Lecture 5

Recap

- HTML Canvas
  - Drawing shapes
  - Basic Trigonometry
  - Basic collision detection
  - Animation
    - Example

Today

- HTML5 Canvas compared to SVG
- Look at this weeks lab:
  - User Interaction
  - jQuery
- More advanced collision detection

Recap – HTML Canvas

- A resolution-dependent bitmap canvas, which can be used for rendering graphs, graphics or art on the fly using a JavaScript programmable interface.
  - Resolution-dependent means <canvas> images may lose quality when enlarged or displayed on retina screens.
- The <canvas> tag exposes a surface where you can create and manipulate rasterized images pixel by pixel.
- Drawing simple shapes is just the tip of the iceberg.
  - HTML5 Canvas API allows you to draw arcs, paths, text, gradients, etc.
  - Can also manipulate images pixel by pixel. This means that you can replace one colour with another in certain areas of the graphics
  - Can animate drawings, even draw a video onto the canvas and change its appearance.

SVG

- SVG is a language for describing vector graphics. As a standalone format, it uses the XML syntax [XML10]. When mixed with HTML5, it uses the HTML5 syntax [HTML]. …
- SVG drawings can be interactive and dynamic. Animations can be defined and triggered either declaratively (in SVG content) or via scripting.
- Being scalable has the advantage, contrary to HTML5 Canvas-generated images, of letting you increase or decrease a vector image while maintaining its crispness and high quality.
- You can do with SVG most of the stuff you can do with <canvas>, e.g., drawing shapes and paths, gradients, patterns, animation, etc. However, these two technologies work in fundamentally different ways. As it will become clearer in the following slides, this difference plays a significant part when you need to make a choice between <canvas> and SVG
  - http://svg-whiz.com/svg/linguistics/theCreepyMouth.svg

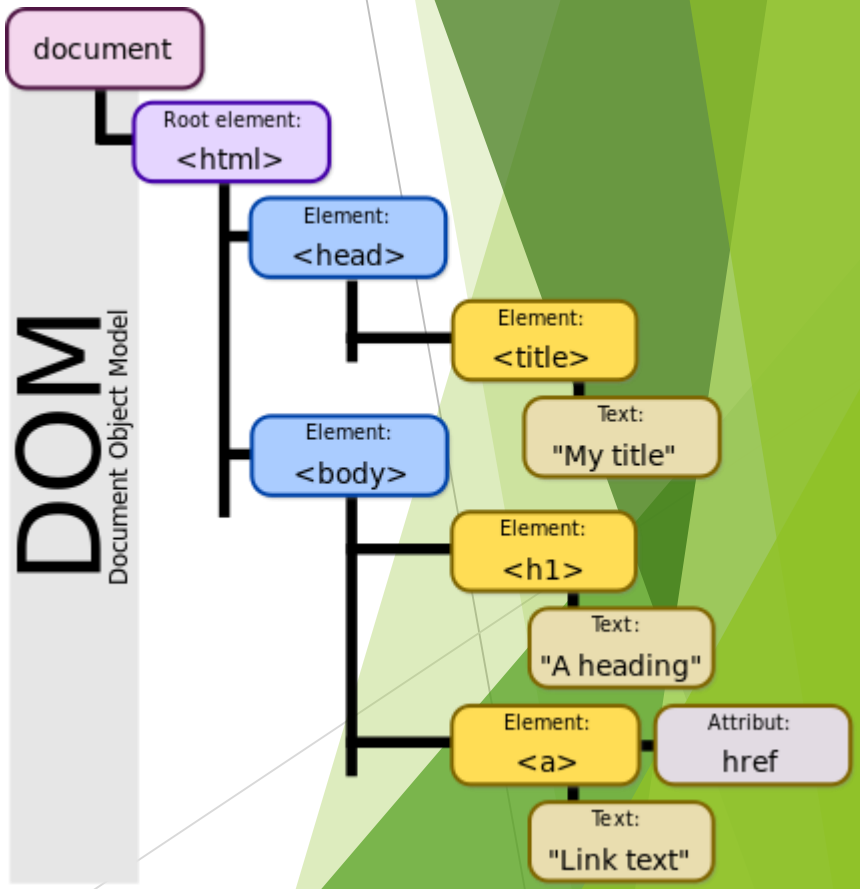
Canvas and SVG
Compare and Contrast

- First distinguish between immediate mode and retained mode graphics
  - Two different approaches of getting content displayed in browser.
- HTML5 Canvas API is an example of immediate mode:
  - Developer works out the commands to draw objects, manually creates and maintains the model or scene, and specifies what needs to be updated. The Canvas Graphics API simply communicates your drawing commands to the browser, which then executes them.
- SVG - retained mode:
  - The browser's Graphics API creates an **in-memory model** or scene of the final output and translates it into drawing commands for your browser.

DOM:

DOM: Document Object Model

- Programming interface for HTML/XML
- For manipulating content, structure and styling
- Tree-like structure
- Elements have attributes



Canvas and SVG
Compare and Contrast

- Being an immediate graphics system, <canvas> **hasn't got a DOM**, i.e., Document Object Model. With <canvas>, you draw your pixels and the system forgets all about them, thereby cutting down on the extra memory needed to maintain an internal model of your drawing.
- With SVG, each object you draw gets added to its internal model, which makes your life as a developer somewhat easier, but at some costs in terms of performance.
- Possible to outline some cases where using one technology over the other is a better option.

HTML5 Canvas applications

- HTML5 Canvas applications:
  - Ray Tracing
  - Drawing a significant number of objects on a small surface
  - Pixel replacement in videos
  - Simple games

SVG applications

- Scalability
  - Where scalability is a requirement SVG will be better suited than <canvas>. High fidelity complex graphics like building and engineering diagrams etc., are examples of this.
  - With SVG, enlarging or printing the images preserves all the details to a high level of quality.
- Database Storage
  - Can generate SVG documents from a database (XML format highly suited)
- Accessibility
  - If accessibility is necessary, using <canvas> is generally not recommended. What is drawn on the canvas surface is just a bunch of pixels, which can't be read or interpreted by assistive technologies or search engine bots. SVG XML is readable by both humans and machines.
- No JavaScript reliance
  - If you don't want to use JavaScript in your application, then <canvas> is not suited.
  - The only way you can work with the <canvas> element is with JavaScript. Conversely, you can draw SVG graphics using a standard vector editing program like Adobe Illustrator or Inkscape, and you can use pure CSS to control its appearance and perform animations.

Canvas vs. SVG conclusion

- No hard and fast rules for when it's best to use <canvas> instead of SVG. However, the distinction between immediate and retained mode provides insight as to which scenarios could benefit from <canvas> over SVG and vice versa.
- From (HTML5 & CSS3 For the Real World, 2nd Edition):
  - If you need to paint pixels to the screen and have no concerns about the ability to retrieve and modify your shapes, canvas is probably the better choice. If, on the other hand, you need to be able to access and change specific aspects of your graphics, SVG might be more appropriate.

User Interaction with Canvas

User Interaction with Canvas

- HTML user input involves using an event-handling system built into browsers from the earliest days of JavaScript-enabled web browsers
  - There's nothing specific to HTML5 for detecting and handling user input.
  - Browser can provide low-level feedback indicating which coordinate (x,y) the user has clicked on, and that's about it.
  - No built-in abstractions to notify when the user has interacted with a specific object that has been rendered on the Canvas.
  - Provides a great degree of low-level control over how you want to handle these events - As long as you can keep various browser quirks at bay

WHAT IS AN EVENT?

- Events are the browser's way of saying, "Hey, this just happened."
- When an event **fires**, your script can then react by running code (e.g. a function).
- By running code when an event fires, your Canvas responds to the user's actions.
  - It becomes **interactive**.

Types of events

- User interaction is handled entirely by the browser's traditional event listener model.
  - There is nothing new with the advent of HTML5; it's the same event model that has been used since the early days of Netscape Navigator.
- An interactive application or game is a marriage between the browser event model for user input and Canvas for graphical output.
  - No logical connection between the two unless you build it yourself.
- Main types of event
  - User Interface events
    - Occur when a user interacts with the browsers user interface (UI) rather than the web page/canvas
  - Focus events
  - Keyboard events
  - Mouse events

Event	Description
load	Web page has finished loading
Unload	Web page is unloading (usually because a new page was requested)
error	Browser encounters a JavaScript error or an asset doesn’t exist
resize	Browser window has been resized
scroll	User has scrolled u or down the page

How events trigger JavaScript code

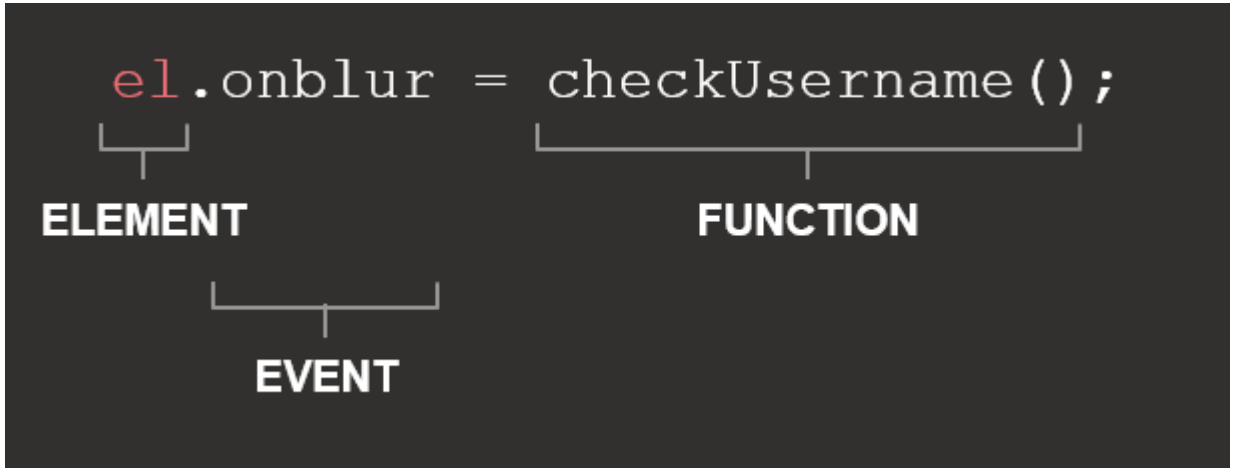
- Event Handlers
- There are three ways to bind an event to an element:
  - HTML event handler attributes
  - Traditional DOM event handlers
  - DOM Level 2 event listeners
- The following examples show a **blur** event on an element stored in a variable called `el` that triggers a function called `checkUsername()`.

HTML event handler attributes

- **(DO NOT USE)**
  - Do not use this way to handle events but just be aware if reviewing older code.

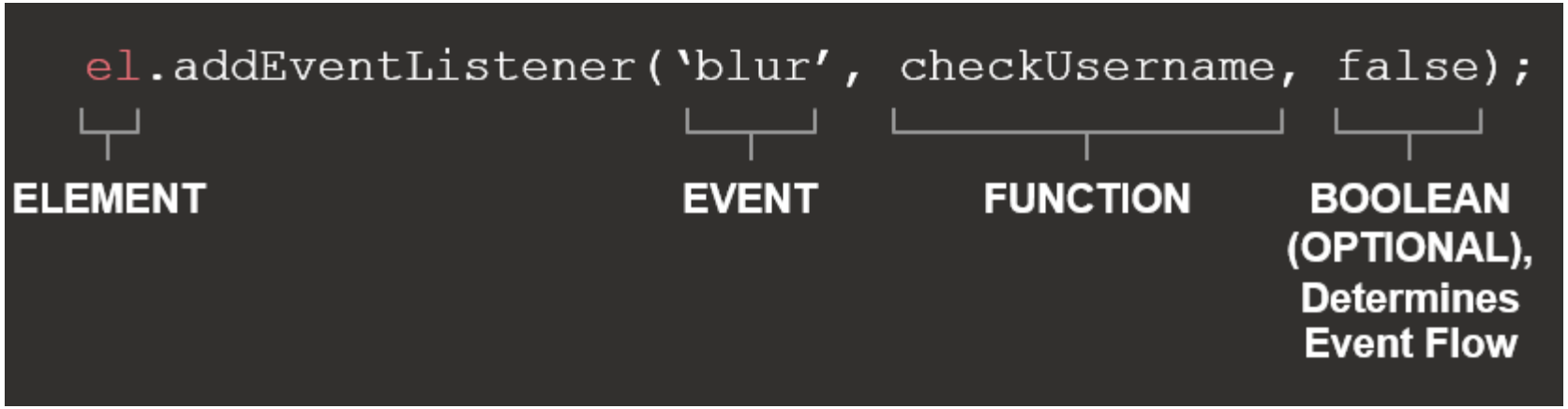
Traditional DOM event handlers

- This way allows you separate the JavaScript from the HTML. The main drawback of this approach is that you can only attach a single function to a any event.



DOM Level 2 event listeners

- This is now the favoured way of handling events.



- Since you can't have parentheses after the function names in event handlers or listeners, passing arguments requires a workaround.
- Parameters with Event Listeners:

```
el.addEventListener('blur', function() {
    checkUsername(5);
}, false);
```

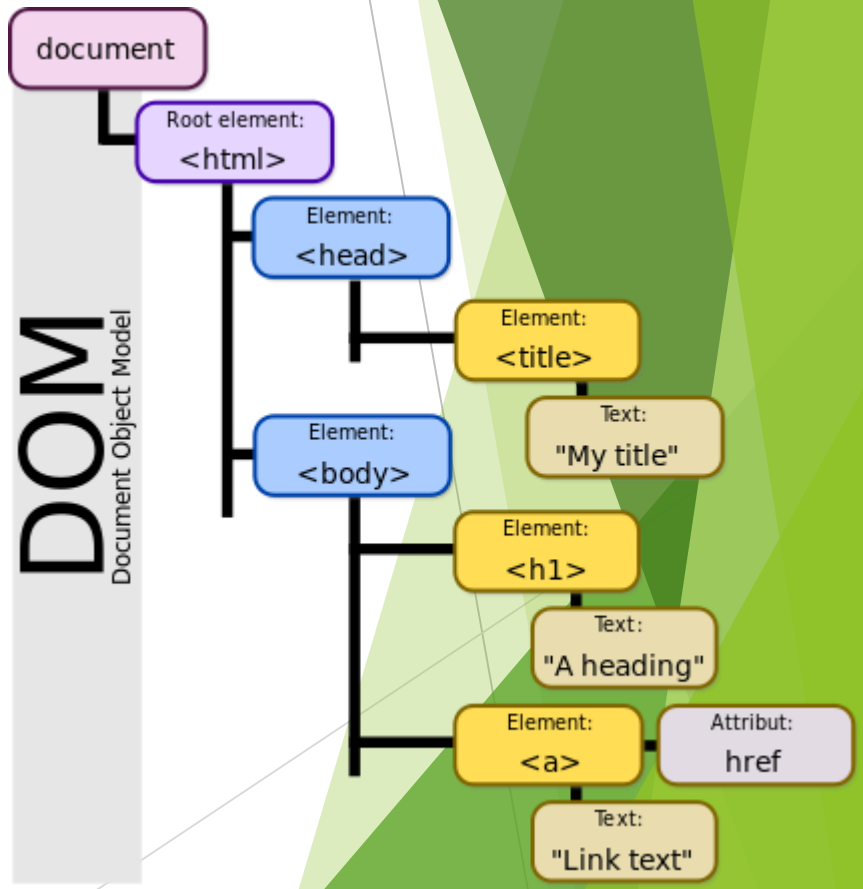
Anonymous function as second parameter enclosing named function

Types of events

- Can use JavaScript event listener, however, the event listener model can vary depending upon the browser implementation
- Quickest way to get up and running is to use a JavaScript library to normalize the handling of events.
  - jQuery
  - Kibo: https://github.com/marquete/kibo

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.



Four principles of jQuery development

- Separation of JavaScript and HTML
  - Provides simple syntax for adding event handlers to the DOM, rather than adding HTML event attributes to call JavaScript functions
- Brevity and clarity:
  - Includes features like chainable functions and shorthand function names.
- Elimination of cross-browser incompatibilities
  - JavaScript engines of different browsers differ slightly so JavaScript code that works for one browser may not work for another. jQuery handles all these cross-browser inconsistencies and provides a consistent interface that works across different browsers.
- Extensibility: New events, elements, and methods can be easily added and then reused as a plugin.

```
$('div.test').add('p.quote').addClass('blue').slideDown('slow');
```

Keyboard events

Event	Description
keydown	User first presses a key
keyup	User releases a key
keypress	Character is being inserted

- Simplest types of events to listen for and handle are keyboard events.
  - Not dependant on the Canvas element or the user's cursor position
  - Keyboard events simply require you to listen for the keydown, keyup, and keypress events at the document level.
  - Keep sensible key-combos in mind

```
// Add an event listener to the keypress event.
window.addEventListener("keypress", function(event) {
    // Just log the event to the console.
    console.log(event);
});
```

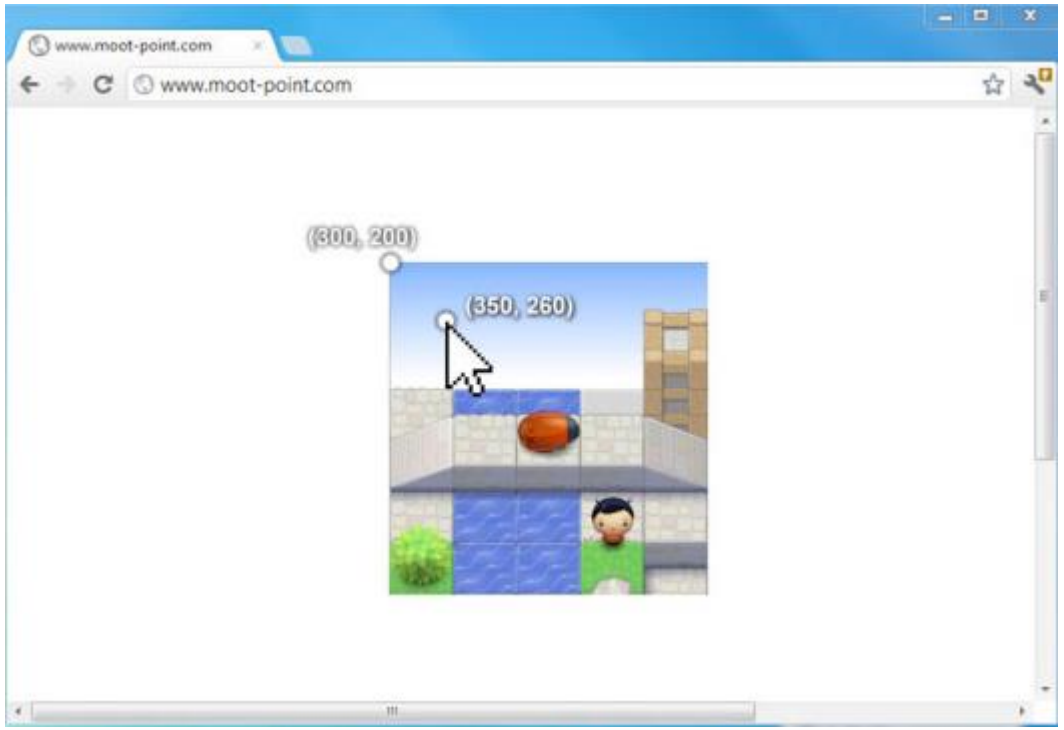
```
$(document.body).on('keydown', function(e) {
    console.log(e.which);
    switch (e.which) {
    // key code for left arrow
    case 37:
        console.log('left arrow key pressed!');
        break;

    // key code for right arrow
    case 39:
        console.log('right arrow key pressed!');
        break;
    }
});
```

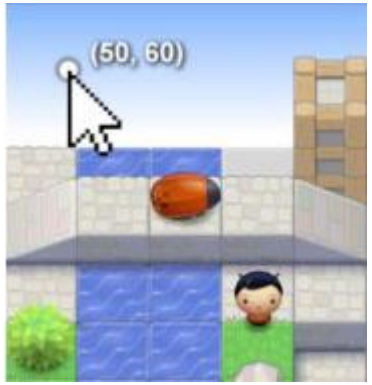
Mouse events

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.pageX and e.pageY 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.

Global coordinates vs. Local coordinates





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 canvas.offset().left and canvas.offset().top
  - Determine the global position of the mouse in relation to the entire document - – Use (e.clientX, e.clientY) or (e.pageX, e.pageY)
  - 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
});
```

Mobile compatibility

- To make the example game compatible with mobile devices, you'll need to work with touch events rather than mouse events.
  - Although a tap of the finger can also be interpreted by the mobile browser as a click event, it is generally not a good approach to rely on listening to only click events on mobile browsers. A better approach is to attach listeners for specific touch events in order to guarantee the best responsiveness.
- Detecting touch events
  - You can write a helper function that first detects whether the device supports touch events and then returns either mouse coordinates or touch coordinates accordingly. This lets calling functions agnostically process input coordinates regardless of whether you're on a desktop or mobile platform.

Mobile compatibility

```
function getPosition(e) {
    var position = {x: null, y: null};

    if (Modernizr.touch) { //global variable detecting touch support
        if (e.touches && e.touches.length > 0) {
            position.x = e.touches[0].pageX - canvasPosition.x;
            position.y = e.touches[0].pageY - canvasPosition.y;
        }
    }
    else {
        position.x = e.pageX - canvasPosition.x;
        position.y = e.pageY - canvasPosition.y;
    }

    return position;
}
```

This week's lab

- User interaction and the Canvas
- Two files provided – mouse.html and keyboard.html
- Have a circle drawn initially on the canvas, and when the user presses any of the arrow keys, have the circle move in that direction. You might have to use the keydown event rather than keypress, if it is not working.
- Edit the code so that the ball cannot go off the edge of the canvas.

More advanced collisions

- Last week, we looked at collision between ball and canvas border
- What if collision between 2 balls?
- How to detect collision?
- How do balls move following collision?

How to detect collision?

- Collision detection is the process of figuring out if two objects are intersecting with one another
- There are many methods but they all fit in two categories
  - A priori collision detection detects if moving the objects in a specific way will cause them to collide
  - A posteriori collision detection detects if two objects are in collision after they have moved

How to detect collisions between circles?

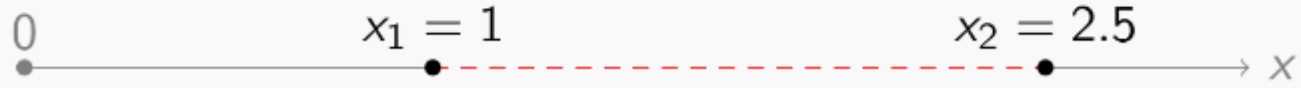
How to detect collisions between circles?

- 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:

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

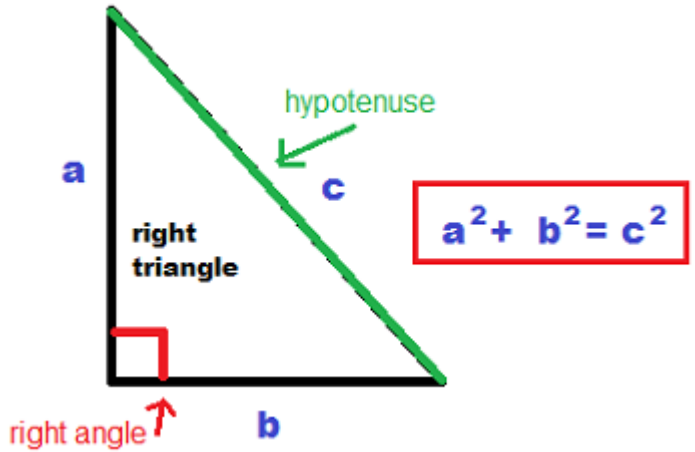


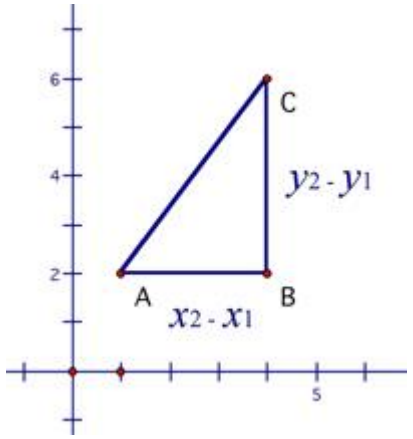
Distance in 2D

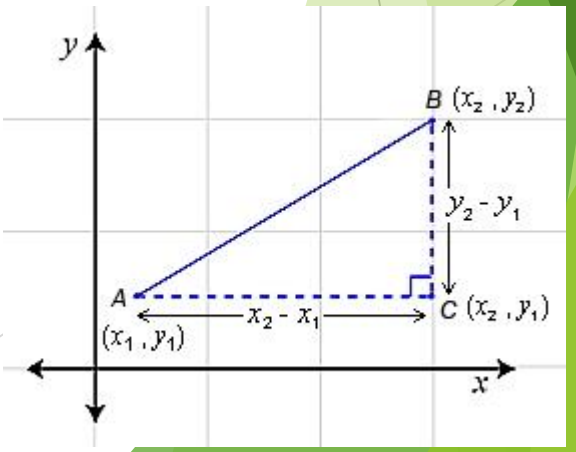
- In two dimensions, the distance between two points (x1,y1) and (x2,y2) can be calculated using the following formula:

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

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







Distance in 3D

- In three dimensions, the distance between two points (x1,y1,z1) and (x2,y2,z2) can be calculated with the formula:

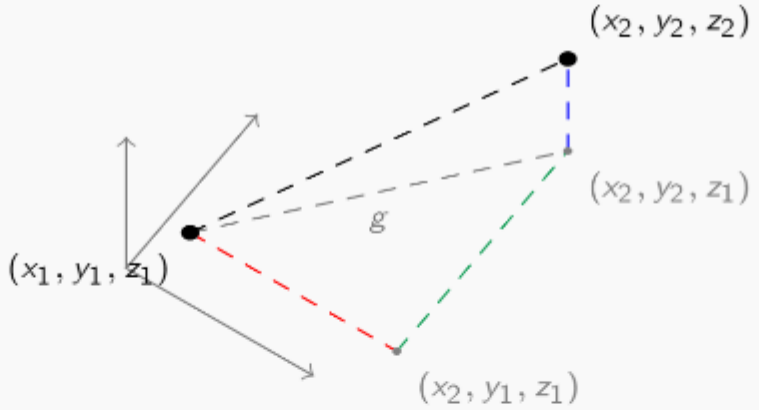
$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

- Why does this work? Again, we use Pythagoras' theorem. See the next slide for explanation

Distance in 3D

In calculating the length of the gray line, we can ignore the z-coordinate as it's the same everywhere on the line. The length is then $g = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$.

Then we can calculate the length of the black line as $\sqrt{g^2 + (z_2 - z_1)^2}$.



Altogether we get the length of the black line as:

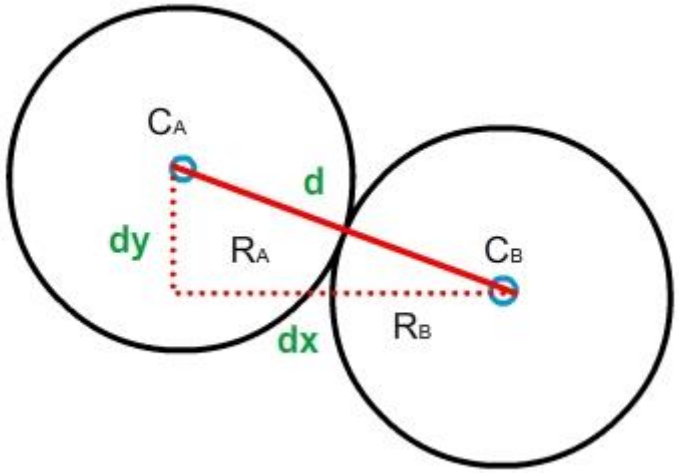
$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

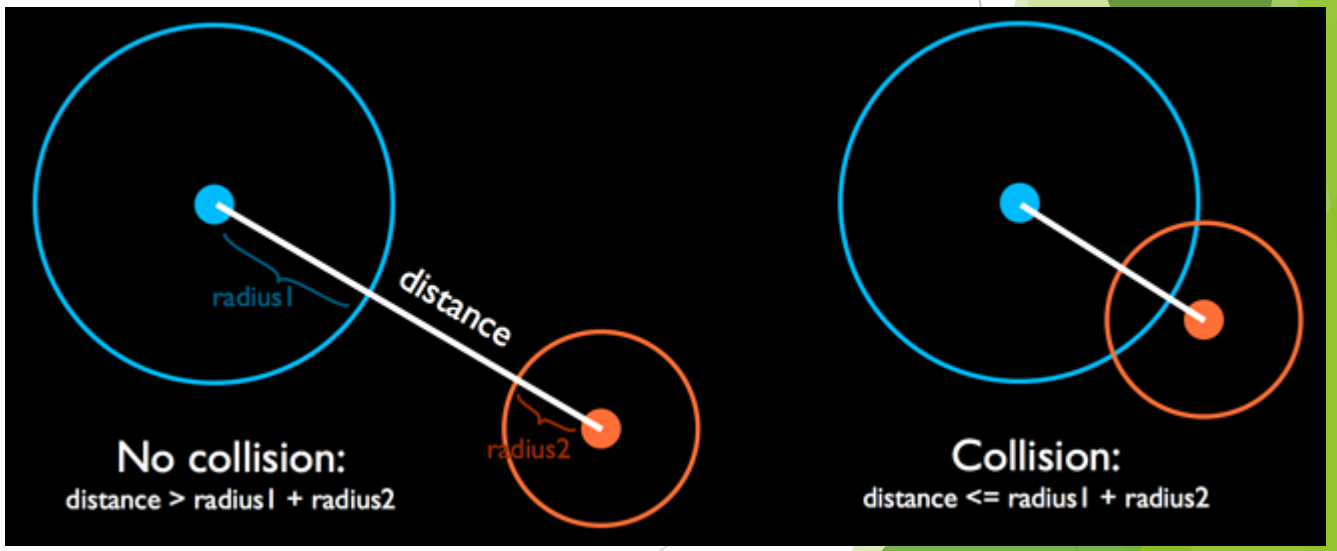
How to detect collisions between circles?

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

How to detect collisions between circles?

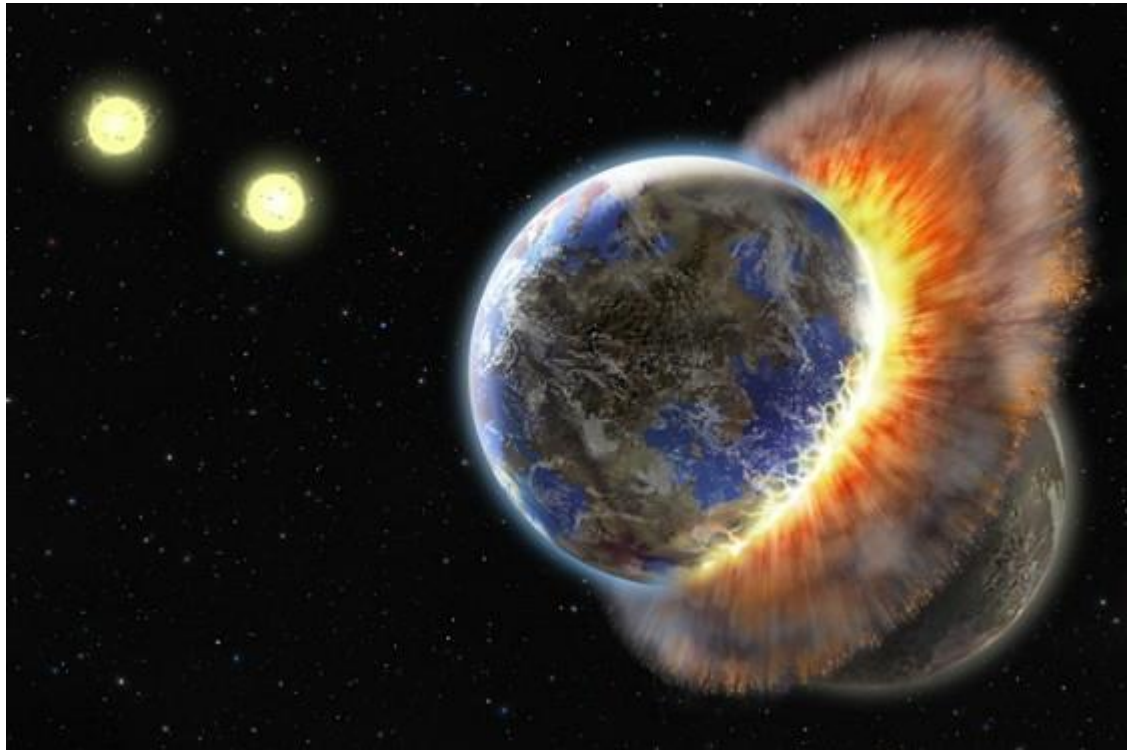
- Now we have the distance between both centres.
- What now?
  - If the distance is less than the sum of the radii, then, the circles are intersecting





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