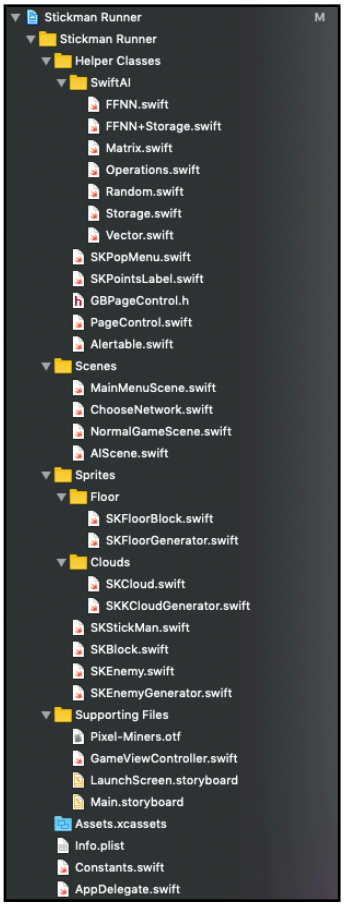
Criterion C: Development

Overview

Martian Runner is a swift based iOS application to help middle school students learn about artificial intelligence. The app can be installed from the app store or the GitHub link provided on the cover page.

Project Structure

The project is contained inside a single Xcode workspace file named Stickman Runner.xcworkspace (the project was initially titled "Stickman Runner"), with 32 individual files divided into: Helper Classes, Scenes, Sprites, and Supporting Files. The Helper Classes folder contains classes that help the application do specific tasks, such as SKPopMenu.swift, which controls the displaying of a user-input popup. The Scenes folder contains all the different scenes in the application. The Sprites folder contains all the different sprites in the game. The Supporting Files folder contains all the files that are required for the application to run but do not require constant editing, such as the Pixel-Miners.otf file that contains the main font used in the app.



Application Structure

GameViewController.swift

```
class GameViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()

        let scene = MainMenuScene(size: view.bounds.size)
        let skView = view as! SKView

        skView.ignoresSiblingOrder = true
        scene.scaleMode = .resizeFill
        skView.presentScene(scene)
    }
```

The app is controlled by a central view controller, titled "GameViewController.swift." It's purpose is simply to display different scenes, when it is initialized as the app launches, it displays MainMenuScene.swift.

MainMenuScene.swift

```
class MainMenuScene : SKScene, Alertable {

    let normalLabel = SKLabelNode()
    let AILabel = SKLabelNode()
    let highScoreLabel = SKLabelNode(fontNamed: "Pixel Miners")

    let instructionLabel = SKLabelNode()
    let mainLabel = SKSpriteNode(imageNamed: "Martian-Runner")
    var musicButton = SKSpriteNode()
    let background = SKSpriteNode(imageNamed: "bg")
```

MaineMenuScene inherits from Apple's SKScene class and my custom Alertable.swift class. It simply acts as a menu screen to the game. The class begins by initializing the different labels and node necessary for the menu. All the visible objects on the screen in this scene, and every other scene in the project, are of type SKNode which is a type provided in Apple's SpriteKit framework which I am using to build this game.

More on *MainMenuScene.swift*

I opted out of using buttons at all in my project, instead I used labels that are touch sensitive. Each label node is given a specific name, such as "choose" for the "Choose Network" label, then the touchesBegan function handles the rest. As shown on the right, the touchesBegan method takes a touch on the screen as input and detects where the touch is and what node was pressed. This function is used throughout the project to detect touches on the screen.

```
override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?){

    let touch:UITouch = touches.first! as UITouch
    let positionInScene = touch.location(in: self)
    let touchedNode = self.atPoint(positionInScene)
    if let name = touchedNode.name {
        if name == "choose" {
            let scene = ChooseNetwork(size: size)
            self.view?.presentScene(scene)
        } else if name == "Normal" {
            showAlert(withTitle: "Name", message: "Enter your name:")
        } else if name == "learn" {
            let scene = AISceneWithInstructions(size: size)
            self.view?.presentScene(scene)
        }
```

Alertable.swift

```
alertController.addAction(UIAlertAction(title: "OK", style: .default, handler: { (action) -> Void in
    let textField = alertController.textFields![0] as UITextField

    if(alertController.title == "Remove Network") {
        let defaults = UserDefaults.standard
        defaults.removeObject(forKey: textField.text ?? "")
        let scene = ChooseNetwork(size: self.size)
        self.view?.presentScene(scene)
    } else {
        if(textField.text != "") {
            print("Text field: \(textField.text ?? "")")
            currentName = textField.text!
            let scene = NormalGameScene(size: self.size)
            self.view?.presentScene(scene)
        }
    }
```

This class is a custom helper class that displays tailored alerts for my application. Since this app was developed in SpriteKit and uses scenes instead of UIViewControllers, displaying alerts natively is quite difficult. Therefore, I opted for writing a custom class that extends SKScene in order to add the ability to display alerts. This method in Alertable is used to make alerts that allow for user input. It is tailored for the two scenarios that alerts are shown in the application: to input a neural network's name and when removing a specific neural network. This class provides code for both scenario and takes the user to the respective scene. For example, in the case of removing a neural network, the class pushes a new ChooseNetwork scene to view in order to refresh the scene. This class was adapted from user crashoverride777's code on stackoverflow.com.

NormalGameScene.swift

NormalGameScene is presented when the user taps on "Train Network." It has a similar architecture to that of *MainMenuScene,* in that it extends SKScene and displays nodes on to the screen using SKNode. In this class the most important variables and methods are shown below:

```
var mainHero: SKStickMan!
var enemyGenerator: SKEnemyGenerator!

var floorGenerator: SKFloorGenerator!
var cloudGenerator: SKCloudGenerator!
```

These variables define the main sprites of the game, all custom classes that I wrote. All the variables, except mainHero, call on generator classes instead of individual sprite classes. This is because these sprites all move across the screen and are therefore generic. The generator classes creates new sprites as deemed necessary. This is explained below.

```
var floorBlocks = [SKFloorBlock]()

init(size: CGSize) {
    super.init(texture: nil, color: UIColor.clear, size: CGSize(width: size.width, height: size.height))
    anchorPoint = CGPoint(x:-size.width, y:0)

    for i in 0 ..< 100 {

        let newBlock = SKFloorBlock(type: "brick")
        newBlock.position = CGPoint(x: CGFloat(i) * newBlock.size.width, y: 0)
        addChild(newBlock)
        floorBlocks.append(newBlock)


    }
}

 func start() {

     var adjustedDuration: TimeInterval
     {
         get {
             return TimeInterval(size.width/xPerSec)
         }
     }

     //Moves ground left by one whole screen
     var moveLeft = SKAction.moveBy(x: -(size.width)*2, y: 0, duration: adjustedDuration)

     //Teleports ground back to initial position
     let resetPosition = SKAction.moveTo(x: -size.width , duration: 0)


     //Creates a sequence that combines above to actions
     let mySequence = SKAction.sequence([moveLeft,resetPosition])

     //Runs the sequence forever
     run(SKAction.repeatForever(mySequence))
 }
```

SKFloorBrick

This the SKFloorGenerator.swift class that is called in *NormalGameScene.swift.* Its purpose, like the rest of the generator classes, is to generate child nodes and remove them when necessary. When initialized, it creates an array floorBlocks of SKFloorBlock, the child class, and initialized 100 SKFloorBricks. When called, the SKFloorBrick creates one brick node. The generator class creates the visual effect of a scrolling ground by calling the start() function and moving the array of sprites across the screen during the time adjustedDuration, which decreases as the game progresses thus speeding the ground up.

As described in the annotation of the SKFloorGenerator photo, the generator-child class architecture is used throughout the project. Game scenes never directly call my custom SKSprite classes, other than in the case of the main character, they call the generator classes that encapsulate the generation of sprites. A generator was not necessary in the case of the main character because he is not a generic type and there is only one of him displayed per scene.

```
var highScore = UserDefaults.standard.integer(forKey: "highscore")
```

The highScore variable is stored using the UserDefaults class provided by Apple for persistent storage. UserDefaults is used throughout the game for storing the high score and saved neural networks. UserDefaults provides a dictionary data structure that stores data with a key, in this case the key is "highscore."

```
@objc func checkScore() {

    if enemyGenerator.allEnemies.count > 0 {
        let enemyPosition = enemyGenerator.convert(enemyGenerator.allEnemies[0].position, to: self)
        print(closestEnemyXPos)
        closestEnemyXPos = enemyPosition.x
        if (closestEnemyXPos < mainHero.position.x)
        {

            enemyGenerator.allEnemies.remove(at: 0)
            scoreLabel.increment()

            if(scoreLabel.number % 3 == 0) {

                floorGenerator.stop()
                floorGenerator.start()


                enemyGenerator.restart()

                LevelNumber += 1
                backgroundColor = .random()
            }
        }

    }
}
```

The checkScore() method is called every second by the update() method (which runs every time the scene updates). It checks the score by measuring if the main hero has a higher x position than that of the closest Enemy, meaning that the main hero successfully jumped over the enemy. It also handles the incrementing of the speed and difficulty of the game and the background color (which is randomized every 3 points).

Neural Network

```
var currentNeuralNetwork = FFNN(inputs: 1, hidden: 300, outputs: 1)

//AI Stuff
var parameters: [[Float]] = []
var indexArray: [Float] = []
var neuralAnswers: [[Float]] = []
```

These variables deal with the neural network aspect of the game. As explained in criteria B, the parameters array collects the x position of the nearest enemy each time the user jumps and the neuralAnswers array records either a 0 or a 1 every time update() runs (1 is collected if the user jumps, 0 is collected if user is not currently jumping.) IndexArray is simply used to index the arrays. The currentNeuralNetwork variable initializes a neuralNetwork from the FFNN class provided by SwiftAI with 1 input, 300 hidden nodes, and one output.

```
func gameOver() {
    cloudGenerator.stopGeneratingMoreClouds()
    let defaults = UserDefaults.standard
    if highScore < scoreLabel.number  {
        highScore = scoreLabel.number

        defaults.set(highScore, forKey: "highscore")
    }

    print("High Score: \(UserDefaults.standard.integer(forKey: "highscore"))")

    let res = parameters.cleanUp(withAnswers: neuralAnswers)

    parameters = res.this
    neuralAnswers = res.answers

    print(neuralAnswers)
    _ = try! currentNeuralNetwork.train(inputs: parameters, answers: neuralAnswers, testInputs: parameters,
        testAnswers: neuralAnswers, errorThreshold: 0.02)

    print("weights\(currentNeuralNetwork.getWeights())")

    defaults.set(currentNeuralNetwork.getWeights(), forKey: currentName)

    print("going to AI Scene now")

    enemyGenerator.onCollision()
    closestEnemyXPos = 0

    LevelNumber = 0
    likelyhoodOfWater = 0.01
    scoreTimerTime = 1
    xPerSec = 150.0

    let scene = ChooseNetwork(size: size)
    self.view?.presentScene(scene)
```

The gameOver method is called when the main character collides with an enemy. In encapsulates the training of the neural network. It first runs the arrays through the cleanUp() method, which simply equals out the data so there isn't repeats of xPositions. It then trains the currentNeuralNetwork using the arrays and saves the neuralNetwork to UserDefaults under the given name of the user. Finally it switches to the ChooseNetwork scene which allows a user to choose which network they want to use.

After a user chooses a network to use, the scene changes to AIScene.swift which is a modified copy of NormalGameScene.swift that instead of taking the user's touch as input for the main hero jumping, uses the neural network instead. The block of code shown on the right displays how the neural network makes the character jumps. It is activated when an enemy crosses the half mark on the ground, it then creates a variable networkValue that is between 0 and 1; the closer to 1, the more certain it is of the jump. If the value is greater than 0.5, it applies a force to the mainHero, making him jump.

```
if(enemyPosition.x < 150) {
    let networkValue = try! currentNeuralNetwork.update(inputs: [Float(closestEnemyXPos)]).first!
    let networkWantsToJump = networkValue > 0.99
    print("Network value: \(networkValue) \(networkWantsToJump)")
    if networkWantsToJump && isOnGround{
        mainHero.physicsBody?.applyForce(CGVector(dx: 0, dy: 20_000))
        isOnGround = false
    } else if !(networkWantsToJump) {
        isOnGround = true
    }
}
```

External Code Used

- Collin Hundley for SwiftAI (https://github.com/Swift-AI/Swift-AI)
- crashoverride777 for Alert code in Spritekit (https://stackoverflow.com/questions/39557344/swift-spritekit-how-to-present-alert-view-in-gamescene)

Word Count: 1213