Help

We Will Add

- Check if Help button is clicked
- Display help text
- When clicked, go back to intro screen
- A little tweak to the Ball component

We will allow the ball to not bounce off certain sides

This will allow us to have a ball die when it gets to the bottom

Bounce Sides

Try it yourself

- Add 4 instance variable flags to indicate if we are to ignore each side

  ignoreTop, ignoreRight, ignoreBottom, ignoreLeft
- Add the 4 new parameters to the constructor, allowing them to be set. They should default to false
- In the various bounce tests, if the ball goes outside of the side that is true, it dies instead of bounces if the flag is set
- Run the intro screen again – it should still work
- Set an ignore flag to true and all the balls should disappear when they hit that side

The Code

Here are the instance variables in Ball

```
bool ignoreTop;   // go through top of screen instead of bouncing off it
bool ignoreBottom;
bool ignoreLeft;
bool ignoreRight;
```

Here is the constructor changes

```
Ball(this.game, {this.x=0, this.y=0, this.sound=false, this.lives=100,
     this.ignoreTop:false, this.ignoreBottom:false, this.ignoreLeft:false,
this.ignoreRight:false,
     Color color=Colors.white, double size=10, double speedX=1, double speedY=1,
```

Next are the changes we make to screenBounce()

```
bool screenBounce() {
  if (x < 0) {  // off the vertical edge
    speedX = -speedX;  // reverse x direction
    x = 0;  // move back into screen
    if (ignoreLeft) lives=0;  // off the screen
    return true;
  } else if (x > width) {
    speedX = -speedX;  // reverse x direction
    x = width;  // move back into screen
    if (ignoreRight) lives=0;  // off the screen
    return true;
  } else if (y < 0) {
    speedY = -speedY;  // reverse y direction
    y = 0;  // move back into screen
    if (ignoreTop) lives=0;  // off the screen
    return true;
  } else if (y > height) {
    speedY = -speedY;  // reverse y direction
    y = height;  // move back into screen
    if (ignoreBottom) lives=0;  // off the screen
    return true;
  } else {
    return false;
  }
}
```

Help Text

- A title which says Help
- Several paragraphs of text underneath explaining the rules
- New method makeHelpScreen() similar to makeBlocks() for making the text boxes for the intro screen
- Try and write as much of makeHelpScreen() as you can
- Here is a handy constant

```
static const HELP_TEXT =
  "Drag your finger to launch a ball. "
  "The longer you drag, the faster the ball will go. "
  "The ball will go in the direction you drag.\n\n"
  "Try to bounce the ball and get rid of the blocks. "
  "You can drag around the red block to aim the ball.\n\n"
  "Tap the screen to close help."
    ;
```

The makeHelpScreen() method is straightforward.

Text box for the title, and one for the help paragraph of text

```
// make help screen
void makeHelpScreen() {
  TextStyle helpStyle = TextStyle(fontSize: 12, color: Colors.blue, );
  TextSpan helpSpan = TextSpan(text: "Help", style: helpStyle);
  TextDraw helpTitle = TextDraw(Rect.fromLTWH(0, 0, width, 50), helpSpan,
    boxColor: null, borderColor: null,
  );
  game.add(helpTitle);

  TextStyle textStyle = TextStyle(fontSize: 12, color: Colors.blue, );
  TextSpan textSpan = TextSpan(text: HELP_TEXT, style: textStyle, );
  TextDraw helpText = TextDraw(Rect.fromLTWH(0, 50, width, 50, ), textSpan,
    boxColor: null, borderColor: null, textAlign: TextAlign.left,
  );
  game.add(helpText);
}
```

Notice we use the text allignment of left for the paragraph

We had the default alignment as centered text.

Destroying Game Components

When we switch from one screen to another, we need a way of easily deleting all the components on the screen

When we display the help paragraph

- Get rid of the 100 bouncing balls
- Get rid of the buttons and title on the intro screen

When we are done displaying the help screen

- Get rid of the help title and paragraph

We will make a handy method on our game class called deleteComponents()

Deletes all balls, blocks, and text boxes on the screen

Method deleteComponents()

You will have seen this technique earlier when we needed to go through all of the components on the game

```
// remove all ball, text, and block components from the game
void clearComponents() {
  components.forEach((c) {
    if (c is Block) {
      c.lives = 0;
    } else if (c is Ball) {
      c.lives = 0;
    } else if (c is TextDraw) {
      c.lives = 0;
    }
  });
}
```

An obvious question

Why not just have one c.lives = 0 and skip all the tests?

Try it and see if that works.

If we just have this code

```
c.lives = 0;
```

It will fail because a component object does not have a lives property

But if we do this

```
if (c is Block) {
  c.lives = 0;
} else if (c is Ball) {
```

The dart compiler knows that the object inside the if is really a Block

So it works

There are better ways of doing this

- Make an interface such as Lives
- Any component with lives in it would implement Lives
- Then we would only need one if instead of 3

```
if (c is Lives) {
  c.lives = 0;
} else if (c is Ball) {
```

Displaying the Help Screen

When our state is BALLS, we need to test the buttons for being pressed

Add this code to update()

```
case IntroState.BALLS:
  // check if any button was tapped
  if (game.wasTapped) {
    // something was tapped, so end it
    game.wasTapped = false;  // reset tap
    if (helpButton.position.contains(Offset(game.tapX,game.tapY))) {
      // clear screen of components and put up help screen
      game.clearComponents();
      makeHelpScreen();
      state = IntroState.HELP;
    }
  }
  break;
```

This code is similar to how we tested for button taps on our drum machine

When the button is tapped, we clear the screen and set the state to HELP

When the help screen is displayed, we test if it is tapped anywhere

If tapped, we clear the screen and start the screen over as though a resize() was just done

This creates the balls again and draws the text and buttons

Add this code to update()

```
case IntroState.HELP:
  // if help screen is tapped, we clear it and start over
  if (game.wasTapped) {
    game.wasTapped = false;  // reset tap
    game.clearComponents();  // removes help block
    state = IntroState.STARTING;  // puts intro screen back and waits for taps again
  }
  break;
```

That's it. Try running the game

The help button should work, and clicking on the help screen should work

The code is here

https://github.com/shawnlg/flutter_ball/tree/17_help

Next Time

- We make a component for actually playing the game
- Add splash screens to show when a player completed a game or lost
- Add the capability for different levels