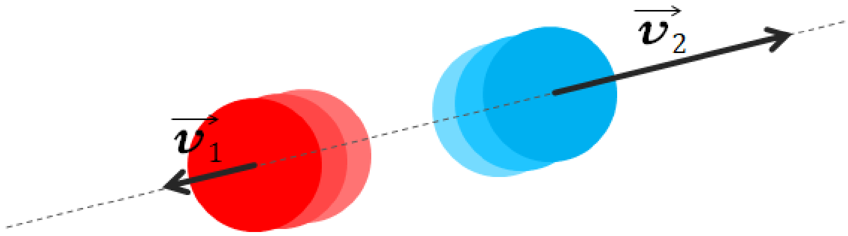


Our aim is to calculate the velocity *v1* and *v2* of these two balls after the collision.

- Programming Terminology – Drag and Drop
- Laser Maze Game in Python
- Battle of the Knights

- View more recent posts...
- View all our challenges...
- Take a Quiz...

LMC Simulator
Flowchart Studio
Python Syntax



Conisidering that both momentum and kinetic energy are conserved quantities in an elastic collision, we can deduct the following two formulas:

Conservation of momentum

$$m_1u_1 + m_2u_2 = m_1v_1 + m_2v_2$$

Conservation of kinetic energy

$$\frac{1}{2}m_1u_1^2 + \frac{1}{2}m_2u_2^2 = \frac{1}{2}m_1v_1^2 + \frac{1}{2}m_2v_2^2$$

We can use both these formulas to calculate the velocity vectors *v1* and *v2* of the two colliding objects after the collision:

$$v_1 = \frac{m_1 - m_2}{m_1 + m_2}u_1 + \frac{2m_2}{m_1 + m_2}u_2$$

$$v_2 = \frac{2m_1}{m_1 + m_2}u_1 + \frac{m_2 - m_1}{m_1 + m_2}u_2$$

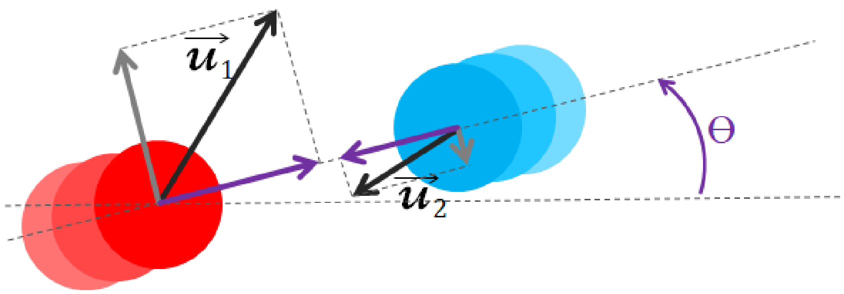
Note that these equations can be simplified when both colliding objects have the same mass: *m1=m2*. In this case we can use the following simplified formulas:

$$v_1 = u_2$$

$$v_2 = u_1$$

2 Dimensional Elastic Collision

In a 2-dimension environment, the velocity vectors may not be aligned on the same straight line.

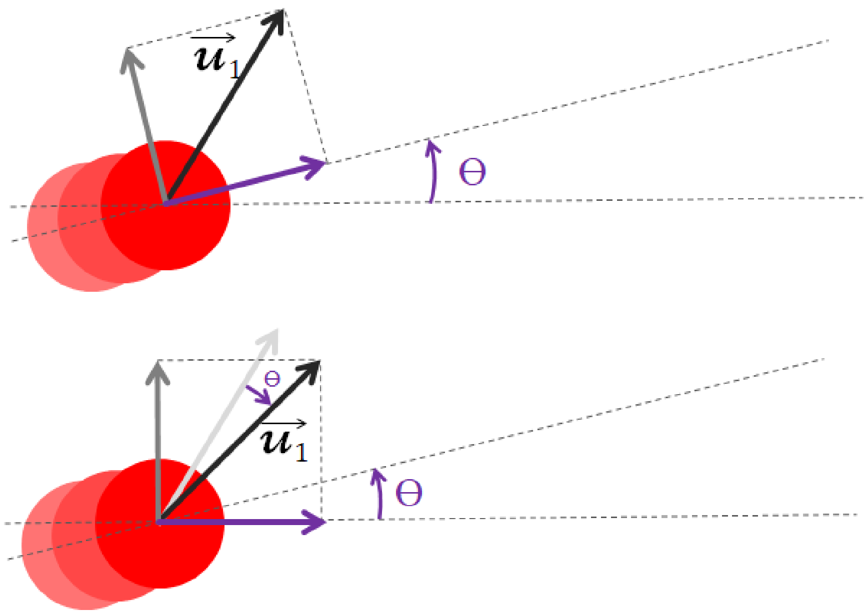


We can however decompose the velocity vectors to identify the component of the velocity that is going along the straight line joining the centre of both moving balls (Purple vectors on the above diagram) and the component that is perpendicular to this straight line (Grey vectors on the above diagram). The first component will be affected by the elastic collision using the 1-Dimensional model/formulas whereas the second component will not be affected by the collision.

In order to calculate the 2 components of our velocity vectors (purple and grey vectors) we will first need to perform a rotation by θ, the angle that can be calculated using the *(x,y)* cartesian coordinates of the centre of both moving objects as follows:

$$\theta = \tan^{-1}\left(\frac{y_2 - y_1}{x_2 - x_1}\right)$$

We will then rotate our velocity vectors by *θ* as shown on the diagram below. Note however, that we would apply the same rotation for both velocity vectors (red and blue balls).



The formulas to perform a 2D rotation are as follows: (You can find out more about these formulas on this page)

$$V_x = v_x . \cos(\theta) \ - \ v_y . \sin(\theta)$$

$$V_y = v_x . \sin(\theta) \ + \ v_y . \cos(\theta)$$

where (V_x,V_y) represents the velocity vector (v_x,v_y) after the rotation.

We can now apply the 1-dimensional elastic collision formulas to the V_x (purple component) of the velocity for each moving object, whereas the V_y components will not be affected by the elastic collision.

And then we will need to rotate our new velocity vectors by $-\theta$ to cancel out the previous rotation.

2-Dimensional Elastic Collision Demonstration

To see how the above formulas can be implemented, we have created a demo using JavaScript. You can investigate this code further to identify how the steps described above have been implemented.

In our demo, all the moving objects have a different mass, pro-rata of their size (radius).

Note that this code could be simplified further using objects of the same mass as this would be the case in a game of pool!

Finally, you will notice that this code also applies some formulas to the velocity vectors of each ball to let the balls bounce against the edge of the canvas. These formulas are explained on this page.

HTML CSS JS Result EDIT

```html
<canvas id="canvas"></canvas>
```

Resources

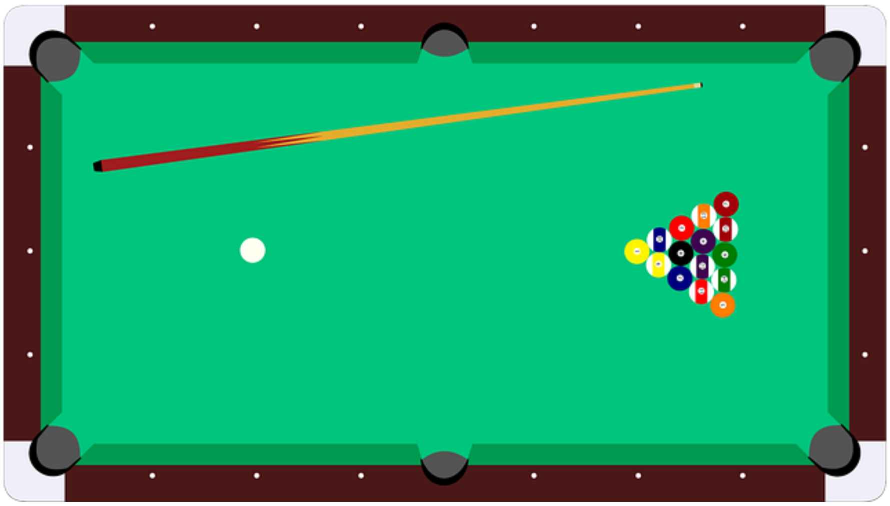
Your Task

Your task consists of tweaking the above code (Click on “Edit On Codepen” button in the top right corner of the above codepen frame) in order to create a pool table with:

- 7 yellow balls
- 7 red balls
- 1 black ball
- 1 white ball

Note that, in a game of pool, all balls have the same mass and size. You can hence simplify the code to use the simplified 1-dimensional elastic collision formulas.

About this task: The aim of this challenge is not to create a full game of pool but just to tweak the above code to make sure that the canvas contains 16 balls of the right colour and of the same weight/size. In a full game of pool, other features would need to be considered such as, implementing the correct size of the pool table, adding pockets and detecting when balls fall into these pockets, adding friction to slow down the rolling balls and adding a mechanism for the player to aim/shoot. All of these features are not part of this task.



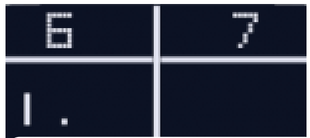
Solution...

The solution for this challenge is available to full members!
Find out how to become a member:

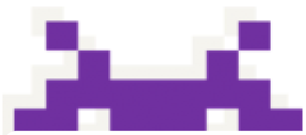
➤ Members' Area

Other challenges you may enjoy...

Breakout Tutorial using Pygame: Adding a Bouncing Ball



Pong Tutorial using Pygame - Adding a Bouncing Ball



Computational Features of Video Games

Bouncing Algorithm in a platform game

‹ The Monty Hall Problem

Home | Computing Concepts | Python Challenges | BBC micro:bit | HTML, CSS & JavaScript | Books | Privacy Policy | Contact | Follow us…