《嵌入式开发技术课程报告》

成 员 白云鹏 1004161221

成 员 高庆胜 1004161230

专 业 计算机科学与技术

班 级 10041612

目录

概述

本项目通过对 UIKit 基本组件的使用，通过纯代码方式完成了基本图形的绘制，在此基础上完成了一个基本 2048 游戏的一个搭建。在基本游戏内容外，还增加了一些额外的小功能比如排行，回退等操作来加强对 Swift 语言的熟悉程度及其开发环境的使用。

1 功能说明

1.基本游戏功能

首先，最重要的过程莫过于其基本的游戏功能，其游戏方法即为将中间无障碍的相同色块合并，每一次合并或者是移动之后都会产生新的色块，直到达到 2048 或者是不能继续进行游戏，那么就停止游戏，进行新的游戏，可以简单的看出，每一次产生新的色块，其值只为 2，那么最少都需要 1024 步才能结束赢得游戏。

2.游戏辅助功能

游戏辅助功能主要有：

- 撤销功能
- 计时功能
- 重置功能
- 更改主题功能
- 提示用户输入昵称
- 点击消除色块

撤销功能就是用户可以进行回退操作，如果当前操作不是用户所期待的操作，那么就会在点击撤销后，整个游戏的状态就会恢复到上一次操作的状态。计时功能即为用户当前的游戏时长，用户可以通过额外的按钮来进行计时器的暂停以及

继续计时。重置功能即重新开始新游戏，当前游戏状态不会保存。用户可以通过更改主题来更改背景或者是色块的颜色。用户在请求进入排行榜时，程序会首先要求用户进行昵称的输入，此时会弹出对话框来要求用户输入昵称。点击消除色块即用户可以随意点击某一个色块就可以消除色块。

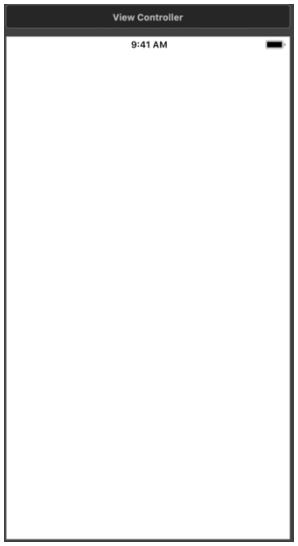
3.排名功能

用户可以通过点击按钮来进行排行榜的进入，用户在提交昵称之后，系统将会将当前的分数与排行榜中的分数进行比较并将其插入到适当的位置，如果该玩家刷新了自己的分数，那么就更新它自己的分数。

2 界面设计说明

2.1 故事板(storyboard)

在学习的过程中，我们学习了故事板(storyboard)的使用。故事板确实是一个很方便的工具，通过对控件的拖拽以及放置，就可以很简单的完成控件的初始化以及其相关联操作的设置。但是，简便背后所带来的坏处自然也就是可控性的降低以及突发情况的增多，而且在设计比较复杂的场景时也会比较困难。因此，很多在 Github 上的项目都没有使用到故事板，而且在上课的过程中，更多的也是强调了故事板的使用，因此，在进行设计的时候，也尝试舍弃了故事板，采用纯



代码对组件进行声明、布局设置以及操作设置等等。

2.2 启动界面（ViewController 类）

在启动界面中，首先四个数字是由四个 UILabel 组成的，下面的三个色块则由三个 UIButton 组成，二者的初始化接口相同，均使用 init(frame : CGRect)，通过传递其左上角的坐标以及其长宽构成的矩形，就可以简单的完成对其位置的设置。最下层的空白界面即为 view 视图。通过设置各个组件的 text、textColror 以及 backgroundColor 等属性，可以很方便的完成基本视图的一个绘制。当然，绘制图形仅仅只是很简单的一个操作，因此，在欢迎界面上，我们还加入了一些动画元素在其中，但是由于没有使用第三方库，因此动画也只完成了很简单的平移以及缩放动画。这个动画则是通过定时器的简单使用完成的。其三个不同颜色的 button，其功能也已展示在其 title 上，分别是开始游戏，展示排名以及更改颜色。

图 2 启动界面

2.3 游戏界面(game 类)

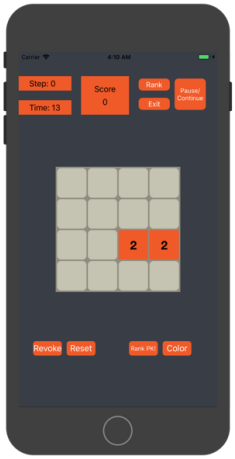


图 3 游戏界面

在游戏界面的设计中，按照从 的顺序，首先左上角的为两个 UILabel，其功能分别是记录当前游戏的时间以及游戏的步数，右边的 UILabel 则为当前游戏的分数，其是在每一步操作结束后将当前棋盘上所有的数字相加作为其值，右边三个为三个 UIButton，分别为展示排名，退出游戏（回到界面）以及暂停/继续计时。下方的主要区域即为游戏区域。最后下方的四个矩形则分别为撤销、重置、申请进入排名以及更改颜色。用户点击申请排行榜时，程序会弹出提示框请求用户输入昵称：

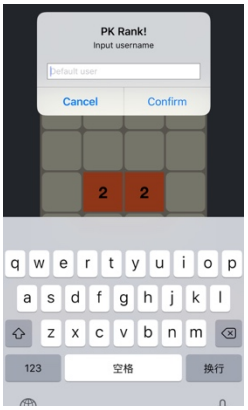


图 4 请求用户输入昵称提示框

在用户输入结束后，点击 confirm 键，程序会自动对排行榜中元素进行处理。

2.4 排行榜界面（rankControl 类）

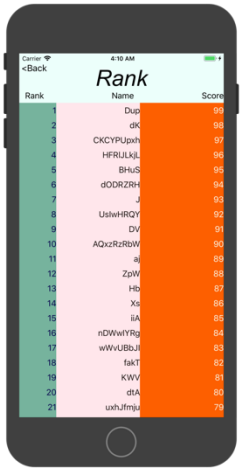


图 5 排行榜界面类

在排行榜界面中，仍旧按照从上到下，从左往右的顺序进行说明。首先最上方是一个 UIButton，其作用是调用 dismiss 函数进行界面的返回，接下来时三个 UIButton 用来表征界面的元素信息，紧接下来的元素使用了 UIScrollView，在其中，每一行则包含了三个 UILabel，分别表征排名、昵称以及得分。通过点击三个 UIButton，可以分别切换其对应元素的升序/降序排列。在 UIScrollView 中进行滑动，可以浏览整个排行榜，排行榜之保留了前 100 名。大于 100 名的将会被自动舍弃。

图 6 排行榜的滚动以及不同的排序方式

2.5 颜色更改界面(changeColor 类)

该类的主要作用就是更改全局的背景颜色或者是单个色块的颜色，用户通过数字代码指定更改的类别 (0 代表背景颜色，2 代表色块 2，4 代表色块 4 等等)，然后通过指定颜色的 RGB 值来确定颜色。

图 7 更改颜色界面

在这个界面中，主体部分由四组形式统一的组件构成，每一组都有一个 UILabel、一个 UITextField 以及一个用来预览当前颜色的 UILabel 组成，在其背景处还有一个透明的 UIButton 用来将各个 UITextField 的第一响应者释放，也就是释放键盘并且同时更新颜色。在输入颜色结束后，点击 Confirm 就可以更新颜色，而点击 Cancel 则放弃更新。

3 连接说明

由于程序时完全以代码进行实现的，因此在故事板的连接上，并没有输出口以及相关操作的设置，取而代之的是全部的代码实现。

3.1 viewController 之间的跳转

viewController 之间的跳转也就是场景的跳转，即使在没有故事板的帮助下，

场景的跳转也十分方便：

```
present(viewController, animated: true, completion: nil)
```

这个函数第一个参数接收一个 viewController 类的参数，这个就表示要跳转的场景，在调用这个函数之后，界面会自动调用 viewController 的 viewDidLoad() 的方法进行重绘。因此，要想进行场景的切换，只需要声明一个场景的变量，然后调用该函数即可。

```
@objc func changecolor(_ sender: Any) {
    let color = colorChange()
    self.present(color, animated: true, completion: nil)
}
@objc func showrank(_ sender: Any) {
    let rank = rankControl()
    self.present(rank,animated: true, completion: nil)
}
@objc func tet(_ sender: Any) {
    let gm = game(maxNumber: 2048)
    self.present(gm , animated:true , completion: nil)
}
```

上述三者即为欢迎界面的三个按钮点击时进行的场景切换。

退出场景的方式也比较简单：

```
dismiss(animated: true, completion: nil)
```

直接调用该函数即可完成返回。

3.2 按钮触发事件的设置

UIButton 触发事件的设置也很简单，主要是使用了其 addtarget 方法

```
UIButton.addTarget(target, action:#selector, for: event)
```

其中，第二个参数就指定了其处理函数，第三个参数则指定了其触发事件。通常的处理函数声明为：

```
@objc func funcname(_ sender: UIButton) {}
```

而对于 UIButton 其触发事件通常为：

```
UIButton.Event.touchUpInside
```

简写为:

```
.touchUpInside
```

由此就可以很简单的为 UIButton 进行触发操作的设置。

3.3 部分控件连接说明

相同控件的连接都有大同小异之处，因此

- UIButton 的连接：

```
let sg = UIButton.init(frame: CGRect(x: 140, y: 300, width: 120,
height:50))//使用一个矩形框来对 UIButton 进行初始化
sg.backgroundColor = UIColor(red:255.0/255.0, green: 233/255.0, blue:
138/255.0, alpha :1.0)//设置 UIButton 的背景颜色
sg.setTitleColor(UIColor(red:15/255.0, green: 10/255.0, blue: 60/255.0,
alpha: 1.0), for: UIControl.State())//设置 UIButton 上方的字体颜色
sg.addTarget(self, action: #selector(tet(_ : )), for: .touchUpInside)//添加触
发响应事件
sg.setTitle("Start Game", for: UIControl.State())//设置 button 上的字
view.addSubview(sg)//将 UIButton 中添加到视图中进行显示
```

- UILabel 的连接：

```
var label2 = UILabel(frame:CGRect(x: 50, y: -150, width: 100,height:100))//
使用矩形框初始化 label
label2.text = "2"//设置 label 的字
label2.font = UIFont(name: "Helvetica", size: 70)//设置 label 的字体
label2.textColor = .white//设置 label 字的颜色
self.view.addSubview(label2)//加入到视图中进行显示
```

- UITextField 的连接：

```
let rtextfield = UITextField(frame: CGRect(x:155, y: 240,
width:100 ,height:30))//使用矩形框对文本输入框进行初始化
rtextfield.backgroundColor = .white //设置背景颜色
rtextfield.placeholder = "57"//设置占位符
rtextfield.layer.cornerRadius = 6 //设置圆角半径
rtextfield.keyboardType = .numberPad //设置键盘类型
```

```
view.addSubview(rtextfield) //加入到视图中进行显示
view.bringSubviewToFront(rtextfield)//将该视图放置到最顶层
```

- UIScrollView 的连接：

```
scroll = UIScrollView(frame: CGRect(x:0, y:title.bounds.height +
rktitle.bounds.height + 10, width: view.bounds.width, height:
view.bounds.height - rktitle.bounds.height - title.bounds.height))//仍然是使用
矩形框对 UIScrollView 进行初始化
scroll.contentSize.height = CGFloat( ranklist.ranklist.count * 30)//设置内容高
度
scroll.contentSize.width = view.bounds.width//设置内容宽度
scroll.addSubview(label) //增加显示内容
scroll.isScrollEnabled = true //允许滚动
scroll.showsVerticalScrollIndicator = false //不显示水平滚动条
scroll.showsHorizontalScrollIndicator = true//不显示垂直滚动条
```

4 场景切换说明

4.1 启动界面（ViewController 类）

在ViewController类中，主要实现的就是对欢迎界面的一个设置，在欢迎界面中，通过点击三个按钮可以进行三个场景的跳转，即跳转到游戏界面、排名界面以及更改颜色界面三个场景。

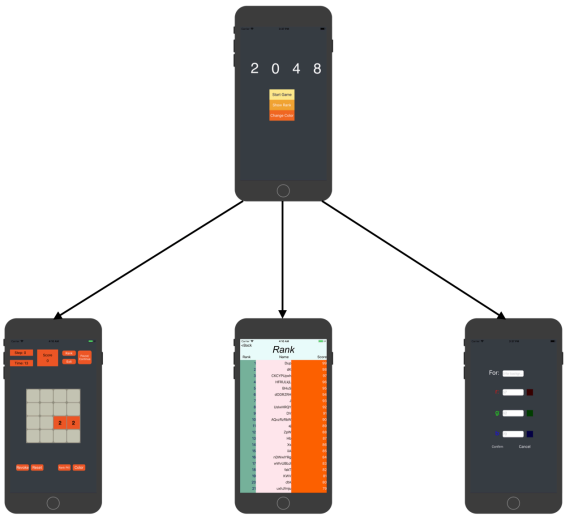


图 8 欢迎界面的跳转

4.2 游戏界面（game 类）

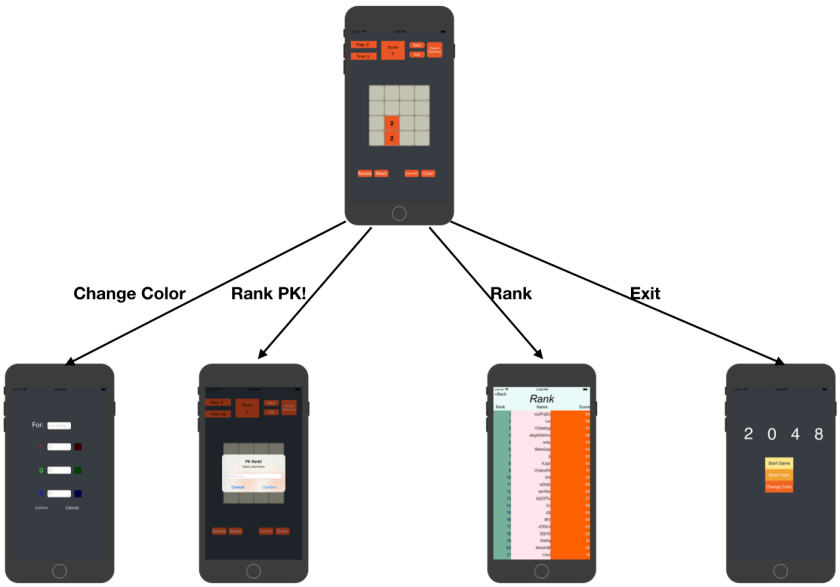


图 9 游戏界面类界面跳转

在游戏界面中，同样的也可以点击游戏中的各个按钮进行不同界面的跳转。

4.3 排行榜界面（rankControl 类）

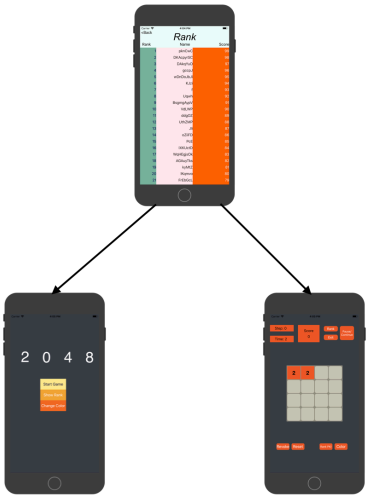


图 10 排行榜界面的跳转

在排行榜界面上，通过其左上角的按钮可以返回上一层界面，如果是从游戏界面进入排行榜，那么回退的时候就会回退到游戏界面，相同的，如果是从游戏界面进入排行榜，那么回退的时候会回退到游戏界面。

4.4 颜色更改界面（ColorChange 类）

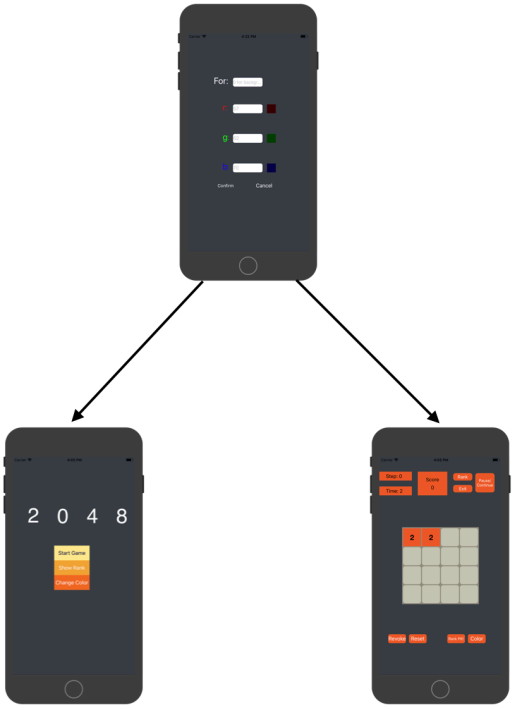


图 11 颜色更改界面的跳转

同样的，如果从开始界面进入到更改颜色界面，那么回退的时候也会回退到开始界面，从游戏界面进入更改颜色界面，则会回退到游戏界面，但是游戏时长不会暂停，仍旧是继续进行游戏。

4.5 提醒用户输入昵称界面（UIAlertController 类）

在该界面中，只有一个弹出框，要求用户输入游戏昵称，用户在输入之后可以点击 Confirm 递交申请或者是直接点击 Cancel 取消申请，在递交申请之后，程序会自动处理其排行。

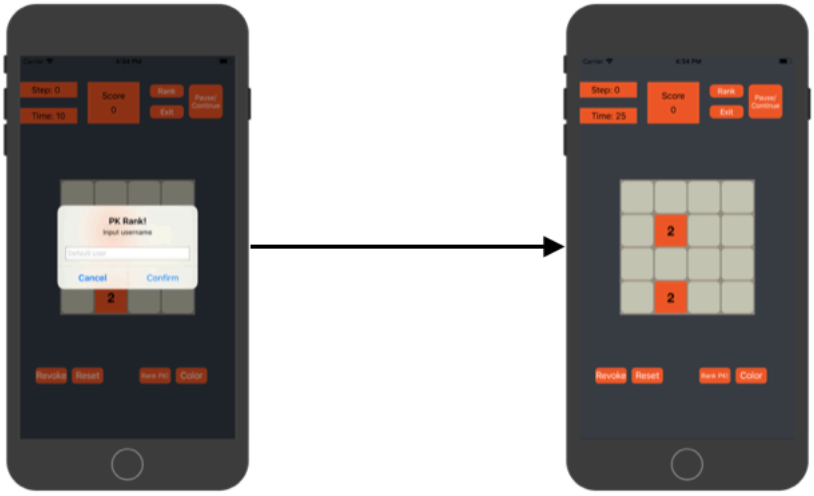


图 12 请求用户输入昵称界面的跳转

5 应用逻辑说明

5.1 游戏主流程

首先对于游戏界面，为其创建了四个滑动事件监听器分别处理上下左右四个方向的滑动事件。每当用户进行操作，相应的滑动事件监听器就会调用处理函数进行处理，由于所有的色块一共只有 16 个，并且每次只处理一个方向的话，即便是最朴素的$\Theta(n^2) = \Theta((4-1)^2)$求法，其每次遍历最多也只有 9 个色块，因此其时间复杂度并不是很高。

1. 在处理之前，首先需要将当前场景进行一个入队操作，这个队列是一个三维数组:

```
var pre : [[[Int]]] = []
```

这个数组用来保存当前状态，如果需要进行撤销操作，那么只需要从 pre 中直接取出最末端元素进行回滚，然后再进行重新绘制，即完成了回退操作。

2. 在入队操作结束之后，就需要对每一个色块进行判断，如果这个色块所在位置的数字为 0，那么就需要将离它最近的不为 0 的色块放在当前位置。如果这个色块所在位置不为 0，那么就需要判断离它最近不为 0 的色块其数字是否与当前色块相同，如果相同，那么就需要进行合并操作，

如果不同，那么就停止遍历。 如果能够保障其按照数组的顺序进行遍历，那么就能保障在处理每一个色块的时候都只需要处理其以后方向的色块。

3. 在处理结束之后，还需要将色块进行移动，这里直接使用了：

```
UIView.animate(withDuration:, delay: ,  animations: , completion: )
```

直接进行色块的移动。

4. 当然，在最后的处理中，如果当前操作是一个无效操作，即非 0 色块不能进行移动或者合并，那么就需要将维护队列中放入的最后一个状态弹出。如果已经到达了要求的分数，游戏结束。如果四个方向均是无效操作，则说明已经无法进行，游戏结束。如果当前操作是有效操作，那么就需要更新当前游戏区域的状态，移除重复的色块并进行色块数字的更新以及游戏步数和分数的重新计算。

5.2 排行榜流程

当用户进行排行榜的查看的时候，

1. 首先会进行一个滚动视图的初始化，初始化的过程也很显然，即从已有的列表中逐一取出所有的元素。
2. 然后再按设置好的长宽高不断对其进行设置。
3. 最后将其加入到滚动视图中进行显示。

如下即为三个 Label 位置的设置

```
let rk = UILabel(frame: CGRect(x: CGFloat(0), y: CGFloat( CGFloat(i) *
height ), width: CGFloat(75), height:CGFloat(30)))
let t = UILabel(frame: CGRect(x: rk.bounds.width, y: CGFloat( CGFloat(i) *
height ), width: wwidth / CGFloat(2), height:CGFloat(30)))
let t2 = UILabel(frame: CGRect(x: rk.bounds.width + t.bounds.width, y:
CGFloat( CGFloat(i) * height), width: wwidth / CGFloat(2),
height:CGFloat(30)))
```

在 rankControl 类中设置了三个布尔类型的标志，用来表征当前是否已经按照其顺序进行排序，在用户申请更改排行榜的顺序的时候首先会读取对应的标志位，然后根据其标志位对列表元素进行重新排序，再排完顺序之后，由于其视图没有刷新，因此需要将所有元素从其父视图中移除，这里用到：

```
removeFromSuperview()
```

来从其父视图移除，移除结束后就需要重新对其元素进行添加。

如果用户请求进入排行榜，那么程序首先会遍历所有的元素，如果当前用户刷新了自己的分数，那么在更新分数结束后直接返回，否则就需要将其插入合适的位置中。这里是用了拷贝的方式进行插入的，其时间复杂度为$\Theta(2n)$，但是由于序列长度最长只有 100，因此时间复杂的开销差距不是很大。其空间复杂度为$\Theta(n)$。总体来说，由于整个程序的规模并不是很大，因此同阶复杂度的情况下，其差距聊胜于无。

在排行榜的处理中，便于测试，增加了随机生成数据的过程，其分数为 0-99，其昵称为首先使用

```
    t = Int(arc4random() % 10)
```

来获得其昵称长度，然后每一位都使用随机数生成下标，将下标在：

```
var strs =
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ"
```

中所对应的字符作为其名字的下一个字符，由此就生成了最终的昵称。

5.3 游戏辅助流程

- 更改颜色流程

该过程比较简单，在用户点击更改颜色的按钮后，弹出更改颜色的界面，然后用户点击输入框之后进行输入，当用户输入完毕后，点击空白区域的隐藏 button 即可释放输入框的第一相应者身份，并且同时可以将用户输入的颜色进行反馈，显示出一个预览效果。在用户完成更改后，不论是确认或者是取消，都会回到上一层界面。

- 计时功能

计时功能则是在进入游戏区域的时候初始化一个计时器，然后让这个计时器每一秒都对游戏时间进行一个更新即可。当然，如果只进行时间的更新是无法完成绘制图形的更新的，因此在每一次更新结束后，都需要将其 view 移除重新添加才能正确显示。

- 记步功能

记步功能也和记时功能类似，但是是在每一次有效操作结束后才对其进行更新，在步数更新结束后，重新对其元素进行添加。

- 撤销功能

撤销功能则用到了前面声明过的 pre 数组，在用户点击撤销键的时候，从 pre 数组中取出最后一个元素（如果有），将其赋值给当前的状态数组，然后重新对游戏区域进行绘制，就完成了撤销功能。

- 重置功能

重置功能的实现则更加简单，重新调用当前 view 的 init（）函数或者是展示入口函数，就能完成对整个游戏的一个重新初始化。

- 点击消除色块功能

这个功能也是通过为每一个色块都放置一个 UIButton 来实现的，在实现的过程中，在实现的时候，由于每一次的操作都会导致色块的重新添加，但是添加的过程中，越晚添加的元素将会置于最顶端，由此就会导致 UIButton 的失效，因此就需要重新将 UIButton 提升到最前端，同样的，通过:

```
view.bringSubviewToFront(UIView: )
```

来将元素提高到最顶层。

用户在点击 Button 之后，会将其在状态数组中的数字清 0，然后对其进行一个重新绘制，绘制结束后，再将所有的 UIButton 提高到最顶层，这样才不会影响下一次的点击操作。

6 相关技术说明

6.1 动画

- 色块的移动动画

色块的移动动画是通过 animate（）函数来实现的:

```
    func xflash(){
        for c in blocks {
            if c.xchange != -1 {
                UIView.animate(withDuration: 0.2, delay: 0,  animations:
{c.center.x -= (CGFloat(c.row - c.xchange)*CGFloat(self.gapWidth +
self.blockWidth) ) }, completion: nil)
            }
```

```
        }
    }
```

其中 animate 的第一个参数为其持续时间，第二个为延迟实现，第三个参数为其所要进行的操作闭包，这里则是将色块的横坐标进行一个平移。由此就完成了基本的平移操作。

- 欢迎界面动画

欢迎界面的移动动画与色块的移动不同，这里使用的是则是一个 Timer 计时器的简单应用：

```
    let st = Timer.scheduledTimer(timeInterval: 0.01, target: self,
selector: #selector(bgrun), userInfo: nil, repeats: true)
    st.fire()//计时器开始计时


    @objc func bgrun(){
            time += 1
            if time < 150 {
                label2.center.y += 2
                self.view.bringSubviewToFront(label2)
            }else if (time < 300) {
                label0.center.y += 2
                self.view.bringSubviewToFront(label0)
            } else if(time < 370) {
                label4.font = UIFont(name: "Helvetica", size: label4size + 1)
                label4size += 1
            } else if (time < 495) {
                label8.center.x -= 2
                self.view.bringSubviewToFront(label0)
            }
    }
```

这里，每一次计时的时候首先都去先增加计数器，然后对计数器进行一个判断，然后对每一个 label 的位置或者大小进行一个控制，这样，连续不断的进行图形的重新绘制，就完成了动画的流程。

6.2 滑动事件监听器的设置

滑动事件监听器则是使用 UISwipeGestureRecognizer 函数来实现的。和为 UIButton 增加很类似，到那时在设置的时候需要额外设置需要的手指数以及其滑动方向，最后将其加入到整个视图中才能完成监听器的注册。以下只说明了一个方向，其余方向类似。

```
let swipeup = UISwipeGestureRecognizer(target:self,
action: #selector(handleup(_: )))
swipeup.numberOfTouchesRequired = 1
swipeup.direction = .up
self.view.addGestureRecognizer(swipeup)
```

6.3 消除按键设置

色块的消除按键一共共有 16 个，如果一个一个的增加，必然是一个非常费时费力的工作，因此，在设置的时候就采用循环来进行设定。但是在设定的过程中，需要指明按键所处的行和列，而在其 selector 传递过程中无法传递其余额外的参数，只能通过 tag 属性来设置额外的消息，因此，在这里将行列进行压缩，即：

```
let bt = UIButton(frame: CGRect(x:xx, y:yy, width: blockWidth, height:
blockWidth))
bt.addTarget(self, action: #selector(hadnleblockbutton(_: )),
for: .touchUpInside)
bt.tag = line * 10 + row
```

将行数乘 10 + 列数，完成压缩，在处理过程中，其解压缩的过程为：

```
let row = sender.tag % 10  //获取列数
let line = sender.tag / 10 //获取行数
nums[line][row] = 0        //清空状态数组
```

由此就完成了按钮的设置。

6.4 滑动视图的设置

在设置滑动视图的时候，只需要将其子 view 直接加入到 UIScorllView 中即可，其子 view 的位移是针对其父 view 的坐标而言的，因此在实际的代码中有：

```
let rk = UILabel(frame: CGRect(x: CGFloat(0), y: CGFloat( CGFloat(i) * height ), width: CGFloat(75), height:CGFloat(30)))
```

实际上起始 rk 的 y 为 0, 但是其视图显示的时候是加上了去父 view 的位移, 因此很方便的就能进行添加操作。

7 分工

姓名	完成任务	占比
白云鹏	主要逻辑的实现，组件的创建以及设置	65%
高庆胜	部分逻辑实现，组件的配色以及内容设置	35%

8 总结感想

8.1 困难以及解决办法

- 首先在各类的初始化方法的实现过程中，需要先对所有的成员进行初始化，然后才能调用 super.init()，否则编译会出错。

解决办法：对每个成员都进行初始化或者是将成员声明为” 类型？”

- UIButton 背景颜色的设置不能通过.titleLabel?.backgroundColor 进行设置，而应该使用.setColor 函数进行更改。
- 各种 view 的 addSubview()遗漏或者其添加顺序出错导致被遮挡，无法显示。
- Swift 4 开发过程中，很多语言接口都已经有所改变，在网上所查阅到的

很多内容都无法使用。

解决办法：查阅官方文档或者其包含属性进行推测。

- 在开发过程中，或多或少的都使用到了 OC 语言，但是对于其语言应用较少，不是很熟练。

解决办法：查阅网上资料根据其已经写好的语句进行改编。

- 由于在开发过程中，没有使用故事板，因此每次对效果进行调试都只能重新进行编译然后进行效果的观察，再对其进行调整。
- 在开发过程中，对于很多接口的复杂性都不了解，因此使用的过程中，可能更侧重于自己手动实现。
- 每次对元素进行更新之后，都需要对画图重新绘制，在开始的时候由于没有重新绘制，导致以为出现了 bug。

8.2 感想

通过这几周对 IOS，主要是 swift 以及 XCode 的使用，在课上是通过简单的控件拖拽完成各个布局，但是在实现的过程中，使用纯代码对其进行控制，因此有很多精力都花费在了控件位置的调试以及布局颜色上，还有样式的重叠搭配，最终导致代码长度也达到了 1200 行。而且在开发的过程中，由于新发布的 Swift 4，很多相关内容在网络上没有找到，只有比较旧的 Swift 3 标准，因此开发效率也很低下，再加上时间的限制，不得不削减掉很多构想中要实现的功能，在新旧标准进行交替的过程中进行开发实属不易。

由于在课上只学过 UI 库很少的一部分组件，而且在以往只有桌面端的开发经验，因此，在开发的时候，有很多的 UI 组件因为不知道而使用不上，很多方便的组件甚至都曾通过更为基础的组件来实现，而且官方文档对于 UIKit 的介绍也很没有像其他库一样，对所有的属性方法都有介绍。因此有很多的组件都没有涉及到。

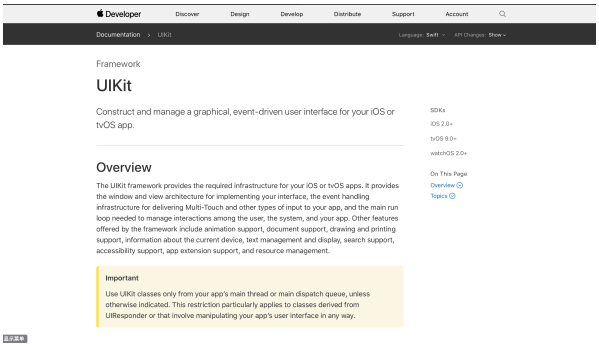


图 13 Apple Developer 对于 UIKit 的介绍

Swift 语言由于其迭代速度一直被开发人员所诟病，在此次开发实践之后，才深有体会，Swift 语言对于其前者的支持实在是很差劲，在将很多已经成熟的代码进行移植的过程中，由于其接口的大改特改，最后导致移植过程中突发事件不断，因此在以后编写代码的过程中，应该要注意自己代码的向后兼容性，使用更多已经广泛使用的接口，这样才能保证程序的鲁棒性。

虽然对 Swift 语言已经有了一些初步的了解，但是最后在选题上，考虑到时间因素的限制，因此开题的时候没有选择特别困难的题目，更加侧重的是对 Swift 语言的一个了解以及熟练运用，因为在以后的工作或者学习过程中，对语言有了一个更深的理解的话，就可以很方便的使用很多已经成熟的第三方库或者 pod 来完成开发。