mgwuSDK Instructions

Table of Contents:

**(Mandatory elements are in bold)**

1. Setting up mgwuSDK

If there are any features you wish to see in the our SDK that aren't there, or things you think can be done better, please drop me a line (ashu@makegameswith.us). We're building this for you, and we want to include anything that could make your life easier!

Additionally, make sure you read the instructions for each part very closely, certain parts have intricacies that are important to understand.

If there is anything you are unclear about, drop me a line (ashu@makegameswith.us)

First create your app on our server at this url: https://dev.makegameswith.us/createapp

Go to the summary of your app:
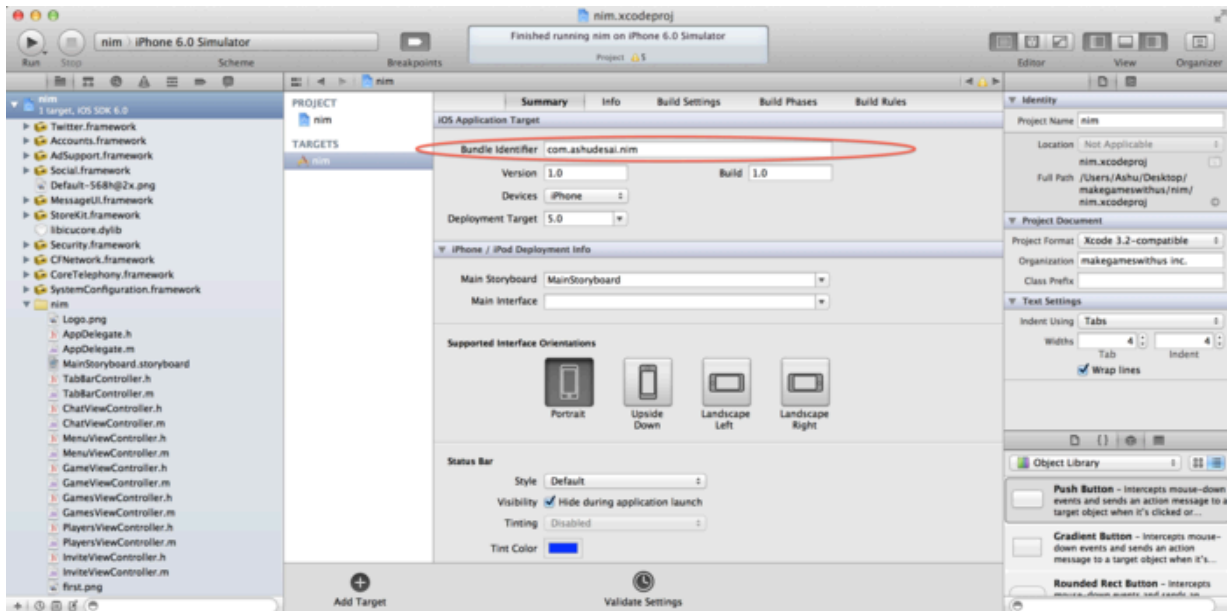
Change the Bundle identifier to "com.xxx.xxx" to match the app you created on the server. Delete the project from any simulator / device you've been testing on. On the top bar go to Product -> Clean. This should ensure you are using the new bundle identifier.

Scroll down on the summary page to where it says Linked Frameworks and Libraries
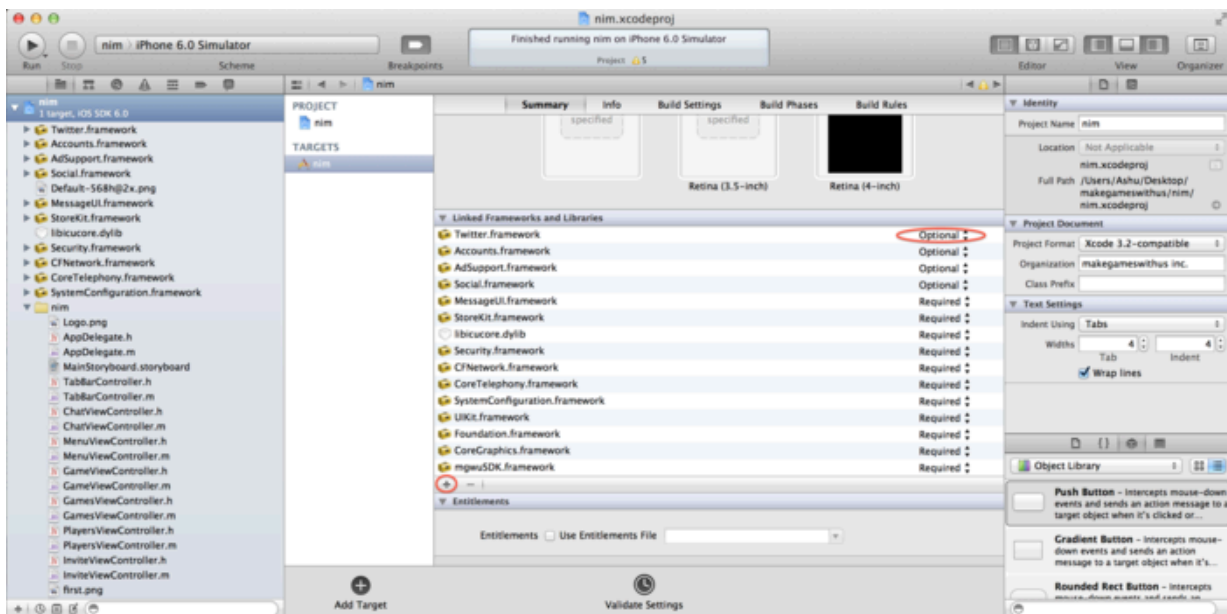
Add the following frameworks:

- Accounts.framework
- AdSupport.framework
- CFNetwork.framework
- CoreTelephony.framework

- libicucore.dylib
- MessageUI.framework
- Security.framework
- Social.framework
- StoreKit.framework
- SystemConfiguration.framework
- Twitter.framework

On the right of each framework there is a selector to let you pick between required and optional. For Accounts, AdSupport, Social and Twitter change it to optional.

Then go into BuildSettings. On the top left, click on all, and combined. On the top right, search "Other Linker Flags". Double click to the right of "Other Linker Flags", and in the bottom left of the popup hit the + button. Add "-lsqlite3.0".
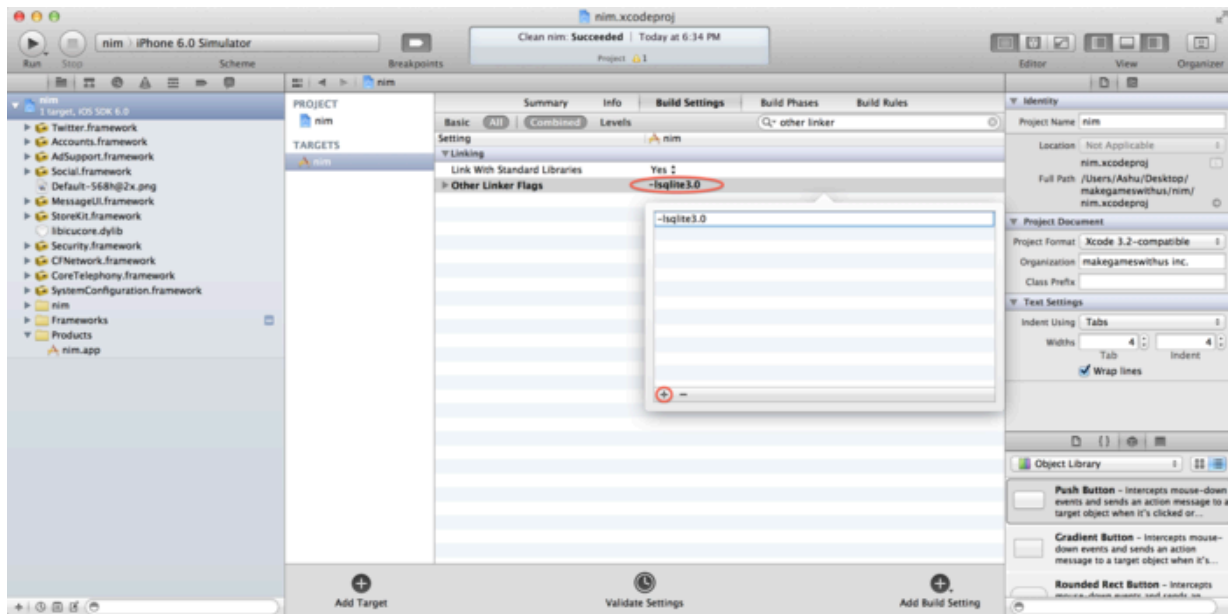
Drag the mgwuSDK.embeddedframework folder into the frameworks folder in your project, make sure you select "copy files and add them to target"!
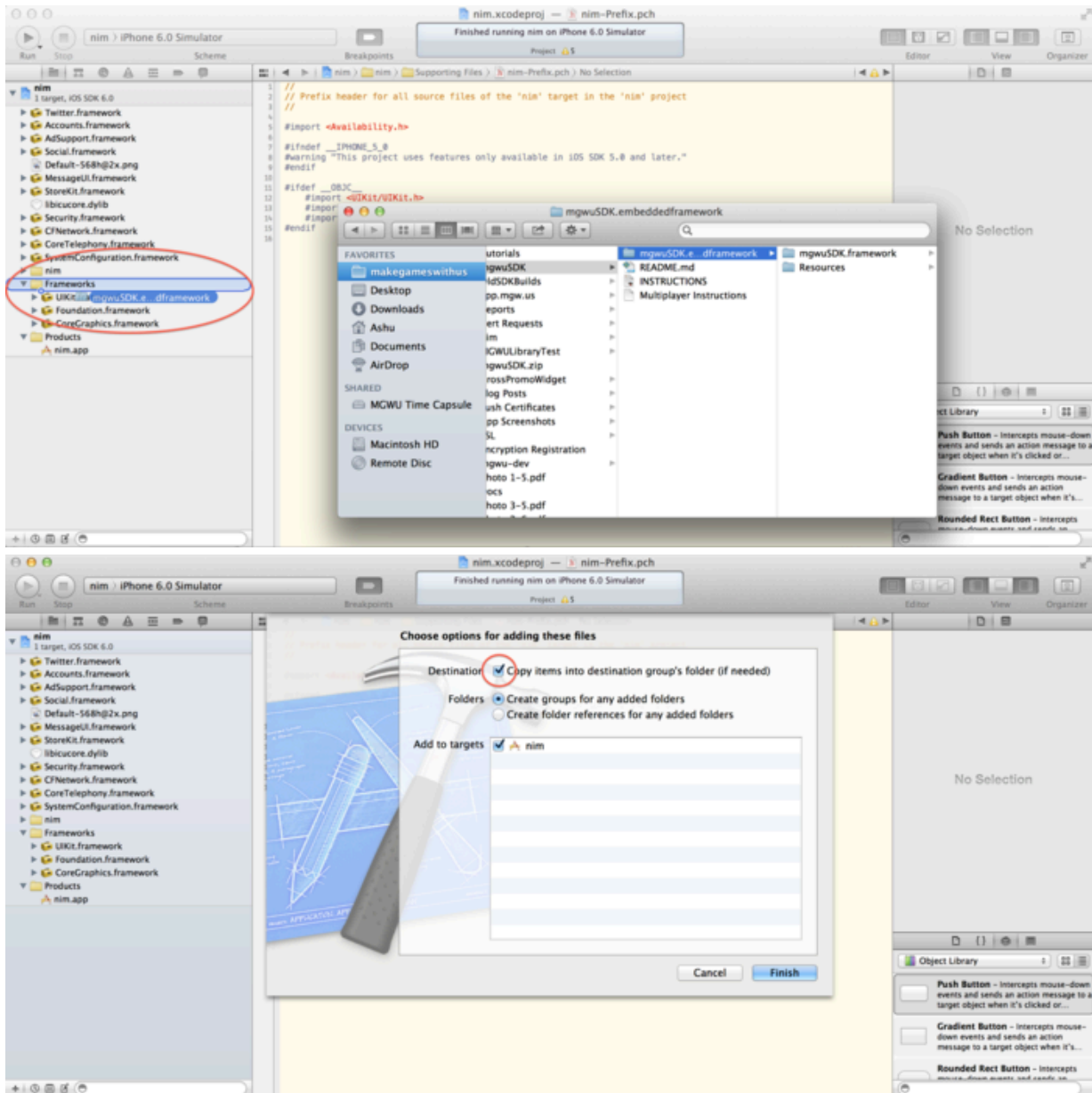
In the bottom left of the project navigator, search ".pch". Select the file that ends in either Prefix.pch or Prefix-iOS.pch. Right under #import <Foundation/Foundation.h> add the line:

```
#import <mgwuSDK/MGWU.h>
```

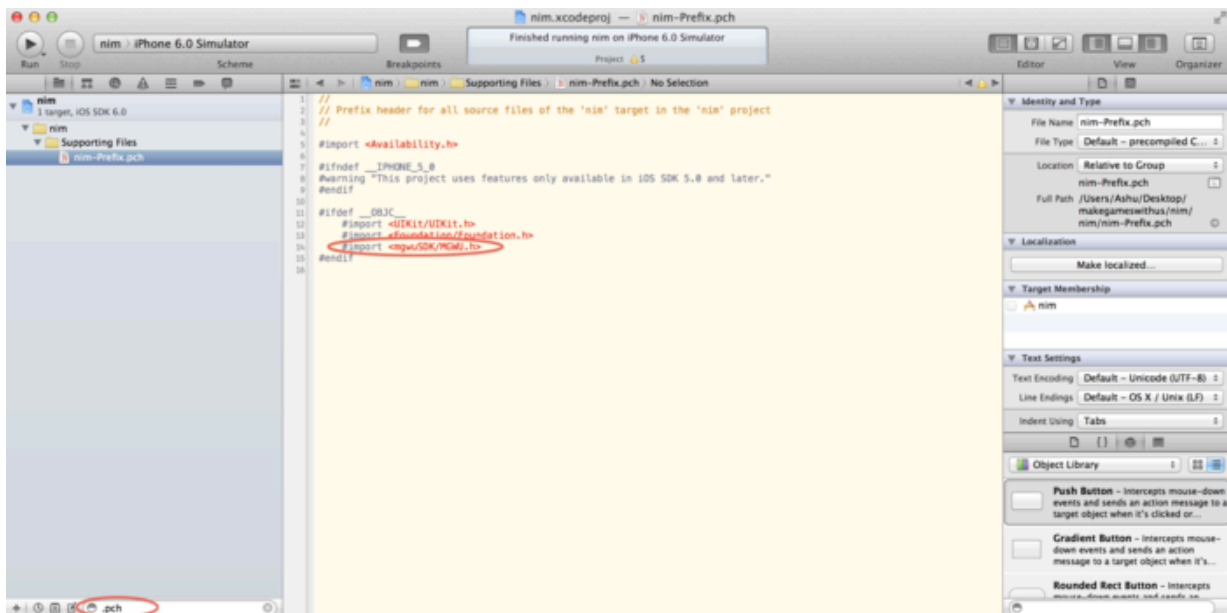Now open the file AppDelegate.m. In the method initializationComplete (or application: didFinishLaunchingWithOptions:) add the lines:

```
[MGWU loadMGWU:@"secretkey"];
[MGWU preFacebook]; //Temporarily disables Facebook until you integrate it later
```

Replace 'secretkey' with the secret key you used when you created your app on the server.

2. Cross Promo

When the More Games button is clicked include the line:

```
[MGWU displayCrossPromo];
```

Make sure this is only called once when the button is pressed

3. About Page

When the About Button is clicked, include the line:

```
[MGWU displayAboutPage];
```

Note: this will show a blank page until we add the about screen image for your game later on.

4. Contact Button

If you have a contact button in your game, when it is clicked, include the line:

```
[MGWU displayHipmob];
```

Note: initially this will do nothing, until we initialize hipmob before submitting the game to the app store. We will set you up on hipmob so you can live chat with users of your app.

5. Analytics

Basic analytics of opening and closing the app are included by default, as are opening the cross promo widget and the about page. We've also included analytics for sharing and buying in app purchases.

Do not make any events called "open", "close", "cross_promo_clicked", "about_clicked", "facebook_loggingin", "facebook_loggedin", "facebook_nothanks", "facebook_ogon", "facebook_ogoff", "facebook_ogpost", "facebook_posting", "facebook_posted", "facebook_inviting", "facebook_invited", "twitter_posting", "twitter_posted", "sms_sending", "sms_sent", "email_sending", "email_sent", "purchasing", "purchased", "restoring", "restored"

To log any additional events (such as completing the tutorial or completing a level) use this line of code:

```
[MGWU logEvent:@"levelcomplete" withParams:nil];
```

Where levelcomplete is replaced by any other event name, don't use special characters.

To log an event with parameters (like level number and score):

```
NSNumber* score = [NSNumber numberWithInt:9001];
NSNumber* levelnumber = [NSNumber numberWithInt:1];
NSDictionary *params = [[NSDictionary alloc] initWithObjectsAndKeys: score, @"score", levelnumber,
@"level_number", nil];
[MGWU logEvent:@"level_complete" withParams:params];
```

Where params is an NSDictionary with any parameters you wish to send up.

You want to make sure to use as few events as possible, if you have too many different types of events it will be really hard to see what is going on on the analytics dashboard. So for things like completing a level, you should pass the level number as a parameter. Or for example, the number of goldfish eaten in a game of Killah Killah Whale, you want to include that as a parameter for the event "game_completed)".

We will give you access to the Mixpanel analytics dashboard when your app goes live on the app store.

6. Prompt for Rating and Crash Reporting

These will work automagically, we will set it up for you before submitting your game to the app store.

7. Push Notifications

We do most of the legwork for setting up Push Notifications for you. First, we send reminder push notifications to a player if they haven't opened the app in a while.

You can set the message for the reminder push notifications with this line of code:

```
[MGWU setReminderMessage:@"Come back and play this game now!"];
```

You also need to add these methods to AppDelegate.m:

```
- (void)application:(UIApplication *)application didRegisterForRemoteNotificationsWithDeviceToken:
(NSData *)tokenId {
    [MGWU registerForPush:tokenId];
}

- (void)application:(UIApplication *)application didReceiveRemoteNotification:(NSDictionary
*)userInfo {
    [MGWU gotPush:userInfo];
}


- (void)application:(UIApplication*)application didFailToRegisterForRemoteNotificationsWithError:
(NSError*)error{
    [MGWU failedPush:error];
}

- (void)application:(UIApplication *)application didReceiveLocalNotification:(UILocalNotification
*)notification {
    [MGWU gotLocalPush:notification];
}
```

For multiplayer games, we automatically send push notifications when players make a move or send a message.

Soon we'll add a function to our website to allow you to send push notifications to your players.

If you want to use push notifications to send data from player to player or trigger events within the app, contact ashu@makegameswith.us for help.

Push notifications require a special provisioning profile, so they won't work at first, contact ashu@makegameswith.us to get set up to test them once your game is nearing completion.

8. In App Notifications

We display any push notifications you receive while in the app with a little dropdown notification. You can also use this dropdown notification to alert players in game.

First, you should make a copy of your app icon sized 34x34px and call it "MGWUIcon.png" and also make a retina version 68x68px called "MGWUIcon@2x.png" (if you don't have your app icon yet, use a placeholder image)

Then display the alert using this line:

```
[MGWU showMessage:@"Hey, how is it going?" withImage:nil];
```

If you have a special image you want to show, you can also instead use:

```
[MGWU showMessage:@"Hey, how is it going?" withImage:@"MyImage.png"];
```

Where MyImage.png is also a 34x34px image with an @2x retina version. (Note: you need to use @2x instead of -hd for this unlike in cocos/kobold)

9. Encrypted NSUserDefaults

NSUserDefaults is easily hackable if you jailbreak your phone. So if you happen to be saving things like the amount of

gold a player has or a players high score in NSUserDefaults, it would be easy for a player to cheat.

For things needing security (like scores or gold) you can use our encrypted version of NSUserDefaults like so (the syntax is almost identical to NSUserDefaults:

```
[MGWU setObject:myObject forKey:@"myKey"];
myObject = [MGWU objectForKey:@"myKey"];
[MGWU removeObjectForKey:@"myKey"];
```

You should only use this for ObjectiveC objects (NSArray, NSDictionary, NSString, NSNumber), not for custom objects you create.

You shouldn't use this for everything as it is slower (since it encrypts and decrypts the values when storing and retrieving)

10. Targets and Callbacks and Data

For many parts of the SDK we need to use Callback methods.

How to use

Normal methods return value's immediately. For example:

```
int i = [self myMethod];
```

However, sometimes methods can't do this. So instead we use a callback. Callbacks work like this:

```
[self myMethodWithCallback:@selector(myOtherMethod:) onTarget:self];
```

What this means, is instead of returning a value immediately, the method will send the value to a method you choose. The target is the object on which the method is called (this will typically be self), while the callback is the name of the method to be called.

myOtherMethod will be declared as:

```
- (void) myOtherMethod:(id)returnedValue
{
    //Do stuff with the returned value
}
```

The returned value can be any object (typically NSDictionary or NSArray) depending on the method (we will specify the type in our methods).

When using a method in our SDK that uses callbacks, the call will typically look like:

```
[MGWU someSDKMethodWithParameter:param withSelector:@selector(yourMethod:) onTarget:self];

- (void) yourMethod:(NSDictionary*)dict
{
    //Do stuff!
}
```

If for some reason the server request failed, the callback method will be passed a nil value, and typically the error will be printed in your console so you can correct the mistake.

When a dictionary is passed to a callback method, you also need to know what format the data is in inside the dictionary so you can work with it properly. In the tutorials we will explain it in this format:

```
NSString - @"String"
NSNumber - @4
NSArray - @[@"String1", @"String2", @"String3"]
NSDictionary - @{@"Key1":@"Value1", @"Key2":@2, @"Key3":@"Value3"}
```

These can also be nested like so:

```
@{@"myArray":@[@1, @2, @3], @"myString":@"string"}
```

This corresponds to the new [literal syntax in Objective-C](#)

Why we need callbacks

Anytime the SDK is retrieving data from the server we show a loading wheel (for a better user experience). Because of the way the iPhone works, you cannot show the loading wheel while calling a method that talks to the server and returns a value the normal way, so we need to use callbacks and targets.

11. High Scores

First read the tutorial on Targets and Callbacks.

Submit a high score using this line:

```
[MGWU submitHighScore:10 byPlayer:@"ashu" forLeaderboard:@"defaultLeaderboard"];
```

Where defaultLeaderboard is what you want to call your leaderboard. You can use different leaderboards for different game modes or levels.

Do not name any leaderboards "wins", "losses" or "rankpoints". Do not use periods or dollar signs in any leaderboard names. Also do not name them the same as any achievement names you use.

You probably want to allow users to input their name when they get a high score. (Otherwise you could call every player @"Player", but this is not as cool). If the player is logged into Facebook you can also use their Facebook username as their name for the leaderboards (see Facebook Methods tutorial).

If you call this method again with a lower score, it will be ignored. With a higher score, it will replace the previous entry on the server.

If your game works with low scores as better (like golf), simply negate all the scores before sending them to the server, and negate them again after retrieving them (magic algebra!)

To retrieve a high score use this line:

```
[MGWU getHighScoresForLeaderboard:@"defaultLeaderboard" withCallback:@selector(receivedScores:)
onTarget:self];
```

This will call a callback method called receivedScores passing in an NSArray, like so:

```
- (void)receivedScores:(NSDictionary*)scores
{
    //Do stuff with scores in here! Display them!
}
```

The scores will be returned like so:

```
@{
    @"user":@{@"name":@"player", @"score":@9001, @"rank":@42},
    @"all":@[
        @{@"name":@"player", @"score":@19001},
        @{@"name":@"player", @"score":@19000},
        ...]
}
```

If the user is logged into facebook, the scores will be returned like so:

```
@{
    @"user" : @{@"name":@"player", @"score":@9001, @"rank":@42, @"friendrank":@1, @"fbname":@"John
A. Smith", @"username":@"johnnys"},
    @"all" : @[
        @{@"name":@"player", @"score":@19001},
        @{@"name":@"player", @"score":@19000},
        ...],
    @"friends" : @[
        @{@"name":@"player", @"score":@9001, @"fbname":@"John A. Smith", @"username":@"johnnys"},
        @{@"name":@"player", @"score":@9000, @"fbname:@"Jill Shaw", @"username":@"jjshaw"},
        ...]
}
```

Both arrays (all and friends) will have the top 50 scores (or less if there are less than 50 players/friends). The username is the players facebook username (aka vanity url), while the fbname is the players real name.

If you submit the high score immediately before retrieving the high scores (for example, submitting the highscore and displaying the high scores in the game over screen), it will not be updated, so instead use this call to submit and retrieve the scores at the same time:

```
[MGWU submitHighScore:10 byPlayer:@"ashu" forLeaderboard:@"defaultLeaderboard"
withCallback:@selector(receivedScores:) onTarget:self];
```

The callback will be passed a dictionary in the same format as described above

If you need to delete all high scoores during testing, you can use this url:
https://dev.makegameswith.us/clearhighscores

High scores you submit in development won't show up when the app is in the wild.

12. Achievements

Submit achievements using this line:

```
[MGWU submitAchievements:arrayOfAchievements];
```

Where array of achievements would be a string of achievements like this:

```
@[@"Achievement1", @"Achievement2", @"Achievement2"];
```

The example array contains two copies of Achievement2 because depending on how you structure achievements, they could be earned multiple times.

Do not name any achievements "wins", "losses" or "rankpoints", also do not name any of the achievements the same as a leaderboard name. Do not use periods or dollar signs in achievement names.

Things that will be incremented very frequently (like number of times the player played a game) are better stored as a high score (where you can arbitrarily increment them). As a general rule of thumb, any type of achievement that you get 5+ of in a play session, you should store it as a high score.

You should also notify the player (using the showMessage method described in the In App Notifications tutorial) when they get an achievement you want them to see.

Retrieve achievements using this line:

```
[MGWU getAchievementsWithCallback:@selector(gotAchievements:) onTarget:self];
```

The callback will be passed a dictionary in this format:

```
@{@"Achievement1":@1, @"Achievement2":@2, @"defaultLeaderboard":@9001}
```

Note: this also will retrieve all the high scores a user has across all their leaderboards.

If you submit achievements immediately before retrieving achievements, it will not be updated with the latest ones. But this should not be a problem, since you would save achievements at the end of a game, and display the achievements on some sort of profile / stats view (plus you should probably also store this info locally).

If the player is logged into Facebook (read the Facebook Methods tutorial), you can retrieve their list of friends who play the game with this line:

```
[MGWU getPlayingFriends];
```

This will return an array in this format:

```
@[
    @{@"name":@"John A. Smith", @"username":@"johnnys"},
    @{@"name:@"Jill Shaw", @"username":@"jjshaw"},
... ]
```

Then you can get the achievements of their friends using this line:

```
[MGWU getAchievementsForPlayer:@"username" withCallback:@selector(gotProfile:) onTarget:self];
```

The callback will be passed a dictionary in the same format as above.

If your game has a different format for achievements than the SDK currently supports, please drop me a line (ashu@makegameswith.us) and I can add the features you want.

If you need to delete all achievements during testing, you can use this url:
https://dev.makegameswith.us/clearachievements

Achievements you submit in development won't show up when the app is in the wild.

13. In App Purchases

While you're building the app, you should use the following methods to test buying and restore in app purchases:

```
[MGWU testBuyProduct:@"com.xxx.xxx.productID1" withCallback:@selector(boughtProduct:)
onTarget:self];
[MGWU testRestoreProducts:@[@"com.xxx.xxx.productID1", @"com.xxx.xxx.productID2"]
withCallback:@selector(restoredProducts:) onTarget:self];
```

The callback method for testBuyProduct will be passed an NSString which will either be nil (if the user canceled the purchase) or the productID you passed it (if it was successful). The callback method for testRestoreProducts will be passed an NSArray which will either be nil (if the user canceled the restore) or the array of productIDs you passed it (if it was successful).

When a user successfully purchases or restores, it would be great to let them know it was successful using the showMessage method (see the In App Notifications tutorial).

You initially want to simply make up product IDs for the things you think you will include as in app purchases. You also need to include the restore functionality in your store for people who get a new device and download the app so they can restore their in app purchases (Apple requires this).

Once your app is nearing completion, you will switch to the real in app purchase methods (they behave identically to the test methods) and real product identifiers. We need to enable these for you, so contact ashu@makegameswith.us to get set up with this.

First, you need to add this line after [MGWU loadMGWU]; in AppDelegate.m:

```
[MGWU useIAPs];
```

Then switch the test methods with these methods:

```
[MGWU buyProduct:@"com.xxx.xxx.productID1" withCallback:@selector(boughtProduct:) onTarget:self];
[MGWU restoreProductsWithCallback:@selector(restoredProducts:) onTarget:self];
```

14. Twitter Sharing

You should add a twitter share button strategically after the player does something cool. Look at other successful games to see where good places to put this might be.

First, you should make a copy of your app icon sized 34x34px and call it "MGWUIcon.png" and also make a retina

version 68x68px called "MGWUIcon@2x.png" (if you don't have your app icon yet, use a placeholder image).

Twitter sharing is only enabled in iOS5 and iOS6 when the user is logged into Twitter on their device. You can check if they meet these conditions using the call:

```
[MGWU isTwitterActive];
```

You should only display the tweet button if the isTwitterActive method returns true. Then when the user clicks on the tweet button, use this method to pull up the widget that lets you send tweets:

```
[MGWU postToTwitter:@"Your Default Message Goes Here"];
```

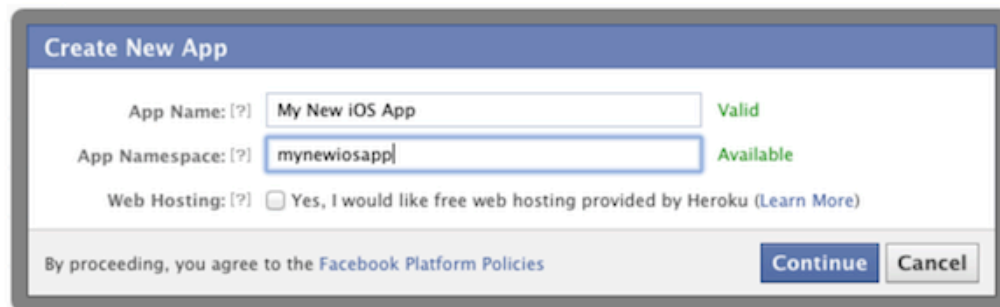The default message should include "@yourgamestwitterhandle" and "@makegameswithus"

The image MGWUIcon.png is automatically added to the tweet as well as a link to your game.

15. Getting Started With Facebook

We're integrating facebook in all our apps for better distribution. This happens both through sharing as well as facebooks new open graph SDK.
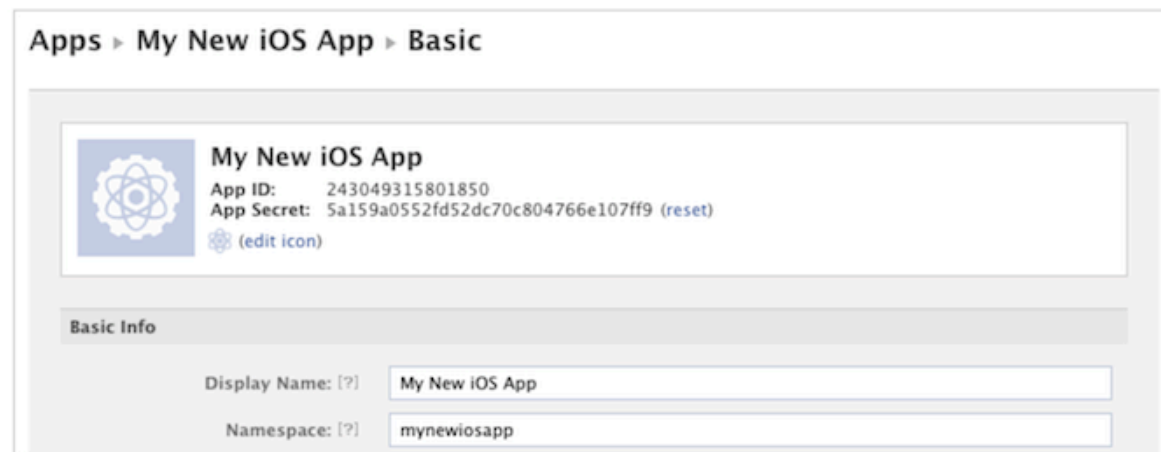
First follow these steps:

Create a new app on the Facebook App Dashboard, enter your app's basic information.

Once created, note the app ID shown at the top of the dashboard page.
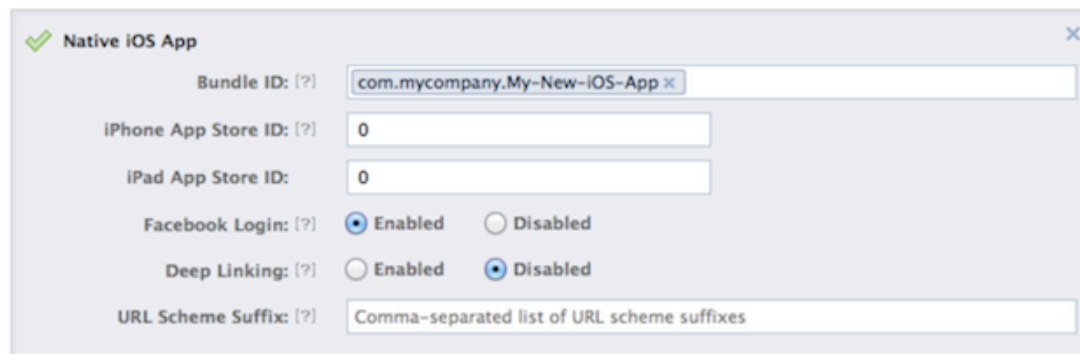
Alternatively, you can of course use the ID of an existing app.

Now, you need to set the Bundle Identifier and configure your Facebook application to support login from an iOS application.

(Note: this is a new step in the Facebook SDK for iOS v3.1. Previous versions of the SDK did not require this step.)

Click on the checkmark next to "Native iOS App" and supply your Bundle Identifier in the "Bundle ID" field. In addition, make sure the Facebook Login radio button is set to "Enabled".

Adding your Facebook App ID

Finally, you need to place the Facebook app ID in two places in your application's main `.plist` file. Create a key called `FacebookAppID` with a string value, and add the app ID there:
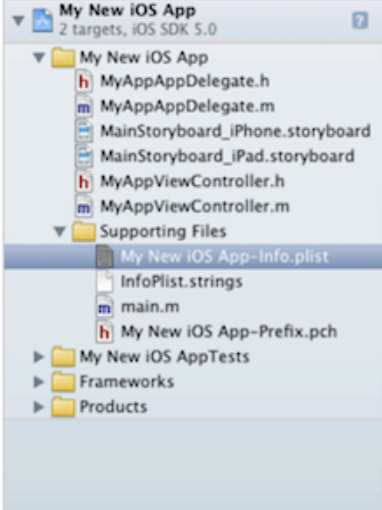
Key	Type	Value
Localization native development region	String	en
Bundle display name	String	${PRODUCT_NAME}
Executable file	String	${EXECUTABLE_NAME}
▶ Icon files	Array	(0 items)
Bundle identifier	String	com.mycompany.${PRODUCT_NAME}
InfoDictionary version	String	6.0
Bundle name	String	${PRODUCT_NAME}
Bundle OS Type code	String	APPL
Bundle versions string, short	String	1.0
Bundle creator OS Type code	String	????
Bundle version	String	1.0
Application requires iPhone environmer	Boolean	YES
Main storyboard file base name	String	MainStoryboard_iPhone
Main storyboard file base name (iPad)	String	MainStoryboard_iPad
▶ Required device capabilities	Array	(1 item)
▶ Supported interface orientations	Array	(3 items)
▶ Supported interface orientations (iPad)	Array	(4 items)
FacebookAppID	String	243049315801850

Also, create an array key called `URL types` with a single array sub-item called `URL Schemes`. Give this a single item with your app ID prefixed with `fb`:

▼ URL types		Array	(1 item)
▼ Item 0		Diction...	(1 item)
▼ URL Schemes		Array	(1 item)
Item 0		String	fb243049315801850

(This is used to ensure the application will receive the callback URL of the web-based OAuth flow.)

Then add this method to AppDelegate.m:

```
- (BOOL)application:(UIApplication *)application openURL:(NSURL *)url sourceApplication:(NSString *)sourceApplication annotation:(id)annotation {
    // attempt to extract a token from the url
    return [MGWU handleURL:url];
}
```

And delete this line from AppDelegate.m:

```
[MGWU preFacebook];
```

Finally, you should make a copy of your games logo smaller than 300px wide by 80px tall and call it "Logo.png" and also make a retina version doublethe site called "Logo@2x.png" (if you don't have your app logo yet, use a placeholder image).

The login flow for Facebook will be handled automatically for you. Also when a player goes to a new device and logs in with Facebook, all their data will be transferred. The user will now be asked to authenticate with Facebook the first

time they open the app, and if they say no, it will ask them again after 12 and 24 play sessions.

16. Facebook Methods

Make sure you have read Getting Started With Facebook to get everything set up.

To find out if a user is currently logged into Facebok, use the method:

```
[MGWU isFacebookActive];
```

To get the username of the logged in player:

```
[MGWU getUsername];
```

This will return a string with the players username

You should add a facebook share button strategically after the player does something cool. Look at other successful games to see where good places to put this might be. You should only display the share button if the user is logged into Facebook in your app, you can check using this method:

```
[MGWU isFacebookActive];
```

Note: you don't need to check for this if you're building a turn based multiplayer game, all users will be logged into facebook.

When the share button is clicked, use this method to post:

```
[MGWU shareWithTitle:@"Title of the Post" caption:@"Caption of the Post"
andDescription:@"Description of the Post"];
```

The app icon will automatically be included in the post. Note: the description will not show up on the Facebook mobile app, so you want to use the title and the caption of the post to sell the app to potential future users. We can help you with the wording of this.

You can also include buttons to allow users to like your games page on facebook or the MakeGamesWithUs page on facebook. For your games page (replace the number with the facebook pageID):

```
[MGWU likeAppWithPageId:@"112244422333"];
```

MakeGamesWithUs:

```
[MGWU likeMGWU];
```

We will also set you up with some OpenGraph actions to help drive distribution to your app. The only thing you need to do is include a button/switch to toggle opengraph on/off in some settings menu of your app. First check if facebook is active (otherwise you don't need to show this button). Then you can check the current state of whether OpenGraph is enabled or disabled with this method:

```
[MGWU isOpenGraphActive];
```

And finally to toggle OpenGraph on or off:

```
[MGWU toggleOpenGraph];
```

Lastly, if you're not developing a turn based multiplayer game, there are two more relevant methods relating to facebook.

The first is to allow the player to login to facebook if they are not already. This button can go in place of the toggle open graph button (since it will only be displayed when the player is not logged in). To log the user in:

```
[MGWU loginToFacebook];
```

You also want a button to allow people to invite their friends to the game. Only display the button if the user is able to invite friends, you can check using this method:

```
[MGWU canInviteFriends];
```

To invite friends:

```
[MGWU inviteFriendsWithMessage:@"Default Invite Message"];
```

This method will try to use facebook to invite friends, if that is not available, it will try to use the text message prompt, and if neither are available it will use mail to invite friends. The link to the app will be prepopulated (as will the app icon for facebook).

17. Turn Based or Social Multiplayer Games

a. Setup

Turn Based Multiplayer games rely on Facebook. These games gain huge benefit from the ability to play and share with your friends. It also simplifies the user creation flow, no need to create an independent login, and we can even use Facebook usernames (aka vanity urls) as the username in game.

Make sure you have read the Callbacks and Targets tutorial as well as the Getting Started with Facebook and Facebook Methods tutorials.

It would also be really good to check out our project templates , we highly recommend starting from those because it will make your life way easier. There is not much point in reinventing the wheel with the menu structure for this kind of game, so better to start from somewhere.

For these games, we require every user uses Facebook. You can do so in your app by adding this line after [MGWU loadMGWU] in AppDelegate.m:

```
[MGWU forceFacebook];
```

The multiplayer system is built around the concept of back and forth games between two players (similar to Words with Friends). If you're building a multiplayer/social game that won't work using our current SDK, please drop me a line (ashu@makegameswith.us), we can most likely add the features you want to the SDK.

b. The Game Object

The Game object represents a single game between two players. It is a dictionary with this format:

```
@{
    @"gameid" : @8,
    @"players" : @[
        @"desaiashu",
        @"shea_sidau"
    ],
    @"turn" : @"shea_sidau"
    @"gamestate" : @"inprogress",
    @"gamedata" : @{
        @"dictionary containing the current gamedata" : @"for example, in chess this would have the
locations of all the pieces on the board"
    },
    @"movecount" : @5,
    @"moves" = @[
        @{
            @"dictionary containing 3 moves ago" = @"for example, in chess this could have info
describing White Queen to E5",
            @"time" : @1349907651
        },
        @{
            @"dictionary containing 2 moves ago" = @"for example, in chess this could have info
describing Black Queen to E5",
            @"time" : @1349916403
        },
        @{
            @"dictionary containing the last move" = @"for example, in chess this could have info
describing Black Rook to E5",
            @"time" : @1349916412
        }
    ],
    @"newmessages" : @0,
    @"datebegan" : @1349907651,
    @"dateplayed" : @1349916492,
}
```

(All timestamps are in epoch time)

The gamedata and moves are defined by you. The array of moves is limited to the last 3 moves (contact me at ashu@makegameswith.us if you have a game that needs more of the past moves).

@"newmessages" is the number of unread messages in the chat for each game.

@"turn" tells you whose turn it is, currently games are required to be strictly turn based (contact me at ashu@makegameswith.us if you have a game that needs a different format).

There are three states a game can be in:

- @"started" when you a new game has just been begun
- @"inprogress" when a game is in progress

- @"ended" a game that is completed

c. Getting the User's Data

To implement the turn based multiplayer, there are two core methods, "getMyInfo" and "move". "getMyInfo" retrieves the user, along with their list of friends and their list of current and past games. Games that were last played over 30 days ago will be forfeited (if in progress) and deleted. "move" enables you to make a move in any game currently started or in progress.

Call getMyInfo like this:

```
[MGWU getMyInfoWithCallback:@selector(gotUserInfo:) onTarget:self];
```

The callback will be passed an NSDictionary with this format:

```
@{
    @"user" : @{
        @"username" : @"shea_sidau";
        @"name" : @"Shea Sidau";
        @"wins" : @3;
        @"losses" : @4;
        @"rankpoints" : @0;
        @"lastplayed" : @1350159215;
    },
    @"friends" : @[
        @{
            @"username" : @"desaiashu";
            @"name" : @"Ashu Desai";
            @"wins" : @4;
            @"losses" : @3;
            @"rankpoints" : @0;
            @"lastplayed" : @1350159615;
        },
    ... ],
    @"games" : @[game1, game2, game3]
}
```

game1, game2 and game3 will be dictionaries of the format described above. The list of friends includes all facebook friends who also play the game.

From here you probably want to create 3 menus, the first is a list of your current and past games, the second is players to start a game with, and finally a list of players to invite. The list of current and past games should be straightforward, you should seperate them into 3 categories, your turn, opponents turn and past games, and order them based on when they were last played to keep the fresh ones at the top.

c. Picking an Opponent

The list of players to start a game with will have a few components.

You want to present a list of facebook friends who also play the game, but make sure to remove the players you are currently in a game with from the list.

You also need to allow players to begin games with random players, or by searching a username. To retrieve a random player:

```
[MGWU getRandomPlayerWithCallback:@selector(gotRandomPlayer:) onTarget:self];
```

The callback will be passed a dictionary containing info of a random player you are not currently in a game with (identical to the example friend dictionary above, just without the @"name" key)

While you are testing, you might be in a game with every player, in which case this function will go into an infinite loop and timeout after a few minutes. To add more players to test the random function, use this url: https://dev.makegameswith.us/creategamer

These fake players won't show up when the app is in the wild.

To search for a player by username:

```
[MGWU getPlayerWithUsername:@"username" withCallback:@selector(gotPlayer:) onTarget:self];
```

The callback will be passed a nil value if the player doesn't exist or if you are currently in a game with that player (and a notification explaining this will be shown), otherwise it will recieve a dictionary containing info of the player (identical to the example friend dictionary above, just without the @"name" key).

You probably also want to provide a function that randomly picks a facebook friend to start a game with (but make sure it won't give you a player you are currently in a game with).

It would also be a good idea to have a recommended friends section, where it randomly suggests you 3 friends, 1 of which has played recently, 1 of which hasn't played in a week or two and 1 of which that hasn't played in a month. If the player does not have 3 friends who play the game, you can supplement it with friends to invite (we'll get to this later).

Finally, if you want to display a profile of another player including their win/loss record, their rankpoints, their achievements and high scores in the game, you can get this info using the call:

```
[MGWU getAchievementsForPlayer:@"username" withCallback:@selector(gotProfile:) onTarget:self];
```

The callback will be passed a dictionary in this format:

```
@{@"Achievement1":@1, @"Achievement2":@2, @"defaultLeaderboard":@9001, @"wins":@4, @"losses":@5,
@"rankpoints":@50}
```

If you want to delete all the players during testing, you can use this url: https://dev.makegameswith.us/cleargamers

d. Making Moves

Now to make a move in one of the games, you use the call:

```
[MGWU move:move withMoveNumber:moveNumber forGame:gameID withGameState:gameState
withGameData:gameData againstPlayer:opponent withPushNotificationMessage:pushMessage
withCallback:@selector(moveCompleted:) onTarget:self];
```

The callback will be passed the new updated game object.

Depending on the state of the game, the parameters you pass to the move method will be as follows:

- **Beginning a game:**
  - move (NSDictionary) should be a dictionary with info about the move that was just performed (in chess this would be along the lines of Queen to E5)
  - moveNumber (int) should be 1
  - gameID (int) should be 0
  - gameState (NSString) should be @"started"
  - gameData (NSDictionary) should be a dictionary with info about the game (in chess this would be where the pieces on the board are)
  - friend (NSString) should be the username of the player you are starting the game against
  - pushMessage (NSString) should be whatever you would like to display to the other user in a push notification about the move (like, @"username invited you to a game")
- **Making a move in the game:**
  - move (NSDictionary) should be a dictionary with info about the move that was just performed (in chess this would be along the lines of Queen to E5)
  - moveNumber (int) should be 1 + the move number of the game retrieved from the list of games
  - gameID (int) should be the gameId retrieved from the list of games
  - gameState (NSString) should be @"inprogress"
  - gameData (NSDictionary) should be a dictionary with info about the game (in chess this would be where the pieces on the board are)
  - opponent (NSString) should be the username of the player you are playing against, retrieved from the list of games
  - pushMessage (NSString) should be whatever you would like to display to the other user in a push notification about the move (like, @"it is your turn against username")
- **Making a move to end a game (checkmate condition):**
  - move (NSDictionary) should be a dictionary with any info about the move that was just performed (in chess this would be along the lines of Queen to E5)
  - moveNumber (int) should be 1 + the move number of the game retrieved from the list of games
  - gameID (int) should be the gameId retrieved from the list of games
  - gameState (NSString) should be @"ended"
  - gameData (NSDictionary) should be a dictionary with info about the game (in chess this would be where the pieces on the board are)
  - opponent (NSString) should be the username of the player you are playing against, retrieved from the list of games
  - pushMessage (NSString) should be whatever you would like to display to the other user in a push notification about the move (like, @"you lost to username")

IMPORTANT NOTE: Do not use a period or dollar sign in the keys of any dictionaries you create for move or gamedata, the data will not be successfully stored on the server. Here are examples of what not to do:

```
@{@"my.key":@"myValue"}
@{@"$myKey":@"myValue"}
@{@"myKey":@{@"my.key":@"myValue"}}
```

Also, make sure you don't add any uninitialized values to dictionaries, otherwise you will get lots of unpredictable errors.

To define the winner (so the server can keep track of wins/losses) as well as to define ranking points. You want to include these keys in gamedata dictionary when the game is ending:

```
@"winner":@"usernameOfWinner",
@"rankpoints":@{
    @"player1":@50,
    @"player2":@-50
}
```

Ranking points can be used as a rating / ELO system for players. Passing in negative numbers will reduce a players rankpoints.

Once you make a move, you might want to allow people to smack talk on each other's facebook walls. This is only possible if you are friends with your opponent, you can check this using this line:

```
[MGWU isFriend:@"username"];
```

If it returns true, show a button that triggers this method:

```
[MGWU postToFriendsWall:@"username" withTitle:@"Title of the Post" caption:@"Caption of the Post"
andDescription:@"Description of the Post"];
```

This will pop up a dialog to allow the user to write on their friends wall. The app icon will automatically be included in the post. Note: the description will not show up on the Facebook mobile app, so you want to use the title and the caption of the post to sell the app to potential future users. We can help you with the wording of this.

e. Reloading a Game

If you want to reload a certain game without reloading all the userinfo, you can use the method:

```
[MGWU getGame:gameID withCallback:@selector(gotGame:) onTarget:self];
```

The callback will be passed the updated game object

If you want to delete all games during testing, you can use the url: https://dev.makegameswith.us/cleargames

f. Chat

Within each game you also want to let users chat with each other. In each game object there is a "newmessages" key which tells you how many unread messages the player has from his opponent. You can use this to indicate to the player that there are new messages. To load the messages, use the method:

```
[MGWU getMessagesWithFriend:@"username" andCallback:@selector(gotMessages:) onTarget:self];
```

The callback will be passed an array in the format:

```
@[
    {
        @"from" : @"desaiashu";
        @"message" : @"yo";
        @"time" : @1349311932;
    },
    {
        @"from" : @"shea_sidau";
        @"message" : @"hi";
        @"time" : @1349314131;
    },
... ]
```

The newer messages will be later in the array, and all include an epoch timestamp which you can convert into a human readable string. The last 100 messages will be retained. The chat does not live update, but the user will receive a push notification when he gets a message.

If you want to clear messages during testing, you can use the url: https://dev.makegameswith.us/clearmessages

g. Inviting Facebook Friends

You should also present a list of facebook friends to invite to the game to the user, this will get you lots more users! You can get a list of facebook friends who have iOS devices but don't have the app using this method:

```
[MGWU friendsToInvite];
```

This will return an array in the format:

```
@[
    @{@"name":@"John A. Smith", @"username":@"johnnys"},
    @{@"name:@"Jill Shaw", @"username":@"jjshaw"},
... ]
```

You can then pop up a dialog to invite the friend using this line:

```
[MGWU inviteFriend:@"username" withMessage:@"Check out this cool app!"];
```

We can help you with the message if needed.

Note: you are able to start a game with a player who hasn't joined the app yet, so you should also have the player start a game with the player they are inviting.

h. Misc

There are a bunch of other intracacies about the best way to implement things in these games that we haven't covered here, the way to pick up these things is by playing around with our templates, and better yet start off with the template!