Game Kitプログラミングガイド

2011-03-08

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
U.S.A.

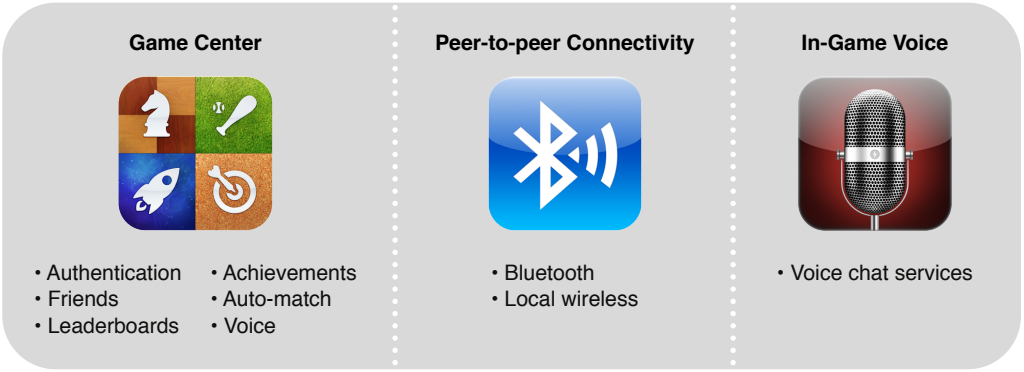
アップルジャパン株式会社
〒163-1450 東京都新宿区西新宿
3 丁目20 番2 号
東京オペラシティタワー
http://www.apple.com/jp/

目次

図、表、リスト

Game Kitについて

Game Kitフレームワークは、優れたソーシャルゲームの作成を支援するObjective Cクラスを提供します。ソーシャルゲームでは、プレーヤー同士で同じ体験を共有できます。プレーヤーが自分のお気に入りのゲームを友だちに紹介すると、そのゲームをダウンロードして楽しむプレーヤーが増えます。顧客からの好意的な口コミ情報は、ゲームの最大の無料広告になります。



概要

Game Kitフレームワークは、Game Center、ピアツーピア接続、ゲーム内ボイスといった3つの異なるテクノロジーを提供します。アプリケーションには、他のテクノロジーに関係なく各テクノロジーを採用できます。

Game Centerは一元管理されたネットワークサービスを提供

Game Centerは、プレー中のゲームについての情報をプレーヤー同士で共有したり、ほかのプレーヤーと一緒にマルチプレーヤー対戦に参加できるソーシャルゲーミングサービスです。Game Centerは、ワイヤレスネットワークと携帯電話ネットワークのどちらからでもサービスを提供できます。Game Centerの主な機能は、次の通りです。

- **認証** - プレーヤーはGame Centerでセキュアなアカウントを作成し、iOSベースのデバイスでGame Centerにアクセスできます。
- **友だち** - プレーヤーはGame Centerのほかのプレーヤーを友だちとして登録できます。友だちは、最近遊んだゲームなどプレーヤーの詳細情報を確認できます。
- **Leaderboard** - プレーヤーのスコアをGame Centerに記録したりGame Centerから取得したりできます。

- **アチーブメント（Achievement、成績）** - そのゲームでのプレーヤーのアチーブメントを管理します。アチーブメントは、Game Centerサービスに記録され、Game Centerアプリケーションとゲームの中で閲覧できます。
- **オートマッチ** - Game Centerを介して複数のプレーヤーとつながるネットワークゲームを作成できます。プレーヤーは友だちを招待したり、まだ会ったことのないプレーヤーと接続して対戦できます。プレーヤーは、ゲームを実行していないときでも対戦への招待を受信できます。その場合、ゲームが自動的に起動し、招待が処理されます。
- **ボイス** - 対戦に接続されたプレーヤー間で音声通信を行うことができます。

Game CenterはiOS 4.1以降でサポートされています。

関連する章（パートI）： 「Game Centerの概要」（15 ページ）、「Game Centerでのプレーヤーの使用」（23 ページ）、「Leaderboard」（33 ページ）、「アチーブメント（Achievements）」（41 ページ）、「マルチプレーヤー」（49 ページ）、「対戦を使用したネットワークゲームの実装」（61 ページ）、および「対戦へのボイスチャットの追加」（67 ページ）

ピアツーピア接続はローカルワイヤレスネットワークとBluetoothネットワークに対応

ピアツーピア接続を利用すると、複数のiOSベースデバイス間にアドホックなBluetoothネットワークまたはローカルワイヤレスネットワークを設定できます。このネットワークはゲームを想定したものですが、アプリケーションのユーザ同士での任意の種類のデータ交換に利用できます。たとえば、ピアツーピア接続機能を使用して、電子名刺などのデータを共有するといったことができます。

ピアツーピア接続機能はiOS 3.0以降で提供されています。

関連する章（パートII）： 「ピアツーピア接続」（73 ページ）、「Peer Pickerによるピアの検索」（81 ページ）、および「セッションの使用」（83 ページ）

ゲーム内ボイスは共通のボイスチャットインフラストラクチャを提供

ゲーム内ボイスを利用すると、2台のiOSベースデバイス間で音声通信が可能になります。ゲーム内ボイスは、マイクをサンプリングしたり、オーディオ再生のミキシングを処理します。ゲーム内ボイスを使用するには、まず、アプリケーションで2台のデバイス間にネットワーク接続を確立します。ゲーム内ボイスは、このネットワーク接続を使用して、独自のハンドシェーク情報を送信し、独自のネットワーク接続を確立します。

ゲーム内ボイスは、iOS 3.0以降で提供されています。

関連する章（パートIII）：「ゲーム内ボイス」（87ページ）と「ボイスチャットの追加」（91ページ）

この文書の使いかた

ゲームにGame Centerのサポートを追加する場合は、最初に「Game Centerの概要」（15ページ）をお読みください。次に、「Game Centerでのプレーヤーの使用」（23ページ）で、デバイス上でプレーヤーを認証する方法を参照してください。ゲームでプレーヤーの認証を行った後は、パートIの残りの章で、ゲームにLeaderboard、アチーブメント、マッチメークの機能を追加する方法をお読みください。

ローカルエリアのピアツーピアネットワークをアプリケーションに追加する場合は、「ピアツーピア接続」（73ページ）で、`GKSession`クラスと`GKPeerPickerController`クラスの概要を参照してください。その後、パートIIの残りの章で、セッションの機能をアプリケーションに追加する方法をお読みください。

アプリケーションにすでにネットワーク接続機能がある場合にボイスチャットを追加する場合は、「ゲーム内ボイス」（87ページ）を読み`GKVoiceChatService`クラスの概要を理解してから、「ボイスチャットの追加」（91ページ）を読んで、ボイスチャットの機能をアプリケーションに追加する方法を一通り学びます。

お読みになる前に

アプリケーションでどのGame Kitコンポーネントを採用するかを問わず、Cocoaプログラミング（特にデリゲートとメモリ管理）に精通していることが必要です。Cocoaの概要については、『*Cocoa Fundamentals Guide*』を参照してください。

Game Centerを使用するゲームを設計する場合は、ViewとView Controllerに慣れている必要があります。Game Kitは、Leaderboard、アチーブメント、およびマッチメーク用の標準的なユーザインターフェイスを表示するためのView Controllerを提供しています。『*View Programming Guide for iOS*』を参照してください。Game Centerクラスは、アプリケーションに結果を返すためにブロックオブジェクトを多用しています。そのためブロックおよび一般的なブロックプログラミング技法を理解しておく必要があります。ブロックプログラミングの概要については、『*A Short Practical Guide to Blocks*』を参照してください。

Game Kitを使用してネットワークゲームを実装する前に、一般的なネットワークプログラミングのデザインパターンを理解しておく必要があります。Game Kitは、下位レベルのネットワークインフラストラクチャを提供しますが、速度の遅いネットワーク、ネットワークの切断、セキュアでないネットワークを介したデータ送信に起因するセキュリティ上の問題などについては、アプリケーション側で対処しなければなりません。

関連項目

特定のGame Kitクラスの詳細については、『*Game Kit Framework Reference*』を参照してください。

iTunes Connectでのゲームの設定方法については、『iTunes Connect Developer Guide』を参照してください。

『*GKTapper*』のサンプルは、Game Centerアプリケーションでユーザ認証、Leaderboard、およびアチーブメントを実装する方法を示しています。

『*GKTank*』のサンプルは、ピアツーピア接続を使用してネットワークゲームを実装する方法を示しています。

Game Center

パートIでは、ゲームでGame Centerを使用する方法を説明します。

Game Centerの概要

この章では、Game Centerの機能と、Game Centerに対応したゲームを開発する際の基本事項について説明します。

Game Centerとは

Game Centerは、iOS 4.1以降が稼働するiOSベースのデバイスで利用可能な新しいソーシャルゲーミングネットワークです。ソーシャルゲーミングネットワークでは、プレー中のゲームについての情報をプレーヤー同士で共有したり、ほかのプレーヤーと一緒にマルチプレーヤーゲームに参加したりできます。Game Centerは、ワイヤレスネットワークと携帯電話ネットワークのどちらからでもサービスを提供できます。

Game Centerの機能は、相互に連携した次の3つのコンポーネントによって提供されます。

- Game Centerアプリケーション - iOS 4.1移行に組み込まれたアプリケーションで、プレーヤーはここからGame Centerのすべての機能にアクセスできます。
- Game Centerサービス — ゲームとGame Centerアプリケーションの両方から接続できるオンラインサービスです。このオンラインサービスは、各プレーヤーについてのデータを保存したり、異なるネットワーク上のデバイス間を中継するネットワークを提供します。
- Game Kitフレームワーク —Game Centerの機能をゲームに追加するために使用するクラスを提供します。

では、Game Centerにはどのような機能があるのでしょうか？

- **認証** - 各プレーヤーは、**ニックネーム**と呼ばれる、オンライン上で個人を識別するものを作成し、管理できます。プレーヤーはニックネームを使用して、Game Center上で自分のIDを認証したり、友だちのリストを管理したり、友だちに見せるステータスメッセージを登録します。

- **Leaderboard**— ゲームからGame Centerサービスにスコアを登録し、プレーヤーはそのスコアを後で閲覧できます。スコアはGame Centerアプリケーションを起動すると表示されます。また、ゲーム側でも、わずか数行のコードを追加するだけでLeaderboardを表示できます。Leaderboardは、Game Centerのプレーヤー同士で競争意識をいだかせるのに役立ちます。

  Leaderboardをゲームに追加する場合は、ゲームのスコアの解釈と表示の方法を決めます。たとえば、スコアを時間、お金、または任意の値（「ポイント」）として表示するようにカスタマイズできます。複数のカテゴリのLeaderboardを作成することもできます。たとえば、ゲームで使用されている難易度ごとにLeaderboardを表示するといったことが可能です。Leaderboardに保存したスコアを、ゲームアプリケーションから取得できます。

- **アチーブメント（Achievement）**— プレーヤーの腕前を測定できるもう1つの手段です。アチーブメントは、プレーヤーがゲーム内で達成できる具体的な目標です（たとえば、「10個の金貨を見つける」、「30秒以内に旗を獲得する」など）。ゲームでは、アチーブメントについて説明するテキスト、アチーブメントを達成することでプレーヤーが獲得できるポイント数、アチーブメントを表すアイコンをカスタマイズします。

  Leaderboardと同様に、プレーヤーは獲得したアチーブメントをGame Centerアプリケーション内で確認できます。また、友だちとアチーブメントを比較することもできます。ゲームでは、数行のコードを追加するだけでアチーブメントを表示できます。また、アチーブメント記述をダウンロードし、それを使用してカスタマイズしたアチーブメント画面を作成することもできます。

- **マルチプレーヤー**— オンラインのマルチプレーヤーゲームに関心があるプレーヤーを互いに発見して、1つの対戦につなぐことができます。ニーズに応じて、ゲームでは、Game Centerを使用してすべての参加者を一緒に接続するか、Game Centerからプレーヤーのリストを配布することができます。後者の場合、自分が用意したサーバに各デバイスを接続するゲーム内で独自のネットワーク実装を提供することになります。

  マッチメークでは、オートマッチと招待の両方が可能です。ユーザがオートマッチを選択している場合は、対戦に招待するプレーヤーの人数と、対戦に参加できるプレーヤーの資格を明記した対戦要求を作成します。すると、Game Centerはその基準を満たすプレーヤーを見つけて対戦に追加します。招待の場合は、ゲーム内またはGame Centerアプリケーション内から友だちを対戦に招待できます。招待された友だちは、ゲームを起動して対戦に参加できるプッシュ通知を受信します。プッシュ通知は、デバイス上にそのゲームがインストールされていなくてもプレーヤーに送信できます。その場合、プレーヤーはその通知から直接App Storeを起動できます。

- **ピアツーピアネットワーク**— ピアツーピアネットワークはマッチメークサービスによって提供され、対戦のほかの参加者にデータや音声情報を送信する簡単なインターフェイスとなります。ゲームでは、複数の音声チャンネルを作成して、各チャンネルに、対戦に参加しているプレーヤーの別々のサブセットを割り当てることができます。たとえば、複数のチームが参加するゲームであれば、チーム外のメンバーに聞かれずに会話できるように、チームごとに別々のチャンネルを作成できます。

Game Centerを使用することで、ゲームの視野が広がります。プレーヤーは、友だちがプレーしているゲームを見たり、ゲームをすぐに購入できる機会を提供して、友だちを対戦に招待できます。ゲームのプレーヤーを互いに接続することによって、ゲームについての情報をすばやく広めることができます。

Game Centerの重要な概念

このセクションでは、Game Center機能をゲームに追加する前に知っておくべき事項について説明します。

すべてのGame Centerゲームはプレーヤーの認証で始まる

マッチメーク、Leaderboard、およびアチーブメントは、いずれもデバイス上の認証済みプレーヤー（**ローカルプレーヤー**と呼ぶ）を使用します。たとえば、ゲームがスコアをLeaderboardに報告する場合は、認証済みプレーヤーのスコアが常に報告されます。ほとんどのGame Centerクラスは、認証済みプレーヤーが存在する場合にのみ動作します。ゲームは、Game Centerのどんな機能を使用する場合も、事前にローカルプレーヤーを認証しなければなりません。ローカルプレーヤーを認証するときは、プレーヤーがすでにデバイス上で認証済みかどうかをチェックします。認証済みのローカルプレーヤーが存在しない場合は、Game Kitが、プレーヤーに既存のアカウントでのログイン、または新規アカウントの作成を許可するユーザインターフェイスを表示します。

認証済みプレーヤーが存在しないときは、Game Centerのすべての機能を無効にしなければなりません。

重要： Game Centerが利用できないデバイス上で、ローカルプレーヤーを認証しようとすると、Game Kitは`GKErrorNotSupported`エラーを返します。Game Centerのすべての機能を無効にすることによって、認証エラーを処理しなければなりません。

ゲームはインターフェイスの管理にView Controllerをすでに使用していなければならない

Game Centerのすべてのビューは、ゲームのView Controllerを親として使用し、モーダルモードで表示されます。Leaderboard、アチーブメント、およびマッチメークには、いずれも関連情報をユーザに表示するためのView Controllerクラスが含まれています。たとえば、`GKLeaderboardViewController`クラスは、ゲームのLeaderboard情報をユーザに表示する標準ユーザインターフェイスを提供します。

View Controllerを使用してユーザインターフェイスを制御することは、それ以外の理由からも優れています。たとえば、View Controllerを使用して、デバイスの向きの変化をサポートできます。ゲームでのView Controllerの使用の詳細については、『*View Controller Programming Guide for iOS*』を参照してください。

Game Centerにアクセスするほとんどのクラスは非同期に動作する

わずかな例外はありますが、Game Kitのクラスは非同期にGame Centerサービスにアクセスします。これには、Game Centerにデータを送信したり、Game Centerからデータを要求したり、Game Centerにタスク（マッチメークなど）の実行を依頼する操作が含まれます。いずれの場合も、ゲームはGame Kitを呼び出すことによって要求を送信します。その後、操作が完了すると（またはエラーのために失敗すると）、Game Kitはゲームを呼び出して結果を返します。

Game Centerクラス用のコールバックは、ブロックオブジェクトを使用して実装されています。ブロックオブジェクトは、エラーパラメータを受け取ります。ゲームが要求した操作によっては、Game Kitがゲームに情報を返せるようにブロックがその他のパラメータを受け取ることもあります。Game Centerをゲームに追加する前に、ブロックオブジェクトに慣れておく必要があります。ブロックプログラミングの概要については、『*A Short Practical Guide to Blocks*』を参照してください。

ネットワークエラーの後に必ずデータを再送信する

Leaderboardやアチーブメント機能を実装する場合、ゲームはプレーヤーのスコアやアチーブメントの達成状況をGame Centerに報告します。ただし、ネットワークは完全に信頼できるとは限りません。ネットワークエラーが発生した場合、ゲームは、Game Kitオブジェクトを保存して、後でそのアクションを再試行しなければなりません。`GKScore`クラスと`GKAchievement`クラスは、両方とも`NSCoding`プロトコルをサポートしています。そのため、ゲームがバックグラウンドに移ったときにそれらのオブジェクトをアーカイブすることができます。

ネットワークエラーの後にデータの一部を受信できる

ゲームがGame Centerからデータを取得するとき（たとえば、Leaderboardからスコアデータをロードするとき）、ネットワークエラーが発生する可能性があります。ネットワークエラーが発生すると、Game Kitは接続エラーを返します。ただし、可能な場合は、Game KitがGame Centerから受信したデータをキャッシュしています。この場合は、Game Kitから、エラーと結果の一部の両方を受信できます。

ゲームでGame Center Awarenessを実装する手順

Game Center機能をゲームに追加する準備ができたら、次の手順に従います。

1. iTunes ConnectでGame Centerを有効にします。詳細については、「iTunes Connectでアプリケーションを設定する」（19 ページ）を参照してください。
2. ゲームのバンドル識別子を設定します。詳細については、「アプリケーションのバンドル識別子を作成する」（19 ページ）を参照してください。
3. Game Kitフレームワークにリンクします。
4. `GameKit/GameKit.h`ヘッダをインポートします。
5. ゲームでGame Centerを必須にするかどうかを決定します。
    - ゲームでGame Centerが必須の場合は、アプリケーションがデバイスに要求する機能のリストにGame Centerキーを追加します。詳細については、「ゲームでGame Centerが必須の場合」（19 ページ）を参照してください。
    - ゲームでGame Centerが必須でない場合、ゲームからGame Kitフレームワークにウィークリンクを確立しなければなりません。そして、ゲームの起動時に、Game Centerがサポートされているかどうかをテストします。詳細については、「ゲームでGame Centerをオプションで使用する」（19 ページ）を参照してください。

6. プレーヤーの認証は、アプリケーションが起動してユーザインターフェイスが表示された直後に行います。詳細については、「Game Centerでのプレーヤーの使用」（23 ページ）を参照してください。

iTunes Connectでアプリケーションを設定する

Game Centerコードをアプリケーションに追加する場合は、事前にiTunes Connectでゲームの設定を行い、Game Centerを使用できるようにする必要があります。また、アプリケーションのLeaderboardやアチーブメント情報の設定もここで行います。

Game Centerをサポートするようにゲームを設定する手順については、『iTunes Connect Developer Guide』を参照してください。

アプリケーションのバンドル識別子を作成する

バンドル識別子は、アプリケーションに対して作成する文字列です（`com.myCompany.myCoolGame`など）。バンドル識別子は、アプリケーションの開発プロセス全体を通して、アプリケーションを一意に識別するために使用されます。たとえば、XCodeを組み込んだアプリケーションをiTunes Connectで設定したデータに対応させるために、バンドル識別子が使われます。

Game Centerを使用しないアプリケーションの場合、アプリケーションを顧客に出荷する前にXcodeプロジェクトにバンドル識別子を提供する必要があるだけです。バンドル識別子を設定しなくても、アプリケーションの開発を始めることができます。ただし、Game Centerアプリケーションの開発に関しては、アプリケーションでGame Centerを実装する前にバンドル識別子を設定しなければなりません。Game Centerは、アプリケーションのバンドル識別子を使用して、Game Centerサービスからアプリケーションデータを取得します。アプリケーションのバンドル識別子をプロジェクトに設定するまではマッチメークを使用したりLeaderboardやアチーブメント情報を取得したりすることはできません。

バンドル識別子の設定の詳細については、『*Information Property List Key Reference*』を参照してください。

ゲームでGame Centerが必須の場合

ゲームが動作するためにGame Centerが必須の場合（たとえば、プレーヤー同士を対戦させるためにGame Centerが必要となるマルチプレーヤーゲーム）、Game Centerに対応したデバイスだけが、アプリケーションをダウンロードできるようにします。対応デバイスでのみアプリケーションが動作していることを確認するには、アプリケーションの`Info.plist`ファイル内の必要なデバイス機能のリストに`gamekit`キーを追加します。『*iOS Application Programming Guide*』の「iTunes Requirements」in *iOS Application Programming Guide*を参照してください。

ゲームでGame Centerをオプションでサポートする

Game Centerを使用するけれども、ゲームが正しく動作するためにGame Centerが必須ではない場合は、Game Kitフレームワークへのリンクをウィークリンクにして、実行時に存在するかどうかをテストできます。

リスト 1-1 （20 ページ）に、Game Center機能のテストに使用するコードを示します。この関数では、GKLocalPlayerクラスが存在するかどうかをテストします。このクラスはプレーヤーの認証を実行するために必要です。また、Game Centerを完全にサポートした最初のiOSバージョンであるiOS 4.1であるかどうかもテストします。

注： Game CenterはiOS 4.0で初めて試験的に導入されました。Game Centerクラスのいくつかはこのバージョンにも存在します。ただし、これらのクラスはiOS 4.1の正式リリースの前に変更されました。iOS 4.0が稼働するデバイス上では、Game Centerを使用しないでください。

リスト 1-1 Game Centerクラスが利用可能かどうかのテスト

```
BOOL isGameCenterAPIAvailable()
{
    // GKLocalPlayerクラスが存在するかどうかをチェックする
    BOOL localPlayerClassAvailable = (NSClassFromString(@"GKLocalPlayer")) !=
nil;

    // デバイスはiOS 4.1以降で動作していなければならない
    NSString *reqSysVer = @"4.1";
    NSString *currSysVer = [[UIDevice currentDevice] systemVersion];
    BOOL osVersionSupported = ([currSysVer compare:reqSysVer
options:NSNumericSearch] != NSOrderedAscending);

    return (localPlayerClassAvailable && osVersionSupported);
}
```

iOS 4.1は、Game Centerをサポートしていないデバイスにもインストールできます。それらのデバイスでは、リスト 1-1 （20 ページ）で定義されたisGameCenterAPIAvailable関数がYESを返します。デバイス上でGame Centerを使用できるかどうかを確認する次の手順は、ローカルプレーヤーの認証を試みることです。Game Centerをサポートするデバイス上では、ゲームがGKErrorNotSupportedエラーを受信します。詳細については、「ローカルプレーヤーの認証」 （26 ページ）を参照してください。

Game Centerアプリケーションのテスト

アプリケーションのテストを支援するために、Appleでは、Game Centerのサンドボックス環境を提供しています。このサンドボックス環境は、Game Centerの実際の機能を再現していますが、実際のサーバとは別になっています。サンドボックスを利用すると、アプリケーションを通常のユーザには見えないようにしてGame Centerの機能をテストできます。承認のためにアプリケーションを提出する前に、サンドボックスでGame Centerアプリケーションを徹底的にテストする必要があります。

デベロッパとして、サンドボックス用に別のGame Centerアカウントを作成する必要があります。常に、テスト用のサンドボックスと実際の環境のどちらにログインするのかを選択しなければなりません。Game Centerアプリケーションを起動して、現在認証済みのプレーヤーをログアウトします。その後、ゲームまたはGame Centerが有効な別のアプリケーションを実行します。アプリケーションの配布方法に応じて、各種の資格情報を入力します。アプリケーションが開発用に準備されている場合は、テストアカウント情報を入力します（サンドボックスにログインします）。それ以外の場合は、ライブアカウント情報を入力します（ライブ環境にログインします）。表 1-1に、どのビルドがどの環境で実行されるかを示します。

重要： Game Centerでアプリケーションを明確にテストするには、常に新しいテストアカウントを作成します。既存のApple IDを使用することはできません。

表 1-1 ビルドの対象ユーザと環境

アプリケーション	対象	Game Center環境
シミュレータのビルド	デベロッパ	サンドボックス環境
デベロッパのビルド	デベロッパ	サンドボックス環境
アドホック配布用ビルド	ベータテスター	サンドボックス環境
署名付き配布用ビルド	エンドユーザ	ライブ環境

サンドボックスでは、プレー中のゲームについての情報は共有できません。これによって、アプリケーションの存在がほかのプレーヤーに漏れるのを防止できます。

シミュレータでのゲームのテスト

Leaderboardとアチーブメントは、シミュレータ上でもデバイス上と同様に動作します。ただし、シミュレータで実行しているときは、マッチメークの招待を送受信することはできません。

Game Centerでのプレーヤーの使用

プレーヤーはGame Centerの基本です。プレーヤーはオンラインゲームに参加して、ゲームで高得点を上げたり、優れた結果を残します。しかし、ゲームでそういったことを行うプレーヤーを準備する前に、Game Centerがプレーヤーをどのように取り扱うかを理解する必要があります。

Game Centerを利用するプレーヤーは、まず、Game Center用のアカウントを作成する必要があります。Game Centerアカウントはいくつかの役割を果たします。それは、Game Centerによるプレーヤーの認証を可能にすること、プレーヤーを一意に識別すること、そしてプレーヤーに固有のデータをGame Centerに保存し、そのデータにアプリケーションからアクセスできるようにすることです。データには、以下の内容が含まれます。

- プレーヤーが獲得したLeaderboardの全スコア
- プレーヤーによるアチーブメントの達成状況
- プレーヤーの友だちリスト

この章では、アプリケーションでプレーヤーの情報を管理する方法について学習します。特に、以下について学習します。

- ゲームがGame Centerのさまざまなプレーヤーを識別する方法
- プレーヤーがGame Centerにログインするための適切な手順
- Game Centerからプレーヤーの詳細情報を取得する方法
- プレーヤーがGame Centerで友だちになるほかのプレーヤーを招待できるようにする方法

プレーヤー識別文字列を使用したプレーヤーの識別

Game Centerは、Game Centerアカウントごとに**プレーヤー識別子**と呼ばれる一意の文字列を割り当て、この識別子によって特定のアカウントを識別します。プレーヤーは自身のアカウントに関連したその他の情報を変更できますが、アカウントのプレーヤー識別子を変更することはできません。プレーヤー識別子は、Game Kitフレームワークのいたるところで特定のプレーヤーを識別する必要があるときに使用されます。たとえば、アプリケーションがGame Centerを使用してネットワーク対戦を作成する場合、対戦に参加している各デバイスは、そのデバイスにログインしているプレーヤーのプレーヤー識別子で識別されます。デバイスで稼働しているアプリケーションがほかの参加者にデータを送信するときは、プレーヤー識別子を使用してネットワークメッセージの宛先を指定します。

Game Centerとのやり取りでプレーヤー識別子を使用する以外に、特定のプレーヤーに情報を関連付ける必要がある場合にもプレーヤー識別子を使用しなければなりません。たとえば、ゲームが、プレーヤーのゲームの達成状況を管理するためにプレーヤーのデバイスにデータを保存する場合は、

プレーヤー識別子を使用して、同じデバイスでプレーする複数のプレーヤーを区別する必要があります。同様に、独自のネットワークサーバにプレーヤーの情報を保存する場合は、サーバでプレーヤー識別子を使用して、各プレーヤーのデータを一意に識別することができます。

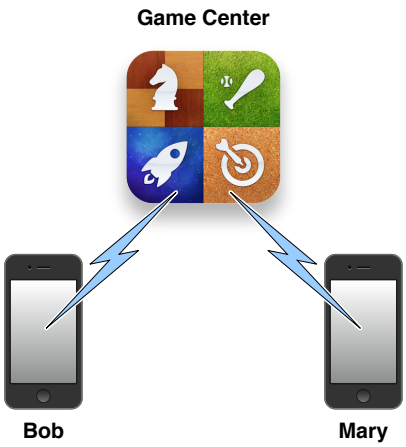
重要： プレーヤー識別文字列の形式や長さについて想定しないでください。個々のプレーヤー識別文字列は変更できませんが、新規のプレーヤー識別文字列の形式と長さは変更されることがあります。

プレーヤー識別子は、ゲームまたはGame Center内のみでの使用を目的としています。プレーヤー識別子はプレーヤーには表示されません。ただし、プレーヤーの名前を表示することは重要です。プレーヤーはすべて各自で名前（**ニックネーム**とも呼ばれる）を選択できます。プレーヤー識別子を表示する代わりに、Game Centerからは常にプレーヤーのニックネームを取得します。

ローカルプレーヤーはデバイスにサインインしたプレーヤーである

iOSベースのデバイスで稼働しているアプリケーションのインスタンスでは、1人のプレーヤーがほかのプレーヤーよりも優先されます。**ローカルプレーヤー**は、特定のデバイス上でのプレーが現在認証されているプレーヤーです。たとえば、図 2-1では、1つの対戦に2人のプレーヤーが接続されています。左側のデバイスでは、Bobがローカルプレーヤーでマリーがリモートプレーヤーです。右側のデバイスでは、Maryがローカルプレーヤーでボブがリモートプレーヤーです。

図 2-1 ローカルプレーヤーとリモートプレーヤー



どのデバイスでも、一度にGame Centerに接続できるプレーヤーは1人だけです。プレーヤーは、Game Centerアプリケーションを起動するか、Game Centerのサポートを実装したゲームを起動することによってGame Centerに接続します。いずれの場合も、アカウント名とパスワードを提示するためのインターフェイスと共に表示されます。デバイスでアカウントが認証された後は、他のアプリケーションに切り替えたりデバイスを再起動しても、プレーヤーはデバイスに接続されたままになります。プレーヤーは、Game Centerアプリケーションを起動して明示的にサインアウトした場合にのみ、Game Centerから切断されます。

重要： マルチタスクをサポートしているゲームでは、この動作に特に留意する必要があります。ゲームがバックグラウンドに移動すると、プレーヤーはGame Centerアプリケーションを起動してサインアウトすることがあります。また、制御がアプリケーションに返される前にサインインすることもあります。ゲームがフォアグラウンドに移動するたびに、認証済みのプレーヤーがいない場合はGame Center機能を無効にしなければならなかったり、新規のローカルプレーヤーのIDに基づいて内部状態を更新しなければならない場合があります。

Game CenterにアクセスするGame Kitのクラスのほとんどは、デバイスに認証済みのローカルプレーヤーがいて、デバイスがプレーヤーの代わりに機能すると考えています。たとえば、ゲームがLeaderboardにスコアを報告するときは、常にローカルプレーヤーが獲得したスコアだけを報告できます。認証済みのローカルプレーヤーがいないのにアプリケーションがLeaderboard（または、他の似たようなタスク）にスコアを報告しようとすると、Game Kitがゲームにエラーを返します。

プレーヤーオブジェクトによるプレーヤーの詳細情報の提供

ゲームがGame Center用アカウントのプレーヤー識別子を認識している場合は、Game Kitを使用してプレーヤーについての情報を取得できます。Game Kitは、`GKPlayer`オブジェクトを返すことによって、アプリケーションにプレーヤーの詳細情報を提供します。プレーヤーオブジェクトには、以下のプロパティが含まれます。

- `playerID`プロパティ - プレーヤー識別文字列を保持します。
- `alias`プロパティ - このプレーヤーによって指定され、プレーヤーに表示される文字列を保持します。
- `isFriend`プロパティ - プレーヤーが現在のローカルプレーヤーの友だちかどうかを示します。

プレーヤーの詳細情報の取得については、「Game Centerプレーヤーの詳細情報の取得」（28 ページ）を参照してください。

`GKLocalPlayer`クラスは、`GKPlayer`クラスの特別なサブクラスです。ローカルプレーヤーの追加プロパティやローカルプレーヤーを認証するためのメソッドが含まれています。`GKLocalPlayer`クラスは、以下の追加プロパティを提供します。

- `friends`プロパティ - ローカルプレーヤーの友だちとして指定されたGame Centerのほかのプレーヤーの識別文字列が読み込まれます。このプロパティは、アプリケーションがGame Centerから明確にそれらの識別子を要求した後にだけ読み込まれます。詳細については、「ローカルプレーヤーの友だちの識別子の取得」（29 ページ）を参照してください。
- `underage`プロパティ - このプレーヤーが未成年であることを表します。Game Centerの機能の中には、このプロパティの値が`YES`の場合に無効になるものがあります。このプロパティは、未成年のプレーヤーについて固有の機能を無効にするオプションを設定できるように提供されます。

ローカルプレーヤーの認証

Game Centerをサポートするすべてのゲームは、Game Centerの機能を使用する前にローカルプレーヤーを認証しなければなりません。ゲームを起動したら、できるだけ早くプレーヤーを認証する必要があります。ゲームがプレーヤーにユーザインタフェイスを表示したらすぐに認証を行うのが理想的です。プレーヤーを認証するときは、まず、Game Kitがデバイス上に認証済みのプレーヤーがいるかどうかをチェックします。認証済みのプレーヤーがいる場合は、Game Kitが少しの間プレーヤーに歓迎バナーを表示します。デバイスに認証済みのプレーヤーが存在しない場合、Game Kitは、プレーヤーに既存のアカウントでのサインイン、または新規のGame Centerアカウントの作成を許可する認証ダイアログを表示します。そのため、ゲームは、ローカルプレーヤーを認証すると、透過的にプレーヤーにGame Centerアカウントの作成も許可します。

プレーヤーがGame Centerを使用しないと決定した場合は、Game Kitも透過的に対応します。プレーヤーがGame Centerの認証ダイアログを何回か消去すると、Game Kitは認証ダイアログを表示せずに、そのまま制御をアプリケーションに返します。Game Kitが認証ダイアログを無効にした後、プレーヤーはGame Centerアプリケーション内でのみアカウントを作成します。

重要： Game Kitは、Game Centerをサポートするすべてのゲームで、Game Centerを使用しないものと見なします。つまり、プレーヤーが他のゲームでアカウントの作成に同意しなかった場合、どのゲームでもダイアログが表示されなくなる可能性があります。Game Kitはすべてのゲームでこの処理に対応するため、ゲームを起動するたびに起動時にプレーヤーの認証を試みる必要があります。独自のメカニズムを実装して、ゲームでGame Centerの認証を無効にすることは避けてください。

リスト 2-1に、ローカルプレーヤーを認証するメソッドの実装方法を示します。通常、このようなメソッドは、アプリケーションデリゲートに実装され、アプリケーションデリゲートの`application:didFinishLaunchingWithOptions:`メソッド内から呼び出されます。メソッドは、`GKLocalPlayer`クラスの共有インスタンスを取得し、そのオブジェクトの`authenticateWithCompletionHandler:`メソッドを呼び出します。ローカルプレーヤーの認証は非同期に発生します。そのため、プレーヤーの認証を開始した後、アプリケーションは起動を続けます。認証が完了したら、Game Kitはゲームが提供するブロックオブジェクトを呼び出します。

リスト 2-1　ローカルプレーヤーの認証

```
- (void) authenticateLocalPlayer
{
    GKLocalPlayer *localPlayer = [GKLocalPlayer localPlayer];
    [localPlayer authenticateWithCompletionHandler:^(NSError *error) {
         if (localPlayer.isAuthenticated)
         {
             // 認証済みプレーヤーの追加タスクを実行する
         }
     }];
}
```

アプリケーションは常に、ローカルプレーヤーオブジェクトの`isAuthenticated`プロパティをチェックして、Game Kitがローカルプレーヤーを認証できたかどうかを判断する必要があります。アプリケーションが受け取ったエラーに頼って、認証済みプレーヤーがデバイスで利用可能かどうかを判断しないでください。エラーがアプリケーションに返されても、Game Kitには、認証済みのプレーヤーをゲームに提供するための情報がキャッシュされている場合があります。また、アプリケーションは、認証が失敗したときにプレーヤーにエラーを表示する必要もありません。代わりに、Game Kitは最も一般的なエラーをプレーヤーにすでに表示しています。ほとんどの認証エラーは、主にアプリケーションのデバッグに役立ちます。

ゲームが受け取る以下の2つの一般的なエラーは、詳細に説明する必要があります。

- アプリケーションが`GKErrorGameUnrecognized`エラーを受け取る場合は、iTunes Connectでアプリケーションの Game Centerが有効になっていません。iTunes Connectアカウントにログインして、アプリケーションでGame Centerが有効になっていることを確認してください。また、Xcodeプロジェクトのバンドル識別子が、iTunes Connectのアプリケーションに割り当てたバンドル識別子と一致することも確認してください。

- アプリケーションが`GKErrorNotSupported`エラーを受け取る場合は、アプリケーションを稼働するデバイスがGame Centerをサポートしていません。このデバイスでプレーヤーを認証する試みを無効にする必要があります。

`isAuthenticated`が`YES`の場合は、認証が正常に終了しています。ゲームは、ローカルプレーヤーオブジェクトのその他のプロパティを読み取り、Game Centerにアクセスするその他のクラスを使用できます。たとえば、以下は、プレーヤーを正常に認証した後にほとんどのゲームが実行しなければならない一般的なタスクです。

- ローカルプレーヤーオブジェクトの`alias`プロパティを読み込み、ローカルプレーヤーのニックネームを取得します。プレーヤーを呼ぶときは、ゲーム全体でこのニックネームを使用します。個別にプレーヤーに名前を尋ねないでください。

- 対戦要求を受信するために、招待ハンドラを追加します。ゲームがマッチメーキングをサポートしている場合は、招待ハンドラをすぐに追加して、ローカルプレーヤーを認証する必要があります。ゲームは、未処理の招待を受け取るために特にiOSで起動されます。プレーヤーを認証した後すぐに招待ハンドラを追加すると、ゲームで適切に招待状が処理されます。詳細については、「ほかのプレーヤーからの招待の処理」（52 ページ）を参照してください。

- ローカルプレーヤーの以前のアチーブメント達成状況を取得します。詳細については、リスト 4-2（44 ページ）を参照してください。

- ローカルプレーヤーの友だちのプレーヤー識別子リストを取得します。プレーヤーの詳細情報をロードする最初の手順です。詳細については、「ローカルプレーヤーの友だちの識別子の取得」（29 ページ）を参照してください。

- ゲームが特定のプレーヤーのカスタム情報（プレーヤーによるゲームの達成状況を示す状態変数など）を保存している場合は、プレーヤーの達成状況を復元できるように、終了ハンドラがこのデータもロードすることがあります。

マルチタスクアプリケーションでのローカルプレーヤーの認証

iOS 4以降では、マルチタスクによって最前面のアプリケーションをバックグラウンドに移動し、別のアプリケーションをフォアグラウンドに移動することができます。マルチタスクを使用すると、いくつかのアプリケーションをバックグラウンドで引き続き処理しながら、アプリケーションをすばやく切り替えることができます。マルチタスクとGame Centerをサポートするゲームは、ローカルプレーヤーを認証するときに追加手順を実行する必要があります。ゲームがバックグラウンドにあるときは、認証済みのプレーヤーの状態が変わる場合があります。プレーヤーは、Game Centerアプリケーションを使用してサインアウトできます。ゲームがフォアグラウンドに戻る前にサインインすることもできます。

ゲームがマルチタスクをサポートしている場合は、「ローカルプレーヤーの認証」で説明されているように、Game Kitが終了ハンドラを呼び出しますが、同様に、後で使用するために終了ハンドラを維持します。アプリケーションをバックグラウンドからフォアグラウンドに移動するたびに、Game Kitは、再びローカルプレーヤーを自動的に認証し、終了ハンドラを呼び出して、認証済みプレーヤーの状態の更新情報を提供します。

重要： フォアグラウンドに移動した後のローカルプレーヤーの自動認証は、iOS 4.2以降でのみ実行できます。iOS 4.1には、アプリケーションをバックグラウンドに移動した後にローカルプレーヤーを認証する適切な方法はありません。

マルチタスクをサポートするゲームでローカルプレーヤーを認証するためのガイドラインをいくつか以下に示します。

- 他のマルチタスクアプリケーションと同様に、ゲームは、バックグラウンドに移動する前に状態をアーカイブする必要があります。この状態には、Game Kitが提供するクラスを使用して作成された、現在のローカルプレーヤーに関連するオブジェクト（スコアやアチーブメントオブジェクトなど）が含まれます。バックグラウンドに移動した後にメモリから削除される可能性があるため、ゲームは状態をアーカイブします。

- ゲームがバックグラウンドに移動するとすぐに、ローカルプレーヤーオブジェクトの`isAuthenticated`プロパティの値は無効になります。ゲームがフォアグラウンドに戻り、Game Kitがプレーヤーを認証し、認証ハンドラが呼び出されるまで無効のままです。ゲームは、終了ハンドラが呼び出されるまで認証済みのプレーヤーがいないかのように動作しなければなりません。ハンドラが呼び出されると、`isAuthenticated`プロパティが再び有効になり、プレーヤーがデバイスで認証されているかどうかを保持します。

- `isAuthenticated`プロパティの値が`NO`に変更されると、Game Centerコンテンツへのアクセス権を付与されたローカルプレーヤーがいなくなります。ゲームは、メモリ内の以前のローカルプレーヤーのGame Kitオブジェクトを破棄する必要があります。ただし、今後の開始時にアーカイブした状態をロードしたり、フォアグラウンドに移動するための準備を整えておく必要があります。プレーヤーが今後デバイスにログインする場合、ゲームはそのプレーヤーに関連するオブジェクトを展開し、それらを使用してプレーヤーにプレーの再開を許可することができます。

- `isAuthenticated`プロパティの値が`YES`の場合は、デバイスに認証されたローカルプレーヤーがいます。ただし、ゲームがバックグラウンドにあったときは、Game Centerにログインしていたプレーヤーが変更されることがあります。アプリケーションは、プレーヤーを認証したら、インスタンス変数にローカルプレーヤーのプレーヤー識別子を保存する必要があります。今後、終了ハンドラを呼び出すときは、ローカルプレーヤーオブジェクトの`playerID`プロパティの値と、アプリケーションが以前保存した識別子を比較します。識別子が変更されている場合は、プレーヤーが変更されています。ゲームは、以前のプレーヤーに関するオブジェクトを破棄してから、新規のローカルプレーヤーの適切な状態を作成またはロードする必要があります。

Game Centerプレーヤーの詳細情報の取得

多くのGame Kitクラスは、アプリケーションにプレーヤー識別子を返します。たとえば、Leaderboardから提供されたすべてのスコアでは、プレーヤー識別子を使用してスコアを獲得したプレーヤーが識別されます。ゲームにプレーヤー識別子のリストがある場合、`GKPlayer`クラスで`loadPlayersForIdentifiers:withCompletionHandler:`クラスメソッドを呼び出すことによって、

それらのプレーヤーの詳細情報をロードすることができます。リスト 2-2に、一般的な実装方法を示します。このGame Kitメソッドは、2つのパラメータを取ります。1つ目はプレーヤー識別子の配列です。2つ目は、Game KitがGame Kitからデータを取得した後に呼び出される終了ハンドラです。Game Kitは、バックグラウンドでデータを非同期にロードし、タスクが完了したら終了ハンドラを呼び出します。終了ハンドラは、エラーパラメータと同様に、`GKPlayer`オブジェクトの配列を（識別子ごとに1つ）受け取ります。

リスト 2-2　プレーヤーオブジェクトのコレクションの取得

```
- (void) loadPlayerData: (NSArray *) identifiers
{
    [GKPlayer loadPlayersForIdentifiers:identifiers
withCompletionHandler:^(NSArray *players, NSError *error) {
        if (error != nil)
        {
            // エラーを処理する
        }
        if (players != nil)
        {
            // GKPlayerオブジェクトの配列を処理する
        }
     }];
}
```

Game Kitがプレーヤー全員の情報をロードできなかった場合は、終了ハンドラにエラーを提供します。そのとき、`players`パラメータは、Game Kitが情報を取得できなかったプレーヤーの配列の一部を提供する場合があります。この理由から、リスト 2-2では、プレーヤーオブジェクトの配列の処理とは別に、エラー条件をテストしています。

友だちの使用

Game Centerは、ソーシャル性を持つことを目的としています。Game Centerを使用すると、プレーヤーは友だちになるほかのプレーヤーを招待できます。2人のプレーヤーが友だちの場合は、Game Centerアプリケーションにお互いの状況を表示したり、スコアを比較したり、より簡単にお互いを対戦に招待することができます。アプリケーションは、Game Kitを通じて、ローカルプレーヤーの友だちの情報にアクセスしたり、友達になるプレーヤーの招待をプレーヤーに許可したりすることもできます。たとえば、この機能を使用すると、ゲーム内でプレーした対戦で知り合ったばかりのプレーヤーに招待状を送ることができます。

ローカルプレーヤーの友だちの取得

ローカルプレーヤーの友だちの詳細情報の取得は、2段階で行われます。まず、ゲームは、ローカルプレーヤーオブジェクトの`friends`プロパティをロードします。その後、ほかのプレーヤー識別子と同様に、Game Centerからそれらのプレーヤーの詳細情報をロードします。

リスト 2-3に、ローカルプレーヤーの友だちのプレーヤー識別子リストをロードする方法を示します。その後、リスト 2-2 （29 ページ）で定義されたメソッドを呼び出し、それらのプレーヤーの詳細情報を取得します。

リスト 2-3　ローカルプレーヤーの友だちの取得

```
- (void) retrieveFriends
{
   GKLocalPlayer *lp = [GKLocalPlayer localPlayer];
   if (lp.authenticated)
   {
      [lp loadFriendsWithCompletionHandler:^(NSArray *friends, NSError *error)
 {
         if (friends != nil)
         {
            [self loadPlayerData: friends];
         }
      }];
   }
}
```

ほかのプレーヤーへの招待状の送信

Game Centerアプリケーションを使用すると、プレーヤーはほかのプレーヤーに招待状を送信できます。iOS 4.2以降では、ゲームは`GKFriendRequestComposeViewController`クラスを使用して、友達リクエストの送信をプレーヤーに許可します。友達リクエストは、ゲームが作成したView Controllerによってモーダルモードで表示されなければなりません。

リスト 2-4に、View Controllerによってプレーヤーがほかのプレーヤーに要求を送信できるようになる仕組みを示します。このメソッドでは、プレーヤー識別子の配列がパラメータとして渡されます。このメソッドでは、`GKFriendRequestComposeViewController`オブジェクトをインスタンス化し、デリゲートを設定し、招待状を受け取るプレーヤーのリストを追加します。その後、View Controllerが友達リクエストを表示して、返します。

リスト 2-4　友達リクエストの表示

```
- (void) inviteFriends: (NSArray*) identifiers
{
    GKFriendRequestComposeViewController *friendRequestViewController =
[[GKFriendRequestComposeViewController alloc] init];
    friendRequestViewController.composeViewDelegate = self;
    if (identifiers)
    {
        [friendRequestViewController addRecipientsWithPlayerIDs: identifiers];
    }
    [self presentModalViewController: friendRequestViewController animated:
YES];
    [friendRequestViewController release];
}
```

プレーヤーが友達リクエストを消去すると、デリゲートの`friendRequestComposeViewControllerDidFinish:`メソッドが呼び出されます。リスト 2-5 （30 ページ）に、View Controllerが要求を消去する方法を示します。

リスト 2-5　プレーヤーが友達リクエストを消去したときの応答

```
-
(void)friendRequestComposeViewControllerDidFinish:(GKFriendRequestComposeViewController
 *)viewController
```

```
{
    [self dismissModalViewControllerAnimated:YES];
}
```

Leaderboard

多くのゲームでは、プレーヤーがゲームのルールをどのくらいマスターしたかを測定できるように、スコアリングシステムを提供しています。プレーヤーの腕前が向上するにつれてスコアも高くなります。Game Centerでは、ゲームをするすべての人が獲得したスコアを記録するためにLeaderboardが使われます。ゲームは、プレーヤーのスコアをGame Centerに送信します。プレーヤーが自分のスコアを見たいときは、Game Centerアプリケーションで Leaderboard画面を表示するか、ゲームで Leaderboardを表示します。スコアをフィルタリングすることもできます。たとえば、Leaderboard画面をカスタマイズして、友だちが獲得したスコアを表示することもできます。

Leaderboardをサポートする際のチェックリスト

Leaderboardをゲームに追加するには、以下の処理を行う必要があります。

- Leaderboardを追加する前に、ローカルプレーヤーを認証するためのコードをゲームに追加する必要があります。詳細については、「Game Centerでのプレーヤーの使用」（23 ページ）を参照してください。
- ゲームでのスコアの計算方法を決定します。ゲームに適した採点法を自由に設計できます。
- iTunes Connectでアプリケーション用にLeaderboardを1つ以上設定します。このステップでは、アプリケーションがGame Centerに送信したスコアをLeaderboardに表示するときの書式を決定します。Leaderboardは、各種の言語と地域に対応するようにローカライズできます。詳細は「iTunes ConnectでのLeaderboardの設定」（34 ページ）を参照してください。
- Game Centerにスコアを報告するためのコードを追加します。詳細については、「Game Centerへのスコアの送信」（35 ページ）を参照してください。
- 報告についての問題を処理するコードを追加します。Game Centerにスコアを報告できない場合、アプリケーションはスコアオブジェクトをアーカイブし、エラーが解消したら再送信を試みる必要があります。詳細については、「スコア報告エラーからの復旧」（36 ページ）を参照してください。
- プレーヤーにアプリケーション内でのLeaderboardの表示を許可するコードを追加します。Game Centerアプリケーションに表示されるLeaderboardに似たLeaderboardを表示するには、「標準Leaderboardの表示」（36 ページ）を参照してください。必要であれば、Game Centerからスコアデータを取得し、カスタマイズして独自のユーザインターフェイスを作成することもできます。スコアデータの取得については、「Leaderboardのスコアの取得」（37 ページ）で説明しています。

iTunes ConnectでのLeaderboardの設定

Game Centerにとっては、スコアはアプリケーションから報告された単なる64ビットの整数値です。スコアの意味と、アプリケーションでのスコアの計算方法は自由に決定できます。Leaderboardをアプリケーションに追加する準備ができたら、スコアの書式とプレーヤーへの表示方法をGame Centerに指定するために、iTunes Connect上でLeaderboardを設定します。また、さまざまな言語でスコアを正しく表示できるように、ローカライズした文字列を指定します。iTunes ConnectでLeaderboardを設定することの大きな利点は、コードをまったく記述せずに、Game Centerアプリケーションにゲームのスコアを表示できる点です。

iTunes connectには、各ゲームに独自のLeaderboard設定があります。つまり、Leaderboardの設定とスコアデータは、ゲーム間で共有されません。

Leaderboardを設定するには、以下について決定する必要があります。

- スコアの測定単位
- スコアの並べ順（昇順か降順か）
- スコアの書式
- アプリケーションから報告されたスコアを1つのLeaderboardにまとめるか、ゲームのプレーモードごとにLeaderboardを分けるか

以下に、Leaderboardを作成するにあたって決める必要のある項目について概要を説明します。iTunes ConnectでのLeaderboardの設定の詳細については、『iTunes Connect Developer Guide』を参照してください。

スコアの書式の定義

スコアの計算方法はアプリケーションごとに自由ですが、スコアを書式化してユーザに表示できる十分な情報を提供する必要があります。Game Centerでは、スコアは、抽象的な測定値、時間、またはお金の3つうち、いずれかの方法で測定されます。書式の種類を決めて、その単位に対するローカライズした文字列を指定します。たとえば、書式の種類として`Integer`を指定し、英語にローカライズした文字列として「point」（単数用）と「points」（複数用）を指定するなどです。こうすると、10のスコアを報告した場合に「10 points」と書式化されます。別のゲームで同じ書式を使用することもできます。また、ローカライズした文字列として異なる文字列を指定することもできます（「laps」や「cars」など）。

注： デフォルトでは、単位の文字列の前に空白文字は入りません。スコアの値の後に空白を入れる場合は、単位の文字列を指定するときに追加する必要があります。

スコアの書式を指定するときに、スコアを昇順に並べるか降順に並べるかを選択できます。たとえば、コースの完走時間をスコアとして記録するレースゲームでは、スコアを昇順に並べて最も低いスコア（最も速いタイム）を最高スコアにします。獲得した収入をスコアとするゲームでは、降順を選択します。最高スコアは、常にリストの最上部に表示されます。

Leaderboardのカテゴリ

ゲームには、しばしば複数のゲームプレー設定があります。たとえば、複数の難易度（易しい、中程度、難しい）があったり、ゲームのプレーや採点にさまざまなルールが指定されていたり、ゲームの中にもさまざまな位置情報（地図、レベル）があります。そのため、Game Centerでゲームを構成するときには、Leaderboardを1つだけ使用するのか、ルール設定ごとにLeaderboardを分けるのかを検討します。

iTunes Connectで設定するLeaderboardはそれぞれ**カテゴリ**で識別されます。カテゴリは、Leaderboardを一意に識別する文字列です。Leaderboardのローカライズしたタイトル、スコアの値、スコアの書式など、Leaderboardの他の設定を行うときに、Leaderboard独自のカテゴリ識別子を作成します。ゲームにLeaderboardの採点方法を実装するときは、カテゴリ文字列を使用して適切なLeaderboardにスコアを送信するようにします。たとえば、ゲームに3つの難易度がある場合は、「mygame.easy」、「mygame.medium」、「mygame.hard」という3つのカテゴリ文字列を使用して、難易度ごとに1つずつLeaderboardを作成します。

ゲームごとに最大25個までカテゴリを作成できます。1つのカテゴリは常に**デフォルト**カテゴリとして定義されます。カテゴリを指定せずにゲームスコアを報告すると、そのスコアはデフォルトカテゴリに関連するスコアリストに追加されます。

総合的なLeaderboard

総合的なLeaderboardは、他のLeaderboardに記録されたスコアを結合する特殊なLeaderboardです。プレーヤーが総合的なLeaderboardを表示すると、総合的なLeaderboardは、他のLeaderboardに記録されたスコアを取得し、1つのスコアの集合体としてそれらを並べ替えます。たとえば、ゲームに複数の地図がある場合、地図ごとに個別のLeaderboardを作成したり、すべての地図から取得したスコアを表示するLeaderboardを作成することができます。

ゲームに総合的なLeaderboardを複数設定するかどうかを決定する場合は、次のガイドラインに留意してください。

- 総合的なLeaderboardを作成するために結合するLeaderboardは、スコアの書式とソート順が同じでなければなりません。
- Leaderboardは、1つの総合的なLeaderboardの一部でしかないことがあります。
- 総合的なLeaderboardは、別の総合的なLeaderboardに結合できません。

Game Centerへのスコアの報告

アプリケーションは、`GKScore`オブジェクトを作成することによって、Game Centerにスコアを送信します。スコアオブジェクトは、スコアを獲得したプレーヤー、スコアを獲得した日時、スコアの報告先となるLeaderboardのカテゴリ、獲得したスコアに対応するプロパティを保持しています。アプリケーションは、スコアオブジェクトを設定したら、`reportScoreWithCompletionHandler:`メソッドを呼び出して、そのデータをGame Centerに送信します。

リスト 3-1に、スコアオブジェクトを使用して、Game Centerにスコアを報告する方法を示します。

リスト 3-1　Game Centerへのスコアの報告

```
- (void) reportScore: (int64_t) score forCategory: (NSString*) category
{
    GKScore *scoreReporter = [[[GKScore alloc] initWithCategory:category]
autorelease];
    scoreReporter.value = score;

    [scoreReporter reportScoreWithCompletionHandler:^(NSError *error) {
       if (error != nil)
       {
            // 報告エラーの処理
        }
    }];
}
```

スコアオブジェクトは、報告先となるLeaderboardのカテゴリ識別子で初期化されます。次に、このメソッドでは、valueプロパティをプレーヤーが獲得したスコアに設定しています。このカテゴリ識別子は、iTunes ConnectでLeaderboardを作成したときに定義したカテゴリ識別子の1つでなければなりません。スコアを報告するためのコードでは、スコアを獲得したプレーヤーや、スコアを獲得した日時は設定しません。これらのプロパティは、スコアオブジェクトを作成したときに自動的に設定されます。常にローカルプレーヤーのスコアが報告されます。

スコア報告エラーからの復旧

アプリケーションがネットワークエラーを受け取った場合は、スコアを破棄する必要があります。代わりに、スコアオブジェクトを保存して、後からプレーヤーのスコア報告を試みます。`GKScore`オブジェクトは`NSCoding`プロトコルをサポートしています。したがって、必要であれば、アプリケーションの終了時にオブジェクトをアーカイブしたり、アプリケーションの起動時に復元することができます。

標準Leaderboardの表示

Game Centerにスコアを送信するだけでなく、プレーヤーがアプリケーション内からスコアを閲覧できるようにする必要もあります。そのための最も簡単な方法は、`GKLeaderboardViewController`オブジェクトを利用することです。Leaderboard View Controllerは、Game Centerアプリケーションに表示されるLeaderboardと同様のユーザインターフェイスを提供します。Leaderboard View Controllerは、アプリケーション内に実装されているView Controllerによってモーダルモードで表示されます。

リスト 3-2に、View Controllerを使用して、デフォルトのLeaderboardを表示するメソッドを示します。このメソッドは、新規のLeaderboard View Controllerをインスタンス化して、その`leaderboardDelegate`プロパティをアプリケーションのView Controllerに設定しています。次に、このView ControllerはLeaderboardを表示して、デリゲートが呼び出されるのを待ちます。

リスト 3-2　デフォルトのLeaderboardの表示

```
- (void) showLeaderboard
{
    GKLeaderboardViewController *leaderboardController =
[[GKLeaderboardViewController alloc] init];
```

```
    if (leaderboardController != nil)
    {
        leaderboardController.leaderboardDelegate = self;
        [self presentModalViewController: leaderboardController animated: YES];
    }
}
```

Leaderboardを表示する前に、Leaderboard View Controllerの`category`プロパティと`timeScope`プロパティを設定できます。

- `category`プロパティを使用すると、Leaderboardを開いたときに表示するカテゴリ画面を設定できます。このプロパティを設定しないと、Leaderboardは、iTunes Connectで設定したデフォルトのカテゴリで開きます。
- `timeScope`プロパティを使用すると、スコアをユーザに表示する対象期間を設定できます。たとえば、`GKLeaderboardTimeScopeAllTime`は、スコアが記録された時期に関係なく、最高スコアを取得します。デフォルト値は、直近の24時間以内に獲得したスコアを表示する`GKLeaderboardTimeScopeToday`です。

ユーザがLeaderboardを消去すると、デリゲートの`leaderboardViewControllerDidFinish:`メソッドが呼び出されます。リスト 3-3 （37 ページ）に、View ControllerがLeaderboardを消去する方法を示します。

リスト 3-3　プレーヤーがLeaderboardを消去したときの応答

```
- (void)leaderboardViewControllerDidFinish:(GKLeaderboardViewController
*)viewController
{
    [self dismissModalViewControllerAnimated:YES];
}
```

Leaderboard View Controllerを消去する前に、`category`プロパティと`timeScope`プロパティを保存しておくこともできます。これらのプロパティには、Leaderboardの表示中にプレーヤーが選択した最後の選択内容が含まれています。したがって、次回プレーヤーがLeaderboardを表示したときに、これらと同じ値を使用してLeaderboard View Controllerを初期化することができます。

Leaderboardのスコアの取得

アプリケーションでスコアデータを調べたり、独自のLeaderboardビューを作成する場合は、Game Centerから直接スコアデータをロードできます。スコアデータを取得するには、`GKLeaderboard`クラスを使用します。`GKLeaderboard`オブジェクトは、Game Centerに保存されているアプリケーションのデータへのクエリを表します。スコアデータをロードするには、アプリケーションは`GKLeaderboard`オブジェクトを作成して、特定のスコアセットをフィルタリングするようにそのプロパティを設定します。アプリケーションは、Leaderboardオブジェクトの`loadScoresWithCompletionHandler:`メソッドを呼び出して、スコアをロードします。Game Centerからデータがロードされると、Game Kitは、指定されたブロックを呼び出します。

リスト 3-4に、典型的なLeaderboardデータのクエリを示します。このクエリのメソッドは、新規のLeaderboardオブジェクトを初期化して、スコアの報告時期に関係なく、すべてのプレーヤーを対象として上位10位までのスコアを取得するために、`playerScope`、`timeScope`、および`range`の各プロパティを設定します。

リスト 3-4　上位10位までのスコアの取得

```
- (void) retrieveTopTenScores
{
    GKLeaderboard *leaderboardRequest = [[GKLeaderboard alloc] init];
    if (leaderboardRequest != nil)
    {
        leaderboardRequest.playerScope = GKLeaderboardPlayerScopeGlobal;
        leaderboardRequest.timeScope = GKLeaderboardTimeScopeAllTime;
        leaderboardRequest.range = NSMakeRange(1,10);
       [leaderboardRequest loadScoresWithCompletionHandler: ^(NSArray *scores,
 NSError *error) {
            if (error != nil)
            {
                // エラー処理
            }
            if (scores != nil)
            {
                // スコア情報の処理
            }
            }];
    }
}
```

スコアを知りたいプレーヤーの識別子を明示的に渡すLeaderboard要求を作成することもできます。プレーヤーのリストを渡した場合は、`playerScope`プロパティは無視されます。リスト3-5に、ある対戦に関連付けられているリストのプレーヤー識別子を使用して、プレーヤーの最高スコアをロードする方法を示します。

リスト 3-5　対戦に参加しているプレーヤーの最高スコアの取得

```
- (void) receiveMatchBestScores: (GKMatch*) match
{
    GKLeaderboard *query = [[GKLeaderboard alloc] initWithPlayerIDs:
match.playerIDs];
    if (query != nil)
    {
        [query loadScoresWithCompletionHandler: ^(NSArray *scores, NSError
*error) {
            if (error != nil)
            {
                // エラー処理
            }
            if (scores != nil)
            {
                // スコア情報の処理
            }
        }];
    }
}
```

どちらの場合も、返された`GKScore`オブジェクトは、アプリケーションがカスタムビューを作成するために必要な情報を提供します。アプリケーションは、スコアオブジェクトの`playerID`を使用して、そのプレーヤーのニックネームをロードできます。詳細については、「Game Centerプレーヤーの詳細情報の取得」（28 ページ）を参照してください。`value`プロパティは、Game Centerに報告された実際の値を保持しています。さらに重要なことは、`formattedValue`プロパティが、iTunes Connectで提供したパラメータに応じて書式化された実際の値を含む文字列を提供する点です。

重要： `formattedValue`プロパティを使用する代わりに、独自の書式化コードを記述したいと思うかもしれませんが、それはしないでください。組み込まれている機能を使用したほうが、スコアの値を他の言語に容易にローカライズできます。また、Game Centerアプリケーションのスコア表示との一貫性も持たせられます。

最適なユーザ体験を維持するには、使用または表示する必要があるデータだけをLeaderboardに照会するべきです。たとえば、Leaderboardに保存されているすべてのスコアを一度に取得しようとしてはいけません。代わりに、Leaderboardの一部を取得して、必要に応じてビューを更新します。

カテゴリタイトルの取得

アプリケーションで独自のLeaderboard画面を表示する場合、アプリケーションでは、iTunes Connectで設定したカテゴリのローカライズされたタイトルが必要です。リスト 3-6 （39 ページ）に、アプリケーションがGame Centerからタイトルをロードする方法を示します。Game Centerにアクセスするその他のほとんどのクラスと同様に、このコードは、提供したブロックオブジェクトを呼び出してアプリケーションに結果を返します。

リスト 3-6　カテゴリタイトルの取得

```
- (void) loadCategoryTitles
{
    [GKLeaderboard loadCategoriesWithCompletionHandler:^(NSArray *categories,
NSArray *titles, NSError *error) {
        if (error != nil)
        {
            // エラー処理
        }
        // カテゴリとタイトル情報を使用する
     }];
}
```

リスト 3-6 （39 ページ）にある2つの配列で返されたデータは、カテゴリ識別子とそれに対応するタイトルです。

アチーブメント

アチーブメント（Achievement、成績）を使って、プレーヤーの目標を作成できます。目標を明示して、プレーヤーがその目標を達成したときに視覚的に認識できるようにすることで、プレーヤーのやる気を起こさせ、友だちと共有できるものを提供します。Game Centerを使用すると、プレーヤーはGame Centerアプリケーションで自分の達成状況を表示できます。また、Game Kitを使用すると、アプリケーション内に同じ達成情報を提供できます。

アチーブメント機能をサポートする際のチェックリスト

アチーブメント機能をアプリケーションに追加するには、以下の手順を行う必要があります。

- まだ実行していない場合は、ローカルプレーヤーを認証するためのコードをアプリケーションに追加します。詳細については、「Game Centerでのプレーヤーの使用」（23 ページ）を参照してください。
- iTunes Connectでゲーム用のアチーブメントを設定します。アチーブメントをプレーヤーに表示するために必要なすべてのデータを用意します。詳細については、「ゲーム用のアチーブメントの設計」（42 ページ）を参照してください。
- ローカルプレーヤーの達成状況をGame Centerに報告するためのコードをゲームに追加します。達成状況をGame Centerに保存することで、Game Centerアプリケーションで達成状況を表示することもできます。詳細については、「アチーブメントの達成状況のGame Centerへの報告」（43 ページ）を参照してください。
- ローカルプレーヤーの以前の達成状況をGame Centerからロードするコードを追加します。ローカルプレーヤーのアチーブメントの達成状況は、プレーヤーが正常に認証されるとすぐにロードされなければなりません。詳細については、「アチーブメントの達成状況のロード」（44 ページ）を参照してください。
- プレーヤーにアプリケーション内でのアチーブメントの表示を許可するコードを追加します。アプリケーションに標準的なアチーブメント画面を表示する方法については、「Achievement View Controllerを使用したアチーブメントの表示」（47 ページ）を参照してください。または、アプリケーションのユーザインターフェイスに合うようにアチーブメント情報の表示方法をカスタマイズする場合は、「カスタムのアチーブメントユーザインターフェイスの作成」（47 ページ）に説明されている、Game Centerからアチーブメント記述をロードする方法を参照してください。

ゲーム用のアチーブメントの設計

新規のアチーブメントを追加するには、まずゲーム内でプレーヤーが目標を達成する方法を決定する必要があります。目標とその達成方法は、ゲームによって異なります。ここでは、アチーブメントを設計する際のガイドラインを2つ示します。

- ゲームのさまざまなセクションごとにアチーブメントを作成します。たとえば、ゲームに異なるプレーモードがある場合は、必ずプレーモードごとにアチーブメントを追加します。これによって、プレーヤーはゲームの個々の部分を試してみようという気になります。

- プレーヤーのさまざまなスキルレベルに対応してアチーブメントを作成します。初心者でもアチーブメントを達成できるようにするべきです。また、スキルのあるプレーヤーが努力しないと達成できないアチーブメントも用意するべきです。さまざまなレベルのアチーブメントによって、プレーヤーは腕前を上げようという気になります。さまざまな目標を達成すると、ゲームがアチーブメントを承認します。

アチーブメントを作成する準備ができたら、記述を作成することによってiTunes Connectでアチーブメントを定義する必要があります。

アチーブメント記述には、以下の情報が含まれます。

- アチーブメントを一意に特定する識別文字列（**identifier**）。この識別文字列を選択すると、アプリケーションは、アチーブメントの達成状況を報告するときにこの文字列を使用します。識別子はプレーヤーには決して表示されません。

- アチーブメントを表すタイトル文字列（**title**）。

- アチーブメントに対する2つの説明文字列（**description**）。この記述で、アチーブメントをプレーヤーに明確に説明しなければなりません。最初の記述は、ユーザがアチーブメントを達成していないときに使用されます。ここでは、報酬を得るためにプレーヤーは何をしなければならないかを明確に説明しなければなりません。2番目の記述は、ユーザがアチーブメントを達成した後に使用されます。ここでは、何を達成したかを明確に説明しなければなりません。

- このアチーブメントを達成することで獲得できるポイント数を表す整数値（**points**）。達成すべきアチーブメントが難しくなればなるれほど、またはアチーブメントを達成するのに時間がかかればかかるほど、より多くのポイントを獲得できるようにするべきです。各ゲームのポイントは合計1000ポイント分までで、個々のアチーブメントはこれを分配した得点になります。1つのアチーブメントが100ポイントを超えることはできません。In App Purchaseを通じた追加コンテンツをゲームでサポートする場合は、In App Purchaseのコンテンツに適用するアチーブメントのために、ポイントをいくらか確保しておくこともできます。

- プレーヤーがアチーブメントを達成したときに表示される**画像**。アチーブメントが達成された場合の画像のみ作成します。アチーブメントが達成されていない場合は、Game Centerが提供する標準の画像が常に表示されます。

- ゲームを初めて起動したときや、プレー中にアチーブメントが発見された場合に、アチーブメントをユーザに表示するかどうかを決定します。ほとんどのアチーブメントは、即座にプレーヤーに表示されるべきです。ただし、サプライズを意図したアチーブメントや、プレーヤーが追加コンテンツを購入したときにのみ入手可能なアチーブメントは、伏せておくこともできます。

  アプリケーションは、アチーブメントの達成状況を報告することによって、伏せておいたアチーブメントを公表します。

アチーブメントに関するデータはすべて、iTunes Connectで編集します。また、サポートする言語に応じて、説明とタイトルをローカライズします。

ゲームごとに固有のアチーブメント記述があります。アチーブメント記述は複数のゲーム間で共有することはできません。

アプリケーションのアチーブメント記述を設定する方法については、『iTunes Connect Developer Guide』を参照してください。

アチーブメントの達成状況のGame Centerへの報告

プレーヤーのアチーブメントに進展があると、ゲームはその達成状況をGame Centerに報告します。達成情報をGame Centerに保存することで、Game Centerアプリケーションでプレーヤーのアチーブメントを表示できます。ローカルプレーヤーのアチーブメントの達成状況が常に報告されます。

アプリケーションは、浮動小数点のパーセンテージ（`0.0`から`100.0`までの値で、プレーヤーが達成したアチーブメントの割合を表す）を設定することによって、プレーヤーの達成状況を報告します。そのパーセンテージの計算方法と、それを変更するタイミングはデベロッパに任されています。たとえば、目標アチーブメントが「10個の金貨を見つけること」である場合、プレーヤーが1個金貨を見つけるたびに、このパーセンテージは10%ずつ増加します。一方、ゲーム内で、ある場所を発見するだけの目標アチーブメントを達成した場合は、初めて達成状況を報告するときに、単に100%達成されたことを報告します。段階的な達成状況をサポートする場合は、その値が変化するたびに達成状況をGame Centerに報告しなければなりません。

達成状況をGame Centerに報告するときは、次の2つのことが生じます。

- アチーブメントが今まで伏せられていた場合は、それがプレーヤーに公表されます。プレーヤーがアチーブメントをまったく達成できなかった場合でも（パーセンテージが`0.0`の場合）、アチーブメントは公表されます。
- 報告された値が、そのアチーブメントに対して以前報告された値より高い場合は、Game Center上の値は、新しい値に更新されます。アチーブメントの達成状況は下がることはありません。

達成状況が100パーセントに達すると目標アチーブメントが達成されたと見なされ、プレーヤーがアチーブメント画面を表示したときに、画像と達成済みの記述が表示されます。

アプリケーションは、`GKAchievement`オブジェクトを使用してGame Centerに達成状況を報告します。リスト 4-1に、Game Centerに達成状況を報告する方法を示します。まず、アプリケーションでサポートするアチーブメントの識別文字列を使用して、新規のアチーブメントオブジェクトを初期化します。次に、プレーヤーの達成状況を反映して、このオブジェクトの`percentComplete`プロパティを変更します。最後に、オブジェクトの`reportAchievementWithCompletionHandler:`メソッドを呼び出して、報告を送信した日時を知らせるブロックを渡します。

リスト 4-1　アチーブメントの達成状況の報告

```
- (void) reportAchievementIdentifier: (NSString*) identifier percentComplete:
(float) percent
{
    GKAchievement *achievement = [[[GKAchievement alloc] initWithIdentifier:
identifier] autorelease];
    if (achievement)
    {
```

```
        achievement.percentComplete = percent;
        [achievement reportAchievementWithCompletionHandler:^(NSError *error)
            {
                if (error != nil)
                {
                    // アチーブメントオブジェクトを保持して、後から再試行します（ここに
は示さない）
                }
            }];
    }
}
```

Game Centerに達成状況を報告できなかった場合、アプリケーションはエラーを処理しなければなりません。たとえば、達成状況を報告しようとしたときに、デバイスがネットワークに接続されていなかった場合です。アプリケーションがネットワークエラーを処理するための適切な方法は、アチーブメントオブジェクトを保持することです（もしかしたら、それを配列に追加しているかもしれません）。その後、アプリケーションは、報告が成功するまで定期的に達成状況の報告を試みる必要があります。`GKAchievement`クラスは、アプリケーションがバックグラウンドに移ったときにアチーブメントオブジェクトをアーカイブできるようにする`NSCoding`プロトコルをサポートしています。

アチーブメントの達成状況のロード

アチーブメントに対するプレーヤーの達成状況を表示するために、アプリケーションは`loadAchievementsWithCompletionHandler:`クラスメソッドを呼び出して、Game Centerからプレーヤーの達成情報を取得します。この操作が正常に終了すると、`GKAchievement`オブジェクトの配列が返されます。アプリケーションが以前達成状況を報告したアチーブメントごとに1つのオブジェクトが存在します。

プレーヤーの達成状況がロードされるのは、プレーヤーが認証された直後です。リスト 4-2（44 ページ）に、アプリケーションでこれを実装する方法を示します。

リスト 4-2　認証ブロックハンドラでのアチーブメント達成状況のロード

```
- (void) authenticatePlayer
{
    [[GKLocalPlayer localPlayer] authenticateWithCompletionHandler:^(NSError
*error) {
        if (error == nil)
        {
            [self loadAchievements];
            // その他の認証終了用のタスクをここで実行する
        }
     }];
}

- (void) loadAchievements
{   [GKAchievement loadAchievementsWithCompletionHandler:^(NSArray *achievements,
 NSError *error) {
        if (error != nil)
        {
            // エラー処理
        }
        if (achievements != nil)
```

```
        {
            // アチーブメントの配列を処理する
        }
       }];
}
```

プレーヤーがゲームを進行するにつれて、Game Centerで達成状況を更新する必要があります。以前にこのアチーブメントの達成状況を報告している場合、ゲームは、まずGame Centerからプレーヤーの達成状況をロードし、アチーブメントオブジェクトを更新しなければなりません。プレーヤーがこれまでに達成していないアチーブメントを達成した場合、アプリケーションは新規のアチーブメントオブジェクトを作成する必要があります。これらのアチーブメントオブジェクトをゲームで管理する簡単な方法は、可変の辞書を使用することです。辞書キーとしてidentifierプロパティを使用し、そのキーのコンテンツとしてアチーブメントオブジェクトを使用します。辞書を使用するためにリスト 4-1 （43 ページ）とリスト 4-2 （44 ページ）を変更する方法を以下に示します。

1. 可変の辞書プロパティをアチーブメントを報告するクラスに追加します。この辞書が、アチーブメントオブジェクトのコレクションを格納します。

```
@property(nonatomic, retain) NSMutableDictionary *achievementsDictionary;
```

2. アチーブメントの辞書を初期化します。

```
achievementsDictionary = [[NSMutableDictionary alloc] init];
```

3. アチーブメントデータをロードするコードを変更し、アチーブメントオブジェクトを辞書に追加します。

```
- (void) loadAchievements
{
   [GKAchievement loadAchievementsWithCompletionHandler:^(NSArray *achievements,
 NSError *error)
        {
            if (error == nil)
            {
                for (GKAchievement* achievement in achievements)
                    [achievementsDictionary setObject: achievement forKey:
achievement.identifier];
            }
        }];
}
```

4. 特定のアチーブメント識別子が辞書に存在するかどうかをテストするメソッドを実装します。識別子が辞書のキーとして存在しない場合は、新規のアチーブメントオブジェクトを作成して辞書に追加します。

```
- (GKAchievement*) getAchievementForIdentifier: (NSString*) identifier
{
    GKAchievement *achievement = [achievementsDictionary objectForKey:identifier];
    if (achievement == nil)
    {
        achievement = [[[GKAchievement alloc] initWithIdentifier:identifier]
autorelease];
        [achievementsDictionary setObject:achievement
forKey:achievement.identifier];
    }
    return [[achievement retain] autorelease];
}
```

5. リスト 4-1 （43 ページ）のコードを変更してgetAchievementForIdentifier:を呼び出し、アチーブメントオブジェクトを取得します。

```
- (void) reportAchievementIdentifier: (NSString*) identifier percentComplete:
(float) percent
{
    GKAchievement *achievement = [self getAchievementForIdentifier:identifier];
    if (achievement)
    {
         achievement.percentComplete = percent;
         [achievement reportAchievementWithCompletionHandler:^(NSError *error)
             {
                  if (error != nil)
                  {
                      // アチーブメントオブジェクトを保持して、後から再試行します（ここに
は示さない）
                  }
             }];
    }
}
```

アチーブメントの達成状況のリセット

アプリケーションで、プレーヤーがアチーブメントの達成状況をリセットできるようにしたい場合もあります。そのためには、resetAchievementsWithCompletionHandler:クラスメソッドを呼び出します。リスト 4-3 （46 ページ）にその方法を示します。まず、このメソッドは、前の例で作成したローカルにキャッシュされているアチーブメントオブジェクトをすべてクリアします。次に、Game Centerに保存されているプレーヤーの達成状況を削除します。

リスト 4-3　アチーブメントの達成状況のリセット

```
- (void) resetAchievements
{
// ローカルに保存されているアチーブメントオブジェクトをすべてクリアする
    achievementsDictionary = [[NSMutableDictionary alloc] init];
// Game Centerに保存されているすべての達成状況をクリアする
[GKAchievement resetAchievementsWithCompletionHandler:^(NSError *error)
    {
        if (error != nil)
            // エラー処理
    }];
}
```

アプリケーションがプレーヤーのアチーブメント達成状況をリセットすると、すべての達成情報が失われます。伏せられているアチーブメントは、今まで表示されていた場合、アプリケーションが達成状況を報告するまで再度伏せられます。しかし、アチーブメントを隠していた理由がIn App Purchaseに関連するからというだけの場合は、これらのアチーブメントを再度公表します。

Achievement View Controllerを使用したアチーブメントの表示

Game Centerアプリケーションには、アプリケーションのアチーブメントを表示する画面があります。アプリケーションにもアチーブメントビューを追加する必要があります。GKAchievementViewControllerクラスは、ゲームのアチーブメントを表示するための標準インターフェイス画面を提供します。Achievement View Controllerを使用するには、アプリケーションで作成した別のView Controllerを使用し、Achievement View Controllerをモーダルモードで表示します。

リスト 4-4 （47 ページ）に、View Controllerがアチーブメント画面を表示する方法を示します。このメソッドは、新規のAchievements Vew Controllerを作成し、アチーブメントのデリゲートをそのオブジェクト自身に設定します。次に、画面を表示します。

リスト 4-4　標準アチーブメントビューをプレーヤーに表示する

```
- (void) showAchievements
{
    GKAchievementViewController *achievements = [[GKAchievementViewController
alloc] init];
    if (achievements != nil)
    {
        achievements.achievementDelegate = self;
        [self presentModalViewController: achievements animated: YES];
    }
    [achievements release];
}
```

ユーザがアチーブメント画面をを消去したことが通知されるように、View ControllerはGKAchievementViewControllerDelegateプロトコルも実装していなければなりません。リスト 4-5に、画面からモーダルビューを削除するデリゲートの実装を示します。

リスト 4-5　プレーヤーがアチーブメントビューを消去したときの応答

```
- (void)achievementViewControllerDidFinish:(GKAchievementViewController
*)viewController
{
    [self dismissModalViewControllerAnimated:YES];
}
```

カスタムのアチーブメントユーザインターフェイスの作成

カスタムのアチーブメントユーザインターフェイスを作成する場合、Game Centerからアチーブメント記述をロードし、その記述をユーザの達成状況と組み合わせることができます。Game Kitは、GKAchievementDescriptionクラスを使用してゲームにアチーブメント記述を提供します。GKAchievementDescriptionオブジェクトのプロパティは、iTunes Connectでアチーブメントに追加した情報と一致します。このセクションでは、アプリケーションがこれらの記述をGame Centerからロードする方法を示します。

アチーブメント記述のロードは、2段階の処理で行われます。まず、アプリケーションはゲーム内のすべてのアチーブメントに対するテキスト記述をロードします。次に、達成済みの各アチーブメントに対して、アプリケーションは達成済みアチーブメントの画像をロードします。この2番目のステップによって、必要な画像だけをロードでき、アプリケーションのメモリ占有率を削減できます。

リスト 4-6に、アチーブメント記述のロード方法を示します。このコードでは、`loadAchievementDescriptionsWithCompletionHandler:`クラスメソッドを呼び出して、アチーブメント記述をロードします。

リスト 4-6　Game Centerからアチーブメントのメタデータを取得する

```
- (void) retrieveAchievmentMetadata
{
    [GKAchievementDescription loadAchievementDescriptionsWithCompletionHandler:
        ^(NSArray *descriptions, NSError *error) {
            if (error != nil)
                // エラー処理
            if (descriptions != nil)
                // このアチーブメント記述を使用する
        }];
}
```

プロパティについては自明ですが、あえて説明が必要な重要なプロパティとして`identifier`プロパティがあります。これは、`GKAchievement`オブジェクトで使用される識別文字列に対応します。カスタムユーザインターフェイスを設計する場合は、`identifier`プロパティを使用して、各`GKAchievementDescription`オブジェクトを、そのアチーブメントのプレーヤーの達成状況を記録した`GKAchievement`オブジェクトに対応させます。

`image`プロパティの値は、画像データのロードをオブジェクトに指示するまでは`nil`です。リスト 4-7に、アプリケーションが画像のロードをアチーブメント記述に指示する方法を示します。

リスト 4-7　達成済みアチーブメント画像のロード

```
[achievementDescription loadImageWithCompletionHandler:^(UIImage *image, NSError
 *error) {
    if (error == nil)
    {
        // ロードされた画像を使用するimageプロパティには、これと同じ画像がロードされる
    }
  }];
```

`GKAchievementDescription`クラスは、アプリケーションが使用できる2つのデフォルト画像を提供します。`incompleteAchievementImage`クラスメソッドは、達成されていないアチーブメントすべてに使用する画像を返します。アプリケーションが達成済みアチーブメント画像をロードできない場合（または、カスタム画像のロード中に画像を表示したい場合）は、`placeholderCompletedAchievementImage`クラスメソッドが達成済みアチーブメント用の汎用の画像を提供します。

マルチプレーヤー

複数のプレーヤーによるマッチメークは、Game Centerを使用するゲームで利用できるサービスです。マッチメークを利用すると、ゲームをプレーしている同じ趣味を持つプレーヤーが知り合い、一緒にゲームを楽しむことが簡単にできるようになります。

マッチメークをゲームに追加する際のチェックリスト

マッチメーク機能をアプリケーションに追加する前に、ゲームについて決定すべき項目がいくつかあります。たとえば、一度にプレーできるプレーヤーの人数や、ネットワーク版のゲームの外観などです。こうした設計上の決定は、Game Centerを使ったマッチメークの実装のために記述しなければならないコードに影響を与えます。マッチメーク機能をアプリケーションに追加する準備ができたら、以下のチェックリストを使用して手順を進めます。

- まだ実行していない場合は、ローカルプレーヤーを認証するためのコードをアプリケーションに追加します。詳細については、「Game Centerでのプレーヤーの使用」（23 ページ）を参照してください。
- マッチメーク画面をユーザに表示するコードを追加します。このコードで、対戦要求を作成し、それを使用してMatchmaker View Controllerを初期化します。対戦要求については、「対戦要求で始まる新規対戦の作成」（50 ページ）を参照してください。また、View Controllerの表示方法については、「マッチメークユーザインターフェイスの表示」（51 ページ）を参照してください。
- 招待を処理するコードを追加します。ローカルプレーヤーへの未処理の招待がある場合、Game Kitは招待ハンドラを呼び出します。詳細については、「ほかのプレーヤーからの招待の処理」（52 ページ）を参照してください。
- 必要であれば、プログラムで対戦相手を検索するコードを追加します。「プログラムによる対戦相手の検索」（54 ページ）を参照してください。
- 必要であれば、高度なマッチメーク機能をゲームに追加します。追加できる機能のリストについては、「高度なマッチメーク機能」（55 ページ）を参照してください。たとえば、複数のモードを提供するゲームの場合は、同じゲームモードに関心があるプレーヤーだけを検索するようにマッチメークを制限できます。

一般的なマッチメークシナリオの理解

Game Centerは、複数のプレーヤーを1つの対戦に集めるためにさまざまな方法を提供しています。

最も一般的なシナリオは、一人でゲームをしているプレーヤーが、友だちとマルチプレーヤ対戦を始めたいという場合です。ゲームは、標準マッチメークビューをモーダルモードで表示します。プレーヤーはこの画面を使用して、友だちを対戦に招待します。ホストプレーヤーが友だちを招待する場合、友だちは、ゲームに参加したいかを尋ねるプッシュ通知を受信します。友だちが招待を受け入れると、デバイスが自動的にゲームを起動します（ゲームを実行中の場合はフォアグラウンドに移します）。すべての友だちがゲームを起動すると、対戦準備が整ったという通知が送信されます。プレーヤーが対戦を開始すると、各デバイスのゲームはマッチメーク画面を消去し、その対戦を使用してゲームを開始します。

プレーヤーは、Game Centerアプリケーションを使用してネットワーク対戦を作成することもできます。マルチプレーヤーゲームに友だちを招待すると、両方のデバイス上でアプリケーションが起動します。そして、各デバイスのアプリケーションは、ゲームに参加してほしいという招待を受け取ります。このシナリオの重要な利点は、アプリケーション側では追加作業の必要がない点です。アプリケーションがすでに招待をサポートしている場合は、無料で招待を受けることができます。

ゲームアプリケーションとGame Centerアプリケーションのどちらを使用して対戦を作成しても、ホストプレーヤーは、標準のマッチメーク画面を利用して、対戦を待っていた他のGame Centerプレーヤーで空のゲームスロットを埋めることができます。たとえば、ゲームに4人のプレーヤーが必要な場合、3人の友だちグループは追加の操作をせずに4番目のスロットを埋めることができます。

ゲームは、対戦を作成するためにオートマッチ機能を提供できます。オートマッチ機能を使用すると、プレーヤーはほかのプレーヤーを対戦に招待せず、Game Centerに要求を送信します。プレーヤーは、対戦への参加を待っていたほかのプレーヤー（友だちではない）との対戦に接続されます。オートマッチ機能では、ユーザインターフェイスは表示されません。対戦を作成するときにプレーヤーに表示する内容はカスタマイズできます。

注： マッチメークは、同じアプリケーションの他のコピー（つまり、同じバンドル識別子を共有するアプリケーション）間でのみ実行できます。2つの異なるアプリケーション間でマッチメークを実行することはできません。

対戦要求で始まる新規対戦の作成

新規の対戦はすべて対戦要求（`GKMatchRequest`オブジェクト）で始まります。対戦要求には、希望の対戦パラメータを記述します。

Game Centerは、対戦要求を使用してマッチメークの動作をカスタマイズします。たとえば、アプリケーションで標準マッチメークインターフェイスを表示する場合、その要求を使用して画面の外観をカスタマイズします。一方、オートマッチ機能を採用する場合は、互換性のあるプレーヤーを検索して新規の対戦に召集するために対戦要求が使用されます。

リスト 5-1に、新規の対戦要求を作成できる最小限のコードを示します。すべての対戦要求では、対戦に参加可能なプレーヤーの最小人数と最大人数を設定しなければなりません。この設定によって、Game Centerは常にその対戦に適したプレーヤー数を対戦させることができます。

リスト 5-1　対戦要求の作成

```
GKMatchRequest *request = [[[GKMatchRequest alloc] init] autorelease];
request.minPlayers = 2;
request.maxPlayers = 2;
```

注： iOS 4.2の時点で、*minPlayers*は最低2、*maxPlayers*は4以下でなければなりません。*ホスト経由*の対戦は高度な機能です。最大16人までのプレーヤーが許されています。詳細については、「独自のサーバ上でゲームをホストする」（59 ページ）を参照してください。

マッチメークユーザインターフェイスの表示

アプリケーションで対戦を作成するときは、そのアプリケーションのView Controllerが、標準のマッチメークインターフェイス画面をモーダルモードでプレーヤーに表示します。次に、そのプレーヤーが自由に友だちを対戦に招待します。または、オートマッチ機能を使って空きスロットを埋めます。

リスト 5-2に、View Controllerがマッチメーク画面を表示する方法を示します。このコードでは、対戦要求を作成して設定し、それを使用して、新規のMatchmaker View Controllerオブジェクトを初期化します。View Controllerは、それ自体をMatchmaker View Controllerのデリゲートとして設定し、マッチメーク画面をモーダルモードでプレーヤーに表示します。プレーヤーはマッチメーク画面とやり取りして、ほかのプレーヤーの招待やマッチングを行い、最終的にゲームを開始します。

リスト 5-2　新規対戦の作成

```
- (IBAction)hostMatch: (id) sender
{
    GKMatchRequest *request = [[[GKMatchRequest alloc] init] autorelease];
    request.minPlayers = 2;
    request.maxPlayers = 2;

    GKMatchmakerViewController *mmvc = [[[GKMatchmakerViewController alloc]
initWithMatchRequest:request] autorelease];
    mmvc.matchmakerDelegate = self;

    [self presentModalViewController:mmvc animated:YES];
}
```

デリゲートは、イベントに応答するためにいくつかのメソッドを実装しなければなりません。これらの各メソッドは、View Controllerを消去してから、アプリケーションで必要なアプリケーション固有のアクションを実行しなければなりません。

`matchmakerViewControllerWasCancelled:`デリゲートメソッドは、対戦を作成する前にプレーヤーが「キャンセル（Cancel）」ボタンをタップしてマッチメーク画面を消去したときに呼び出されます。ほとんどの場合、以前のゲーム画面に戻るだけです。

リスト 5-3　キャンセルメソッドの実装

```
- (void)matchmakerViewControllerWasCancelled:(GKMatchmakerViewController
*)viewController
{
    [self dismissModalViewControllerAnimated:YES];
    // アプリケーション固有のコードをここで実装する
}
```

matchmakerViewController:didFailWithError:デリゲートメソッドは、対戦の設定中にマッチメークにエラーが発生した場合に呼び出されます。たとえば、デバイスがネットワークから切断されることがあります。このメソッドの実装では、errorプロパティを読み取ってメッセージをユーザに表示してから、以前の画面に戻ります。

リスト 5-4　エラー処理メソッドの実装

```
- (void)matchmakerViewController:(GKMatchmakerViewController *)viewController
didFailWithError:(NSError *)error
{
    [self dismissModalViewControllerAnimated:YES];
    // ユーザにエラーを表示する
}
```

最後に、対戦が作成されて、各プレーヤーの開始準備ができたら、デリゲートは、対戦が正常に作成されたことを知らせる呼び出しを受け取ります。Game Kitは、GKMatchオブジェクトをゲームに提供します。リスト 5-5 （52 ページ）に、マッチオブジェクトを受け取るために自身のクラスで実装するメソッドを示します。このメソッドは、保持用のプロパティにマッチオブジェクトを割り当て、対戦にデリゲートを追加します。すべてのプレーヤーがすでに対戦に接続している場合は、ほかの参加者とデータ交換を開始します。そうでない場合は、残りのプレーヤーが対戦に接続するのを待ちます。その場合は、マッチデリゲートが対戦を開始します。マッチデリゲートの実装方法と、対戦を使用してほかの参加者と情報を交換する方法の詳細については、「対戦を使用したネットワークゲームの実装」 （61 ページ）を参照してください。

リスト 5-5　対戦検索メソッドの実装

```
- (void)matchmakerViewController:(GKMatchmakerViewController *)viewController
didFindMatch:(GKMatch *)match
{
    [self dismissModalViewControllerAnimated:YES];
    self.myMatch = match; // 保持用のプロパティを使用して対戦を保持する
    match.delegate = self;
    if (!self.matchStarted && match.expectedPlayerCount == 0)
    {
       self.matchStarted = YES;
       // アプリケーション固有のコードを挿入して対戦を開始する
    }
}
```

ほかのプレーヤーからの招待の処理

プレーヤーが友だちを対戦に招待すると、友だちのデバイスにプッシュ通知が表示されます。その友だちが招待を受け入れると、対戦に接続できるように友だちのデバイスでアプリケーションが自動的に起動されます。アプリケーションは、招待ハンドラを実装することで招待を受け取ることができます。

招待ハンドラは、Matchmaker View Controllerを作成し、Game Kitから提供されたデータでそれを初期化します。招待ハンドラは、リスト 5-2 （51 ページ）で示されたのと同様の動作を実行します。ほとんどの場合、同じデリゲートコードを再利用できます。招待ハンドラは2つのパラメータを取ります。招待ハンドラの呼び出し時には、そのうちの1つだけにnil以外の値が格納されています。

- acceptedInviteパラメータは、ゲームが別のプレーヤーから直接招待を受け取るとnil以外の値になります。この場合は、ほかのプレーヤーのゲームがすでに対戦要求を作成しています。そのため、招待された側のデバイスで実行しているアプリケーションで対戦要求を作成する必要はありません。

- playersToInviteパラメータは、対戦をホストするGame Centerアプリケーションから直接ゲームが起動されるとnil以外の値になります。このパラメータは、ゲームが対戦に招待すべきプレーヤーを示したプレーヤー識別子の配列を保持します。ゲームは新しい対戦要求を作成し、通常通りにパラメータを割り当ててから、対戦要求のplayersToInviteプロパティをplayersToInviteパラメータで渡された値に設定する必要があります。マッチメーク画面が表示されると、すでに対戦に参加しているプレーヤーのリストがあらかじめ読み込まれます。

重要： アプリケーションがローカルプレーヤーを認証したら、できるだけ早く招待ハンドラを提供しなければなりません。このハンドラを設定するのに適した場所は、ローカルプレーヤーの認証後に実行される終了ハンドラ内です。アプリケーションが起動したらできるだけ早くローカルプレーヤーを認証して招待ハンドラを設定し、アプリケーションを起動するきっかけとなった招待を処理できるようにすることは非常に重要です。

リスト5-6に、典型的な招待ハンドラを示します。プレーヤーがすでに招待を受け入れているため、ゲームは実行中のプレーをすべて停止し、招待ハンドラのパラメータを使用して新規のView Controllerオブジェクトをインスタンス化し、それをプレーヤーに表示します。

リスト 5-6　招待ハンドラの追加

```
[GKMatchmaker sharedMatchmaker].inviteHandler = ^(GKInvite *acceptedInvite,
NSArray *playersToInvite) {
   // アプリケーション固有のコードをここに挿入して、実行中のゲームをクリーンアップする
   if (acceptedInvite)
    {
        GKMatchmakerViewController *mmvc = [[[GKMatchmakerViewController alloc]
 initWithInvite:acceptedInvite] autorelease];
        mmvc.matchmakerDelegate = self;
        [self presentModalViewController:mmvc animated:YES];
    }
    else if (playersToInvite)
    {
        GKMatchRequest *request = [[[GKMatchRequest alloc] init] autorelease];
        request.minPlayers = 2;
        request.maxPlayers = 4;
        request.playersToInvite = playersToInvite;

        GKMatchmakerViewController *mmvc = [[[GKMatchmakerViewController alloc]
 initWithMatchRequest:request] autorelease];
        mmvc.matchmakerDelegate = self;
        [self presentModalViewController:mmvc animated:YES];
    }
};
```

プログラムによる対戦相手の検索

アプリケーションは、Game Centerに、ユーザインターフェイスを表示せずにプレーヤーの対戦相手を検索するよう要求することもできます。たとえば、アプリケーションには、1回のボタンクリックで新規の対戦を要求できるように「インスタントマッチ（instant match）」ボタンを設定できます。

リスト 5-7に、プログラムで対戦相手を検索する方法を示します。このコードでは、まず対戦要求を作成し、次にマッチメーカーシングルトンオブジェクトを取得して、対戦相手を要求します。対戦相手が見つかった場合やエラーが発生した場合は、提供済みの終了ハンドラが呼び出されます。対戦が終了ハンドラに返された場合、その対戦は保持用のプロパティに割り当てられ、マルチプレーヤー対戦を開始するために使用されます。

リスト 5-7　プログラムによる対戦相手の検索

```
- (IBAction)findProgrammaticMatch: (id) sender
{
    GKMatchRequest *request = [[[GKMatchRequest alloc] init] autorelease];
    request.minPlayers = 2;
    request.maxPlayers = 4;

    [[GKMatchmaker sharedMatchmaker] findMatchForRequest:request
withCompletionHandler:^(GKMatch *match, NSError *error) {
        if (error)
        {
            // エラー処理
        }
        else if (match != nil)
        {
            self.myMatch = match; // 保持用のプロパティを使用して対戦を保持する
            match.delegate = self;
            if (!self.matchStarted && match.expectedPlayerCount == 0)
            {
               self.matchStarted = YES;
               // アプリケーション固有のコードを挿入して対戦を開始する
            }
        }
    }];
}
```

既存の対戦へのプレーヤーの追加

対戦がすでにあって、プレーヤーだけを追加したい場合もあります。たとえば、ゲームに4人のプレーヤーが必要で、1人との接続が切れてしまった場合、実行中の対戦を中止するのではなく、代わりのプレーヤーを検索するオプションを提供できます。

それには、リスト 5-7 （54 ページ）で示したようなコードを使用します。ただし、`findMatchForRequest:withCompletionHandler:`ではなく`addPlayersToMatch:matchRequest:completionHandler:`メソッドを呼び出して、プレーヤーを追加するための対戦を補足パラメータに追加します。

対戦要求のキャンセル

プログラムで対戦を検索する機能がアプリケーションに含まれている場合は、プレーヤーが対戦要求をキャンセルできるユーザインターフェイスを提供する必要があります。リスト 5-8に、処理中の検索を終わらせる方法を示します。

リスト 5-8　対戦相手検索のキャンセル

```
[[GKMatchmaker sharedMatchmaker] cancel];
```

高度なマッチメーク機能

ゲームでマッチメーク処理を実行する場合は、高度なマッチメーク機能を1つ以上アプリケーションに追加することを検討してください。

- **プレーヤーグループ**を利用すると、プレーヤーのサブセットを作成できます。アプリケーションがプレーヤーグループを使用すると、同じグループのプレーヤーだけが互いに自動的にマッチングされます。詳細については、「プレーヤーグループ」（55 ページ）を参照してください。
- **プレーヤー属性**を利用すると、ゲーム内でプレーヤーが果たすことのできる特定の役割を定義できます。対戦要求にプレーヤーの役割が含まれていると、オートマッチは、その役割を埋めるプレーヤーを必要とする対戦にのみプレーヤーを配置します。詳細については、「プレーヤー属性」（56 ページ）を参照してください。
- **プレーヤーアクティビティ**を利用すると、オンラインでプレーしているプレーヤーのおおよその数をGame Centerに照会できます。これによって、すぐに対戦相手が見つかるかどうかをプレーヤーに伝えることができます。詳細については、「プレーヤーアクティビティの検索」（58 ページ）を参照してください。
- **ホスト型対戦**を利用すると、Game Centerのマッチメークサービスを使用してプレーヤーを検索できます。ただし、独自のサーバを使用してプレーヤーを互いに接続します。ホスト型対戦には、独自のネットワークコードを記述する必要がありますが、より大きなプレーヤーグループと独自のサーバを対戦に追加する機能を提供できます。詳細については、「マッチメークをゲームに追加する際のチェックリスト」（49 ページ）を参照してください。

これらの高度な機能は、別々に使用できます。また、同じアプリケーション内で組み合わせて使用することもできます。

プレーヤーグループ

Game Centerのデフォルトの動作では、複数のプレーヤーを必要とする対戦ゲームでのプレーを待っているすべてのプレーヤーを自動的にマッチングします。これには、対戦相手を素早く選び出せるという利点がありますが、それが希望通りの動作でない場合もあります。同じ趣味を持つプレーヤーだけを互いにマッチングしたい場合もあります。そのため、プレーヤーを小さなグループにまとめたいことがあります。たとえば、以下のいずれかを実行できます。

- プレーヤーのスキルレベルごとにプレーヤーを分ける。

- 対戦の判定に使用されるルールセットごとにプレーヤーを分ける。
- 対戦の舞台となる地図に基づいてプレーヤーを分ける。

プレーヤーグループを使用して、このプレーヤーの分類を実装します。プレーヤーグループを利用すると、自分が希望するタイプの対戦相手と直接マッチングされます。

プレーヤーグループの実装

プレーヤーグループは対戦要求の`playerGroup`プロパティで表します。`playerGroup`プロパティに0以外の値を保存すると、プレーヤーは、対戦要求に同じ32ビットの値を指定したプレーヤーとのみマッチングされます。ゲームを設計するときに、提供する値と値を提供する時期を決定します。たとえば、プレーヤーが興味のあるプレースタイルを定義するために使用するパラメータを選択できるように、ゲームに独自のユーザインターフェイスを表示します。プレーヤーが選択する各オプションによって、`playerGroup`プロパティに保存される値の一部が提供されることがあります。

リスト5-9では、対戦要求を初期化してプレーヤーグループを追加しています。この例では、プレーヤーグループの値は、対戦の舞台となる地図を表す定数と、対戦で使用されるルールを表す定数の間でOR演算を実行することによって計算されます。

リスト 5-9　地図とルールセットに基づくプレーヤーグループの作成

```
GKMatchRequest *request = [[[GKMatchRequest alloc] init] autorelease];
request.minPlayers = 2;
request.maxPlayers = 4;
request.playerGroup = MyMap_Forest | MyRulesCaptureTheFlag;
```

プレーヤー属性

ゲームによっては、プレーヤーがマルチプレーヤ対戦で演じることのできるさまざまな役割を提供するものがあります。次に例を示します。

- 多くのロールプレーイングゲームには、さまざまなキャラクターの役割（クラス）があります。役割ごとに、さまざまな強み、弱点、およびプレーヤーが対戦に持ち込める機能を定義します。
- 同様に、スポーツゲームでは、プレーヤーがフィールド上でプレーする特定のポジション（クオーターバックやラインバッカーなど）を役割と考えます。

プレーヤー属性を使用すると、対戦の開始後にプレーヤーに特定の役割の選択を許可する機能をゲームに追加できます。Game Centerは、特定の役割を埋めるためのプレーヤーを必要とする対戦に、自動的にプレーヤーをマッチングします。

プレーヤー属性には以下の制限があります。

- 各役割を埋めることのできるプレーヤーは1人だけです。
- 一連の役割を定義し、ゲームで定義されたすべての役割が埋まっていなければなりません。
- 対戦要求ごとに1つの役割しか要求できません。
- 役割は、オートマッチ機能で選ばれたプレーヤーに対してのみチェックされます。友だちを対戦に招待した場合、友だちは役割を選択できません。

- 役割は、Game Kitから提供される標準のマッチングユーザインターフェイスには表示されません。プレーヤーが役割を選択できるようにするには、アプリケーションでカスタムユーザインターフェイスを用意しなければなりません。
- アプリケーションに返されるマッチオブジェクトからは、プレーヤーが選択した役割はわかりません。役割選択情報は、対戦を作成した後で別途送信する必要があります。

プレーヤー属性の実装

プレーヤー属性は、対戦要求の`playerAttributes`プロパティを使用して実装します。デフォルトでは、このプロパティの値は`0`で、このプロパティが無視されることを意味します。値が0以外の場合、Game Centerはその値を使用して役割を見つけます。

ゲーム内でプレーヤーが果たす役割を作成するには、プレーヤーがゲーム内で果たすことのできる役割ごとに32ビットのマスクを1つ作成します。プレーヤーが役割を選択できるように、カスタムユーザインターフェイスをゲームに追加するのはデベロッパの責任です。対戦要求の`playerAttributes`プロパティをプレーヤー選択のマスクに設定します。その後、通常通り、マッチメークインターフェイスを表示するか、オートマッチを実行します。

リスト 5-10　対戦要求のプレーヤー属性の設定

```
GKMatchRequest *request = [[[GKMatchRequest alloc] init] autorelease];
request.minPlayers = 4;
request.maxPlayers = 4;
request.playerAttributes = MyRole_Wizard;
```

Game Centerがオートマッチを使用して対戦相手を検索するとき、プレーヤーのマスクが対戦にすでに参加しているプレーヤーのマスクと重複しない場合にのみ、プレーヤーを対戦に追加します。このアルゴリズムは、大まかにまとめると次のようになります。

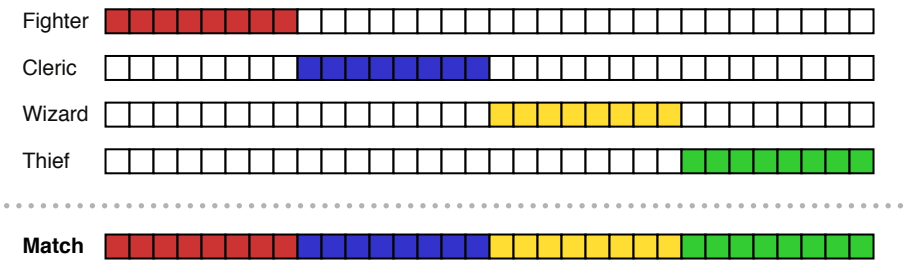
1. 対戦のマスクは、招待側のプレーヤーのマスクで始まります。
2. Game Centerは、プレーヤー属性値が0以外の対戦要求を使用してプレーヤーを検索します。Game Centerは、要求元のプレーヤーの属性マスクが、対戦の現在のマスクと重複しない場合にのみ、プレーヤーを対戦に追加します。つまり、対戦のマスクとプレーヤーのマスクの間に論理演算子ANDを使用している場合は、結果が`00000000h`になります。
3. プレーヤーを対戦に追加したら、新規のプレーヤーのプレーヤー属性値が対戦のマスクに追加されます。
4. 対戦のマスクが`FFFFFFFFh`に等しくなったら、マッチングが終了したと見なします。それ以外の場合は、別のプレーヤーを探します。

わかりやすくするために例を示します。Fighter（戦士）、Wizard（魔法使い）、Thief（盗賊）、Cleric（聖職者）の4つの役割があるロールプレーイングゲームを作成中だとします。すべての対戦は4人のプレーヤーで構成され、対戦にはそれぞれの役割に1人だけプレーヤーが必要です。この要件を実装するために、上記の4ステップのアルゴリズムに合うように役割ごとのマスクを作成します。リスト 5-11 （57 ページ）にマスクの設定例を示します。

リスト 5-11　キャラクタークラスのマスクの作成

```
#define MyRole_Fighter 0xFF000000
#define MyRole_Cleric 0x00FF0000
```

```
#define MyRole_Wizard 0x0000FF00
#define MyRole_Thief 0x000000FF
```



どの役割マスクも重複しません。任意の2つのマスクがANDで結合される場合、結果の値は常に00000000hになります。4つのすべての役割マスクを1つにまとめると、すべての対戦マスクが埋まります（FFFFFFFFh）。

プレーヤーアクティビティの検索

通常、マルチプレーヤーの対戦相手を探しているプレーヤーは、すぐに対戦できることを望んでいます。少なくとも、マッチメークに長い時間がかかるかどうかを知りたいと思っています。たとえば、多くのプレーヤーが通常オンラインにいない時間帯に接続している場合は、対戦に参加しそうなプレーヤーの数は、ゴールデンタイムよりかなり少ない可能性があります。オンラインのプレーヤー数の特定を**プレーヤーアクティビティ**と呼びます。GKMatchmakerクラスは、アプリケーションに関連するGame Center上のアクティビティをテストするために使用できる2つのメソッドを提供しています。

リスト 5-12　Game Center上でアプリケーションに関連するすべてのアクティビティを検索する

```
- (void)findAllActivity
{
    [[GKMatchmaker sharedMatchmaker]
queryActivityWithCompletionHandler:^(NSInteger activity, NSError *error) {
        if (error)
        {
            // エラー処理
        }
        else
        {
            // このアクティビティ値を使用して、アクティビティをプレーヤーに表示する
        }
    }];
}
```

アプリケーションでプレーヤーグループを使用している場合は、queryPlayerGroupActivity:withCompletionHandler:メソッドを使用して、特定のプレーヤーグループのアクティビティを取得できます。

いずれかのメソッドから返された値は、最近対戦を要求したプレーヤーの数です。

独自のサーバ上でゲームをホストする

デフォルトでは、Game Kitを使用して対戦を作成します。Game Kitは、対戦に接続される各デバイスでGKMatchオブジェクトのインスタンスを返します。各デバイスは、マッチオブジェクトを使用してほかの参加者とやり取りします。ほとんどのゲームでは、このデフォルトの動作がまさに希望通りの動作です。GKMatchクラスは、データの送信方法を気にせずにほかの参加者にデータを送ることにだけ集中できるように、基盤となるネットワークトポロジーを抽象化します。ただし、Game Centerでサポートされる最大プレーヤー数よりも多くのプレーヤーをサポートする必要があるゲームや、独自のサーバで対戦を仲介する必要があるゲームもあります。このような場合は、マッチメークを使用して**ホスト型対戦**のためのプレーヤーを検索します。ホスト型対戦では、GKMatchオブジェクトを作成しません。代わりに、各デバイスのアプリケーションが対戦に参加するすべてのプレーヤーのプレーヤー識別子を受け取ります。サーバにプレーヤーを接続するには、追加ステップを実行する必要があります。

ホスト型対戦を作成するには、アプリケーション側で、アプリケーションに必要な下位レベルのネットワーク処理をすべて実装する必要があります。特に、以下の処理をすべてアプリケーション側で行わなければなりません。

- 各デバイスをサーバに接続するために、独自のネットワークコードを設計、実装する。
- 対戦へのすべての参加者の状態を他のデバイスに通知するために、独自のネットワークプロトコルを設計、実装する。
- Game Kitの標準マッチメークユーザインターフェイスを使用する場合は、サーバに接続したら各デバイスがGame Kitに必ず通知するようにしなければなりません。この情報により、Game Kitはユーザインターフェイスを更新できます。
- プレーヤーのプレーヤー識別文字列を使用してデータをデバイスに送信するには、独自のサーバ実装を設計して実行する必要があります。
- GKMatchオブジェクトを使用していないため、ボイスチャットは提供されません。ただし、GKVoiceChatServiceクラスを使って、デベロッパが用意したネットワーク接続によるボイスチャットを実装できます。詳細については、「ゲーム内ボイス」（87 ページ）を参照してください。

注： ホスト型対戦は最低2人、最高16人のプレーヤーをサポートできます。

これ以降は、ホスト型対戦を作成するためにマッチメークコードを変更する方法を説明します。

標準のマッチメークユーザインターフェイスを使用してホスト型対戦を作成するか、プログラムによってホスト型対戦を作成するかにかかわらず、動作は同じです。初期化されたGKMatchオブジェクトを受け取る代わりに、アプリケーションは、対戦に接続されたプレーヤーのプレーヤー識別子のリストを受け取ります。各ゲームクライアントは、サーバとの接続を確立して、対戦相手のプレーヤーのリストと、認証済みのローカルプレーヤーのプレーヤー識別子をサーバに送信しなければなりません。次に、プレーヤー識別子のリストを取得して、参加者同士をつなぐために必要なゲームのロジックを実行します。

View Controllerを利用したホスト型対戦の作成

View Controllerを利用してホスト型対戦を作成するには、プレーヤーにマッチメーク画面を表示する前に、View Controllerの`hosted`プロパティをyesに設定します。リスト 5-13は、リスト 5-2 （51 ページ）で示したメソッドを修正したものです。

リスト 5-13　ホスト型対戦の作成

```
- (IBAction)hostMatch: (id) sender
{
    GKMatchRequest *request = [[[GKMatchRequest alloc] init] autorelease];
    request.minPlayers = 2;
    request.maxPlayers = 2;

    GKMatchmakerViewController *mmvc = [[[GKMatchmakerViewController alloc]
initWithMatchRequest:request] autorelease];
    mmvc.matchmakerDelegate = self;
    mmvc.hosted = YES;

    [self presentModalViewController:mmvc animated:YES];
}
```

同様に、招待ハンドラが招待を受け取ったら、`GKInvite`オブジェクトの`isHosted`プロパティが、対戦がホスト型対戦として作成されたかどうかを示します。招待ハンドラは、Matchmaker View Controllerの`hosted`プロパティを適切に設定しなければなりません。

ユーザインターフェイスを表示するだけでなく、各デバイスはサーバに接続しなければなりません。デバイスがサーバに接続したら、サーバはホスト型対戦に接続されているすべてのデバイスに通知しなければなりません。各デバイスは、View Controllerの`setHostedPlayerReady:`メソッドを呼び出して、いま接続されたプレーヤーのプレーヤー識別子を渡さなければなりません。これにより、参加者全員のデバイスのマッチメーク画面が更新されます。

すべての参加者がサーバに接続して、プレーヤーの準備が整ったら、ゲームを始めるためにデリゲートオブジェクトが呼び出されます。標準のGame Centerのマッチメークでは、終了したマッチオブジェクトを受け取るために`matchmakerViewController:didFindMatch:`メソッドを実装しました。ホスト型対戦の場合は、代わりにアプリケーションに`matchmakerViewController:didFindPlayers:`メソッドを実装します。

プログラムによるホスト型対戦の作成

プログラムによるホスト型対戦の作成は、通常の対戦の作成と同じです。対戦要求を作成して、共有のマッチメーカーシングルトンを取得し、それを使用して対戦を検索します。ホスト型対戦を作成するには、代わりにマッチメーカーの`findPlayersForHostedMatchRequest:withCompletionHandler:`メソッドを呼び出します。このメソッドを呼び出すときに、対戦に参加するプレーヤーのプレーヤー識別子の配列を受け取ります。

対戦を使用したネットワークゲームの実装

対戦は、Game Centerによってネットワーク経由で互いに接続されたデバイスを持つプレーヤーのグループです。対戦を利用すると、その対戦のほかの参加者にデータや音声を送信できます。Game Centerは、ほかのプレーヤーを検索したり、プレーヤー間のネットワークを確立する際の難しい作業をこなします。そのため、デベロッパは自身のネットワークゲームを自由に設計できます。

対戦を使用する際のチェックリスト

ゲーム用のネットワークコードを実装するには、以下の手順に従います。

- まだ記述していない場合は、Game Centerのマッチメークサービスを使用して対戦を作成するためのコードを記述します。マッチメークサービスは、プレーヤー間の接続を確立して、`GKMatch`オブジェクトをアプリケーションに返します。次に、アプリケーションはこのマッチオブジェクトとやり取りして、ほかの参加者にデータを送信します。対戦相手の作成の詳細については、「マルチプレーヤー」（49 ページ）を参照してください。
- 対戦のほかの参加者とやり取りするためにゲームで使用するネットワークメッセージを設計します。ほとんどのゲームでは、対戦の開始時に情報を交換して、各参加者に同じ初期ゲーム状態を提供します。その後、ゲーム内でイベントが起こったら更新を送信します。詳細については、「ネットワークゲームの設計」（62 ページ）を参照してください。
- マッチオブジェクトを使用して、他の対戦参加者にデータを送信するためのコードを記述します。詳細については、「ほかのプレーヤーへのデータの送信」（64 ページ）を参照してください。
- 対戦中に発生する可能性がある必要なイベントをすべて処理するために、次のようなマッチデリゲートを実装します。
    - デリゲートには、ゲームの起動時にほかのプレーヤーが接続されたことが通知されます。デリゲートは、ゲームを開始する前に全員が接続されるまで待たなければなりません。詳細について「対戦の開始」（64 ページ）を参照してください。
    - デリゲートは、ほかのプレーヤーが送信したデータを受信します。詳細については、「ほかのプレーヤーからのデータの受信」（65 ページ）を参照してください。
    - ゲームの実行中にプレーヤーとの接続が切れた場合は、デリゲートに通知されます。このときに対戦を中断するか、プレーヤー数の減少に対応するためにゲームを再設定するかを決定しなければなりません。詳細については、「対戦からの切断」（65 ページ）を参照してください。
- 必要であれば、対戦参加者同士の音声サポートを追加します。詳細については、「対戦へのボイスチャットの追加」（67 ページ）を参照してください。

ネットワークゲームの設計

ゲームの各インスタンスは、GKMatchオブジェクトを利用して、対戦に接続された他のインスタンスとデータを交換します。GKMatchクラスには、ネットワークメッセージの書式や内容は定義されていません。ネットワークメッセージは単に転送すべきバイトと見なされます。このため、ネットワークゲームの設計が非常に柔軟になります。このセクションの残りの部分では、ネットワークゲームを実装する前に理解しておかなければならない重要な概念について説明します。

データをほかの参加者に送信するときはいつも、データを送信するために対戦で使用する作業量を判断します。対戦では、ターゲットがデータを受信するまでデータを再送信する**確実な**データ送信、またはデータを一度しか送信しない**確実性の低い**データ送信が可能です。

- 確実な送信のほうがシンプルですが、速度が遅くなる可能性があります。遅かったり、エラーの発生しやすいネットワークでは、目的の送信先に正常に到達するまでにデバイスが何回もメッセージを送信しなければならないことがあります。また、対戦では、1つのデバイスから同じ送信先に確実な送信方法で複数回送信したメッセージは、送信した順番で届けられます。

- 確実性の低い方法で送信したメッセージは、送信先に届かなかったり、順番とは関係なく配送される場合があります。確実性の低い送信は、送信が遅れるとメッセージの内容が役に立たなくなるリアルタイムのトランザクションに対して最も有用です。たとえば、ゲームが推測航法アルゴリズム用の位置と速度の情報を送信する場合、送信が遅れると送信先に届くまでに位置データが大幅に狂うことになります。この場合は、確実性の低い方法でメッセージを送信する代わりに、更新された位置情報と計算情報を含んだ新しいメッセージを送信します。

また、メッセージのサイズも、送信先ににデータを迅速に配送するうえで重要な役割を果たします。大きなメッセージは、より小さなパケット（**フラグメント**と呼ぶ）に分割してから、送信先で再度組み立てる必要があります。これらの小さなパケットは、送信中に失われたり、順番が狂って配送される場合もあります。大きなメッセージは確実に送信されなければならないため、Game Kitはフラグメント化とアセンブリを処理できます。ただし、再送信とアセンブリの処理には時間がかかります。大量のリアルタイムデータを送信する場合は、確実な送信方法を採用するべきではありません。

ネットワークデータは、自分では管理できないサーバやルータを経由して送信されていることを忘れないでください。メッセージは、ネットワーク上の他のデバイスによって閲覧されたり修正される可能性があります。デバイスのアプリケーションが他のデバイスの参加者からネットワークデータを受信したら、そのメッセージを*信頼*できないデータと見なして、使用する前に内容を検証しなければなりません。セキュリティの脆弱性を回避する方法の詳細については、『*Secure Coding Guide*』を参照してください。

以下に、ゲームのネットワークを設計する際の一般的なガイドラインを示します。

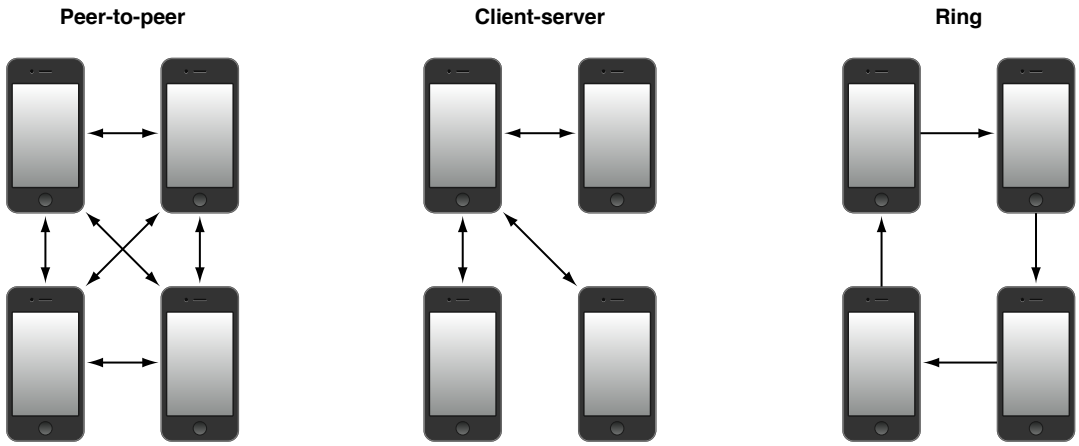
- メッセージの書式には、さまざまな種類のメッセージを識別する方法を含める必要があります。GKMatchクラスはメッセージの内容をまったく理解しないため、アプリケーションにその機能を実装する必要があります。たとえば、さまざまな種類のメッセージを識別する列挙型を作成し、各メッセージをこの列挙型で始めます。

- メッセージは、ゲームが正常に動作できる最低限の頻度で送信します。ゲームのグラフィックエンジンは、毎秒30～60フレームで動作しますが、ネットワークコードはそれよりもはるかに低い頻度で更新を送信できます。

- ジョブが終了したことを知らせるメッセージの書式は最小サイズにします。頻繁に送信されるメッセージや、ほかの参加者にすぐに届かなければならないメッセージは、不必要なデータを送信しないように慎重に定義する必要があります。
- データは、重要な情報を失わない程度の最小限の表現に圧縮します。たとえば、プログラム内の整数は、32ビットまた64ビットを使用してデータを格納します。整数に格納される値が、常に1から10の範囲にあれば、わずか4ビットでネットワークメッセージに格納できます。
- 確実性の低い方法で送信するメッセージのサイズは、1000バイト以下に抑えます。
- 確実な方法で送信するメッセージのサイズは、87キロバイト以下に抑えます。
- メッセージ内の情報を必要とする参加者にのみ、メッセージを送信します。たとえば、ゲーム内に2つの異なるチームがある場合、チームに関連するメッセージは、そのチームにメンバーにのみ送信するべきです。対戦のすべての参加者にデータを送信すると、わずかなネットワーク帯域幅を使い果たしてしまいます。

  `GKMatch`オブジェクトは、すべての参加者の間に完全なピアツーピア接続を作成しますが、その上にリングネットワークやクライアントサーバネットワークのアーキテクチャレイヤを作成すると、ネットワークトラフィックを削減できます。図6-1（63ページ）に、4人のプレーヤーによるゲームで考えられる3つのネットワークトポロジーを示します。左側のピアツーピアのゲームは、さまざまなデバイス間に12個の接続を持ちます。しかし、デバイスの1つをホストとして機能させることによって、この上にクライアントサーバアーキテクチャのレイヤを作成できます。アプリケーションがホストとのみ送受信する場合は、接続数を半分にできます。リングアーキテクチャを利用すると、デバイスは隣のデバイスにネットワークパケットを転送するだけになり、さらに接続数を削減できます。それぞれのトポロジーには異なるパフォーマンス特性があります。このため、アプリケーションで必要なパフォーマンスを発揮するモデルを見つけるためにテストを行う必要があります。

  図 6-1　ネットワークトポロジー



- ネットワークの中断を処理する方法を指定します。ネットワークは本来確実性の低い通信手段です。参加者は、対戦中にいつでも切断される可能性があります。そのため、ゲームで切断メッセージを処理する必要があります。たとえば、クライアントサーバトポロジーを使用してゲームを実装している場合は、対戦から切断されると、新規のサーバとなる新しいデバイスを指名する必要があります。

対戦の開始

Game KitがGKMatchオブジェクトをゲームに渡すときは、対戦中のほかのプレーヤーへの接続が確立していないことがあります。ユーザのデバイス上のplayerIDs配列は、ほかのプレーヤーが誰も接続していない場合に空になったり、すでに接続されているプレーヤーのサブセットを保持したりする可能性があります。ゲームは、対戦を開始する前に、すべてのプレーヤーが接続されるまで待たなければなりません。対戦への参加を待っているプレーヤーが何人いるかを特定するには、その対戦のexpectedPlayerCountプロパティを読み取ります。このプロパティの値が0になると、すべてのプレーヤーが接続され、対戦の開始準備ができたことになります。

このチェックを実施するのに適した場所は、マッチデリゲートのmatch:player:didChangeState:メソッド内です。このメソッドは、対戦のメンバが接続または切断されるたびに呼び出されます。リスト 6-1 （64 ページ）は、match:player:didChangeState:メソッドの実装例です。この例では、マッチデリゲートが自身のmatchStartedプロパティを使用して、対戦がすでに実行中かどうかを記録します。対戦がまだ開始されていない場合や参加予定のプレーヤー数が0になった場合は、メソッドが対戦を開始します。ゲームでは、ここで、対戦の初期状態をほかのプレーヤーに送信したり、さまざまな参加者との間で追加のネゴシエーションを実行します。

リスト 6-1　対戦の開始

```
- (void)match:(GKMatch *)match player:(NSString *)playerID
didChangeState:(GKPlayerConnectionState)state
{
    switch (state)
    {
        case GKPlayerStateConnected:
            // 新規のプレーヤー接続の処理
           break;
        case GKPlayerStateDisconnected:
            // プレーヤーが切断した場合
           break;
    }
    if (!self.matchStarted && match.expectedPlayerCount == 0)
    {
        self.matchStarted = YES;
        // 最初の対戦ネゴシエーションを処理する
    }
}
```

ほかのプレーヤーへのデータの送信

1つのデバイスから他のデバイスにデータを送信するには、メッセージを作成してNSDataオブジェクトにカプセル化します。sendDataToAllPlayers:withDataMode:error:メソッドを使用すると、接続中のすべてのプレーヤーにこのメッセージが送信されます。また、sendData:toPlayers:withDataMode:error:を使用すると、プレーヤーのサブセットに送信されます。

リスト 6-2に、アプリケーションがほかの参加者に位置の更新を送信する様子を示します。リスト 6-2は構造に位置データを埋め、NSDataオブジェクトでラップします。次に、対戦のsendDataToAllPlayers:withDataMode:error:メソッドを呼び出します。

リスト 6-2　すべてのプレーヤーに位置を送信する

```
- (void) sendPosition
{
    NSError *error;
    PositionPacket msg;
    msg.messageKind = PositionMessage;
    msg.x = currentPosition.x;
    msg.y = currentPosition.y;
    NSData *packet = [NSData dataWithBytes:&msg length:sizeof(PositionPacket)];
    [match sendDataToAllPlayers: packet withDataMode: GKMatchSendDataUnreliable
 error:&error];
    if (error != nil)
    {
        // エラー処理
    }
}
```

エラー報告なしでメソッドが戻ると、メッセージはキューに入り、ネットワークが利用可能になった時点で送信されます。

ほかのプレーヤーからのデータの受信

対戦はほかの参加者から送信されたデータを受信すると、マッチデリゲートの`match:didReceiveData:fromPlayer:`メソッドを呼び出してメッセージを渡します。このメソッドを実装すると、メッセージをデコードし、その内容を実行する必要があります。

```
- (void)match:(GKMatch *)match didReceiveData:(NSData *)data fromPlayer:(NSString
 *)playerID
{
    Packet *p = (Packet*)[data bytes];
    if (p.messageKind == PositionMessage)
        // 位置メッセージの処理
}
```

対戦からの切断

プレーヤーが対戦を終了する準備ができたら、マッチオブジェクトの`disconnect`メソッドを呼び出さなければなりません。さらに、デバイスが一定時間応答しない場合は、プレーヤーの接続が自動的に切断される場合もあります。プレーヤーが対戦から切断されると、マッチデリゲートの`match:player:didChangeState:`メソッドを呼び出すことによって対戦のほかの参加者に通知されます。

対戦へのボイスチャットの追加

GKMatchオブジェクトを使用してプレーヤー間にネットワーク接続を提供する場合、ゲームにはゲーム内ボイスの組み込みサポートがあります。1つ以上の異なるボイスチャット**チャンネル**を作成して、対戦に接続しているプレーヤーの一部を各チャンネルに入れることができます。1人のプレーヤーがチャンネルに話しかけると、同じチャンネルに接続されている参加者だけがそのプレーヤーの話を聞くことができます。

ボイスチャットは、Wi-Fi接続を使用している対戦参加者のみが利用できます。

ボイスチャットを対戦に追加する際のチェックリスト

ゲームでは、対戦を作成してプレーヤーが接続した後に、1つ以上のボイスチャンネルが作成され、アクティブ化されます。次の手順に従ってください。

- ゲームで必要なチャンネル数を決定します。たとえば、自由参加型のゲームであれば、すべての参加者用にチャンネルが1つ必要です。複数のチームが参加するゲームであれば、チームごとに別々のチャンネルと、ゲーム内のプレーヤー全員用の追加チャンネルが必要です。
- オーディオセッションを設定して、マイクを有効にします。オーディオを録音または再生するアプリケーションにはすべて、オーディオセッションが必要です。詳細については、「オーディオセッションの作成」（68 ページ）を参照してください。
- ボイスチャットのチャンネルを作成するには、マッチオブジェクトの`voiceChatWithName:`メソッドを呼び出します。このオブジェクトはチャンネルを提供する`GKVoiceChat`オブジェクトを返します。詳細については、「ボイスチャンネルの作成」（68 ページ）を参照してください。
- チャンネルをアクティブにするときは、ボイスチャットオブジェクトの`start`メソッドを呼び出します。詳細については、「ボイスチャットの開始と停止」（69 ページ）を参照してください。
- プレーヤーがチャンネルに話しかけられるようにする場合は、そのチャンネルのマイクを有効にします。アプリケーションに複数のチャンネルがある場合、マイクを使用できるのは一度に1つのチャンネルに限られます。詳細については、「マイクの有効化と無効化」（69 ページ）を参照してください。
- ユーザがボイスチャットの有効／無効を切り替えるためのコントロールをアプリケーション側で用意します。また、ユーザが音量を設定したり、チャンネル内のプレーヤーをミュートにするためのコントロールも用意します。詳細については、「ボイスチャットの音量調節」（69 ページ）を参照してください。

- アプリケーションがプッシュツートークによる通話方式をサポートするかどうか、または連続的にマイクをサンプリングするかどうかを決定します。プッシュツートークを採用する場合は、音声データをほかのプレーヤーに送信するために押すコントロールを提供する必要があります。連続的にマイクをサンプリングするアプリケーションでは、送信のオンとオフを切り替えるコントロールを提供します。
- 必要に応じて、プレーヤーが接続または切断したときや、会話を開始または停止したときに呼び出される更新ハンドラを実装します。たとえば、更新ハンドラを使用すると、プレーヤーが話すと変更されるようにユーザインターフェイスを更新できます。詳細については、「プレーヤーの状態更新ハンドラの実装」（70 ページ）を参照してください。

オーディオセッションの作成

マッチオブジェクトが提供するボイスチャットサービスを使用する前に、音の再生と録音が両方とも可能なオーディオセッションを作成しなければなりません。ゲームでその他の音響効果が使われている場合は、すでにオーディオセッションが作成されているかもしれません。リスト 7-1（68 ページ）に、マイクを使用できるオーディオセッションの作成に必要なコードを示します。

リスト 7-1　オーディオセッションの再生と録音の設定

```
AVAudioSession *audioSession = [AVAudioSession sharedInstance];
[audioSession setCategory:AVAudioSessionCategoryPlayAndRecord error:myErr];
[audioSession setActive: YES error: myErr];
```

オーディオセッションの作成と使用の詳細については、『*Audio Session Programming Guide*』を参照してください。

ボイスチャンネルの作成

アプリケーションでは、直接`GKVoiceChat`オブジェクトを作成せずに、`GKMatch`オブジェクトによってボイスチャットオブジェクトを作成する必要があります。1つの対戦で複数のチャンネルを作成できます。また、1人のプレーヤーを一度に複数のチャンネルに割り当てることもできます。いずれかのチャンネルで受信したオーディオはミキシングされて、スピーカーを通して出力されます。

異なるデバイス上の複数の参加者が同じチャンネルに参加するには、特定のチャンネルを識別する手段が必要です。この識別には**チャンネル名**を使用します。チャンネル名は、アプリケーションが定義したチャンネルに一意に指定される文字列です。2人以上の参加者が同じ名前のチャンネルに参加すると、お互いにボイスチャットに自動で接続されます。

リスト 7-2に、2つのチャンネルを作成するコードを示します。`voiceChatWithName:`への最初の呼び出しでチームチャンネルを作成し、2回目の呼び出しでグローバルチャンネルを作成します。このコードでは、retainを指定して両方のチャンネルを保持しています。

リスト 7-2　ボイスチャンネルの作成

```
GKMatch* match;
GKVoiceChat *teamChannel = [[match voiceChatWithName:@"redTeam"] retain];
GKVoiceChat *allChannel = [[match voiceChatWithName:@"allPlayers"] retain];
```

ボイスチャットの開始と停止

ボイスチャットが開始されるまで、音声データがボイスチャンネルを通じて送受信されることはありません。ボイスチャットを開始するには、ボイスチャットオブジェクトのstartメソッドを呼び出します。

```
[teamChannel start];
```

startメソッドが呼び出された後、デバイスがWi-Fiネットワーク上にありデバイスにマイクが接続されていれば、デバイス上のボイスチャットオブジェクトはそのチャンネルのほかの参加者に接続します。この2つの条件が満たされるまで、ボイスチャットオブジェクトはチャンネルへの接続を待機します。

同様に、プレーヤーがチャンネルを離れる準備ができた場合や、一時的にチャンネルを無効にしたい場合は、次のようにしてチャットを停止します。

```
[teamChannel stop];
```

（単にほかのプレーヤーをミュートするのではなく）チャンネルを停止することの利点は、ほかのプレーヤーがチャンネルを離れているプレーヤーにデータを送信する必要がなくなる点です。データ伝送が減少するため、より多くの帯域幅をゲームのメッセージングに使用できます。

マイクの有効化と無効化

プレーヤーが話せるようにする場合は、マイクを有効にします。ゲームの性質によっては、ゲームの実行中に常時マイクを有効にしたり、インターフェイスにプッシュツートークボタンを追加したりする必要があります。

チャンネルでマイクを有効にするには、次のようにして、ボイスチャットオブジェクトのactiveプロパティをYESに設定します。

```
teamChannel.active = YES;
```

マイクを有効にできるのは一度に1つのチャンネルだけです。1つのチャンネルに対してマイクを有効にすると、以前の所有者のactiveプロパティが自動的にNOに設定されます。

ボイスチャットの音量調節

ゲームでボイスチャットの音量を制御するには2つの方法があります。1つは、ボイスチャットオブジェクトのvolumeプロパティを変更して、そのチャット全体の音量を設定する方法です。

```
allChannel.volume = 0.5;
```

volumeプロパティは、0.0以上1.0以下の値を取ります。0.0の値は、チャンネル全体を消音します。1.0の値は音声データを最大の音量で出力します。

もう1つは、1つのチャンネル内のプレーヤーを選択的にミュートする方法です。プレーヤーをミュートする場合は通常、ミュートするプレーヤーを選択できるようにするユーザインターフェイスを用意しなければなりません。プレーヤーをミュートするには、ボイスチャットオブジェクトの`setMute:forPlayer:`メソッドを呼び出します。

```
[teamChannel setMute: YES forPlayer inPlayerID];
```

プレーヤーのミュートを解除するには、`NO`を引数として渡して同じメソッドを呼び出します。

```
[teamChannel setMute: NO forPlayer inPlayerID];
```

プレーヤーの状態更新ハンドラの実装

ゲームでプレーヤーの状態変化を通知する必要がある場合は、更新ハンドラを実装できます。更新ハンドラは、プレーヤーがチャンネルを接続または切断したときや、通話を開始または停止したときに、ボイスチャットオブジェクトが呼び出すブロックオブジェクトです。たとえば、更新ハンドラを使用すると、プレーヤーが話しているときに、ユーザインターフェイスにそのプレーヤーの名前を強調表示することができます。

リスト 7-3 （70 ページ）に、プレーヤーの状態の更新ハンドラの実装を示します。

リスト 7-3　プレーヤーの更新の受信

```
teamChat.playerStateUpdateHandler = ^(NSString *playerID, GKVoiceChatPlayerState
 state) {
     switch (state)
     {
         case GKVoiceChatPlayerSpeaking:
            // プレーヤーの画像を強調表示するコードを挿入する
             break;
         case GKVoiceChatPlayerSilent:
            // プレーヤーの画像をグレー表示にするコードを挿入する
             break;
      }
   };
```

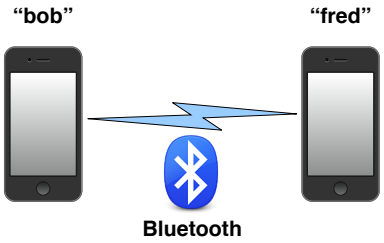
ピアツーピア接続

パートIIでは、アプリケーションでピアツーピア接続機能を使用する方法を説明します。

ピアツーピア接続

GKSessionクラスを使うと、図 1に示すような、アドホックなBluetoothまたはローカルワイヤレスネットワークをアプリケーションで作成し、管理できます。複数のデバイス上で実行しているアプリケーションのコピーが互いを検出し、情報を交換することで、iOS上にマルチプレーヤーゲームを作成するためのシンプルかつパワフルな手段を提供します。さらにセッションは、ユーザ同士が互いに連携できるような刺激的な方法を、すべてのアプリケーションに提供します。

図 1　Bluetoothとローカルワイヤレスネットワーク



Bluetoothネットワークは、初期のiPhoneや第一世代のiPod touchではサポートされていません。また、iPhone Simulatorでもサポートされていません。

ピアツーピアアプリケーションを開発する場合、セッションで検出された他のユーザを表示する独自のユーザインターフェイスを実装するか、GKPeerPickerControllerオブジェクトを使用して2台のデバイス間にセッションを設定するための標準ユーザインターフェイスを提供することができます。

デバイス間にネットワークが確立されると、GKSessionクラスはネットワーク上で送受信されるデータのフォーマットには関知しません。アプリケーションにとって最適なデータフォーマットを自由に設計できます。

注： この文書では、ピアツーピア接続クラス群で提供されるインフラストラクチャについて説明します。ネットワークゲームや、アプリケーションの設計および実装については取り上げません。

ピアツーピア接続が必要な場合

アプリケーションにピアツーピア接続が必要な場合は、ピアツーピア接続対応のデバイスを持っているユーザだけがアプリケーションを購入してダウンロードできるようにする必要があります。ピアツーピアサポートが必要な場合は、アプリケーションのInfo.plistファイル内の必要なデバイス機能のリストにpeer-peerキーを追加します。詳細については、『*iOS Application Programming Guide*』の「iTunes Requirements」 in *iOS Application Programming Guide*を参照してください。

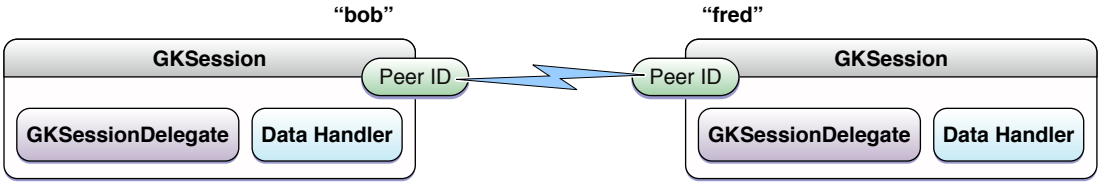
セッション

セッションは、作成されてからセッション同士が互いを検出すると1つのネットワークに接続されます。アプリケーションから他のデバイスにデータを送信するには、接続済みのセッションを使用します。アプリケーション側では、接続要求を処理するデリゲートと、他のデバイスからアプリケーションへ送られたデータを受信するデータハンドラを用意します。

ピア

アドホックなワイヤレスネットワークに接続されているiOSベースのデバイスを**ピア**と呼びます。ピアはアプリケーション内で実行しているセッションオブジェクトと同義です。各ピアが、ネットワーク上でほかのピアの識別に使用する一意のピア識別子文字列（ピアID）を作成しているかどうか確認する必要があります。ネットワーク上のほかのピアとのやり取りは、ピアIDを使用して行われます。たとえば、あるピアがほかのピアのIDを知っている場合、図 2に示すように、セッションオブジェクトの`displayNameForPeer:`メソッドを呼び出すことで、ユーザによる判読が可能なピア名を取得できます。

図 2　ほかのピアとのやり取りに使用されるピアID



ネットワーク上のほかのピアは、ローカルセッションに関連したさまざまな状態で表示されます。ピアはアドホックなネットワークに現れたり、消えたり、セッションに接続したり、セッションから切断したりします。アプリケーションでは、デリゲートの`session:peer:didChangeState:`メソッドを実装して、ピアの状態が変更された場合に通知されるようにします。

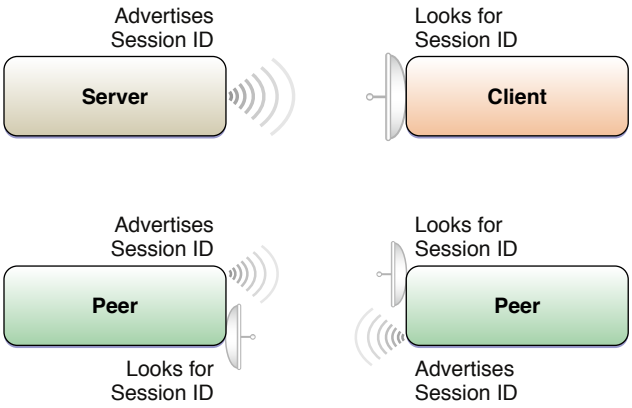
ほかのピアの検出

セッションはすべて、それぞれ独自のサービスタイプを実装しています。このサービスは特定のゲームであったり、名刺交換のような機能であったりします。デベロッパには、サービスタイプに必要なものや、ピア間での交換に必要なデータを決定する責任があります。

セッションは、セッションの初期化時に設定される**セッションモード**に基づいてネットワーク上のほかのピアを検出します。アプリケーションでは、セッションを**サーバ**（ネットワーク上にサービスタイプをアドバタイズする）、**クライアント**（ネットワーク上でアドバタイズしているサーバを検索する）、または**ピア**（サーバのようにアドバタイズすると同時にクライアントのように検索もする）に設定できます。図 3にセッションモードを示します。

サーバは、自身のサービスタイプを**セッション識別子**文字列（`sessionID`）によってアドバタイズします。クライアントは、一致するセッションIDを持つサーバのみを検索します。

図 3 サーバ、クライアント、およびピア



セッションIDは、登録済みのBonjourサービスの短縮名でなければなりません。Bonjourサービスの詳細については、Bonjour Networkingを参照してください。セッション作成時にセッションIDを指定していない場合、セッションはアプリケーションのバンドル識別子を使用してIDを生成します。

接続を確立するには、少なくとも1台のデバイスがサーバとしてアドバタイズを行い、別のデバイスがそれを検索する必要があります。アプリケーションは、アドバタイズと検索の両方にコードを提供します。これを実装するには、アドバタイズと検索を同時に行うピアがもっとも柔軟な方法です。ただし、双方がアドバタイズと検索を行うため、セッションが他のデバイスを検出するのにはより長い時間がかかります。

注： クライアント／サーバ型のゲームの最大規模は、プレーヤー16人です。

サーバの実装

サーバとして動作するアプリケーションのインスタンスは、セッションモードパラメータを`GKSessionModeServer`または`GKSessionModePeer`のいずれかに指定した`initWithSessionID:displayName:sessionMode:`を呼び出してセッションを初期化します。アプリケーションでセッションを設定した後、セッションの`available`プロパティを`YES`に設定してサービスをアドバタイズします。

サーバは、クライアントがサービスへの接続を要求すると通知を受け取ります。クライアントが接続要求を送信すると、サーバのデリゲートで`session:didReceiveConnectionRequestFromPeer:`メソッドが呼び出されます。デリゲートの典型的な動作は、`peerID`文字列を使用して、`displayNameForPeer:`メソッドを呼び出し、ユーザによる判読が可能な名前を取得します。次に、サーバは接続を受け入れるかどうかを決定するインターフェイスをユーザに表示します。

デリゲートは、セッションの`acceptConnectionFromPeer:error:`メソッドを呼び出して要求を受け入れるか、`denyConnectionFromPeer:`メソッドを呼び出して要求を拒否します。

接続が正常に作成されると、デリゲートの`session:peer:didChangeState:`メソッドが呼び出され、デリゲートに新しいピアが接続したことを知らせます。

サービスへの接続

クライアントとして動作するアプリケーションのインスタンスは、セッションモードパラメータを`GKSessionModeClient`または`GKSessionModePeer`のいずれかに指定した`initWithSessionID:displayName:sessionMode:`メソッドを呼び出してセッションを初期化します。セッションの設定後、アプリケーションは、セッションの`available`プロパティを`YES`に設定してアドバタイズしているサーバを探そうとネットワークを検索します。「サーバの実装」（75ページ）で説明したように、セッションが`GKSessionModePeer`モードに設定されている場合、サーバとしてアドバタイズも行います。

クライアントが利用可能なサーバを検出すると、クライアントセッションのデリゲートは`session:peer:didChangeState:`メソッドへの呼び出しを受信し、Game Kitは検出したサーバの`peerID`文字列を提供します。アプリケーションではセッションの`displayNameForPeer:`メソッドを呼び出して、ユーザによる判読が可能な名前を取得し、ユーザに表示することができます。ユーザが接続するピアを選択すると、アプリケーションはセッションの`connectToPeer:withTimeout:`メソッドを呼び出して接続要求を行います。

接続が正常に作成されると、デリゲートの`session:peer:didChangeState:`メソッドが呼び出され、アプリケーションに新しいピアが接続したことを知らせます。

データ交換

セッションに接続したピアは他の接続済みピアとデータを交換できます。アプリケーションは、`sendDataToAllPeers:withDataMode:error:`メソッドを呼び出すことですべての接続済みピアへ、または`sendData:toPeers:withDataMode:error:`メソッドを呼び出すことで一部のピアへデータを送信できます。ほかのピアに送信されるデータは、`NSData`オブジェクトにカプセル化されます。アプリケーションは、データに使用したい任意のデータフォーマットを設計し、使用できます。独自のデータフォーマットはアプリケーションで自由に作成できます。最高のパフォーマンスを引き出すには、データオブジェクトのサイズを小さく（長さ1000バイト以下に）することをお勧めします。1000バイトより大きいメッセージは、小さい固まりに分割して送信先で組み立てなおさなければならないことがあります。それにより遅延やオーバヘッドが増える可能性があります。

注： 送信可能な最大メッセージサイズは、87キロバイトです。これより大きなメッセージを送信する必要がある場合は、データを複数のメッセージに分割しなければなりません。

データを送信するには、送信先にデータが届かなかった場合にセッションがデータを再送信する**確実な**方法か、送信を1度しか行わない**確実性の低い**方法を選ぶことができます。確実性の低いメッセージは、データがほかのピアにとって有益なリアルタイムに到着しなければならない場合や、意図する受信者に届かなかったデータを再送するのではなく更新されたパケットを送信するほうが重要な場合（推測航法情報を送信する場合など）に適しています。

確実なメッセージは、送信側が送信した順序通りに参加者に受信されます。

ほかのピアから送信されたデータをアプリケーションで受信するには、オブジェクト上に`receiveData:fromPeer:inSession:context:`メソッドを実装する必要があります。アプリケーションでは`setDataReceiveHandler:withContext:`メソッドを呼び出してこのオブジェクトをセッションに提供します。接続済みのピアからデータを受信すると、アプリケーションのメインスレッド上にデータハンドラが呼び出されます。

重要： ほかのピアから受信したデータはすべて*信頼できない*データとして取り扱う必要があります。ほかのピアから受信したデータは必ず検証し、セキュリティの脆弱性を避けるために慎重にコードを書くよう気をつけてください。詳細については、『*Secure Coding Guide*』を参照してください。

ピアの切断

セッションを終了するには、`disconnectFromAllPeers`メソッドを呼び出します。

アプリケーションは`disconnectPeerFromAllPeers:`メソッドを呼び出して特定のピアを切断できます。

ネットワークは本質的に不確かなものです。ピアが一定期間応答しなくなると、そのピアは自動的にセッションから切断されます。アプリケーションでは、`disconnectTimeout`プロパティを変更して、セッションがピアを切断するまでの時間を制御できます。

セッションデリゲートの`session:peer:didChangeState:`メソッドは、ピアがセッションから切断されるときに呼び出されます。

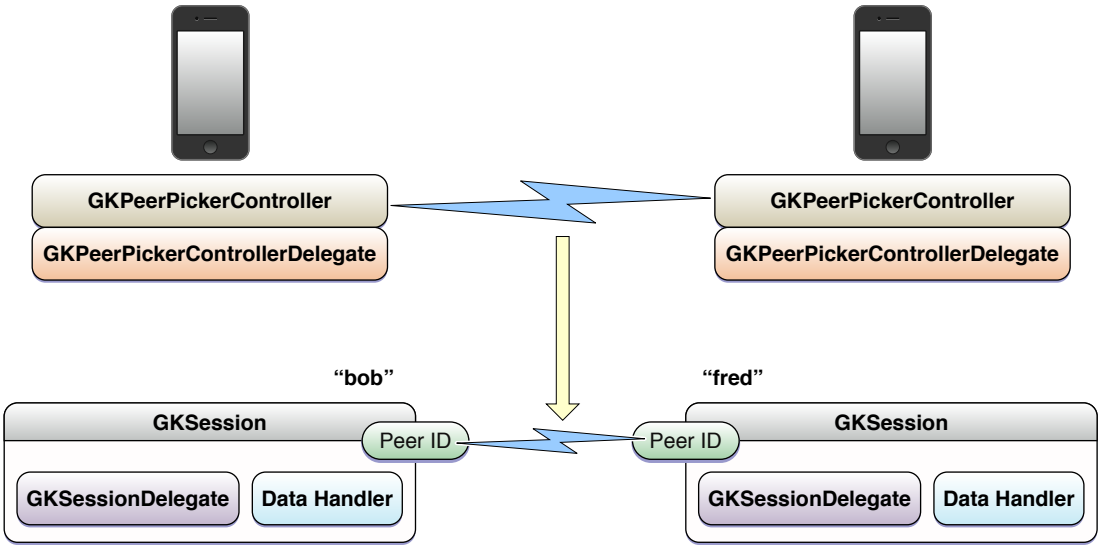
クリーンアップ

アプリケーションでセッションを廃棄する準備ができたら、アプリケーションはほかのピアから切断し、`available`フラグを`NO`に設定し、データハンドラおよびデリゲートを削除してセッションを解放します。

Peer Picker

`GKSession`のデリゲートを使用して独自のユーザインターフェイスを実装することもできますが、Game Kitは検出および接続処理の標準的なユーザインターフェイスを提供しています。`GKPeerPickerController`オブジェクトは、別のデバイスへのピアツーピア接続の作成を可能にするPeer Pickerユーザインターフェイスを提供しています。`GKPeerPickerController`オブジェクトは、2つのピアを接続する完全に設定された状態の`GKSession`を返します。図 4にPeer Pickerがどのように動作するかを示します。

図 4　ネットワーク上の2つのピアを接続するセッションを作成するPeer Picker



Peer Pickerコントローラの設定

アプリケーションでPeer Pickerとのユーザインターフェイスとしてコントローラが呼び出すデリゲートを提供します。

Peer Pickerコントローラの`connectionTypesMask`プロパティを使用して、アプリケーションがユーザに選択を許可する利用可能な接続メソッドのリストを設定します。ユーザは、ローカルのBluetoothネットワークかインターネットネットワークを選択できます。アプリケーションで複数のネットワーク形式を含むようにマスクを設定すると、Peer Pickerコントローラは、ユーザがどちらのネットワークを使用するかを選択するための追加のダイアログを表示します。ユーザがネットワークを選択すると、コントローラはデリゲートの`peerPickerController:didSelectConnectionType:`メソッドを呼び出します。

重要： Peer PickerはBluetooth接続のみ作成します。アプリケーションでインターネット接続を提供していて、ユーザがインターネット接続を選択した場合、アプリケーションはPeer Pickerをキャンセルし、インターネット接続を設定するための独自のユーザインターフェイスを提供する必要があります。

Peer Pickerによって作成されたセッションをアプリケーションでカスタマイズする場合は、デリゲートの`peerPickerController:sessionForConnectionType:`メソッドを実装することができます。アプリケーションでこのメソッドを実装しない場合、Peer Pickerはアプリケーション用にデフォルトセッションを作成します。

Peer Pickerの表示

アプリケーションは、コントローラの`show`メソッドを呼び出してPeer Pickerを表示します。ユーザがほかのピアに接続すると、デリゲートの`peerPickerController:didConnectPeer:toSession:`メソッドが呼び出されます。アプリケーションはセッションの所有権を取得し、コントローラの`dismiss`メソッドを呼び出してダイアログを非表示にします。

ユーザが接続の試みをキャンセルすると、Peer Pickerデリゲートの`peerPickerControllerDidCancel:`メソッドが呼び出されます。

Peer Pickerによるピアの検索

Peer Pickerは、Bluetooth経由で2人のユーザを接続するための標準ユーザインターフェイスを提供します。必要に応じてアプリケーションでは、インターネット接続かBluetooth接続かをユーザが選択できるようにPeer Pickerを設定できます。インターネット接続が選択された場合、アプリケーションはPeer Pickerのダイアログを閉じ、接続を完了させるための独自のユーザインターフェイスを表示する必要があります。

この章を読み終えたら「セッションの使用」（83 ページ）を読み、Peer Pickerで作成したセッションを使用してアプリケーションで実施できることを確認してください。

アプリケーションにPeer Pickerを追加するには、Peer Pickerコントローラのデリゲートメソッドを保持する新しいクラスを作成します。次の手順に従ってください。

1. `GKPeerPickerController`オブジェクトを作成し、初期化します。

```
picker = [[GKPeerPickerController alloc] init];
```

2. デリゲートをアタッチします（この手順を進める中でメソッドを定義します）。

```
picker.delegate = self;
```

3. 許可するネットワークタイプを設定します。

```
picker.connectionTypesMask = GKPeerPickerConnectionTypeNearby |
GKPeerPickerConnectionTypeOnline;
```

通常Peer Pickerのデフォルトは Bluetooth接続のみです。接続種別マスクにインターネット（オンライン）接続を追加することもできます。インターネット接続を追加する場合は、インターネット接続が選択された場合にダイアログを終了させるために`peerPickerController:didSelectConnectionType:`メソッドも実装する必要があります。

```
- (void)peerPickerController:(GKPeerPickerController *)picker
didSelectConnectionType:(GKPeerPickerConnectionType)type {
    if (type == GKPeerPickerConnectionTypeOnline) {
        picker.delegate = nil;
        [picker dismiss];
        [picker autorelease];
       // ここで独自のインターネットユーザインターフェイスを実装する
    }
}
```

4. （Peer Pickerコントローラによって作成されたデフォルトのオブジェクトではなく）カスタムのセッションオブジェクトを提供する場合は、デリゲートの`peerPickerController:sessionForConnectionType:`メソッドを実装します。

```
- (GKSession *)peerPickerController:(GKPeerPickerController *)picker
sessionForConnectionType:(GKPeerPickerConnectionType)type
{
    GKSession* session = [[GKSession alloc] initWithSessionID:myExampleSessionID
 displayName:myName sessionMode:GKSessionModePeer];
```

```
    [session autorelease];
    return session;
}
```

5. 設定済みセッションの所有権を受け取るためにデリゲートの`peerPickerController:didConnectPeer:toSession:`メソッドを実装します。

```
- (void)peerPickerController:(GKPeerPickerController *)picker
didConnectPeer:(NSString *)peerID toSession: (GKSession *) session {
// セッションの所有権を受け取るために保持しているプロパティを使用する
    self.gameSession = session;
// オブジェクトがセッションのデリゲートにもなると想定する
    session.delegate = self;
    [session setDataReceiveHandler: self withContext:nil];
// Pickerを削除する
    picker.delegate = nil;
    [picker dismiss];
    [picker autorelease];
// ゲームを開始する
}
```

6. ユーザがPickerをキャンセルした場合に対処できるように、`peerPickerControllerDidCancel:`メソッドを実装します。

```
- (void)peerPickerControllerDidCancel:(GKPeerPickerController *)picker
{
    picker.delegate = nil;
    // コントローラは自動的にダイアログを解除する
    [picker autorelease];
}
```

7. アプリケーションでPickerダイアログを表示するためのコードを追加します。

```
[picker show];
```

セッションの使用

この章ではPeer Pickerによって設定されたGKSessionオブジェクトの使用方法を説明します。Peer Pickerの設定方法の詳細については、「Peer Pickerによるピアの検索」（81 ページ）を参照してください。

ピアのセッションは、ほかのピアに関する情報および接続済みピアから送信されるデータの2種類のデータを受信します。アプリケーションは、デリゲートを提供してほかのピアに関する情報を受信し、データハンドラを提供してほかのピアからの情報を受信します。

アプリケーション内でセッションを使用するには、コードが「Peer Pickerによるピアの検索」（81 ページ）の手順に従っていることを確認してから、これらの手順を続けてください。

1. セッションデリゲートのsession:peer:didChangeState:メソッドを実装します。

   ほかのピアがセッションに対して状態を変化させると、セッションのデリゲートは通知を受けます。これらの状態変化のほとんどはPeer Pickerにより自動的に処理されますが、ユーザがネットワークに対して接続または切断した場合はアプリケーションで対応しなければなりません。

```
- (void)session:(GKSession *)session peer:(NSString *)peerID
didChangeState:(GKPeerConnectionState)state
{
    switch (state)
    {
        case GKPeerStateConnected:
// ほかのピアのpeerIDを記録する
// ピアが接続したことをゲームに通知する
        break;
        case GKPeerStateDisconnected:
// ピアがいなくなったことをゲームに通知する
        break;
    }
}
```

2. ほかのピアにデータを送信します。

```
- (void) mySendDataToPeers: (NSData *) data
{
    [session sendDataToAllPeers: data withDataMode: GKSendDataReliable error:
nil];
}
```

3. ほかのピアからデータを受け取ります。

```
- (void) receiveData:(NSData *)data fromPeer:(NSString *)peer inSession:
(GKSession *)session context:(void *)context
{
    // データのバイトを読み取り、アプリケーション固有のアクションを実行する
}
```

データはアプリケーションで即座に処理するか、または保持しておいて後でアプリケーション内で処理することができます。アプリケーションでは、このメソッド内での非常に長い計算は実行すべきではありません。

4. 接続を終了するときは、ほかのピアから切断してセッションオブジェクトを開放します。

```
[session disconnectFromAllPeers];
session.available = NO;
[session setDataReceiveHandler: nil withContext: nil];
session.delegate = nil;
[session release];
```

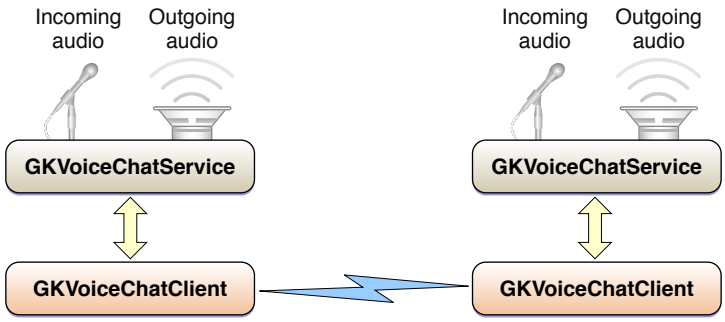
ゲーム内ボイス

パートIIIでは、アプリケーションでゲーム内ボイス機能を追加する方法を説明します。

ゲーム内ボイス

図 1に示すように、GKVoiceChatServiceオブジェクトによって、アプリケーションにおいて2台のiOSベースデバイス間のボイスチャット機能を簡単に作成できます。ボイスチャットサービスは、マイクの音声を受信したり、記録したデータをほかの参加者に送信したり、デバイスのオーディオサブシステムを使用してほかの参加者から受信したオーディオデータを再生したりします。ゲーム内ボイスは、GKVoiceChatClientプロトコルを実装したクライアントをアプリケーションが提供することを前提としています。クライアントの主な責任は、2人の参加者を接続し、ボイスチャットサービスが設定データを交換できるようにすることです。

図 1　ゲーム内ボイス



ボイスチャットの設定

参加者識別子

ボイスチャットのすべての参加者は、クライアントが提供する固有の**参加者識別子**文字列で識別されます。参加者識別文字列の形式と意味はクライアントの決定に任されます。

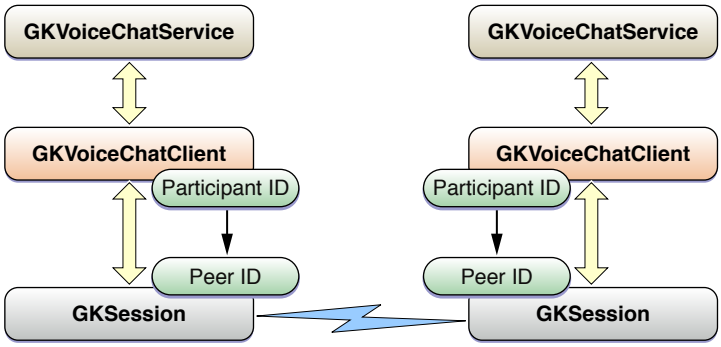
ほかの参加者の検出

ボイスチャットサービスは、クライアントのネットワーク接続を使用して参加者間の設定データを交換し、2台のデバイス間に直接の接続を作成します。ただし、ボイスチャットサービスはほかの参加者の参加者識別子を検出するためのメカニズムは提供しません。他のユーザの参加者識別子を提供し、これらの識別子をほかの参加者との接続へと変換するのはアプリケーションの責任です。

たとえば、GKSessionオブジェクトを通じて、1つのデバイスのアプリケーションが他のデバイスのアプリケーションにすでに接続されている場合（「ピアツーピア接続」　（73 ページ）を参照）、ネットワーク上のそれぞれのピアはすでにpeerID文字列により一意に識別されています。セッショ

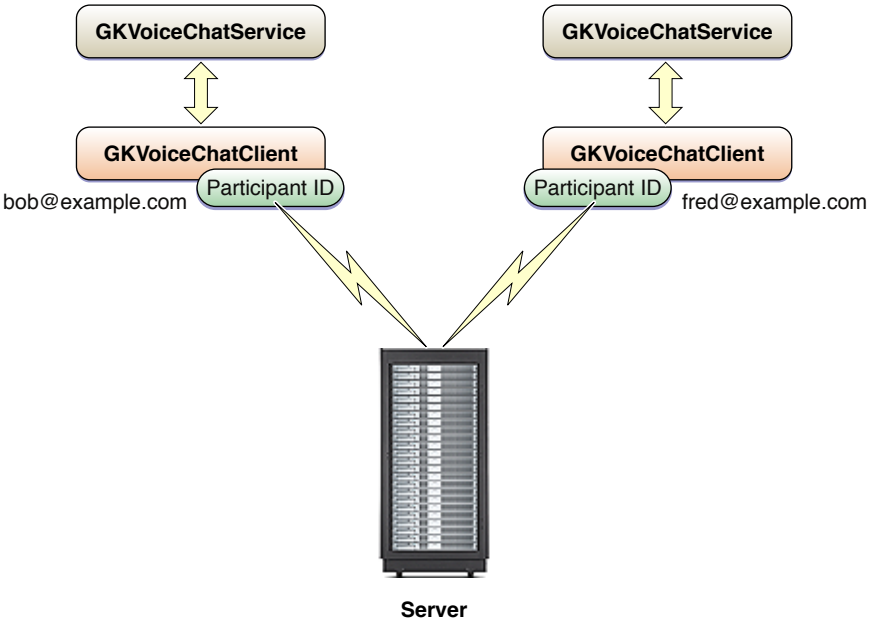
ンは、すでにほかの参加者のpeerID文字列を認識しています。図 2に示すように、クライアントはそれぞれのピアのIDを参加者識別子として再利用し、セッションを使用してデータの送受信を行うことができます。

図 2 ピアツーピアベースの検出



2台のデバイスが互いを直接認識していない場合、アプリケーションでは2人の参加者が互いを検出し接続を行うための他のサービスが必要です。図 3では、サーバが電子メールアドレスで参加者をそれぞれ識別し、二者間のデータを送信できます。

図 3 サーバベースの検出



サーバの設計によっては、サーバがクライアントに参加者識別子のリストを提供できたり、他のユーザの参加者識別子（電子メールアドレス）を提供するためにクライアントを必要とする場合があります。どちらの場合でも、サーバは2人のユーザ間のデータを転送する仲介者です。

1つのデバイスのボイスチャットサービスが設定データを他のデバイスの参加者に送信する場合、クライアントのvoiceChatService:sendData:toParticipantID:メソッドを呼び出します。クライアントはほかの参加者に確実かつ即座にデータを送信できる必要があります。他のクライアント

はデータを受信すると、サービスの`receivedData:fromParticipantID:`メソッドを呼び出してサービスにデータを転送します。ボイスチャットサービスはこのデータ交換を使用して、2人の参加者の間に独自のリアルタイムネットワーク接続を設定します。ボイスチャットサービスは、独自のチャット接続を作成するのに十分なデータを送信するためにのみ既存の接続を使用します。

リアルタイムデータ転送

場合によってはファイアウォールやNATベースのネットワークによって、ボイスチャットサービスが独自のネットワーク接続を確立するのが妨げられることがあります。アプリケーションはクライアント内にオプションのメソッドを実装して、参加者間のデータのリアルタイム転送を提供できます。クライアントで`voiceChatService:sendRealTimeData:toParticipantID:`メソッドを実装していると、ボイスチャットサービスが独自のリアルタイム接続を作成できない場合に、クライアントはフォールバックし、このメソッドを呼び出してデータを転送します。

チャットの開始

ボイスチャットを開始するには、参加者の1人がボイスチャットサービスの`startVoiceChatWithParticipantID:error:`メソッドをほかの参加者の`participantID`と共に呼び出してチャットを始めます。サービスは、「ほかの参加者の検出」（87 ページ）で説明したように、クライアントのネットワークを使用して新規のチャットを要求します。

デバイス上のボイスチャットサービスが接続要求を受信すると、要求を処理するためにクライアントの`voiceChatService:didReceiveInvitationFromParticipantID:callID:`メソッドが呼び出されます。クライアントはサービスの`acceptCallID:error:`メソッドを呼び出してチャット要求を受け入れるか、`denyCallID:`を呼び出して要求を拒否します。接続を受け入れるかどうかを、まずユーザに確認しなければならない場合があります。

一度接続が確立されて受け入れられると、クライアントは自身の`voiceChatService:didStartWithParticipantID:`メソッドの呼び出しを受け取ります。

ほかの参加者からの切断

アプリケーションは、サービスの`stopVoiceChatWithParticipantID:`メソッドを（いずれかのデバイス上で）呼び出してボイスチャットを終了します。他のユーザを利用できないことがわかった場合も、アプリケーションではチャットを停止する必要があります。

チャットの制御

2人の参加者がボイスチャットで接続されると、会話は2台のiOSベースデバイス間で自動的に転送されます。アプリケーションでサービスの`microphoneMuted`プロパティを設定することでローカルのマイクをミュートしたり、サービスの`remoteParticipantVolume`プロパティを設定することでリモートの参加者のボリュームを調整することができます。

アプリケーションはまた、接続の両端の音量レベルを監視できるようにすることも可能です。たとえば、これを使用して、参加者が話しているときにユーザインターフェイスにインジケータを設定するといったことができます。ローカルユーザに対しては、アプリケーションは

inputMeteringEnabledをYESに設定してメータを有効にし、inputMeterLevelプロパティを読み取ってユーザの音量レベルを監視します。同様に、アプリケーションではoutputMeteringEnabledをYESに設定し、outputMeterLevelプロパティを読み取ることで、他のユーザが受信した音量レベルを監視できます。アプリケーションのパフォーマンスを向上させるためには、測定を有効にするのは2人の参加者のメータレベルを読み取りたい時だけに限定するべきです。

ボイスチャットの追加

ボイスチャットを実装するには、ネットワークに接続された2つの異なるデバイスでアプリケーションを実行する必要があります。この例では、別のデバイスへのネットワークを作成するために`GKSession`オブジェクトを使用する、ボイスチャットの実装方法の1つを示します。`GKSession`オブジェクトの詳細については、「ピアツーピア接続」（73 ページ）を参照してください。

この例を実装するには、次の手順を実行します。

1. 再生と録音を行うためのオーディオセッションを設定します。

```
AVAudioSession *audioSession = [AVAudioSession sharedInstance];
[audioSession setCategory:AVAudioSessionCategoryPlayAndRecord error:myErr];
[audioSession setActive: YES error: myErr];
```

2. クライアントの`participantID`メソッドを実装します。

```
- (NSString *)participantID
{
    return session.peerID;
}
```

参加者識別子はクライアントを一意に識別する文字列です。セッションの`peerID`文字列がすでにピアを一意に識別しているため、ボイスチャットクライアントはそれを参加者識別子として再利用します。

3. クライアントの`voiceChatService:sendData:toParticipantID:`メソッドを実装します。

```
- (void)voiceChatService:(GKVoiceChatService *)voiceChatService sendData:(NSData
  *)data toParticipantID:(NSString *)participantID
{
    [session sendData: data toPeers:[NSArray arrayWithObject: participantID]
withDataMode: GKSendDataReliable error: nil];
}
```

ボイスチャットサービスは、チャットのほかの参加者にデータを送信する必要があるときに、クライアントを呼び出します。最も一般的には、サービスは、ほかの参加者との独自のリアルタイム接続を確立するためにクライアントを呼び出します。`GKSession`と`GKVoiceChatService`クラスはどちらも`NSData`オブジェクトを使用してメッセージデータを保持するため、ボイスチャットサービスから受信した`NSData`オブジェクトを単純にセッションに渡します。

両方のボイスチャットで独自の情報を送信するためにネットワークが使用されている場合、自身が送信するデータとボイスチャットサービスが送信するデータを区別できるように、セッション経由で送信するメッセージに追加情報を提供する必要があります。

4. セッションの受信ハンドラを実装してボイスチャットサービスにデータを転送します。

```
- (void) receiveData:(NSData *)data fromPeer:(NSString *)peer inSession:
(GKSession *)session context:(void *)context;
{
```

```
    [[GKVoiceChatService defaultVoiceChatService] receivedData:data
fromParticipantID:peer];
}
```

受信ハンドラ関数はクライアントの`voiceChatService:sendData:toParticipantID:`メソッドのミラーリングであり、セッションから受信したデータをボイスチャットサービスへ転送します。

5. クライアントをボイスチャットサービスに接続します。

```
MyChatClient *myClient = [[MyChatClient alloc] initWithSession: session];
[GKVoiceChatService defaultVoiceChatService].client = myClient;
```

6. ほかの参加者に接続します。

```
[[GKVoiceChatService defaultVoiceChatService] startVoiceChatWithParticipantID:
 otherPeer error: nil];
```

セッションの接続が確立された後にアプリケーションがこれを自動で行うか、またはユーザにボイスチャットを作成する機会が提供されます。ボイスチャットの自動作成に最も適しているのは、セッションデリゲートの`session:peer:didChangeState:`メソッド内です。

7. オプションのクライアントメソッドを実装します。

アプリケーションで他のユーザの確認にネットワーク接続を使わない場合、`GKVoiceChatClient`プロトコルの追加メソッドを実装する必要が生じることがあります。`GKVoiceChatClient`プロトコルは、ほかの参加者が接続を試みた場合や状態を変化させた場合に、クライアントに通知するための多くのメソッドを提供します。

書類の改訂履歴

この表は「*Game Kit プログラミングガイド*」の改訂履歴です。

日付	メモ
2011-03-08	ローカルプレーヤーの認証プロセスを改訂しました。Game Centerのさまざまな使用方法を明確にしました。
2010-10-25	サーバベースのマッチメークを行うための要件を明確にしました。Game Centerアプリケーションでボイスチャットを実装するためのガイドラインを改訂しました。
2010-08-27	Game Centerクラスの説明を追加しました。
2009-05-28	概念説明を追加して改訂しました。
2009-03-12	GameKitを使用してBluetooth上のローカルネットワークを実装する方法、および任意のネットワーク上のボイスチャットサービスを実装する方法について説明した新規文書。

書類の改訂履歴