Bounce the Ball

We have a game component and a game that can run it.

resize() called once
render() called 60 times a second
update() called 60 times a second
destroy() called 60 times a second

Tweak your instance variables:

```
// instance variables
double x = 0;  // the x location of the ball
double y = 0;  // the y location of the
double width;  // width of screen
double height;  // height of screen
double speedX;  // speed in the x direction
double speedY;  // speed in the y direction
int lives = 10;  // how many bounces until the ball dies
```

One more instance variable

```
Paint paint = Paint();  // paint the ball circle
```

The Paint class is in the dart:ui library.

We use the paint object to draw on the canvas.

Set paint color, brush width, and if we want an outline or filled-in shape

We will set up our paint variable in the constructor.

```
// create a ball

Ball() : super() {
  paint.color = Colors.green;
  paint.strokeWidth = 1;
  paint.style = PaintingStyle.stroke;
}
```

Need an import for the Colors class

```
import 'package:flutter/material.dart';
```

Imports

Here are the imports we have so far for our Ball class.

```
import 'dart:ui';
import 'package:flutter/material.dart';
import 'package:flame/components/component.dart';
```

How do we know what imports to use?

How do we know what classes are available to us with these imports?

There are thousands of classes we can use.

Look at Flutter and Dart documentation

Flame documentation

Google – StackOverflow is your friend

Resize and the Screen Size

When a game component is created, it does not know the size of the screen.

We need to set our height and width instance variables as soon as resize() is called by the game engine.

Resize and the Screen Size

When a game component is created, it does not know the size of the screen.

We need to set our height and width instance variables as soon as resize() is called by the game engine.

Add this code to the resize() method.

```
// save screen width and height
width = size.width;
height = size.height;
```

Remember, size is an object passed to us by the game engine. It has two properties, width and height which give us our screen dimensions.

Ball Speed and Location

In our instance variables, x and y are defaulted to 0. This means the ball will start out in the upper left hand of the screen.

We need to calculate the ball's speed so that we can make it move

There are 2 speed variables

- speedX – how fast the ball moves in the horizontal direction
- speedY – how fast the ball moves in the vertical direction
- Speed is in pixels per second
- Game time is always calculated in seconds

Calculating Speed

Our ball will move horizontally at the speed of 1/5 screen width per second.

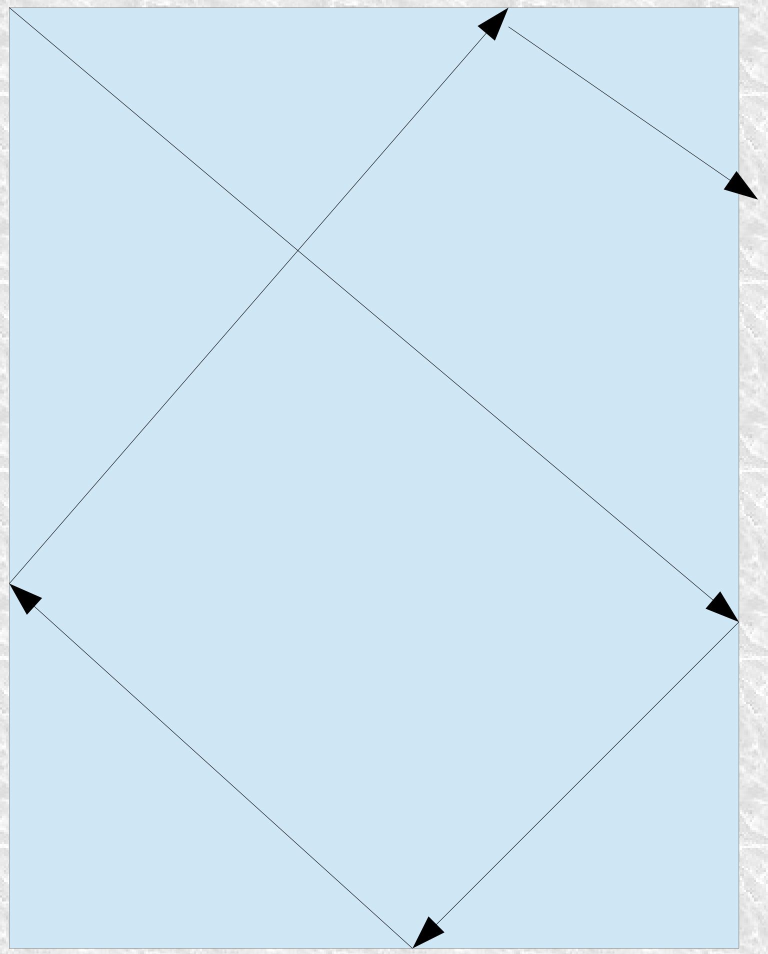
5 seconds to cross the screen

The vertical speed will be the same as the horizontal speed.

Add our speed calculation code to the resize() method.

```
speedX = width/5;
speedY = width/5;
```

Since the phone is taller than it is wide, here is how the ball will bounce.



Drawing the Ball

When our render() method is called, it will get a completely black screen.

No other components have drawn on it.

It needs to draw the ball – a circle at the current x,y coordinates.

Add this code to render()

```
c.drawCircle(Offset(x,y), 10, paint);
```

The circle is 10 pixels in radius, painted by our paint object which is a green paint brush.

If you were to run the game now, you would see a small green circle in the upper left of the screen.

Moving the Ball

It's in our update() method that we move the ball across the screen.

How much do we move the ball?

Example: screen width 100 pixels

SpeedX is 1/5 of a screen per second or 100 / 5 = 20 pixels per second

Our phone is running at 10 frames per second

Time per frame is .1 second

Multiply frame time by speed to get how much we move per frame

20 * .1 = 2 pixels.

In our update method, we would move the ball 2 pixels

speedX = speedX + 2;

Add this code to the update() method to move the ball.

```
// move the ball
x += t*speedX;
y += t*speedY;
```

Notice that it doesn't matter what the frame rate is. The ball will always move at the same speed.

If you run the game now, the ball will move diagonally down the screen until it goes off the edge and will disappear.

We need to control the bounce.

What makes up a bounce?

A Bounce

What happens when a ball bounces off the right edge of the screen?

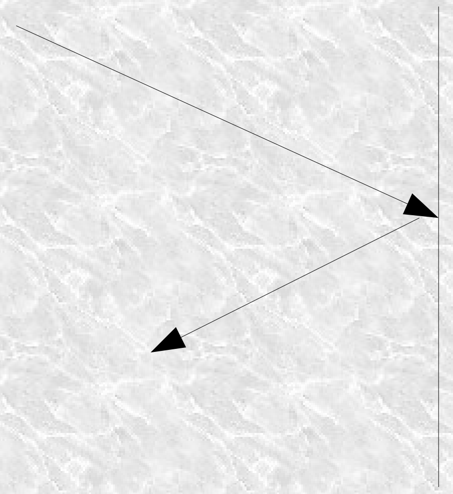
The ball changes direction in the horizontal direction

(from moving right to moving left) but keeps the same speed

The ball keeps going in the same vertical direction (down)

How do we change the ball's horizontal direction?

speedX = -speedX;



What happens when a ball bounces off the left edge of the screen?

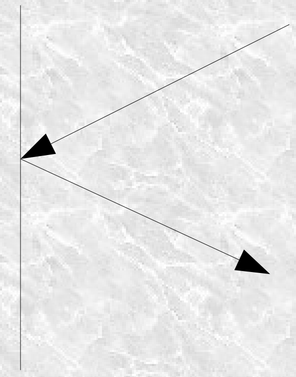
The ball changes direction in the horizontal direction

(from moving left to moving right) but keeps the same speed

The ball keeps going in the same vertical direction (down)

How do we change the ball's horizontal direction?

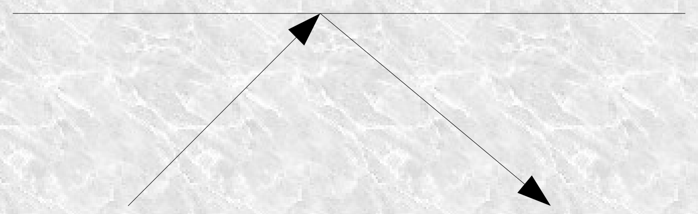
speedX = -speedX;



So every time the ball bounces off a left or right edge, we just reverse the speedX variable

Try to guess what will happen when the ball bounces off the top and bottom of the screen

What happens when a ball bounces off the top edge of the screen?



The ball changes direction in the vertical direction

(from moving up to moving down) but keeps the same speed

The ball keeps going in the same horizontal direction (to the right)

How do we change the ball's vertical direction?

```
speedY = -speedY;
```

It works the same way when the ball bounces off the bottom edge.

Every time the ball bounces off a top or bottom edge, we just reverse the speedY variable

We put this bounce logic in our update method, and our ball will bounce.

A Bouncing Ball

Add this code to your update() method

```
// change direction if ball crosses edge
if (x < 0) { // hit the left edge
  speedX = -speedX;
  x = 0;
  lives--;
}
if (x > width) { // hit the right edge
  speedX = -speedX;
  x = width;
  lives--;
}
if (y < 0) { // hit the top edge
  speedY = -speedY;
  y = 0;
  lives--;
}
if (y > height) { // hit the bottom edge
  speedY = -speedY;
  y = height;
  lives--;
}
```

A Dying Ball

The only thing left to do is kill the ball when it runs out of lives. This is a simple block of code in our destroy method.

```
// tell the game engine if this component should be destroyed
// whenever it asks
bool destroy() {
  if (lives > 0) {  // not dead yet
    return false;
  } else {  // dead
    return true;
  }
}
```

Or if you are into writing really concise code, you could write it like this:

```
bool destroy() => lives <= 0;
```

Next Up

Your ball should bounce around the screen and then die. If you are having problems, go to my github page.

https://github.com/shawnlg/flutter_ball/tree/02_bounce_circle

In the next section, we will make some improvements to our ball.

- Allow for many balls at the same time
- Make the Ball class easier to use
- Learn a little more about Dart classes