# Messages

## Text on the Game Screen

We want to have two things on the screen as the player plays the game

- How many balls they have left
- How many bounces the current ball has left

The text will be on the bottom corners of the screen

They are updated whenever there is a bounce or a launch

We'll put the functions in the game_text.dart file

One for creating the message text box

One for updating the block

4 total new functions

## Functions and Methods

You may have noticed sometimes I say functions and sometimes I say methods

They both have parameters and can return something

A function is not inside a class – not associated with an object

A method is within a class and can be called on the object

addLaunchMessage() is a function

ball.render() is a method in the Ball class

Functions were around much longer than objects – from the very start

Dart lets you use both, and they both have their uses

# The Messages

We will add the 2 text boxes in the GamePlay class instance variables

```
TextDraw ballsLeftMessage;   // show balls left for current level
TextDraw bouncesLeftMessage;  // show bounces left for current ball
```

In our game_text.dart file, here is the method for adding the balls left message

```
void addBallsLeftMessage(FlutterballGame game, GamePlay gp) {
  TextStyle messageStyle = TextStyle(fontSize: 12, color: Colors.white);
  TextSpan messageSpan = TextSpan(text: "balls: ${gp.ballsLeft}", style: 
messageStyle);
  gp.ballsLeftMessage = TextDraw(Rect.fromLTWH(5, gp.height-30, 100, 20), messageSpan,
    boxColor: null, borderColor: null, textAlign: TextAlign.left,
  );
  game.add(gp.ballsLeftMessage);
}
```

The message is 30 pixels from the bottom and 5 from the right

And here is the method to update that message on the screen

```
void updateBallsLeftMessage(GamePlay gp) {
  gp.ballsLeftMessage.text = "balls: ${gp.ballsLeft}";
}
```

The other two methods are very similar and are for the other message

```
void addBouncesLeftMessage(FlutterballGame game, GamePlay gp) {
  TextStyle messageStyle = TextStyle(fontSize: 12, color: Colors.white);
  TextSpan messageSpan = TextSpan(text: "bounces: ", style: messageStyle);
  gp.bouncesLeftMessage = TextDraw(Rect.fromLTWH(gp.width-150, gp.height-30, 200, 20), 
messageSpan,
    boxColor: null, borderColor: null, textAlign: TextAlign.left,
  );
  game.add(gp.bouncesLeftMessage);
}

void updateBouncesLeftMessage(GamePlay gp, int bounces) {
  gp.bouncesLeftMessage.text = "bounces: $bounces";
}
```

The main difference is that for update, we have to pass in the bounce number.

This is because it will be on a ball added to the game and can't easily be obtained from the controller.

# Updating the Messages

The game controller is where the messages will be added to the screen and where they will be updated when necessary.

Look through GamePlay and see if you can find likely paces where this will happen.

The checkPlay() method has several places where the messages need to be updated

```
} else if (ball == null) {  // still balls left to launch
  // need to launch another ball, but don't put up a splash screen
  updateBouncesLeftMessage(this,0);
  state = GameState.BALL_OVER;  // launch new ball
} else {
  // still playing, so get the count for the total bounces left
  updateBallsLeftMessage(this);
  updateBouncesLeftMessage(this, ball.lives);
}
```

## The BALL_OVER state in update()

```
case GameState.BALL_OVER:
  if (game.currentTime() > splashOver) {
    // put up level screen if we just put up splash screen
    if (state == GameState.SPLASH) {
      makeLevel(game,this);
      addBallsLeftMessage(game,this);
      addBouncesLeftMessage(game,this);
    }

    // add component that launches the ball
    launcher = InteractiveBallReleaser(game, lives: ballBounces, speedScale:
speedScale);
    game.add(launcher);
    addLaunchMessage(game,this);
    state = GameState.LAUNCHING;
  }
  break;
```

Here we add the new message because we have just created a new level screen

# Try it Out

Run the game and you will see the two text boxes at the bottom of the screen.

They should update as you bounce and launch balls.

The code is here

https://github.com/shawnlg/flutter_ball/tree/19_game_messages

Awhile back, we added a feature to the Ball class to tell it which sides of the screen to ignore.  We will add this to the game, allowing the different levels to have sides ignored. Usually this is the bottom of the screen

Balls will disappear if they get to the bottom