Game Centerを使った通信対戦ゲーム

Cocoa勉強会 関西

2011年4月2日

Game Centerとは？

GameKitフレームワークが提供する
ネットワークサービス

Game Centerとは？

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 ― 新しく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 ― ゲームアプリケーションと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 ― ゲームアプリケーションをGame Center対応にするためのクラスを提 供します。

Game Centerとは？

■ ニックネーム

ニックネームを作成してほかのプレーヤー とやりできる。プレーヤーは、ステータスメッセージを設定したり、ほかのプレーヤーを 友だちとして登録可能。

■ 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る。

■ アチーブメント(Achievement、成績)

ゲームでのプレーヤーのアチーブメントを管理。アチーブメントはGame Centerに記録され、Game Centerアプリケーションとゲームアプ リケーションの中で閲覧可能。

■ マルチプレーヤー

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る。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る。ゲームを実行していな いときでも、対戦への招待可能。

Game Centerの重要な概念

■ すべてのGame Centerアプリケーションはプレーヤーの認証で 始まる

■ ゲームアプリケーション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

■ ほとんどのGame Centerクラスは非同期に動作する

■ ネットワークエラーの後にはGame Centerへのデータ送信を再 試行しなければならない

■ ネットワークエラー後にGame Centerからデータを取得すると キャッシュされたデータを取得できる

■ ブロック構文があたりまえのように使われている

Game Center対応アプリの作成

iTunes Connectでの設定

プロジェクトの設定

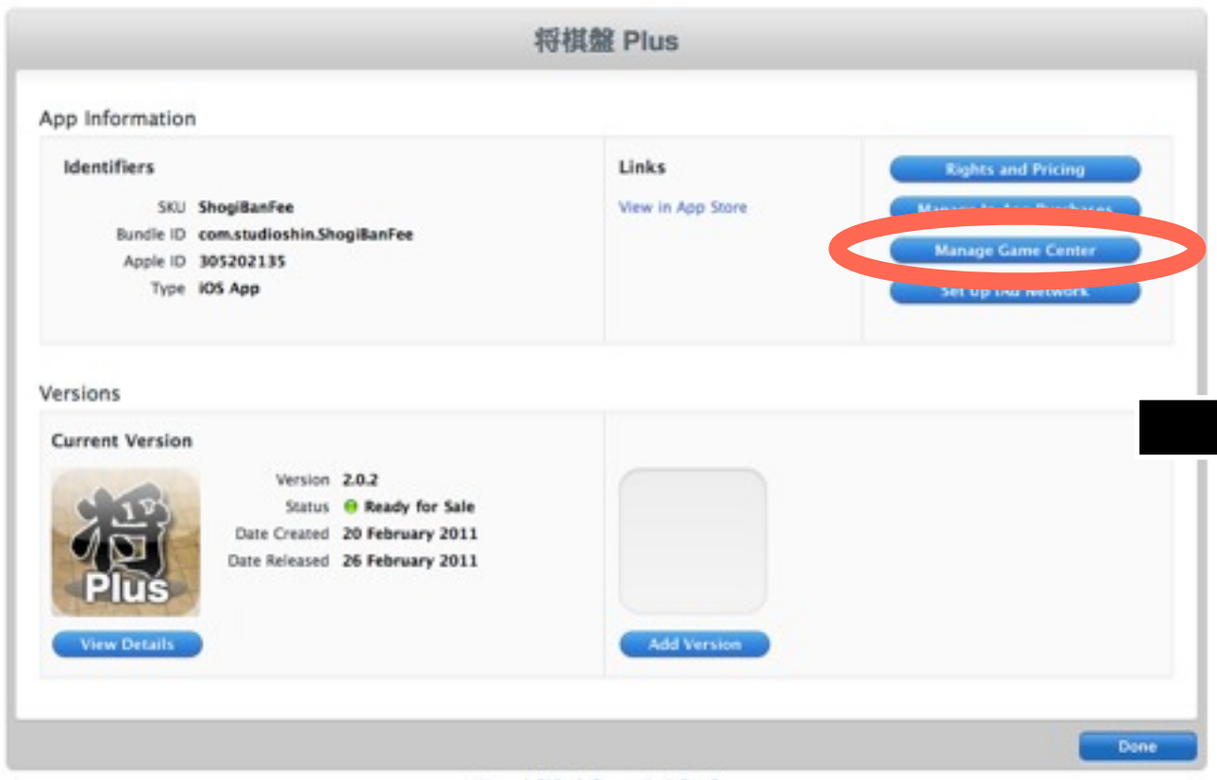
Game Centerでのプレーヤー

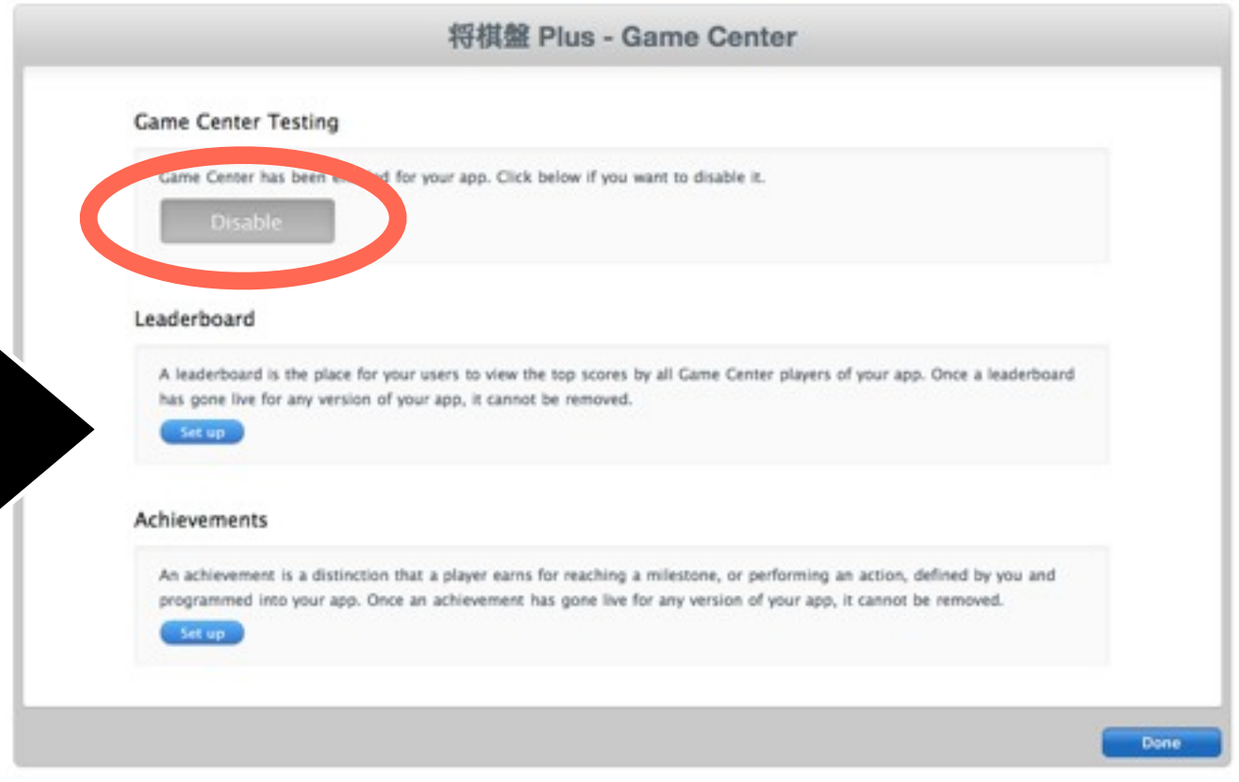
マッチメイク

iTunes Connectでの設定

アプリケーションを設定

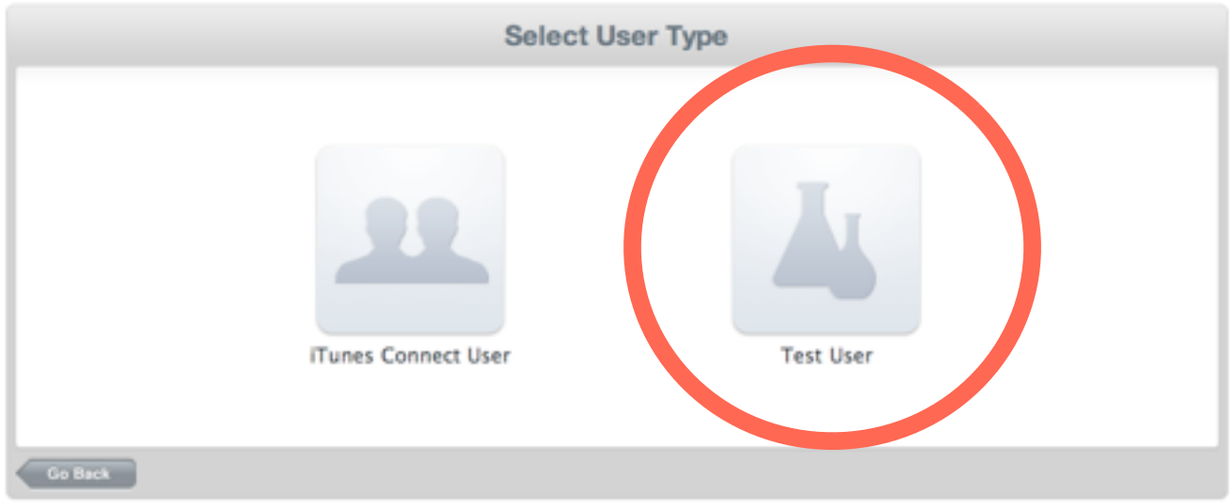
Game Centerコードをアプリケーションに追加する場合は、事前にiTunes Connectでアプリケーショ ンの設定を行い、Game Centerを使用できるようにする必要があります。
また、アプリケーションの Leaderboardやアチーブメント情報の設定もここで行います。





iTunes Connectでの設定

テストアカウントの作成



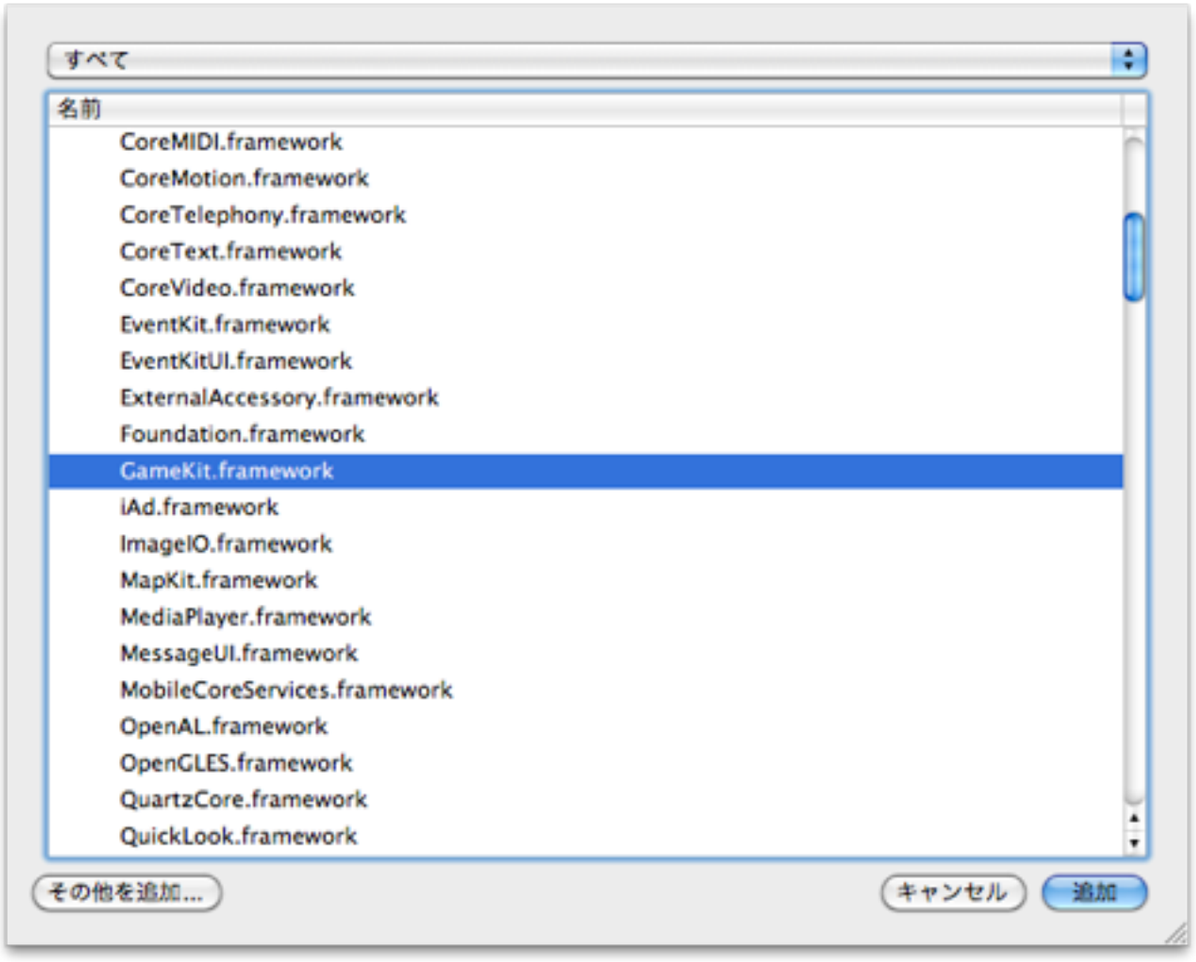
アプリケーション	対象	Game Center環境
シミュレータのビルド	デベロッパ	サンドボックス環境
デベロッパのビルド	デベロッパ	サンドボックス環境
アドホック配布用ビルド	ベータテスター	サンドボックス環境
署名付き配布用ビルド	エンドユーザ	ライブ環境

テストはなるべく実機で！

シミュレーターではマッチメークの招待を受けることはできない。

プロジェクトの設定

GameKitフレームワークをリンク



GameKit/GameKit.h をインポート

```
#import <GameKit/GameKit.h>
```

プロジェクトの設定

バンドルIDの設定

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サービス上の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。

Key	Value
▼ Information Property List	(13 items)
Localization native development re	English
Bundle display name	${PRODUCT_NAME}
Executable file	${EXECUTABLE_NAME}
Icon file	
Bundle identifier	com.yourcompany.${PRODUCT_NAME:rfc1034identifier}
InfoDictionary version	6.0
Bundle name	${PRODUCT_NAME}
Bundle OS Type code	APPL
Bundle creator OS Type code	????
Bundle version	1.0
Application requires iPhone enviror	☑
Main nib file base name	MainWindow
▶ Supported interface orientations	(4 items)

プロジェクトの設定

アプリケーションでGame Centerが必須の場合

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する。それにはアプリケーションのInfo.plistの UIRequiredDeviceCapabilities に gamekitキー を追加

▼ UIRequiredDeviceCapabilities	Array	(1 item)
Item 0	String	gamekit

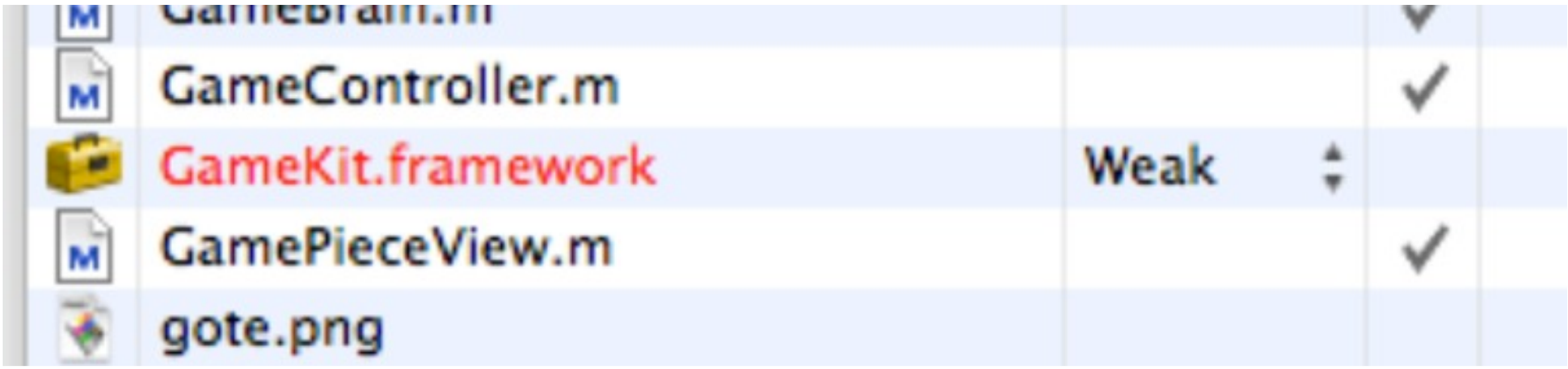
```
<key>UIRequiredDeviceCapabilities</key>
<array>
    <string>gamekit</string>
</array>
```

プロジェクトの設定

アプリケーションでGame Centerが必須では無い場合

Game Kitフレームワークへのリンクはweakに設定。

プロジェクトのターゲットを選択...



プロジェクトの設定

アプリケーションでGame Centerが必須では無い場合

アプリケーションの起動時にGame Centerがサポートされてい るかどうかをテストする。

```
-(BOOL)checkGKSupported
{
   UIDevice   *device = [UIDevice currentDevice];
   systemVer = [[device systemVersion] floatValue];
   Class gcClass = (NSClassFromString(@"GKLocalPlayer"));
   // デバイスはiOS 4.1以降で動作していなければならない
   if( gcClass && systemVer >= 4.1 ){
       NSLog(@"Game Centerはサポートされている");
       isGameCenterSupported = YES;
   }
   else {
       NSLog(@"Game Centerはサポートされていない !!");
       isGameCenterSupported = NO;
   }

   return  isGameCenterSupported;
}
```

Game Centerでのプレーヤー

プレイヤーを表すクラス

GKPlayer

```
@interface GKPlayer : NSObject

// Load the players for the identifiers provided. Error will be nil on success.
// Possible reasons for error:
// 1. Unauthenticated local player
// 2. Communications failure
// 3. Invalid player identifier
+ (void)loadPlayersForIdentifiers:(NSArray *)identifiers
            withCompletionHandler:(void(^)(NSArray *players, NSError *error))
completionHandler;

@property(nonatomic, readonly, retain)  NSString *playerID;  // プレイヤー識別子
@property(nonatomic, readonly, copy)    NSString *alias;     // プレイヤーのニックネーム
@property(nonatomic, readonly)          BOOL isFriend;       // ローカルプレイヤーのともだちか？

@end

// プレイヤーの詳細変更を通知するNotification
GK_EXTERN NSString *GKPlayerDidChangeNotificationName;
```

Game Centerでのプレーヤー

ローカルプレイヤーを表すクラス

GKLocalPlayer

```
@interface GKLocalPlayer : GKPlayer {
}

+ (GKLocalPlayer *)localPlayer;

@property(nonatomic, readonly, getter=isAuthenticated)  BOOL authenticated; // 認証状態
@property(nonatomic, readonly, getter=isUnderage)       BOOL underage;      // 未成年者

// ローカルプレイヤー認証の確認を行う
- (void)authenticateWithCompletionHandler:(void(^)(NSError *error))completionHandler;

// ローカルプレイヤーの友人の識別子の配列
@property(nonatomic, readonly, retain) NSArray *friends;

// 非同期的にプレーヤーの識別子の配列としてフレンドリストをロード
- (void)loadFriendsWithCompletionHandler:(void(^)(NSArray *friends, NSError *error))
completionHandler;

@end

// 認証状態の変更を通知するNotification
GK_EXTERN NSString *GKPlayerAuthenticationDidChangeNotificationName;
```

Game Centerでのプレーヤー

プレーヤーがGame Centerにアクセスするにはアカウントが必要。
Game Centerはアカウントごとに**プレーヤー識別子（文字列）**を割り当て、この識別 子によってアカウントを識別する。

プレーヤー識別子の中身について想定をしてはいけない！！
プレーヤー識別子の形式と長 さは変更されることがある。

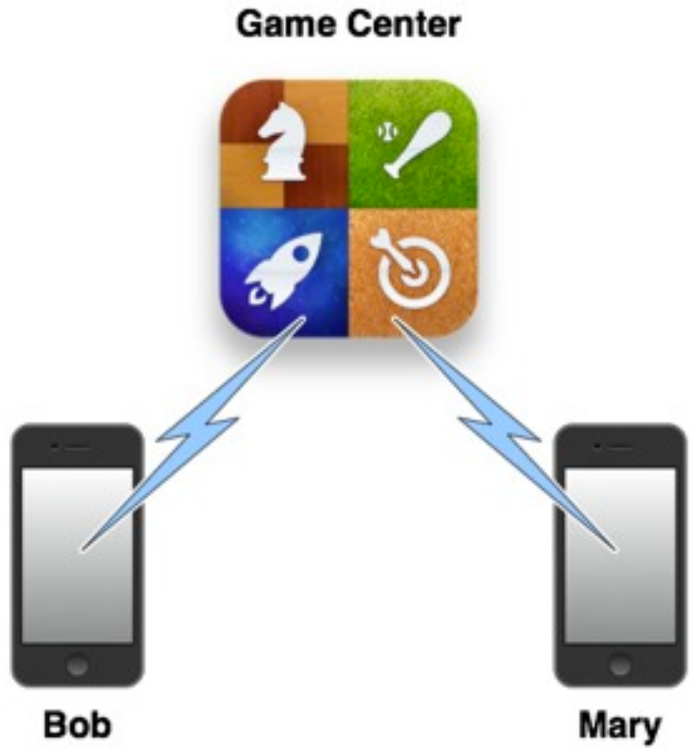
プレーヤー識別子をユーザに表示してはいけない！！
プレーヤーはすべて各自でニックネームを選択できプレーヤー識別子が 与えられると、Game Centerからプレーヤーのニックネームを取得してそれをプレーヤーに表示できる。

Game Centerでのプレーヤー

ローカルプレイヤー

デバイス上でプレーすることが現在認証されているプレーヤー

GKLocalPlayer （GKPlayerのサブクラス）



認証済みのローカルプレーヤーが存在しない場合は、アプリケーションは Game Center クラスを使用してはいけない！！

Game Centerでのプレーヤー

ローカルプレイヤーの認証

Game Center対応のアプリケーションを起動したら、できるだけ早くローカルプレーヤーを認証する（UIをプレーヤーに表示したらすぐに認証を行うのが理想）。

```
-(void)checkGKPLauthenticate {

    [[GKLocalPlayer localPlayer] authenticateWithCompletionHandler:^(NSError *error) {
        if(error==nil){
            //ローカルプレーヤーの認証後の処理を実装する（ 招待ハンドラの設定など）
            {

                ........
            }
        }
        else {
            //ローカルプレーヤーの未認証後の処理を実装する
            {

                ........
            }
        }
    }];
}
```

Game Centerでのプレーヤー

ローカルプレイヤーの認証済みの場合に行うこと

■ 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 クネームを取得する。

■ ローカルプレーヤーの友だちのプレーヤー識別子のリストを取得する。

■ 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する。

■ 対戦要求を受信するために招待ハンドラを追加する。
マッチメーク機能を取り入れる場合はプレーヤーの認証が終了したら直ちにこの手順を実行しなければならない（アプリケーショ ンは招待を処理するために起動されている可能性があるため）

Game Centerでのプレーヤー

ローカルプレーヤーがGame Centerから切断したことを検出する通知の登録

■ バックグラウンドに移動した場合Game Centerアカウントをログアウトできる。

■ 別のプレーヤーがログインしている場合もあり得る。

すべての Game Centerアプリケーションは、ローカルプレーヤーがGame Centerとの接続を切ったことを知らせる通知ハンドラを登録しなければならない。

Game Centerでのプレーヤー

ローカルプレーヤーがGame Centerから切断したことを検出する通知の登録

以下のNotificationを起動直後などに登録

```
//ローカルプレイヤーの承認ステータス変更通知ハンドラ通知の登録
NSNotificationCenter *nc = [NSNotificationCenter defaultCenter];

[nc addObserver:self selector:@selector(authenticationChangedNotification)
          name:GKPlayerAuthenticationDidChangeNotificationName
        object:nil];
```

```
- (void) authenticationChangedNotification {
  // 認証に成功した場合の処理コードをここに挿入する
  if ([GKLocalPlayer localPlayer].isAuthenticated){
      .........
  }
  // 未処理のGame Center関連クラスをクリーンアップするコードをここに挿入する
  else {
      .........
  }
}
```

Game Centerでのプレーヤー

プレーヤーの詳細情報の取得

■ ローカルプレーヤーの友だちの識別子の取得

```
- (void) retrieveFriends {

    GKLocalPlayer *lp = [GKLocalPlayer localPlayer];
    // 認証済み
    if ( lp.authenticated ){
        [lp loadFriendsWithCompletionHandler:^(NSArray *friends, NSError *error) {
            // プレーヤー識別子を使用して、プレーヤーオブジェクトを作成する
            if(error==nil){
                // ローカルプレーヤーの友だちの取得
                [self loadPlayerData:friends];
            }
            // ユーザにエラーを報告する
            else {

            }
        }];
    }
}
```

Game Centerでのプレーヤー

プレーヤーの詳細情報の取得

■ プレーヤーの情報取得

```
- (void) loadPlayerData:(NSArray *) identifiers {

    [GKPlayer loadPlayersForIdentifiers:identifiers withCompletionHandler:
                                ^(NSArray *players, NSError *error) {
        // エラー処理
        if (error != nil) {
            .........
        }
        // GKPlayerオブジェクトの配列を処理する
        if (players != nil){
            // プレーヤーオブジェクトのコレクションの取得
            for (int i=0; i < [players count]; i++) {
                GKPlayer* ply = [players objectAtIndex:i];
                NSLog(@"playerID = %@", ply.playerID);
                NSLog(@"alias    = %@", ply.alias);
                NSLog(@"isFriend = %d", ply.isFriend);
            }
        }
    }];
}
```

マルチプレイヤー

■ オンラインでプレーヤーを互いに発 見して1つのゲームにつなぐ

■ 自分が用意したサーバにプレーヤー を接続することも可能（かなり高度！！）

■ オートマッチと招待が可能

▼オートマッチ

対戦要求を作成し、Game Centerがプレーヤーを見つけてゲームに追加。

▼招待

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る。招待された友だちはゲームを立ち上げて対戦に参加できるプッシュ通知を 受信。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 ヤーに送信できる。その場合プレーヤーはその通知から直接App Storeを起動できる。

マッチメークは同じアプリケーション(同じバンドルIDのアプリケーション)間でのみ実行て可能。
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ない。

マルチプレイヤー

対戦開始までの流れ

必須

Game Centerの有効確認

↓

ローカルプレイヤー認証

↓

マッチメイクハンドラ登録

↓

ゲーム作成

↓

ゲーム確立、開始

＜任意＞
自分のニックネーム/ID取得
友人リスト取得(ニックネーム/ID)

マルチプレイヤー

対戦要求を行うクラス

GKMatchRequest

```
@interface GKMatchRequest : NSObject {
}

@property(nonatomic, assign) NSUInteger minPlayers;      // 最小のプレイヤー数
@property(nonatomic, assign) NSUInteger maxPlayers;      // 最大のプレイヤー数
@property(nonatomic, assign) NSUInteger playerGroup;     // プレイヤーグループの識別子
@property(nonatomic, assign) uint32_t playerAttributes; // オプションフラグ
@property(nonatomic, retain) NSArray *playersToInvite;  // プレイヤー識別子の配列

@end
```

iOS4.1では、minPlayersは最低2、maxPlayersは4以下でなければならない。

ホストサーバー経由の対戦であれば最大16人までのプレーヤーが許されている。

マルチプレイヤー

オートマッチ

■ 新規対戦要求の作成

```
-(void)startGKMatch {

    GKMatchRequest*        req = [[[GKMatchRequest alloc] init] autorelease];
    req.minPlayers = 2;  //最小プレイヤー数

    req.maxPlayers = 2;  //最大プレイヤー数

    //マッチメイクを行うビューコントローラーをモーダルビューとして開く

    GKMatchmakerViewController*matchview = [[[GKMatchmakerViewController alloc]
                                                    initWithMatchRequest:req]];
    matchview.matchmakerDelegate = self;  //デリゲート設定

    [self presentModalViewController:matchview animated:YES];
    [matchview release];
}
```

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する必要がある。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する。

GKMatchmakerViewController

マルチプレイヤー

GKMatchmakerViewControllerDelegate

■ ユーザーが対戦要求をキャンセルした

```
- (void)matchmakerViewControllerWasCancelled:(GKMatchmakerViewController *)viewController
{
    [self dismissModalViewControllerAnimated:YES]; //モーダルビュー削除
    //キャンセル後の必要な処理を実装する…
    {
        ……
    }
}
```

■ エラーが発生した

```
- (void)matchmakerViewController:(GKMatchmakerViewController *)viewController
                 didFailWithError:(NSError *)error
{
    [self dismissModalViewControllerAnimated:YES]; //モーダルビュー削除
    //エラー後の必要な処理を実装する…
    {
        ……
    }
}
```

マルチプレイヤー

GKMatchmakerViewControllerDelegate

■ 対戦要求が確立した

```
- (void)matchmakerViewController:(GKMatchmakerViewController *)viewController
                    didFindMatch:(GKMatch *)match
{
    [self dismissModalViewControllerAnimated:YES]; //モーダルビュー削除

    if( [match expectedPlayerCount]==0 ){
        // 全てのプレイヤーとの接続確立。 対戦を開始する
        self.match = match; // GKMatchオブジェクトを使用して対戦する
        ........
    }
}
```

対戦が作成されて各プレーヤーの開始準備ができたらView Controllerは、GKMatchオブジェクトをデリゲートに返す。保存用のプロパティなどを用意してマッチオブジェク トを代入してから、それを使用して対戦を開始する。

マルチプレイヤー

プログラム内でオートマッチを行う

```
-(void)findProgrammaticMatchStart {

   GKMatchRequest*      req = [[[GKMatchRequest alloc] init] autorelease];
   req.minPlayers = 2;  //最小プレイヤー数
   req.maxPlayers = 2;  //最大プレイヤー数

   [[GKMatchmaker sharedMatchmaker] findMatchForRequest:request
                           withCompletionHandler:^(GKMatch *match, NSError *error) {
         // エラー処理
         if(error){
             ........
         }
         // 対戦を開始する
         else if( [match expectedPlayerCount]==0 ) {
             self.match = match;
             ........
         }
   }];
}
```

マルチプレイヤー

通信対戦を行うクラス

GKMatch

```
@interface GKMatch : NSObject {
}

@property(nonatomic, readonly) NSArray *playerIDs;                  // プレイヤー識別子

@property(nonatomic, assign) id<GKMatchDelegate> delegate;
@property(nonatomic, readonly) NSUInteger expectedPlayerCount; // 未接続のプレイヤー数

// 非同期で指定のプレイヤーに送信する
- (BOOL)sendData:(NSData *)data toPlayers:(NSArray *)playerIDs
                            withDataMode:(GKMatchSendDataMode)mode
                                   error:(NSError **)error;

// 非同期で全てのプレイヤーに送信する
- (BOOL)sendDataToAllPlayers:(NSData *)data
                withDataMode:(GKMatchSendDataMode)mode
                       error:(NSError **)error;

// ゲームを中断する 。GKMatchのインスタンスを解放前に呼ぶ。
- (void)disconnect;

// ボイスチャット
- (GKVoiceChat *)voiceChatWithName:(NSString *)name;

@end
```

マルチプレイヤー

GKMatDelegate

```
@protocol GKMatchDelegate <NSObject>

@required
// プレイヤーからのデータを受信する
- (void)match:(GKMatch *)match
                      didReceiveData:(NSData *)data
                          fromPlayer:(NSString *)playerID;

@optional
// プレイヤーのステータスが変わった
- (void)match:(GKMatch *)match
                                player:(NSString *)playerID
                           didChangeState:(GKPlayerConnectionState)state;


// エラーにより通信不可となった
- (void)match:(GKMatch *)match
                connectionWithPlayerFailed:(NSString *)playerID
                                 withError:(NSError *)error;

// エラーのためゲームが開始できない
- (void)match:(GKMatch *)match didFailWithError:(NSError *)error;

@end
```

マルチプレイヤー

招待

■ 招待ハンドラの追加（プレイヤーの認証直後に行う）

```
-(void)matchMakeChmaker {

    [GKMatchmaker sharedMatchmaker].inviteHandler = ^(GKInvite *acceptedInvite, NSArray *playersToInvite) {
        GKMatchmakerViewController* matchview=nil;

        // どちらか１つのパラメータが nil 以外になっている

        if(acceptedInvite){
            // 別のプレーヤーから直接招待を受け取った。相手がすでに対戦要求を作成している

            matchview = [[[GKMatchmakerViewController alloc] initWithInvite:acceptedInvite]
                                                                        autorelease];
        }
        else if(playersToInvite){
            // Game Centerアプリから起動された。 playersToInviteはゲームに招待するプレイヤー識別子の配列

            GKMatchRequest*    req = [[[GKMatchRequest alloc] init] autorelease];
            req.minPlayers = 2;    //最小プレイヤー数

            req.maxPlayers = 2;    //最大プレイヤー数

            req.playersToInvite = playersToInvite;
            matchview = [[[GKMatchmakerViewController alloc] initWithMatchRequest:req] autorelease];
        }
        if(matchview){
            //マッチメイクを行うビューコントローラーをモーダルビューとして開く

            matchview.matchmakerDelegate = self;
            [self presentModalViewController:matchview animated:YES];
        }
    };
}
```