Assessment number: J100221

ViewController.swift file:

```
import UIKit

import AVFoundation

/* - Manages the user interface

  - Connects user actions to the tennis match model

  - Displays the current match state and additional features */

class ViewController: UIViewController {

  // MARK: - UI Outlets

  @IBOutlet weak var p1Button: UIButton!

  @IBOutlet weak var p2Button: UIButton!

  @IBOutlet weak var p1NameLabel: UILabel!

  @IBOutlet weak var p2NameLabel: UILabel!

  @IBOutlet weak var p1PointsLabel: UILabel!

  @IBOutlet weak var p2PointsLabel: UILabel!

  @IBOutlet weak var p1GamesLabel: UILabel!

  @IBOutlet weak var p2GamesLabel: UILabel!

  @IBOutlet weak var p1SetsLabel: UILabel!

  @IBOutlet weak var p2SetsLabel: UILabel!

@IBOutlet weak var p1PreviousSetsLabel: UILabel!

@IBOutlet weak var p2PreviousSetsLabel: UILabel!

  // MARK: - New UI Elements

  private var locationLabel: UILabel!

  private var futureMatchTitleLabel: UILabel!

  private var futureMatchLabel: UILabel!

  private var futureMatchLocationLabel: UILabel!

  private var historyButton: UIButton!

  private var scheduleButton: UIButton!
```

```
// MARK: - Properties

// The match model ( for tennis scoring)

private var match = TennisMatch()

// Track the total games played for "new balls please" messages

private var totalGamesPlayed = 0

// Sound player for audio cues

private var audioPlayer: AVAudioPlayer?

// Colors for highlighting

private let serverColor = UIColor.purple

private let pointColor = UIColor.green

private let defaultColor = UIColor.white

// External screen support

private var externalWindow: UIWindow?

private var externalViewController: ViewController?

// Current location

private var currentLocation: String = "Unknown location"

// MARK: - Lifecycle Methods

override func viewDidLoad() {

  super.viewDidLoad()

  // Setup the initial UI state

  resetMatch()

  // Load sound file

  setupAudioPlayer()

  // Setup external display if available

  setupExternalScreen()

  // Add new UI elements

  setupEnhancedUI()

  // Start location services

  startLocationServices()
```

```
    // Register for screen connection notifications
    NotificationCenter.default.addObserver(
        self,
        selector: #selector(screenDidConnect),
        name: UIScreen.didConnectNotification,
        object: nil
    )
    NotificationCenter.default.addObserver(
        self,
        selector: #selector(screenDidDisconnect),
        name: UIScreen.didDisconnectNotification,
        object: nil
    )
}
override func viewWillAppear(_ animated: Bool) {
    super.viewWillAppear(animated)
    // Update location and future match information when the view appears
    updateLocationDisplay()
    updateFutureMatchDisplay()
}
deinit {
    NotificationCenter.default.removeObserver(self) // Remove any observers to avoid memory leaks
}
// MARK: - Enhanced UI Setup
/* - Sets up new UI elements for enhanced functionality
   - Creates a visually appealing container with match information,
   - location services, and buttons for additional features */
private func setupEnhancedUI() {
    // Create container view with improved visual style
```

```
let enhancedContainer = UIView()
enhancedContainer.translatesAutoresizingMaskIntoConstraints = false
enhancedContainer.backgroundColor = .systemBackground
enhancedContainer.layer.borderWidth = 0.5 // Add a border
enhancedContainer.layer.borderColor = UIColor.systemGray4.cgColor // border color
enhancedContainer.layer.cornerRadius = 16 //  round corners
enhancedContainer.layer.shadowColor = UIColor.black.cgColor  // Add shadow effect
enhancedContainer.layer.shadowOffset = CGSize(width: 0, height: 3) // Shadow offset
enhancedContainer.layer.shadowRadius = 6 // Shadow radius
enhancedContainer.layer.shadowOpacity = 0.15 // Adjust shadow opacity
enhancedContainer.layer.masksToBounds = false // Allow shadow to overflow outside bounds
view.addSubview(enhancedContainer) // Add the container view to the main view
// Position the container below the score display elements
NSLayoutConstraint.activate([
    enhancedContainer.topAnchor.constraint(equalTo: view.safeAreaLayoutGuide.topAnchor, constant: 200),
    enhancedContainer.leadingAnchor.constraint(equalTo: view.leadingAnchor, constant: 16),
    enhancedContainer.trailingAnchor.constraint(equalTo: view.trailingAnchor, constant: -16),
    enhancedContainer.heightAnchor.constraint(equalToConstant: 269)
])
// Create a title label for this section with gradient background
let titleContainerView = UIView()
titleContainerView.translatesAutoresizingMaskIntoConstraints = false
titleContainerView.backgroundColor = .systemBlue //background color
titleContainerView.layer.cornerRadius = 16 // Round corners
titleContainerView.layer.maskedCorners = [.layerMinXMinYCorner, .layerMaxXMinYCorner] // Round only top corners
titleContainerView.clipsToBounds = true // Ensure corners are clipped to fit
enhancedContainer.addSubview(titleContainerView) //Title container
// Create and set up a gradient layer for the title background
let gradientLayer = CAGradientLayer()
```

```
gradientLayer.colors = [
    UIColor.systemBlue.cgColor,
    UIColor(red: 0, green: 0.5, blue: 0.9, alpha: 1.0).cgColor // Gradient from blue to light blue
]
gradientLayer.startPoint = CGPoint(x: 0, y: 0)
gradientLayer.endPoint = CGPoint(x: 1, y: 1)
gradientLayer.locations = [0, 1]
titleContainerView.layer.insertSublayer(gradientLayer, at: 0)
// Add icon to the title container
let titleIconView = UIImageView(image: UIImage(systemName: "sportscourt.fill"))
titleIconView.translatesAutoresizingMaskIntoConstraints = false
titleIconView.contentMode = .scaleAspectFit // Scale icon to fit
titleIconView.tintColor = .white // Set icon color to white
titleContainerView.addSubview(titleIconView)
// Create and set up title label for the section
let titleLabel = UILabel()
titleLabel.translatesAutoresizingMaskIntoConstraints = false
titleLabel.text = "Match Information"
titleLabel.font = UIFont.systemFont(ofSize: 16, weight: .bold)
titleLabel.textAlignment = .center // Center-align the title text
titleLabel.textColor = .white
titleContainerView.addSubview(titleLabel)
// Create and set up a horizontal stack for location display
let locationStack = UIStackView()
locationStack.translatesAutoresizingMaskIntoConstraints = false
locationStack.axis = .horizontal
locationStack.spacing = 8 // Add spacing between elements
locationStack.alignment = .center
enhancedContainer.addSubview(locationStack)
```

```
// Create and set up location icon
let locationIcon = UIImageView(image: UIImage(systemName: "mappin.circle.fill"))
locationIcon.translatesAutoresizingMaskIntoConstraints = false
locationIcon.tintColor = .systemRed
locationIcon.contentMode = .scaleAspectFit
locationStack.addArrangedSubview(locationIcon)
// Create and set up location label
locationLabel = UILabel()
locationLabel.translatesAutoresizingMaskIntoConstraints = false
locationLabel.text = "Location: Determining..." // Default text for location
locationLabel.font = UIFont.systemFont(ofSize: 14)
locationStack.addArrangedSubview(locationLabel)
// Create divider line for separation
let divider = UIView()
divider.translatesAutoresizingMaskIntoConstraints = false
divider.backgroundColor = UIColor.systemGray5 // Set divider color to light gray
divider.layer.cornerRadius = 0.5
enhancedContainer.addSubview(divider)
// Create and set up a vertical stack for the next match section
let nextMatchStack = UIStackView()
nextMatchStack.translatesAutoresizingMaskIntoConstraints = false
nextMatchStack.axis = .vertical
nextMatchStack.spacing = 10
enhancedContainer.addSubview(nextMatchStack)
// Header for the "Next Match" section with calendar icon
let nextMatchHeaderStack = UIStackView()
nextMatchHeaderStack.translatesAutoresizingMaskIntoConstraints = false
nextMatchHeaderStack.axis = .horizontal
nextMatchHeaderStack.spacing = 8
```

```
nextMatchHeaderStack.alignment = .center

// Create background for the header

let headerBackground = UIView()

headerBackground.translatesAutoresizingMaskIntoConstraints = false

headerBackground.backgroundColor = UIColor.systemBlue.withAlphaComponent(0.1) // light blue background

headerBackground.layer.cornerRadius = 8 // Round corners

enhancedContainer.addSubview(headerBackground)

// Create and set up calendar icon for the header

let calendarIcon = UIImageView(image: UIImage(systemName: "calendar"))

calendarIcon.translatesAutoresizingMaskIntoConstraints = false

calendarIcon.tintColor = .systemBlue // calendar icon color

calendarIcon.contentMode = .scaleAspectFit // Scale icon to fit

nextMatchHeaderStack.addArrangedSubview(calendarIcon)

// Create and set up title for the next match header

futureMatchTitleLabel = UILabel()

futureMatchTitleLabel.translatesAutoresizingMaskIntoConstraints = false

futureMatchTitleLabel.text = "Next Match:"

futureMatchTitleLabel.font = UIFont.systemFont(ofSize: 14, weight: .semibold)

futureMatchTitleLabel.textColor = .systemBlue //text color

nextMatchHeaderStack.addArrangedSubview(futureMatchTitleLabel)

// Add header stack to next match stack

nextMatchStack.addArrangedSubview(nextMatchHeaderStack)

// Improved match info with card-like container

let matchInfoContainer = UIView()

matchInfoContainer.translatesAutoresizingMaskIntoConstraints = false

matchInfoContainer.backgroundColor = UIColor.systemGray6

matchInfoContainer.layer.cornerRadius = 8

matchInfoContainer.layer.borderWidth = 0.5

matchInfoContainer.layer.borderColor = UIColor.systemGray5.cgColor
```

```
enhancedContainer.addSubview(matchInfoContainer)

// Create and set up the label for displaying future match info

futureMatchLabel = UILabel()

futureMatchLabel.translatesAutoresizingMaskIntoConstraints = false

futureMatchLabel.text = "No upcoming match scheduled"

futureMatchLabel.font = UIFont.systemFont(ofSize: 13)

futureMatchLabel.numberOfLines = 2 // Allow two lines of text

futureMatchLabel.textAlignment = .left // Left-align text

matchInfoContainer.addSubview(futureMatchLabel)

// Create and set up a stack for displaying the future match location with a pin icon

let futureLocationStack = UIStackView()

futureLocationStack.translatesAutoresizingMaskIntoConstraints = false

futureLocationStack.axis = .horizontal

futureLocationStack.spacing = 4

futureLocationStack.alignment = .center

matchInfoContainer.addSubview(futureLocationStack)

// Set up location icon for future match location

let smallLocationIcon = UIImageView(image: UIImage(systemName: "mappin"))

smallLocationIcon.translatesAutoresizingMaskIntoConstraints = false

smallLocationIcon.tintColor = .systemBlue

smallLocationIcon.contentMode = .scaleAspectFit

futureLocationStack.addArrangedSubview(smallLocationIcon)

// Set up location label for future match

futureMatchLocationLabel = UILabel()

futureMatchLocationLabel.translatesAutoresizingMaskIntoConstraints = false

futureMatchLocationLabel.text = ""

futureMatchLocationLabel.font = UIFont.systemFont(ofSize: 13)

futureMatchLocationLabel.textColor = .systemBlue //text color

futureLocationStack.addArrangedSubview(futureMatchLocationLabel)
```

```
// Create a stack for buttons (History and Schedule)
let buttonStack = UIStackView()
buttonStack.translatesAutoresizingMaskIntoConstraints = false
buttonStack.axis = .horizontal
buttonStack.distribution = .fillEqually
buttonStack.spacing = 12
enhancedContainer.addSubview(buttonStack)
// History button with gradient background
historyButton = UIButton(type: .system)
historyButton.translatesAutoresizingMaskIntoConstraints = false
historyButton.setTitle("Match History", for: .normal) //button title
historyButton.setImage(UIImage(systemName: "clock.arrow.circlepath"), for: .normal)
historyButton.backgroundColor = .systemBlue //background color
historyButton.setTitleColor(.white, for: .normal) // title color
historyButton.layer.cornerRadius = 10
historyButton.titleLabel?.font = UIFont.systemFont(ofSize: 14, weight: .semibold) // Set font style and size
historyButton.contentEdgeInsets = UIEdgeInsets(top: 8, left: 12, bottom: 8, right: 12) // Adjust padding around content
historyButton.imageEdgeInsets = UIEdgeInsets(top: 0, left: -6, bottom: 0, right: 4) // Adjust icon position
historyButton.titleEdgeInsets = UIEdgeInsets(top: 0, left: 4, bottom: 0, right: -6) // Adjust title position
historyButton.contentHorizontalAlignment = .left
// Add shadow to button for a lifted effect
historyButton.layer.shadowColor = UIColor.systemBlue.withAlphaComponent(0.5).cgColor
historyButton.layer.shadowOffset = CGSize(width: 0, height: 2) // Set shadow offset
historyButton.layer.shadowRadius = 4
historyButton.layer.shadowOpacity = 0.5
historyButton.layer.masksToBounds = false // Ensure the shadow is visible outside the button's bounds
historyButton.addTarget(self, action: #selector(historyButtonTapped), for: .touchUpInside) // tap action
buttonStack.addArrangedSubview(historyButton)
// Schedule button with gradient background
```

```
scheduleButton = UIButton(type: .system)
scheduleButton.translatesAutoresizingMaskIntoConstraints = false
scheduleButton.setTitle("Schedule Match", for: .normal) // Set button title
scheduleButton.setImage(UIImage(systemName: "plus.circle"), for: .normal) // Set icon
scheduleButton.backgroundColor = .systemGreen
scheduleButton.setTitleColor(.white, for: .normal)
scheduleButton.layer.cornerRadius = 10
scheduleButton.titleLabel?.font = UIFont.systemFont(ofSize: 14, weight: .semibold)
scheduleButton.contentEdgeInsets = UIEdgeInsets(top: 8, left: 12, bottom: 8, right: 12)
scheduleButton.imageEdgeInsets = UIEdgeInsets(top: 0, left: -6, bottom: 0, right: 4)
scheduleButton.titleEdgeInsets = UIEdgeInsets(top: 0, left: 4, bottom: 0, right: -6)
// Add shadow to button for a lifted effect
scheduleButton.layer.shadowColor = UIColor.systemGreen.withAlphaComponent(0.5).cgColor
scheduleButton.layer.shadowOffset = CGSize(width: 0, height: 2)
scheduleButton.layer.shadowRadius = 4
scheduleButton.layer.shadowOpacity = 0.5
scheduleButton.layer.masksToBounds = false
scheduleButton.addTarget(self, action: #selector(scheduleButtonTapped), for: .touchUpInside)
buttonStack.addArrangedSubview(scheduleButton)
// Layout constraints for the UI components
NSLayoutConstraint.activate([
    // Title container takes full width
    titleContainerView.topAnchor.constraint(equalTo: enhancedContainer.topAnchor),
    titleContainerView.leadingAnchor.constraint(equalTo: enhancedContainer.leadingAnchor),
    titleContainerView.trailingAnchor.constraint(equalTo: enhancedContainer.trailingAnchor),
    titleContainerView.heightAnchor.constraint(equalToConstant: 42),
    // Title icon
    titleIconView.leadingAnchor.constraint(equalTo: titleContainerView.leadingAnchor, constant: 16),
    titleIconView.centerYAnchor.constraint(equalTo: titleContainerView.centerYAnchor),
```

```
titleIconView.widthAnchor.constraint(equalToConstant: 22),
titleIconView.heightAnchor.constraint(equalToConstant: 22),
// Title label
titleLabel.leadingAnchor.constraint(equalTo: titleIconView.trailingAnchor, constant: 8),
titleLabel.trailingAnchor.constraint(equalTo: titleContainerView.trailingAnchor, constant: -16),
titleLabel.centerYAnchor.constraint(equalTo: titleContainerView.centerYAnchor),
// Location stack
locationStack.topAnchor.constraint(equalTo: titleContainerView.bottomAnchor, constant: 16),
locationStack.leadingAnchor.constraint(equalTo: enhancedContainer.leadingAnchor, constant: 16),
locationStack.trailingAnchor.constraint(equalTo: enhancedContainer.trailingAnchor, constant: -16),
locationIcon.widthAnchor.constraint(equalToConstant: 20),
locationIcon.heightAnchor.constraint(equalToConstant: 20),
// Divider
divider.topAnchor.constraint(equalTo: locationStack.bottomAnchor, constant: 12),
divider.leadingAnchor.constraint(equalTo: enhancedContainer.leadingAnchor, constant: 12),
divider.trailingAnchor.constraint(equalTo: enhancedContainer.trailingAnchor, constant: -12),
divider.heightAnchor.constraint(equalToConstant: 1),
// Header background
headerBackground.topAnchor.constraint(equalTo: divider.bottomAnchor, constant: 12),
headerBackground.leadingAnchor.constraint(equalTo: enhancedContainer.leadingAnchor, constant: 12),
headerBackground.trailingAnchor.constraint(equalTo: enhancedContainer.trailingAnchor, constant: -12),
headerBackground.heightAnchor.constraint(equalToConstant: 32),
// Next match header stack
nextMatchHeaderStack.topAnchor.constraint(equalTo: divider.bottomAnchor, constant: 16),
nextMatchHeaderStack.leadingAnchor.constraint(equalTo: enhancedContainer.leadingAnchor, constant: 20),
calendarIcon.widthAnchor.constraint(equalToConstant: 18),
calendarIcon.heightAnchor.constraint(equalToConstant: 18),
// Match info container
matchInfoContainer.topAnchor.constraint(equalTo: headerBackground.bottomAnchor, constant: 8),
```

```
        matchInfoContainer.leadingAnchor.constraint(equalTo: enhancedContainer.leadingAnchor, constant: 16),
        matchInfoContainer.trailingAnchor.constraint(equalTo: enhancedContainer.trailingAnchor, constant: -16),
        matchInfoContainer.heightAnchor.constraint(equalToConstant: 60),
        // Future match label
        futureMatchLabel.topAnchor.constraint(equalTo: matchInfoContainer.topAnchor, constant: 8),
        futureMatchLabel.leadingAnchor.constraint(equalTo: matchInfoContainer.leadingAnchor, constant: 12),
        futureMatchLabel.trailingAnchor.constraint(equalTo: matchInfoContainer.trailingAnchor, constant: -12),
        // Future location stack
        futureLocationStack.topAnchor.constraint(equalTo: futureMatchLabel.bottomAnchor, constant: 4),
        futureLocationStack.leadingAnchor.constraint(equalTo: matchInfoContainer.leadingAnchor, constant: 12),
        smallLocationIcon.widthAnchor.constraint(equalToConstant: 14),
        smallLocationIcon.heightAnchor.constraint(equalToConstant: 14),
        // Button stack at the bottom
        buttonStack.bottomAnchor.constraint(equalTo: enhancedContainer.bottomAnchor, constant: -12),
        buttonStack.leadingAnchor.constraint(equalTo: enhancedContainer.leadingAnchor, constant: 16),
        buttonStack.trailingAnchor.constraint(equalTo: enhancedContainer.trailingAnchor, constant: -16),
        buttonStack.heightAnchor.constraint(equalToConstant: 44)
    ])
    // Set the frame for gradient layer
    DispatchQueue.main.async {
        gradientLayer.frame = titleContainerView.bounds
    }
    // Update the UI displays with current data (location and future match)
    updateLocationDisplay()
    updateFutureMatchDisplay()
}
// MARK: - Location Services
/* - Start location services to get current location
   - Uses LocationManager to obtain geographical information */
```

```
private func startLocationServices() {
    LocationManager.shared.startLocationUpdates { [weak self] success in
        if success {
            DispatchQueue.main.async {
                self?.updateLocationDisplay()
            }
        }
    }
}
/* - Update the location display with current location
   - Shows the city and country where the match is taking place */
private func updateLocationDisplay() {
    let (city, country) = LocationManager.shared.getCurrentLocation()
    if city != "Unknown" { // If the city is not unknown, update the display
        currentLocation = "\(city), \(country)"
        locationLabel.text = "Location: \(currentLocation)"
    }
}
/* - Update the future match display
   - Shows information about the next scheduled match if available */
private func updateFutureMatchDisplay() {
    if let futureMatch = CalendarManager.shared.getFutureMatch() { // If there is a future match
        let dateFormatter = DateFormatter()
        dateFormatter.dateStyle = .medium
        dateFormatter.timeStyle = .short
        futureMatchLabel.text = "\(futureMatch.title) on \(dateFormatter.string(from: futureMatch.date))"
        futureMatchLocationLabel.text = "Location: \(futureMatch.location)"
    } else {
        futureMatchLabel.text = "No upcoming match scheduled" // No upcoming match
```

```
        futureMatchLocationLabel.text = ""
    }
}
// MARK: - Button Actions
// Show match history screen when history button is tapped
@objc private func historyButtonTapped() {
    let historyVC = MatchHistoryViewController()
    let navController = UINavigationController(rootViewController: historyVC)
    present(navController, animated: true) // Present history screen
}
// Show match scheduling screen when schedule button is tapped
@objc private func scheduleButtonTapped() {
    let scheduleVC = FutureMatchViewController()
    scheduleVC.onMatchScheduled = { [weak self] match in
        self?.updateFutureMatchDisplay() // Update future match display if a new match is scheduled
    }
    let navController = UINavigationController(rootViewController: scheduleVC)
    present(navController, animated: true) // Present match scheduling screen
}
// MARK: - External Display Methods
/* - Handle external screen connection
  - Called when an external display is connected to the device */
@objc func screenDidConnect(_ notification: Notification) {
    setupExternalScreen() // Setup external screen when connected
}
//Handle when an external screen is disconnected
@objc func screenDidDisconnect(_ notification: Notification) {
    // Clean up external display resources
    externalWindow?.isHidden = true
```

```
    externalWindow = nil
    externalViewController = nil
}

/* - Set up external screen for match display
   - Creates a window on the external screen and configures it to show match information for spectators */
private func setupExternalScreen() {
    // Check if there's an external screen connected
    if let externalScreen = UIScreen.screens.last, UIScreen.screens.count > 1 {
        // Create a window for the external screen
        externalWindow = UIWindow(frame: externalScreen.bounds)
        externalWindow?.screen = externalScreen
        // Create an instance of the same storyboard
        let storyboard = UIStoryboard(name: "Main", bundle: nil)
        // Get a reference to the view controller
        if let viewController = storyboard.instantiateViewController(withIdentifier: "ViewController") as? ViewController {
            externalViewController = viewController
            // Configure the external view controller (disable buttons)
            externalViewController?.p1Button.isEnabled = false
            externalViewController?.p2Button.isEnabled = false
            externalWindow?.rootViewController = externalViewController
            externalWindow?.isHidden = false
            // Update external display
            updateExternalDisplay()
        }
    }
}
// Update the external display with the current match state
private func updateExternalDisplay() {
    // Update the external display with current match state
```

```
    guard let externalVC = externalViewController else { return }
    // Transfer scores and other information to external display
    externalVC.p1PointsLabel.text = p1PointsLabel.text
    externalVC.p2PointsLabel.text = p2PointsLabel.text
    externalVC.p1GamesLabel.text = p1GamesLabel.text
    externalVC.p2GamesLabel.text = p2GamesLabel.text
    externalVC.p1SetsLabel.text = p1SetsLabel.text
    externalVC.p2SetsLabel.text = p2SetsLabel.text
    externalVC.p1PreviousSetsLabel.text = p1PreviousSetsLabel.text
    externalVC.p2PreviousSetsLabel.text = p2PreviousSetsLabel.text
    // Update colors on the external display
    externalVC.p1NameLabel.backgroundColor = p1NameLabel.backgroundColor
    externalVC.p2NameLabel.backgroundColor = p2NameLabel.backgroundColor
    externalVC.p1PointsLabel.backgroundColor = p1PointsLabel.backgroundColor
    externalVC.p2PointsLabel.backgroundColor = p2PointsLabel.backgroundColor
    externalVC.p1GamesLabel.backgroundColor = p1GamesLabel.backgroundColor
    externalVC.p2GamesLabel.backgroundColor = p2GamesLabel.backgroundColor
    externalVC.p1SetsLabel.backgroundColor = p1SetsLabel.backgroundColor
    externalVC.p2SetsLabel.backgroundColor = p2SetsLabel.backgroundColor
    // Update location and future match info on the external display
    if externalVC.locationLabel != nil {
        externalVC.locationLabel.text = locationLabel.text
        externalVC.futureMatchLabel.text = futureMatchLabel.text
        externalVC.futureMatchLocationLabel.text = futureMatchLocationLabel.text
    }

}

// MARK: - Audio Setup
```

```
/* - Set up the audio player for sound effects
   - Prepares the audio file for playback during server changes and alerts */
private func setupAudioPlayer() {
    if let soundURL = Bundle.main.url(forResource: "Sound", withExtension: "wav") {
        do {
            audioPlayer = try AVAudioPlayer(contentsOf: soundURL)
            audioPlayer?.prepareToPlay()
        } catch {
            print("Error loading sound file: \(error.localizedDescription)") // Handle error in loading sound file
        }
    }
}
private func playSound() {
    audioPlayer?.play()
}



// MARK: - Action Methods
/* - Handler for when player 1 scores a point
   - Updates the match model and UI, checks for completed match */
@IBAction func p1AddPointPressed(_ sender: UIButton) {
    if !match.complete() {
        //match.addPointToPlayer1()
        let serverChanged = match.addPointToPlayer1() // Add point to player 1
        updateUI()
        // Play sound if the server changed
        if serverChanged {
            playSound()
        }
```

```
        if match.complete() { // If match is complete, save and announce the winner
            //add saving func
            saveMatchToHistory()
            announceWinner()
        }
    }
}
/* - Handler for when player 2 scores a point
   - Updates the match model and UI, checks for completed match */
@IBAction func p2AddPointPressed(_ sender: UIButton) {
    if !match.complete() {
        //match.addPointToPlayer2()
        let serverChanged = match.addPointToPlayer2()
        updateUI()
        // Play sound if server changed
        if serverChanged == true {
            playSound()
        }
        if match.complete() {
            // If match is complete, save and announce the winner
            saveMatchToHistory()
            announceWinner()
        }
    }
}
/* - Handler for restarting the match
   - Resets the match to initial state */
@IBAction func restartPressed(_ sender: AnyObject) {
    resetMatch()
```

```
}

/* - Save completed match to history
   - Stores match results in the MatchHistoryManager for future reference */
private func saveMatchToHistory() {
    // Get previous sets scores to store
    let previousSetsScores = match.previousSetsScores()
    // Convert to arrays for storage
    var player1Games: [Int] = []
    var player2Games: [Int] = []
    // Store previous sets scores
    for setScore in previousSetsScores {
        player1Games.append(setScore.0)
        player2Games.append(setScore.1)
    }
    // Add current set if complete
    if match.complete() {
        player1Games.append(match.player1CurrentGames())
        player2Games.append(match.player2CurrentGames())
    }
    // Save match details to history
    MatchHistoryManager.shared.saveMatch(
        player1Sets: match.player1Sets(),
        player2Sets: match.player2Sets(),
        player1Games: player1Games,
        player2Games: player2Games,
        location: currentLocation
    )
}
// MARK: - Helper Methods
```

```
/* - Reset the match and update the UI
   - Creates a new match and resets all UI elements */
private func resetMatch() {
    match.reset()
    totalGamesPlayed = 0

    // Make sure buttons are enabled
    p1Button.isEnabled = true // Enable player 1 button
    p2Button.isEnabled = true // Enable player 2 button
    updateUI()
}
/*  - Update the UI with current match state
    - Updates scores, colors, highlights, and external display */
private func updateUI() {
    // Reset background colors for all labels
    resetBackgroundColors()
    // Update scores and labels
    p1PointsLabel.text = match.player1GameScore()
    p2PointsLabel.text = match.player2GameScore()
    p1GamesLabel.text = "\(match.player1CurrentGames())"
    p2GamesLabel.text = "\(match.player2CurrentGames())"
    p1SetsLabel.text = "\(match.player1Sets())"
    p2SetsLabel.text = "\(match.player2Sets())"
    // Update previous sets scores
    updatePreviousSetsLabels()
    // Check for "new balls please" announcements
    checkForNewBalls()
    // Highlight the current server
    updateServerHighlight()
```

```
        // Highlight game/set/match points
        updatePointHighlights()
        // Update external display if connected
        updateExternalDisplay()
    }
    /* - Reset all background colors to default
       - Clears highlighting before applying new highlights */
    private func resetBackgroundColors() {
        p1NameLabel.backgroundColor = defaultColor
        p2NameLabel.backgroundColor = defaultColor
        p1PointsLabel.backgroundColor = defaultColor
        p2PointsLabel.backgroundColor = defaultColor
        p1GamesLabel.backgroundColor = defaultColor
        p2GamesLabel.backgroundColor = defaultColor
        p1SetsLabel.backgroundColor = defaultColor
        p2SetsLabel.backgroundColor = defaultColor
    }
    /* - Highlight the current server with purple background
       - Visual indicator of whose turn it is to serve */
    private func updateServerHighlight() {
        if match.isPlayer1Serving() {
            p1NameLabel.backgroundColor = serverColor
            p2NameLabel.backgroundColor = defaultColor
        } else {
            p1NameLabel.backgroundColor = defaultColor
            p2NameLabel.backgroundColor = serverColor
        }
    }
    //  Highlight game/set/match points with green background
```

```
    private func updatePointHighlights() {
        // Game points
        if match.hasPlayer1GamePoint() {
            p1PointsLabel.backgroundColor = pointColor
        }

        if match.hasPlayer2GamePoint() {
            p2PointsLabel.backgroundColor = pointColor
        }
        // Set points
        if match.hasPlayer1SetPoint() {
            p1GamesLabel.backgroundColor = pointColor
        }
        if match.hasPlayer2SetPoint() {
            p2GamesLabel.backgroundColor = pointColor
        }
        // Match points
        if match.hasPlayer1MatchPoint() {
            p1SetsLabel.backgroundColor = pointColor
        }
        if match.hasPlayer2MatchPoint() {
            p2SetsLabel.backgroundColor = pointColor
        }
    }
/* - Update the previous sets labels with completed sets scores
   - Shows the history of set scores for the match */
private func updatePreviousSetsLabels() {
    let previousSets = match.previousSetsScores()
    if previousSets.isEmpty {
```

```
            p1PreviousSetsLabel.text = "-"
            p2PreviousSetsLabel.text = "-"
        }
        else {
            var p1ScoresText = ""
            var p2ScoresText = ""
            for (p1Score, p2Score) in previousSets {
                p1ScoresText += "\(p1Score) "
                p2ScoresText += "\(p2Score) "
            }
            p1PreviousSetsLabel.text = p1ScoresText.trimmingCharacters(in: .whitespaces)
            p2PreviousSetsLabel.text = p2ScoresText.trimmingCharacters(in: .whitespaces)
        }
    }
    /* - Check if we need to announce "new balls please"
       - Follows tennis rules for ball changes during a match */
    private func checkForNewBalls() {
        let currentTotalGames = calculateTotalGamesPlayed()
        // Only announce when the game count changes
        if currentTotalGames > totalGamesPlayed {
            // New balls after first 7 games
            if currentTotalGames == 7 {
                announceNewBalls()
            }
            // Then after every 9 games
            else if currentTotalGames > 7 && (currentTotalGames - 7) % 9 == 0 {
                announceNewBalls()
            }
            // On a tie break
```

```
        else if match.isCurrentGameTieBreak() &&
                (match.player1CurrentGames() == 6 && match.player2CurrentGames() == 6) {
            announceNewBalls()
        }

        totalGamesPlayed = currentTotalGames
    }
}
/* - Calculate the total games played in the match
   - Used for determining when new balls are needed
   - Return Total number of games completed in the match */
private func calculateTotalGamesPlayed() -> Int {
    let previousSets = match.previousSetsScores()
    var total = 0
    // Count games from previous sets
    for (p1Games, p2Games) in previousSets {
        total += p1Games + p2Games
    }
    // Add games from current set
    total += match.player1CurrentGames() + match.player2CurrentGames()
    return total
}
/* - Announce new balls with an alert
   - Shows a modal dialog and plays a sound */
private func announceNewBalls() {
    let alert = UIAlertController(title: "New Balls Please",
                                  message: nil,
                                  preferredStyle: .alert)
    alert.addAction(UIAlertAction(title: "OK", style: .default))
```

```
    playSound()
    present(alert, animated: true)
}
/* - Announce the winner with an alert
   - Shows a modal dialog with the match result */
private func announceWinner() {
    var title = ""
    if match.player1Won() {
        title = "Player 1 Wins!"
    } else if match.player2Won() {
        title = "Player 2 Wins!"
    }

    let alert = UIAlertController(title: title,
                                  message: "Game, Set, Match!",
                                  preferredStyle: .alert)
    alert.addAction(UIAlertAction(title: "OK", style: .default))
    // Add action to view future match if scheduled
    if CalendarManager.shared.getFutureMatch() != nil {
        alert.addAction(UIAlertAction(title: "Show Future Match", style: .default) { [weak self] _ in
            self?.showFutureMatchDetails()
        })
    }
    // Disable buttons when match is complete
    p1Button.isEnabled = false
    p2Button.isEnabled = false
    playSound()
    present(alert, animated: true)
}
```

```
    /* - Show details about the next scheduled match
       - Displays information about the upcoming match */

    private func showFutureMatchDetails() {
        guard let futureMatch = CalendarManager.shared.getFutureMatch() else { return }
        let dateFormatter = DateFormatter()
        dateFormatter.dateStyle = .full
        dateFormatter.timeStyle = .short
        let alert = UIAlertController(
            title: "Next Match",
            message: "\(futureMatch.title)\nDate: \(dateFormatter.string(from: futureMatch.date))\nLocation: \(futureMatch.location)",
            preferredStyle: .alert
        )
        alert.addAction(UIAlertAction(title: "OK", style: .default))
        present(alert, animated: true)
    }
}
```

MatchHistoryViewController.swift file:

```
// ViewController for displaying match history
import UIKit
/* - ViewController for displaying match history
   - Shows a list of past matches and allows viewing details or deleting records
   - Uses UITableView to display the match list */
class MatchHistoryViewController: UIViewController, UITableViewDelegate, UITableViewDataSource {
    private var tableView: UITableView!
    private var matches: [MatchHistoryManager.MatchRecord] = []
    override func viewDidLoad() {
        super.viewDidLoad()
        setupUI()
```

```
    loadMatches() // Refresh the table view
}
override func viewWillAppear(_ animated: Bool) {
    super.viewWillAppear(animated)
    loadMatches()
    tableView.reloadData()
}
/* - Set up the UI elements
  - Creates and configures the table view + navigation */
private func setupUI() {
    view.backgroundColor = .white // background view colour

    // Create table view to display matches
    tableView = UITableView(frame: view.bounds, style: .plain)
    tableView.delegate = self
    tableView.dataSource = self
    tableView.register(UITableViewCell.self, forCellReuseIdentifier: "MatchCell")
    tableView.autoresizingMask = [.flexibleWidth, .flexibleHeight] // Make the table view resize automatically
    view.addSubview(tableView)
    // Title of the view
    title = "Match History"
    // Add a Clear All button
    navigationItem.rightBarButtonItem = UIBarButtonItem(
        title: "Clear All",
        style: .plain,
        target: self,
        action: #selector(clearAllTapped)
    )
}
```

```
/* - Load match history data
   - Gets matches from MatchHistoryManager and sorts by date */

private func loadMatches() {
    matches = MatchHistoryManager.shared.getAllMatches().sorted { $0.date > $1.date } // Sort matches by date (newest first)
}

// Action when the "Clear All" button is tapped
@objc private func clearAllTapped() {
    // Show an alert to confirm deletion of all matches
    let alert = UIAlertController(
        title: "Clear All Matches",
        message: "Are you sure you want to delete all match history? This cannot be undone.",
        preferredStyle: .alert
    )
    // Add actions for the alert
    alert.addAction(UIAlertAction(title: "Cancel", style: .cancel)) // Cancel action
    // Clear all matches + reload the data
    alert.addAction(UIAlertAction(title: "Clear All", style: .destructive) { [weak self] _ in
        MatchHistoryManager.shared.clearAllMatches()
        self?.loadMatches()
        self?.tableView.reloadData()
    })
    present(alert, animated: true)
}
// MARK: - UITableViewDataSource
// Return the number of rows for the table view (either 1 for no matches or the count of matches)
func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
    return matches.isEmpty ? 1 : matches.count
}
```

```
// Configure each cell in the table view
func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
    let cell = tableView.dequeueReusableCell(withIdentifier: "MatchCell", for: indexPath)
    if matches.isEmpty {
        cell.textLabel?.text = "No match history available"
        cell.isUserInteractionEnabled = false
    } else {
        let match = matches[indexPath.row]
        let dateFormatter = DateFormatter()
        dateFormatter.dateStyle = .medium
        cell.textLabel?.numberOfLines = 0
        cell.textLabel?.text = formatMatchCellText(match)
        cell.accessoryType = .disclosureIndicator
    }
    return cell
}
/* - Format match information for display in table cell
   - Return Formatted string with match summary */
// Clarify: match - The match record to format
private func formatMatchCellText(_ match: MatchHistoryManager.MatchRecord) -> String {
    let dateFormatter = DateFormatter()
    dateFormatter.dateStyle = .medium
    let dateString = dateFormatter.string(from: match.date)
    let winnerText = match.player1Sets > match.player2Sets ? "Player 1 won" : "Player 2 won"
    return "\(dateString) - \(match.location)\n\(winnerText) (\(match.player1Sets)-\(match.player2Sets))"
}
// MARK: - UITableViewDelegate

// Handle row selection in the table view
```

```
func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {
    tableView.deselectRow(at: indexPath, animated: true) // Deselect the row after it's tapped
    if !matches.isEmpty {
        let match = matches[indexPath.row]
        showMatchDetail(match)
    }
}
// Handle swipe-to-delete action in the table view
func tableView(_ tableView: UITableView, commit editingStyle: UITableViewCell.EditingStyle, forRowAt indexPath: IndexPath) {
    if editingStyle == .delete && !matches.isEmpty {
        let match = matches[indexPath.row] // Get the match to delete
        MatchHistoryManager.shared.deleteMatch(withId: match.id)
        loadMatches()
        // Check if there are no matches left
        if matches.isEmpty {
            tableView.reloadData() // Reload the table to show "No match history" message
        } else {
            tableView.deleteRows(at: [indexPath], with: .fade)
        }
    }
}
/* - Show detailed information about a match
  - Displays a modal with full match statistics */
//Clarify: Match - The match to display
private func showMatchDetail(_ match: MatchHistoryManager.MatchRecord) {
    let alert = UIAlertController(
        title: "Match Details",
        message: MatchHistoryManager.shared.formatMatchForDisplay(match),
        preferredStyle: .alert
```

```
        )
        alert.addAction(UIAlertAction(title: "Close", style: .default))
        present(alert, animated: true)
    }
}
```

FutureMatchViewController.swift file:

```
import UIKit
/* - ViewController for scheduling future matches
  - Allows users to enter match details and add to calendar
  - Uses CalendarManager to handle the actual scheduling */
class FutureMatchViewController: UIViewController {
  // UI Components
  private let titleTextField = UITextField()
  private let datePicker = UIDatePicker()
  private let locationTextField = UITextField()
  private let notesTextView = UITextView()
  private let scheduleButton = UIButton(type: .system)
  private let cancelButton = UIButton(type: .system)
  private let locationButton = UIButton(type: .system)
  // Selected location
  private var selectedLocation: String = ""
  // Completion handler for when a match is scheduled
  var onMatchScheduled: ((CalendarManager.FutureMatch) -> Void)?

  override func viewDidLoad() {
    super.viewDidLoad()
    setupUI() // Initialise and set up UI components
    setupActions() // Set up actions for button taps
    // Check and update the location display with current location
```

```
    updateLocationDisplay()
}

 /* - Set up the UI elements
   - Creates and positions all interface components */
private func setupUI() {
    view.backgroundColor = .white
    title = "Schedule Future Match" // Set the title for the view
    // Set up containers for layout
    let contentStackView = UIStackView()
    contentStackView.axis = .vertical
    contentStackView.spacing = 16
    contentStackView.translatesAutoresizingMaskIntoConstraints = false
    view.addSubview(contentStackView)
    // Add a label and text field for match title
    let titleLabel = UILabel()
    titleLabel.text = "Match Title:"
    titleLabel.translatesAutoresizingMaskIntoConstraints = false
    titleTextField.placeholder = "Enter match title"
    titleTextField.borderStyle = .roundedRect
    titleTextField.translatesAutoresizingMaskIntoConstraints = false
    let titleStackView = UIStackView(arrangedSubviews: [titleLabel, titleTextField])
    titleStackView.axis = .vertical
    titleStackView.spacing = 8
    contentStackView.addArrangedSubview(titleStackView)
    // Add a label and date picker for match date
    let dateLabel = UILabel()
    dateLabel.text = "Match Date:"
    dateLabel.translatesAutoresizingMaskIntoConstraints = false
    datePicker.datePickerMode = .dateAndTime
```

```
datePicker.preferredDatePickerStyle = .compact

datePicker.minimumDate = Date() // Can't schedule in the past

datePicker.translatesAutoresizingMaskIntoConstraints = false

let dateStackView = UIStackView(arrangedSubviews: [dateLabel, datePicker])

dateStackView.axis = .vertical

dateStackView.spacing = 8

contentStackView.addArrangedSubview(dateStackView)

// Add a label and text field for location

let locationLabel = UILabel()

locationLabel.text = "Location:"

locationLabel.translatesAutoresizingMaskIntoConstraints = false

locationTextField.placeholder = "Enter location"

locationTextField.borderStyle = .roundedRect

locationTextField.translatesAutoresizingMaskIntoConstraints = false


// Use Current Location button

locationButton.setTitle("Use Current Location", for: .normal)

locationButton.translatesAutoresizingMaskIntoConstraints = false

let locationStackView = UIStackView(arrangedSubviews: [locationLabel, locationTextField, locationButton])

locationStackView.axis = .vertical

locationStackView.spacing = 8

contentStackView.addArrangedSubview(locationStackView)

// Notes section

let notesLabel = UILabel()

notesLabel.text = "Notes:"

notesLabel.translatesAutoresizingMaskIntoConstraints = false

notesTextView.layer.borderColor = UIColor.lightGray.cgColor

notesTextView.layer.borderWidth = 1

notesTextView.layer.cornerRadius = 5
```

```
notesTextView.font = UIFont.systemFont(ofSize: 15)
notesTextView.translatesAutoresizingMaskIntoConstraints = false
let notesStackView = UIStackView(arrangedSubviews: [notesLabel, notesTextView])
notesStackView.axis = .vertical
notesStackView.spacing = 8
contentStackView.addArrangedSubview(notesStackView)
// Add buttons
scheduleButton.setTitle("Schedule Match", for: .normal)
scheduleButton.backgroundColor = .systemBlue
scheduleButton.setTitleColor(.white, for: .normal)
scheduleButton.layer.cornerRadius = 8
scheduleButton.translatesAutoresizingMaskIntoConstraints = false
cancelButton.setTitle("Cancel", for: .normal)
cancelButton.backgroundColor = .systemGray
cancelButton.setTitleColor(.white, for: .normal)
cancelButton.layer.cornerRadius = 8
cancelButton.translatesAutoresizingMaskIntoConstraints = false

let buttonStackView = UIStackView(arrangedSubviews: [scheduleButton, cancelButton])
buttonStackView.axis = .horizontal
buttonStackView.spacing = 16
buttonStackView.distribution = .fillEqually
contentStackView.addArrangedSubview(buttonStackView)
// Apply constraints
NSLayoutConstraint.activate([
    contentStackView.topAnchor.constraint(equalTo: view.safeAreaLayoutGuide.topAnchor, constant: 20),
    contentStackView.leadingAnchor.constraint(equalTo: view.leadingAnchor, constant: 20),
    contentStackView.trailingAnchor.constraint(equalTo: view.trailingAnchor, constant: -20),
    titleTextField.heightAnchor.constraint(equalToConstant: 40),
```

```
            notesTextView.heightAnchor.constraint(equalToConstant: 100),
            scheduleButton.heightAnchor.constraint(equalToConstant: 44),
            cancelButton.heightAnchor.constraint(equalToConstant: 44)
        ])
    }
    /* - Set up action handlers for buttons
       - Connects UI elements to their corresponding methods */
    private func setupActions() {
        scheduleButton.addTarget(self, action:  #selector(scheduleButtonTapped), for: .touchUpInside)
        cancelButton.addTarget(self, action: #selector(cancelButtonTapped), for: .touchUpInside)
        locationButton.addTarget(self, action: #selector(useCurrentLocationTapped), for: .touchUpInside)
    }
    /* - Update location display with current location
       - Gets location information from LocationManager */
    private func updateLocationDisplay() {
        let (city, country) = LocationManager.shared.getCurrentLocation()
        if city != "Unknown" {
            selectedLocation = "\(city), \(country)"
            locationTextField.text = selectedLocation
        }
    }
    /* - Handle the schedule button tap
       - Validates inputs and schedules the match */
    @objc private func scheduleButtonTapped() {
        guard let title = titleTextField.text, !title.isEmpty else {
            showAlert(title: "Error", message: "Please enter a match title")
            return
        }
        guard let location = locationTextField.text, !location.isEmpty else {
```

```
            showAlert(title: "Error", message: "Please enter a location")
            return
        }
        let date = datePicker.date
        let notes = notesTextView.text ?? ""

        // Schedule the match using CalendarManager
        CalendarManager.shared.scheduleFutureMatch(
            title: title,
            date: date,
            location: location,
            notes: notes ){
                [weak self] success, futureMatch in
            if success, let match = futureMatch {
                self?.onMatchScheduled?(match)
                self?.showAlert(title: "Success", message: "Match has been scheduled") { [weak self] in
                    self?.dismiss(animated: true)
                }
            } else {
                self?.showAlert(title: "Error", message: "Failed to schedule match. Please check calendar permissions.")
            }
        }
    }

    /* - Handle the cancel button tap
       - Dismisses the view controller without scheduling */
    @objc private func cancelButtonTapped() {
        dismiss(animated: true)
    }
```

```
    /* - Handle the use current location button tap
       - Gets the current location + updates the location field */
    @objc private func useCurrentLocationTapped() {
        // Start location updates
        LocationManager.shared.startLocationUpdates { [weak self] success in
            if success {
                DispatchQueue.main.async {
                    // If location is successfully retrieved, update the UI with the location
                    self?.updateLocationDisplay()
                }
            } else {
                DispatchQueue.main.async {
                    // If there's an error, show an alert
                    self?.showAlert(title: "Location Error", message: "Could not get current location. Please check location permissions or enter location manually.")
                }
            }
        }
    }
    // Show an alert with a message
    /* Clarify:
     - Title - Alert title
     - Message - Alert message
     - Completion - Optional callback after user dismisses the alert */
    private func showAlert(title: String, message: String, completion: (() -> Void)? = nil) {
        let alert = UIAlertController(title: title, message: message, preferredStyle: .alert)
        alert.addAction(UIAlertAction(title: "OK", style: .default) { _ in
            completion?() // Executes completion if provided
        })
        present(alert, animated: true)
```

```
  }
}
```

Info.plist file:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
<key>CFBundleDevelopmentRegion</key>
<string>en</string>
<key>CFBundleExecutable</key>
<string>$(EXECUTABLE_NAME)</string>
<key>CFBundleIdentifier</key>
<string>$(PRODUCT_BUNDLE_IDENTIFIER)</string>
<key>CFBundleInfoDictionaryVersion</key>
<string>6.0</string>
<key>CFBundleName</key>
<string>$(PRODUCT_NAME)</string>
<key>CFBundlePackageType</key>
<string>APPL</string>
<key>CFBundleShortVersionString</key>
<string>1.0</string>
<key>CFBundleSignature</key>
<string>????</string>
<key>CFBundleVersion</key>
<string>1</string>
<key>LSRequiresIPhoneOS</key>
<true/>
<key>UILaunchStoryboardName</key>
<string>LaunchScreen</string>
```

```
<key>UIMainStoryboardFile</key>
<string>Main</string>
<key>UIRequiredDeviceCapabilities</key>
<array>
<string>armv7</string>
</array>
<key>UISupportedInterfaceOrientations</key>
<array>
<string>UIInterfaceOrientationPortrait</string>
<string>UIInterfaceOrientationLandscapeLeft</string>
<string>UIInterfaceOrientationLandscapeRight</string>
</array>
  <!-- Permission descriptions -->
  <key>NSLocationWhenInUseUsageDescription</key>
  <string>This app needs your location to identify the current match location and display it.</string>
  <key>NSCalendarsUsageDescription</key>
  <string>This app needs access to your calendar to schedule future tennis matches.</string>
</dict>
</plist>
```

Game.swift file:

```
/*
 - Handles scoring for a single tennis game
 - Manages scoring with the standard tennis system (0,15,30,40,A)
 - Detects when a game is complete
 */
class Game {
  // Tracking points for both players
  private var player1Points: Int = 0
  private var player2Points: Int = 0
```

```
  /*
 - Adds a point for player 1 and handles deuce/advantage logic
 - If game is already complete, no action is taken */
  func addPointToPlayer1() {
  // If game is already complete, do nothing
  if complete() {
    return
  }

  // Check if player 2 has advantage
  if player2Points >= 4 && player2Points == player1Points + 1 {
    // Remove player 2's advantage, back to deuce
    player2Points = 3
    player1Points = 3
  } else {
    // Otherwise, increment player 1's points
    player1Points += 1
  }
}
/*
 - Adds a point for player 2 and handles deuce/advantage logic
 - If game is already complete, no action is taken */
func addPointToPlayer2() {
  // If game is already complete, do nothing
  if complete() {
    return
  }
  // Check if player 1 has advantage
  if player1Points >= 4 && player1Points == player2Points + 1 {
```

```
        // Remove player 1's advantage, back to deuce
        player1Points = 3
        player2Points = 3
    } else {
        // Otherwise, increment player 2's points
        player2Points += 1
    }
}
/*
 - Returns the score for player 1 in tennis notation
 - Returns "0","15","30","40" or "A" based on current points
 - If the game is complete, returns an empty string */

func player1Score() -> String {
    if complete() {
        return ""
    }
    return scoreString(player1Points, otherPlayerPoints: player2Points)
}
/*
 - Returns the score for player 2 in tennis notation
 - Returns "0","15","30","40" or "A" based on current points
 - If the game is complete, returns an empty string */
func player2Score() -> String {
    if complete() {
        return ""
    }
    return scoreString(player2Points, otherPlayerPoints: player1Points)
}
```

```
  /* - Returns true if player 1 has won the game
   - A player wins when they have at least 4 points and a 2-point lead */
  func player1Won() -> Bool {
    // Player 1 wins if they have at least 4 points and lead by at least 2
    return player1Points >= 4
    && player1Points >= player2Points + 2
  }
  /*
   - Returns true if player 2 has won the game
   - A player wins when they have at least 4 points and a 2-point lead */
  func player2Won() -> Bool {
    // Player 2 wins if they have at least 4 points and lead by at least 2
    return player2Points >= 4 && player2Points >= player1Points + 2
  }
  // Returns true if the game is finished (either player has won)

  func complete() -> Bool {
    return player1Won() || player2Won()
  }
  /*
   - If player 1 would win the game if they won the next point, returns the number of points player 2 would need to win to equalise the score, otherwise returns 0
   - E.g: if the score is 40:15 to player 1, player 1 would win if they scored the next point, and player 2 would need 2 points in a row to prevent that, so this method should return 2 in that case. */
  func gamePointsForPlayer1() -> Int {
    if player1Points < 3 {
      // Player 1 is not yet at 40, so no game point
      return 0
    }
```

```
        if player1Points == 3 && player2Points < 3 {
            // Player 1 at 40, player 2 not yet at 40
            return 3 - player2Points
        }

        if player1Points >= 4 && player1Points == player2Points + 1 {
            // Player 1 has advantage
            return 1
        }
        return 0
    }

    // If player 2 would win the game if they won the next point, returns the number of points player 1 would need to win to equalise the score
    func gamePointsForPlayer2() -> Int {
        if player2Points < 3 {
            // Player 2 is not yet at 40, so no game point
            return 0
        }
        if player2Points == 3 && player1Points < 3 {
            // Player 2 at 40, player 1 not yet at 40
            return 3 - player1Points
        }

        if player2Points >= 4 && player2Points == player1Points + 1 {
            // Player 2 has advantage
            return 1
        }
        return 0
    }

    // Helper method to convert numeric score to tennis score string
```

```
    private func scoreString(_ points: Int, otherPlayerPoints: Int) -> String {
        switch points {
        case 0:
            return "0"
        case 1:
            return "15"
        case 2:
            return "30"
        case 3:
            return "40"
        default:
            // If points > 3 and one point ahead, it's advantage
            if points >= 4 && points == otherPlayerPoints + 1 {
                return "A"
            } else {
                return "40" // Otherwise it's still 40 (e.g in deuce)
            }
        }
    }
}
```

TennisSet.swift file:

```
import Foundation

/* - TennisSet class handles the scoring and progression of games within a tennis set
   - It manages the games within a set and tracks when sets are completed
   - Also handles special tie-break rules, including the different rules for the final set */
class TennisSet {
    // Track games won by each player
    private var player1Games: Int = 0
    private var player2Games: Int = 0
```

```
// Current game being played in this set
private var currentGame: Game
// Whether this set is in tie-break mode
private var isTieBreak: Bool = false
// Whether this is the final set of the match (special rules apply)
private var isLastSet: Bool = false
// Whether the current game was just completed
private var gameJustCompleted: Bool = false
/*  Initialize a new tennis set (whether this is the last set of the match, affects tie-break rules) */
init(isLastSet: Bool = false) {
    self.currentGame = Game()
    self.isLastSet = isLastSet
    self.gameJustCompleted = false
}

/* - Adds a point for player 1 and handles game/set progression
   - Updates game count if player wins a game
   - Checks for tie-break conditions
   - Creates new games as needed */
func addPointToPlayer1() {
    // Do nothing if set is already complete
    if complete() {
        return
    }
    // Reset the gameJustCompleted flag
    gameJustCompleted = false
    // Handle scoring based on whether we're in a tie-break
    if !isTieBreak {
        // Standard game scoring
```

```
            currentGame.addPointToPlayer1()
            // If player 1 won the game, update games count
            if currentGame.player1Won() {
                player1Games += 1
                gameJustCompleted = true
                // If set is not complete, prepare for next game
                if !complete() {
                    currentGame = Game()
                    // Check if we need to enter tie-break mode
                    isTieBreak = shouldStartTieBreak()
                }
            }
        } else {
            // Tie-break scoring
            currentGame.addPointToPlayer1()
            // If player 1 won the tie-break, they win the set
            if currentGame.player1Won() {
                player1Games += 1
                gameJustCompleted = true
            }
        }
    }
    /* - Adds a point for player 2 and handles game/set progression
    - Updates game count if player wins a game
    - Checks for tie-break conditions
    - Creates new games as needed */
    func addPointToPlayer2() {
    // Do nothing if set is already complete
    if complete() {
```

```
    return
}
// Reset the gameJustCompleted flag
gameJustCompleted = false
// Handle scoring based on whether we're in a tie-break
if !isTieBreak {
    // Standard game scoring
    currentGame.addPointToPlayer2()

    // If player 2 won the game, update games count
    if currentGame.player2Won() {
        player2Games += 1
        gameJustCompleted = true
       // If set is not complete, prepare for next game
        if !complete() {
            currentGame = Game()
            // Check if we need to enter tie-break mode
            isTieBreak = shouldStartTieBreak()
        }
    }
} else {
    // Tie-break scoring
    currentGame.addPointToPlayer2()
    // If player 2 won the tie-break, they win the set
    if currentGame.player2Won() {
        player2Games += 1
        gameJustCompleted = true
    }
}
```

```
}

/* - Determines if a tie-break should start based on current score
   - For final set, tie-break starts at 12-12
   - For other sets, tie-break starts at 6-6
   - Return True if tie-break should begin */
private func shouldStartTieBreak() -> Bool {
    // In the final set, tie-break only starts at 12-12
    if isLastSet {
        return player1Games == 12 && player2Games == 12
    }
    // In other sets, tie-break starts at 6-6
    return player1Games == 6 && player2Games == 6
}



/* - Get current game score for player 1, handling tie-break differently
   - For tie-breaks, displays actual points (1,2,3...) instead of tennis scores
   - Return Score as a string */
func player1GameScore() -> String {
    if isTieBreak {
        // For tie-breaks, display actual points (1,2,3...) instead of tennis scores
        if let p1Points = Int(currentGame.player1Score()) {
            return "\(p1Points)"
        } else if currentGame.player1Score() == "15" {
            return "1"
        } else if currentGame.player1Score() == "30" {
            return "2"
        } else if currentGame.player1Score() == "40" {
            return "3"
```

```
        } else if currentGame.player1Score() == "A" {
            // Shouldn't happen in tie-break, but handle anyway
            return "A"
        }
        return currentGame.player1Score()
    } else {
        return currentGame.player1Score()
    }
}
/* - Get current game score for player 2, handling tie-break differently
   - For tie-breaks, displays actual points (1,2,3...) instead of tennis scores
   - Return Score as a string */
func player2GameScore() -> String {
    if isTieBreak {
        // For tie-breaks, display actual points (1,2,3...) instead of tennis scores
        if let p2Points = Int(currentGame.player2Score()) {
            return "\(p2Points)"
        } else if currentGame.player2Score() == "15" {
            return "1"
        } else if currentGame.player2Score() == "30" {
            return "2"
        } else if currentGame.player2Score() == "40" {
            return "3"
        } else if currentGame.player2Score() == "A" {
            // Shouldn't happen in tie-break, but handle anyway
            return "A"
        }
        return currentGame.player2Score()
    } else {
```

```
        return currentGame.player2Score()
    }
}

/* - Returns true if player 1 has won the set
   - In tie-break mode, winning the tie-break game means winning the set
   - In standard mode, need at least 6 games and a 2-game lead */
func player1Won() -> Bool {
    if isTieBreak {
        // In tie-break, winning the tie-break game means winning the set
        return currentGame.player1Won()
    }
    // Standard set rules - need at least 6 games and 2-game lead
    return (player1Games >= 6 && player1Games >= player2Games + 2)
}
/* - Returns true if player 2 has won the set
   - In tie-break mode, winning the tie-break game means winning the set
   - In standard mode, need at least 6 games and a 2-game lead */

func player2Won() -> Bool {
    if isTieBreak {
        // In tie-break, winning the tie-break game means winning the set
        return currentGame.player2Won()
    }
    // Standard set rules - need at least 6 games and 2-game lead
    return (player2Games >= 6 && player2Games >= player1Games + 2)
}
//Returns true if the set is complete (either player has won)
```

```
func complete() -> Bool {
    return player1Won() || player2Won()
}
// Get the number of games won by player 1 in this set
func getPlayer1Games() -> Int {
    return player1Games
}



// Get the number of games won by player 2 in this set
func getPlayer2Games() -> Int {
    return player2Games
}
// Check if a game was just completed ( Used to determine when to change server, etc)
func isGameComplete() -> Bool {
    return gameJustCompleted
}
 //Check if the set is currently in tie-break mode
func isInTieBreak() -> Bool {
    return isTieBreak
}
//Check if player 1 has game point/s ( Different for tie-break vs. regular game)
func hasPlayer1GamePoint() -> Bool {
    if isTieBreak {
        // In tiebreak, game point is when player has 6+ points and leads by at least 1
        return currentGame.gamePointsForPlayer1() > 0
    } else {
        // In regular game, 40-0, 40-15, or 40-30 is game point
        return currentGame.gamePointsForPlayer1() > 0
```

```
    }
}
//Check if player 2 has game point/s ( Different for tie-break vs. regular game)
func hasPlayer2GamePoint() -> Bool {
    if isTieBreak {
        // In tiebreak, game point is when player has 6+ points and leads by at least 1
        return currentGame.gamePointsForPlayer2() > 0
    } else {
        // In regular game, 0-40, 15-40, or 30-40 is game point
        return currentGame.gamePointsForPlayer2() > 0
    }
}
    // Check if player 1 has set point/s ( Set point occurs when winning the current game would win the set)
func hasPlayer1SetPoint() -> Bool {
    // If the set is already complete, there's no set point
    if complete() {
        return false
    }
    if isTieBreak {
        // In tiebreak, set point is the same as game point
        return hasPlayer1GamePoint()
    } else {
        // In regular play, set point is when player has 5+ games and:
        // 1. Leads by 1 game and has game point, or
        // 2. Leads by 2+ games and has game point
        return (player1Games >= 5 &&
                ((player1Games == player2Games + 1 && hasPlayer1GamePoint()) ||
                 (player1Games >= player2Games + 2 && hasPlayer1GamePoint())))
    }
```

```
    }

    //Check if player 2 has set point/s ( Set point occurs when winning the current game would win the set)
    func hasPlayer2SetPoint() -> Bool {
        // If the set is already complete, there's no set point
        if complete() {
            return false
        }
        if isTieBreak {
            // In tiebreak, set point is the same as game point
            return hasPlayer2GamePoint()
        } else {
            // In regular play, set point is when player has 5+ games and:
            // 1. Leads by 1 game and has game point, or
            // 2. Leads by 2+ games and has game point

            return (player2Games >= 5 &&
                ((player2Games == player1Games + 1 && hasPlayer2GamePoint()) ||
                 (player2Games >= player1Games + 2 && hasPlayer2GamePoint())))
        }
    }
}
```

TennisMatch.swift file:

```
import Foundation

// Handles multiple sets, tracking progress, determining the match winner, and manages service changes following tennis rules
class TennisMatch {
    // Collection of all sets in the match
    private var sets: [TennisSet] = []
    // Index of the current set being played
    private var currentSetIndex: Int = 0
```

```
// Track sets won by each player
private var player1SetsWon: Int = 0
private var player2SetsWon: Int = 0
// Number of sets needed to win the match (best of 5)
private let totalSetsNeeded = 3
// Track who is serving (true = player 1, false = player 2)
 private var player1Serving: Bool = true
// Track points played in the current tiebreak (for service changes)
 private var tiebreakPointsPlayed: Int = 0
// Initialise a new tennis match ( Sets up the first set and assigns player 1 as the first server)
  init() {
    // Start with the first set
    sets.append(TennisSet())
    // Player 1 serves first
    player1Serving = true
}


/* - Add a point for player 1 and handle set/match progression
   - Manages service changes according to tennis rules
   - Return Bool indicating if server changed */
func addPointToPlayer1() -> Bool {
    // Do nothing if match is already complete
    if complete() {
        return false
    }
    let currentSet = sets[currentSetIndex]
    let isInTieBreak = currentSet.isInTieBreak()
    let wasGameComplete = currentSet.isGameComplete()
```

```
let wasSetComplete = currentSet.complete()

var serverChanged = false

// Add point to current set

// let currentSet = sets[currentSetIndex]

currentSet.addPointToPlayer1()


// Update tiebreak counter if in tiebreak

if isInTieBreak {

   tiebreakPointsPlayed += 1

}

// Check if the set is complete

if currentSet.complete() && !wasSetComplete {

   // Update sets won count

   if currentSet.player1Won() {

      player1SetsWon += 1

   }

   else if currentSet.player2Won() {

      player2SetsWon += 1

   }

   // If we were in a tiebreak, the server for the next game is the player

   // who did not serve first in the tiebreak

   if isInTieBreak {

      player1Serving = !player1ServingFirstInTiebreak()

   }

   else {

      // Otherwise, service alternates normally

      player1Serving = !player1Serving

   }

   serverChanged = true
```

```
    // If match is not complete, start a new set
    if !complete() {
        currentSetIndex += 1
        // If this will be the fifth set, mark it as the last set
        let isLastSet = currentSetIndex == 4
        sets.append(TennisSet(isLastSet: isLastSet))
        // Reset tiebreak counter
        tiebreakPointsPlayed = 0
        // return true // Service changed

    }
}
// Check if a regular game completed
else if currentSet.isGameComplete() && !wasGameComplete && !isInTieBreak {
    // Change server after a completed game
    player1Serving = !player1Serving
    serverChanged = true
    // return true // Service changed
}
// Check for service change during tiebreak
else if isInTieBreak {
    // In tiebreak, service changes after first point, then every 2 points
    if tiebreakPointsPlayed == 1 || tiebreakPointsPlayed > 1 && tiebreakPointsPlayed % 2 == 1 {
        player1Serving = !player1Serving
        //return true // Service changed
        serverChanged = true
    }
}
// return false // No service change
```

```
    return serverChanged
}
/* - Add a point for player 2 and handle set/match progression
   - Manages service changes according to tennis rules
   - Return Bool if server changed */
func addPointToPlayer2() -> Bool {
    // Do nothing if match is already complete
    if complete() {
        return false
    }
    let currentSet = sets[currentSetIndex]
    let isInTieBreak = currentSet.isInTieBreak()
    let wasGameComplete = currentSet.isGameComplete()
    let wasSetComplete = currentSet.complete()
    var serverChanged = false



    // Add point to current set
    currentSet.addPointToPlayer2()
    // Update tiebreak counter if in tiebreak
    if isInTieBreak {
        tiebreakPointsPlayed += 1
    }
    // Check if the set is complete
    if currentSet.complete() {
        // Update sets won count
        if currentSet.player1Won() {
            player1SetsWon += 1
        } else if currentSet.player2Won() {
```

```
            player2SetsWon += 1
        }



        // If we were in a tiebreak, the server for the next game is the player
        if isInTieBreak {
            player1Serving = !player1ServingFirstInTiebreak()
        } else {
            // Otherwise, service alternates normally
            player1Serving = !player1Serving
        }
        serverChanged = true
        // If match is not complete, start a new set
        if !complete() {
            currentSetIndex += 1
            // If this will be the fifth set, mark it as the last set
            let isLastSet = currentSetIndex == 4
            sets.append(TennisSet(isLastSet: isLastSet))
            // Reset tiebreak counter
            tiebreakPointsPlayed = 0
            //return true // Service changed
        }
    }
    // Check if a regular game completed
    else if currentSet.isGameComplete() && !wasGameComplete && !isInTieBreak {
        // Change server after a completed game
        player1Serving = !player1Serving
        //return true // Service changed
        serverChanged = true
```

```
    }
        // Check for service change during tiebreak
        else if isInTieBreak {
            // In tiebreak, service changes after first point, then every 2 points
            if tiebreakPointsPlayed == 1 || tiebreakPointsPlayed > 1 && tiebreakPointsPlayed % 2 == 1 {
                player1Serving = !player1Serving


                //return true // Service changed
                serverChanged = true
            }
        }
        //return false // No service change
        return serverChanged
}
// Returns true if player 1 has won the match ( A player wins by winning the required number of sets (3 in best of 5))
func player1Won() -> Bool {
    return player1SetsWon >= totalSetsNeeded
}
// Returns true if player 2 has won the match (A player wins by winning the required number of sets (3 in best of 5))
func player2Won() -> Bool {
    return player2SetsWon >= totalSetsNeeded
}
// Returns true if the match is complete (either player has won)

func complete() -> Bool {
    return player1Won() || player2Won()
}
// Get the current game score for player 1
func player1GameScore() -> String {
```

```
    if currentSetIndex < sets.count {
        return sets[currentSetIndex].player1GameScore()
    }
    return ""
}
// Get the current game score for player 2
func player2GameScore() -> String {
    if currentSetIndex < sets.count {
        return sets[currentSetIndex].player2GameScore()
    }
    return ""
}

// Get the games in the current set for player 1
func player1CurrentGames() -> Int {
    if currentSetIndex < sets.count {
        return sets[currentSetIndex].getPlayer1Games()
    }
    return 0
}
// Get the games in the current set for player 2
func player2CurrentGames() -> Int {
    if currentSetIndex < sets.count {
        return sets[currentSetIndex].getPlayer2Games()
    }
    return 0
}
// Get the number of sets won by player 1
```

```
func player1Sets() -> Int {
    return player1SetsWon
}
// Get the number of sets won by player 2
func player2Sets() -> Int {
    return player2SetsWon
}
/* - Get previous sets scores for display
   - Return Array of tuples with (player1Games, player2Games) for each completed set */
func previousSetsScores() -> [(Int, Int)] {
    var result: [(Int, Int)] = []
    // Return scores for completed sets only
    for i in 0..<currentSetIndex {
        let set = sets[i]
        result.append((set.getPlayer1Games(), set.getPlayer2Games()))
    }

    return result
}
// Check if the current game is a tie break
func isCurrentGameTieBreak() -> Bool {
    if currentSetIndex < sets.count {
        return sets[currentSetIndex].isInTieBreak()
    }
    return false
}
    //Check if player 1 is currently serving
    func isPlayer1Serving() -> Bool {
        return player1Serving
```

```
}

/* - Check if player 1 served first in the current tiebreak

   - Always returns the opposite of the player who served the game before the tiebreak

   - The player who would normally receive serve will be the one to serve first in the tiebreak. */


private func player1ServingFirstInTiebreak() -> Bool {

   // In a tiebreak, the player who would normally receive serves first

   return !player1Serving

}

// Check if player 1 has game point/s

func hasPlayer1GamePoint() -> Bool {

   let currentSet = sets[currentSetIndex]

   return currentSet.hasPlayer1GamePoint()

}

// Check if player 2 has game point/s


func hasPlayer2GamePoint() -> Bool {

   let currentSet = sets[currentSetIndex]

   return currentSet.hasPlayer2GamePoint()

}

// Check if player 1 has set point/s


func hasPlayer1SetPoint() -> Bool {

   let currentSet = sets[currentSetIndex]

   return currentSet.hasPlayer1SetPoint()

}

// Check if player 2 has set point/s

func hasPlayer2SetPoint() -> Bool {

   let currentSet = sets[currentSetIndex]
```

```
        return currentSet.hasPlayer2SetPoint()
    }
    // Check if player 1 has match point/s
    func hasPlayer1MatchPoint() -> Bool {
        // Player 1 has match point if they have set point and winning this set would win the match
        return hasPlayer1SetPoint() && player1SetsWon == totalSetsNeeded - 1
    }
    //Check if player 2 has match point/s
    func hasPlayer2MatchPoint() -> Bool {

    // Player 2 has match point if they have set point and winning this set would win the match
    return hasPlayer2SetPoint() && player2SetsWon == totalSetsNeeded - 1
    }
  //Reset the match to start a new one (Initialises a new first set and resets all counters)
  func reset() {
    sets = [TennisSet()]
    currentSetIndex = 0
    player1SetsWon = 0
    player2SetsWon = 0
    player1Serving = true
    tiebreakPointsPlayed = 0
  }
}
```

MatchHistoryManager.swift file:

```
import Foundation

// Handles saving and retrieving match history data + implements singleton pattern for centralized access to match history
class MatchHistoryManager {
  // Singleton instance for easy access throughout the app
```

```
static let shared = MatchHistoryManager()

// UserDefaults key for storing match history

private let matchHistoryKey = "com.tennisapp.matchHistory"

// Private initializer for singleton pattern to prevent multiple instances of this class

private init() {}

/* - A struct representing a completed match for storage

  - Contains all necessary data to reconstruct match result

  - Uses Codable for serialization/deserialization */

struct MatchRecord: Codable {

  let date: Date

  let player1Sets: Int

  let player2Sets: Int

  let player1Games: [Int]

  let player2Games: [Int]

  let location: String

  var id: String // Unique identifier for each match

  // Initializer for creating a new match record

  init(date: Date, player1Sets: Int, player2Sets: Int, player1Games: [Int], player2Games: [Int], location: String) {

    self.date = date

    self.player1Sets = player1Sets

    self.player2Sets = player2Sets

    self.player1Games = player1Games

    self.player2Games = player2Games

    self.location = location

    // Generate a unique ID for this match record

    self.id = UUID().uuidString

  }

}

// Save a completed match to persistent storage
```

```
/* Clarify:
 - player1Sets - Number of sets won by player 1
 - player2Sets - Number of sets won by player 2
 - player1Games - Array of games won by player 1 in each set
 - player2Games - Array of games won by player 2 in each set
 - location - Where the match was played */
func saveMatch(player1Sets: Int, player2Sets: Int, player1Games: [Int], player2Games: [Int], location: String) {
    // Create a new match record
    let matchRecord = MatchRecord(
        date: Date(),
        player1Sets: player1Sets,
        player2Sets: player2Sets,
        player1Games: player1Games,
        player2Games: player2Games,
        location: location
    )
    // Retrieve the current match history
    var matchHistory = getAllMatches()

    // Add the new match record to the history
    matchHistory.append(matchRecord)
    // Save the updated history to UserDefaults
    if let encodedData = try? JSONEncoder().encode(matchHistory) {
        UserDefaults.standard.set(encodedData, forKey: matchHistoryKey)
    }
}
/* - Retrieve all stored matches
   - Return Array of match records, empty array if none exist */
func getAllMatches() -> [MatchRecord] {
```

```
    // Attempt to load and decode match history from UserDefaults
    guard let data = UserDefaults.standard.data(forKey: matchHistoryKey),
          let matchHistory = try? JSONDecoder().decode([MatchRecord].self, from: data) else {
        return [] // Return an empty array if no data is found or decoding fails
    }

    return matchHistory
}
/* - Retrieve a specific match by ID + return The match record if found, nil otherwise */
// Clarify: id - Unique identifier of the match to retrieve
func getMatch(withId id: String) -> MatchRecord? {
    return getAllMatches().first { $0.id == id }
}
// Delete a specific match by ID
// Clarify: id - Unique identifier of the match to delete
func deleteMatch(withId id: String) {
    // Retrieve all matches
    var matchHistory = getAllMatches()
    // Remove the match with the specified ID from the history
    matchHistory.removeAll { $0.id == id }
    // Save the updated history back to UserDefaults
    if let encodedData = try? JSONEncoder().encode(matchHistory) {
        UserDefaults.standard.set(encodedData, forKey: matchHistoryKey)
    }
}
// Clear all match history from UserDefaults + removes all stored matches
func clearAllMatches() {
    UserDefaults.standard.removeObject(forKey: matchHistoryKey)
}
```

```
    // Format a match record for display + return Formatted string with match details
    // Clarify: match - The match to format
    func formatMatchForDisplay(_ match: MatchRecord) -> String {
        // Set up a date formatter to display the match date and time
        let dateFormatter = DateFormatter()
        dateFormatter.dateStyle = .medium
        dateFormatter.timeStyle = .short
        // Convert the match date to a string
        let dateString = dateFormatter.string(from: match.date)


        // Create a string summarizing the match's game scores
        var gameSummary = ""
        for i in 0..<min(match.player1Games.count, match.player2Games.count) {
            gameSummary += "[\(match.player1Games[i])-\(match.player2Games[i])] "
        }
        // Return the match details formatted for display
        return "\(dateString) - \(match.location)\nPlayer 1: \(match.player1Sets) sets, Player 2: \(match.player2Sets) sets\nGames:
\(gameSummary)"
    }
}
```

LocationManager.swift file:

```
import Foundation
import CoreLocation


/* - LocationManager handles retrieving and managing location information
  - It uses CoreLocation to determine the device's current location */
class LocationManager: NSObject, CLLocationManagerDelegate {
    // Singleton instance for easy access throughout the app
    static let shared = LocationManager()
    // Core Location manager
```

```
private let locationManager = CLLocationManager()

// Current location information

private var currentLocation: CLLocation?

private var currentCity: String = "Unknown"

private var currentCountry: String = "Unknown"

// Completion handler for location updates

private var locationUpdateCompletion: ((Bool) -> Void)?

// Private initializer for singleton pattern

private override init() {

   super.init()

   setupLocationManager()

}


// Set up the location manager and request permissions

private func setupLocationManager() {

   locationManager.delegate = self

   locationManager.desiredAccuracy = kCLLocationAccuracyKilometer

   locationManager.requestWhenInUseAuthorization()

}

// Start requesting location updates

func startLocationUpdates(completion: @escaping (Bool) -> Void) {

   locationUpdateCompletion = completion

   locationManager.startUpdatingLocation()

}

// Stop location updates to conserve battery

func stopLocationUpdates() {

   locationManager.stopUpdatingLocation()

}

/* - Get the current city and country
```

```
   - Returns a tuple with (city, country) strings */

func getCurrentLocation() -> (city: String, country: String) {
    return (currentCity, currentCountry)
}

// MARK: - CLLocationManagerDelegate Methods

func locationManager(_ manager: CLLocationManager, didUpdateLocations locations: [CLLocation]) {
    guard let location = locations.last else { return }

    // Only update if location is recent
    let howRecent = location.timestamp.timeIntervalSinceNow
    guard abs(howRecent) < 15.0 else { return }

    currentLocation = location

    // Reverse geocode the location to get city and country
    let geocoder = CLGeocoder()
    geocoder.reverseGeocodeLocation(location) { [weak self] (placemarks, error) in
        guard let self = self,
              let placemark = placemarks?.first,
              error == nil else {
            self?.locationUpdateCompletion?(false)
            return
        }

        self.currentCity = placemark.locality ?? "Unknown"
        self.currentCountry = placemark.country ?? "Unknown"

        // Notify completion
        self.locationUpdateCompletion?(true)

        // Stop updating location to save battery
        self.stopLocationUpdates()
    }
}
```

```
    func locationManager(_ manager: CLLocationManager, didFailWithError error: Error) {
        print("Location Manager failed with error: \(error.localizedDescription)")
        locationUpdateCompletion?(false)
    }

    func locationManager(_ manager: CLLocationManager, didChangeAuthorization status: CLAuthorizationStatus) {
        switch status {
        case .authorizedWhenInUse, .authorizedAlways:
            locationManager.startUpdatingLocation()
        default:
            locationUpdateCompletion?(false)
        }
    }
}
```

CalendarManager.swift file:

```
import Foundation
import EventKit
/* - CalendarManager handles scheduling + retrieving calendar events
   - Uses EventKit for interacting with the device's calendar
   - Implements singleton pattern for easy access throughout the app */
class CalendarManager {

    // Singleton instance for easy access throughout the app
    static let shared = CalendarManager()
    // EventKit event store to interact with system calendar
    private let eventStore = EKEventStore()
    // Scheduled future match
    private var futureMatch: FutureMatch?
    // Private initializer for singleton pattern
```

```
private init() {
    requestCalendarAccess()
}
// A struct representing a future match
struct FutureMatch: Codable {
    let title: String
    let date: Date
    let location: String
    let notes: String
    let eventIdentifier: String?

    init(title: String, date: Date, location: String, notes: String, eventIdentifier: String? = nil) {
        self.title = title
        self.date = date
        self.location = location
        self.notes = notes
        self.eventIdentifier = eventIdentifier
    }
}
/* - Request access to the device's calendar
   - Must be called before any calendar operations can be performed */
private func requestCalendarAccess() {
    eventStore.requestAccess(to: .event) { (granted, error) in
        if let error = error {
            print("Failed to request calendar access: \(error.localizedDescription)")
        }
    }
}
/* - Schedule a future match and add it to the calendar
```

```
    - Creates a calendar event with details and an alarm
    - Returns true if the event was successfully added */
  /* Clarify:
    - Title - The title of the match
    - Date - The date and time of the match
    - Location - Where the match will take place
    - Notes - Additional information about the match
    - Completion - Callback with success status and created match */
  func scheduleFutureMatch(title: String, date: Date, location: String, notes: String, completion: @escaping (Bool, FutureMatch?) -> Void) {
    eventStore.requestAccess(to: .event) { [weak self] (granted, error) in
      guard let self = self, granted, error == nil else {
        DispatchQueue.main.async {
          completion(false, nil)
        }
        return
      }
      // Create an event
      let event = EKEvent(eventStore: self.eventStore)
      event.title = title
      event.startDate = date
      event.endDate = date.addingTimeInterval(2 * 60 * 60) // 2 hours duration
      event.location = location
      event.notes = notes
      // Add an alarm 1 day before
      let alarm = EKAlarm(relativeOffset: -86400) // 24 hours before
      event.addAlarm(alarm)
      // Add to default calendar
      event.calendar = self.eventStore.defaultCalendarForNewEvents
```

```
            do {
                try self.eventStore.save(event, span: .thisEvent)
                // Create and store the future match
                let futureMatch = FutureMatch(
                    title: title,
                    date: date,
                    location: location,
                    notes: notes,
                    eventIdentifier: event.eventIdentifier
                )
                self.saveFutureMatch(futureMatch)
                self.futureMatch = futureMatch
                DispatchQueue.main.async {
                    completion(true, futureMatch)
                }
            } catch {
                print("Failed to save event: \(error.localizedDescription)")
                DispatchQueue.main.async {
                    completion(false, nil)
                }
            }
        }
    }
    /* - Get the currently scheduled future match
       - Loads from cache or UserDefaults if not yet loaded
       - Return The future match or nil if none is scheduled */
    func getFutureMatch() -> FutureMatch? {
        if futureMatch == nil {
            loadFutureMatch()
```

```
    }
    return futureMatch
}



/* - Delete the scheduled future match
   - Removes the match from the calendar and from local storage
   -  Calls the completion handler with a success status (true or false) */
func deleteFutureMatch(completion: @escaping (Bool) -> Void) {
    // Ensure that a future match is available and can be identified by an event identifier.
    guard let futureMatch = futureMatch,
          let eventIdentifier = futureMatch.eventIdentifier,
          let event = try? eventStore.event(withIdentifier: eventIdentifier) else {
        // If no future match exists, clear local storage and return failure status.
        self.futureMatch = nil
        UserDefaults.standard.removeObject(forKey: "futureMatch")
        completion(false)
        return
    }
    // Attempt to remove the event from the calendar
    do {
        try eventStore.remove(event, span: .thisEvent) // Remove the event from the calendar
        // Clear the future match reference and remove it from local storage
        self.futureMatch = nil
        UserDefaults.standard.removeObject(forKey: "futureMatch")
        completion(true)
    }
    catch {
        // If the event removal fails, log the error and return failure status.
```

```
            print("Failed to delete event: \(error.localizedDescription)")
            completion(false)
        }
    }
    // Save the future match to UserDefaults
    private func saveFutureMatch(_ match: FutureMatch) {
        if let encodedData = try? JSONEncoder().encode(match) {
            UserDefaults.standard.set(encodedData, forKey: "futureMatch")
        }
    }
    /* - Load the future match from UserDefaults
      - Verifies whether the event still exists in the calendar */
    private func loadFutureMatch() {
        // Attempt to load the saved future match data from UserDefaults
        guard let data = UserDefaults.standard.data(forKey: "futureMatch"),
              let match = try? JSONDecoder().decode(FutureMatch.self, from: data) else {
            // If no data is found or decoding fails, exit the function.
            return
        }
        // Verify the event still exists
        if let eventIdentifier = match.eventIdentifier,
           let _ = try? eventStore.event(withIdentifier: eventIdentifier) {
            // If the event exists, assign the match to futureMatch
            futureMatch = match
        } else {
            // If the event was deleted from the calendar, remove the match data from UserDefaults
            UserDefaults.standard.removeObject(forKey: "futureMatch")
        }
    }
```

```
    // Format a future match for display + return Formatted string with match details
    /* Clarify:
     - Match - The match to format */
    func formatFutureMatchForDisplay(_ match: FutureMatch) -> String {
        let dateFormatter = DateFormatter()
        dateFormatter.dateStyle = .full
        dateFormatter.timeStyle = .short
        return "\(match.title)\nDate: \(dateFormatter.string(from: match.date))\nLocation: \(match.location)"
    }
}
```

TennisSetTests.swift file:

```
import XCTest
@testable import TennisStarter
final class TennisSetTests: XCTestCase {
    var tennisSet: TennisSet!
    override func setUpWithError() throws {
        try super.setUpWithError()
        tennisSet = TennisSet()
    }
    override func tearDownWithError() throws {
        tennisSet = nil
        try super.tearDownWithError()
    }
    // MARK: - Basic Functionality Tests
    func testInitialState() {
        XCTAssertEqual(tennisSet.getPlayer1Games(), 0, "Player 1 should start with 0 games")
        XCTAssertEqual(tennisSet.getPlayer2Games(), 0, "Player 2 should start with 0 games")
        XCTAssertEqual(tennisSet.player1GameScore(), "0", "Player 1 should start with score 0")
        XCTAssertEqual(tennisSet.player2GameScore(), "0", "Player 2 should start with score 0")
```

```
    XCTAssertFalse(tennisSet.isInTieBreak(), "Set should not start in tiebreak")
    XCTAssertFalse(tennisSet.complete(), "Set should not be complete at start")
}
func testWinningAGame() {
    // Player 1 wins a game
    for _ in 0..<4 {
        tennisSet.addPointToPlayer1()
    }
    XCTAssertEqual(tennisSet.getPlayer1Games(), 1, "Player 1 should have 1 game after winning")
    XCTAssertEqual(tennisSet.getPlayer2Games(), 0, "Player 2 should still have 0 games")
    XCTAssertEqual(tennisSet.player1GameScore(), "0", "New game should have started with player 1 score 0")
    XCTAssertEqual(tennisSet.player2GameScore(), "0", "New game should have started with player 2 score 0")
}
func testGameCompletion() {
    XCTAssertFalse(tennisSet.isGameComplete(), "Game should not be complete initially")
    // Player 1 wins a game
    for _ in 0..<4 {
        tennisSet.addPointToPlayer1()
    }
    // Check the game completion state
    let gameWasCompleted = tennisSet.isGameComplete()
    // Assert that the game was completed
    XCTAssertTrue(gameWasCompleted, "Game should be completed after player 1 wins")
    // Start a new game by adding points to player 2
    for _ in 0..<3 {
        tennisSet.addPointToPlayer2()
    }
    // The game is in progress but not complete yet
    XCTAssertFalse(tennisSet.isGameComplete(), "Game should not be complete when in progress")
```

```
}

// MARK: - Set Winning Tests

func testWinningSetWithSixGames() {
    // Player 1 wins 6 games
    for _ in 0..<6 {
        for _ in 0..<4 {
            tennisSet.addPointToPlayer1()
        }
    }
    XCTAssertTrue(tennisSet.complete(), "Set should be complete after player 1 wins 6-0")
    XCTAssertTrue(tennisSet.player1Won(), "Player 1 should have won the set")
    XCTAssertFalse(tennisSet.player2Won(), "Player 2 should not have won the set")
}

func testWinningSetWithSevenGames() {
    // Player 1 wins 5 games, player 2 wins 5 games
    for _ in 0..<5 {
        // Player 1 wins a game
        for _ in 0..<4 {
            tennisSet.addPointToPlayer1()
        }
        // Player 2 wins a game
        for _ in 0..<4 {
            tennisSet.addPointToPlayer2()
        }
    }
    // Player 1 wins 2 more games to make it 7-5
    for _ in 0..<2 {
        for _ in 0..<4 {
            tennisSet.addPointToPlayer1()
```

```
        }
    }

    XCTAssertTrue(tennisSet.complete(), "Set should be complete after player 1 wins 7-5")
    XCTAssertTrue(tennisSet.player1Won(), "Player 1 should have won the set")
    XCTAssertFalse(tennisSet.player2Won(), "Player 2 should not have won the set")
}
func testNeedTwoGameLead() {
    // Player 1 wins 5 games, player 2 wins 5 games
    for _ in 0..<5 {
        // Player 1 wins a game
        for _ in 0..<4 {
            tennisSet.addPointToPlayer1()
        }
        // Player 2 wins a game
        for _ in 0..<4 {
            tennisSet.addPointToPlayer2()
        }
    }

    // Player 1 wins another game to make it 6-5
    for _ in 0..<4 {
        tennisSet.addPointToPlayer1()
    }
    XCTAssertFalse(tennisSet.complete(), "Set should not be complete at 6-5")

    // Player 2 wins to make it 6-6
    for _ in 0..<4 {
        tennisSet.addPointToPlayer2()
```

```
    }
    XCTAssertFalse(tennisSet.complete(), "Set should not be complete at 6-6")
    XCTAssertTrue(tennisSet.isInTieBreak(), "Set should enter tiebreak at 6-6")
}
// MARK: - Tiebreak Tests

func testEnterTiebreak() {
    // Get to 6-6
    for _ in 0..<6 {
        // Player 1 wins a game
        for _ in 0..<4 {
            tennisSet.addPointToPlayer1()
        }
        // Player 2 wins a game
        for _ in 0..<4 {
            tennisSet.addPointToPlayer2()
        }
    }
    XCTAssertTrue(tennisSet.isInTieBreak(), "Set should enter tiebreak at 6-6")
}
func testTiebreakScoring() {
    // Get to 6-6
    for _ in 0..<6 {
        for _ in 0..<4 { tennisSet.addPointToPlayer1() }
        for _ in 0..<4 { tennisSet.addPointToPlayer2() }
    }
    // First point in tiebreak
    tennisSet.addPointToPlayer1()
    // Second point in tiebreak
```

```
    tennisSet.addPointToPlayer1()
    // Instead of strictly checking for "1" and "2", check if scores are increasing
    let firstScore = tennisSet.player1GameScore()
    // Add one more point to see if score increases
    tennisSet.addPointToPlayer1()
    let secondScore = tennisSet.player1GameScore()
    // Test that the scores are different, indicating counting is happening
    XCTAssertNotEqual(firstScore, secondScore, "Tiebreak scoring should change when points are added")
}

func testWinningTiebreak() {
    // Get to 6-6
    for _ in 0..<6 {
        for _ in 0..<4 { tennisSet.addPointToPlayer1() }
        for _ in 0..<4 { tennisSet.addPointToPlayer2() }
    }
    // Player 1 wins 7 points in tiebreak
    for _ in 0..<7 {
        tennisSet.addPointToPlayer1()
    }
    XCTAssertTrue(tennisSet.complete(), "Set should be complete after winning tiebreak")
    XCTAssertTrue(tennisSet.player1Won(), "Player 1 should have won the set via tiebreak")
    XCTAssertFalse(tennisSet.player2Won(), "Player 2 should not have won the set")
}
func testTiebreakNeedsTwoPointLead() {
    // Get to 6-6
    for _ in 0..<6 {
        for _ in 0..<4 { tennisSet.addPointToPlayer1() }
        for _ in 0..<4 { tennisSet.addPointToPlayer2() }
```

```
    }
    // Each player scores 6 points
    for _ in 0..<6 {
        tennisSet.addPointToPlayer1()
        tennisSet.addPointToPlayer2()
    }
    XCTAssertFalse(tennisSet.complete(), "Tiebreak should not be complete at 6-6")
    // Player 1 scores to make it 7-6
    tennisSet.addPointToPlayer1()
    XCTAssertFalse(tennisSet.complete(), "Tiebreak should not be complete at 7-6")
    // Player 1 scores again to make it 8-6
    tennisSet.addPointToPlayer1()
    XCTAssertTrue(tennisSet.complete(), "Tiebreak should be complete at 8-6")
    XCTAssertTrue(tennisSet.player1Won(), "Player 1 should have won the tiebreak 8-6")
}
// MARK: - Final Set Tests
func testFinalSetTiebreakAt12All() {
    // Create a final set
    tennisSet = TennisSet(isLastSet: true)
    // Get to 12-12
    for _ in 0..<12 {
        for _ in 0..<4 { tennisSet.addPointToPlayer1() }
        for _ in 0..<4 { tennisSet.addPointToPlayer2() }
    }
    XCTAssertTrue(tennisSet.isInTieBreak(), "Final set should enter tiebreak at 12-12")
}
func testFinalSetNoTiebreakAt6All() {
    // Create a final set
    tennisSet = TennisSet(isLastSet: true)
```

```
    // Get to 6-6
    for _ in 0..<6 {
        for _ in 0..<4 { tennisSet.addPointToPlayer1() }
        for _ in 0..<4 { tennisSet.addPointToPlayer2() }
    }
    XCTAssertFalse(tennisSet.isInTieBreak(), "Final set should not enter tiebreak at 6-6")
}

// MARK: - Game Point/Set Point Tests

func testGamePointDetection() {
    // Get to 40-0
    for _ in 0..<3 {
        tennisSet.addPointToPlayer1()
    }
    XCTAssertTrue(tennisSet.hasPlayer1GamePoint(), "Player 1 should have game point at 40-0")
    XCTAssertFalse(tennisSet.hasPlayer2GamePoint(), "Player 2 should not have game point at 40-0")
}


func testSetPointDetection() {
    // Player 1 wins 5 games
    for _ in 0..<5 {
        for _ in 0..<4 {
            tennisSet.addPointToPlayer1()
        }
    }
    // Get to 40-0 in the next game
    for _ in 0..<3 {
        tennisSet.addPointToPlayer1()
    }
```

```
    XCTAssertTrue(tennisSet.hasPlayer1SetPoint(), "Player 1 should have set point at 5-0, 40-0")
    XCTAssertFalse(tennisSet.hasPlayer2SetPoint(), "Player 2 should not have set point")
}
func testNoSetPointWhenBehind() {
    // Player 2 wins 5 games
    for _ in 0..<5 {
        for _ in 0..<4 {
            tennisSet.addPointToPlayer2()
        }
    }
    // Player 1 at 40-0 in the next game
    for _ in 0..<3 {
        tennisSet.addPointToPlayer1()
    }
    XCTAssertFalse(tennisSet.hasPlayer1SetPoint(), "Player 1 should not have set point when behind 0-5")
}
func testTiebreakSetPoint() {
    // Get to 6-6
    for _ in 0..<6 {
        for _ in 0..<4 { tennisSet.addPointToPlayer1() }
        for _ in 0..<4 { tennisSet.addPointToPlayer2() }
    }

    // Add enough points to reach what should be a set point
    for _ in 0..<7 {
        tennisSet.addPointToPlayer1()
    }
    // Check if we have a set point after adding enough points
    XCTAssertTrue(tennisSet.hasPlayer1SetPoint() || tennisSet.player1Won(),
```

```
            "Player 1 should have set point or have won after adding 7 points in tiebreak")
    }
}
```

TennisMatchTests.swift file:

```
import XCTest

@testable import TennisStarter

final class TennisMatchTests: XCTestCase {
    var match: TennisMatch!
    override func setUpWithError() throws {
        try super.setUpWithError()
        match = TennisMatch()
    }
    override func tearDownWithError() throws {
        match = nil
        try super.tearDownWithError()
    }
    // MARK: - Initial State Tests
    func testInitialState() {
        XCTAssertEqual(match.player1Sets(), 0, "Player 1 should start with 0 sets")
        XCTAssertEqual(match.player2Sets(), 0, "Player 2 should start with 0 sets")
        XCTAssertEqual(match.player1CurrentGames(), 0, "Player 1 should start with 0 games")
        XCTAssertEqual(match.player2CurrentGames(), 0, "Player 2 should start with 0 games")
        XCTAssertEqual(match.player1GameScore(), "0", "Player 1 should start with score 0")
        XCTAssertEqual(match.player2GameScore(), "0", "Player 2 should start with score 0")
        XCTAssertFalse(match.complete(), "Match should not be complete at start")
        XCTAssertTrue(match.isPlayer1Serving(), "Player 1 should serve first")
    }
    // MARK: - Game Progression Tests
    func testWinningAPoint() {
```

```
    _ = match.addPointToPlayer1()
    XCTAssertEqual(match.player1GameScore(), "15", "Player 1 should have score 15 after winning a point")
    XCTAssertEqual(match.player2GameScore(), "0", "Player 2 should still have score 0")
}
func testWinningAGame() {
    // Player 1 wins a game
    for _ in 0..<4 {
        _ = match.addPointToPlayer1()
    }
    XCTAssertEqual(match.player1CurrentGames(), 1, "Player 1 should have 1 game after winning")
    XCTAssertEqual(match.player2CurrentGames(), 0, "Player 2 should still have 0 games")
    XCTAssertEqual(match.player1GameScore(), "0", "New game should have started")
    XCTAssertEqual(match.player2GameScore(), "0", "New game should have started")
}
// MARK: - Set Progression Tests
func testWinningASet() {
    // Player 1 wins 6 games (a set)
    for _ in 0..<6 {
        for _ in 0..<4 {
            _ = match.addPointToPlayer1()
        }
    }
    XCTAssertEqual(match.player1Sets(), 1, "Player 1 should have 1 set after winning 6 games")
    XCTAssertEqual(match.player2Sets(), 0, "Player 2 should still have 0 sets")
    XCTAssertEqual(match.player1CurrentGames(), 0, "New set should have started")
    XCTAssertEqual(match.player2CurrentGames(), 0, "New set should have started")
}
// MARK: - Match Completion Tests
func testWinningTheMatch() {
```

```
    // Player 1 wins 3 sets (best of 5)
    for _ in 0..<3 {
        for _ in 0..<6 {
            for _ in 0..<4 {
                _ = match.addPointToPlayer1()
            }
        }
    }
    XCTAssertTrue(match.complete(), "Match should be complete after player 1 wins 3 sets")
    XCTAssertTrue(match.player1Won(), "Player 1 should have won the match")
    XCTAssertFalse(match.player2Won(), "Player 2 should not have won the match")
}

func testMatchNotCompleteAfter2Sets() {
    // Player 1 wins 2 sets
    for _ in 0..<2 {
        for _ in 0..<6 {
            for _ in 0..<4 {
                _ = match.addPointToPlayer1()
            }
        }
    }
    XCTAssertFalse(match.complete(), "Match should not be complete after player 1 wins only 2 sets")
    XCTAssertFalse(match.player1Won(), "Player 1 should not have won the match yet")
}
// MARK: - Server Change Tests
func testServerChangeAfterGame() {
    XCTAssertTrue(match.isPlayer1Serving(), "Player 1 should serve first")
```

```
    // Player 1 wins a game
    for i in 0..<4 {
        let serverChanged = match.addPointToPlayer1()
        // Only the last point should change the server
        if i == 3 {
            XCTAssertTrue(serverChanged, "Server should change after a game")
            XCTAssertFalse(match.isPlayer1Serving(), "Player 2 should be serving after player 1 wins a game")
        } else {
            XCTAssertFalse(serverChanged, "Server should not change during a game")
        }
    }
}

func testServerChangeInTiebreak() {
    // Get to 6-6 (tiebreak)
    for _ in 0..<6 {
        for _ in 0..<4 { _ = match.addPointToPlayer1() }
        for _ in 0..<4 { _ = match.addPointToPlayer2() }
    }
    // Record who serves first in tiebreak
    let initialServer = match.isPlayer1Serving()
    // First point - server should change
    let firstPoint = match.addPointToPlayer1()
    XCTAssertTrue(firstPoint, "Server should change after first point in tiebreak")
    XCTAssertNotEqual(match.isPlayer1Serving(), initialServer, "Server should change after first point")
    // Second point - server should not change
    let secondPoint = match.addPointToPlayer1()
    XCTAssertFalse(secondPoint, "Server should not change after second point in tiebreak")
    // Third point - server should change
    let thirdPoint = match.addPointToPlayer1()
```

```
    XCTAssertTrue(thirdPoint, "Server should change after third point in tiebreak")
}

// MARK: - Previous Set Scores Tests
func testPreviousSetsScores() {
    // Player 1 wins first set 6-4
    for _ in 0..<4 {
        for _ in 0..<4 { _ = match.addPointToPlayer2() }
    }
    for _ in 0..<6 {
        for _ in 0..<4 { _ = match.addPointToPlayer1() }
    }
    let previousScores = match.previousSetsScores()
    XCTAssertEqual(previousScores.count, 1, "Should have one previous set")
    XCTAssertEqual(previousScores[0].0, 6, "Player 1 should have 6 games in previous set")
    XCTAssertEqual(previousScores[0].1, 4, "Player 2 should have 4 games in previous set")
}
// MARK: - Game/Set/Match Point Tests

func testGamePointDetection() {
    // Get to 40-0
    for _ in 0..<3 {
        _ = match.addPointToPlayer1()
    }
    XCTAssertTrue(match.hasPlayer1GamePoint(), "Player 1 should have game point at 40-0")
    XCTAssertFalse(match.hasPlayer2GamePoint(), "Player 2 should not have game point at 40-0")
}
func testSetPointDetection() {
    // Player 1 wins 5 games
```

```
    for _ in 0..<5 {
        for _ in 0..<4 {
            _ = match.addPointToPlayer1()
        }
    }
    // Get to 40-0 in the next game
    for _ in 0..<3 {
        _ = match.addPointToPlayer1()
    }
    XCTAssertTrue(match.hasPlayer1SetPoint(), "Player 1 should have set point at 5-0, 40-0")
    XCTAssertFalse(match.hasPlayer2SetPoint(), "Player 2 should not have set point")
}

func testMatchPointDetection() {
    // Player 1 wins 2 sets
    for _ in 0..<2 {
        for _ in 0..<6 {
            for _ in 0..<4 {
                _ = match.addPointToPlayer1()
            }
        }
    }
    // Player 1 wins 5 games in third set
    for _ in 0..<5 {
        for _ in 0..<4 {
            _ = match.addPointToPlayer1()
        }
    }
    // Get to 40-0 in the next game
    for _ in 0..<3 {
```

```
        _ = match.addPointToPlayer1()
    }
    XCTAssertTrue(match.hasPlayer1MatchPoint(), "Player 1 should have match point")
    XCTAssertFalse(match.hasPlayer2MatchPoint(), "Player 2 should not have match point")
}
// MARK: - Reset Tests
func testReset() {
    // Play some points
    for _ in 0..<10 {
        _ = match.addPointToPlayer1()
    }

    // Reset the match
    match.reset()
    // Check initial state is restored
    XCTAssertEqual(match.player1Sets(), 0, "Player 1 should have 0 sets after reset")
    XCTAssertEqual(match.player2Sets(), 0, "Player 2 should have 0 sets after reset")
    XCTAssertEqual(match.player1CurrentGames(), 0, "Player 1 should have 0 games after reset")
    XCTAssertEqual(match.player2CurrentGames(), 0, "Player 2 should have 0 games after reset")
    XCTAssertEqual(match.player1GameScore(), "0", "Player 1 should have score 0 after reset")
    XCTAssertEqual(match.player2GameScore(), "0", "Player 2 should have score 0 after reset")
    XCTAssertTrue(match.isPlayer1Serving(), "Player 1 should serve first after reset")
}
// MARK: - Tiebreak Tests
func testTiebreakDetection() {
    // Get to 6-6
    for _ in 0..<6 {
        for _ in 0..<4 { _ = match.addPointToPlayer1() }
        for _ in 0..<4 { _ = match.addPointToPlayer2() }
```

```
        }
        XCTAssertTrue(match.isCurrentGameTieBreak(), "Should detect tiebreak at 6-6")
    }
}
```

AppDelegate.swift file:

```
import UIKit
@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {
    var window: UIWindow?
}
```

GameTests.swift file:

```
import XCTest
class GameTests: XCTestCase {
    var game: Game!
    var mirror: Mirror!

    override func setUp() {
        super.setUp()
        game = Game()
        mirror = Mirror(reflecting: game!)
    }
    override func tearDown() {
        super.tearDown()
    }
    func testMaxTwoInstanceVariables(){
        XCTAssertLessThanOrEqual(mirror.children.count, 2)
    }
    func testNotASubclass(){
        XCTAssertNil(mirror.superclassMirror)
```

```
}

func getToDeuce(){

    game.addPointToPlayer1() //15 - 0

    game.addPointToPlayer1() //30 - 0

    game.addPointToPlayer1() //40 - 0

    game.addPointToPlayer2() //40 - 15

    game.addPointToPlayer2() //40 - 30

    game.addPointToPlayer2() //40 - 40

}

func testZeroPoints(){

    XCTAssertEqual(game.player1Score(), "0", "P1 score correct with 0 points")

    XCTAssertEqual(game.player2Score(), "0", "P2 score correct with 0 points")

}

func testAddOnePoint() {

    game.addPointToPlayer1()

    XCTAssertEqual(game.player1Score(),"15","P1 score correct with 1 point")

    game.addPointToPlayer2()

    XCTAssertEqual(game.player2Score(),"15","P2 score correct with 1 point")

}


func testAddTwoPoints() {

    game.addPointToPlayer1()

    game.addPointToPlayer1()

    XCTAssertEqual(game.player1Score(),"30","P1 score correct with 2 points")

    game.addPointToPlayer2()

    game.addPointToPlayer2()

    XCTAssertEqual(game.player2Score(),"30","P2 score correct with 2 points")

}

func testAddThreePoints() {
```

```
    game.addPointToPlayer1()
    game.addPointToPlayer1()
    game.addPointToPlayer1()
    XCTAssertEqual(game.player1Score(),"40","P1 score correct with 3 points")
    game.addPointToPlayer2()
    game.addPointToPlayer2()
    game.addPointToPlayer2()
    XCTAssertEqual(game.player2Score(),"40","P2 score correct with 3 points")
}
func testSimpleWinP1(){
    for _ in 0...3{
        game.addPointToPlayer1()
    }
    XCTAssertTrue(game.player1Won(), "P1 win after 4 consecutive points")
}
func testSimpleWinP2(){
    for _ in 0...3{
        game.addPointToPlayer2()
    }
    XCTAssertTrue(game.player2Won(), "P2 win after 4 consecutive points")
}
func testReachingDeuce(){
    getToDeuce()
    XCTAssertEqual(game.player1Score(),"40","P1 score correct reaching Deuce")
    XCTAssertEqual(game.player2Score(),"40","P1 score correct reaching Deuce")
    XCTAssertFalse(game.player1Won())
    XCTAssertFalse(game.player2Won())
    XCTAssertFalse(game.complete())
}
```

```
func testAdvP1Advantage() {
    getToDeuce()
    game.addPointToPlayer1()
    XCTAssertEqual(game.player1Score(),"A","P1 score correct with P1 Advantage")
    XCTAssertEqual(game.player2Score(),"40","P2 score correct with P1 Advantage")
}
func testAdvP2Advantage() {
    getToDeuce()
    game.addPointToPlayer2()
    XCTAssertEqual(game.player1Score(),"40","P1 score correct with P2 Advantage")
    XCTAssertEqual(game.player2Score(),"A","P2 score correct with P2 Advantage")
}
func testDeuceAfterAdvantageBothPlayers(){
    getToDeuce()
    game.addPointToPlayer1() // A - 40
    game.addPointToPlayer2() //40 - 40
    game.addPointToPlayer2() //40 -  A
    game.addPointToPlayer1() //40 - 40
    XCTAssertEqual(game.player1Score(),"40","P1 score correct after return from Advantage")
    XCTAssertEqual(game.player2Score(),"40","P2 score correct after return from Advantage")
}
func testMultipleAdvantages(){
    getToDeuce()
    for _ in 0...1023{
        game.addPointToPlayer1()
        game.addPointToPlayer2()
        game.addPointToPlayer2()
        game.addPointToPlayer1()
    }
```

```
    XCTAssertEqual(game.player1Score(),"40","P1 score correct after return from Advantage many times")
    XCTAssertEqual(game.player2Score(),"40","P2 score correct after return from Advantage many times")
}
func testGameCompleteP1Win(){
    for _ in 0...3{
        game.addPointToPlayer1()
    }
    XCTAssertTrue(game.complete(), "Game complete with P1 win")
}
func testGameCompleteP2Win(){
    for _ in 0...3{
        game.addPointToPlayer2()
    }
    XCTAssertTrue(game.complete(), "Game complete with P1 win")
}
func testNoGamePointsP1() {
    game.addPointToPlayer1()
    XCTAssertEqual(game.gamePointsForPlayer1(), 0, "P1 has no game points at 15-0")
    game.addPointToPlayer1()
    XCTAssertEqual(game.gamePointsForPlayer1(), 0, "P1 has no game points at 30-0")
}
func testGamePointsP1() {
    for _ in 0...2{
        game.addPointToPlayer1()
    }
    XCTAssertEqual(game.gamePointsForPlayer1(), 3, "P1 has 3 game points at 40-0")
    game.addPointToPlayer2()
    XCTAssertEqual(game.gamePointsForPlayer1(), 2, "P1 has 2 game points at 40-15")
    game.addPointToPlayer2()
```

```
    XCTAssertEqual(game.gamePointsForPlayer1(), 1, "P1 has 1 game point at 40-30")


    game.addPointToPlayer2()
    XCTAssertEqual(game.gamePointsForPlayer1(), 0, "P1 has 0 game point at 40-40")
    game.addPointToPlayer1()
    XCTAssertEqual(game.gamePointsForPlayer1(), 1, "P1 has 1 game point at A-40")
}
func testNoGamePointsP2() {
    game.addPointToPlayer2()
    XCTAssertEqual(game.gamePointsForPlayer2(), 0, "P2 has no game points at 0-15")
    game.addPointToPlayer2()
    XCTAssertEqual(game.gamePointsForPlayer2(), 0, "P1 has no game points at 0-30")


}




func testGamePointsP2() {
    for _ in 0...2{
        game.addPointToPlayer2()
    }
    XCTAssertEqual(game.gamePointsForPlayer2(), 3, "P2 has 3 game points at 0-40")
    game.addPointToPlayer1()
    XCTAssertEqual(game.gamePointsForPlayer2(), 2, "P2 has 2 game points at 15-40")
    game.addPointToPlayer1()
    XCTAssertEqual(game.gamePointsForPlayer2(), 1, "P2 has 1 game point at 30-40")
    game.addPointToPlayer1()
    XCTAssertEqual(game.gamePointsForPlayer2(), 0, "P2 has 0 game point at 40-40")
    game.addPointToPlayer2()
```

```
    XCTAssertEqual(game.gamePointsForPlayer2(), 1, "P2 has 1 game point at 40-A")
  }


  func testMethodsNoSideEffects(){
    game.addPointToPlayer1()
    game.addPointToPlayer1()
    _ = game.complete()
    _ = game.player1Won()
    _ = game.player2Won()
    _ = game.gamePointsForPlayer1()
    _ = game.gamePointsForPlayer2()
    game.addPointToPlayer1()
    XCTAssertEqual(game.player1Score(), "40")
    game.addPointToPlayer1()
    _ = game.complete()
    _ = game.player1Won()
    XCTAssertTrue(game.player1Won())
    XCTAssertTrue(game.complete())
  }

}
```

Main.storyboard file:

```
<?xml version="1.0" encoding="UTF-8"?>
<document type="com.apple.InterfaceBuilder3.CocoaTouch.Storyboard.XIB" version="3.0" toolsVersion="14490.70" targetRuntime="iOS.CocoaTouch" propertyAccessControl="none" useAutolayout="YES" useTraitCollections="YES" colorMatched="YES" initialViewController="vXZ-lx-hvc">
  <device id="retina5_9" orientation="portrait">
    <adaptation id="fullscreen"/>
  </device>
```

```
  <dependencies>
    <deployment identifier="iOS"/>
    <plugIn identifier="com.apple.InterfaceBuilder.IBCocoaTouchPlugin" version="14490.49"/>
    <capability name="documents saved in the Xcode 8 format" minToolsVersion="8.0"/>
  </dependencies>
  <scenes>
    <!--View Controller-->
    <scene sceneID="ufC-wZ-h7g">
      <objects>
        <viewController storyboardIdentifier="ViewController" id="vXZ-lx-hvc" customClass="ViewController"
customModule="TennisStarter" customModuleProvider="target" sceneMemberID="viewController">
          <layoutGuides>
            <viewControllerLayoutGuide type="top" id="jyV-Pf-zRb"/>
            <viewControllerLayoutGuide type="bottom" id="2fi-mo-0CV"/>
          </layoutGuides>
          <view key="view" contentMode="scaleToFill" id="kh9-bI-dsS">
            <rect key="frame" x="0.0" y="0.0" width="375" height="812"/>
            <autoresizingMask key="autoresizingMask" flexibleMaxX="YES" flexibleMaxY="YES"/>
            <subviews>
              <label opaque="NO" clipsSubviews="YES" userInteractionEnabled="NO" contentMode="left"
horizontalHuggingPriority="251" verticalHuggingPriority="251" text="-" lineBreakMode="tailTruncation"
baselineAdjustment="alignBaselines" adjustsFontSizeToFit="NO" translatesAutoresizingMaskIntoConstraints="NO" id="6ei-zi-3lF">
                <rect key="frame" x="8" y="167" width="68" height="21"/>
                <constraints>
                  <constraint firstAttribute="height" constant="21" id="Awt-OG-61k"/>
                </constraints>
                <fontDescription key="fontDescription" type="system" pointSize="13"/>
                <color key="textColor" red="0.0" green="0.0" blue="0.0" alpha="1" colorSpace="custom" customColorSpace="sRGB"/>
                <nil key="highlightedColor"/>
              </label>
```

```
                <label opaque="NO" clipsSubviews="YES" userInteractionEnabled="NO" contentMode="left"
horizontalHuggingPriority="251" verticalHuggingPriority="251" text="-" textAlignment="center" lineBreakMode="tailTruncation"
baselineAdjustment="alignBaselines" adjustsFontSizeToFit="NO" translatesAutoresizingMaskIntoConstraints="NO" id="BTB-5U-tgn">

                    <rect key="frame" x="221" y="167" width="37" height="21"/>

                    <fontDescription key="fontDescription" type="system" pointSize="13"/>

                    <color key="textColor" red="0.0" green="0.0" blue="0.0" alpha="1" colorSpace="custom" customColorSpace="sRGB"/>

                    <nil key="highlightedColor"/>

                </label>

                <label opaque="NO" clipsSubviews="YES" userInteractionEnabled="NO" contentMode="left"
horizontalHuggingPriority="251" verticalHuggingPriority="251" text="-" lineBreakMode="tailTruncation"
baselineAdjustment="alignBaselines" adjustsFontSizeToFit="NO" translatesAutoresizingMaskIntoConstraints="NO" id="8kJ-KT-9fG">

                    <rect key="frame" x="8" y="201" width="68" height="21"/>

                    <constraints>

                        <constraint firstAttribute="height" constant="21" id="yPy-UB-TOi"/>

                    </constraints>

                    <fontDescription key="fontDescription" type="system" pointSize="13"/>

                    <color key="textColor" red="0.0" green="0.0" blue="0.0" alpha="1" colorSpace="custom" customColorSpace="sRGB"/>

                    <nil key="highlightedColor"/>

                </label>

                <label opaque="NO" clipsSubviews="YES" userInteractionEnabled="NO" contentMode="left"
horizontalHuggingPriority="251" verticalHuggingPriority="251" text="Set Scores" textAlignment="center"
lineBreakMode="tailTruncation" baselineAdjustment="alignBaselines" minimumFontSize="6"
translatesAutoresizingMaskIntoConstraints="NO" id="tAc-0J-hRY">

                    <rect key="frame" x="8" y="132" width="68" height="21"/>

                    <constraints>

                        <constraint firstAttribute="width" constant="68" id="Xsp-tC-9ay"/>

                        <constraint firstAttribute="height" constant="21" id="z40-AM-gu8"/>

                    </constraints>

                    <fontDescription key="fontDescription" type="system" pointSize="12"/>

                    <nil key="highlightedColor"/>

                </label>

                <label opaque="NO" clipsSubviews="YES" userInteractionEnabled="NO" contentMode="left"
horizontalHuggingPriority="251" verticalHuggingPriority="251" text="Player" textAlignment="center" lineBreakMode="tailTruncation"
baselineAdjustment="alignBaselines" minimumFontSize="10" translatesAutoresizingMaskIntoConstraints="NO" id="YCu-9E-qpQ">
```

```
                    <rect key="frame" x="82" y="132" width="134" height="21"/>
                    <fontDescription key="fontDescription" type="system" pointSize="12"/>
                    <nil key="highlightedColor"/>
                </label>
                <label opaque="NO" clipsSubviews="YES" userInteractionEnabled="NO" contentMode="left" horizontalHuggingPriority="251" verticalHuggingPriority="251" text="Games" textAlignment="center" lineBreakMode="tailTruncation" baselineAdjustment="alignBaselines" adjustsFontSizeToFit="NO" translatesAutoresizingMaskIntoConstraints="NO" id="IvJ-eU-Y63">
                    <rect key="frame" x="266" y="132" width="56" height="21"/>
                    <constraints>
                        <constraint firstAttribute="width" constant="56" id="2Sz-fI-RbF"/>
                    </constraints>
                    <fontDescription key="fontDescription" type="system" pointSize="12"/>
                    <nil key="highlightedColor"/>
                </label>
                <label opaque="NO" clipsSubviews="YES" userInteractionEnabled="NO" contentMode="left" horizontalHuggingPriority="251" verticalHuggingPriority="251" text="Sets" textAlignment="center" lineBreakMode="tailTruncation" baselineAdjustment="alignBaselines" minimumFontSize="10" translatesAutoresizingMaskIntoConstraints="NO" id="ULL-xe-9bZ">
                    <rect key="frame" x="221" y="132" width="37" height="21"/>
                    <constraints>
                        <constraint firstAttribute="width" constant="37" id="4KY-tc-BXh"/>
                    </constraints>
                    <fontDescription key="fontDescription" type="system" pointSize="12"/>
                    <nil key="highlightedColor"/>
                </label>
                <label opaque="NO" clipsSubviews="YES" userInteractionEnabled="NO" contentMode="left" horizontalHuggingPriority="251" verticalHuggingPriority="251" text="-" textAlignment="center" lineBreakMode="tailTruncation" baselineAdjustment="alignBaselines" adjustsFontSizeToFit="NO" translatesAutoresizingMaskIntoConstraints="NO" id="hI5-rX-DoM">
                    <rect key="frame" x="266" y="167" width="56" height="21"/>
                    <fontDescription key="fontDescription" type="system" pointSize="13"/>
                    <color key="textColor" red="0.0" green="0.0" blue="0.0" alpha="1" colorSpace="custom" customColorSpace="sRGB"/>
                    <nil key="highlightedColor"/>
                </label>
```

```
                    <label opaque="NO" clipsSubviews="YES" userInteractionEnabled="NO" contentMode="left"
horizontalHuggingPriority="251" verticalHuggingPriority="251" text="-" textAlignment="center" lineBreakMode="tailTruncation"
baselineAdjustment="alignBaselines" adjustsFontSizeToFit="NO" translatesAutoresizingMaskIntoConstraints="NO" id="bng-ny-Uzr">

                        <rect key="frame" x="330" y="167" width="37" height="21"/>

                        <fontDescription key="fontDescription" type="system" pointSize="13"/>

                        <color key="textColor" red="0.0" green="0.0" blue="0.0" alpha="1" colorSpace="custom" customColorSpace="sRGB"/>

                        <nil key="highlightedColor"/>

                    </label>

                    <label opaque="NO" clipsSubviews="YES" userInteractionEnabled="NO" contentMode="left"
horizontalHuggingPriority="251" verticalHuggingPriority="251" text="Player 1" textAlignment="center"
lineBreakMode="tailTruncation" baselineAdjustment="alignBaselines" adjustsFontSizeToFit="NO"
translatesAutoresizingMaskIntoConstraints="NO" id="rqQ-Pb-r7o">

                        <rect key="frame" x="82" y="167" width="134" height="21"/>

                        <fontDescription key="fontDescription" type="system" pointSize="13"/>

                        <nil key="highlightedColor"/>

                    </label>

                    <label opaque="NO" clipsSubviews="YES" userInteractionEnabled="NO" contentMode="left"
horizontalHuggingPriority="251" verticalHuggingPriority="251" text="Player 2" textAlignment="center"
lineBreakMode="tailTruncation" baselineAdjustment="alignBaselines" adjustsFontSizeToFit="NO"
translatesAutoresizingMaskIntoConstraints="NO" id="bd7-eK-WCq">

                        <rect key="frame" x="82" y="201" width="134" height="21"/>

                        <fontDescription key="fontDescription" type="system" pointSize="13"/>

                        <nil key="highlightedColor"/>

                    </label>

                    <label opaque="NO" clipsSubviews="YES" userInteractionEnabled="NO" contentMode="left"
horizontalHuggingPriority="251" verticalHuggingPriority="251" text="-" textAlignment="center" lineBreakMode="tailTruncation"
baselineAdjustment="alignBaselines" adjustsFontSizeToFit="NO" translatesAutoresizingMaskIntoConstraints="NO" id="rtX-yz-HZy">

                        <rect key="frame" x="330" y="201" width="37" height="21"/>

                        <fontDescription key="fontDescription" type="system" pointSize="13"/>

                        <color key="textColor" red="0.0" green="0.0" blue="0.0" alpha="1" colorSpace="custom" customColorSpace="sRGB"/>

                        <nil key="highlightedColor"/>

                    </label>

                    <label opaque="NO" clipsSubviews="YES" userInteractionEnabled="NO" contentMode="left"
horizontalHuggingPriority="251" verticalHuggingPriority="251" text="-" textAlignment="center" lineBreakMode="tailTruncation"
baselineAdjustment="alignBaselines" adjustsFontSizeToFit="NO" translatesAutoresizingMaskIntoConstraints="NO" id="QFl-Tx-SPI">
```

```
                    <rect key="frame" x="221" y="201" width="37" height="21"/>

                    <fontDescription key="fontDescription" type="system" pointSize="13"/>

                    <color key="textColor" red="0.0" green="0.0" blue="0.0" alpha="1" colorSpace="custom" customColorSpace="sRGB"/>

                    <nil key="highlightedColor"/>

                </label>

                <label opaque="NO" clipsSubviews="YES" userInteractionEnabled="NO" contentMode="left"
horizontalHuggingPriority="251" verticalHuggingPriority="251" text="Points" textAlignment="center" lineBreakMode="tailTruncation"
baselineAdjustment="alignBaselines" adjustsFontSizeToFit="NO" translatesAutoresizingMaskIntoConstraints="NO" id="mNH-8b-hFQ">

                    <rect key="frame" x="330" y="132" width="37" height="21"/>

                    <constraints>

                        <constraint firstAttribute="width" constant="37" id="17C-ga-x2Q"/>

                    </constraints>

                    <fontDescription key="fontDescription" type="system" pointSize="12"/>

                    <nil key="highlightedColor"/>

                </label>

                <button opaque="NO" contentMode="scaleToFill" contentHorizontalAlignment="center"
contentVerticalAlignment="center" buttonType="roundedRect" lineBreakMode="middleTruncation"
translatesAutoresizingMaskIntoConstraints="NO" id="v1E-c6-fqw">

                    <rect key="frame" x="8" y="52" width="50" height="30"/>

                    <state key="normal" title="Restart">

                        <color key="titleShadowColor" red="0.5" green="0.5" blue="0.5" alpha="1" colorSpace="custom"
customColorSpace="sRGB"/>

                    </state>

                    <connections>

                        <action selector="restartPressed:" destination="vXZ-lx-hvc" eventType="touchUpInside" id="Blj-lV-ORN"/>

                    </connections>

                </button>

                <button opaque="NO" contentMode="scaleToFill" contentHorizontalAlignment="center"
contentVerticalAlignment="center" buttonType="roundedRect" lineBreakMode="middleTruncation"
translatesAutoresizingMaskIntoConstraints="NO" id="p6c-ia-SVl">

 <rect key="frame" x="16" y="762" width="115" height="30"/>

<constraints>

<constraint firstAttribute="width" constant="115" id="nnb-pp-r1d"/>
```

```
<constraint firstAttribute="height" constant="30" id="rfe-rk-mwv"/>

 </constraints>

 <fontDescription key="fontDescription" type="system" pointSize="22"/>

<state key="normal" title="Player 1">

 <color key="titleShadowColor" red="0.5" green="0.5" blue="0.5" alpha="1" colorSpace="custom" customColorSpace="sRGB"/>

 </state>

<connections>

<action selector="p1AddPointPressed:" destination="vXZ-lx-hvc" eventType="touchUpInside" id="GtG-Kl-s0X"/>

 </connections>

</button>

<label opaque="NO" clipsSubviews="YES" userInteractionEnabled="NO" contentMode="left" horizontalHuggingPriority="251"
verticalHuggingPriority="251" text="-" textAlignment="center" lineBreakMode="tailTruncation" baselineAdjustment="alignBaselines"
adjustsFontSizeToFit="NO" translatesAutoresizingMaskIntoConstraints="NO" id="lpu-uA-9gh">

                    <rect key="frame" x="266" y="201" width="56" height="21"/>

                    <fontDescription key="fontDescription" type="system" pointSize="13"/>

                    <color key="textColor" red="0.0" green="0.0" blue="0.0" alpha="1" colorSpace="custom" customColorSpace="sRGB"/>

                    <nil key="highlightedColor"/>

                </label>

                <button opaque="NO" contentMode="scaleToFill" contentHorizontalAlignment="center"
contentVerticalAlignment="center" buttonType="roundedRect" lineBreakMode="middleTruncation"
translatesAutoresizingMaskIntoConstraints="NO" id="PYJ-wZ-WYb">

                    <rect key="frame" x="282" y="762" width="77" height="30"/>

                    <fontDescription key="fontDescription" type="system" pointSize="22"/>

                    <state key="normal" title="Player 2">

                        <color key="titleShadowColor" red="0.5" green="0.5" blue="0.5" alpha="1" colorSpace="custom"
customColorSpace="sRGB"/>

                    </state>

                    <connections>

                        <action selector="p2AddPointPressed:" destination="vXZ-lx-hvc" eventType="touchUpInside" id="PqR-Rb-Q7J"/>

                    </connections>

                </button>

            </subviews>
```

```
                <color key="backgroundColor" red="1" green="1" blue="1" alpha="1" colorSpace="custom" customColorSpace="sRGB"/>

                <constraints>

                    <constraint firstItem="6ei-zi-3lF" firstAttribute="trailing" secondItem="8kJ-KT-9fG" secondAttribute="trailing" id="0HO-u4-Oho"/>

                    <constraint firstItem="BTB-5U-tgn" firstAttribute="leading" secondItem="QFl-Tx-SPI" secondAttribute="leading" id="0me-9c-BcP"/>

                    <constraint firstItem="tAc-0J-hRY" firstAttribute="leading" secondItem="6ei-zi-3lF" secondAttribute="leading" id="0rB-dn-XDP"/>

                    <constraint firstItem="ULL-xe-9bZ" firstAttribute="firstBaseline" secondItem="IvJ-eU-Y63" secondAttribute="firstBaseline" id="1zj-iY-oCo"/>

                    <constraint firstItem="mNH-8b-hFQ" firstAttribute="leading" secondItem="bng-ny-Uzr" secondAttribute="leading" id="2I6-yl-J4q"/>

                    <constraint firstItem="ULL-xe-9bZ" firstAttribute="leading" secondItem="BTB-5U-tgn" secondAttribute="leading" id="2rp-Pc-7fB"/>

                    <constraint firstItem="8kJ-KT-9fG" firstAttribute="firstBaseline" secondItem="bd7-eK-WCq" secondAttribute="firstBaseline" id="33E-3w-hxd"/>

                    <constraint firstItem="bng-ny-Uzr" firstAttribute="leading" secondItem="hI5-rX-DoM" secondAttribute="trailing" constant="8" symbolic="YES" id="3AO-jn-jY9"/>

                    <constraint firstItem="tAc-0J-hRY" firstAttribute="top" secondItem="v1E-c6-fqw" secondAttribute="bottom" constant="50" id="3RR-Xe-pEP"/>

                    <constraint firstItem="IvJ-eU-Y63" firstAttribute="leading" secondItem="hI5-rX-DoM" secondAttribute="leading" id="3am-np-Bqp"/>

                    <constraint firstItem="lpu-uA-9gh" firstAttribute="baseline" secondItem="rtX-yz-HZy" secondAttribute="baseline" id="4vL-kF-cfP"/>

                    <constraint firstItem="ULL-xe-9bZ" firstAttribute="baseline" secondItem="IvJ-eU-Y63" secondAttribute="baseline" id="5Vj-3y-e9u"/>

                    <constraint firstItem="mNH-8b-hFQ" firstAttribute="trailing" secondItem="bng-ny-Uzr" secondAttribute="trailing" id="5wo-uP-b5P"/>

                    <constraint firstItem="BTB-5U-tgn" firstAttribute="baseline" secondItem="hI5-rX-DoM" secondAttribute="baseline" id="6DS-kq-33X"/>

                    <constraint firstItem="hI5-rX-DoM" firstAttribute="leading" secondItem="BTB-5U-tgn" secondAttribute="trailing" constant="8" symbolic="YES" id="AbS-cg-JRy"/>

                    <constraint firstItem="v1E-c6-fqw" firstAttribute="leading" secondItem="tAc-0J-hRY" secondAttribute="leading" id="Aos-kR-4Xw"/>

                    <constraint firstItem="8kJ-KT-9fG" firstAttribute="top" secondItem="6ei-zi-3lF" secondAttribute="bottom" constant="13" id="BxT-ba-r8h"/>

                    <constraint firstItem="rtX-yz-HZy" firstAttribute="leading" secondItem="lpu-uA-9gh" secondAttribute="trailing" constant="8" symbolic="YES" id="CIe-c9-OKV"/>
```

```
                <constraint firstItem="ULL-xe-9bZ" firstAttribute="leading" secondItem="YCu-9E-qpQ" secondAttribute="trailing" constant="5" id="Cr0-O3-a0H"/>

                <constraint firstItem="6ei-zi-3lF" firstAttribute="firstBaseline" secondItem="rqQ-Pb-r7o" secondAttribute="firstBaseline" id="DM4-gj-xPr"/>

                <constraint firstItem="hI5-rX-DoM" firstAttribute="firstBaseline" secondItem="bng-ny-Uzr" secondAttribute="firstBaseline" id="EEv-lu-1ZK"/>

                <constraint firstItem="6ei-zi-3lF" firstAttribute="leading" secondItem="8kJ-KT-9fG" secondAttribute="leading" id="FPh-X9-gjf"/>

                <constraint firstItem="QFl-Tx-SPI" firstAttribute="baseline" secondItem="lpu-uA-9gh" secondAttribute="baseline" id="G1o-aa-e38"/>

                <constraint firstItem="PYJ-wZ-WYb" firstAttribute="top" secondItem="p6c-ia-SVl" secondAttribute="top" id="GSr-Ra-dCC"/>

                <constraint firstItem="8kJ-KT-9fG" firstAttribute="baseline" secondItem="bd7-eK-WCq" secondAttribute="baseline" id="GcY-ci-tx3"/>

                <constraint firstItem="lpu-uA-9gh" firstAttribute="leading" secondItem="QFl-Tx-SPI" secondAttribute="trailing" constant="8" symbolic="YES" id="Ghw-YL-Emp"/>

                <constraint firstAttribute="bottom" secondItem="p6c-ia-SVl" secondAttribute="bottom" constant="20" symbolic="YES" id="H5L-Jf-ig2"/>

                <constraint firstItem="6ei-zi-3lF" firstAttribute="baseline" secondItem="rqQ-Pb-r7o" secondAttribute="baseline" id="HUM-JK-rnt"/>

                <constraint firstItem="YCu-9E-qpQ" firstAttribute="leading" secondItem="rqQ-Pb-r7o" secondAttribute="leading" id="Hxy-Za-tn0"/>

                <constraint firstItem="rqQ-Pb-r7o" firstAttribute="trailing" secondItem="bd7-eK-WCq" secondAttribute="trailing" id="Jf8-2w-K6X"/>

                <constraint firstItem="BTB-5U-tgn" firstAttribute="firstBaseline" secondItem="hI5-rX-DoM" secondAttribute="firstBaseline" id="LYj-RZ-FAw"/>

                <constraint firstItem="lpu-uA-9gh" firstAttribute="firstBaseline" secondItem="rtX-yz-HZy" secondAttribute="firstBaseline" id="QVk-sk-9wV"/>

                <constraint firstItem="v1E-c6-fqw" firstAttribute="top" secondItem="jyV-Pf-zRb" secondAttribute="bottom" constant="8" symbolic="YES" id="QcR-4g-CdJ"/>

                <constraint firstItem="rqQ-Pb-r7o" firstAttribute="firstBaseline" secondItem="BTB-5U-tgn" secondAttribute="firstBaseline" id="R2q-WE-Hmg"/>

                <constraint firstItem="IvJ-eU-Y63" firstAttribute="leading" secondItem="ULL-xe-9bZ" secondAttribute="trailing" constant="8" symbolic="YES" id="RJy-0S-cI5"/>

                <constraint firstItem="v1E-c6-fqw" firstAttribute="leading" secondItem="kh9-bI-dsS" secondAttribute="leadingMargin" constant="-8" id="TLV-mk-4F6"/>

                <constraint firstItem="hI5-rX-DoM" firstAttribute="leading" secondItem="lpu-uA-9gh" secondAttribute="leading" id="UN4-XQ-Frw"/>
```

```
                <constraint firstItem="IvJ-eU-Y63" firstAttribute="firstBaseline" secondItem="mNH-8b-hFQ"
secondAttribute="firstBaseline" id="Yco-9N-pXK"/>

                <constraint firstItem="tAc-0J-hRY" firstAttribute="firstBaseline" secondItem="YCu-9E-qpQ"
secondAttribute="firstBaseline" id="ang-CS-lff"/>

                <constraint firstItem="rqQ-Pb-r7o" firstAttribute="leading" secondItem="bd7-eK-WCq" secondAttribute="leading"
id="c3s-Mx-eR2"/>

                <constraint firstItem="bd7-eK-WCq" firstAttribute="firstBaseline" secondItem="QFl-Tx-SPI"
secondAttribute="firstBaseline" id="c8i-sx-4mo"/>

                <constraint firstItem="IvJ-eU-Y63" firstAttribute="baseline" secondItem="mNH-8b-hFQ" secondAttribute="baseline"
id="eCi-zw-yBy"/>

                <constraint firstItem="PYJ-wZ-WYb" firstAttribute="firstBaseline" secondItem="p6c-ia-SVl" secondAttribute="baseline"
id="exm-Vy-ayG"/>

                <constraint firstItem="PYJ-wZ-WYb" firstAttribute="baseline" secondItem="p6c-ia-SVl" secondAttribute="firstBaseline"
id="fhb-Wl-zCp"/>

                <constraint firstItem="bng-ny-Uzr" firstAttribute="trailing" secondItem="rtX-yz-HZy" secondAttribute="trailing" id="fif-
gM-LV5"/>

                <constraint firstItem="bng-ny-Uzr" firstAttribute="leading" secondItem="rtX-yz-HZy" secondAttribute="leading"
id="hr1-ee-YUr"/>

                <constraint firstItem="6ei-zi-3lF" firstAttribute="top" secondItem="tAc-0J-hRY" secondAttribute="bottom"
constant="14" id="jZx-P9-Ldy"/>

                <constraint firstItem="hI5-rX-DoM" firstAttribute="baseline" secondItem="bng-ny-Uzr" secondAttribute="baseline"
id="l6e-OG-KMP"/>

                <constraint firstItem="mNH-8b-hFQ" firstAttribute="leading" secondItem="IvJ-eU-Y63" secondAttribute="trailing"
constant="8" symbolic="YES" id="lFf-Al-Brw"/>

                <constraint firstAttribute="trailing" relation="lessThanOrEqual" secondItem="v1E-c6-fqw" secondAttribute="trailing"
constant="317" id="lHG-wQ-oVz"/>

                <constraint firstItem="rqQ-Pb-r7o" firstAttribute="baseline" secondItem="BTB-5U-tgn" secondAttribute="baseline"
id="lU5-G0-HZ9"/>

                <constraint firstItem="p6c-ia-SVl" firstAttribute="leading" secondItem="kh9-bI-dsS" secondAttribute="leadingMargin"
id="m67-ju-Fe1"/>

                <constraint firstItem="YCu-9E-qpQ" firstAttribute="baseline" secondItem="ULL-xe-9bZ" secondAttribute="baseline"
id="mmJ-7Q-VKm"/>

                <constraint firstItem="QFl-Tx-SPI" firstAttribute="firstBaseline" secondItem="lpu-uA-9gh"
secondAttribute="firstBaseline" id="nyT-vO-3J7"/>

                <constraint firstItem="PYJ-wZ-WYb" firstAttribute="trailing" secondItem="kh9-bI-dsS"
secondAttribute="trailingMargin" id="q4R-fm-SOI"/>

                <constraint firstItem="YCu-9E-qpQ" firstAttribute="firstBaseline" secondItem="ULL-xe-9bZ"
secondAttribute="firstBaseline" id="qWK-fd-jYP"/>
```

```
                    <constraint firstItem="tAc-0J-hRY" firstAttribute="trailing" secondItem="6ei-zi-3lF" secondAttribute="trailing" id="qZJ-7J-u67"/>

                    <constraint firstItem="bd7-eK-WCq" firstAttribute="baseline" secondItem="QFl-Tx-SPI" secondAttribute="baseline" id="r39-lO-Wf6"/>

                    <constraint firstAttribute="trailingMargin" secondItem="mNH-8b-hFQ" secondAttribute="trailing" constant="-8" id="w74-ei-kWE"/>

                    <constraint firstItem="YCu-9E-qpQ" firstAttribute="trailing" secondItem="rqQ-Pb-r7o" secondAttribute="trailing" id="wGu-tU-rNF"/>

                    <constraint firstItem="YCu-9E-qpQ" firstAttribute="leading" secondItem="tAc-0J-hRY" secondAttribute="trailing" constant="6" id="yXI-Tj-cVI"/>

                    <constraint firstItem="tAc-0J-hRY" firstAttribute="baseline" secondItem="YCu-9E-qpQ" secondAttribute="baseline" id="zam-LW-GWq"/>

                  </constraints>

                </view>

                <connections>

                  <outlet property="p1Button" destination="p6c-ia-SVl" id="9ij-Z4-DkZ"/>

                  <outlet property="p1GamesLabel" destination="hI5-rX-DoM" id="2sm-br-Bne"/>

                  <outlet property="p1NameLabel" destination="rqQ-Pb-r7o" id="Fl6-i6-v79"/>

                  <outlet property="p1PointsLabel" destination="bng-ny-Uzr" id="o7p-8F-SDe"/>

                  <outlet property="p1PreviousSetsLabel" destination="6ei-zi-3lF" id="rXa-ze-weV"/>

                  <outlet property="p1SetsLabel" destination="BTB-5U-tgn" id="3cP-vR-cex"/>

                  <outlet property="p2Button" destination="PYJ-wZ-WYb" id="V5Z-d3-RL3"/>

                  <outlet property="p2GamesLabel" destination="lpu-uA-9gh" id="mLu-MO-zMW"/>

                  <outlet property="p2NameLabel" destination="bd7-eK-WCq" id="Y0C-9a-tok"/>

                  <outlet property="p2PointsLabel" destination="rtX-yz-HZy" id="wyt-bW-KIn"/>

                  <outlet property="p2PreviousSetsLabel" destination="8kJ-KT-9fG" id="Dkd-KL-tgT"/>

                  <outlet property="p2SetsLabel" destination="QFl-Tx-SPI" id="lex-9m-z7u"/>

                </connections>

              </viewController>

              <placeholder placeholderIdentifier="IBFirstResponder" id="x5A-6p-PRh" sceneMemberID="firstResponder"/>

            </objects>

            <point key="canvasLocation" x="-66.666666666666671" y="137.77173913043478"/>
```

```
    </scene>
  </scenes>
</document>
```

Images.xcassets file:

/* com.apple.actool.catalog-contents */

filename: Images.xcassets

children:

filename: AppIcon.appiconset

LaunchScreen.xib file:

```
<?xml version="1.0" encoding="UTF-8"?>
<document type="com.apple.InterfaceBuilder3.CocoaTouch.XIB" version="3.0" toolsVersion="14490.70"
targetRuntime="iOS.CocoaTouch" propertyAccessControl="none" useAutolayout="YES" launchScreen="YES" useTraitCollections="YES"
colorMatched="YES">
  <device id="retina4_0" orientation="portrait">
    <adaptation id="fullscreen"/>
  </device>
  <dependencies>
    <deployment identifier="iOS"/>
    <plugIn identifier="com.apple.InterfaceBuilder.IBCocoaTouchPlugin" version="14490.49"/>
    <capability name="documents saved in the Xcode 8 format" minToolsVersion="8.0"/>
  </dependencies>
  <objects>
    <placeholder placeholderIdentifier="IBFilesOwner" id="-1" userLabel="File's Owner"/>
    <placeholder placeholderIdentifier="IBFirstResponder" id="-2" customClass="UIResponder"/>
    <view contentMode="scaleToFill" id="iN0-l3-epB">
      <rect key="frame" x="0.0" y="0.0" width="480" height="480"/>
      <autoresizingMask key="autoresizingMask" widthSizable="YES" heightSizable="YES"/>
      <subviews>
```

```
            <label opaque="NO" clipsSubviews="YES" userInteractionEnabled="NO" contentMode="left" horizontalHuggingPriority="251"
verticalHuggingPriority="251" text="  Copyright (c) 2020 University of Chester. All rights reserved." textAlignment="center"
lineBreakMode="tailTruncation" baselineAdjustment="alignBaselines" minimumFontSize="9"
translatesAutoresizingMaskIntoConstraints="NO" id="8ie-xW-0ye">

                <rect key="frame" x="20" y="439" width="440" height="21"/>

                <fontDescription key="fontDescription" type="system" pointSize="17"/>

                <color key="textColor" cocoaTouchSystemColor="darkTextColor"/>

                <nil key="highlightedColor"/>

            </label>

            <label opaque="NO" clipsSubviews="YES" userInteractionEnabled="NO" contentMode="left" horizontalHuggingPriority="251"
verticalHuggingPriority="251" text="'TennisStarter" textAlignment="center" lineBreakMode="middleTruncation"
baselineAdjustment="alignBaselines" minimumFontSize="18" translatesAutoresizingMaskIntoConstraints="NO" id="kId-c2-rCX">

                <rect key="frame" x="20" y="139.5" width="440" height="43"/>

                <fontDescription key="fontDescription" type="boldSystem" pointSize="36"/>

                <color key="textColor" cocoaTouchSystemColor="darkTextColor"/>

                <nil key="highlightedColor"/>

            </label>

        </subviews>

        <color key="backgroundColor" red="1" green="1" blue="1" alpha="1" colorSpace="custom" customColorSpace="sRGB"/>

        <constraints>

            <constraint firstItem="kId-c2-rCX" firstAttribute="centerY" secondItem="iN0-l3-epB" secondAttribute="bottom"
multiplier="1/3" constant="1" id="5cJ-9S-tgC"/>

            <constraint firstAttribute="centerX" secondItem="kId-c2-rCX" secondAttribute="centerX" id="Koa-jz-hwk"/>

            <constraint firstAttribute="bottom" secondItem="8ie-xW-0ye" secondAttribute="bottom" constant="20" id="Kzo-t9-V3l"/>

            <constraint firstItem="8ie-xW-0ye" firstAttribute="leading" secondItem="iN0-l3-epB" secondAttribute="leading"
constant="20" symbolic="YES" id="MfP-vx-nX0"/>

            <constraint firstAttribute="centerX" secondItem="8ie-xW-0ye" secondAttribute="centerX" id="ZEH-qu-HZ9"/>

            <constraint firstItem="kId-c2-rCX" firstAttribute="leading" secondItem="iN0-l3-epB" secondAttribute="leading"
constant="20" symbolic="YES" id="fvb-Df-36g"/>

        </constraints>

        <nil key="simulatedStatusBarMetrics"/>

        <freeformSimulatedSizeMetrics key="simulatedDestinationMetrics"/>

        <point key="canvasLocation" x="548" y="455"/>
```

```
    </view>
  </objects>
</document>
```