Turn-based Matches with Game Center

About Nick

- iOS developer since early 2007
- iOS consulting at Fresh App Factory
- @n_dubbs
- https://github.com/ndubbs/TickTackToe

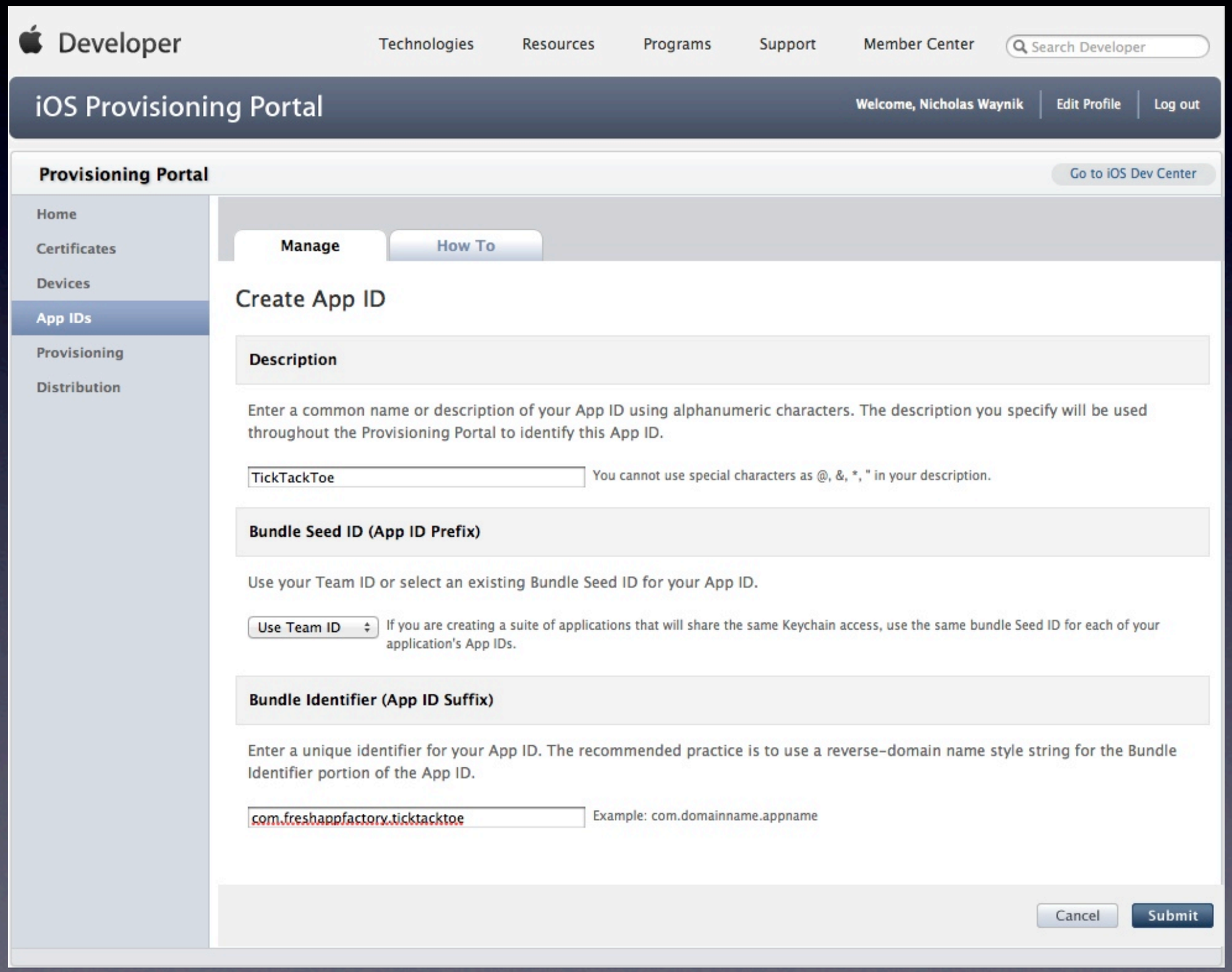
Word By Word


iPod
10:23 PM
nw played ANBETEN
Pass
Swap
Return
Resign
nw 47
n_dubbs 35
55 Tiles

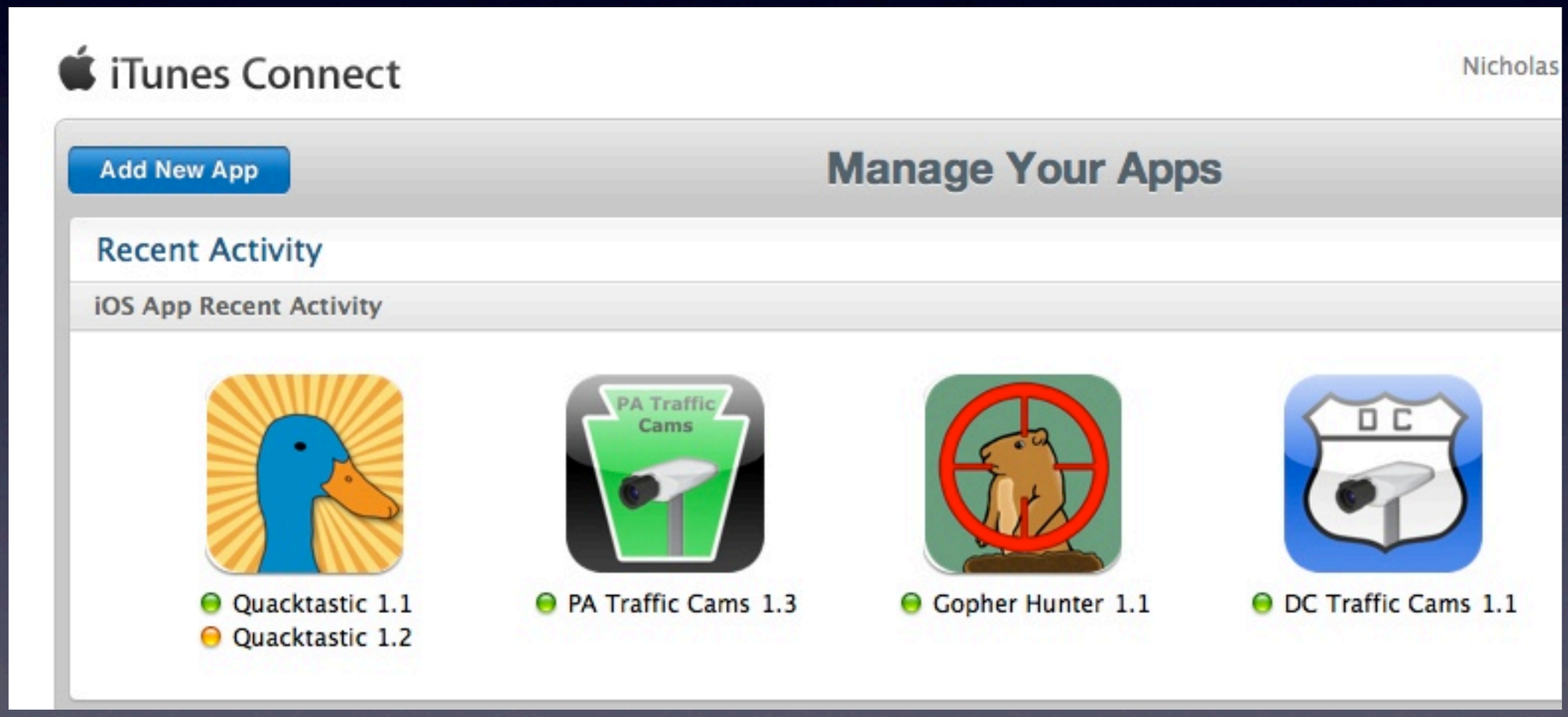
Overview

- Setup app in Dev Portal / iTunes Connect
- Tick Tack Toe game review
- Delegate protocol review
- GKTurnBasedMatch methods
- Wrap up of game

iOS Dev Portal



iTunes Connect



iTunes Connect

Nicholas Waynik, Nicholas Waynik Sign Out

App Information

Enter the following information about your app.

Default Language: English

App Name: Tick Tack Toe: Cocoaheads Edition

SKU Number: ticktacktoe00001

Bundle ID: TickTackToe - com.freshappfactory.ticktacktoe

You can register a new Bundle ID here.

Note that the Bundle ID cannot be changed if the first version of your app has been approved or if you have enabled Game Center or the iAd Network.

Does your app have specific device requirements? Learn more

Cancel Continue

iTunes Connect

Nicholas Waynik, Nicholas Waynik Sign Out

Tick Tack Toe: Cocoaheads Edition

Select the availability date and price tier for your app.

Availability Date 04/Apr 17 2012

Price Tier Free

View Pricing Matrix ▶

Discount for Educational Institutions

Custom B2B App

Unless you select specific stores, your app will be for sale in all App Stores worldwide.

Go Back Continue

Home | FAQs | Contact Us | Sign Out

iTunes Connect

Nicholas Waynik, Nicholas Waynik Sign Out

Tick Tack Toe: Cocoaheads Edition

Enter the following information in **English**.

Version Information

Version Number	1.0
Copyright	Fresh App Factory
Primary Category	Games
Subcategory	Select
Subcategory	Select
Secondary Category (optional)	Select
Review Notes (optional)	

iTunes Connect

Nicholas Waynik, Nicholas Waynik Sign Out

Tick Tack Toe: Cocoaheads Edition

App Information Edit

Identifiers

SKU **ticktacktoe00001**

Bundle ID **com.freshappfactory.ticktacktoe**

Apple ID **520202568**

Type **iOS App**

Default Language **English**

Links

View in App Store

Rights and Pricing

Manage In-App Purchases

Manage Game Center

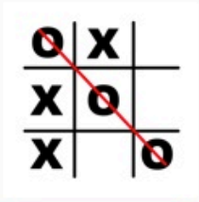
Set Up iAd Network

Newsstand

Delete App

Versions

Current Version



Version **1.0**

Status **Prepare for Upload**

Date Created **Apr 17, 2012**

View Details

Done

iTunes Connect

Nicholas Waynik, Nicholas Waynik Sign Out

Tick Tack Toe: Cocoaheads Edition - Game Center

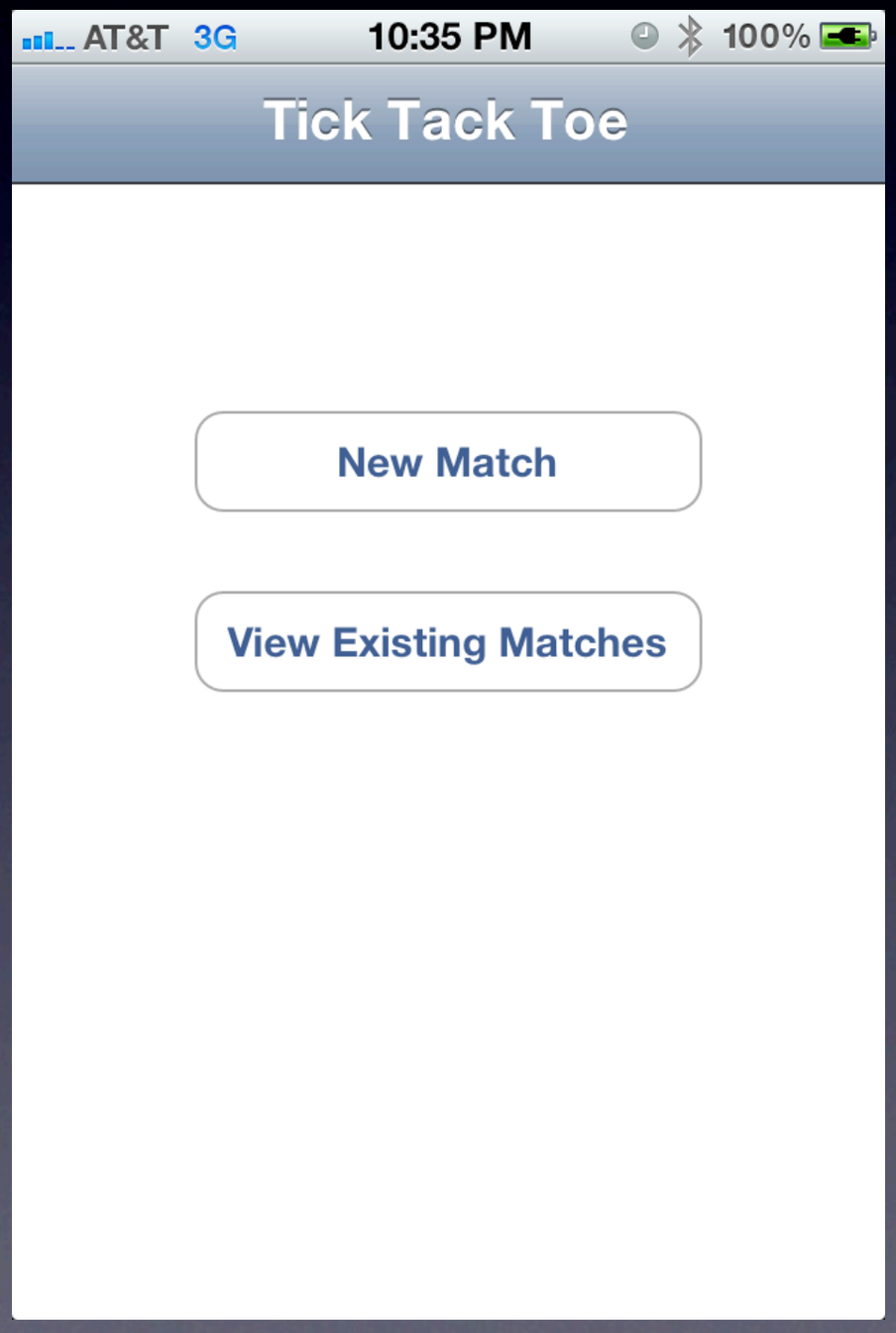
Game Center

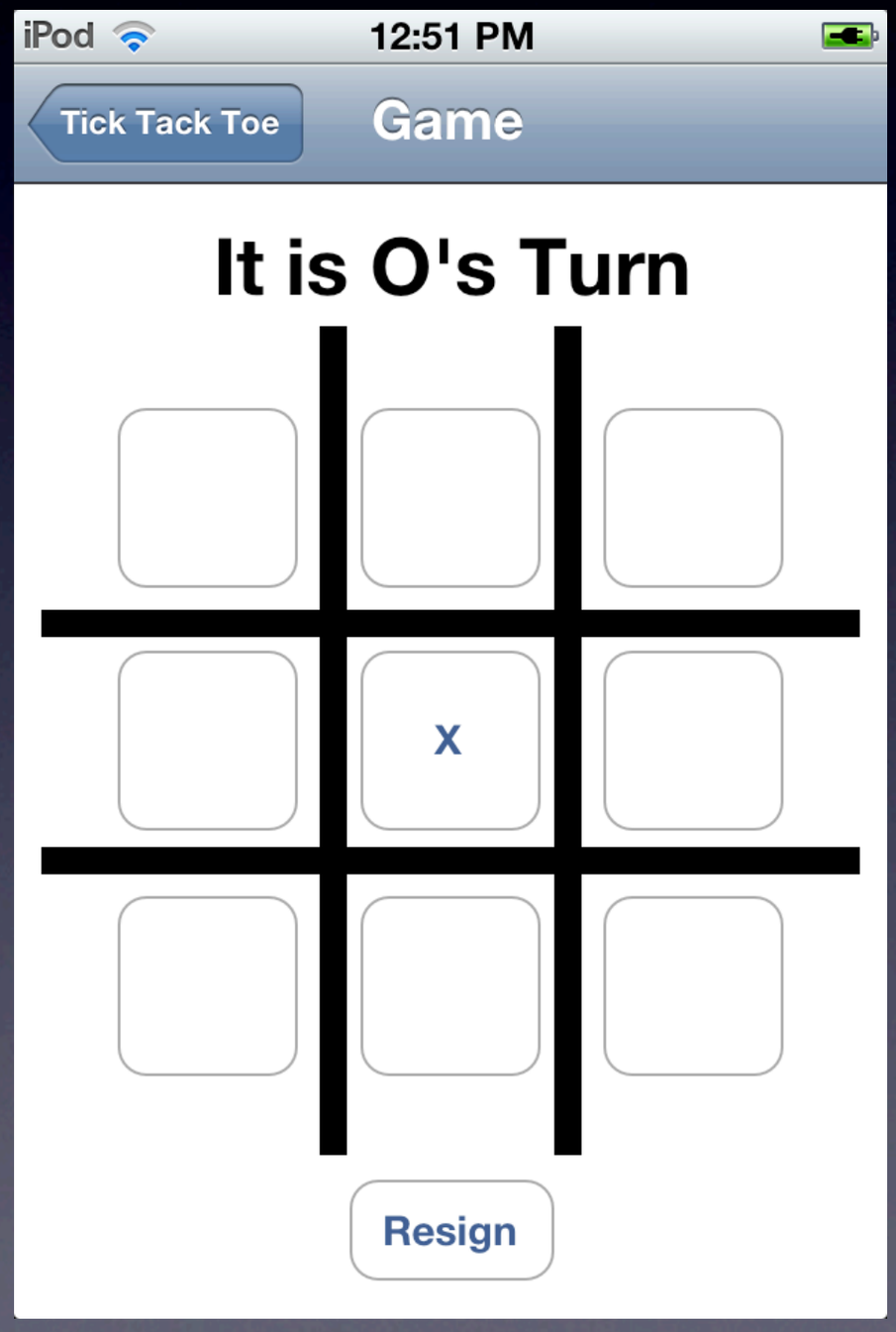
In order for your app to be viewed in Game Center, you must have used GameKit to include the capability in your binary. To set up Game Center for your app, click Enable, below.

Enable

Done

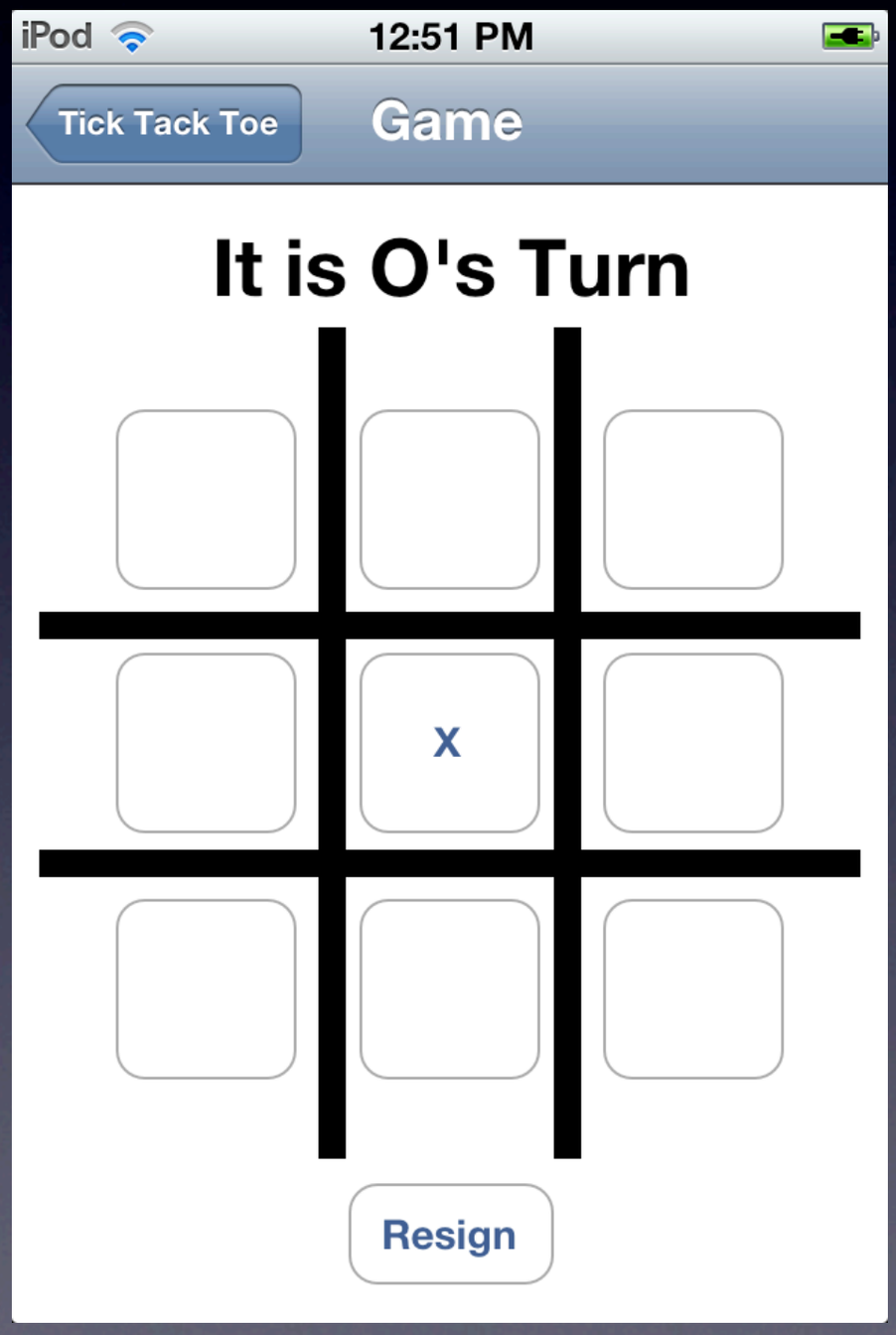
Tick Tack Toe

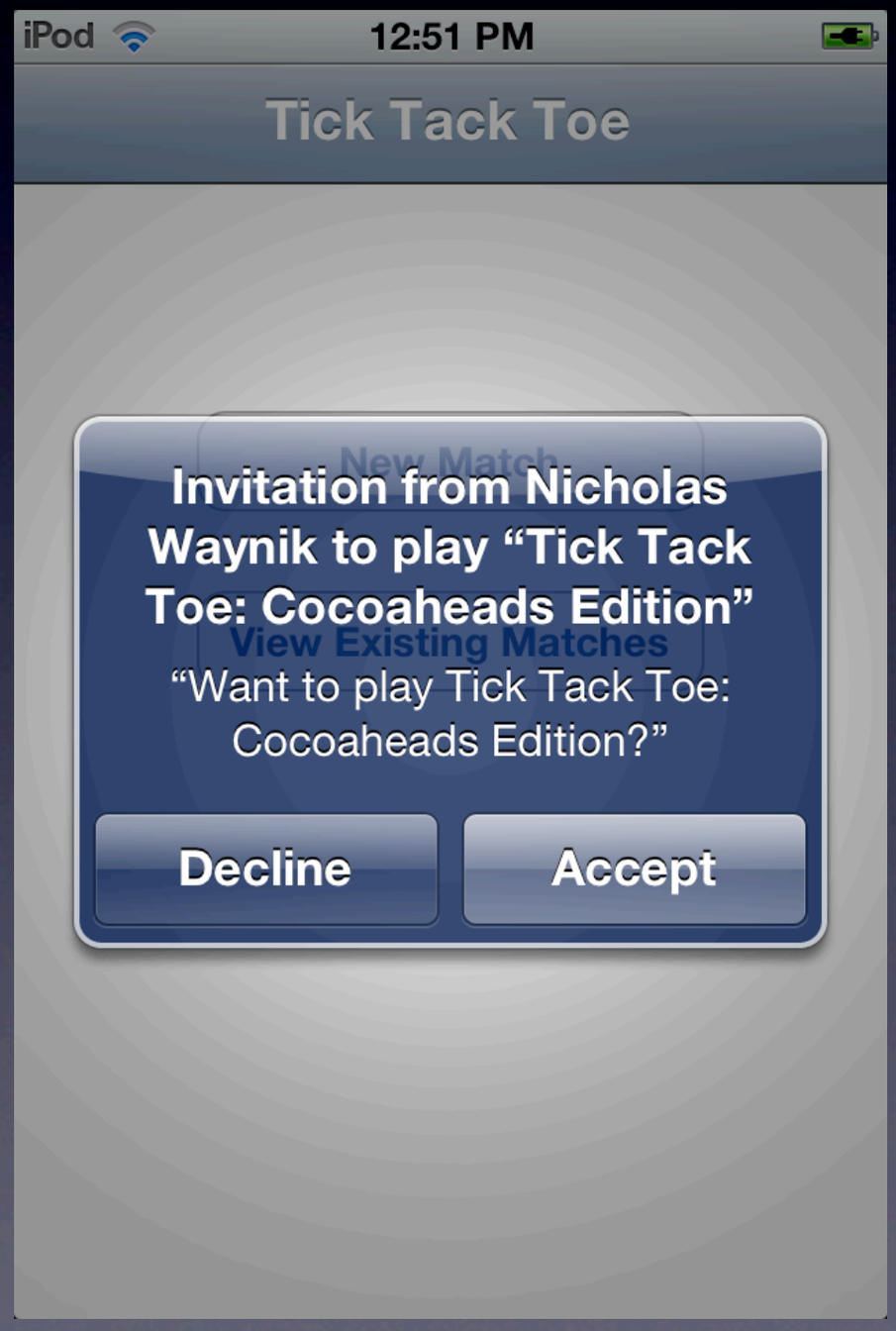




Matchmaker View Controller

Invitations and Game Start





Delegate Protocol

- A delegate is an object that's given an opportunity to react to changes in another object or influence the behavior of another object.

GCHelper.h

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

@class GCHelper;

@protocol GCMenuHelperDelegate <NSObject>

@optional

- (void)startGCGameWithMatch:(GKTurnBasedMatch *)theMatch;

@end



@protocol GCHelperDelegate <NSObject>

- (void)turnSubmissionDidSucceed:(GKTurnBasedMatch *)theMatch;
- (void)turnSubmissionDidFail:(GKTurnBasedMatch *)theMatch;
- (void)startTurnWithMatch:(GKTurnBasedMatch *)theMatch;
- (void)matchDidEnd:(GKTurnBasedMatch *)theMatch;
- (void)resignDidSucceed:(GKTurnBasedMatch *)theMatch;
- (void)resignDidFail:(GKTurnBasedMatch *)theMatch;
- (void)endGameForDecline:(GKTurnBasedMatch *)theMatch;
- (void)endGame:(GKTurnBasedMatch *)theMatch forResign:(BOOL)didResign;

@end
```

GCHelper.h

```objc
@interface GCHelper : NSObject <GKTurnBasedMatchmakerViewControllerDelegate,GKTurnBasedEventHandlerDelegate>

@property (assign, readonly) BOOL gameCenterAvailable;
@property (assign, readonly) BOOL userAuthenticated;
@property (nonatomic, assign) BOOL isGameVisible;
@property (nonatomic, strong) GKTurnBasedMatch *currentMatch;
@property (nonatomic, strong) GKLocalPlayer *localPlayer;
@property (retain) UIViewController *presentingViewController;
@property (assign) id <GCHelperDelegate> delegate;
@property (assign) id <GCMenuHelperDelegate> menuDelegate;
```

MenuViewController

```objc
#import <UIKit/UIKit.h>
#import "DetailViewController.h"
#import "GCHelper.h"

@interface MenuViewController : UIViewController <GCMenuHelperDelegate>
```

```objc
- (void)viewDidLoad
{
    [super viewDidLoad];

    [self setGcHelper:[GCHelper sharedInstance]];
    [self.gcHelper setMenuDelegate:self];
    [self.gcHelper authenticateLocalUser];
}
```

GKTurnBasedMatch

- creationDate
- currentParticipant
- matchID
- message
- participants
- status
- matchData

GKTurnBasedMatch ViewController

```objc
- (void)showMatchmakerViewControllerWithMinPlayers:(int)minPlayers maxPlayers:(int)maxPlayers viewController:(UIViewController *)viewController
    showExistingMatches:(BOOL)showMatches
{
    if (!gameCenterAvailable) return;

    self.presentingViewController = viewController;
    GKMatchRequest *matchRequest = [[GKMatchRequest alloc] init];
    matchRequest.minPlayers = minPlayers;
    matchRequest.maxPlayers = maxPlayers;

    GKTurnBasedMatchmakerViewController *mmvc = [[GKTurnBasedMatchmakerViewController alloc] initWithMatchRequest:matchRequest];
    mmvc.turnBasedMatchmakerDelegate = self;

    mmvc.showExistingMatches = showMatches;

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

Delegate Methods

```objc
#pragma mark - GKTurnBasedMatchmakerViewControllerDelegate

// The user has cancelled matchmaking
- (void)turnBasedMatchmakerViewControllerWasCancelled:(GKTurnBasedMatchmakerViewController *)viewController
{
    [self.presentingViewController dismissViewControllerAnimated:YES completion:nil];
}

// Matchmaking has failed with an error
- (void)turnBasedMatchmakerViewController:(GKTurnBasedMatchmakerViewController *)viewController didFailWithError:(NSError *)error
{
    //handle error
    [self.presentingViewController dismissViewControllerAnimated:YES completion:nil];

    UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"Error" message:@"The Internet connection appears to be offline." delegate:self
        cancelButtonTitle:@"OK" otherButtonTitles:nil];
    [alert show];
}
```

Quitting a Match via View Controller

```objc
// Quiting a match from the GKTurnBasedMatchmakerViewController
- (void)turnBasedMatchmakerViewController:(GKTurnBasedMatchmakerViewController *)viewController playerQuitForMatch:(GKTurnBasedMatch *)theMatch
{
    // Check to see if local player is current participant
    GKTurnBasedParticipant *nextParticipant = [self getNextParticipantWithMatch:theMatch];
    NSString *currentPlayerID = theMatch.currentParticipant.playerID;
    if ([currentPlayerID isEqualToString:[GKLocalPlayer localPlayer].playerID]) {

        [theMatch participantQuitInTurnWithOutcome:GKTurnBasedMatchOutcomeQuit nextParticipant:nextParticipant matchData:theMatch.matchData
            completionHandler:^(NSError*error){

        }];
    } else {
        [theMatch participantQuitOutOfTurnWithOutcome:GKTurnBasedMatchOutcomeQuit withCompletionHandler:^(NSError*error){

        }];
    }
}
```

Selecting a Match

```objc
// A match has been found, the game should start
- (void)turnBasedMatchmakerViewController:(GKTurnBasedMatchmakerViewController *)viewController didFindMatch:(GKTurnBasedMatch *)theMatch
{
    self.presentingViewController = viewController;
    [self.presentingViewController dismissViewControllerAnimated:YES completion:nil];

    self.currentMatch = theMatch;
    [self.menuDelegate startGCGameWithMatch:theMatch];
}
```

```objc
#pragma mark - GCMenuHelperDelegate

- (void)startGCGameWithMatch:(GKTurnBasedMatch *)theMatch
{
    DetailViewController *gameViewController = [[DetailViewController alloc] init];
    [gameViewController setMatch:theMatch];
    [self.navigationController pushViewController:gameViewController animated:YES];
}
```

GKTurnBasedEvent HandlerDelegate

- handleInviteFromGameCenter:
- handleTurnEventForMatch:
- handleMatchEnded:

The Turn Event

```objc
- (void)handleTurnEventForMatch:(GKTurnBasedMatch *)theMatch
{
    //Sent to the delegate when it is the local player's turn to act in a turn-based match.

    /*
     When your delegate receives this message, the player has accepted a push notification for
     a match already in progress. Your game should end whatever task it was performing and
     switch to the match information provided by the match object. For more information on
     handling player actions in a turn-based match, see GKTurnBasedMatch Class Reference.
     */

    if (self.isGameVisible && [theMatch.matchID isEqualToString:currentMatch.matchID]) {
        self.currentMatch = theMatch;

        if ([self opponentDidEndMatch:theMatch]) {
            // This case is for when the other player quit the match in turn
            [self submitEndForMatch:theMatch withData:theMatch.matchData winner:YES endType:1 withMessage:@"Game Over"];
            return;

        } else if (theMatch.currentParticipant.status == GKTurnBasedParticipantStatusDeclined) {

            // Other player declined the game invite
            for (GKTurnBasedParticipant *participant in theMatch.participants) {
                if (participant.status == GKTurnBasedParticipantStatusDeclined) {
                    [self endMatchForDecline:theMatch];
                    return;
                }
            }
        }

        if ([theMatch.currentParticipant.playerID isEqualToString:[GKLocalPlayer localPlayer].playerID]) {
            // Update UI and allow turn submission
            [delegate startTurnWithMatch:theMatch];
        }
    }
}
```

Match Ended Event

```objc
- (void)handleMatchEnded:(GKTurnBasedMatch *)theMatch
{
    //Sent to the delegate when a match the local player is participating in has ended.

    /*
     When your delegate receives this message, it should display the match's final results to
     the player and allow the player the option of saving or removing the match data from
     Game Center.
     */

    if (self.isGameVisible && [theMatch.matchID isEqualToString:currentMatch.matchID]) {
        currentMatch = theMatch;
        BOOL isResign = NO;

        // Opponent ended the current match
        for (GKTurnBasedParticipant *participant in theMatch.participants) {
            if (participant.matchOutcome == GKTurnBasedMatchOutcomeQuit) {
                isResign = YES;
                break;
            }
        }

        if (isResign) {
            [delegate endGame:theMatch forResign:YES];
        } else {
            [delegate endGame:theMatch forResign:NO];
        }
    }
}
```

```objc
- (void)endGame:(GKTurnBasedMatch *)theMatch forResign:(BOOL)didResign
{
    [self setButtonEnabled:NO];
    [self.resignButton setEnabled:NO];

    if (didResign) {
        UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"Opponent Resigned" message:@"Your opponent resigned from the match!"
            delegate:nil cancelButtonTitle:@"OK" otherButtonTitles:nil, nil];
        [alert show];
    } else {

        NSString *matchOutcome = [self checkWinner];

        if ([matchOutcome isEqualToString:@"Tie"]) {
            UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"Game Over" message:@"Its a draw" delegate:nil cancelButtonTitle:
                @"Dimiss" otherButtonTitles: nil];
            [alert show];
        } else {
            UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"Game Over" message:theMatch.message delegate:nil cancelButtonTitle:
                @"Dimiss" otherButtonTitles: nil];
            [alert show];
        }
    }
}
```

Submitting a Turn

```
- (void)submitTurnForMatch:(GKTurnBasedMatch *)theMatch withData:(NSData *)turnData withMessage:(NSString *)turnMessage
{
    if(theMatch != nil) {

        GKTurnBasedParticipant *nextParticipant = [self getNextParticipantWithMatch:theMatch];
        theMatch.message = turnMessage;
        [theMatch endTurnWithNextParticipant:nextParticipant matchData:turnData completionHandler:^(NSError *error){
            if(error == nil) {
                dispatch_async(dispatch_get_main_queue(), ^{
                    [delegate turnSubmissionDidSucceed:theMatch];
                });
            } else {
                dispatch_async(dispatch_get_main_queue(), ^{
                    [delegate turnSubmissionDidFail:theMatch];
                });
            }
        }];

    }
}
```

Ending the Match

```objc
- (void)submitEndForMatch:(GKTurnBasedMatch *)theMatch withData:(NSData *)turnData winner:(BOOL)localWon endType:(int)endType withMessage:
    (NSString *)turnMessage
{
    // Set the participants match outcome status
    for (GKTurnBasedParticipant *participant in theMatch.participants) {
        if (endType == 1) {
            // Won/Lost end type
            if ([participant.playerID isEqualToString:[self localPlayer].playerID]) {
                if (localWon) {
                    participant.matchOutcome = GKTurnBasedMatchOutcomeWon;
                } else {
                    participant.matchOutcome = GKTurnBasedMatchOutcomeLost;
                }
            } else {
                if (localWon) {
                    participant.matchOutcome = GKTurnBasedMatchOutcomeLost;
                } else {
                    participant.matchOutcome = GKTurnBasedMatchOutcomeWon;
                }
            }
        } else {
            // Draw end type
            participant.matchOutcome = GKTurnBasedMatchOutcomeTied;
        }
    }

    // Set the match message
    theMatch.message = turnMessage;

    [theMatch endMatchInTurnWithMatchData:turnData completionHandler:^(NSError *error) {
        if(error == nil) {
            dispatch_async(dispatch_get_main_queue(), ^{

                [delegate matchDidEnd:theMatch];

            });
        } else {
            dispatch_async(dispatch_get_main_queue(), ^{
                [delegate turnSubmissionDidFail:theMatch];
            });
        }
    }];
}
```

Quitting the Match Out of Turn

```objc
- (void)quitMatchOutOfTurn:(GKTurnBasedMatch *)theMatch
{
    for (GKTurnBasedParticipant *participant in theMatch.participants) {
        if ([participant.playerID isEqualToString:[GKLocalPlayer localPlayer].playerID]) {
            participant.matchOutcome = GKTurnBasedMatchOutcomeQuit;
        } else {
            participant.matchOutcome = GKTurnBasedMatchOutcomeWon;
        }
    }

    [theMatch participantQuitOutOfTurnWithOutcome:GKTurnBasedMatchOutcomeQuit withCompletionHandler:^(NSError *error) {
        if (error == nil) {
            [delegate resignDidSucceed:theMatch];
        } else {
            dispatch_async(dispatch_get_main_queue(), ^{
                [delegate resignDidFail:theMatch];
            });
        }
    }];
}
```

Summary

- Setup app in iTunes Connect
- Tick Tack Toe game review
- Delegate protocol review
- GKTurnBasedMatch methods
- Wrap up of game

References

- http://developer.apple.com/library/ios/#documentation/GameKit/Reference/GKTurnBasedMatch_Ref/Reference/Reference.html
- http://developer.apple.com/library/ios/#documentation/GameKit/Reference/GKTurnBasedEventHandlerDelegate_Ref/Reference/Reference.html

References

- http://developer.apple.com/library/ios/#documentation/GameKit/Reference/GKTurnBasedMatchmakerViewController_Ref/Reference/Reference.html#//apple_ref/occ/cl/GKTurnBasedMatchmakerViewController
- Beginning iOS Game Center and Game Kit http://www.apress.com/9781430235279
- Cocoa Design Patterns http://my.safaribooksonline.com/book/programming/cocoa/9780321591210