Lets Bounce Off Blocks

Who Bounces the Ball?

Imagine we have a screen full of colored blocks.

We have a ball bouncing around the screen

It bounces off the edge

It bounces off the blocks

Do the blocks determine if they have been bounced into?

Then change the direction of the ball

Does the ball check all the blocks to see if it hit one?

Then change its own direction

Either way will work

If I Were a Ball

- I would want to have control of my own destiny
- I would want to be the one to react, not have something tell me how to react
- This allows for different balls to follow different rules
- Maybe a big ball breaks a block instead of bouncing off it
- Maybe a ball decides to bounce differently depending on the block

We will have the ball check the blocks to see if it bounces into one

First it checks if it bounces off the side

Then it checks all the blocks to see if it is inside one

If so, it changes direction and gets out of it

Changes to Update Method

Update is where we check if we bounce off the side

Lots of logic is in there to check all the sides

There will be much more logic if we check for block bounces

Messy

We will start by taking out the bounce code and replacing it with a method call

We will make 2 methods

- screenBounce() will return true if the ball bounces off the edge
- blockBounce() will return true if the ball bounces off a block
- If either are true, we make the bounce sound and decrease the lives variable

Let's do the easy one first

Add this method to the Ball class before update()

```
// check if we bounced off the edge of the screen
bool screenBounce() {
  if (x < 0) {  // off the vertical edge
    speedX = -speedX;  // reverse x direction
    x = 0;  // move back into screen
    return true;
  } else if (x > width) {
    speedX = -speedX;  // reverse x direction
    x = width;  // move back into screen
    return true;
  } else if (y < 0) {
    speedY = -speedY;  // reverse y direction
    y = 0;  // move back into screen
    return true;
  } else if (y > height) {
    speedY = -speedY;  // reverse y direction
    y = height;  // move back into screen
    return true;
  } else {
    return false;
  }
}
```

It's basically the code we took out of update() to check for bouncing

blockBounce()

Here is what the blockBounce() method needs to do.

- It is passed a time variable. This helps it move the ball at the right speed out of the block if finds itself in one
- It gets a list of all components that have been added to the game
- It only cares about Block ones
- For each block, it checks to see if it is inside the block rectangle

  This means it must have hit the edge of that block to get inside of it
- If it's inside the block, it needs to figure out which edge it is closes to

  This way it will know in which direction to bounce

  Just like the sides of the screen, different edges change the direction differently

Make the method

```
// check if we bounced off a block
bool blockBounce(double t) {
  bool bounced = false;  // set to true if we bounced off any block
  ...more code goes here
  return bounced;
}
```

The method is passed the time parameter

We have a flag that tells if we have bounced or not

After doing other stuff that may set the flag to true, we return it

Since update() call this method, it has the time already, so it can pass it to this method.

Next we get a list of all block components

```
// go through the game components
game.components.forEach((component) {
  if (component is Block) {
    Block block = component;  // reference this component as a block
    .. more code goes here ..
  } // if component is a block
});
```

How We Found the Blocks

We have used the forEach() method on a list before.

Remember checking all the drums to see if any were tapped during recording

We do a similar thing here

The game object has a built-in list called components

It is a list of all the components added to the game

Some balls, some blocks, some messages, whatever

We have to determine if the component is made from the Block class or not

That's what the `is` operator does in Dart

It tests an object to see if that object was made from a specific class

All components are made from Component, but only some are also made from Block

Inside the Block

We test to see if the ball is inside the block

Remember, we do this test over and over for each of the block components in the list

Add this code inside the test for being a Block

```
if (block.position.contains(Offset(x,y))) {
  // ball is inside this block, so we bounced
  bounced = true;
  block.lives--;
  … more code …
} // if ball inside block
```

In this code, we determine that we are inside the block.

We do two things. We set the bounced flag to true

We decrease the lives of the block we bounced on

Which Edge Did We Bounce Off

- If we hit the top edge, we want to change the down direction to up
- If we hit the bottom edge, we want to change the up direction to down
- If we hit the left edge, we want to change our right direction to left
- If we hit the right edge, we want to change our left direction to right
- Just like bouncing off the sides

Here comes some nasty math code

```
// see if we are closest to an x side of the block (left, right) or a y side (top, buttom)
double closestX = min(x - block.position.left, block.position.right - x);
double closestY = min(y - block.position.top, block.position.bottom - y);
```

We will look at this more carefully

How close are we to the left side?

If we are inside the block, the ball's x must be bigger than the block's left edge

```
x - block.position.left // how close the ball is to the left of the block
```

Similar for the right side, but the ball x is less than the right edge

```
block.position.right - x
```

We want the smallest of these 2 numbers which will tell us how close we are to a vertical edge

We use the min() method from the dart math library

```
min(x - block.position.left, block.position.right - x)
```

Similar logic for the top and bottom side. We want to see how close we are to a horizontal edge

```
min(y - block.position.top, block.position.bottom - y)
```

If we are closer to a horizontal edge, we change the up/down direction

If we are closer to a vertical edge, we change the left/right direction

The Bounce

Here is how we do the bounce

```
if (closestX < closestY) {
  // we are closest to the left/right of the block, so we hit a vertical edge
  speedX = -speedX;  // reverse x direction
} else {
  // we are closest to the top/bottom of the block, so we hit a horizontal edge
  speedY = -speedY;  // reverse y direction
}
```

Once we know which direction to bounce, we change that direction. Just like bouncing off the sides

We are still inside the block. If we don't get out, we might bounce again on that same block. This would happen if the next update had a smaller time and didn't move the ball all the way outside the block.

Just to make sure we are out, we move the block until it's out from inside the block

Add this code after the bounce code

```
// move ball until it is outside of block again
while (block.position.contains(Offset(x,y))) {
  // move the ball
  x += t*speedX;
  y += t*speedY;
} // while ball inside block
```

We keep moving the ball until we are no longer inside the block

The move is very similar to the normal move we do inside update()

Our update() method becomes pretty simple now

- We only do an update if a resize() has been done
- We move the ball in the current direction it is going
- We check for bouncing off the sides
- We check for bouncing off the blocks
- We play the sound if we bounced

New and Improved update()

Here is our new update() method

```
// update this component whenever the game engine tells you to
void update(double t) {
  // don't update until size is set
  if (width < 1 || height < 1) return;

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

  if (screenBounce() || blockBounce(t)) {
    // play sound
    Flame.audio.play('bounce.wav');
    lives--;  // lost a life after a bounce
  } // if we did a bounce
}
```

Try it Out

Add some blocks and a ball to the game and run it

```
// make new block game components
var block = Block(this, position: Rect.fromLTWH(100, 200, 50, 50), draggableBlock:
true);
var block2 = Block(this, position: Rect.fromLTWH(200, 400, 50, 50), draggableBlock:
false);

// tell the game about this blocks
add(block);
add(block2);
Block.canDrag = true;   // ok to drag blocks

// add a ball
Ball ball = Ball(this, speedX: 0.2, speedY: 0.3, lives: 50);
add(ball);
```

Remember, you can drag around one of those blocks

Next Time

If you are stuck, the code is here.

https://github.com/shawnlg/flutter_ball/tree/15_bounce_block

Next time, we will change the bounce to behave differently if it bounces off a draggable block than one that cannot be dragged

We will call the draggable block our “aiming block”

We can change the direction of the bounce by where on the edge we bounce the ball