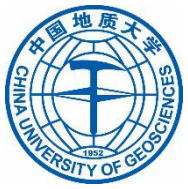


中国地质大学（北京）

《嵌入式开发》

期末课程报告

学　　院：信息工程学院

专　　业：计算机科学与技术

班　　级：10041812、10041811

学　　号：1019181113、1005183121

姓　　名：胡杭天、周子杰

联系方式：13269130118

邮　　箱：1227863273@qq.com

指导老师：管青

日　　期：2020 年 12 月 21 日

目　录

概述

本项目通过对 UIKit 的基本组件的使用，用纯代码的方式完成了一个综合的游戏模式的搭建，游戏有创新性且可玩性强，并且在基本功能之外，还加入了一些另外的小功能，比如加入了显示当前帧数、计分板等功能，并且可以显示历史最高分数，以此来加强我们对 Swift 语言运用的熟练度和综合运用能力。

1 背景介绍

1.1 灵感来源

大声告诉我们，世界第一的 IP 是啥？？？宝可梦!!! 这个游戏的灵感就是来自我们可爱的宝可梦们。

1.2 背景描述

重视亲情的嘎啦嘎啦在母亲死后悲痛欲绝，但是它的旅程却不能止步于此。于是，嘎啦嘎啦将母亲的头骨套在头上，将母亲的骨头当作武器，开始了属于自己的冒险。但是，本游戏最大的反派--千面避役出现了，它使出了极巨化的能力，将尾巴竖成一个高塔企图阻挡嘎啦嘎啦的去路。我们可爱而又可怜的嘎啦嘎啦能否摆脱困境呢？当然，这个就需要作为玩家的我们的实力来决定了。（彩蛋：然而皮卡丘一直默默的看着这一切，你知道它在哪吗？）

2 功能说明

2.1 主要功能

“Pokemon Jump”游戏的设计逻辑是通过点击方式控制我们的主角（嘎啦嘎啦）进行跳跃使得主角能够尽可能躲避所有的障碍到达远方，但是一旦主角碰到障碍那么游戏就结束了，但是幸运的是在这个游戏中我们有无数次的试错机会…

2.2 辅助功能

在“Pokemon Jump”app 中，我们包含了如下辅助功能：

- 显示当前帧数
- 计分板
- 更新最高得分
- 重置功能

显示当前帧数功能，即在游戏界面的右下角的一个小小的区域显示当前的游戏帧数（预示着游戏的稳定性，帧数越高则画面越流畅，稳定性越高）。（如下图所示）

1 帧数显示

计分板功能，即在界面上方显示一个数字，预示着当前的分数值，用来记录玩家已经通过控制主角已经通过了多少个障碍。玩家可以通过计分板来实时地得到自己的分数。

更新最高得分功能，即在结算界面中，除了会显示玩家的获取分数，还可以显示当前设备的玩家历史最高分数记录，当玩家破纪录时更新。

重置功能，在玩家控制主角碰撞到障碍物时，游戏结束，玩家可以选择重置游戏进行下一次全新的游戏。

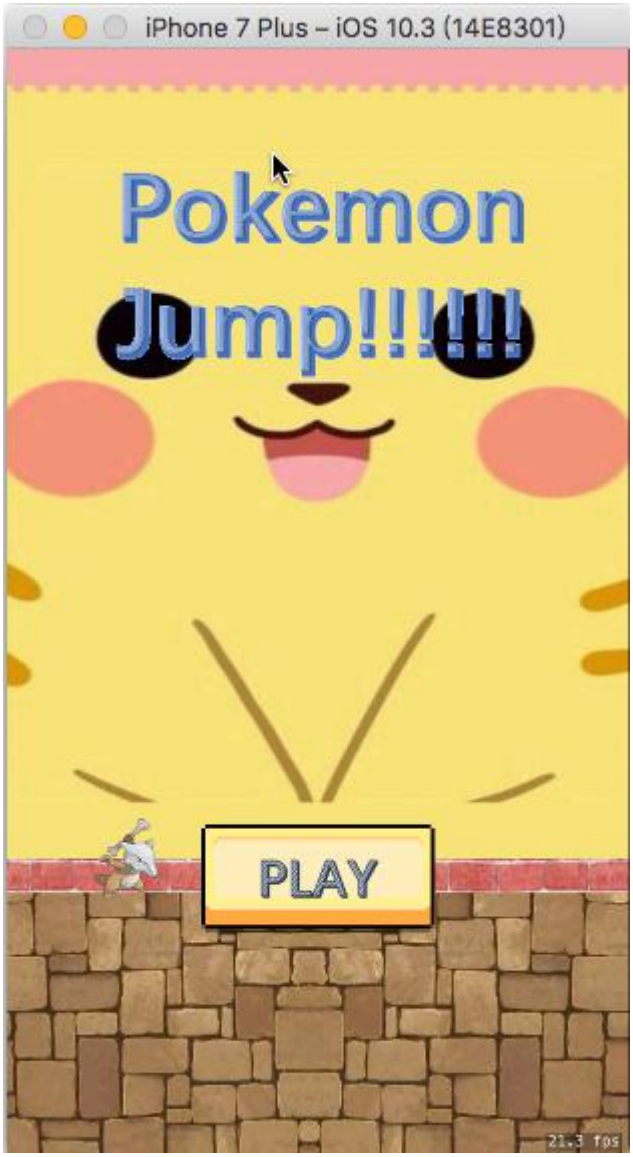
3 界面说明

在本次的综合程序设计中，所有的界面都以代码形式完成。在课上，我们学用到了利用图形界面生成界面的方法，这是一种非常方便的工具，我们可以通过拖曳控件来简单的完成控件初始化和关联操作。但是在复杂场景设计中，使用这种方法会带来可控性降低，比如我们在程序设计过程中由于自己对部分地知识掌握地不够熟练导致反复出现 thread1 等现象，很是烦恼。于是在第 n 次 thread1 之后决定使用代码来生成和控制我们的图形界面，对组件进行声明、布局设置和功能控制等。

3.1 启动游戏界面

在我们的启动界面中，有一个和游戏界面交界的区域，可以看作对正式的游戏场景的一个预处理，即在布置完场景基础元素之后加上一些点缀的按钮作为场景的初始启动界面。

具体界面如下图所示：



2 启动界面图

可以发现在启动界面中，首先由 UILabel 显示游戏名“Pokemon Jump!!!”，在靠近下方的中间，有 PLAY 按钮，属于 UIButton，使用了 init(frame:CGRect)的接口来初始化，通过传递左上角的坐标和长宽构成的矩形，可以简单地完成设置。再通过设置几个组件，分别设置贴图等，来完成一个基本视图的绘制。在欢迎界面上，我们还加入了一些动画元素在其中，是我们的主角（嘎啦嘎啦）的一个微小的位移，以此来吸引玩家开始游戏。

3.2 等待开始界面

等待开始的界面是在第一个启动界面中，玩家点击“Play”按钮之后跳转来到的一个界面。

具体界面如下：

3 准备游戏界面

如图所示，有两个 UILabel，一个显示“Ready”表示游戏已经准备就绪，并且通过第二个 UILabel“Tap to Start!”提示玩家游戏方式（通过点击屏幕来完成交互）。

3.3 游戏界面

在游戏界面中，基本元素如下：障碍物（千面避役），主角（嘎啦嘎啦）、地面、背景、计分板、帧数板。上方的计分板用来显示当前的游戏分数，它的值每当主角通过一个障碍时加一，右下角的帧数板实时地显示当前的帧数值。主角（嘎啦嘎啦）有行走的动画（通过地面图片的左移来实现视觉上的相对运动）。用户点击界面时，会触发主角的跳跃动作（在空中可进行二段跳）。

具体界面如下图所示：

4 游戏界面

3.4 结算界面

结算界面主要由如下几部分组成，一个 UILabel 显示游戏结束“Game Over”，一个 UIButton 用来显示“OK”，单击此按钮可以到达初始启动界面，进行下一轮的新游戏。UITextFiled 用来显示当前的分数和最高分数的面板，用作结算界面。

具体界面如下图所示：

5 结算界面

4 连接说明

4.1 页面跳转

这里列一下主要的逻辑，具体的会在下文场景切换中展开：初始页面→（Play 按钮）→准备页面→（任意地方单机）→游戏页面→（碰到千面避役（障碍物））→结算页面→（OK 按钮）→初始页面→……

4.2 具体连接

背景：

```
let backGround = SKSpriteNode.init(imageNamed: "Background")
backGround.anchorPoint = CGPoint.init(x: 0.5, y: 1.0)  //锚点
backGround.position  = CGPoint.init(x: size.width/2, y: size.height) //
位置
backGround.zPosition = fbLayer.backGround.rawValue //z 轴的原始值
gameNode.addChild(backGround) //添加背景到游戏世界上面
```

Play 按钮：

```
// 开始游戏按钮
let startGameBtn = SKSpriteNode(imageNamed: "Button")
startGameBtn.position = CGPoint(x: size.width * 0.5, y: size.height * 0
.25)
startGameBtn.name = "MainMenu"
startGameBtn.zPosition = fbLayer.UI.rawValue
gameNode.addChild(startGameBtn)

let playBtn = SKSpriteNode(imageNamed: "Play")
playBtn.position = CGPoint.zero
startGameBtn.addChild(playBtn)
```

地面：

```
func settingForeGround(){
    for i in 0..<foreGroundCount {
        let foreGround = SKSpriteNode.init(imageNamed: "Ground")
        foreGround.anchorPoint = CGPoint.init(x: 0, y: 1.0) //锚点
        foreGround.position = CGPoint.init(x: CGFloat(i) * foreGround.s
```

```
ize.width, y: regionalSart)//位置
        foreGround.zPosition = fbLayer.foreGround.rawValue//z 轴的原始值
        foreGround.name = "foreGround" //方便查找到 foreGround
        gameNode.addChild(foreGround)//添加前景到游戏世界上面
    }
}
```

得分标签：

```
func settingScoreLabel(){
    scoreLabel = SKLabelNode.init(fontNamed: fontFamily) //字体
    scoreLabel.fontColor = SKColor.init(red: 101.0/255.0, green: 71.0/2
55.0, blue: 73.0/255.0, alpha: 1.0) //字体颜色
    scoreLabel.position = CGPoint.init(x: size.width/2, y: size.height
- topWhite) //位置
    scoreLabel.verticalAlignmentMode = .top //垂直方向的对齐方式
    scoreLabel.text = "0"
    scoreLabel.zPosition = fbLayer.UI.rawValue //设置 z 轴位置
    gameNode.addChild(scoreLabel)
}
```

主菜单 logo（也就是那个“Pokemon Jump!!!!!”）

```
//logo
let logo = SKSpriteNode(imageNamed: "Logo")
logo.position = CGPoint(x: size.width/2, y: size.height * 0.8)
logo.name = "MainMenu"
logo.zPosition = fbLayer.UI.rawValue
gameNode.addChild(logo)
```

开始游戏按钮（也就是那个“Play”）

```
let startGameBtn = SKSpriteNode(imageNamed: "Button")
startGameBtn.position = CGPoint(x: size.width * 0.5, y: size.height * 0
.25)
startGameBtn.name = "MainMenu"
startGameBtn.zPosition = fbLayer.UI.rawValue
gameNode.addChild(startGameBtn)

let playBtn = SKSpriteNode(imageNamed: "Play")
playBtn.position = CGPoint.zero
startGameBtn.addChild(playBtn)
```

“Tap to Start”标签（这里只是提示，实际点任意区域都可开始）：

```
let course = SKSpriteNode.init(imageNamed: "Tutorial")
course.position = CGPoint.init(x: size.width * 0.5, y: regionalHeight *
 0.4 + regionalSart) //位置
course.name = "course" //标示
course.zPosition = fbLayer.UI.rawValue //图层
gameNode.addChild(course)  //添加到游戏世界
```

“Ready”标签：

```
let  prepare = SKSpriteNode.init(imageNamed: "ready")
prepare.position = CGPoint.init(x: size.width * 0.5, y: regionalHeight
* 0.7 + regionalSart)//位置
prepare.name  = "course"//标示
prepare.zPosition = fbLayer.UI.rawValue//图层
gameNode.addChild(prepare) //添加到游戏世界
```

计分板：

```
let scoreCard = SKSpriteNode.init(imageNamed: "ScoreCard")
scoreCard.position = CGPoint.init(x: size.width/2, y: size.height/2)  /
/位置
scoreCard.zPosition = fbLayer.UI.rawValue  //z 轴位置(图层)
gameNode.addChild(scoreCard)  //添加到游戏世界
```

当前分数标签：

```
let currentScoreLabel = SKLabelNode.init(fontNamed: fontFamily) //字体
currentScoreLabel.fontColor = SKColor.init(red: 101.0/255.0, green: 71.
0/255.0, blue: 73.0/255.0, alpha: 1.0)//颜色
currentScoreLabel.position = CGPoint.init(x: -scoreCard.size.width / 4,
 y: -scoreCard.size.height / 3) //位置
currentScoreLabel.text = "\(currentScore)"
currentScoreLabel.zPosition = fbLayer.UI.rawValue //z 轴位置(图层)
scoreCard.addChild(currentScoreLabel)  //添加到游戏世界
```

最高分数标签：

```
let highestScoreLabel = SKLabelNode.init(fontNamed: fontFamily) //字体
highestScoreLabel.fontColor = SKColor.init(red: 101.0/255.0, green: 71.
0/255.0, blue: 73.0/255.0, alpha: 1.0)//颜色
highestScoreLabel.position = CGPoint.init(x: scoreCard.size.width / 4,
```

```
y: -scoreCard.size.height / 3) //位置
highestScoreLabel.text = "\(highestScore())"
highestScoreLabel.zPosition = fbLayer.UI.rawValue //z 轴位置(图层)
scoreCard.addChild(highestScoreLabel)  //添加到游戏世界
```

“Game Over”标签：

```
let gameOverLabel = SKSpriteNode.init(imageNamed: "GameOver")
gameOverLabel.position = CGPoint.init(x: size.width/2, y: size.height/2
 + scoreCard.size.height/2 + topWhite + gameOverLabel.size.height/2)//位
置
gameOverLabel.zPosition = fbLayer.UI.rawValue //z 轴位置(图层)
gameNode.addChild(gameOverLabel) //添加到游戏世界
```

“OK”按钮：

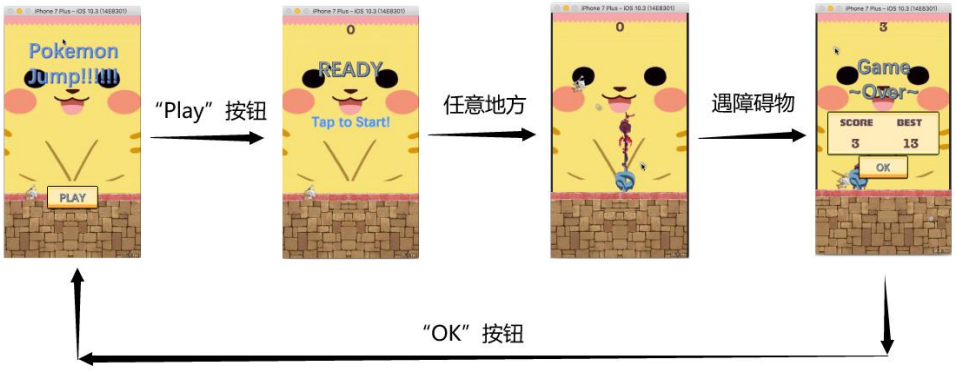
```
//OK 按钮
let okBtn = SKSpriteNode.init(imageNamed: "Button")
okBtn.position = CGPoint.init(x: size.width/2, y: size.height/2 - score
Card.size.height/2 - okBtn.size.height/2)//位置
okBtn.zPosition = fbLayer.UI.rawValue//z 轴位置(图层)
gameNode.addChild(okBtn)//添加到游戏世界

//OK 按钮上面的文字
let ok = SKSpriteNode.init(imageNamed: "OK")
ok.position = CGPoint.zero
ok.zPosition = fbLayer.UI.rawValue
okBtn.addChild(ok) //ok 文字添加到 OK 按钮之上
```

障碍物：

```
let bottomBarrie = makeBarrier(imgName: "CactusBottom")
bottomBarrie.name = "bottomBarrie"
let startX = size.width + bottomBarrie.size.width/2
let minY =  (regionalSart - bottomBarrie.size.height/2) + regionalHeigh
t * barrierMin
let maxY = (regionalSart - bottomBarrie.size.height/2) + regionalHeight
 * barrierMax
bottomBarrie.position = CGPoint.init(x: startX, y: CGFloat.random(min:
minY, max: maxY))
gameNode.addChild(bottomBarrie)
```

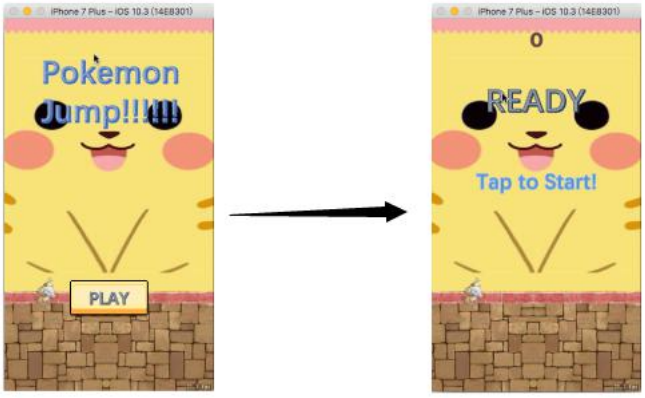
5 场景切换说明

初始页面→（Play 按钮）→准备页面→（任意地方单机）→游戏页面→（碰到千面避役（障碍物））→结算页面→（OK 按钮）→初始页面→……



6 场景切换概况

5.1 初始页面→准备页面



7 场景切换 1

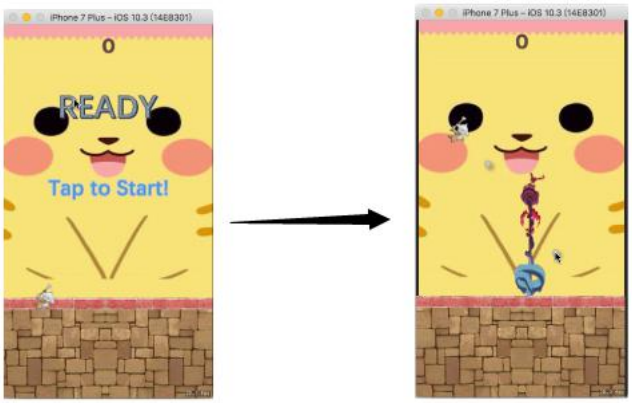
我们的页面跳转并不是传统意义上的页面跳转，也就是说，并不是从一

个完整的页面跳转到另一个完整的页面，而是弹出式的跳转。

比如说，当我们打开游戏的时候，首先会弹出初始页面，实际上它是由一个主页面加上一个菜单界面组成：

```
//切换到主菜单
func switchMainMenu(){
    currentGameState = .mainMenu
    settingUpBackGround()  //设置背景
    settingForeGround()    //设置前景
    settingGameUser()  //设置游戏主角(嘎啦嘎啦)
    settingMainMenu() //设置主菜单 UI
}

//设置主菜单
func settingMainMenu(){
    //logo
    let logo = SKSpriteNode(imageNamed: "Logo")
    logo.position = CGPoint(x: size.width/2, y: size.height * 0.8)
    logo.name = "MainMenu"
    logo.zPosition = fbLayer.UI.rawValue
    gameNode.addChild(logo)

    // 开始游戏按钮
    let startGameBtn = SKSpriteNode(imageNamed: "Button")
    startGameBtn.position = CGPoint(x: size.width * 0.5, y: size.height
 * 0.25)
    startGameBtn.name = "MainMenu"
    startGameBtn.zPosition = fbLayer.UI.rawValue
    gameNode.addChild(startGameBtn)

    let playBtn = SKSpriteNode(imageNamed: "Play")
    playBtn.position = CGPoint.zero
    startGameBtn.addChild(playBtn)
}
```

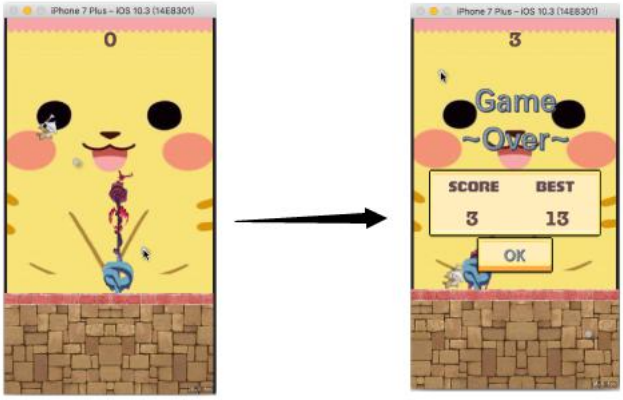
5.2 准备页面→游戏页面



8 场景切换 2

然后，再点击开始游戏之后，我们就进入了准备界面，这里加入了一个标签：“Tap to Start”，单机屏幕任意区域跳转到游戏页面。

游戏页面→结算页面：

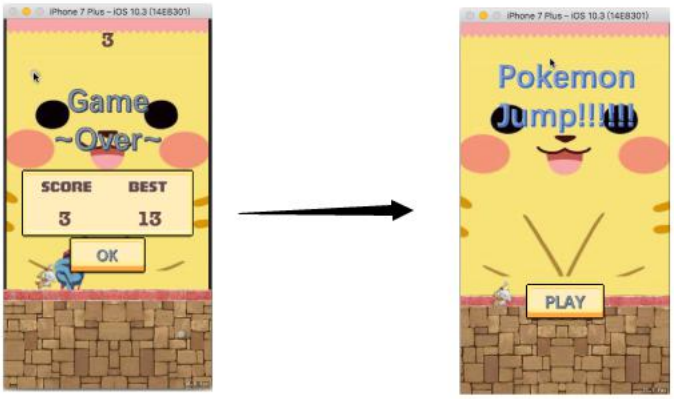


9 场景切换 3

玩家开始游戏，当遇到障碍物即跳转结算页面：

```
func crashBarrierDetection(){
    if crashBarrier {
        crashBarrier = false
        switchFallState() //切换到跌落状态
        gameUser.position = CGPoint.init(x: gameUser.position.x, y:regi
onalSart + gameUser.size.width/2 ) //嘎啦嘎啦的位置
        switchShowScore()    //切换到显示分数
    }
}
```

5.3 结算页面→初始页面



10 场景切换 4

显示分数以及最高分，同时还有一个“OK”按钮，单机它即可开始新一轮的游戏：

```
func switchPlayingState (){
    currentGameState = .playing
    gameNode.enumerateChildNodes(withName: "course") { (matchingUnit, _
) in
        matchingUnit.run(SKAction.sequence([
            SKAction.removeFromParent(),
            SKAction.fadeOut(withDuration: 0.05)
            ]))
    }
    infiniteMakeBarrier()    //无限重生障碍
    birdFly() //跳一下
}
```

6 应用逻辑说明

6.1 跳跃功能

主角的跳跃模式在此被定义为双段跳，这个功能的实现主要是靠一个变

量来记录离地后跳越次数，如果大于 2 则不可再次跳跃，当然，如果判断接触地面则跳跃次数清 0。

定义计数变量：

```
var jumpCount = 0 //跳跃次数，最大二段跳
```

每次调用 jump 函数时有以下代码：

```
jumpCount = jumpCount + 1
if(jumpCount > 2)
{
    return
}
```

撞击地面检查以及计数变 0

```
func crashFloorDetection(){
if crashFloor {
    crashFloor = false
    jumpCount = 0
}
```

6.2 障碍物生成和检测

生成其实很简单，就是每隔固定时间生成一些不同长度的障碍物：

一些变量：

```
let flyUpSpeed = 800.0  //嘎啦嘎啦跳起速度
let barrierMin:CGFloat = 0.1 //障碍物最小乘数
let barrierMax:CGFloat = 0.6 //障碍物最大乘数
let firstMakeBarrierInterval:TimeInterval = 3 //首次生成障碍延迟
let rebirthMakeBarrierInterval:TimeInterval = 1.75 //每次重生障碍延迟
```

障碍物检测：

```
/**
    * 撞击障碍物检查
    */
func crashBarrierDetection(){
    if crashBarrier {
        crashBarrier = false
```

```
        switchFallState() //切换到跌落状态
        gameUser.position = CGPoint.init(x: gameUser.position.x, y:regi
onalSart + gameUser.size.width/2 ) //嘎啦嘎啦的位置
        switchShowScore()   //切换到显示分数
    }
}
```

障碍物生成：

```
/**
    * 创建障碍物
    */
func makeBarrier(imgName:String) -> SKSpriteNode{
    let barrie = SKSpriteNode.init(imageNamed: imgName)  //障碍物
    barrie.zPosition = fbLayer.barrie.rawValue //z 轴的原始值

    barrie.userData = NSMutableDictionary() //初始化数据



    let offsetX = barrie.size.width * barrie.anchorPoint.x
    let offsetY = barrie.size.height * barrie.anchorPoint.y

    let path = CGMutablePath()

    path.move(to: CGPoint(x:  3 - offsetX, y: 1 - offsetY), transform:
CGAffineTransform.identity)
    path.addLine(to: CGPoint(x:  3 - offsetX, y: 27 - offsetY), transfo
rm: CGAffineTransform.identity)
    path.addLine(to: CGPoint(x:  2 - offsetX, y: 70 - offsetY), transfo
rm: CGAffineTransform.identity)
    path.addLine(to: CGPoint(x:  5 - offsetX, y: 216 - offsetY), transf
orm: CGAffineTransform.identity)
    path.addLine(to: CGPoint(x:  6 - offsetX, y: 311 - offsetY), transf
orm: CGAffineTransform.identity)
    path.addLine(to: CGPoint(x:  49 - offsetX, y: 312 - offsetY), trans
form: CGAffineTransform.identity)
    path.addLine(to: CGPoint(x:  48 - offsetX,  y: 185 - offsetY), tran
sform: CGAffineTransform.identity)
    path.addLine(to: CGPoint(x:  47 - offsetX,  y: 85 - offsetY), trans
form: CGAffineTransform.identity)
    path.addLine(to: CGPoint(x:  49 - offsetX,  y: 1 - offsetY), transf
orm: CGAffineTransform.identity)

    path.closeSubpath()
```

```
    barrie.physicsBody = SKPhysicsBody(polygonFrom: path)
    barrie.physicsBody?.collisionBitMask = 0 //关闭系统的碰撞处理
    barrie.physicsBody?.categoryBitMask = physicsLayer.barrier//类别源码
    barrie.physicsBody?.contactTestBitMask = physicsLayer.gameUser  //打开碰撞检测

    return barrie
}
/**
    * 生成障碍
    */
func outPutBarrier(){
    //底部障碍
    let bottomBarrie = makeBarrier(imgName: "CactusBottom")
    bottomBarrie.name = "bottomBarrie"
    let startX = size.width + bottomBarrie.size.width/2
    let minY =  (regionalSart - bottomBarrie.size.height/2) + regionalHeight * barrierMin
    let maxY = (regionalSart - bottomBarrie.size.height/2) + regionalHeight * barrierMax
    bottomBarrie.position = CGPoint.init(x: startX, y: CGFloat.random(min: minY, max: maxY))
    gameNode.addChild(bottomBarrie)
    //顶部障碍
    let topBarrier = makeBarrier(imgName: "CactusTop")
    topBarrier.name = "topBarrier"
    topBarrier.zRotation = CGFloat(180).degreesToRadians()
    topBarrier.position = CGPoint.init(x: startX, y: 0)
    gameNode.addChild(topBarrier)
    //移动参数
    let moveX = -(size.width + bottomBarrie.size.width)  //X 轴移动距离
    let moveTime = moveX / CGFloat(foreGroundMoveSpeed)   //移动持续时间
    let moveQueue = SKAction.sequence([
        SKAction.moveBy(x: moveX, y: 0, duration: TimeInterval(moveTime)),
        SKAction.removeFromParent()
        ])  //移动队列
    topBarrier.run(moveQueue)
    bottomBarrie.run(moveQueue)
}
/**
    * 无限重生障碍
    */
```

```
func infiniteMakeBarrier(){
    let firstInterval = SKAction.wait(forDuration: firstMakeBarrierInte
rval) //首次延迟
    let rebirthBarrie = SKAction.run (outPutBarrier)  //重生障碍
    let rebirthInterval = SKAction.wait(forDuration: firstMakeBarrierIn
terval) //每次重生间隔
    let rebirthQueue = SKAction.sequence([rebirthBarrie,rebirthInterval
])  //重生队列
    let infiniteRebirth = SKAction.repeatForever(rebirthQueue)  //无限重
生
    let totalQueue = SKAction.sequence([firstInterval,infiniteRebirth])
 //总的动作队列
    run(totalQueue, withKey: "infiniteMake")
}
/**
    * 停止生成障碍
    */
func stopMakeBarrier(){
    removeAction(forKey: "infiniteMake")
    gameNode.enumerateChildNodes(withName: "topBarrier") { (matchingUni
t, _) in
        matchingUnit.removeAllActions()
    }
    gameNode.enumerateChildNodes(withName: "bottomBarrie") { (matchingU
nit, _) in
        matchingUnit.removeAllActions()
    }

}
```

6.3 计分功能

这里实际上就是判断 x 距离，如果超过障碍物的 x 值加上障碍物的宽度，我们就判断嘎啦嘎啦通过了这个障碍物，记录分数+1。否则就跳转结算界面，如果分数大于最大分数，则更新最大分数。

```
/**
    * 更新得分
    */
func updateScore(){
    gameNode.enumerateChildNodes(withName: "topBarrier") { (matching
```

```
Unit, _) in
        if let barrier = matchingUnit as? SKSpriteNode {
            if let  passed = barrier.userData?["passed"] as? NSNumbe
r {
                if passed.boolValue {
                    return  //已经计算过一次得分
                }
            }
            if (self.gameUser.position.x > barrier.position.x + barr
ier.size.width/2){
                //嘎啦嘎啦已经完全通过障碍物
                self.currentScore = self.currentScore + 1
                self.scoreLabel.text = "\(self.currentScore)"
                barrier.userData?["passed"]  = NSNumber.init(value:
true)
            }
        }
    }
}
```

7 相关技术说明

7.1 动画实现技术

动画其实就是对多个图片或者动作按一定的时间间隔来进行连续不断地播放。同时，我们这里也有对物体坐标的改变这样子的动画来实现跳跃等动画。

一些变量

```
let animationDelay = 0.3 //动画延迟
let userframeNumber = 4  //角色动画总帧数
```

各种动画

```
//动画特效 gameOver
```

```
gameOverLabel.setScale(0)  //先缩 0
gameOverLabel.alpha = 0
let animationQueue = SKAction.group([  //动画组
    SKAction.fadeIn(withDuration: animationDelay),
    SKAction.scale(to: 1.0, duration: animationDelay)
    ])
animationQueue.timingMode = .easeInEaseOut  //动画函数
gameOverLabel.run(SKAction.sequence([
    SKAction.wait(forDuration: animationDelay),  //延迟
    animationQueue
    ]))
//动画特效计分板
scoreCard.position = CGPoint.init(x: size.width / 2, y: -scoreCard.size
.height/2) //位置
let moveAnimation = SKAction.move(to: CGPoint.init(x: size.width / 2, y
: size.height / 2), duration: animationDelay) //向上移动动画
moveAnimation.timingMode = .easeInEaseOut  //动画函数
scoreCard.run(SKAction.sequence([
    SKAction.wait(forDuration: animationDelay * 2),
    moveAnimation
    ]))
//OK 按钮特效
okBtn.alpha = 0
let btnAnimation = SKAction.sequence([
    SKAction.wait(forDuration: animationDelay * 3),
    SKAction.fadeIn(withDuration: animationDelay)
    ])
okBtn.run(btnAnimation)
//动画
let songAnimation = SKAction.sequence([
    SKAction.run(switchGameOverState)  //切换至游戏结束状态
    ])
run(songAnimation)
```

7.2 更新检查

其实就是每隔一段时间检查下状态，比如遇到障碍物了呀（跳转结算页面）、成功避开障碍物了呀（计分+1）、单机跳跃了呀（跳跃次数+1）、接触到地面了呀（跳跃次数清 0）等等等等，然后做出相应的响应。

```
/**
```

```
    * 每一帧都会调用更新
    */
override func update(_ currentTime: TimeInterval) {
    if (lastUpdateTime > 0){
        dt = currentTime  - lastUpdateTime
    } else {
        dt = 0
    }
    lastUpdateTime = currentTime

    switch currentGameState {
    case .mainMenu:
        break
    case .course:
        break
    case .playing:
        updateGameUser()  //更新主角
        updateForeGround()  //更新前景
        crashBarrierDetection() //撞击障碍物检查
        crashFloorDetection()  //撞击地面检查
        updateScore() //更新得分

        break
    case .fall:
        updateGameUser()  //更新主角
        crashFloorDetection()  //撞击地面检查
        break
    case .showScore:
        break
    case .gameOver:
        break

    }
}
```

8 分工

姓名	完成任务	占比
胡杭天	框架设计（页面跳转以及部件布局）、部分功能实现（主要在计分板及动画部分）、逻辑结构设计（游戏逻辑）、素材收集与报告撰写	50%
周子杰	框架设计（整体菜单以及部件更新）、部分功能实现（主要在障碍物与运动部分）、逻辑结构设计（游戏背景）、素材收集与报告撰写	50%

9 总结感想

9.1 困难及解决办法

我们组的两个人都是第一次接触 swift 语言，作为一门 iOS 嵌入式开发的课程，最大的困难其实在于系统的问题，习惯了 Windows 系统上操作的我们在初接触 swift 时，主要还是对 mac 系统的不适应。一些常见的问题，明知如果是在 Windows 系统中自己能够轻而易举的解决问题，但是在 mac 的系统中，我们可能就会迷茫慌张不知所措，但是我们也因此学到了很多新的东西，所以说这些困难对我们而言也算是一种提升和历练。

另外的一些技术上的问题，比较经典的、我们在课设过程中不断遇到的一个问题就是利用图形界面生成 UI 时经常会遇到的一个“thread1”问题和“can’t be saved”的问题，这两个问题困扰了我们许久许久。其中，对于前者，我们的解决方案是在遇到这种情况时不断 WIN+Z 直到不在有这个问题，但是有的时候可能程序行进得太远了就会导致我们回不到那个能够正常运行的点了，后者我们至今没找到解决方案，我们也上各种网站和论坛找了各种解决方案，可惜都没有找到特别有效的方案。所以我们最终的解决方案是利用代码来生成图形界面。但是由此也会引起一个新的问题是，没用图形界面去绘制界面，具体的显示位置需要运行项目才能知道效果如何，反复如此会很麻烦。

9.2 感想

胡杭天：

经过几周的密集的 iOS 集中课程，我感觉收获了非常多的知识，并且都在开发中得到了实践和应用。这些天来我们主要是在 mac 的虚拟机中对 swift 和 XCode 的使用来进行学习。在课上，我们通过控件的拖拽来完成页面布局，并且完成各组件的代码定义，感觉每节课循序渐进，使我们逐步地掌握了 iOS 嵌入式开发的各种技能点。另外的感受就是有了 C++的语言基础之后，再对一门全新的语言进行学习是一件很快的事情，在管老师的引导之下，我感觉

每过一节课我都对 swift 的理解更深了一层。这种感觉非常的棒。但是话说回来，在知识的学习中，老师更多的只是作为一个引导者和引路人，更多的还需要我们自己在课后花更多的时间去掌握和学习，才能更好的掌握这门语言和开发技术。但是最后的大作业开发中 Swift 语言给我的体验不是很好，也让我理解了其一直被开发人员诟病的原因，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，其实这也是作为程序员个人在进行程序开发的时候应该注意的一个问题，那就是代码的可移植性和鲁棒性。

总体的来说，这次课程设计给我带来的更多是愉悦的享受，去实现一个自己想到小游戏想法，然后再通过自己新学的知识，一步步地建立起代码框架、一步步地去完善加工、一步步地完善体系并不断润色，最终出成品，那种满足感和欢愉感，只有真的认真去做的人才能感受到。但是总体来说，我对于这门语言的了解也还是初步的较浅的，还是有很长的路要走，就像我们的程序也仍然存在很大的改进空间一样。希望在今后的工作和学习中，有需要的情况下能对这门语言有更深层次的理解，并且能够熟练掌握第三方的库来实现更多的更复杂的功能的开发。

周子杰：

这次试验对我来说还是收获颇多的。Swift 作为一个新的语言，其本身对我而言还是很有吸引力的。可惜的是，由于要配合虚拟机来安装 mac 系统来进行编程，会有明显的卡顿等感受，编程体验还是不太友好的。同时，还会有很多神奇的错误，错一步可能就得重新开项目，这一点还是挺麻烦的。

但是，发现错误并且解决错误的过程确实令人欣慰的。比如说，我们发现如果在 smb 中运行项目有时会报错，但是转移到本地即可消除这种错误。反复尝试之后我们发现是路劲的问题，最终我们将我们的文件都改成了相对路径，这样大大提高了我们的项目的可移植性。

当然，在写项目的时候我们也遇到了很多问题。比如说，我们一开始不知道如何不断地生成障碍物。但是再查阅网上资料之后我发现我们可以通过

建立一个菜单并且通过调用一个时间函数不断执行这个菜单来刷新所有的状态。同样的道理，我们可以每隔一定的时间就调用我们的障碍物生成函数来进行重新生成一个新的障碍物（这个障碍物的长度是随机的）。

最后的最后，对于我们的最终结果，我还是很满意的。这是我最喜欢的题材：宝可梦！也是我付出了很多心血才得以实现的效果。我很开心最后它能有如此完整的一个完成度，不过它确实还有提升的空间，比如我们可以随着时间的增加加快障碍物生成的速度。

（附：之所以选嘎啦嘎啦作为主角是因为这是我最喜欢的宝可梦之一，之所以选千面避役作为反派是因为这是我在玩宝可梦剑盾时拥有的第一个宝可梦，我觉得它的气质真的很适合做反派欸，哈哈哈）