Dragging a Line

As part of the ball launching process, the ball launcher will draw a line to help you aim the ball. One end of the line will start where you first touch the screen, and the other end will be where you drag your finger. We need some instance variables to let the ball releaser know if it is time to draw a line or release a ball. Add these.

```
// instance variables
FlutterballGame game;
bool makingLine = false;  // if we are in the middle of drawing a line during a drag
Offset lineStart;
Offset lineEnd;
Paint paint = Paint();  // paint the line
double width;  // width of screen
double height;  // height of screen
```

Some of these you will already have if you made the component beforehand.
The variable makingLine helps us determine when we start drawing the line. The offset variables are the two x,y coordinates that define the ends of the line.

Let's start with an easy method – render().

```
void render(Canvas c) {
  if (makingLine) {
    c.drawLine(lineStart, lineEnd, paint);
  }
}
```

We use the canvas and another drawing command to draw a line – but only when we are actually supposed to draw, determined by drawLine.

And the constructor which is like many others we have made.

```
// create the component
InteractiveBallReleaser(this.game) : super() {
  paint.color = Colors.white;
  paint.strokeWidth = 1;
  paint.style = PaintingStyle.stroke;
}
```

The paint variable is set up to draw thin white lines.

Here's our standard resize method.

```
// the game engine will tell you what the screen size is
void resize(Size size) {
  // save screen width and height
  width = size.width;
  height = size.height;
}
```

We need to fix our Ball class.

If we want to launch our ball from any location, we need to be able to pass a starting location to our Ball constructor. Right now, our ball just starts out at 0,0. So tweak the constructor in ball.dart.

```
Ball({this.x:0, this.y:0, color:Colors.white, size:10.0, speedX=1.0, speedY=1.0, style
= PaintingStyle.stroke, this.lives:10}) : super() {
  paint.color = color;
  paint.strokeWidth = 1;
  paint.style = style;
  ballSize = size;
  speedScaleX = speedX;
  speedScaleY = speedY;
}
```

Line Logic

The update method has to do several things to handle the line and ball launch

- When dragging first starts, the first end of the line is recorded – where the finger is
- As long as the user keeps dragging, a line is drawn connecting the first point with the current finger location
- Once dragging stops, we stop drawing a line and launch the ball
- The cycle repeats every time the user drags.

Here are the basic tests we need to perform in update().

```
if (!game.isDragging && !makingLine) { // not dragging and not making a line
} else if (game.isDragging && !makingLine) {
  // user is dragging, but we haven't yet started making a line
} else if (!game.isDragging && makingLine) {
  // user no longer dragging but we are still making the line
}
```

The first case we don't care about, so we do nothing.

```
if (!game.isDragging && !makingLine) {
  // user is not dragging and we are not making a line, so nothing to do
```

The next case is also fairly simple

```
} else if (game.isDragging && !makingLine) {
  // user is dragging, but we haven't yet started making a line
  lineStart = Offset(game.dragX, game.dragY);  // line starts where finger is
  lineEnd = Offset(game.dragX, game.dragY);  // line ends where it starts - single
point
  makingLine = true;  // start rendering the line
```

We start by making the start and the end of the line be where the user first starts dragging.

We'll skip to the last case because that is also an easy one.

```
} else {
  // user is still dragging and we are still making a line
  // just update the endpoint of the line in case user has moved finger
  lineEnd = Offset(game.dragX, game.dragY);  // line ends where it starts - single
point
}
```

In this case, we just move the end of the line with the finger drag position, so as the user moves their finger, that end of the line moves with it.

We finally get to the most complicated of the cases.

```
} else if (!game.isDragging && makingLine) {
  // user no longer dragging but we are still making the line
  makingLine = false;  // stop making the line
  // launch ball
  double speedX = (lineEnd.dx - lineStart.dx) / width;
  double speedY = (lineEnd.dy - lineStart.dy) / height;
  Ball ball = Ball(x:lineStart.dx, y:lineStart.dy, speedX: speedX, speedY: speedY);
  game.add(ball);
```

When we are done dragging, we launch the line. The x and y speeds are proportional to the length of the line in that direction.

We create the ball at the start of the line and give it the speed based on the length of the line, and the ball will go in the direction of the line.

We'll show the entire update() method, since it has so many parts.

```
void update(double t) {
  if (!game.isDragging && !makingLine) {
    // user is not dragging and we are not making a line, so nothing to do
  } else if (game.isDragging && !makingLine) {
    // user is dragging, but we haven't yet started making a line
    lineStart = Offset(game.dragX, game.dragY);  // line starts where finger is
    lineEnd = Offset(game.dragX, game.dragY);  // line ends where it starts - single
point
    makingLine = true;  // start rendering the line
  } else if (!game.isDragging && makingLine) {
    // user no longer dragging but we are still making the line
    makingLine = false;  // stop making the line
    // launch ball
    double speedX = (lineEnd.dx - lineStart.dx) / width;
    double speedY = (lineEnd.dy - lineStart.dy) / height;
    Ball ball = Ball(x:lineStart.dx, y:lineStart.dy, speedX: speedX, speedY: speedY);
    game.add(ball);
  } else {
    // user is still dragging and we are still making a line
    // just update the endpoint of the line in case user has moved finger
    lineEnd = Offset(game.dragX, game.dragY);  // line ends where it starts - single
point
  }
}
```

In Conclusion

Try and run the game. You can launch as many balls as you want. Try and change the lives of the balls and see how many you can get bouncing.

As usual, the source code for this section is available in GitHub.

https://github.com/shawnlg/flutter_ball/tree/06_interactive_balls

Next, we will learn how to add images and sound to a game. We will take a break from bouncing balls and make...

A drum machine!