Lecture 6

Recap

- HTML Canvas
  - Comparison to SVG browser graphics
  - Drawing shapes
  - Basic Trigonometry
  - Basic collision detection
  - Animation and User Interaction
    - Examples

Today

- More advanced collision detection
- Transformations

Lab 3 Solution

- Bouncing Ball

Object Oriented JavaScript

```
// ball object
function Ball(x, y, r, xVel, yVel, mass) {
    this.x = x;
    this.y = y;
    this.r = r;
    this.xVel = xVel;
    this.yVel = yVel;
    this.mass = mass;

    this.draw = function () {
        ctx.beginPath();
        ctx.arc (this.x, this.y, this.r, 0, Math.PI * 2, false);
        ctx.stroke ();
    }

    this.resize = function (radius) {
        this.r = radius;
    }

    this.move = function (xpos,ypos) {
        this.x += xpos;
        this.y += ypos;
    }
}

var b1 = new Ball(100, 100, 30, 3, 6, 10);
```

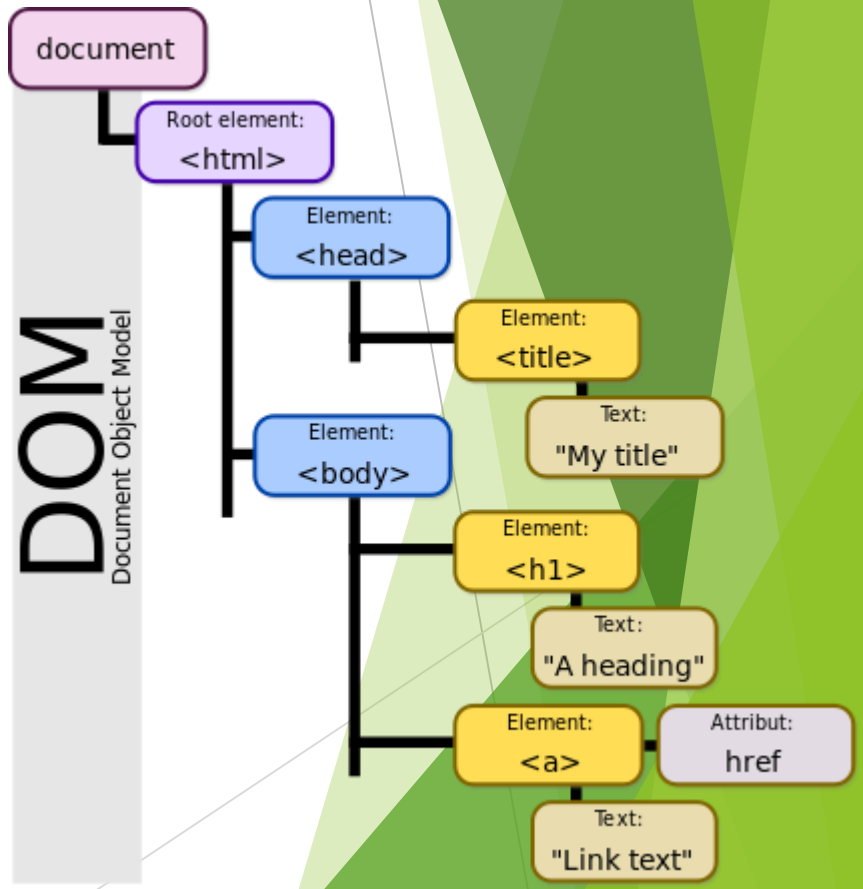
```
var ball = {
    x: 150,
    y: 150,
    r: 50,

    draw: function () {
        ctx.beginPath();
        ctx.arc(this.x, this.y, this.r, 0, Math.PI * 2, false);
        ctx.stroke();
    },

    move: function (xpos, ypos) {
        this.x += xpos;
        this.y += ypos;
    },

    resize: function (radius) {
        this.r = radius;
    }
};
```

Animation and Wall collisions

- Look at code

jQuery

- Cross-platform JavaScript library designed to simplify client-side scripting of HTML
- Most popular JavaScript library in use today, with installation on 65% of the top 10 million highest-trafficked sites on the Web
- Free and open-source software
- Syntax makes it easier to navigate a document, select DOM elements, handle events, and develop Ajax applications.



Mouse events

Event	Description
click	User presses and releases a button over the same element
dblclick	User presses and releases a button twice over the same element
mousedown	User presses a mouse button while over an element
mouseup	User releases a mouse button while over an element
mousemove	User moves the moue
mouseover	User moves the mouse over an element
mouseout	User moves the mouse off an element

- Occur when a user interacts with a mouse.
- More complicated than keyboard events.
  - Must be aware of the position of the Canvas element within the browser window as well as the position of the user's cursor.
- Listening for mouse events
  - Easy to get the position of the mouse relative to the entire browser window using the e.clientX and e.clientY properties. In this case, the origin of (0,0) would be located at the top left of the entire browser window.
  - Would be better to consider the origin (0,0) to be located at the top left of the Canvas element. Ideally, you want to be working within the local coordinate system that's relative to the Canvas area rather than a global coordinate system that's relative to the entire browser window.

Mouse event strategies

- Use the following steps to transform global window coordinates to local Canvas coordinates.
  - Calculate the (x,y) position of the Canvas DOM element on the page – Use (e.clientX, e.clientY) or (e.pageX, e.pageY)
  - Determine the global position of the mouse in relation to the entire document.
  - To locate the origin (0,0) at the top left of the Canvas element, and effectively transform the global coordinates to relative coordinates, take the difference between the global mouse position calculated in step 2 and the Canvas position calculated in step 1.

Mouse event strategies

- Translating Global coordinates to Local coordinates

```
var canvas = $('#my_canvas');

// calculate position of the canvas DOM element on the page

var canvasPosition = {
    x: canvas.offset().left,
    y: canvas.offset().top
};

canvas.on('click', function(e) {

    // use pageX and pageY to get the mouse position
    // relative to the browser window

    var mouse = {
        x: e.pageX - canvasPosition.x,
        y: e.pageY - canvasPosition.y
    }

    // now you have local coordinates,
    // which consider a (0,0) origin at the
    // top-left of canvas element
});
```

Lab 4 Mouse events solution

```
var canvas = document.getElementById("canvas-for-ball");
var canvasPosition = {
    x: canvas.offsetLeft,
    y: canvas.offsetTop
}
```

```
canvas.addEventListener("click", function (event) {
    var mouse = {
        x: event.clientX - canvasPosition.x,
        y: event.clientY - canvasPosition.y
    }
    ctx.arc(mouse.x, mouse.y, 10, 0, Math.PI * 2, false);
});
```

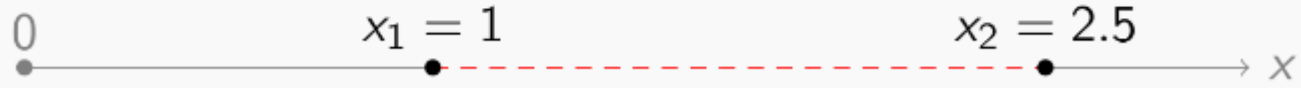
How to detect collisions between circles?

- Step 1: Determine distance between both circle's centre points
  - How to determine distance between two points?
    - Length of a line equation
  - In one dimension, the length of a line segment between two points x2 and x1 is just x2-x1. Obviously x2 needs to be bigger than x1.
    - If not, one trick is to just square the distance and take the square root: $\sqrt{(x_2 - x_1)^2}$
  - For example, the distance between 2.5 and 1 is:

How to detect collisions between circles?

- Step 1: Determine distance between both circle's centre points
  - How to determine distance between two points?
    - Length of a line equation
  - In one dimension, the length of a line segment between two points x2 and x1 is just x2-x1. Obviously x2 needs to be bigger than x1.
    - If not, one trick is to just square the distance and take the square root: $\sqrt{(x_2 - x_1)^2}$
  - For example, the distance between 2.5 and 1 is:

$$\sqrt{(2.5 - 1)^2} = 1.5$$

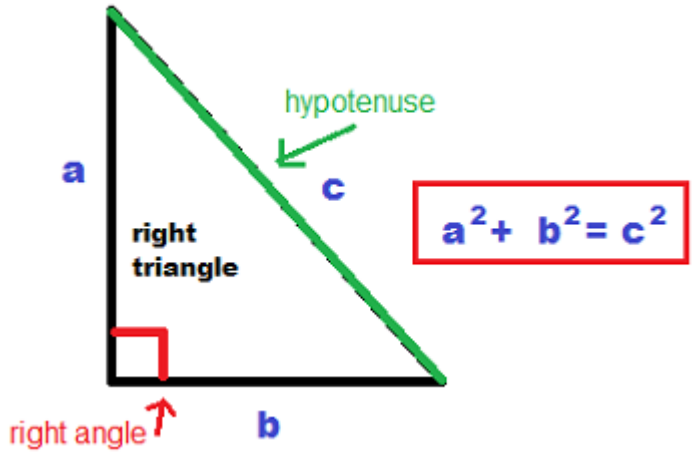


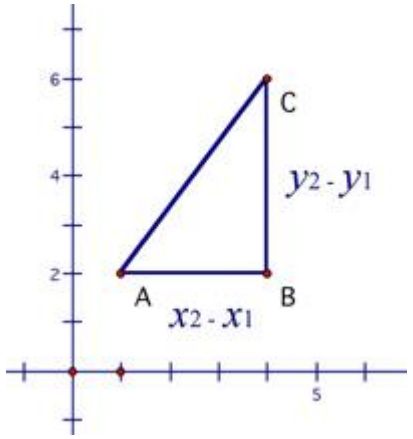
Distance in 2D

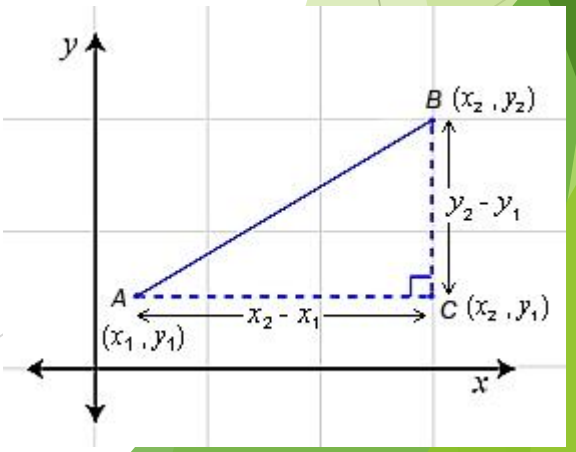
- In two dimensions, the distance between two points (x1,y1) and (x2,y2) can be calculated using the following formula:

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

- Why does this work? We can see using Pythagoras's theorem:





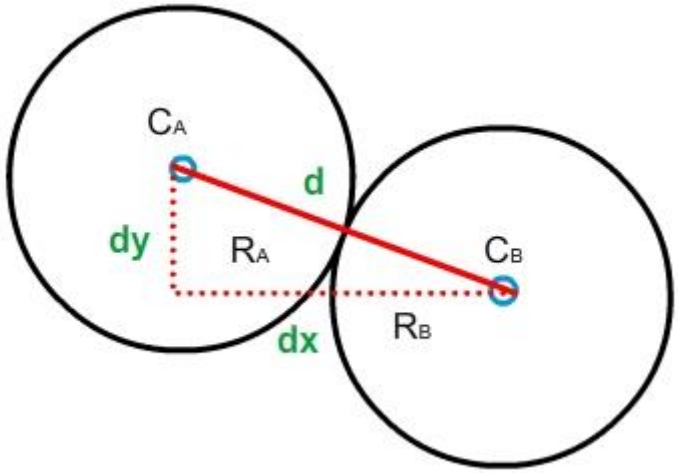


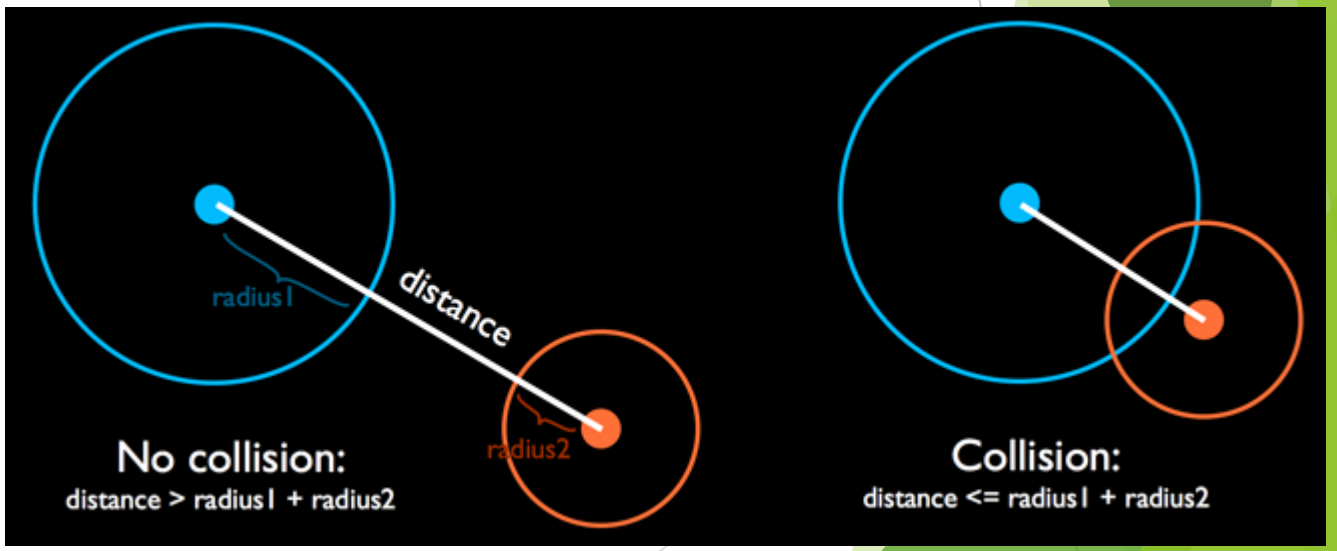
How to detect collisions between circles?

- Now we have the distance between both centres.
- What now?

How to detect collisions between circles?

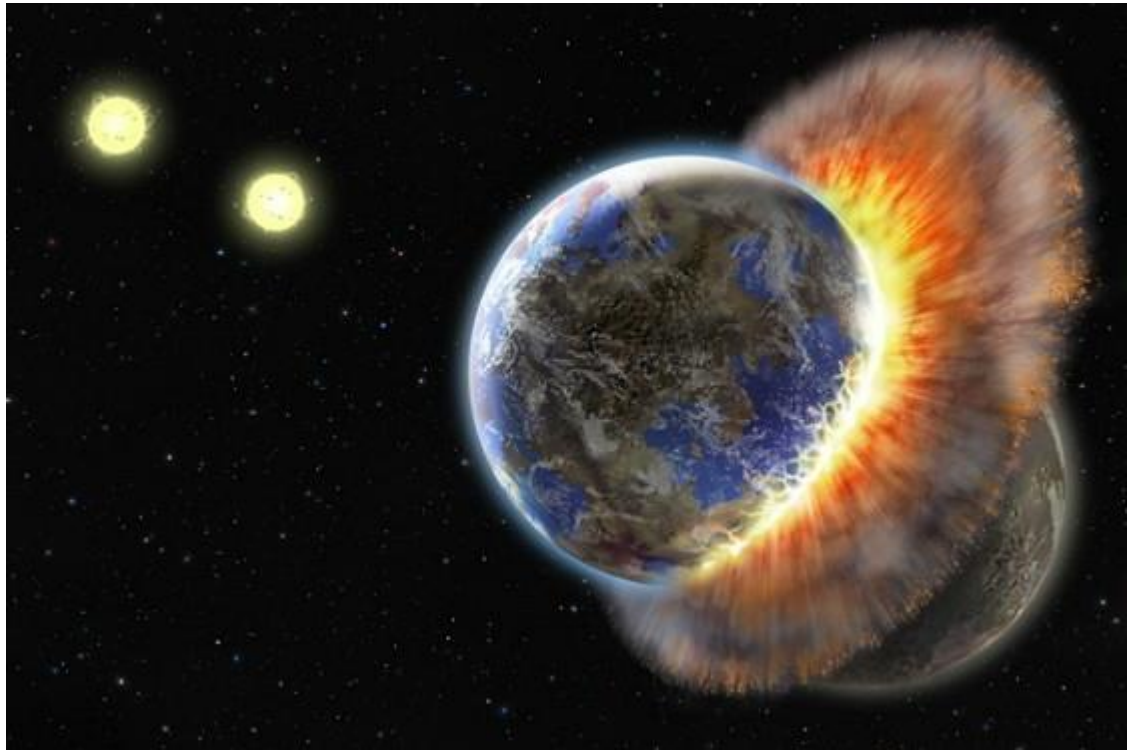
- Now we have the distance between both centres.
- What now?
  - If the distance is less than the sum of the radii, then, the circles are intersecting





What to do if balls have collided?

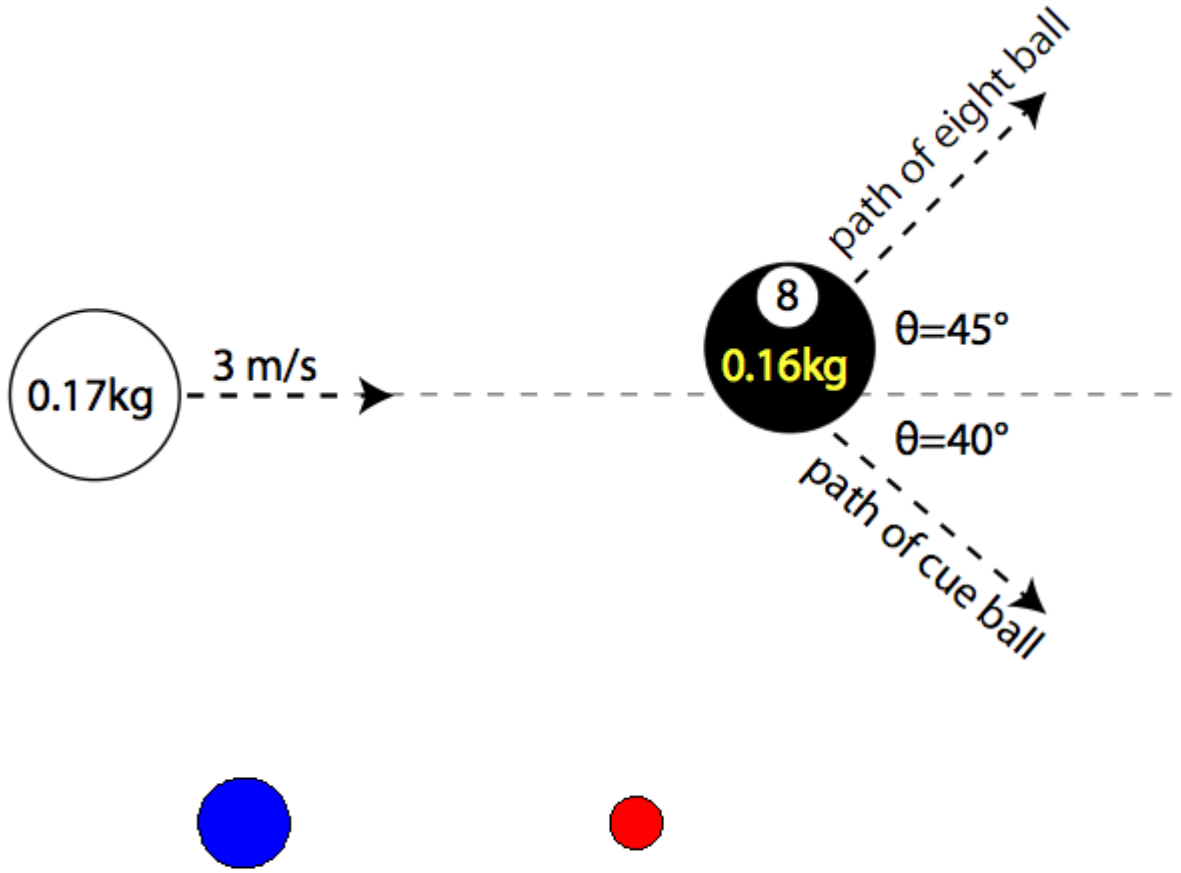
What to do if balls have collided?



What to do if balls have collided?

- Not quite:
- More like:

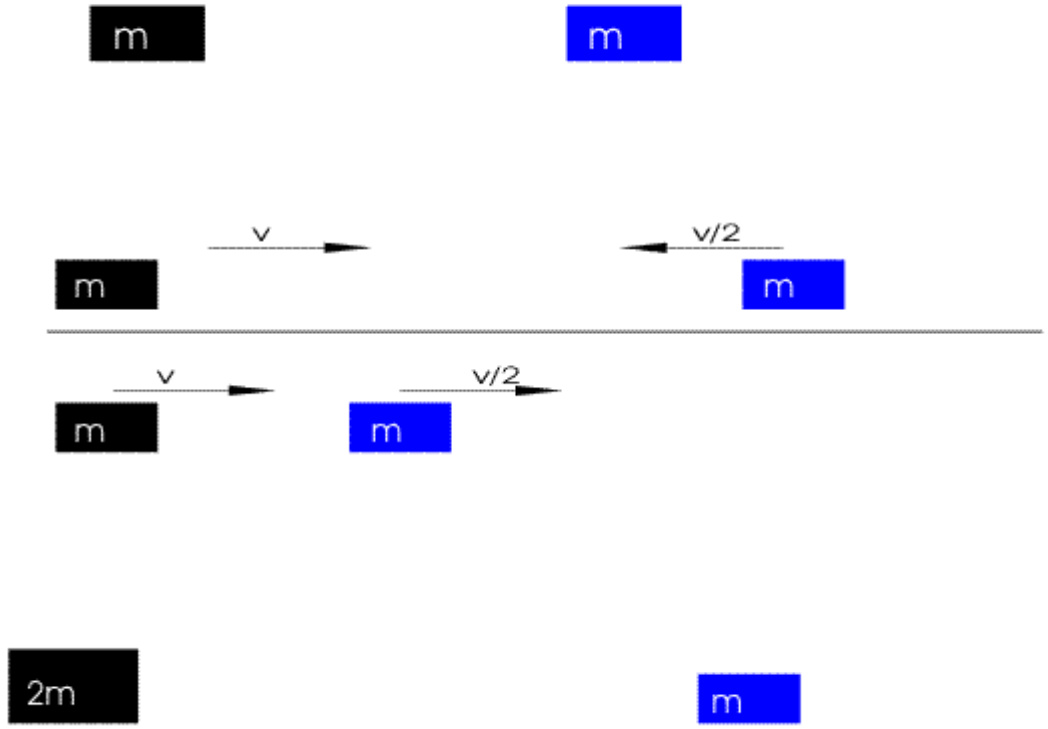
Elastic collision



Elastic and Inelastic collisions

- Elastic collision definition
  - An encounter between two bodies in which the total kinetic energy of the two bodies after the encounter is equal to their total kinetic energy before the encounter. Elastic collisions occur only if there is no net conversion of kinetic energy into other forms (such as heat or noise).
- Inelastic collision definition
  - An inelastic collision, in contrast to an elastic collision, is a collision in which kinetic energy is not conserved due to the action of internal friction. In collisions of macroscopic bodies, all kinetic energy is turned into vibrational energy of the atoms, causing a heating effect, and the bodies are deformed.
- Partially inelastic collision definition
  - Partially inelastic collisions are the most common form of collisions in the real world. In this type of collision, the objects involved in the collisions do not stick, but some kinetic energy is still lost. Friction, sound and heat are some ways the kinetic energy can be lost through partial inelastic collisions.

Elastic or Inelastic



Elastic collision – 2Balls – 1 dimension

- Collision between 2 objects along 1 axis (1 dimension) is relatively straightforward to calculate
- Defined by the equations:

$$v_1 = \frac{u_1(m_1 - m_2) + 2m_2u_2}{m_1 + m_2}$$

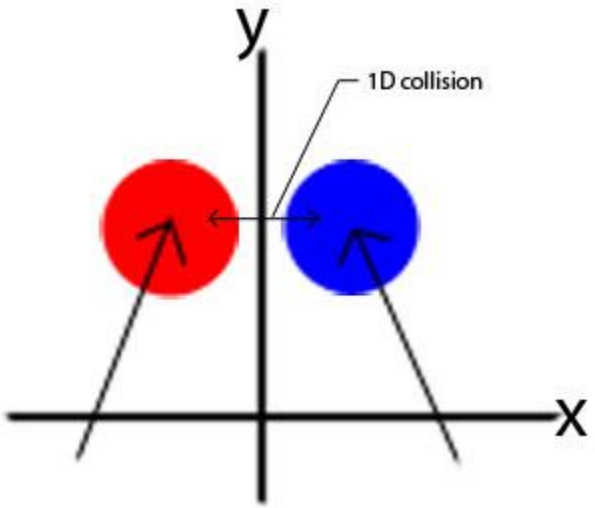
$$v_2 = \frac{u_2(m_2 - m_1) + 2m_1u_1}{m_1 + m_2}$$

  - where:
    m_1 = mass of object 1
    u_1 = initial velocity of object 1
    v_1 = final velocity of object 1
    m_2 = mass of object 2
    u_2 = initial velocity of object 2
    v_2 = final velocity of object 2

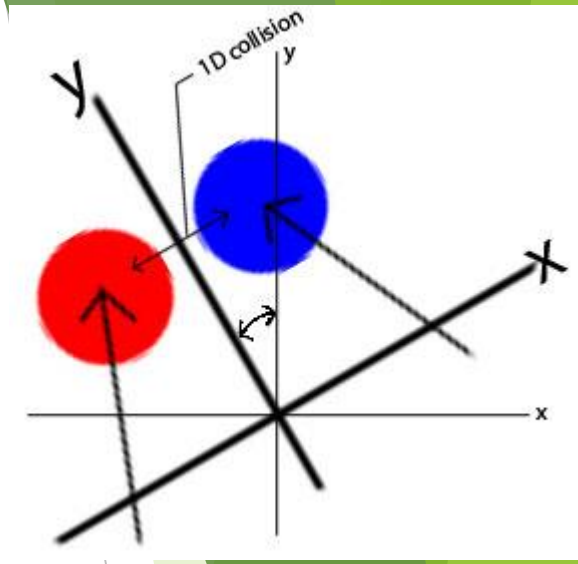
Elastic collision – 2Balls – 2 dimensions

- What happens in 2-dimensions?
- The collision is also quite straightforward when objects collide directly along x or y axes.
- To keep things simple, let's assume the two balls will collide somewhere along the y-axis, exactly parallel to one another.
  - As we can see in the diagram, the collision can still be calculated as a 1D collision since all of the force is being exchanged across the x-axis of the balls (same as with wall collisions from lab 3).
  - This means we can still use the same equation for a 2D collision as we did for a 1D collision:

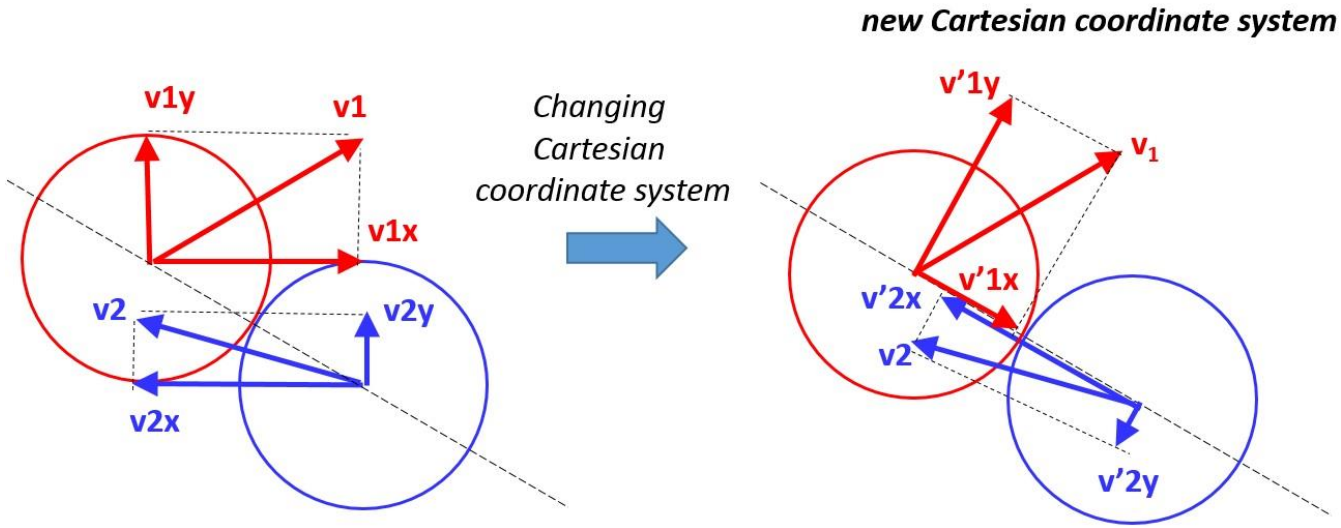
$$v_1 = \frac{u_1(m_1 - m_2) + 2m_2u_2}{m_1 + m_2}$$
$$v_2 = \frac{u_2(m_2 - m_1) + 2m_1u_1}{m_1 + m_2}$$



Elastic collision – 2Balls – 2 dimensions

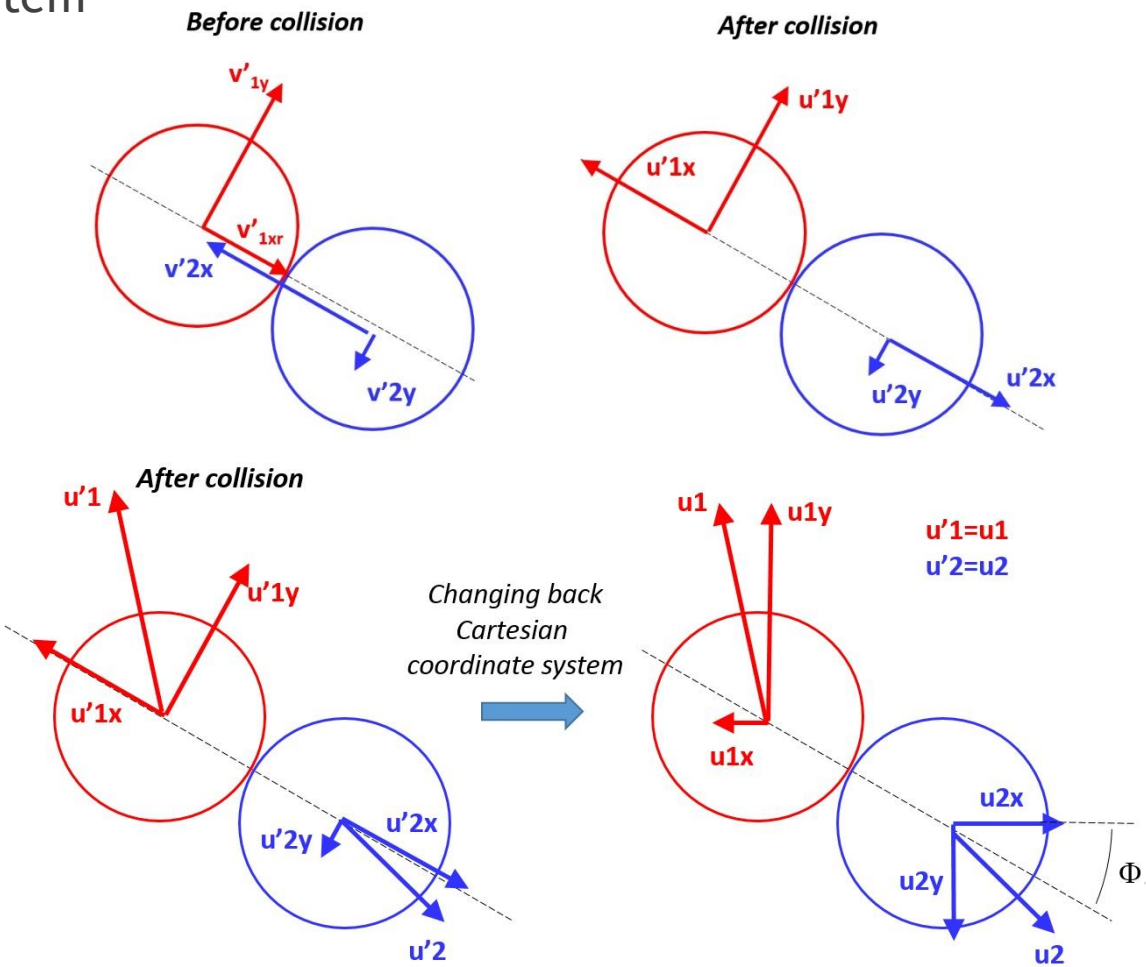


- The only thing left to consider are collisions that occur which are not parallel to the y-axis (which will happen most of the time).
  - The easiest way to solve this problem is to rotate the axes (point-of-view) so the y-axis is once again perpendicular to the collision point (see diagram). After the rotation is complete, we once again have a scenario where the collision can be calculated as if it took place in one dimension.

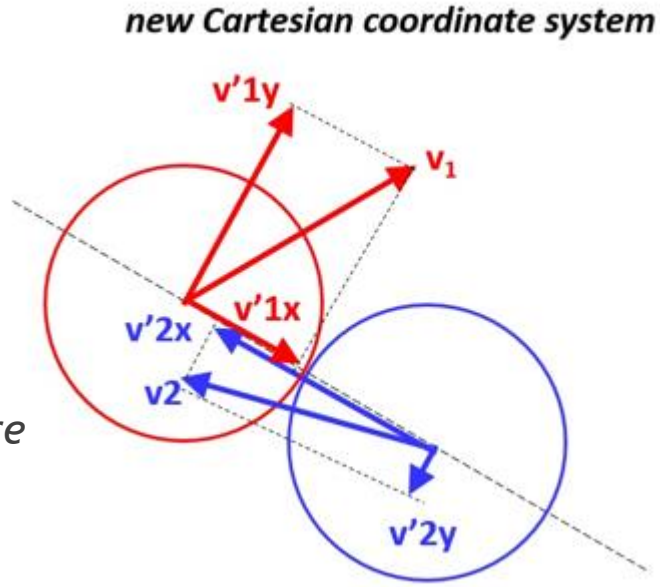


Elastic collision – 2Balls – 2 dimensions

- Once 1d collision has been computed, translate back into original Cartesian coordinate system



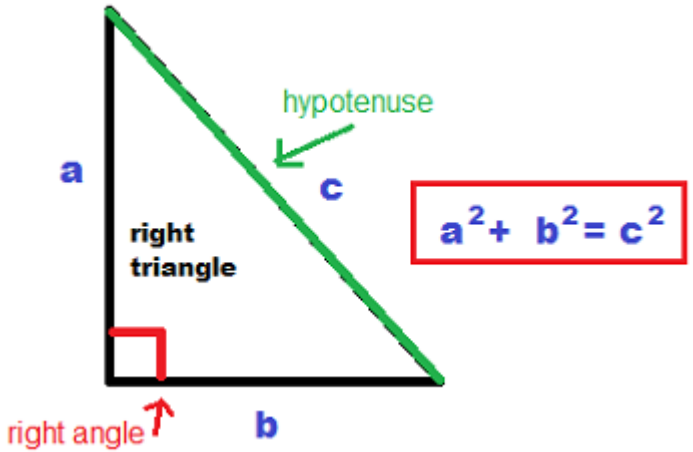
Elastic collision – 2Balls – 2 dimensions

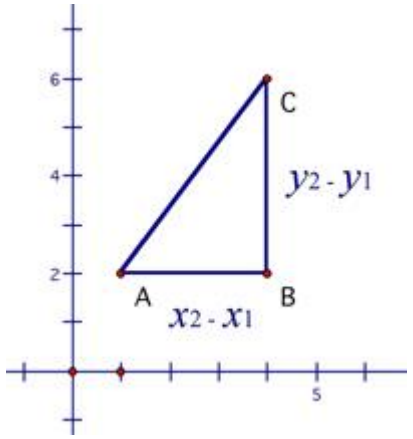


- The Math:
  - *Calculate the collision angle (A) by using atan2 (*$\tan^{-1}\{y,x\}$*) applied to the centre coordinates of each object:*

    A = atan2(y_1 - y_2, x_1 - x_2)
  - Calculate the 2 balls' actual speed (u_1 and u_2) (not speed in separate x and y directions)

    ActualSpeed (u) = $\sqrt{xVel^2 + yVel^2}$





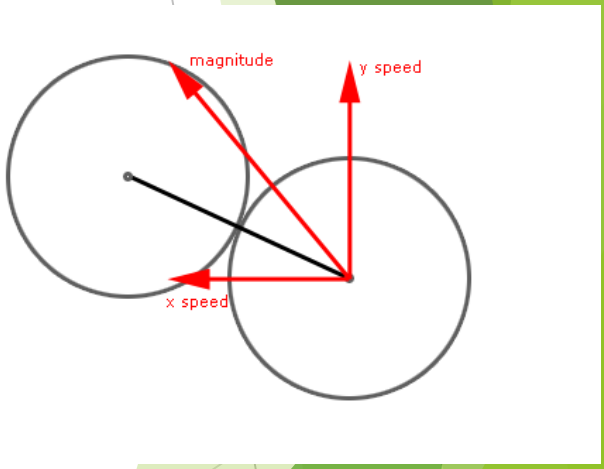
Elastic collision – 2Balls – 2 dimensions

- The Math:
  - Calculate the collision angle (A) by using atan2 applied to the centre coordinates of each object:
    $A = \text{atan2}(y_1 - y_2, x_1 - x_2)$
  - Calculate the 2 ball's actual speed (known as Magnitude - u_1 and u_2) - (not speed in separate x and y directions)

    Magnitude (u) = $\sqrt{xVel^2 + yVel^2}$
  - Determine the 2 balls' directions (D_1 and D_2) using trigonometry

    Direction (D) = $aTan2(yVel, xVel)$
  - Use the collision angle (A), the ball's initial velocity (u) and ball's initial direction (D) to derive it's x/y velocity in the new rotated coordinate system:

    $v_{1x} = u_1 \times \cos(D_1 - A)$
    $v_{1y} = u_1 \times \sin(D_1 - A)$
    $v_{2x} = u_2 \times \cos(D_2 - A)$
    $v_{2y} = u_2 \times \sin(D_2 - A)$



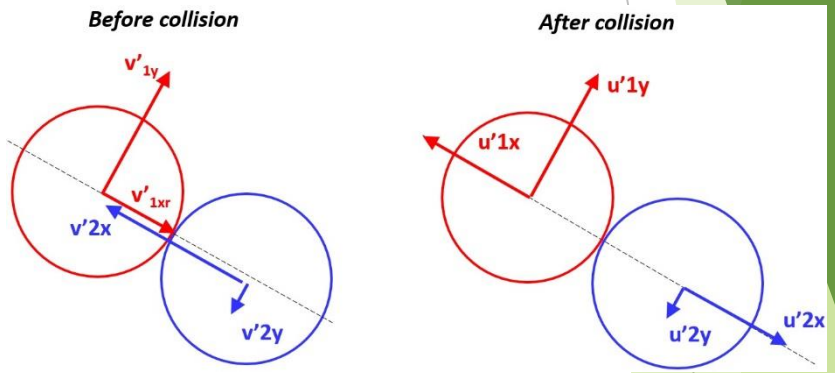
Elastic collision – 2Balls – 2 dimensions

$$v_1 = \frac{u_1(m_1 - m_2) + 2m_2u_2}{m_1 + m_2}$$
$$v_2 = \frac{u_2(m_2 - m_1) + 2m_1u_1}{m_1 + m_2}$$

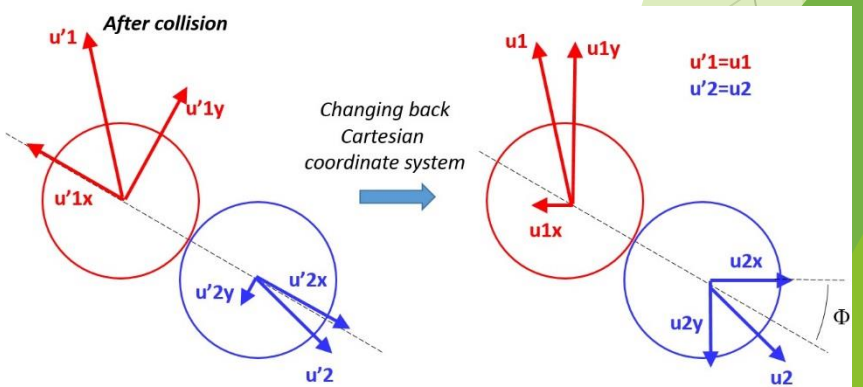
- The Math:
  - Now that we have the collision aligned along the x-axis, all we have to do is apply the 1D collision equation. We will call the final x-velocities for each ball f_{1x} and f_{2x}:

$$f_{1x} = \frac{v_{1x}(m_1 - m_2) + 2m_2v_{2x}}{m_1 + m_2}$$

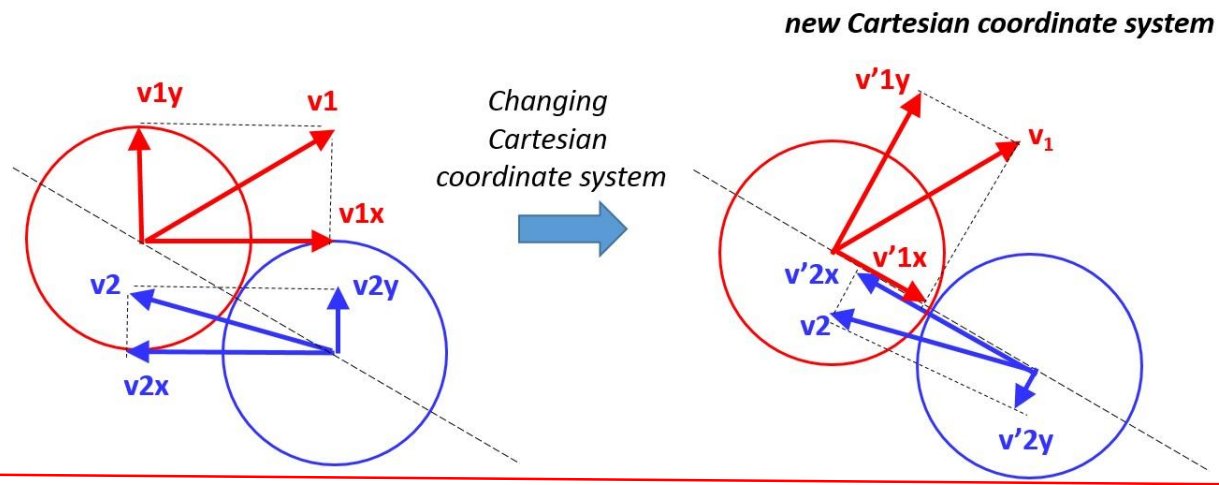
$$f_{2x} = \frac{v_{2x}(m_1 - m_2) + 2m_2v_{1x}}{m_1 + m_2}$$



  - Note y-speed is preserved (does not change)
  - Now that we have the final x and y velocities in the rotated coordinate system, we must convert everything back (for both balls) to a normal Cartesian coordinate system, as follows:
    - $newXVel = (\cos(A) * f_{1x}) + (\cos\left(A + \frac{\pi}{2}\right) * v_{1y})$
    - $newYVel = (\sin(A) * f_{1x}) + (\sin\left(A + \frac{\pi}{2}\right) * v_{1y})$



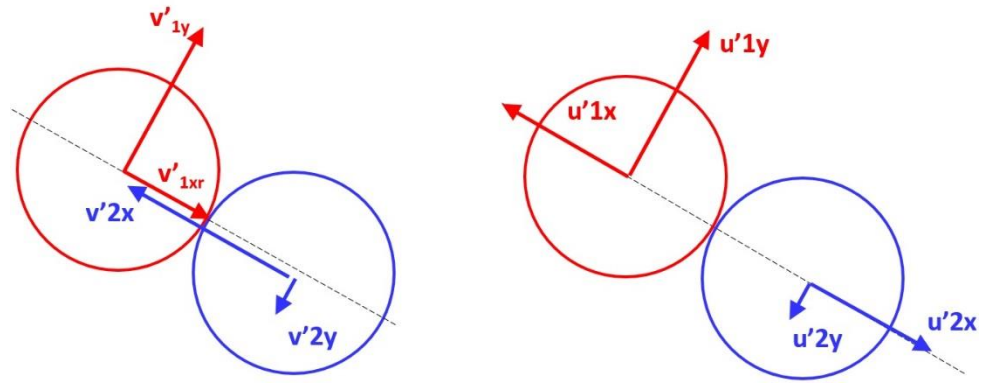
Step 1



Before collision

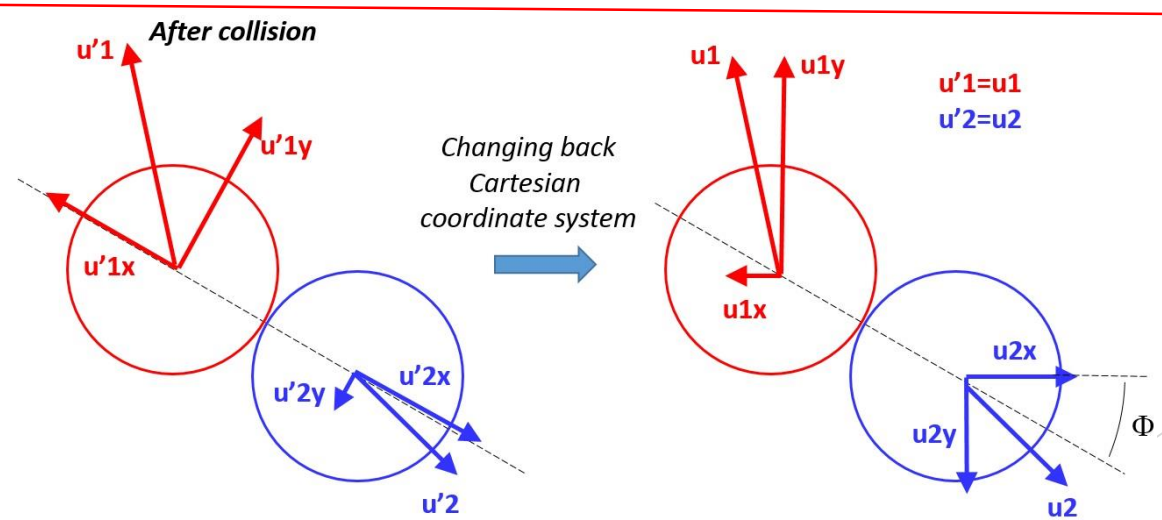
After collision

Step 2



After collision

Step 3



Elastic collision – 2Balls – 2 dimensions

- The good news

Elastic collision – 2Balls – 2 dimensions

```
//Once collision is detected, handle the collision
dx = balls[0].x-balls[1].x;     //Calculate horizontal distance between balls
dy = balls[0].y-balls[1].y;     //Calculate vertical between balls
collision_angle = Math.atan2(dy, dx);   //Calculate the collision angle using trig
//Calculate the ball1 speed, here called the Magnitude
magnitude_1 = Math.sqrt(balls[0].xVel*balls[0].xVel+balls[0].yVel*balls[0].yVel);
// Calculate the ball2 speed in the same way
magnitude_2 = Math.sqrt(balls[1].xVel*balls[1].xVel+balls[1].yVel*balls[1].yVel);
//Determine balls' direction using trigonometry
direction_1 = Math.atan2(balls[0].yVel, balls[0].xVel);
direction_2 = Math.atan2(balls[1].yVel, balls[1].xVel);
//Calculate new xVel using trigonometry applied to the difference between the direction angle and the collision angle
new_xVel_1 = magnitude_1 * Math.cos(direction_1-collision_angle);
//Same thing for other vectors: ball 1 yVel and ball 2 xVel and yVel
new_yVel_1 = magnitude_1 * Math.sin(direction_1-collision_angle);
new_xVel_2 = magnitude_2 * Math.cos(direction_2-collision_angle);
new_yVel_2 = magnitude_2 * Math.sin(direction_2-collision_angle);
//Determine final x speed for ball 1
final_xVel_1 = ((balls[0].mass-balls[1].mass)*new_xVel_1+(balls[1].mass+balls[1].mass)*new_xVel_2)/(balls[0].mass+balls[1].mass);
//Determine final y speed for ball 2
final_xVel_2 = ((balls[0].mass+balls[0].mass)*new_xVel_1+(balls[1].mass-balls[0].mass)*new_xVel_2)/(balls[0].mass+balls[1].mass);
//y speed does not changes (it's a 1D collision)
final_yVel_1 = new_yVel_1;
final_yVel_2 = new_yVel_2;
//Determine x and y speeds on the original axis system using trig. Math.PI/2 is used because the angle between xVel and yVel must always be 90 degrees (pi/2 r
balls[0].xVel = Math.cos(collision_angle)*final_xVel_1+Math.cos(collision_angle+Math.PI/2)*final_yVel_1;
balls[0].yVel = Math.sin(collision_angle)*final_xVel_1+Math.sin(collision_angle+Math.PI/2)*final_yVel_1;
balls[1].xVel = Math.cos(collision_angle)*final_xVel_2+Math.cos(collision_angle+Math.PI/2)*final_yVel_2;
balls[1].yVel = Math.sin(collision_angle)*final_xVel_2+Math.sin(collision_angle+Math.PI/2)*final_yVel_2;
```

Lab

- As usual, create and resize a canvas
- Create two ball objects
  - draw, move, setColour, setMass methods
  - x, y, r, xVel, yVel, colour, mass attributes
- First implement collision detection and reaction between balls and walls
- Next implement inter-ball collision detection
- Once collisions can be successfully detected, implement ball reactions
- Advanced:
  - Experiment with different sphere masses
  - Implement collision system for several balls (scalable code if possible)

Nice demo

- https://scratch.mit.edu/projects/116144988/#player