2015年度 卒業論文

SNSのデータストア利用の効果と評価

大阪産業大学 工学部 情報システム工学科
情報教育システム研究室

12H108 本田将之

目次

1　はじめに

現在、若者のゲームセンター離れが続いており、店舗数が激減している。[1] 家庭用ゲームでのオンライン環境が揃い、ゲームセンターの旨味が消えてきていることが伺える。その状況下でゲームセンターへ足を運ぶ人は減少こそしているが、居なくなっているわけではない。だが、様々な理由がある中、現在普及している家庭用ゲームやスマホゲームだけではなく、ゲームセンターの様な外での環境でゲームをプレイしたく足を運ぶユーザーは存在する。そして、減少していくゲームセンターでの人口から起こる問題がある。多人数プレイが目的でゲームセンターに向かった場合、近所にゲームセンターが二つありどちらに人が多いのかわからずに向かい、人が居なくて後悔したが別店舗では人で賑わっていた場合だ。多人数プレイを目的としていたので、大勢が居るゲームセンターに向かいたかったはずだが結果的に無駄な時間を過ごしたのだ。この場合どちらも人が居ないのかそれとも、片方しか人が居ない状況なのかを把握していない為、結果的に無駄な労力と時間を使っている。逆に人が居ないゲームセンターに行きたい場合でも同じことが考えられる。そこでこの無駄な時間と労力を無くせないのかと考えた。

なぜ、人が居るゲームセンターを探す必要があるのか。その問題の理由としてあがるのがゲームセンターには、一人用ゲームと他に、対戦を目的とした 2 人以上でプレイするゲームが存在するからである。対戦を目的をしたゲームで、人が居ない日には一日中対戦が発生しない場合も考えられる。その場合、他店舗へ行きそこで対戦をしようと思い他店舗に向かうが、そこに人が居なければ対戦できる確証もない。現状、知り合いのプレイヤーを SNS やチャットツールなので同じゲームセンターに集めるようにし対人戦を発生させるようにしている。だがこの手段では知り合い以外とゲームをプレイしようとすることができない。そこで、複数ある近場のゲームセンターで、何処に人が多く居るのかを表示できるようにし、わざわざゲームセンターまで行き確認しなくてゲームセンターの状況を把握できるアプリの開発を行った。第 2 章では本研究の目的について述べる。第 3 章では本研究に類似したアプリ、システムを示す。第 4 章では本研究で開発したシステムの概要を述べる。第 5 章ではデータストアとしての効果と、その結果を考察とともに述べる。第 6 章には研究の成果とともに結論を述べる。

2　本研究の目的

本研究では、SNS をデータストア利用をする事によって起きる効果を検証する事を目的とする。ゲームセンターに行かなければ人が居るのか確認できない為、無駄な時間を使ってしまう事がある。そこで、外出時にでもゲームセンターに人が居るかを把握できるスマートフォンアプリを開発を行った。制作にする際、新たにユーザー情報や施設の情報を登録する場合に、サーバーを開発する必要があるが労力や時間がかかる。本研究では、SNS に登録されているユーザー情報や、既に登録されている施設情報を API で扱う事で、サーバーを構築する手間を省きアプリを開発する。これらを踏まえて、SNS をデータストアとして利用するアプリを開発を行い効果を検証する。

3 本システムに類似したアプリ、システム

本システムに類似したアプリ、システムを列挙する。

3.1 FoursquareAPI を使った類似アプリ

FoursquareAPI [2] を活用しているアプリを以下に列挙する。

JOIN US

JOIN US（ジョイナス）は、いま近くにいるユーザーと合流して、飲んだり、食事が出来る Facebook を利用した会員制のマッチングアプリです。正午から翌朝 5 時限定で、“ 今夜の飲み仲間 ” を探す事ができます。自身のいる場所から半径 20km 圏内にいる、【飲みに行きたい・今まさに飲んでいる】ユーザー（ログイン中のユーザー）が、距離が近い順、またはログイン時間が近い順にマッチングして表示し、ユーザーが選択する。位置情報と Facebook を連動させているアプリ。

Pinnote

店舗や施設にチェックインするとその回数に応じて、同じ関心をもつであろう「ピノトモ」と友達になることができるアプリ。これも位置情報を使っているアプリ。

Girls Around Me

このアプリで自分の近くにいる女の子を見つけることができる。メーカーのサイトには「キミの街をデート天国に変える画期的なアプリだと記載されています（男の子を捜すこともできます)。アプリでは、チェックインしている女の子を地図上で表示し、その子がプロフィールで公開している情報を、写真も含めてすべて見ることができ、気に入った女の子にはアプリから友達申請だってできます。Foursquare の API を使用して女性の位置を把握している為 Facebook と foursquare のアカウントが必要になる。JOIN US に少し似ているが、誘える範囲や難易度は格段に違うと思われる。

Queuespot

お気に入りのお店や、ディズニーランドのアトラクションの待ち時間を可視化できるアプリ。Queuespot は位置情報、ソーシャル、そしてクラウドソーシングのパワーでリアルタイムに世界中の行列を可視化します。Facebook アカウントもそのまま使え、Twitter との連携も可能である。

eyeland

自分の周辺範囲だけの出合い系アプリ。アカウントを登録し「Check in」すると『Google マップ』に自分の居場所と、ほかのユーザーの場所が表示される。青色は男性、赤色は女性という風に区別されており、さらにタップすることにより、その人のプロフィールを見ることができる。

カフェサーチ

GPS 機能によって近くにあるカフェを素早く地図上に表示してくれる「カフェサーチ」というアプリ。Wi-Fi 環境か、分煙はされているのかといった情報が確認可能です。スターバックスやドトールコーヒー

といったカフェチェーン、モスバーガーやサブウェイなどのファストフード店も表示されますが、特定の店舗を表示させることも可能である。

コンビニどこだ？

近くのコンビニがわからない時に使用するアプリ。起動したらすぐに地図が表示され、周辺のコンビニを探し地図上にアイコンを表示させる。尚コンビニの種類によってアイコンの表示が変わる。各コンビニの種類別で検索結果を操作できる為ユーザーにわかりやすい設計になっている。

3.2 その他の類似システム

本研究のアプリに類似しているシステムなどは以下のシステムがある。

GUILTY GEAR Xrd -REVELATOR-の Twitter 連携システム

”GUILTY GEAR Xrd -REVELATOR-”と言うゲームを、Aime カードを利用してプレイし「GGXrd R プレイヤーズギルド」に登録をすると、Twitter と連携することが可能になります。今現在自分がプレイしている店舗名をツイートすることが出来ますので、対戦相手や仲間を探したい方や、プレイ中の活躍を残したい方はこれを使うといいだろう。

バスキャッチ

バスキャッチとはバスの正確な到着案内を可能にしたシステム。1 分に 1 回のデータ通信間隔と位置情報予測プログラムなどで、高層ビルや高架下などの GPS 位置情報の捕捉がしにくい場所でも、より精度の高い到着予測を可能にしたシステムです。幼稚園や保育園、教習場などで利用されている。

4 本研究で提案したシステム

この章では動作環境と開発環境、システム仕様について説明する。

4.1 仕様・動作

本研究で扱う iPhone アプリケーション上で動作するシステムを開発した。Twitter と Foursquare の機能を用いてシステムの雛形を作成した。

4.1.1 Fousquare Plugin

FousquerAPI を用いて近辺のゲームセンターを情報を取得するシステムの開発を行った。アプリを起動したタイミングで位置情報を取得し、その情報を FoursquerAPI に与えて周辺の施設の情報を取得することで、地域が離れたゲームセンターのような不要な情報をなくすことができる。周辺施設情報の結果を Arcade のカテゴリに属する施設だけを表示する事で、さらに不要な情報を省く事ができる。また、ここで取得した施設の名称の情報をツイートさせる事で、ユーザーがどこのゲームセンターに居るのかをツイートする手間が省け、更にユーザーが変更しないハッシュタグができるので正確な検索結果を取得する事できる。

- 仕様
  - venues から始まる API（venues/search, venues/explore 等）は、ひとつのアプリケーションにつき、1 時間あたり合計 5,000 回まで情報を取得できる。
  - 位置情報を取得し、緯度経度のようにカンマ区切りで指定するのが必要。
  - iOS8 から位置情報を仕様するには NSLocationWhenInUseUsageDescription か NSLocationAlwaysUsageDescription を指定する必要がある。
  - 最大検索結果件数は 50 件である。
- 動作
  - 緯度経度から周辺施設の情報を検索する為、現在地をアプリ起動時に取得。
  - 図 2 に表示された画面からアプリを起動後”検索する”ボタンをタッチする事で、位置情報を取得し 周辺施設を表示する。実際に図 3 に示す。
  - 選択したゲームセンターの店名情報を、Twitter Plugin に渡す。

4.1.2 Twitter Plugin

TwitterAPI [3] を用いて、データストアとして扱った Twitter から情報を呼び出すシステムの開発を行った。TimeLine から特定のキーワードやハッシュタグを検索する事によって、Twitter をデータの保存場所として利用する事でデータストアとして活用させる。Twitter - PublicAPI を利用して、店舗の名前に特定のハッシュタグが付いたツイートを検索し、検索結果をアプリ画面上に表示する。ユーザーが任意の店舗に到着し、そのゲームセンターに居ることを本研究のアプリ内でツイートをすることで、Twitter にデータを保存することができる。さらに、特定のツイートを投稿したユーザーに応じて、現在どの程度人がこのゲームセンターに居るのかを画面に表示して把握できるようにした。その為、店舗名が入った特定のツイートをできるよう、選択した店舗名と特定のツイートが、コメント欄に初期から書き込まれるようにした。

- 仕様
  - TwitterAPI の PublicAPI を用いているため、API 制限が存在する。
  - Timeline の最大検索結果件数は 100 件である。
  - ユーザーだけの認証ならば、15 分の間に 180 回検索結果を呼び出す事ができる。
  - Twitter アカウントを切り替え可能である。
  - TimeLine 上の Image-icon を取得している。
- 動作
  - TwitterAPI を用いて得た特定のツイートを、対応した店毎にツイートされた内容を画面に表示。実際に周辺施設の取得し表示した画面を図 4 に示す。
  - 店毎に、対応したツイートの数をカウントして表示する。実際に人数を表示している画面を図 5 に示す。
  - Twiiter にツイートする為のボタンを用意し、選択した店名とハッシュタグがコメント欄に表示する画面を, 図 6 に示す。

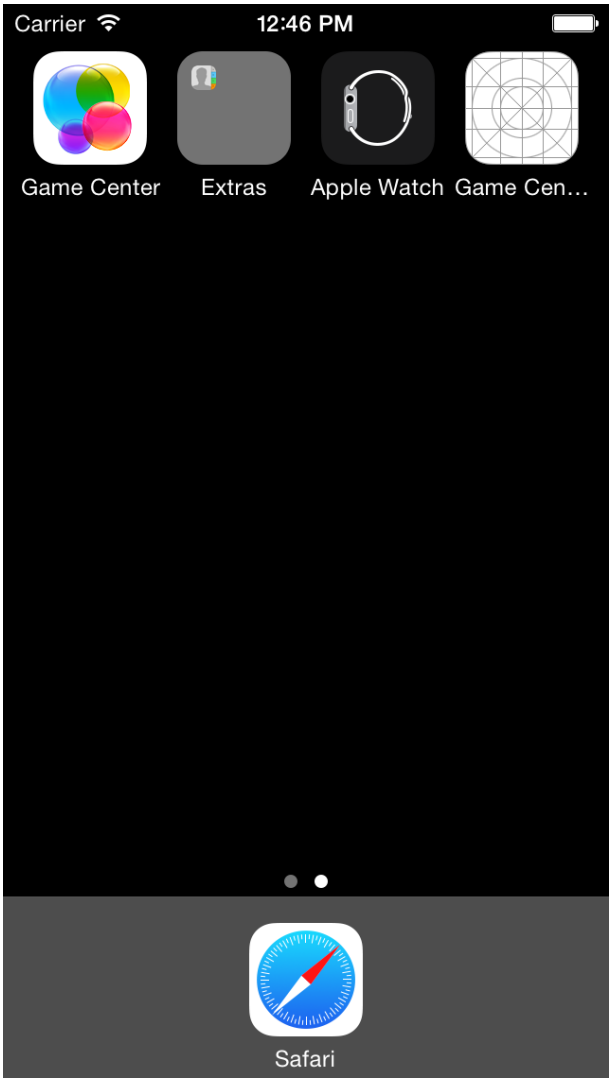


図 1　端末起動時画面。

図 2　アプリ起動画面。

図 1 の画面は端末起動時に表示される。図 1 に表示されている”Game Cen...”のアイコンをタッチする事で、図 2 の画面に移行する。
図 2 の画面にはアプリ名と”検索する”ボタンを配置している。”検索する” ボタンをタッチする事で、現在地の情報を取得し周辺施設を検索する。そして周辺施設を検索した後、周辺施設表示画面の図 3 に移行する。

図 3　周辺施設の中からゲームセンターを表示。

図 4　各店舗毎のツイートを取得し表示する。

図 2 の”検索する”ボタンをタッチする事で現在地の情報を取得した後、FoursquareAPI を使って現在地から周辺施設を図 3 に表示させる。
周辺施設の中から”Arcade”と言うカテゴリでソートをかける事で、ゲームセンターだけを表示。
表示例から”アミューズメントパークエルロフト”を選択して図 4 を表示させる。当該店舗のツイート図 4 に示す。
図 4 の右上のアイコンを選択する事で図 5 の画面を表示させる。

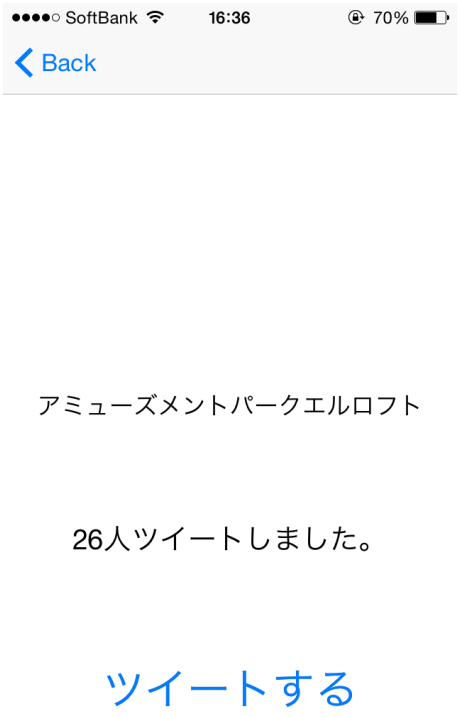


図 5　賑わい度を表示する。

図 6　自動で店名を挿入。

図 5 では当該店舗の賑わい度を表示する。
”アミューズメントパークエルロフト”は図 3 で選択された店舗名を表示し、当該店舗のツイート数を”XX 人ツイートしました”の様に、XX の中にツイート数が入る。

”ツイートする”ボタンをタッチする事で、図 6 を表示させる。

図 6 で Twitter への投稿画面を表示させる。予め選択された店舗の名称と専用のハッシュタグが、投稿フォームに書かれているよう表示させる。

4.2 動作環境

本研究では iPhone アプリケーションの開発を行う。この章では動作環境と開発環境、システム仕様について説明する。

- 開発環境
  - Xcode ver 6.4
  - Mac Book Pro
- 動作環境
  - iPhone5s iOS 8.4
  - iOS simulator

4.3 本研究のアプリの有用性

類似アプリについては検索するだけでも、多くの類似アプリと他に類似したシステムが見つかった。その中には賑わっているゲームセンターを探すようなアプリはなかったが、近いシステムが存在した。”GUILTY GEAR Xrd -REVELATOR-の Twitter 連携システム”は Twitter の自身のアカウントを使って、この「GUILTY GEAR Xrd -REVELATOR」と言うゲーム限定だが、自分がどこのゲームセンター、何連勝中、段位昇格、使用キャラクターを自動でツイートしてくれるシステムである。このシステムの有用性はこのゲームのプレイヤーが一番理解している。自身の主観だが、どこのゲームセンターに行けばいいのかを、自身で決める基準になっていると言っても過言ではないシステムだ。その理由として、このシステムを利用してるユーザーが、このシステムで自動ツイートされた内容に反応している所を Twitter で多々見ることがある事。どのプレイヤーがどのゲームセンターでプレイしているのかを把握できる部分は、本研究で作成するアプリの目的と一致している。尚且つゲーム内の情報ではなくゲームセンター内の情報を、どのくらい人が居るのかを把握できるのが本研究のアプリである。ゲームと言う一つの要素ではなく、ゲームセンターと言う複数の要素を含んだ情報はユーザーにとっては、ゲームのみでは無くさらに価値のある情報になると考える。

5 考察

本研究で提案するアプリによって、Twitter 連携と Foursquare 連携したスマートフォンアプリを作成した。これにより SNS をデータストアとして利用した場合の効果の検証を行うことができた。なお、本研究に対する導入の理解等の箇所は本研究では実施しない。

5.1 利点

本研究で、開発するアプリのデータストアを SNS にする事により生まれた利点と、本研究で作成したアプリの利点を示す。

5.1.1 アプリ間通信の提供

本アプリでは、他ユーザーのツイートを閲覧する最低限の機能を提供した。それによりスマートフォンを用いて遠隔地の環境でも、どこにユーザーが多く居るのかを把握することができるようになった。サーバー等のインフラを、一から作成しなくても、簡易にアプリ間の通信ができるデータストアとして完成することができた。

5.1.2 Foursquare Plugin による 限定した周辺施設の表示

現在地から近い周辺の施設情報を、Fousquare で検索し抽出し、ゲームセンターだけを表示することはできなかったが、50 件 Arcade カテゴリに属する周辺施設を一覧表示した時には 9 割ゲームセンターを表示する事ができた。Foursquare を活用する事で、自身で全国のゲームセンターの名称と場所を、データとして作成しなくても本研究のアプリを完成する事ができた。

5.1.3 データストアの管理

Twitter をデータストアとして活用している間は、自ら管理する範囲が極端に減る。本来サーバーの問題や不具合があると管理者なら復旧作業に当たらなければいけないが、Twitter や Foursquare に依存するがその心配はなくなる。

5.2 問題点

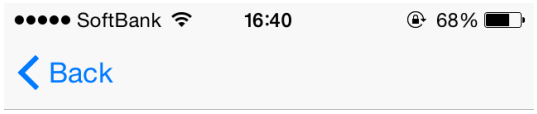
本研究で開発したアプリと、SNS をデータストアとして活用した時の問題点を示す。

5.2.1 TwitterAPI の制限

TwitterAPI の PublicAPI には制限がある。最大検索結果件数が 100 件な為、ユーザーがツイートした内容を 1 度で表示するには 100 件までとなる。ここで問題なのは、101 件目のツイートした場合にはどのように動作するかなのだが、1 番最初のツイートは画面には表示されなくなり 101 件目のツイートは一番上に来る。そして人数のカウントは 100 人でストップしてしまう。この為、101 人目からはカウントされず 1 件目のユーザーのツイートは閲覧できなくなる。過去のツイートに関しては TwitterAPI の制限通り、一週間前までのツイートしか扱えず、一週間以上経ったツイートは検索結果として表示されなった。実際の画面を図 7 に示す。

5.2.2 Foursquare Plugin の機能

Fousquare Plugin の機能でゲームセンター限定で表示するようにしたが、表示したゲームセンターの中に、多人数対戦ゲームが設置しているゲームセンターを表示したいがそれが存在しないゲームセンターも表示してしまった。店舗にアミューズメントスペースが存在する施設がヒットしており、クレーンゲームのみのアミューズ



ラウンドワン 大東店

100人ツイートしました。

ツイートする

図 7　100 件以上のツイートを取得しても 100 と表示される。

メントスペースなどが表示された。実際に、どのゲームセンターに居るのかを選択する際に自身で検証した所、問題はないと考えるが不必要な情報を表示してしまう。

5.2.3　予期しないツイートに対して

Twitter Plugin を使って特定のツイートを検索するシステムだが、フィルタ機能は実装できていない為、悪意あるツイートに対しての考慮をしておらず、これに対しての対処が現在の時点ではできていない。もう一つは非公開アカウントに対してのツイートに関して、Twitter Plugin での検索して情報を取得する事ができず、表示も人数としてカウントできない。

5.2.4　データの読み込み部分

データを読み込むタイミングで問題が二つある。一つは Foursquare Plugin の部分。本研究のアプリを起動時に位置情報を取り、Foursquare Public を使って周辺のゲームセンターを検索する部分はインターネットに繋がってるのと、位置情報を取れた時にしか結果がでてこないので一つ欠けると場所を移動しないといけなくなる。二つ目は、Twitter からデータを取得する為 Twitter Plugin を使用して、大量の検索結果が返ってきた時に著しくアプリの動作が遅くなる。

5.2.5 連携システムの依存性

本研究では、Twitter や Foursquare と連携したシステムを開発し運用している。その為、どちらかのシステムとの連携が切れてしまった場合、このアプリは使用する事ができなくなる。使用する度に、TwitterAPI と FoursquareAPI からの通信を必要とするので、API の制限やバージョンアップによって、現状動作している状態と同じ動作ができるかどうかを保証する事ができない。

5.3 今後の改善点

本研究で開発した、データストアの活用方法の改善点とアプリの改善点を示す。

5.3.1 Twitter Plugin への機能追加

悪意あるツイートに対するフィルタ機能追加、ユーザーに対する通知機能や他ユーザーへの連絡機能を追加していく。

5.3.2 Foursquare Plugin への機能追加

周辺のゲームセンターを表示する画面には、店舗の名称を表示しているだけだがここに、人数が多い順に表示したり各ゲームの種類の賑わい順でソートできる機能があれば、更に本研究のアプリの目的を達成しやすくなるだろう。

5.3.3 データストアからデータの抽出方法

今回は、Twitter の検索機能を使って特定のツイートをデータとして取得していたので、TwitterAPI 制限に規制されることが多々あった。制限の中で問題無く動作する方法を探すのが今後の課題となる。

5.3.4 様々な環境に対応

利用者の環境は様々なので、スマートフォンアプリ以外でも利用が可能にする事が有効だと考える。web システムとして開発する事である程度対応できると考える。

6 結論

本研究では、SNS をデータストアとして扱ったアプリを開発することによって、データストアとしての効果と動作がどのような形になるのか検証する事を行った。実際にスマートフォン向けのアプリを開発し、SNS をデータストアとして運用する事ができた。Twitter Plugin で TwitterAPI を利用してツイートをデータとして扱い、必要なタイミングで呼び出す事で Twitter をデータストアとして利用する事が可能になり、データストアの構築の簡略化の一部として実用できることが分かった。Fousqueare Plugin で FousquareAPI を利用して、周辺の施設の中でゲームセンターを限定して表示する事で、近辺でどのゲームセンターが流行っているのかが可視化できるようになった。本研究の結果で、いくつか制約はあるがその中であれば簡単にアプリ間通信が可能になる事がわかり、さらに簡易のデータストアを作成する事ができる事が分かった。

謝辞

本研究を進めていく上で、大垣 斉講師からご指導及びご協力を頂きました。また、情報教育システム研究室メンバーならびに、卒業生の方々に感謝します。

参考文献

[1] gamecenter. http://www.nikkei.com/article/DGXMZO91164740R30C15A8000000/.

[2] Foursquareapi. https://developer.foursquare.com/.

[3] Twitterapi 使い方まとめ. https://syncer.jp/twitter-api-matome.

付録 A　ソースコード

A.1　AppDelegate.h

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

@interface AppDelegate : UIResponder <UIApplicationDelegate>

@property (strong, nonatomic) UIWindow *window;

@end
```

A.2　AppDelegate.m

```
#import "AppDelegate.h"

@interface AppDelegate ()

@end

@implementation AppDelegate


- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOp
    return YES;
}

- (void)applicationWillResignActive:(UIApplication *)application {

}

- (void)applicationDidEnterBackground:(UIApplication *)application {

}

- (void)applicationWillEnterForeground:(UIApplication *)application {

}

- (void)applicationDidBecomeActive:(UIApplication *)application {
```

```
}

- (void)applicationWillTerminate:(UIApplication *)application {

}

@end
```

A.3 ViewController.h

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

@interface ViewController : UIViewController


@end
```

A.4 ViewController.m

```
#import "ViewController.h"

@interface ViewController ()

@end

@implementation ViewController

- (void)viewDidLoad {
    [super viewDidLoad];

}

- (void)didReceiveMemoryWarning {
    [super didReceiveMemoryWarning];

}

@end
```

A.5 FoursquareTableViewController.h

```
#import <UIKit/UIKit.h>
#import <CoreLocation/CoreLocation.h>
#import "PostViewController.h"

@interface FoursquareTableViewController : UITableViewController<CLLocationManagerDelegate>
{
    //次の画面へ渡す引数
    NSString *_arguments;
    NSMutableArray *venues_;
    CLLocationManager *locationManager_;
}
@property (nonatomic) NSString *arguments;

@end
```

A.6 FoursquareTableViewController.m

```
#import "FoursquareTableViewController.h"
#import <CoreLocation/CoreLocation.h>
#import "PostViewController.h"
#import "SearchViewController.h"

@interface FoursquareTableViewController ()

@end

@implementation FoursquareTableViewController

- (void)viewDidLoad {
    [super viewDidLoad];
    if (nil == locationManager_) {
        locationManager_ = [[CLLocationManager alloc] init];
        // iOS 8以上
        if ([[[UIDevice currentDevice] systemVersion] floatValue] >= 8.0) {
  // NSLocationWhenInUseUsageDescriptionに設定したメッセージでユーザに確認
            [locationManager_ requestWhenInUseAuthorization];
            // NSLocationAlwaysUsageDescriptionに設定したメッセージでユーザに確認
            //[locationManager requestAlwaysAuthorization];
        }
    }
```

```
    [locationManager_ setDelegate:self];
    [locationManager_ startUpdatingLocation];
}

- (void)didReceiveMemoryWarning {
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}

#pragma mark - Table view data source

- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {
    return 1;
}

- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section {
    return [venues_ count];
}

//セルが選択されたとき
- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
    NSLog(@"「%@」が選択されました", [[venues_ objectAtIndex:indexPath.row] objectForKey:@"name"]);
    NSString *str = [[venues_ objectAtIndex:indexPath.row] objectForKey:@"name"];
    _arguments = str;
    //次の画面へ遷移
    [self performSegueWithIdentifier:@"toPost" sender:self];
}

//画面遷移時に呼ばれるメソッド
- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender {
    //2つ目の画面にパラメータを渡して遷移する
    if ([segue.identifier isEqualToString:@"toPost"]) {
        //ここでパラメータを渡す
//        PostViewController *postViewController = segue.destinationViewController;
//        postViewController.arguments = _arguments;
        SearchViewController *searchViewController = segue.destinationViewController;
        searchViewController.arguments = _arguments;
    }
}

- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
```

```
    NSString *cellIdentifier = @"Cell";
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:cellIdentifier];

    // セルが作成されていないか?
    if (!cell) { // yes
        // セルを作成
        cell = [[UITableViewCell alloc]initWithStyle:UITableViewCellStyleDefault reuseIdentifier:cellId
    }
    // セルにテキストを設定
    cell.textLabel.text = [[venues_ objectAtIndex:indexPath.row] objectForKey:@"name"];
    return cell;
}

- (void)locationManager:(CLLocationManager *)manager didUpdateLocations:(NSArray *)locations
{
    CLLocation* location = [locations lastObject];
    CLLocationDegrees latitude = location.coordinate.latitude; // 緯度
    CLLocationDegrees longitude = location.coordinate.longitude; // 経度

    //API からリスト作成
    NSString *urlString = [NSString stringWithFormat:@"https://api.foursquare.com/v2/venues/search?ll=%
    //HTTP から NSURL 作成
    NSURL *url = [NSURL URLWithString:urlString];
    NSString *response = [NSString stringWithContentsOfURL:url encoding:NSUTF8StringEncoding error:nil]
    NSData *jsonData = [response dataUsingEncoding:NSUTF32BigEndianStringEncoding];
    NSDictionary *jsonDic = [NSJSONSerialization JSONObjectWithData:jsonData options:0 error:nil];

    // エラーコードをログに出力
    NSInteger errorCode = [[[jsonDic objectForKey:@"meta"] objectForKey:@"code"] intValue];
    NSLog(@"errorCode = %ld", (long)errorCode);

    // 結果取得
    NSArray *venues = [[jsonDic objectForKey:@"response"] objectForKey:@"venues"];
    NSLog(@"all = %@",venues);
    venues_ = [venues mutableCopy];
    [self.tableView reloadData];
    [locationManager_ stopUpdatingLocation];
}

@end
```

A.7 PostViewController.h

```
//
//  PostViewController.h
//  Production
//
//  Created by honda masayuki on 2015/10/29.
//  Copyright (c) 2015年 honda masayuki. All rights reserved.
//

#import <UIKit/UIKit.h>

@interface PostViewController : UIViewController{
    //前の画面から受け取る引数
    NSString *_arguments;
    NSString *_result;

}
@property (nonatomic) NSString *arguments;
@property (weak, nonatomic) IBOutlet UILabel *ArcadeName;
@property (weak, nonatomic) IBOutlet UILabel *people;


@end
```

A.8 PostViewController.m

```
#import <Social/Social.h>
#import <Accounts/Accounts.h>
#import "PostViewController.h"
#import "SearchViewController.h"

@interface PostViewController ()

@end

@implementation PostViewController
@synthesize arguments = _arguments;

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

```
    NSString *str = @"人ツイートしました。";
    //ラベルに前の画面から受け取った引数を表示
    NSString *text = _arguments;
    self.ArcadeName.text = text;
    SearchViewController *searchViewController = [SearchViewController initMyClass];
    NSString *pepole = [searchViewController myMethod];
    NSLog(@"%@", pepole);
    if(pepole == nil){
        NSString *inai = 0;
        inai = [inai stringByAppendingString:str];
        self.people.text = inai;
    }else{
    pepole = [pepole stringByAppendingString:str];
    self.people.text = pepole;
    }
}

- (void)didReceiveMemoryWarning {
    [super didReceiveMemoryWarning];
}

- (IBAction)Tweetbutton:(id)sender {
    NSString *str1 = @"#Test_";
    //文字列を結合
    SLComposeViewController *twitterPostVC = [SLComposeViewController composeViewControllerForServiceTy
    // 文字列を置換
    NSString *mergeStr = [str1 stringByAppendingString:_arguments];
    NSString *result = [mergeStr stringByReplacingOccurrencesOfString:@" " withString:@"_"];
    [twitterPostVC setInitialText:result];
    [self presentViewController:twitterPostVC animated:YES completion:nil];

}


@end
```

A.9 SearchViewController.h

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

@interface SearchViewController : UITableViewController{
    NSString *_arguments;
    NSString *_result;
```

```
}
// クラスメソッドの宣言
+ (id)initMyClass;
// インスタンスメソッドの宣言
- (NSString *) myMethod;
- (void)viewDidLoad;

// User twitter search string
@property (nonatomic) NSString *arguments;
@property (nonatomic, copy) NSString *query;
@property (nonatomic) NSString *result;

@end
```

A.10 SearchViewController.m

```
#import "FoursquareTableViewController.h"
#import "SearchViewController.h"
#import <Accounts/Accounts.h>
#import <Social/Social.h>

static NSString *pepole;

// Various twitter request states
typedef NS_ENUM(NSUInteger, UYLTwitterSearchState)
{
    UYLTwitterSearchStateLoading,
    UYLTwitterSearchStateNotFound,
    UYLTwitterSearchStateRefused,
    UYLTwitterSearchStateFailed
};

@interface SearchViewController ()

// Twitter search variables
@property (nonatomic,strong) NSURLConnection *connection;
@property (nonatomic,strong) NSMutableData *buffer;
@property (nonatomic,strong) NSMutableArray *results;
@property (nonatomic,strong) ACAccountStore *accountStore;
@property (nonatomic,assign) UYLTwitterSearchState searchState;

// Active dictionary used to save selected/deselected tweets
@property (nonatomic,strong) NSMutableDictionary *selectedTweetData;
```

```
// Ensures user only sees funtionality description once
@property (nonatomic,assign) BOOL firstSearch;

@end

@implementation SearchViewController
@synthesize arguments = _arguments;

+ (id)initMyClass {
    return [[self alloc] init];
}
- (ACAccountStore *)accountStore
{
    if (_accountStore == nil)
    {
        _accountStore = [[ACAccountStore alloc] init];
    }
    return _accountStore;
}

- (NSString *)searchMessageForState:(UYLTwitterSearchState)state
{
    // Handle all connection cases
    switch (state)
    {
        case UYLTwitterSearchStateLoading:
            return @"Loading...";
            break;
        case UYLTwitterSearchStateNotFound:
            return @"No results found";
            break;
        case UYLTwitterSearchStateRefused:
            return @"Twitter Access Refused";
            break;
        default:
            return @"Not Available";
            break;
    }
}

#pragma mark -
#pragma mark === View Setup ===
#pragma mark -
```

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    _firstSearch = false;
    _selectedTweetData = [NSMutableDictionary new];
    pepole = @"0";
    // Save search string and query twitter
    self.title = self.result;
    [self loadQuery];
}

- (void)viewWillDisappear:(BOOL)animated
{
    [super viewWillDisappear:animated];
    [self cancelConnection];
}

- (void)dealloc
{
    [self cancelConnection];
}

- (BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)interfaceOrientation
{
    return YES;
}

//画面遷移時に呼ばれるメソッド
- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender {
    //2つ目の画面にパラメータを渡して遷移する
    if ([segue.identifier isEqualToString:@"toPost"]) {
        //ここでパラメータを渡す
        PostViewController *postViewController = segue.destinationViewController;
        postViewController.arguments = _arguments;
    }
}

- (NSManagedObjectContext *)managedObjectContext {
    NSManagedObjectContext *context = nil;

    id delegate = [[UIApplication sharedApplication] delegate];
    if ([delegate performSelector:@selector(managedObjectContext)]) {
```

```
        context = [delegate managedObjectContext];
    }

    return context;
}

#pragma mark -
#pragma mark === UITableViewDataSource Delegates ===
#pragma mark -

- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
{
    return 1;
}

- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section
{
    // Setup table. Either the search has returned results, or the table contains once description cell
    NSUInteger count = [self.results count];
    return count > 0 ? count : 1;
}

- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    // Loading cell is the placeholder cell while the query connects to twitter
    static NSString *LoadCellIdentifier = @"LoadingCell";
    static NSString *ResultCellIdentifier = @"ResultCell";
    NSUInteger count = [self.results count];
    if ((count == 0) && (indexPath.row == 0))
    {
        // No search results and first cell. Display connect state
        UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:LoadCellIdentifier];
        cell.textLabel.text = [self searchMessageForState:self.searchState];
        return cell;
    }

    // Populate table with resultant tweet contents
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:ResultCellIdentifier];
    NSDictionary *tweet = (self.results)[indexPath.row];

    // Parse through the tweet for image URL and save to UIImage
    NSURL *url = [NSURL URLWithString:[[tweet[@"user"]valueForKey: @"profile_image_url"] stringByAdding
    NSData *data = [NSData dataWithContentsOfURL : url];
```

```
    UIImage *image = [UIImage imageWithData: data];

    // Setup cell
    cell.textLabel.text = tweet[@"text"];
    cell.imageView.image = image;

    if([_selectedTweetData objectForKey:@(indexPath.row).stringValue])
    {
        cell.accessoryType = UITableViewCellAccessoryCheckmark;
    }
    else
        cell.accessoryType = UITableViewCellAccessoryNone;

    //NSLog(@"%ld",(long)count);
    pepole = [NSString stringWithFormat:@"%ld", count];

    NSLog(@"%@",pepole);
    return cell;
}

// インスタンスメソッドの実装
- (NSString *) myMethod{
    if(pepole == nil){
        pepole = @"0";
        NSLog(@"NULL あるで");
    }
    return pepole;
}

- (void)tableView:(UITableView *)tableView willDisplayCell:(UITableViewCell *)cell forRowAtIndexPath:(N
{
    cell.backgroundColor = [UIColor whiteColor];
}

- (CGFloat)tableView:(UITableView *)tableView heightForRowAtIndexPath:(NSIndexPath *)indexPath
{
    // Set cell height to handle larges tweet size. Plus the uniformity looks better
    return 84;
}

#pragma mark -
#pragma mark === Private methods ===
#pragma mark -
```

```
#define RESULTS_PERPAGE @"100"

- (void)loadQuery
{
    // Set searchState and begin twitter query
    self.searchState = UYLTwitterSearchStateLoading;
    NSString *str1 = @"#Test_";
    NSString *mergeStr = [str1 stringByAppendingString:_arguments];
    NSString *result = [mergeStr stringByReplacingOccurrencesOfString:@" " withString:@"_"];

    ACAccountType *accountType = [self.accountStore accountTypeWithAccountTypeIdentifier:ACAccountTypeI
    [self.accountStore requestAccessToAccountsWithType:accountType
                                               options:NULL
                                            completion:^(BOOL granted, NSError *error)
     {
         if (granted)
         {
             // User is logged in/has internet. load twitter search url and begin tweet search
             NSURL *url = [NSURL URLWithString:@"https://api.twitter.com/1.1/search/tweets.json"];
             NSDictionary *parameters = @{@"count" : RESULTS_PERPAGE,
                                          @"q" : result};

             SLRequest *slRequest = [SLRequest requestForServiceType:SLServiceTypeTwitter
                                                       requestMethod:SLRequestMethodGET
                                                                 URL:url
                                                          parameters:parameters];

             NSArray *accounts = [self.accountStore accountsWithAccountType:accountType];
             slRequest.account = [accounts lastObject];
             NSURLRequest *request = [slRequest preparedURLRequest];
             dispatch_async(dispatch_get_main_queue(), ^{
                 self.connection = [[NSURLConnection alloc] initWithRequest:request delegate:self];
                 [UIApplication sharedApplication].networkActivityIndicatorVisible = YES;
             });
         }
         else
         {
             // Twitter search unsuccesful, update search state and reload
             self.searchState = UYLTwitterSearchStateRefused;
             dispatch_async(dispatch_get_main_queue(), ^{
                 [self.tableView reloadData];
             });
```

```
        }
    }];
}

// Connection response functions from twitter search completion handler
- (void)connection:(NSURLConnection *)connection didReceiveResponse:(NSURLResponse *)response
{
    self.buffer = [NSMutableData data];
}

- (void)connection:(NSURLConnection *)connection didReceiveData:(NSData *)data {

    [self.buffer appendData:data];
}

// Connection completed.
- (void)connectionDidFinishLoading:(NSURLConnection *)connection
{
    [UIApplication sharedApplication].networkActivityIndicatorVisible = NO;
    self.connection = nil;

    NSError *jsonParsingError = nil;
    NSDictionary *jsonResults = [NSJSONSerialization JSONObjectWithData:self.buffer options:0 error:&js

    // Save statuses for table use
    self.results = jsonResults[@"statuses"];
    if ([self.results count] == 0)
    {
        // Search Unsuccessful. Update search state variable
        NSArray *errors = jsonResults[@"errors"];
        if ([errors count])
        {
            self.searchState = UYLTwitterSearchStateFailed;
        }
        else
        {
            self.searchState = UYLTwitterSearchStateNotFound;
        }
    }

    // Reset and reload regardless of success/failure
    self.buffer = nil;
    [self.refreshControl endRefreshing];
```

```
    [self.tableView reloadData];
    [self.tableView flashScrollIndicators];
}

// Connection failed. Set search state and reset connection variables
- (void)connection:(NSURLConnection *)connection didFailWithError:(NSError *)error
{
    [UIApplication sharedApplication].networkActivityIndicatorVisible = NO;
    self.connection = nil;
    self.buffer = nil;
    [self.refreshControl endRefreshing];
    self.searchState = UYLTwitterSearchStateFailed;

    [self handleError:error];
    [self.tableView reloadData];
}

// Notify user of the issue
- (void)handleError:(NSError *)error
{
    NSString *errorMessage = [error localizedDescription];
    UIAlertView *alertView = [[UIAlertView alloc] initWithTitle:@"Connection Error"
                                                        message:errorMessage
                                                       delegate:nil
                                              cancelButtonTitle:@"OK"
                                              otherButtonTitles:nil];
    [alertView show];
}

- (void)cancelConnection
{
    if (self.connection != nil)
    {
        [UIApplication sharedApplication].networkActivityIndicatorVisible = NO;
        [self.connection cancel];
        self.connection = nil;
        self.buffer = nil;
    }
}

@end
```

A.11 TopViewController.h

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

@interface TopViewController : UIViewController

@end
```

A.12 TopViewController.m

```
#import "TopViewController.h"

@interface TopViewController ()

@end

@implementation TopViewController

- (void)viewDidLoad {
    [super viewDidLoad];
    // Do any additional setup after loading the view.
}

- (void)didReceiveMemoryWarning {
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}

/*
#pragma mark - Navigation

// In a storyboard-based application, you will often want to do a little preparation before navigation
- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender {
    // Get the new view controller using [segue destinationViewController].
    // Pass the selected object to the new view controller.
}
*/

@end
```