Sample Code

Creating real-time games

Develop games where multiple players interact in real time.

Download

iOS 17.4+ | iPadOS 17.4+ | Xcode 15.3+

Overview

This sample code project uses the GameKit framework to create a simple real-time game where two players are immediately aware of the actions each other takes. The sample shows both the matchmaker interface that enables players to invite others, and the automatch feature that lets Game Center find players. It provides access to Game Center using the access point and dashboard to show the players' progress toward an achievement, as well as scores on the game leaderboard. It uses the match object to send game data and text messages between players. It also uses voice chat for additional communications.

To create your own real-time game, replace the SwiftUI files with your gameplay interface, and modify the `RealTimeGame` class to support your game. To extend this sample to use matchmaking rules, see Finding players using matchmaking rules.

Configure the sample code project

To configure the sample code project, perform the following steps in Xcode:

1. Choose Xcode > Settings > Accounts, click the Add button (+), and add your Apple ID account.
2. Select the project in the Project navigator, click the Signing & Capabilities pane, select the target, and assign your team to it. Optionally, enter a unique identifier in the Bundle Identifier text field. Otherwise, use the existing bundle ID that ends in your team ID.
3. If necessary, add the Game Center capability (see Enabling and configuring Game Center).

4. In App Store Connect, create an app record that matches the bundle ID (see Add a new app in App Store Connect Help).
5. Add a classic leaderboard with `123` as the leaderboard ID, Integer as the score format type, Best Score as the score submission type, High to Low as the sort order, and `0` to `100` as the score range. Add one localization with an integer format. See Configure leaderboards.
6. Add an achievement with `1234` as the achievement ID, `100` as the point value, and No for the hidden option (see Configure achievements). Add one localization with an image.
7. Connect two iOS devices to your Mac.
8. If necessary, click Register Device in the Signing & Capabilities pane to create the provisioning profile.
9. Build and run the sample on the two iOS devices.
0. If the Welcome to Game Center sheet appears, sign in using a different Apple ID on each device.

Initialize the player and register for real-time events

Before using any GameKit APIs, the game needs to initialize the local player by presenting an interface for them to sign in to Game Center on their device. The `RealTimeGame.authenticatePlayer()` method handles the initialization flow and, when complete, registers for real-time game events.

```
// Register for real-time invitations from other players.
GKLocalPlayer.local.register(self)
```

Start a real-time match

To start a real-time game, the player taps Choose Player on the content view. Then the `RealTimeGame.choosePlayer()` method creates a match request for a two-player game and presents a `GKMatchmakerViewController` interface where the player can invite friends or automatch to fill empty slots.

```
// Create a match request.
let request = GKMatchRequest()
request.minPlayers = 2
request.maxPlayers = 2

// If you use matchmaking rules, set the GKMatchRequest.queueName property here. If
// game-specific properties, set the local player's GKMatchRequest.properties too.
```

```
// Present the interface where the player selects opponents and starts the game.
if let viewController = GKMatchmakerViewController(matchRequest: request) {
    viewController.matchmakerDelegate = self
    rootViewController?.present(viewController, animated: true) { }
}
```

After the player selects an opponent in the Game Center interface, GameKit sends the invitation and changes the view controller interface to show the invitation status.

After the opponent accepts the invitation by tapping the notification that appears, GameKit launches the game on their device and invokes the `GKLocalPlayerListener.player(didAccept:)` protocol method. This method presents the matchmaker view controller that shows the connection status.

```
// Present the matchmaker view controller in the invitation state.
if let viewController = GKMatchmakerViewController(invite: invite) {
    viewController.matchmakerDelegate = self
    rootViewController?.present(viewController, animated: true) { }
}
```

When the status of the players changes from Connecting to Ready in the interface, GameKit invokes the `GKMatchmakerViewControllerDelegate.matchmakerViewController(_:didFind:)` method in both game instances, passing the local view controller and new `GKMatch` object. The `matchmakerViewController(_:didFind:)` method first dismisses the view controller and then starts the match by invoking the `RealTimeGame.startMyMatchWith(match:)` method.

The `startMyMatchWith(match:)` method sets the `playingGame` property to `true`, which displays the game interface, and loads the opponent's avatar.

Automatch the local player with others

To let Game Center find an opponent without presenting the matchmaker view controller, the player taps the Automatch toggle. The `RealTimeGame.findPlayer()` method creates a match request and passes it to the `GKMatchmaker.findMatch(for:withCompletionHandler:)` method. When a player running another game instance taps the Automatch toggle, the `GKMatchmaker.findMatch(for:withCompletionHandler:)` method returns a match object.

Then the `findPlayer()` method invokes `startMyMatchWith(match:)`, which sets the match's delegate so that when the opponent connects to the match, GameKit calls the `GKMatchDelegate.match(_:player:didChange:)` delegate method. The game view displays a

placeholder for the opponent's name until they connect. If the state is `.connected` when GameKit invokes the `RealTimeGame.match(_:player:didChange:)` method, it sets the `opponent` property and loads the opponent's avatar, which updates the game view.

To stop finding players, the player taps Automatch again and it invokes `GKMatchmaker.shared().cancel()`.

Exchange game data between players

Each time the player takes an action, the game shares the results with the other player. When the player taps Take Action, their score increases in both game instances. If the player taps End Game or Forfeit, both game instances exit the match.

The `RealTimeGame.takeAction()` method increments the score, then encodes and sends the game data to the opponent.

```
do {
    let data = encode(score: myScore)
    try myMatch?.sendData(toAllPlayers: data!, with: GKMatch.SendDataMode.unreliable
} catch {
    print("Error: \(error.localizedDescription).")
}
```

When the player taps the message bubble in the game view, the chat view sheet appears so the player can send a text message to the opponent. The `sendMessage(content:)` method sends the text message as game data.

Similarly, the `endMatch()` and `forfeitMatch()` methods encode and send game data, except they send the game outcome.

In the recipient's game instance, GameKit invokes the `GKMatchDelegate.match(_:didReceive:fromRemotePlayer:)` method. The `RealTimeGame` implementation of `match(_:didReceive:fromRemotePlayer:)` decodes the data object and updates the game state, depending on its contents.

For example, this method exits the match if the opponent ends or forfeits the game. In both game instances, an alert appears showing each player the outcome of the game. When the player taps OK, the game returns to the content view.

Start voice chat between players

When the player taps the telephone bubble in the game view, the `RealTimeGame.startVoiceChat()` method starts a voice chat audio session with the opponent.

First it creates a GKVoiceChat object, providing a unique name for the channel, sets a connection change handler, and sets the volume.

```
if voiceChat == nil {
    // Create the voice chat object.
    voiceChat = myMatch?.voiceChat(withName: "RealTimeGameChannel")
}

// Exit early if the app can't start a voice chat session.
guard let voiceChat = voiceChat else { return }

// Handle an unknown, connected, or disconnected player state.
voiceChat.playerVoiceChatStateDidChangeHandler = voiceChatChangeHandler

// Set the audio volume.
voiceChat.volume = 0.5
```

Next it activates a shared audio session.

```
do {
    let audioSession = AVAudioSession.sharedInstance()

    try audioSession.setActive(true, options: [])
} catch {
    print("AVAudioSession error: \(error.localizedDescription).")
}
```

Then it starts and activates the voice chat.

```
voiceChat.start()
voiceChat.isActive = true
```

After both players tap the telephone bubble, the state changes to `GKVoiceChat.PlayerState.connected` and they can begin speaking.

The first time the sample starts voice chat, the system displays a dialog asking the player whether the sample may use the microphone. This dialog displays the value of the NSMicrophoneUsageDescription information property list key as the reason for requesting access permission.

When the player taps the telephone bubble again, the `stopVoiceChat()` method stops the voice chat and deactivates the shared audio session.

```
// Stop the voice chat.
voiceChat?.stop()
voiceChat = nil

// Deactivate the shared audio session.
do {
    let audioSession = AVAudioSession.sharedInstance()

    try audioSession.setActive(false, options: [])
} catch {
    print("Error: \(error.localizedDescription).")
}
```

You can enhance this sample to provide more feedback and controls, such as:

- Show when players connect to the voice chat and start speaking in the connection change handler.
- Add a control to mute the voice chat using the `muteVoiceChat()` and `unmuteVoiceChat()` methods.
- Let players change the volume using the `voiceChat(volume:)` method.
- Notify the player when the opponent wants to voice chat.

Connect players with friends

This sample uses Game Center features that let players request friends and let the game access the player's friends list.

When the player taps Add Friends in the Friends section of the content view, the `RealTimeGame.addFriends()` method presents a view controller that allows the player to send a friend request to another player.

```
do {
    try GKLocalPlayer.local.presentFriendRequestCreator(from: viewController)
} catch {
    print("Error: \(error.localizedDescription).")
}
```

When the player taps Access Friends, the `accessFriends()` method checks the authorization status before accessing the player's friends.

```
let authorizationStatus = try await GKLocalPlayer.local.loadFriendsAuthorizationStat
```

If the status is `GKFriendsAuthorizationStatus.notDetermined` or `GKFriendsAuthorizationStatus.authorized`, the method loads the friends and displays their names in a sheet.

```
// Load the local player's friends.
let players = try await GKLocalPlayer.local.loadFriends()
```

The first time the sample loads the player's friends, the system displays a dialog asking the player whether the sample may access their friends. This dialog displays the value of the `NSGKFriendListUsageDescription` information property list key as the reason for requesting access permission.

An error occurs if the sample calls `loadFriends()` without permission or without providing a reason using the `NSGKFriendListUsageDescription` key.

Report progress toward an achievement

The `startMyMatchWith(match:)` method also reports progress toward an achievement. Each time the player starts a match, they earn 10% toward this achievement until they reach 100%.

```
// Set the progress for the achievement.
achievement?.percentComplete = achievement!.percentComplete + 10.0

// Report the progress to Game Center.
GKAchievement.report(achievementsToReport, withCompletionHandler: {(error: Error?) i
    if let error {
        print("Error: \(error.localizedDescription).")
    }
})
```

For details about configuring achievements for a game, see Rewarding players with achievements.

Submit scores to the leaderboard

The `saveScore()` method saves the local player's score to the leaderboard when either the player wins or the opponent forfeits.

```
GKLeaderboard.submitScore(myScore, context: 0, player: GKLocalPlayer.local,
    leaderboardIDs: ["123"]) { error in
    if let error {
        print("Error: \(error.localizedDescription).")
    }
}
```

For details about configuring leaderboards, see Creating recurring leaderboards.

Display achievements and leaderboards

The player can see their achievements and top scores using the access point that appears in the upper-left corner, or by tapping the buttons below Game Center Data in the content view. For example, when the player taps Top Scores, the `RealTimeGame.topScore()` method shows the scores in the Game Center dashboard.

```
let viewController = GKGameCenterViewController(leaderboardID: "123", playerScope: (
                                                timeScope: GKLeaderboard.TimeScope.a
viewController.gameCenterDelegate = self
rootViewController?.present(viewController, animated: true) { }
```

For more information about displaying Game Center information, see Adding an access point to your game and Displaying the Game Center dashboard.

See Also

Real-time games

- Finding multiple players for a game
  Discover and invite other players to participate in a real-time game.
- Exchanging data between players in real-time games
  Send data between players in a real-time multiplayer game.
- Adding voice chat to multiplayer games
  Enable players to voice chat with all, or groups of, players in a multiplayer game.
- Matchmaking rules

Game Center applies different type of rules you create in a particular order to find the best matches.

`class` GKMatchRequest

An object that encapsulates the parameters to create a real-time or turn-based match.

`class` GKMatchmaker

An object that creates matches with other players without presenting an interface to the players.

`class` GKMatchmakerViewController

An interface that allows a player to invite other players to a real-time game and automatch to fill any empty slots.

`protocol` GKInviteEventListener

A protocol that handles invite events from Game Center.

`class` GKInvite

An invitation to join a match sent to the local player from another player.

`class` GKMatch

A peer-to-peer network between a group of players that sign into Game Center.