Introducing...

The Intro Screen

When you play a game, there is usually a screen before the game that lets you do things

Our intro screen

- The game title in big letters
- A help button to get help on how to play the game
- A start button to play the game
- Lots of tiny random colored balls bouncing around

Text boxes to make the text

Code from the ball releaser to do the randome balls

Need some minor tweaks in the existing code

Ball Tweak

We will have 100 random balls bouncing around the screen.

Right now, the ball makes a bounce sound whenever it hits something

We need to stop a ball from making a sound when it bounces

Try to do it yourself first

An optional parameter in the constructor that says whether we want the ball to make sound – defaults to true

We add a new instance variable to Ball

```
final bool sound;  // if we play the bounce sound
```

We add it to the constructor parameters

```
  Ball(this.game, {this.x:0, this.y:0, this.sound:true,
       color:Colors.white, size:10.0, speedX=1.0, speedY=1.0,
       style = PaintingStyle.stroke, this.lives:10}) : super() {
```

We check the flag before playing the bounce sound

```
if (sound) Flame.audio.play('bounce.wav');
```

TextDraw Tweak

Remember when we paint the lives on the block

We moved the text down some so it wouldn't paint at the very top of the block

We should allow the same thing for a TextDraw block

Try this one on your own

We need to add an optional parameter in the constructor that tells how far down from the top we put the text

We add a new instance variable to TextDraw to hold the distance from the top we will draw the text

```
final topMargin;
```

Add this to the constructor parameters, defaulting to 0

```
this.topMargin:0.0,
```

Finally, add this to the y coordinate where we paint the text

```
tp.paint(c, Offset(position.left, position.top + topMargin));
```

A New Component

We usually add new functionality in the game using a new type of game component

We will make a new GameIntro component

- Draw text boxes for title and the 2 buttons
- Releases random bouncing balls
- Checks if buttons are pressed and does appropriate action
- Can destroy itself, allowing another game component to take over the screen
- Keeps track of what state it is in by a state enum variable

State of Intro Screen

Make a new file in the components folder called game_intro.dart

Below any imports needed for a normal game component, add an enum

```
enum IntroState {
  WAITING,  // waiting for screen size info
  STARTING, // putting up the intro screen
  BALLS,    // showing random balls
  PLAY,// ready to play game
  HELP, // showing help screen
  DEAD, // destroy component
}
```

We don't want to draw anything until we have a screen size

After resize() is called, we need to draw things

Clicking the buttons will put us in PLAY or HELP

Once we load the game to play, we need to disappear render() is super simple, since we don't draw anything directly

```
void render(Canvas c) => null;
```

And destroy returns true when the state is DEAD

```
bool destroy() => state == IntroState.DEAD;
```

We add this component to the Flutterball game

```
GameIntro intro = GameIntro(this);
add(intro);
```

Next, we make two methods to keep our code from being too messy and long

```
// make blocks to display title and buttons
void makeBlocks() {
  ...code goes here...
}


// start the balls bouncing
void startBalls() {
  ...code goes here...
}
```

Bouncing Balls

Adding bouncing balls will look familiar

We did it in the BallReleaser component

See how much of it you can make on your own

Look a BallReleaser for help

- Loop once for each ball (NUMBER_OF_BALLS times)
- Create a random x and y starting location between 0 and the maximum

  x max is width, y max is height
- Create a random color
- Create a ball
- Add the ball

GameIntro Component

We now make our GameIntro class, patterned after the other classes we have which extend Component

Feel free to create a bare-bones class with the needed methods and constructor

Add some constants to the class to tell us how the bouncing balls will look

```
// constants
static const int NUMBER_OF_BALLS = 100;
static const double BALL_SIZE = 5.0;
static const PaintingStyle BALL_STYLE = PaintingStyle.fill;
```

If I accidentally use a hard-coded value in the code, feel free to replace it with a new constant here

Lazy coders use hard-coded values

Instance Variables

We will have our usual instance variables we have seen before along with some new ones.

```
// instance variables
final FlutterballGame game;
IntroState state = IntroState.WAITING;
double width=0;  // size of the screen in the x direction
double height=0;  // size of the screen in the y direction
TextDraw gameTitle;  // game title text
TextDraw startButton;
TextDraw helpButton;
Random rnd = Random();  // rnandom number generator
```

We initialize our state to WAITING, meaning we are waiting for resize() to give us a screen size

We need 3 TextDraw text boxes for the title and 2 buttons

We need a random number generator for the random balls

Easy Ones

The constructor is very simple, we just need to set the game variable

```
// constructor
GameIntro(this.game, ) : super() {
}
```

The resize() method is straightforward

```
void resize(Size size) {
  if (size.width <= 0) return;

  // save screen width and height
  width = size.width;
  height = size.height;
  state = IntroState.STARTING;
}
```

We set the state to STARTING when we have receive a good size

```
// start the balls bouncing
void startBalls() {
  for (int i=0; i<NUMBER_OF_BALLS; i++) {
    double x = rnd.nextDouble()*width;
    double y = rnd.nextDouble()*height;
    double speedX = rnd.nextDouble()*0.4 - 0.2;
    double speedY = rnd.nextDouble()*0.4 - 0.2;
    Color color;
    switch (rnd.nextInt(6)) {
      case 0:
        color = Colors.white;
        break;
      case 1:
        color = Colors.blue;
        break;
      case 2:
        color = Colors.green;
        break;
      case 3:
        color = Colors.deepOrange;
        break;
      case 4:
        color = Colors.pink;
        break;
      case 5:
        color = Colors.purple;
        break;
    }
    var ball = Ball(game, color: color, size: BALL_SIZE,
        speedX: speedX, speedY: speedY, style: BALL_STYLE,
        lives: 99, sound: false, x: x, y: y );
    game.add(ball);
  } // for all balls
}
```

Making Text

We add the 3 text boxes in our makeBlocks method

- Our title block will be near the top of the screen

  The block has no color and no border, just colored text
- Below the title is a smaller start button

  It will have light blue Help text on a colored button
- Below the start button will be a help button

  same size and color as the start button

  red Help text

See if you can create the 3 text boxes on your own

Title Text

Here is the code to create the title text on the screen

```
TextStyle titleStyle = TextStyle(fontSize: 40, color: Colors.blue);
TextSpan titleSpan = TextSpan(text: "Flutter\nBall", style: titleStyle);
gameTitle = TextDraw(Rect.fromLTWH(width*0.1, 10.0, width*0.8, 200.0), titleSpan,
  boxColor: null, borderColor: null,
);
game.add(gameTitle);
```

- It takes experimentation to get the values correct
- The text of the title is size 40 and blue
- The actual text has a line break between Flutter and Ball, so they are on separate lines
- We use fractions of screen width and height to position the box

Buttons

The buttons are similarly created

```
TextStyle startStyle = TextStyle(fontSize: 25, color: Colors.blue);
TextSpan startSpan = TextSpan(text: "Start", style: startStyle);
startButton = TextDraw(Rect.fromLTWH(width*0.3, height*0.6, width*0.4, 80), startSpan,
  boxColor: Colors.blueGrey, borderColor: null, topMargin: 10.0,
);
game.add(startButton);

TextStyle helpStyle = TextStyle(fontSize: 25, color: Colors.red);
TextSpan helpSpan = TextSpan(text: "Help", style: helpStyle);
helpButton = TextDraw(Rect.fromLTWH(width*0.3, height*0.8, width*0.4, 80), helpSpan,
  boxColor: Colors.blueGrey, borderColor: null, topMargin: 10.0,
);
game.add(helpButton);
```

Feel free to test this out

- Add the code directly to the game for testing and seeing the results
- Tweak the colors, sizes, borders to the way you like it

The Update Method

This is the last method we need to put up the intro screen

```
void update(double t) {
  switch (state) {
    case IntroState.STARTING:
      startBalls();
      makeBlocks();
      state = IntroState.BALLS;
      break;
    default:
  }
}
```

For now, the buttons do nothing

The code is here

https://github.com/shawnlg/flutter_ball/tree/16_intro_screen

Next time we add the help screen and button presses