C323 Final Project:
Marble Maze Mobile

Team 3 (Umber):
Lukas Vatistas, Connor Hands

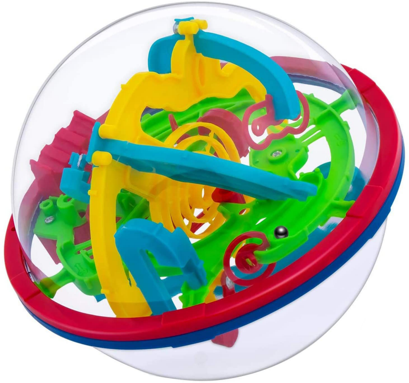
General Description

Our app is a 2D ball maze game with three tabs. **The goal is to guide the ball through the maze to reach the center. The ball's movement is controlled by CoreMotion, requiring players to tilt their phone to navigate the maze.**

The second tab will be the settings that allows players to select their **difficulty level**: easy, medium, or hard. Players can also customize features like the **color of their ball and the background**. The settings are also persistently stored.

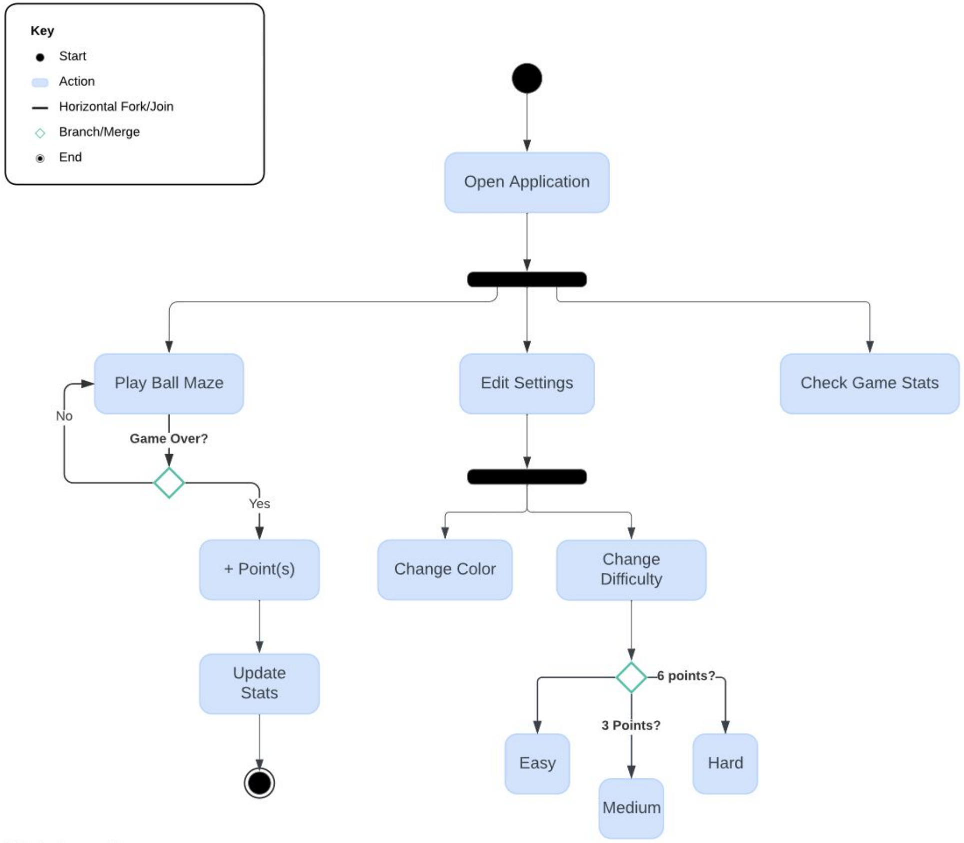
The third tab is the stats tab, which records achievements such as the player's fastest time for each difficulty, the number of times they have completed a maze, and the number of times they have attempted a maze. This data is stored persistently, so players can track their progress over time, even after closing and reopening the app.

We will be implementing **Core Data, CoreMotion, and SpriteKit**. Core Data for the persistent storage, CoreMotion for the ball movement, and SpriteKit for the animation of the game. We will save all high scores and settings to storage, allowing players to pick up where they left off.



Activity Diagram

Ball Maze App Activity Diagram



Key

- ● Start
- ▭ Action
- — Horizontal Fork/Join
- ◇ Branch/Merge
- ◉ End

Open Application

Play Ball Maze

Edit Settings

Check Game Stats

Game Over?

No

Yes

+ Point(s)

Update Stats

Change Color

Change Difficulty

6 points?

3 Points?

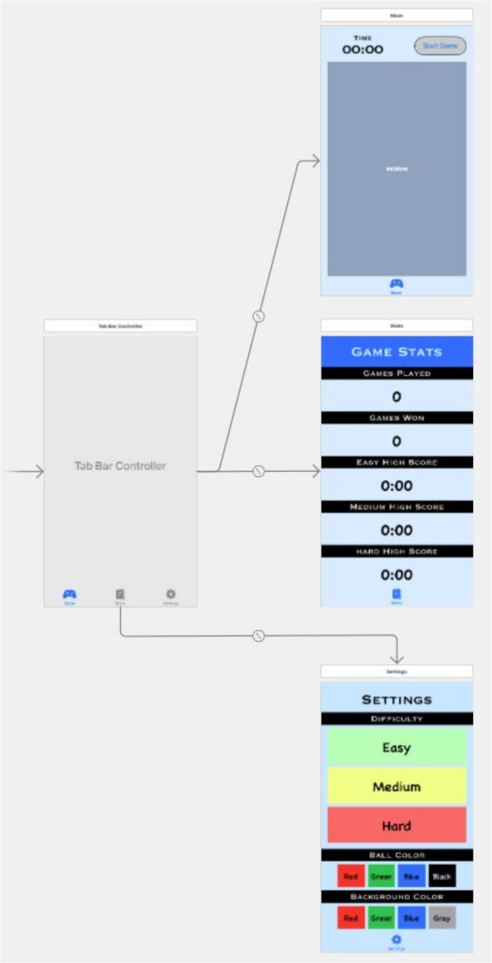
Easy

Medium

Hard

C323 Assignment 3
Lukas Vatistas, Connor Hands

StoryBoard



Maze

Time
00:00
Start Game

Tab Bar Controller

Game Stats
Games Played
0
Games Won
0
Easy High Score
0:00
Medium High Score
0:00
Hard High Score
0:00

Settings
Difficulty
Easy
Medium
Hard
Ball Color
Red Green Blue Black
Background Color
Red Green Blue Gray

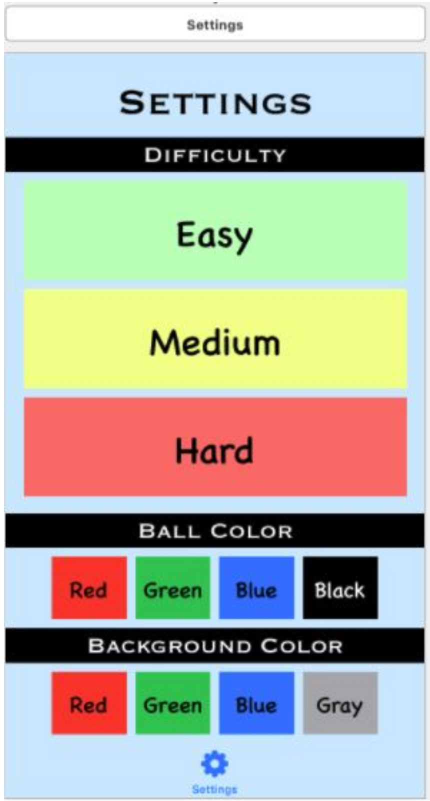
Maze

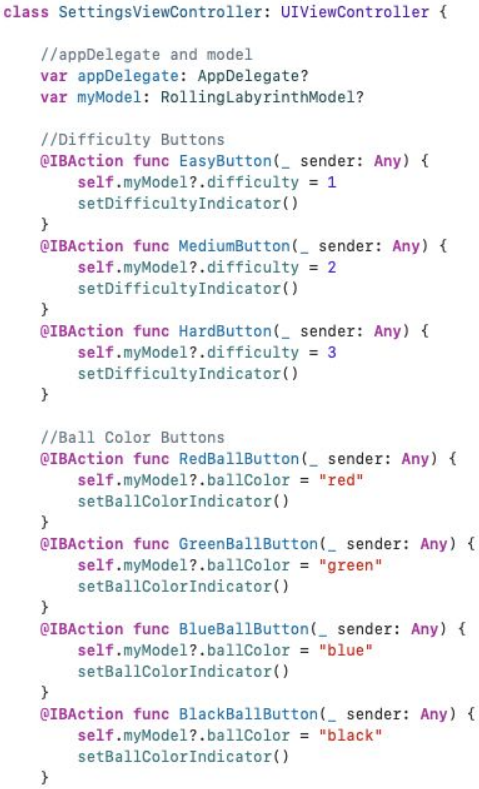
Tab Bar
Controller

Stats

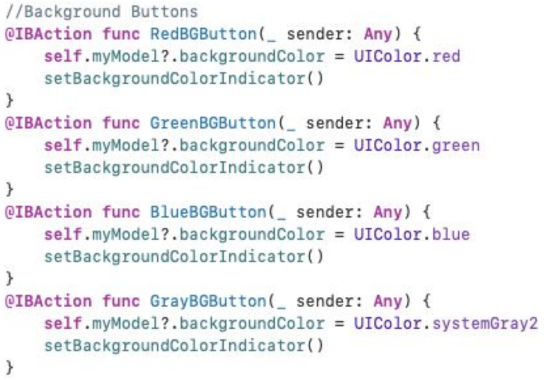
Settings

Settings

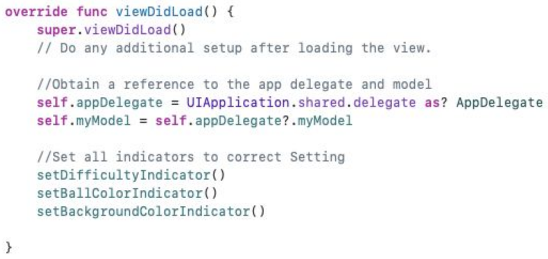




```swift
class SettingsViewController: UIViewController {

    //appDelegate and model
    var appDelegate: AppDelegate?
    var myModel: RollingLabyrinthModel?

    //Difficulty Buttons
    @IBAction func EasyButton(_ sender: Any) {
        self.myModel?.difficulty = 1
        setDifficultyIndicator()
    }
    @IBAction func MediumButton(_ sender: Any) {
        self.myModel?.difficulty = 2
        setDifficultyIndicator()
    }
    @IBAction func HardButton(_ sender: Any) {
        self.myModel?.difficulty = 3
        setDifficultyIndicator()
    }

    //Ball Color Buttons
    @IBAction func RedBallButton(_ sender: Any) {
        self.myModel?.ballColor = "red"
        setBallColorIndicator()
    }
    @IBAction func GreenBallButton(_ sender: Any) {
        self.myModel?.ballColor = "green"
        setBallColorIndicator()
    }
    @IBAction func BlueBallButton(_ sender: Any) {
        self.myModel?.ballColor = "blue"
        setBallColorIndicator()
    }
    @IBAction func BlackBallButton(_ sender: Any) {
        self.myModel?.ballColor = "black"
        setBallColorIndicator()
    }
```

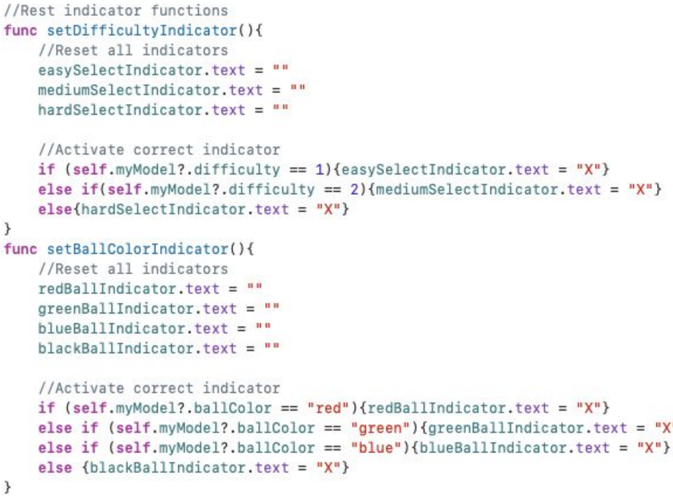


```swift
//Background Buttons
@IBAction func RedBGButton(_ sender: Any) {
    self.myModel?.backgroundColor = UIColor.red
    setBackgroundColorIndicator()
}
@IBAction func GreenBGButton(_ sender: Any) {
    self.myModel?.backgroundColor = UIColor.green
    setBackgroundColorIndicator()
}
@IBAction func BlueBGButton(_ sender: Any) {
    self.myModel?.backgroundColor = UIColor.blue
    setBackgroundColorIndicator()
}
@IBAction func GrayBGButton(_ sender: Any) {
    self.myModel?.backgroundColor = UIColor.systemGray2
    setBackgroundColorIndicator()
}
```

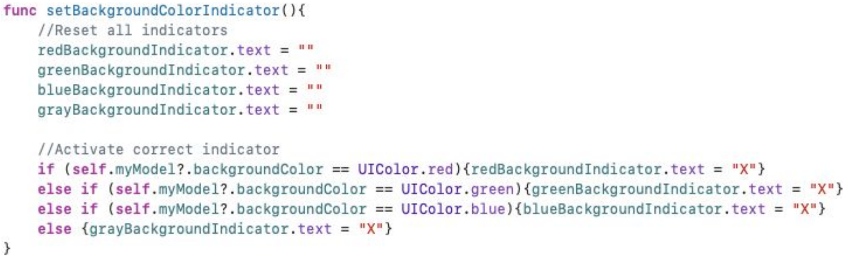


```swift
override func viewDidLoad() {
    super.viewDidLoad()
    // Do any additional setup after loading the view.

    //Obtain a reference to the app delegate and model
    self.appDelegate = UIApplication.shared.delegate as? AppDelegate
    self.myModel = self.appDelegate?.myModel

    //Set all indicators to correct Setting
    setDifficultyIndicator()
    setBallColorIndicator()
    setBackgroundColorIndicator()

}
```

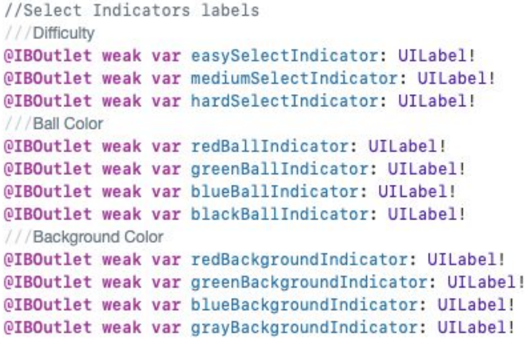
Setting Indicators



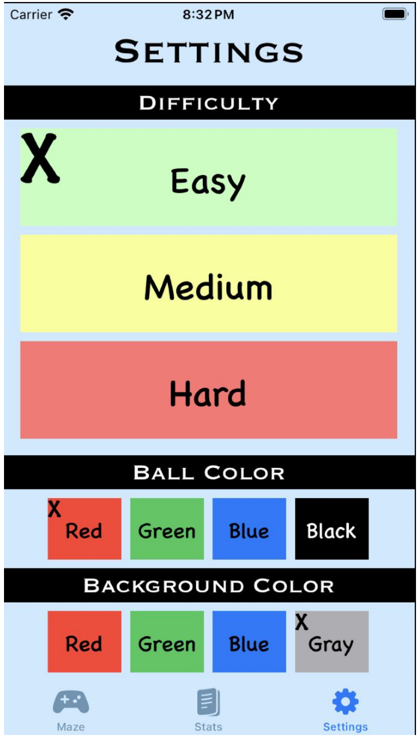
```
//Rest indicator functions
func setDifficultyIndicator(){
    //Reset all indicators
    easySelectIndicator.text = ""
    mediumSelectIndicator.text = ""
    hardSelectIndicator.text = ""

    //Activate correct indicator
    if (self.myModel?.difficulty == 1){easySelectIndicator.text = "X"}
    else if(self.myModel?.difficulty == 2){mediumSelectIndicator.text = "X"}
    else{hardSelectIndicator.text = "X"}
}
func setBallColorIndicator(){
    //Reset all indicators
    redBallIndicator.text = ""
    greenBallIndicator.text = ""
    blueBallIndicator.text = ""
    blackBallIndicator.text = ""

    //Activate correct indicator
    if (self.myModel?.ballColor == "red"){redBallIndicator.text = "X"}
    else if (self.myModel?.ballColor == "green"){greenBallIndicator.text = "X"}
    else if (self.myModel?.ballColor == "blue"){blueBallIndicator.text = "X"}
    else {blackBallIndicator.text = "X"}
}
```



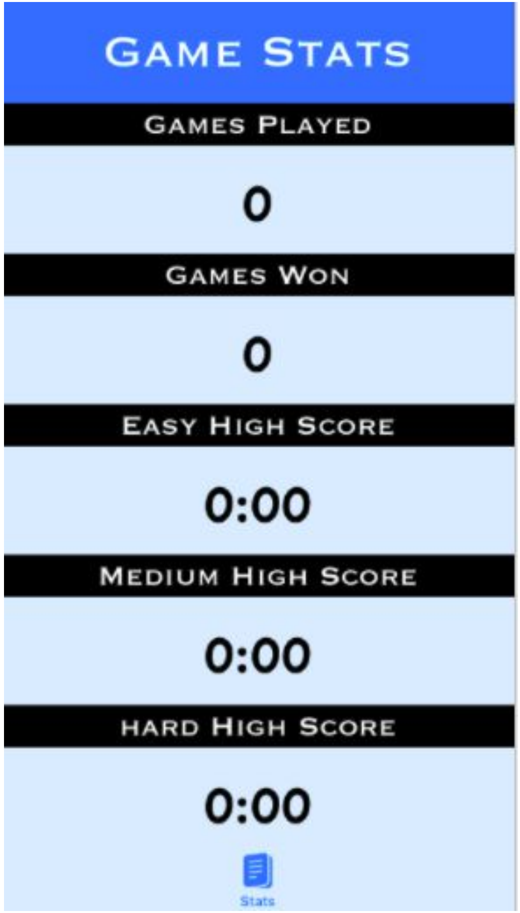
```
func setBackgroundColorIndicator(){
    //Reset all indicators
    redBackgroundIndicator.text = ""
    greenBackgroundIndicator.text = ""
    blueBackgroundIndicator.text = ""
    grayBackgroundIndicator.text = ""

    //Activate correct indicator
    if (self.myModel?.backgroundColor == UIColor.red){redBackgroundIndicator.text = "X"}
    else if (self.myModel?.backgroundColor == UIColor.green){greenBackgroundIndicator.text = "X"}
    else if (self.myModel?.backgroundColor == UIColor.blue){blueBackgroundIndicator.text = "X"}
    else {grayBackgroundIndicator.text = "X"}
}
```



```
//Select Indicators labels
///Difficulty
@IBOutlet weak var easySelectIndicator: UILabel!
@IBOutlet weak var mediumSelectIndicator: UILabel!
@IBOutlet weak var hardSelectIndicator: UILabel!
///Ball Color
@IBOutlet weak var redBallIndicator: UILabel!
@IBOutlet weak var greenBallIndicator: UILabel!
@IBOutlet weak var blueBallIndicator: UILabel!
@IBOutlet weak var blackBallIndicator: UILabel!
///Background Color
@IBOutlet weak var redBackgroundIndicator: UILabel!
@IBOutlet weak var greenBackgroundIndicator: UILabel!
@IBOutlet weak var blueBackgroundIndicator: UILabel!
@IBOutlet weak var grayBackgroundIndicator: UILabel!
```



Stats



GAME STATS

GAMES PLAYED

0

GAMES WON

0

EASY HIGH SCORE

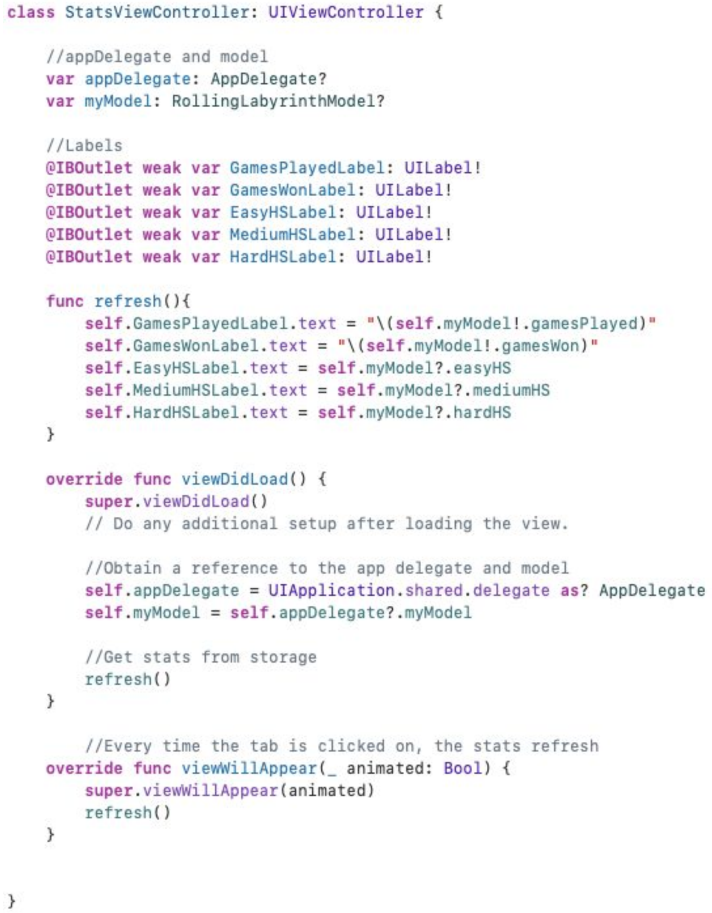
0:00

MEDIUM HIGH SCORE

0:00

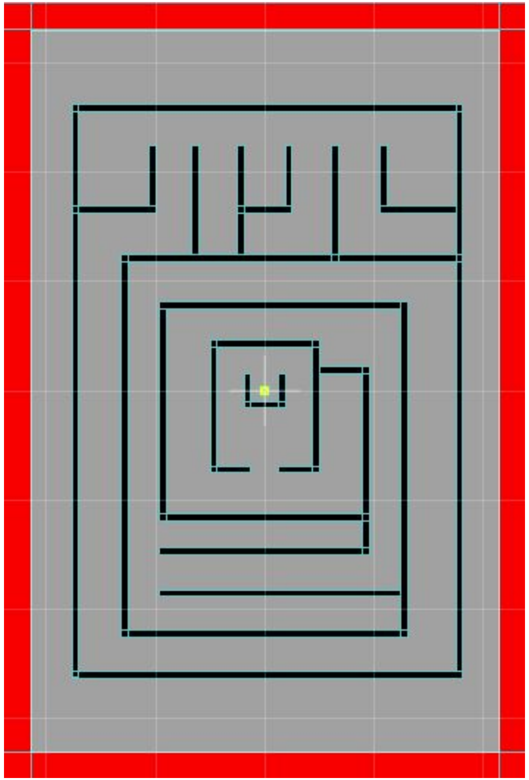
HARD HIGH SCORE

0:00

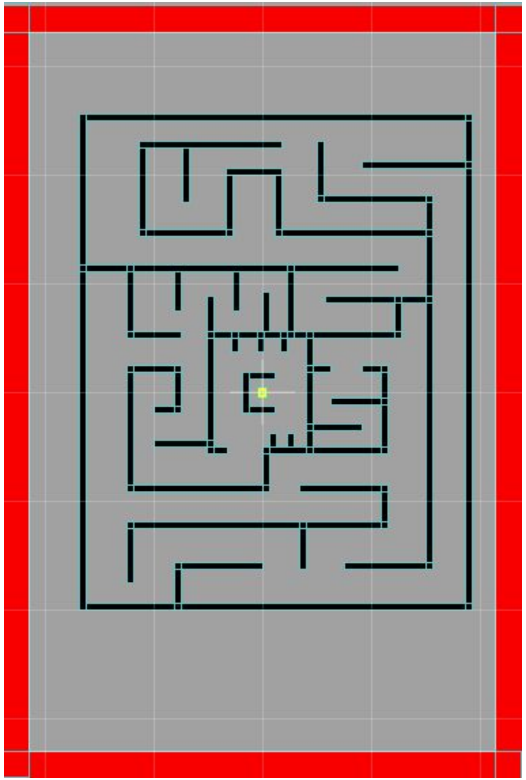


```
class StatsViewController: UIViewController {

    //appDelegate and model
    var appDelegate: AppDelegate?
    var myModel: RollingLabyrinthModel?

    //Labels
    @IBOutlet weak var GamesPlayedLabel: UILabel!
    @IBOutlet weak var GamesWonLabel: UILabel!
    @IBOutlet weak var EasyHSLabel: UILabel!
    @IBOutlet weak var MediumHSLabel: UILabel!
    @IBOutlet weak var HardHSLabel: UILabel!

    func refresh(){
        self.GamesPlayedLabel.text = "\(self.myModel!.gamesPlayed)"
        self.GamesWonLabel.text = "\(self.myModel!.gamesWon)"
        self.EasyHSLabel.text = self.myModel?.easyHS
        self.MediumHSLabel.text = self.myModel?.mediumHS
        self.HardHSLabel.text = self.myModel?.hardHS
    }

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view.

        //Obtain a reference to the app delegate and model
        self.appDelegate = UIApplication.shared.delegate as? AppDelegate
        self.myModel = self.appDelegate?.myModel

        //Get stats from storage
        refresh()
    }

        //Every time the tab is clicked on, the stats refresh
    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)
        refresh()
    }


}
```

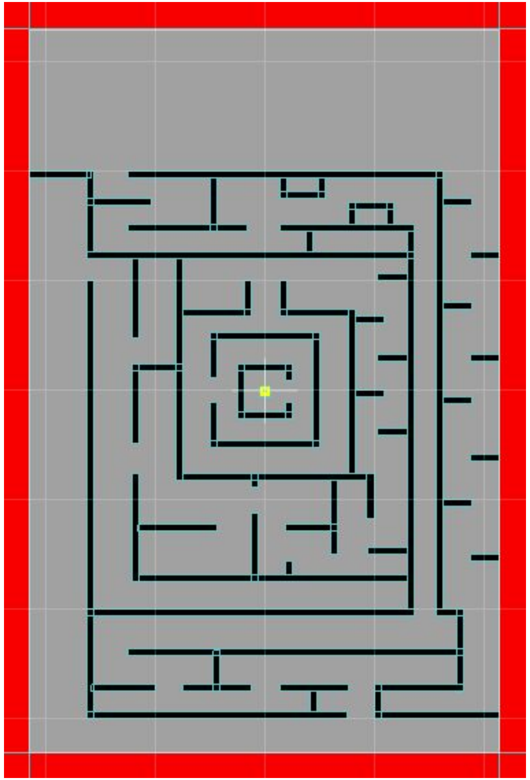
Easy



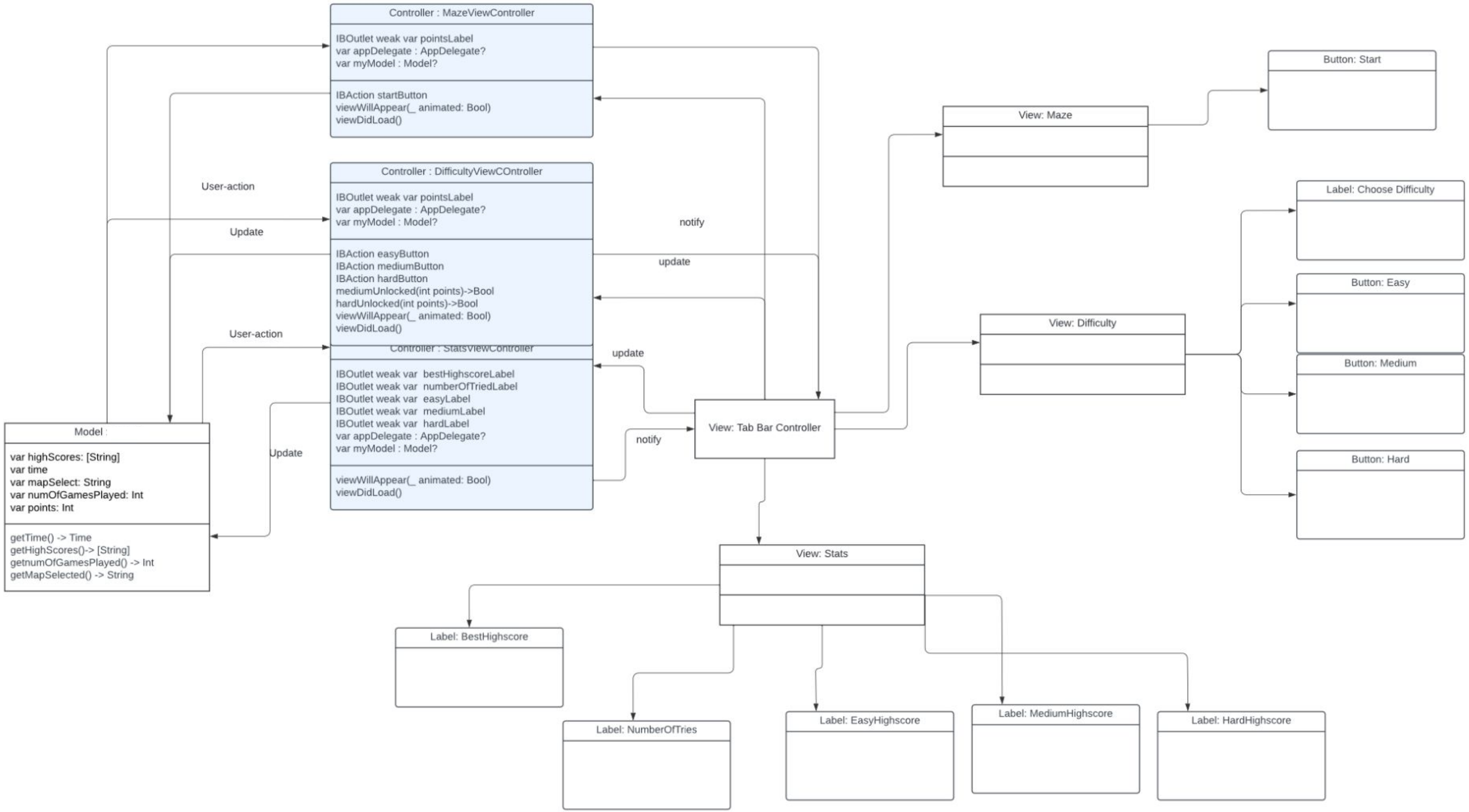
Medium



Hard



Class Diagram



Controller : MazeViewController

IBOutlet weak var pointsLabel
var appDelegate : AppDelegate?
var myModel : Model?

IBAction startButton
viewWillAppear(_ animated: Bool)
viewDidLoad()

Controller : DifficultyViewCOntroller

IBOutlet weak var pointsLabel
var appDelegate : AppDelegate?
var myModel : Model?

IBAction easyButton
IBAction mediumButton
IBAction hardButton
mediumUnlocked(int points)->Bool
hardUnlocked(int points)->Bool
viewWillAppear(_ animated: Bool)
viewDidLoad()

Controller : StatsViewController

IBOutlet weak var bestHighscoreLabel
IBOutlet weak var numberOfTriedLabel
IBOutlet weak var easyLabel
IBOutlet weak var mediumLabel
IBOutlet weak var hardLabel
var appDelegate : AppDelegate?
var myModel : Model?

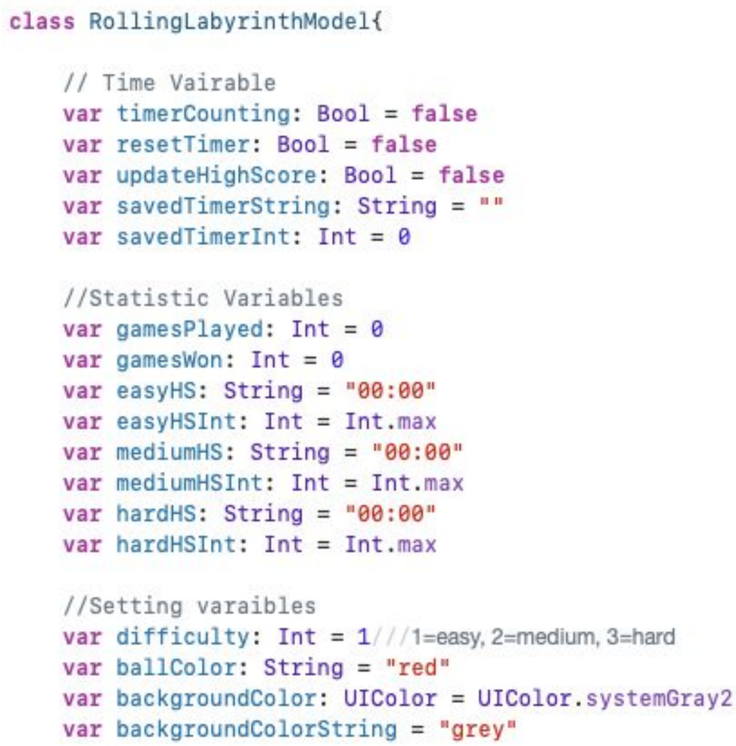
viewWillAppear(_ animated: Bool)
viewDidLoad()

Model :

var highScores: [String]
var time
var mapSelect: String
var numOfGamesPlayed: Int
var points: Int

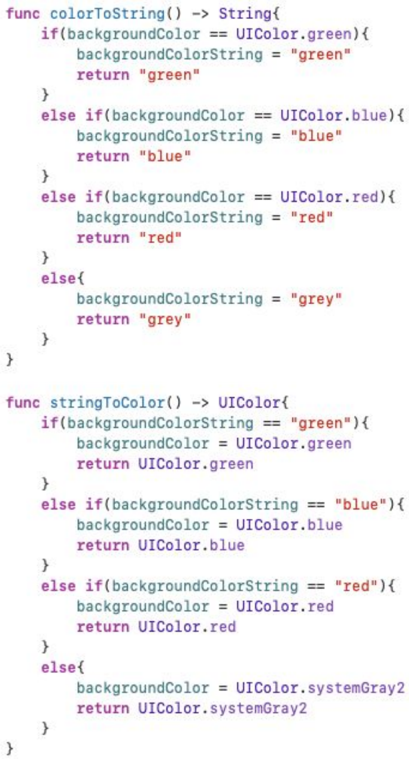
getTime() -> Time
getHighScores()-> [String]
getnumOfGamesPlayed() -> Int
getMapSelected() -> String

Model



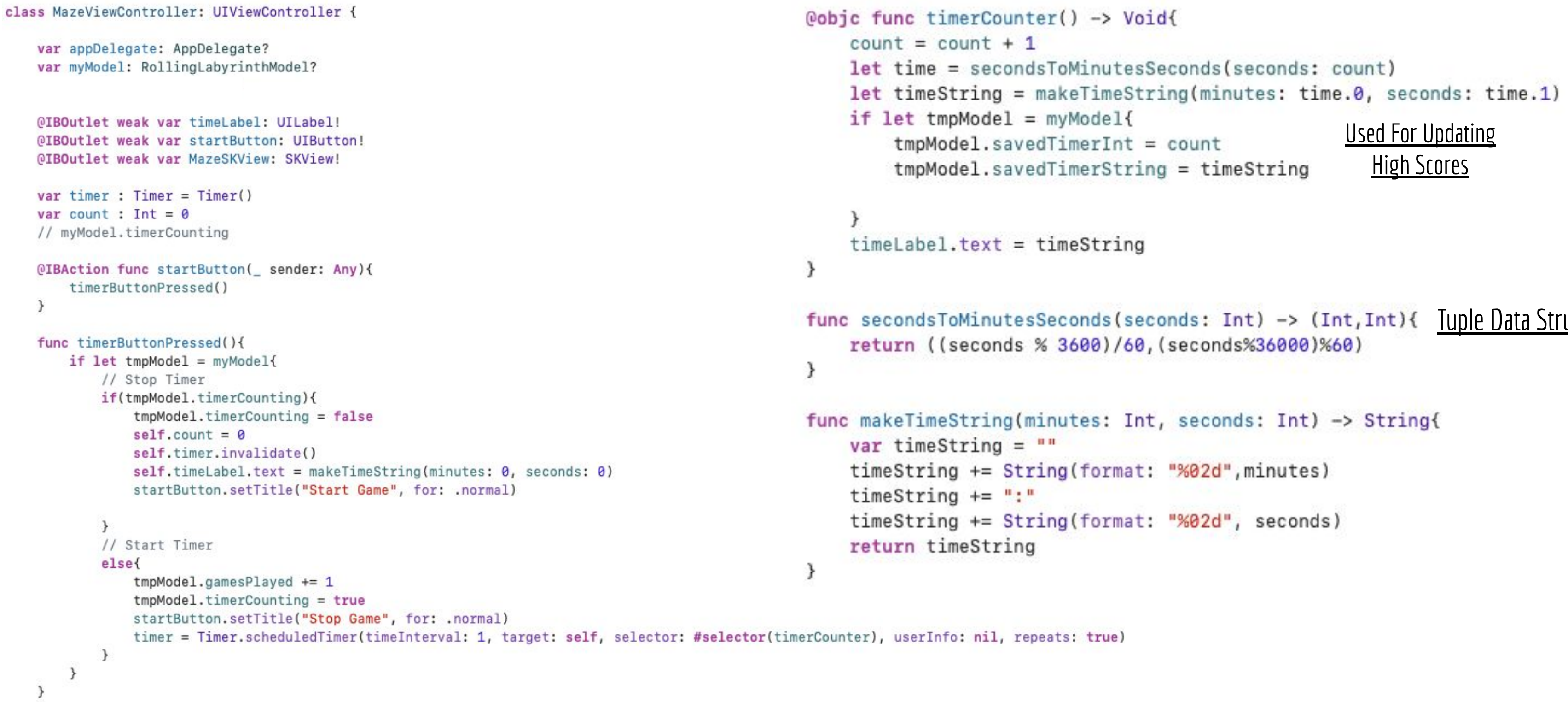
```
class RollingLabyrinthModel{

    // Time Vairable
    var timerCounting: Bool = false
    var resetTimer: Bool = false
    var updateHighScore: Bool = false
    var savedTimerString: String = ""
    var savedTimerInt: Int = 0

    //Statistic Variables
    var gamesPlayed: Int = 0
    var gamesWon: Int = 0
    var easyHS: String = "00:00"
    var easyHSInt: Int = Int.max
    var mediumHS: String = "00:00"
    var mediumHSInt: Int = Int.max
    var hardHS: String = "00:00"
    var hardHSInt: Int = Int.max

    //Setting varaibles
    var difficulty: Int = 1///1=easy, 2=medium, 3=hard
    var ballColor: String = "red"
    var backgroundColor: UIColor = UIColor.systemGray2
    var backgroundColorString = "grey"
```

Used to change UIColor into persistent storage



```
func colorToString() -> String{
    if(backgroundColor == UIColor.green){
        backgroundColorString = "green"
        return "green"
    }
    else if(backgroundColor == UIColor.blue){
        backgroundColorString = "blue"
        return "blue"
    }
    else if(backgroundColor == UIColor.red){
        backgroundColorString = "red"
        return "red"
    }
    else{
        backgroundColorString = "grey"
        return "grey"
    }
}

func stringToColor() -> UIColor{
    if(backgroundColorString == "green"){
        backgroundColor = UIColor.green
        return UIColor.green
    }
    else if(backgroundColorString == "blue"){
        backgroundColor = UIColor.blue
        return UIColor.blue
    }
    else if(backgroundColorString == "red"){
        backgroundColor = UIColor.red
        return UIColor.red
    }
    else{
        backgroundColor = UIColor.systemGray2
        return UIColor.systemGray2
    }
}
```

Maze View Controller



```
class MazeViewController: UIViewController {

    var appDelegate: AppDelegate?
    var myModel: RollingLabyrinthModel?


    @IBOutlet weak var timeLabel: UILabel!
    @IBOutlet weak var startButton: UIButton!
    @IBOutlet weak var MazeSKView: SKView!

    var timer : Timer = Timer()
    var count : Int = 0
    // myModel.timerCounting

    @IBAction func startButton(_ sender: Any){
        timerButtonPressed()
    }

    func timerButtonPressed(){
        if let tmpModel = myModel{
            // Stop Timer
            if(tmpModel.timerCounting){
                tmpModel.timerCounting = false
                self.count = 0
                self.timer.invalidate()
                self.timeLabel.text = makeTimeString(minutes: 0, seconds: 0)
                startButton.setTitle("Start Game", for: .normal)

            }
            // Start Timer
            else{
                tmpModel.gamesPlayed += 1
                tmpModel.timerCounting = true
                startButton.setTitle("Stop Game", for: .normal)
                timer = Timer.scheduledTimer(timeInterval: 1, target: self, selector: #selector(timerCounter), userInfo: nil, repeats: true)
            }
        }
    }
```

```
@objc func timerCounter() -> Void{
    count = count + 1
    let time = secondsToMinutesSeconds(seconds: count)
    let timeString = makeTimeString(minutes: time.0, seconds: time.1)
    if let tmpModel = myModel{
        tmpModel.savedTimerInt = count
        tmpModel.savedTimerString = timeString

    }
    timeLabel.text = timeString
}
```

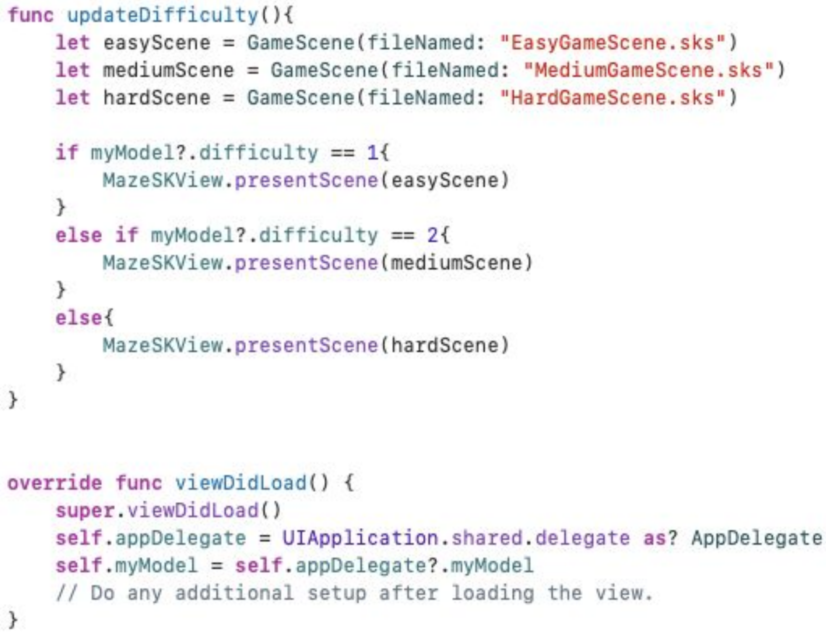
Used For Updating High Scores

```
func secondsToMinutesSeconds(seconds: Int) -> (Int,Int){
    return ((seconds % 3600)/60,(seconds%36000)%60)
}
```

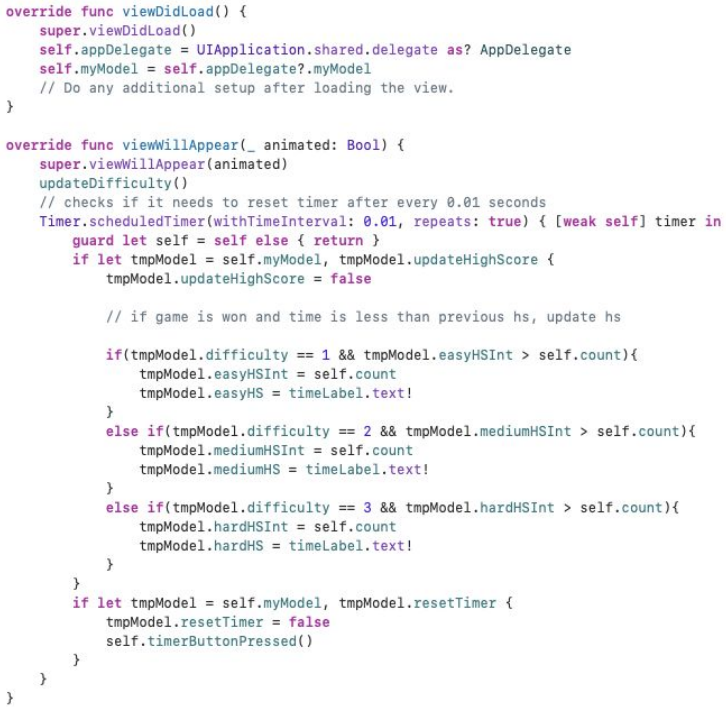
Tuple Data Structure

```
func makeTimeString(minutes: Int, seconds: Int) -> String{
    var timeString = ""
    timeString += String(format: "%02d",minutes)
    timeString += ":"
    timeString += String(format: "%02d", seconds)
    return timeString
}
```

Maze View Controller

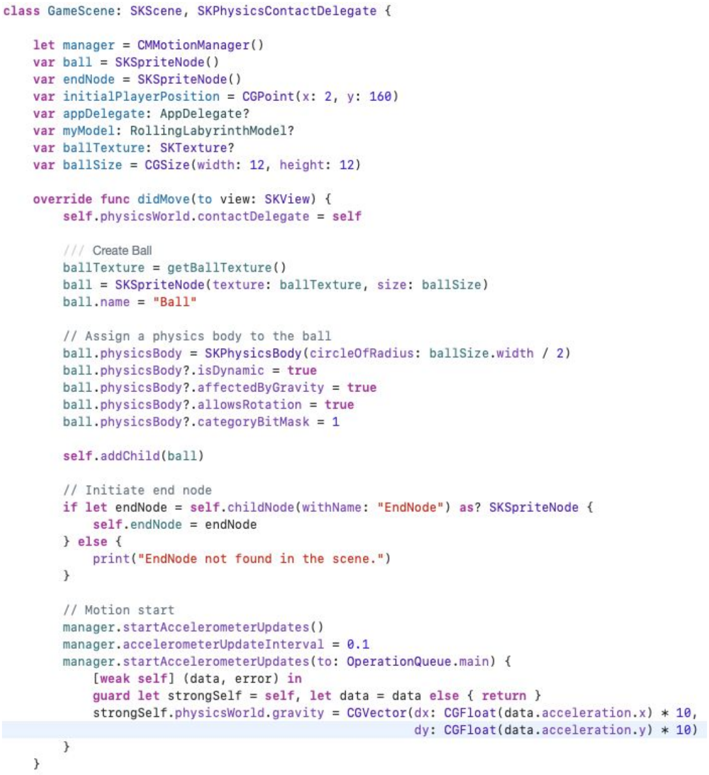


```swift
func updateDifficulty(){
    let easyScene = GameScene(fileNamed: "EasyGameScene.sks")
    let mediumScene = GameScene(fileNamed: "MediumGameScene.sks")
    let hardScene = GameScene(fileNamed: "HardGameScene.sks")

    if myModel?.difficulty == 1{
        MazeSKView.presentScene(easyScene)
    }
    else if myModel?.difficulty == 2{
        MazeSKView.presentScene(mediumScene)
    }
    else{
        MazeSKView.presentScene(hardScene)
    }
}


override func viewDidLoad() {
    super.viewDidLoad()
    self.appDelegate = UIApplication.shared.delegate as? AppDelegate
    self.myModel = self.appDelegate?.myModel
    // Do any additional setup after loading the view.
}
```

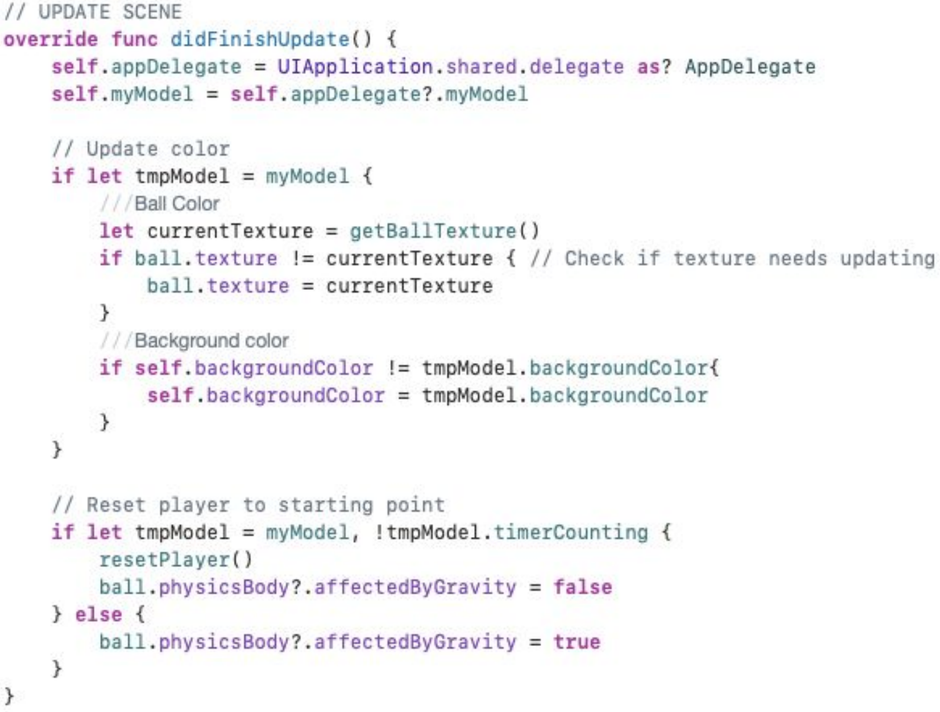


```swift
override func viewDidLoad() {
    super.viewDidLoad()
    self.appDelegate = UIApplication.shared.delegate as? AppDelegate
    self.myModel = self.appDelegate?.myModel
    // Do any additional setup after loading the view.
}

override func viewWillAppear(_ animated: Bool) {
    super.viewWillAppear(animated)
    updateDifficulty()
    // checks if it needs to reset timer after every 0.01 seconds
    Timer.scheduledTimer(withTimeInterval: 0.01, repeats: true) { [weak self] timer in
        guard let self = self else { return }
        if let tmpModel = self.myModel, tmpModel.updateHighScore {
            tmpModel.updateHighScore = false

            // if game is won and time is less than previous hs, update hs

            if(tmpModel.difficulty == 1 && tmpModel.easyHSInt > self.count){
                tmpModel.easyHSInt = self.count
                tmpModel.easyHS = timeLabel.text!
            }
            else if(tmpModel.difficulty == 2 && tmpModel.mediumHSInt > self.count){
                tmpModel.mediumHSInt = self.count
                tmpModel.mediumHS = timeLabel.text!
            }
            else if(tmpModel.difficulty == 3 && tmpModel.hardHSInt > self.count){
                tmpModel.hardHSInt = self.count
                tmpModel.hardHS = timeLabel.text!
            }
        }
        if let tmpModel = self.myModel, tmpModel.resetTimer {
            tmpModel.resetTimer = false
            self.timerButtonPressed()
        }
    }
}
```

Game Scene (Sprite Kit & Core Motion)

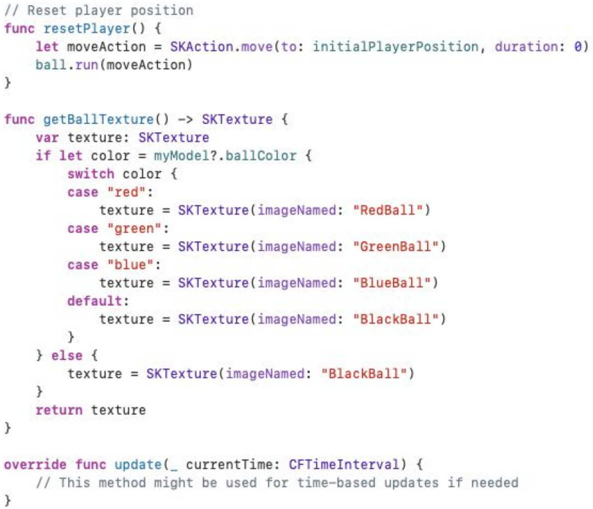


```
class GameScene: SKScene, SKPhysicsContactDelegate {

    let manager = CMMotionManager()
    var ball = SKSpriteNode()
    var endNode = SKSpriteNode()
    var initialPlayerPosition = CGPoint(x: 2, y: 160)
    var appDelegate: AppDelegate?
    var myModel: RollingLabyrinthModel?
    var ballTexture: SKTexture?
    var ballSize = CGSize(width: 12, height: 12)

    override func didMove(to view: SKView) {
        self.physicsWorld.contactDelegate = self

        /// Create Ball
        ballTexture = getBallTexture()
        ball = SKSpriteNode(texture: ballTexture, size: ballSize)
        ball.name = "Ball"

        // Assign a physics body to the ball
        ball.physicsBody = SKPhysicsBody(circleOfRadius: ballSize.width / 2)
        ball.physicsBody?.isDynamic = true
        ball.physicsBody?.affectedByGravity = true
        ball.physicsBody?.allowsRotation = true
        ball.physicsBody?.categoryBitMask = 1

        self.addChild(ball)

        // Initiate end node
        if let endNode = self.childNode(withName: "EndNode") as? SKSpriteNode {
            self.endNode = endNode
        } else {
            print("EndNode not found in the scene.")
        }

        // Motion start
        manager.startAccelerometerUpdates()
        manager.accelerometerUpdateInterval = 0.1
        manager.startAccelerometerUpdates(to: OperationQueue.main) {
            [weak self] (data, error) in
            guard let strongSelf = self, let data = data else { return }
            strongSelf.physicsWorld.gravity = CGVector(dx: CGFloat(data.acceleration.x) * 10,
                                                       dy: CGFloat(data.acceleration.y) * 10)
        }
    }
```

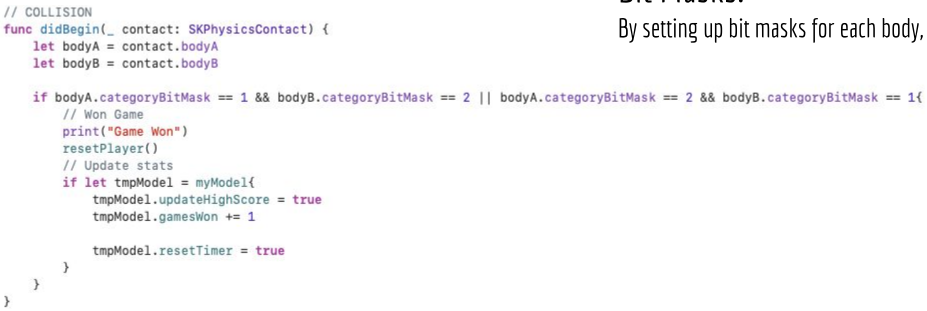


```
// UPDATE SCENE
override func didFinishUpdate() {
    self.appDelegate = UIApplication.shared.delegate as? AppDelegate
    self.myModel = self.appDelegate?.myModel

    // Update color
    if let tmpModel = myModel {
        ///Ball Color
        let currentTexture = getBallTexture()
        if ball.texture != currentTexture { // Check if texture needs updating
            ball.texture = currentTexture
        }
        ///Background color
        if self.backgroundColor != tmpModel.backgroundColor{
            self.backgroundColor = tmpModel.backgroundColor
        }
    }

    // Reset player to starting point
    if let tmpModel = myModel, !tmpModel.timerCounting {
        resetPlayer()
        ball.physicsBody?.affectedByGravity = false
    } else {
        ball.physicsBody?.affectedByGravity = true
    }
}
```

Game Scene (Sprite Kit & Core Motion)



```
// Reset player position
func resetPlayer() {
    let moveAction = SKAction.move(to: initialPlayerPosition, duration: 0)
    ball.run(moveAction)
}

func getBallTexture() -> SKTexture {
    var texture: SKTexture
    if let color = myModel?.ballColor {
        switch color {
        case "red":
            texture = SKTexture(imageNamed: "RedBall")
        case "green":
            texture = SKTexture(imageNamed: "GreenBall")
        case "blue":
            texture = SKTexture(imageNamed: "BlueBall")
        default:
            texture = SKTexture(imageNamed: "BlackBall")
        }
    } else {
        texture = SKTexture(imageNamed: "BlackBall")
    }
    return texture
}

override func update(_ currentTime: CFTimeInterval) {
    // This method might be used for time-based updates if needed
}
```

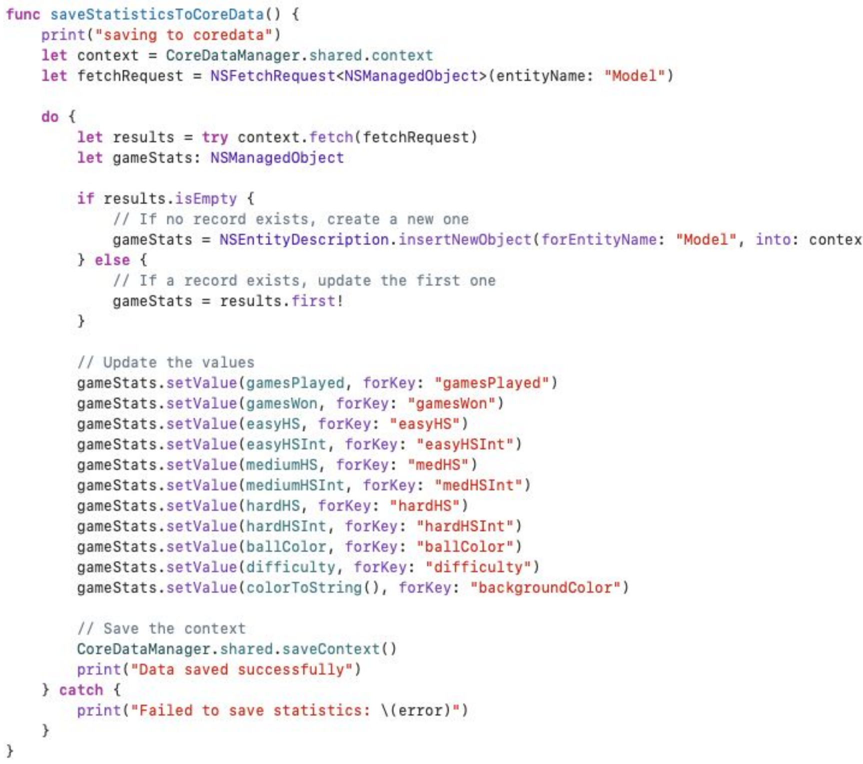


```
// COLLISION
func didBegin(_ contact: SKPhysicsContact) {
    let bodyA = contact.bodyA
    let bodyB = contact.bodyB

    if bodyA.categoryBitMask == 1 && bodyB.categoryBitMask == 2 || bodyA.categoryBitMask == 2 && bodyB.categoryBitMask == 1{
        // Won Game
        print("Game Won")
        resetPlayer()
        // Update stats
        if let tmpModel = myModel{
            tmpModel.updateHighScore = true
            tmpModel.gamesWon += 1

            tmpModel.resetTimer = true
        }
    }
}
```

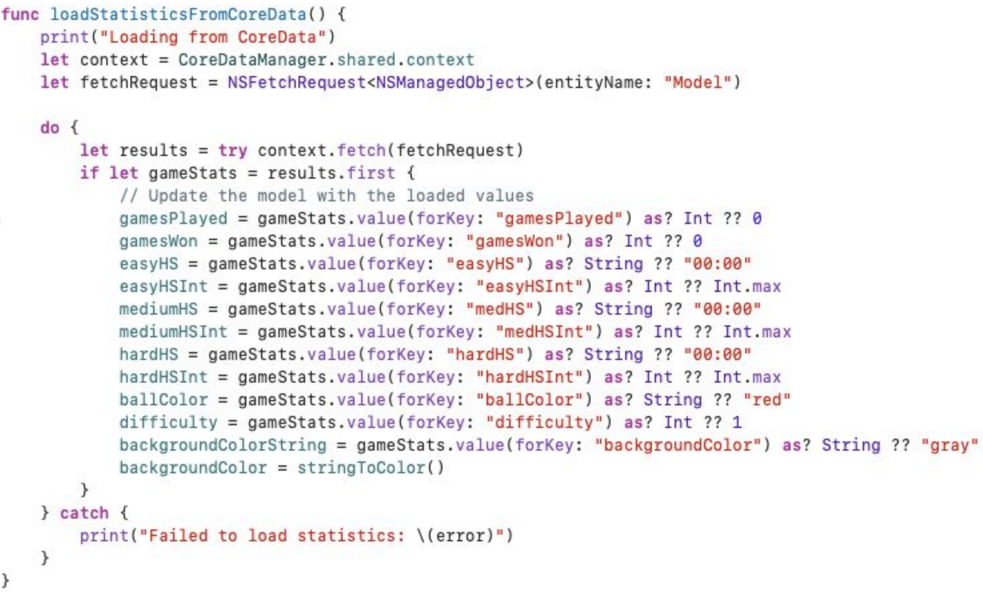
Bit Masks:

By setting up bit masks for each body, the programmer can detect when sprites of certain bitmasks collide and/or make contact with each other

Core Data

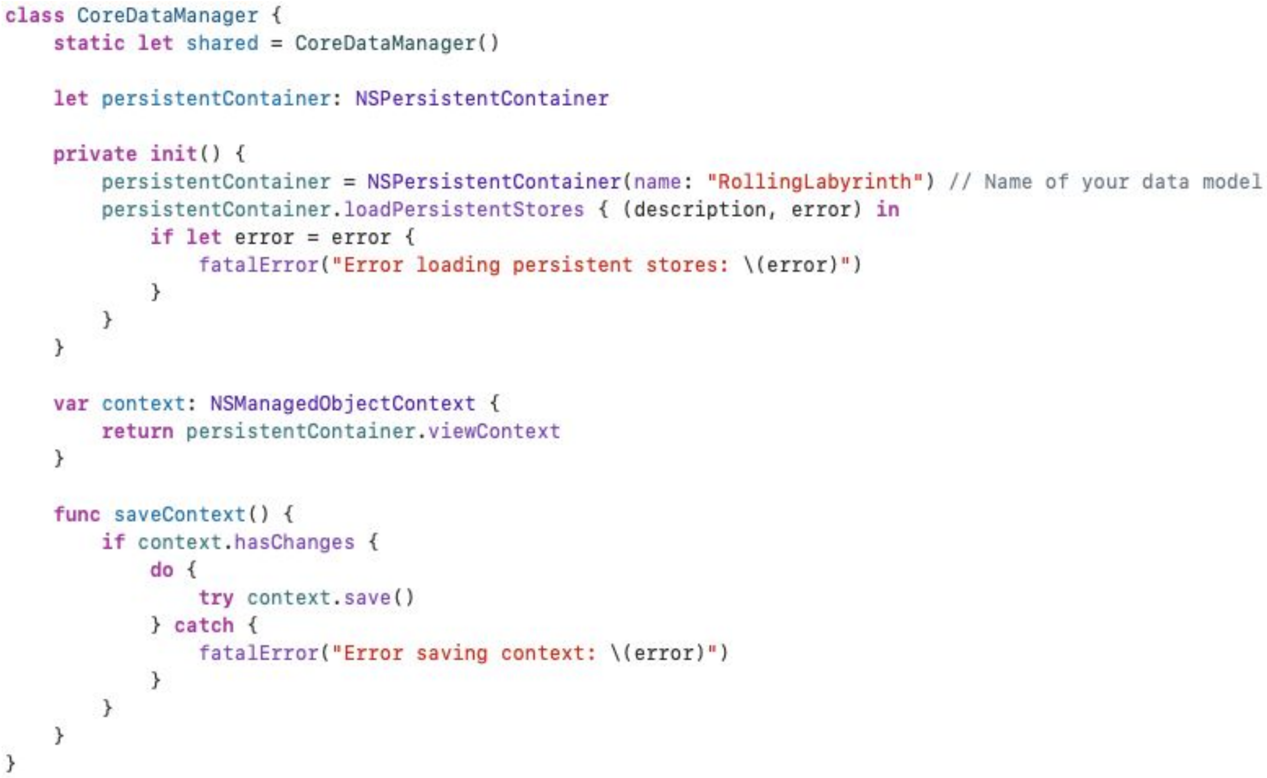


```swift
func saveStatisticsToCoreData() {
    print("saving to coredata")
    let context = CoreDataManager.shared.context
    let fetchRequest = NSFetchRequest<NSManagedObject>(entityName: "Model")

    do {
        let results = try context.fetch(fetchRequest)
        let gameStats: NSManagedObject

        if results.isEmpty {
            // If no record exists, create a new one
            gameStats = NSEntityDescription.insertNewObject(forEntityName: "Model", into: contex
        } else {
            // If a record exists, update the first one
            gameStats = results.first!
        }

        // Update the values
        gameStats.setValue(gamesPlayed, forKey: "gamesPlayed")
        gameStats.setValue(gamesWon, forKey: "gamesWon")
        gameStats.setValue(easyHS, forKey: "easyHS")
        gameStats.setValue(easyHSInt, forKey: "easyHSInt")
        gameStats.setValue(mediumHS, forKey: "medHS")
        gameStats.setValue(mediumHSInt, forKey: "medHSInt")
        gameStats.setValue(hardHS, forKey: "hardHS")
        gameStats.setValue(hardHSInt, forKey: "hardHSInt")
        gameStats.setValue(ballColor, forKey: "ballColor")
        gameStats.setValue(difficulty, forKey: "difficulty")
        gameStats.setValue(colorToString(), forKey: "backgroundColor")

        // Save the context
        CoreDataManager.shared.saveContext()
        print("Data saved successfully")
    } catch {
        print("Failed to save statistics: \(error)")
    }
}
```



```swift
func loadStatisticsFromCoreData() {
    print("Loading from CoreData")
    let context = CoreDataManager.shared.context
    let fetchRequest = NSFetchRequest<NSManagedObject>(entityName: "Model")

    do {
        let results = try context.fetch(fetchRequest)
        if let gameStats = results.first {
            // Update the model with the loaded values
            gamesPlayed = gameStats.value(forKey: "gamesPlayed") as? Int ?? 0
            gamesWon = gameStats.value(forKey: "gamesWon") as? Int ?? 0
            easyHS = gameStats.value(forKey: "easyHS") as? String ?? "00:00"
            easyHSInt = gameStats.value(forKey: "easyHSInt") as? Int ?? Int.max
            mediumHS = gameStats.value(forKey: "medHS") as? String ?? "00:00"
            mediumHSInt = gameStats.value(forKey: "medHSInt") as? Int ?? Int.max
            hardHS = gameStats.value(forKey: "hardHS") as? String ?? "00:00"
            hardHSInt = gameStats.value(forKey: "hardHSInt") as? Int ?? Int.max
            ballColor = gameStats.value(forKey: "ballColor") as? String ?? "red"
            difficulty = gameStats.value(forKey: "difficulty") as? Int ?? 1
            backgroundColorString = gameStats.value(forKey: "backgroundColor") as? String ?? "gray"
            backgroundColor = stringToColor()
        }
    } catch {
        print("Failed to load statistics: \(error)")
    }
}
```

Core Data



```swift
class CoreDataManager {
    static let shared = CoreDataManager()

    let persistentContainer: NSPersistentContainer

    private init() {
        persistentContainer = NSPersistentContainer(name: "RollingLabyrinth") // Name of your data model
        persistentContainer.loadPersistentStores { (description, error) in
            if let error = error {
                fatalError("Error loading persistent stores: \(error)")
            }
        }
    }

    var context: NSManagedObjectContext {
        return persistentContainer.viewContext
    }

    func saveContext() {
        if context.hasChanges {
            do {
                try context.save()
            } catch {
                fatalError("Error saving context: \(error)")
            }
        }
    }
}
```

Thank you!

Time for demo!