Game Controller

What Is a Controller

Remember our drum controller

Components can do many things by themselves

Drums can display, can play when tapped on

We need something else to coordinate things when many components are involved

Lots of blocks

Keep track of when the game is over

Show information on the screen

We are going to make a component called GamePlay

Create a file called game_play.dart in the components folder

Add the usual methods for a new game component

General Tweaking

When designing a game, you often have some ideas of things to add to the game that will require little tweaks throughout the game.

You may want to have borders around the blocks which will mean the Block component gets tweaked.

You realize that you need to control how many lives the ball has that is released from the Interactive Ball Releaser, so you add that parameter

You find that some things are hard to do with the existing code, so you tweak things to make the code easier to use.

We will be doing a number of such tweaks before continuing on with the game

Changing Text

Right now, if we want to put a message on the screen, we create a TextDraw component, passing it the text to draw.

No easy way to change the text

We will add a setter method that lets you pass in new text and it just changes that and leaves everything else alone

Add this method in the TextDraw class.

```
// change text being displayed
set text(String text) {
  tp.text = TextSpan(text: text, style: tp.text.style);
  tp.layout(minWidth: position.width, maxWidth: position.width,);

  // see if box is too short
  if (tp.height > position.height) {
    position = Rect.fromLTRB(position.left, position.top, position.right,
        position.top + tp.height);
  }
}
```

Why do we this?

```
tp.text = TextSpan(text: text, style: tp.text.style);
```

Why not just change the text in the paint's text span?

Once you create a text span, you cannot change it. We need to make a new one

Our new text span uses the text style in the old text span

```
tp.text = TextSpan(text: text, style: tp.text.style);
```

This is because the tp.text variable is the old text span until after the new text span is created.

We create a new text span, passing it the new text and the old style

We call layout() again to compile the new text

We check the size again incase the new text makes the block grow longer

Now, if we have a score text box, all we have to do is replace the text on it and it will change the next time render() is called.

Borders

We have borders on our text boxes, but we don't have them on our blocks. We need to add in the same border code to our Block class. Add these changes.

```
Paint borderPaint = Paint();  // paint the border


Block(this.game, {this.position, color:Colors.white, borderColor:null, lives:10,
      this.draggableBlock = false,}) : super() {


if (borderColor != null) {
  borderPaint.color = borderColor;
  borderPaint.style = PaintingStyle.stroke;
  borderPaint.strokeWidth = 1.0;
} else {
  borderPaint = null;
}


if (borderPaint != null) c.drawRect(position, borderPaint);
```

Ball Releaser

We want to pass in the lives for the ball that gets released.

```
int lives; // how many lives the ball launched should have

InteractiveBallReleaser(this.game, {this.lives:1}) : super() {

Ball ball = Ball(game, x: lineStart.dx, y: lineStart.dy,
    speedX: speedX, speedY: speedY,
    lives: lives, sound: true);
```

We want to control the speed of the launch. We will add a speedScale constructor parameter. If it is 0, we launch as usual where the speed is determined by the length of the line. If the speedScale is non-zero, it represents how many times the speed of 1 screen-width per second we go.

A speedScale of 5 means the ball goes 5 screen widths per second.

Add this code to the ball releaser:

```
double speedScale;  // how much we scale speed based on line length

InteractiveBallReleaser(this.game, {this.lives:1, this.speedScale:1.0}) : super() {

// launch ball
double speedX = (lineEnd.dx - lineStart.dx) / width;
double speedY = (lineEnd.dy - lineStart.dy) / height;
if (speedScale != 0) { // fixed speed
  // get the speed of 1 screen width per second
  double currentSpeed = sqrt(speedX*speedX + speedY*speedY);
  double toNominalSpeed = 1.0 / currentSpeed;
  speedX = speedX * toNominalSpeed * speedScale;
  speedY = speedY * toNominalSpeed * speedScale;
}
```

And we need a way of destroying the ball releaser

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

Level Buttons

We now make some changes to our game_intro.dart file.

We will add two level buttons – a “-” and “+” button on either side of the Start button

Part of keeping a game interesting is coming up with lots of levels. We will work on some different levels next time, but we will add the buttons now.

```
TextDraw levelPlus;
TextDraw levelMinus;
```

We need to tweak our start button slightly. The text needs to be a little smaller because it will now say Start Level 1 or whatever level we are on.

```
TextStyle startStyle = TextStyle(fontSize: 20, color: Colors.blue);
TextSpan startSpan = TextSpan(text: "Start\nLevel 1", style: startStyle);
startButton = TextDraw(Rect.fromLTWH(width*0.3, height*0.6, width*0.4, 116),
startSpan,
  boxColor: Colors.blueGrey, borderColor: null, topMargin: 5.0,
);
game.add(startButton);
```

And here are the 2 level buttons

```
TextStyle style = TextStyle(fontSize: 50, color: Colors.blue);
TextSpan span = TextSpan(text: "-", style: style);
levelMinus = TextDraw(Rect.fromLTWH(0, height*0.6, 100, 0), span,
  boxColor: Colors.blueGrey, borderColor: null,
);
game.add(levelMinus);

span = TextSpan(text: "+", style: style);
levelPlus = TextDraw(Rect.fromLTWH(width*0.75, height*0.6, 100, 0), span,
  boxColor: Colors.blueGrey, borderColor: null,
);
game.add(levelPlus);
```

Try running the game and make sure the level buttons line up with the start button and everything still looks ok.

To handle game levels, we will add an instance variable to the flutterball game class itself.

```
int level=1;  // level we are on
```

We keep it in the game so that even if all the components are destroyed, the game will still keep track of the level we are on.

Game Controller

It's time to start working on our game controller

You should have created a new blank component in the game_play.dart file called GamePlay.

We will start by adding some useful constants above the instance variables

```
// constants
static const int MAX_LEVELS = 10;
static const double BALL_SIZE = 5.0;
static const PaintingStyle BALL_STYLE = PaintingStyle.fill;
```

Now that we have the number of levels in our game, we can finish our GameIntro class and handle pressing the level + and – buttons

Add this code to test if the level buttons are pressed

```
if (levelMinus.position.contains(Offset(game.tapX,game.tapY))) {
  // subtract one from the game level and update start button text
  if (game.level > 1) {
    game.level--;
    startButton.text = "Start\nLevel ${game.level}";
  }
}
if (levelPlus.position.contains(Offset(game.tapX,game.tapY))) {
  // subtract one from the game level and update start button text
  if (game.level < GamePlay.MAX_LEVELS) {
    game.level++;
    startButton.text = "Start\nLevel ${game.level}";
  }
}
```

Notice we are using our handy text setter on our level buttons to change the text.

Try it out. Make sure the buttons change the level and don't let you get out of range.

Next Time

The game controller is the most complex part of the game. It has to keep track of lots of things. We need to have it launch the ball, detect when the game is over, go to the next level or put up a lose message.

We will continue adding functionality to the game until we have a basic working game with one level.