让不懂编程的人爱上iPhone开发(2017秋iOS11+Swift4+Xcode9版)-第10篇

还是让我们继续iPhone开发的学习之旅吧。

其实在第9篇的课程结束后，我们的游戏机制已经基本上实现，游戏逻辑也没有任何大的漏洞。至少到目前为止，我们没有制造出新的bug。

不过这款产品仍然存在着巨大的改进空间。

当然你可能首先想到的游戏界面太丑了，这一点我们会尽快来优化。其实还有一些用户体验的小细节需要我们来进行调整。

首先，提示玩家游戏表现的对话框标题可以改进一下。我们可以让它的内容和玩家的游戏表现紧密相连。

如果玩家把滑动条正好放在目标位置上，那么提示对话框会说，“完美表现！“，如果滑动条的位置非常接近目标，但还没有百分百对准，那么对话框会说，”太棒了！差一点就到了！“如果滑动条的位置偏的太远，那么对话框会说，”远的没边了！“通过这样的方式，玩家不再只看到冷冰冰的游戏结果数字，而是被这种情感化人性化的反馈所吸引。

小练习：

想想看怎么来实现上面的想法？我们应该把相关的逻辑放在哪里？以及应该如何编码呢？提示：显然我们会用到一大堆if。

最适合放上述逻辑的地方就是showAlert()，因为我们就是在那里创建了UIAlertViewController。之前我们已经可以动态显示消息中的文字，现在则需要动态修改标题文字。

回到Xcode,点击ViewController.swift，更改showAlert()方法如下：

```
    @IBAction func showAlert(){

        let difference = abs(targetValue - currentValue)
        let points = 100 - difference
        score += points

        //添加以下代码对玩家的表现作出评价~
        let title: String
        if difference == 0 {
            title = "运气逆天！赶紧去买注彩票吧！"
```

```
        }else if difference < 5 {
            title = "太棒了！差一点就到了！"
        }else if difference < 10 {
            title = "很不错！继续努力！"
        }else {
            title = "差太远了，君在长江头，我在长江尾~"
        }


        let mesage = "大家好，给大家介绍一下我的得分，是 \(points) 分"

        let alert = UIAlertController(title: title,
                                      message: mesage,
                                      preferredStyle: .alert)
        let action = UIAlertAction(title:"ok",style: .default,handler: nil)
        alert.addAction(action)

        present(alert, animated: true, completion: nil)

        startNewRound()
    }
```

在上面的代码中，我们创建了一个新的名为title的本地字符串类型变量，用它来保存提示对话框中要显示的文字标题。

为了决定我们该使用哪个标题文字，需要查看滑动条结点所在位置和目标数值之间的差异。

如果差值为0，说明玩家运气太好了，我们会在title里面提醒他赶紧去买双色球。
如果差值小于5，说明玩家运气很不错，赶紧求抱大腿。
如果差值小于10，说明玩家水平过得去~
如果差值大于等于10，说明玩家根本没有愉快的在玩耍~

上面的逻辑还能看得懂吗？我们用了一大堆的if条件语句，其中考虑到各种不同的可能性，并选择对应的字符串。

当我们创建UIAlertController对象的时候，会把title变量的内容给它，而不是直接提供一个固定的文本内容。

常量的初始化

在以上的代码中，我们可以看到对于title这个变量，我们使用了显式的声明方式。

let title: String

那么问题来了，之前我们提到过Swift中提供了类型推断的特性，为何这里不使用呢?
此外，你可能还注意到了title是一个常量，但是在以上代码中我们在if语句中多次使用到它，这又是怎么回事呢?

为了回答这个问题，我们需要了解常量（更直白的说是使用let声明的变量）是如何在Swift中初始化的。
当然，你可能会想到使用类型推断的方式来声明title变量，比如使用以下语句：

let title = “”

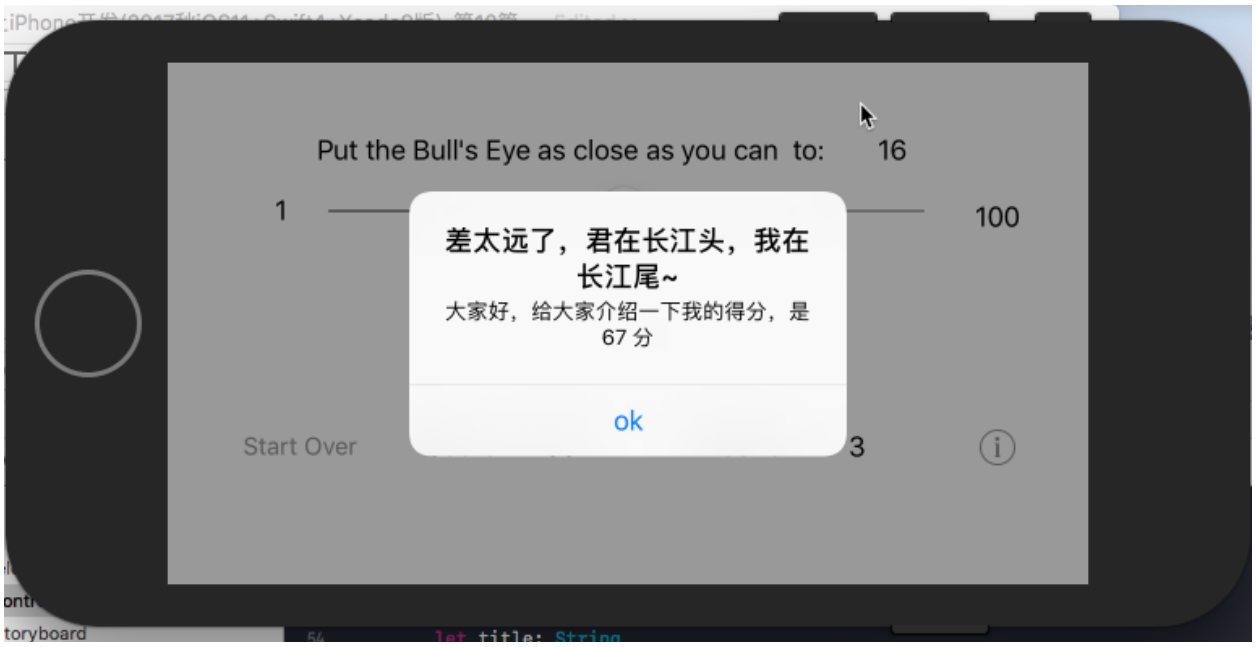
但是这样就会出现一个问题，因为如果使用以上代码，相当于为title指定了一个值，而同时它又是一个常量，因此后面我们将无法更改其值。所以，如果这样写，后面if条件语句中对title的设置就会导致编译器报错。

当然，对于这一类问题，有个非常简单的方式，那就是使用变量而非常量，比如使用以下语句：

var title = “”

这种方法当然可以解决问题，编译器不会再报错，一切恢复正常。但是扪心自问~你真的需要一个变量吗？我个人的习惯是，除非有必要，那么尽量使用常量，以避免一些意外情况的发生。比如某个猪队友一不小心就修改了相关的代码，让某个本来不会更改数值的变量重新设置数值。
当然，这个仅仅是个人偏好，是否这样做完全取决于你自己的习惯。
好了继续~ 如果我们将title设置为常量，那么下面if语句中的代码为何还能给其赋予多个数值？事实上是，因为if条件语句的限制，title最终只可能有一种数值。

好了，现在点击Run看看效果。



小练习：

当玩家表现完美的时候额外奖励100分。

游戏中我们常常会使用一些奖励机制。

比如在这里，当玩家有完美表现时，我们可以奖励100分。这样玩家就有动力继续去达到完美。否则，98分，95分和100分就没有任何区别了。通过这个小小的奖励，玩家就有了去体验Hard模式的动力。一个完美表现的得分不再仅仅是100分，而是200分。当然，我们还可以给99分的玩家奖励50分。

好了，回到Xcode，修改showAlert()方法的代码如下：

```
@IBAction func showAlert(){

    let difference = abs(targetValue - currentValue)
    var points = 100 - difference
```

```
        //添加以下代码对玩家的表现作出评价~
        let title: String
//        let title = ""
        if difference == 0 {
            title = "运气逆天！赶紧去买注彩票吧！"
            points += 100
        }else if difference < 5 {
            title = "太棒了！差一点就到了！"
            if(difference == 1){
                points += 50
            }
        }else if difference < 10 {
            title = "很不错！继续努力！"
        }else {
            title = "差太远了，君在长江头，我在长江尾~"
        }

        score += points
        let mesage = "大家好，给大家介绍一下我的得分，是 \(points) 分"

        let alert = UIAlertController(title: title,
                                      message: mesage,
                                      preferredStyle: .alert)
        let action = UIAlertAction(title:"ok",style: .default,handler: nil)
        alert.addAction(action)

        present(alert, animated: true, completion: nil)

        startNewRound()
    }
```

你应该注意到了几点不同：

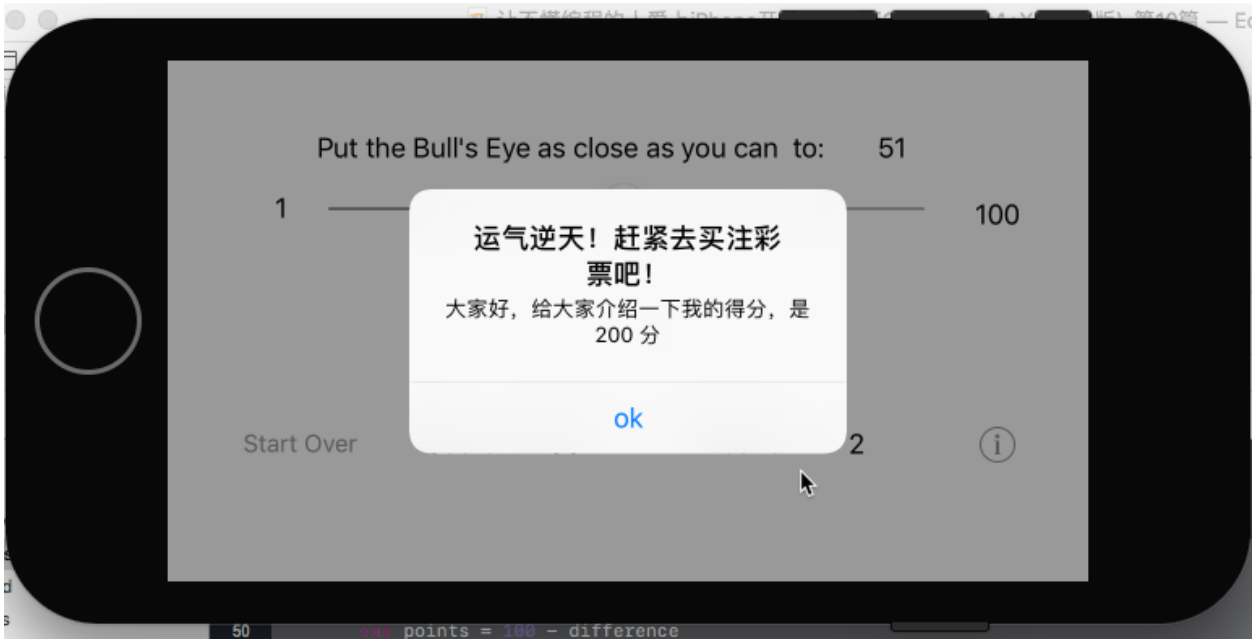
• 在第一个if条件语句中，我们添加了一行新的语句。
当差值为0的时候，我们不仅让玩家得到”完美表现！“的反馈，还额外奖励100分。

• 第二个if条件语句的内容也发生了变化：

这里我们学到了一个新东西，也就是在if条件语句中还可以内嵌if条件语句。如果差值大于0且小于5，就有可能是1（当然也可能不是1）。此时我们会检查差值是否为1，如果是，就额外奖励玩家50分。

- 考虑到这些if判断语句，points不可能再是一个常量，因此需要将let更改为var。
- 最后，我把score += points 这行代码移到了if条件语句结束之后。这一点是必须的，因为在if条件语句执行的过程中我们很可能会更新points变量的数值。

点击Run运行一下应用，看看效果吧！



这里需要再次指出本地变量和实例变量之间的差别。如你所知，一个local variable（本地变量）只在所定义的方法中生存，而一个instance variable(实例变量）的寿命则与视图控制器同齐。这一点对于常量也是相同的。

在showAlert()方法中，一共有5个本地变量和3个实例变量。

let difference = abs(targetValue - currentValue)
var points = 100 - difference
let title = …
score += points
let message = . . .
let alert = . . . let action = . . .

小练习：

指出showAlert()方法中哪些是实例变量，哪些是本地变量。在本地变量中，哪些是变量，哪些是常量?

答案：

本地变量其实很容易分辨，因为它们的定义都是在方法体内部，在前面有let或者var：

```
let difference = . . .
var points = . . .
let title = …
let message = . . .
let alert = . . .
let action = . . .
```

以上6个变量-difference,points,message,alert和action都被限定在showAlert()方法的内部，因此无法在方法外使用。只要方法调用完成，这些本地变量就会消失。

你可能会想，既然difference是个常量（我们使用let前缀），而每次用户触碰按钮的时候都会产生一共新的数值。但常量的数值不是不能更改的吗？这该如何解释?

好吧，这里是答案：每次调用一个方法的时候，里面的本地变量和常量都会重新创建。之前所保存的数值都会被遗忘，我们得到的是全新的数值。

当调用showAlert()方法时，会创建一个全新的difference实例，虽然它和之前的那个同名，但彼此却没有任何关系。这个特殊的常量会活到showAlert()方法结束的时候，然后再次被遗忘。

而随后再次调用showAlert()方法时，又会创建一个新的difference变量，如此循环往复。。。

这让我想到了科幻小说里面的克隆人。在某个时空中，星际移民管理局使用克隆人去充当敢死队开疆拓土。每当需要去一个新的星球探险时，就会将具备某种独特能力的人的克隆体复活，用曲速引擎送到这个星球，往往是去当炮灰。而当这个星球会攻克之后，这些克隆体要吗战死，要吗老死，总之星际移民局是不会再理会。克隆体的生存从此和这个新星球的生存息息相关。而当人类需要再次开辟新的领地时，又会制造出一批新的克隆体，如此循环往复。。。

不过在showAlert()方法执行的过程中，difference的数值却不能发生任何变化，唯一可变的是points，因为它使用var来定义。

换句话说，为了保障军队秩序，星际移民局不会在同一个星球投放两个完全相同的克隆体。

实例变量和本地变量相反，是在方法之外定义的。通常我们在class这行代码之后定义实例变量：

```
class ViewController: UIViewController {
    var currentValue = 0
    var targetValue = 0
    var score = 0
    var round = 0
```

我们可以在任何方法中使用这些变量，而无需再次声明。

不过如果你写出了下面的这段代码：

```
@IBAction func showAlert() {
    let difference = abs(targetValue - currentValue)
    var points = 100 - difference
    var score = score + points  //会出错~
...
}
```

显然你不会得到想要的结果。因为你在score的前面又添加了一个关键词var，相当于定义了一个新的本地变量，而这个本地变量只能在当前方法中使用。

也就是说，如果这样做的话score变量中不会添加points，实例变量score的数值不会改变，即便它们用了相同的名字。

显然这不是你想要的结果。不过好在这样做的话Xcode不会放过你的。

注意：

为了区分本地变量和实例变量，有些程序猿习惯在实例变量的前面添加一个下划线前缀。也就是说，它们会用_score这种形式。这个只是个习惯问题，也有些程序猿喜欢用m(代表member成员）或是f(代表field域）。有些甚至会在变量名称后面添加下划线。

对于这种行为，我个人表示没必要，看自己喜欢了。

等待提示对话框消失！

如果你仔细观察的话，可能会发现一个小小的不爽之处。

每次触碰按钮的时候，提示对话框都会显示出来，而滑动条就会迅速回到中央位置，游戏回合数会增加，而目标数值标签已经获取了新的随机数。

也就是说，当我们还在查看上次游戏的结果时，实际上程序已经开启了新一轮的游戏。有时候哥真是困惑不解。

如果我们可以调整一下游戏逻辑机制，也就是当玩家关闭了提示对话框之后才开启新一轮的游戏，岂不是更合理?

直到玩家关闭了提示对话框，当前的游戏回合才可以真正结束。或许你要说，我们不正是这样做的吗？比如在showAlert()方法里，我们在提示对话框显示完毕之后才会调用startNewRound()方法。

```
@IBAction func showAlert() {
...
let alert = UIAlertController(. . .)
let action = UIAlertAction(. . .)
alert.addAction(action)

// 使用以下代码呈现提示对话框
present(alert, animated: true, completion: nil)

// 开启新的游戏回合
startNewRound()
```

好吧，这里哥需要稍微解释一下原因了。和其它的应用平台不同，在iOS中，提示对话框在显示的时候并没有让当前方法的其它代码停止运行。

事实上，present(alert, animated: true, completion: nil)
这行代码只是让对话框显示在屏幕上，然后就迅速返回，此时showAlert()方法的其它代码会继续顺利执行下去，而不会等待提示对话框的关闭。

用程序猿的术语，在iOS里面提示对话框的运行是异步(asynchronously)的。关于同步和异步我们后面会详细解释，这里你只需要明白一点点常识。

在同步运行的情况下，其它的代码必须等待当前的代码完成后才能继续，而在异步运行的情况下，其它的代码无需等待当前的代码完成就可以继续执行。

比如某些基于网络的应用需要在后台通过网络上传或发送信息（邮件，照片），考虑到网络传输速度的问题，我们不可能等待这个过程结束后才能进行其它的界面交互，否则用户早就疯掉了。

在这里，事实上我们无法知道提示对话框何时结束，唯一可以确定的是当showAlert方法结束的时候它肯定会结束。

好吧，这样一来我们怎么可能知道showAlert()什么时候会消失呢，我们怎么来等它关闭呢?

答案很简单：使用事件！正如我们已经了解到的，在iOS的开发中，很多代码都需要等待特定的事件发生（按钮被触碰，滑动条被移动，等等）。这里同样如此。我们需要等待某个“让提示框消失”的事件发生。在此之前，我们什么也不做。

下面就是它的工作原理：

对提示框上的每个按钮，我们都需要提供一个UIAlertAction对象。这个对象会告诉提示框按钮上的文字内容，以及它的外观显示（当然这里我们使用了默认的样式）：

let action = UIAlertAction(title: "OK", style: .Default, handler: nil)

这里的第三个参数handler告诉提示框对象当按钮被触碰时应该发生什么事情。这就是我们所需要的”让提示框消失“事件。

当前的handler时nil，意味着什么也不会发生。因此我们需要向UIAlertAction提供一些源代码，以便当按钮被触碰的时候运行。

当玩家触碰了提示对话框上的按钮之后，它就会将自己从屏幕上删除，并发送给我们一条消息。这就是我们的线索。

这就是传说中的callback（回调）模式。在iOS中我们有几种方式来实现这一模式。通常我们需要为此创建一个新的方法以处理事件，不过这里可以用一个新东西来完成：closure。

更改showAlert()的方法如下：

```
@IBAction func showAlert(){

    let difference = abs(targetValue - currentValue)
    var points = 100 - difference
```

```
        //添加以下代码对玩家的表现作出评价~
        let title: String
//        let title = ""
        if difference == 0 {
            title = "运气逆天！赶紧去买注彩票吧！"
            points += 100
        }else if difference < 5 {
            title = "太棒了！差一点就到了！"
            if(difference == 1){
                points += 50
            }
        }else if difference < 10 {
            title = "很不错！继续努力！"
        }else {
            title = "差太远了，君在长江头，我在长江尾~"
        }

        score += points
        let mesage = "大家好，给大家介绍一下我的得分，是 \(points) 分"

        let alert = UIAlertController(title: title,
                                      message: mesage,
                                      preferredStyle: .alert)
        let action = UIAlertAction(title:"ok",style: .default,handler: {action in
                                                        self.startNewRound()
                                                    })
        alert.addAction(action)

        present(alert, animated: true, completion: nil)

    }
```

这里我们做了两件事情：

1.删除了方法底部对startNewRound()方法的直接调用。

2.把它们放到一个块语句中，作为UIAlertAction的handler参数。

这种块语句被成为closure（闭包）。

点击Run，再次运行一下游戏，现在就好多了。

科普-Self

刚才在所谓的闭包块语句中我们使用self.startNewRound()，而不是单纯的startNewRound()。

这里的self是个神马东西呢？self单词当然是自己，自我的意思。因此这个关键词其实是让视图控制器好称呼自己。

对于自我的称呼，古往今来有很多说法，可以自称”寡人“，”朕“，”本人“，”草民“，”本官“，”小女子”，“本屌”。。。

这里的self就是程序世界生命体对自己的叫法，不服不行。

通常情况下我们无需使用self 向视图控制器自身发送消息，即便你这样做了编译器不会冲你咆哮。不过在这里的闭包(closure)中我们必须使用self来代表视图控制器。

这就是Swift世界的生存法则。如果你忘了闭包中的self,Xcode自然不会放过你，不信你大可以自己试试看。之所以有这个法则，是因为闭包可以“capture(捕获）”变量，从而带来一些奇怪的副作用。在后面的教程中我们会了解到这一点的。

Ok，今天的学习就到此结束了。还是送福利1张。