

# TwitrArcade

*by* Andrew John Rogers

---

|                |                                                                      |                 |       |
|----------------|----------------------------------------------------------------------|-----------------|-------|
| FILE           | 17743_ANDREW_JOHN_ROGERS_TWITRARCADE_78196_1697238982.DOCX (559.02K) |                 |       |
| TIME SUBMITTED | 20-APR-2016 11:35AM                                                  | WORD COUNT      | 13236 |
| SUBMISSION ID  | 56111647                                                             | CHARACTER COUNT | 81486 |



---

PRIFYSGOL CYMRU

**Y Drindod Dewi Sant**

UNIVERSITY OF WALES

**Trinity Saint David**

---

SWANSEA - ABERTAWE

## **TwtrArcade**

By

Andrew Rogers

Supervisor: Dr Nik Whitehead

Project Submitted as part of the requirements for the award of BSc(Hons) Software  
Engineering

April 2016

## **DECLARATION**

I, Andrew John Rogers declare that I am the sole author of this Project; that all references cited have been consulted; that I have conducted all work of which this is a record, and that the finished work lies within the prescribed word limits.

This work has not previously been accepted as part of any other degree submission.

*Signed :*

*Date :*

## FORM OF CONSENT

I Andrew John Rogers hereby consent that my Project, submitted in candidature for the B.Sc.(Hons) Software Engineering degree, if successful, may be made available for inter-library loan or photocopying (subject to the law of copyright), and that the title and abstract may be made available to outside organisations.

*Signed :*

*Date :*

## **Abstract**

The following project will discuss the process of creating a game that can be played through twitter without the need of any third party client or website. This will include looking at the history of twitter and how it has changed over the years as well as looking at already existing twitter games.

This report will also walkthrough the design and development stages from the start where aims and objectives are defined all the way through until the end where the software is tested using various techniques.

Upon completion of the project, there will be an evaluation of my findings outlining any issues with both the project as well as twitter as a gaming platform as a whole.

## Contents Page

|                                                     |    |
|-----------------------------------------------------|----|
| DECLARATION .....                                   | 2  |
| FORM OF CONSENT .....                               | 3  |
| Abstract .....                                      | 4  |
| Contents Page .....                                 | 5  |
| Table of Figures .....                              | 7  |
| Table of Tables .....                               | 8  |
| Introduction .....                                  | 9  |
| Literature Review .....                             | 10 |
| Twitter .....                                       | 10 |
| Twitter Profiles .....                              | 11 |
| Tweet .....                                         | 13 |
| Different Twitter Users .....                       | 13 |
| Twitter Uses .....                                  | 14 |
| Casual Gaming .....                                 | 15 |
| What is casual gaming? .....                        | 15 |
| Casual gaming and social media .....                | 15 |
| Pre-existing Twitter Games .....                    | 15 |
| What makes a game playable on Twitter .....         | 16 |
| Twitter vs. Facebook .....                          | 17 |
| Why is Facebook more Popular for Social Games ..... | 17 |
| Possible Games .....                                | 18 |
| Reading Material .....                              | 19 |
| Twitter Terms of Service .....                      | 19 |
| TweetInvi API Documentation .....                   | 19 |
| Aims & Objectives Revised .....                     | 20 |
| Research Methodology .....                          | 20 |
| What research method will be used .....             | 20 |
| Approach Justification .....                        | 20 |
| Advantages .....                                    | 20 |
| Development Methodology .....                       | 22 |
| Why Feature driven development will be used .....   | 22 |

|                                          |    |
|------------------------------------------|----|
| Possible Disadvantages .....             | 23 |
| Tools require to ensure completion ..... | 24 |
| Design and Implementation.....           | 25 |
| About the Project.....                   | 25 |
| Moscow List .....                        | 26 |
| Time boxes .....                         | 28 |
| Timebox 1.....                           | 28 |
| Timebox 2.....                           | 34 |
| Timebox 3.....                           | 41 |
| Timebox 4.....                           | 46 |
| Implementation .....                     | 51 |
| Problems and successes.....              | 73 |
| Testing.....                             | 74 |
| Usability Testing.....                   | 74 |
| Acceptance Testing .....                 | 79 |
| Evaluation .....                         | 80 |
| Conclusions and Recommendations .....    | 81 |
| Reflection .....                         | 82 |
| References .....                         | 83 |
| Appendix .....                           | 86 |
| Installation .....                       | 86 |
| Setting up the database .....            | 86 |
| General System Setup .....               | 86 |

## Table of Figures

|                                                  |    |
|--------------------------------------------------|----|
| Figure 1 - Send Tweet .....                      | 29 |
| Figure 2 - Receive Tweet.....                    | 30 |
| Figure 3- Testing receive a tweet.....           | 31 |
| Figure 4 - Testing send a tweet .....            | 31 |
| Figure 5 - Register use case.....                | 35 |
| Figure 6 - Twitter APP details.....              | 36 |
| Figure 7 - Testing new receive function.....     | 37 |
| Figure 8 - Testing new send tweet function ..... | 38 |
| Figure 9 - Updated Register usecase.....         | 42 |
| Figure 10 - Equip item usecase .....             | 43 |
| Figure 11 - Battle use case .....                | 47 |

## Table of Tables

|                                                          |    |
|----------------------------------------------------------|----|
| Table 1 - Functional Requirements MUST .....             | 26 |
| Table 2 - Functional Requirements SHOULD .....           | 26 |
| Table 3 - Functional Requirements COULD.....             | 26 |
| Table 4 - Functional Requirements WONT .....             | 26 |
| Table 5 - Non-Functional Requirements WONT .....         | 27 |
| Table 6 - Send Tweet Use Case .....                      | 29 |
| Table 7- Receive Tweet Use Case.....                     | 30 |
| Table 8 - Timebox 1 Testing .....                        | 32 |
| Table 9 - Receive Tweet With Database Intergration ..... | 35 |
| Table 10 - Timebox 2 Testing .....                       | 39 |
| Table 11 - Character Generation Use Case .....           | 42 |
| Table 12 - Inventory Management Use Case .....           | 43 |
| Table 13 - Timebox 3 Testing .....                       | 44 |
| Table 14 - Player Battle Use Case.....                   | 47 |
| Table 15 - Loot Generation Use Case.....                 | 48 |
| Table 16 - Timebox 4 Testing .....                       | 49 |
| Table 17 - Usability Testing User 1.....                 | 75 |
| Table 18 - Usability Testing User 2.....                 | 76 |
| Table 19 - Usability Testing User 3.....                 | 77 |

## **Introduction**

This project is about creating a game that can be played through twitter without the need of having access to any third party client or website and by doing so should allow users to play an engaging and 'fun' game anywhere they have access to twitter.

The overall aims and objectives for this project are to create a variety of different games such as blackjack that they can play with their friends.

Other content in this report will discuss the different research carried out as well at literature that has been reviewed, this is so that you can get a fully understand of how the project proceeded as well as any issues or findings that were found along the way of development.

This project will also contain all of the design, implementation and testing aspects to show the process of how it went from the defined requirements right up until being implemented and tested.

## Literature Review

### Twitter

Twitter, is according to their official website [1] an information network which is made up of 140 character messages called tweets. According to their website [1] twitter is useful because it contains information that each individual would find useful and even compares its services to that of a newspaper where you get the scoop in real time.

Twitter has a rather interesting history, according to [2] the first tweet was sent out by the founder of twitter Jack Dorsey on March 21<sup>st</sup> 2006 which said "Just setting up my twttr" [3] which at the time of this publication has 66,100 retweets and 45,767 likes.

Probably the next biggest milestone for twitter was in April 2007 where twitter separated from the original company "Obvious Corp" and became a separate entity [4]. Shortly after, in august 2007 twitter would receive a huge innovation known as the Hashtag, this idea was proposed by a twitter employee known by the name of Chris Messina[4]. He did this by tweeting "how do you feel about using # (pound) for groups. As in #barcamp [msg]?"[5].

Although twitter first introduced the hashtag, the creator Chris Messina decided not to patent the idea, he states that the reasons for not patenting the hashtag are that he believed that[6]

"claiming a government-granted monopoly on the use of hashtags would have likely inhibited their adoption, which was the antithesis of what I was hoping for, which was broad-based adoption and support — across networks and mediums."

And also that he had no interest in making any money directly from the hashtags as it was something that was born from the internet and because of this should not be owned by a single person. He also states that [6]

"The value and satisfaction I derive from seeing my funny little hack used as widely as it is today is valuable enough for me to be relieved that I had the foresight not to try to lock down this stupidly simple but effective idea."

Before twitter, there wasn't really an easy way to get out your information to a large audience, this was especially apparent in January 2009 where a US Airways plane crashed

in the Hudson river and one twitter user tweeted about the incident before traditional media even knew about it [4].

In September 2009 Dick Costolo joined the twitter team as COO where he would later become CEO of twitter in 2010 and remained so until 2015 [7].

A few months later, in 2010 it soon became apparent that twitter was starting to gear towards businesses and promoting things. In April they introduced Promoted tweets [8] which are tweets that are influenced by companies, an example that they give on the website is redbull.

Around the same time as Promoted tweets twitter granted access to its archive of tweets to the library of congress, the largest library in the world [9].

A little later, to expand on the Promoted tweet feature; twitter then expanded their services to put promoted tweets on trending topics [10]. This is where a company would be able to promote something such as a new movie and then it would be put on the side of the website next to all of the other popular tweets so that is more likely to get attention.

Finally, the last thing that twitter introduced in regards to promoting is promoted accounts [11]. The way that promoted accounts work is by looking at the users public list of followers and see what types of people they are following and then promote accounts of similar taste which they are not already following.

### **Twitter Profiles**

Now that the major events in the history of twitter have been explained, it is time to move on to twitter and its current state as a social media network.

Twitter is a social networking platform that works by allowing users to create a 'Profile' and then communicate with other people using 'Tweets'.

A twitter profile is a page that contains various different elements that the user can modify such as display name, description, location and even a profile picture to allow them to be easily identified by other users.

It is possible for users to 'Follow' other users, this can be done by simply visiting the profile of the person that you would like to follow and then press the button at the top right of their page.

By following a user's twitter profile, it will ensure that you are notified whenever they send out a tweet; this is done both via email as well as displaying it in the feed section of your own twitter page.

**Tweet**

A tweet is a message that a user can send out to either specific users or as a general message without being directed at someone. Something important to note with a tweet is that there is a 140 character limit to the message length; this is because of how twitter was originally implemented.

As mentioned above, you are able to direct tweets at different users; this can be done by simply putting the @ symbol followed by the users ID for example @TwitrArcade would direct a message to the TwitterArcade twitter account.

As well as being able to direct messages, another feature that a tweet has to offer is the ability to set a subject line (kind of) this can be done by using the # symbol followed by what you want to call the subject for example #TwitterArcade. By using this, it allows users to find related tweets by anyone who used the same 'subject'.

**Different Twitter Users**

There are a few different types of twitter users but generally they can be categorised into 3 different categories.

The first is just the general user, these are regular people who use twitter to contact their friends, and generally just browse what is going on.

The second type of twitter user is celebrities; these twitter accounts are of high profile people such as politicians, singers or actors. Originally, these twitter accounts were not run by the celebrity themselves but instead had someone do it for them however it is gradually becoming more common for celebrities to control their own twitter account.

The final type of twitter account is a business twitter account, unlike the previous 2 options this twitter account is not focused on a single person but instead a business for example Microsoft.

### **Twitter Uses**

As discussed above, there are different types of twitter users and as such the way that twitter is used will vary depending on the type of user.

#### ***General User***

As a regular user, the main interested with twitter is being able to communicate with people all around the world that would not normally be possible without a social media platform such as twitter.

As well as being able to follow people and find out what's happening in people's lives; twitter can also be a news outlet to find out what events are happening in the world.

#### ***Celebrities***

Celebrities (assuming that they are running their own twitter account) get the same benefits as a regular twitter user would, however they also have an opportunity to better themselves in regards to their career, generally a celebrity will have a relatively large following which means that it is a great opportunity to communicate with their fans as well as promote any events that they have upcoming such as tour dates ect.

#### ***Business***

Twitter has many uses that would benefit almost any business that adapts the social media platform. One of which being able to directly get feedback from customers, by allowing customers to easily communicate their thoughts about a specific product or service then it can be greatly beneficial for the business.

As well as gaining customer feedback, twitter is often used as a support medium in which customer can gain advice from the business about a particular product or situation, the advantage of doing it this way over say their personal website is that usually the tweets will be public so that people that are having the same issues might be able to find a solution without actually needing to contact support and wait for a response.

## **Casual Gaming**

### **What is casual gaming?**

A casual game is a game that can be played by all different types of people regardless of their previous background with gaming. A casual game can be any genre of game given that they are simple to play and don't require much time to be invested into them [12].

A desirable feature of a casual game is to allow the users to be able to play in short bursts, this is because casual gamers don't usually have a lot of spare time to invest into the games but instead are usually something they do between tasks for example commuting to work on the train.

Something else that a casual game requires is that the game can be completed relatively quickly or for the game to have the ability to be played continuously without the need to save, a good example of this could be temple run.

### **Casual gaming and social media**

Casual gaming has become increasingly popular with social media, particularly with Facebook. Facebook has the ability to allow game developers to host and integrate their games onto the social networking platform allow players to not only play these casual games but also socially interact with their friends whilst doing so. Although twitter is quite a bit different from Facebook but this does not mean that casual games do not have its place on this social networking platform [13].

### **Pre-existing Twitter Games**

Although there are already existing games on twitter such as hangman and various trivia based games there isn't really a huge variety available and most of which many twitter users won't find engaging [14].

**What makes a game playable on Twitter**

From looking at the few examples that there are of existing twitter games it is obvious to see that there are various limitations in how a game can be played on twitter and these factors will need to be considered when determining if a game is viable to be played on twitter or not.

The two main factors that decide on if a game is viable as a twitter game is both the complexity (mainly gameplay) and time.

If the game is too complex then it will be very difficult to integrate as a fun twitter experience, for example it wouldn't be very reasonable to turn something like call of duty into a twitter game simply because the controls would be too much. A twitter games controls need to be simple for example chess, chess controls are simple because you just move a chess piece from one location to another.

As for the time factor, due to the way that the game would be implemented into twitter the only games that would work are non-real time games.

A final factor which could affect the viability of a game is how complex the graphics are, the reason for this is that twitter has a maximum upload size and if the graphics required to display to the user is bigger than this size (and it's not possible to reduce it) then the game would not be viable.

## **Twitter vs. Facebook**

There has always been a battle between the social networking giants Twitter and Facebook but this report will only outline any details relevant to games playable on the social networking sites.

As discussed above, there are very few games when it comes to the social networking platform Twitter; however Facebook is quite the opposite.

It would be nearly impossible to count every single game that is available on Facebook, however according to Appdata.com [17], at the time of this there are 4680 apps with at least 10,000 players.

### **Why is Facebook more Popular for Social Games**

The reason that Facebook is more popular when it comes having social games and users play games on their social networking site could be for a few different reasons. Twitter seems to be more focused around celebrities and world events rather than close friends and family like Facebook is; this isn't the main reason behind the lack of social games however.

Twitter simply doesn't provide support for creating games for their social networking website, unlike Facebook that has an entire developer process that lets you sign up and register your app on their website, twitter games can only be made using the basic functions that twitter provides to every user.

### **Possible Games**

From looking at what makes a game playable on twitter there are a few different ideas that become obvious choices and will work effectively on twitter.

The obvious choices that come to mind are the traditional game such as hangman, twitter, and battleships. Although these game ideas are quite possible and would work effectively they are also rather boring and standard.

The more unique ideas are things such as blackjack, higher or lower and even a simple RPG style game.

## **Reading Material**

To be able to complete this project, there are 2 main things that will be looked into and understood.

### **Twitter Terms of Service**

Although twitter has a user terms of service (guidelines that twitter outlines and that each user agrees to upon signing up to a twitter account), a developer must also abide by developer terms of service too.

Generally the terms of service state ethical issues such as not manipulate tweets in anyway, for example if you are displaying a tweet that someone has said then you may not attempt to modify it as well as respecting users privacy settings and not encouraging publishing personal information [18].

As one of the possible game ideas had a form of gambling, it was important to also read into twitters gambling policy. At the very start of twitters gambling policy it states that "Twitter prohibits the promotion of gambling content, except for campaigns targeting specified countries where it is allowed with restrictions as explained below." [19].

### **TweetInvi API Documentation**

TweetInvi is the API that will be used to implement the end product, the official TweetInvi website [20] has a link to a github repository that has downloads to the TweetInvi API as well as written examples [21].

Although there was a link to the documentation, it seems that although the API it's self was continued to be updated the documentation on the API was not, this could prove problematic when it comes to implementing the API.

## **Aims & Objectives Revised**

The overall aims and objectives have pretty much stayed the same even after the research; however after reading the twitter terms of service it is clear that the various gambling games mentioned might not be viable and is a good idea to stay away from them.

Objectives:

- Create a system that will communicate with the TwitrArcade Twitter account.
- The system will allow new users to register with the service so they may play the various games.
- The system will grant the ability for registered users to participate in games.
- The system will keep track of various player stats.
- The system will provide 'rewards' to winners.

## **Research Methodology**

### **What research method will be used**

The research approach that will be used for this project is the Deductive approach, by following the deductive it allows for the use of experiments, software development and document review. The process is that the literature review is completed before continuing onto the software development stage before finally the experiment. The software development part of it will be intertwined with the development methodology which describes the plan that the development phase will follow in order to be successful. The experiment part of the project is a way to test how successful the project has been [15].

### **Approach Justification**

Due to the nature of the project it is believe that the most appropriate approach would be a positivist approach using a deductive philosophy. Through the use of the deductive philosophy it means that there is an opportunity where theories can be presented and tested [16].

### **Advantages**

Through the use of literature review, it means that project that are similar to TwitrArcade can be researched and investigated to see what kind of things are required to ensure TwitrArcade's success [16].



## **Development Methodology**

The particular development methodology that will be used for the use of this project is Feature Driven Development (FDD). Feature driven development is best suited for this particular type of project. When using Feature Driven Development there are five stages or steps that are taken, these are as follows:

1. Develop Overall Model
2. Build Feature List
3. Plan by Feature
4. Design by Feature
5. Build Feature

These stages are repeated for each feature of the entire system until the entire project is completed.

## **Why Feature driven development will be used**

The main reason for using feature driven development for this project is that it allows the use of the MoSCoW list. By using the MoSCoW list it outlines the different features as well as their importance to the system this allows for priority development.

MoSCoW stands for:

- Must have – This is what the system MUST have.
- Should Have – This is what the system SHOULD have but will continue to function if not implemented.
- Could Have – This is what the system COULD have if there is enough time left to implement them
- Won't Have – this is what the system WON'T have.

As well as being able to implement the MoSCoW list and being able to prioritize the required features, it also means that communication between consumers and developer is more common ensuring that the consumers are happy with the direction the project is taking and in the event that there is something that they're unsatisfied this can be conversed before the project gets too far along.

**Possible Disadvantages**

Although it can be advantageous for the developer to have constant communication with the consumer, this can also be a disadvantage especially when the consumer is unsure of what they want and keep making changes. This particularly matters where the consumer doesn't understand the workings of the system (which is the case most of the time) and they want to make what they believe to be a small change but in reality it requires a lot of changes to the system.

### **Tools require to ensure completion**

To ensure the completion of the project there are various different tools that are required, in terms of hardware, the only hardware required to develop the project is a computer that has a stable internet connection as well as being able to run visual studio. As well the hardware required, there is also the software such as visual studio and the TweetInvi API. The only other requirement to ensure the completion of this project is at least 2 twitter accounts (1 to act as the server and the other as a user).

The language that the project will be developed in is C#. The reasons for this is because the TweetInvi API is a C# API also C# is the language that I am most knowledgeable in.

## **Design and Implementation**

### **About the Project**

The game created is a text based RPG game with visual elements that is playable entirely through twitter. The objective of the game is to try and battle your way through monsters in hope to get loot that is good enough allowing you to progress to battle harder and stronger monsters.

The game can be played through any twitter client by simply tweeting @TwtrArcade any of the relevant commands.

The current available commands for the TwtrArcade RPG game are:

**!REGISTER** - This command registers a player to the game, by doing this it allows them to use the other commands listed as well as gives them starting armour so they can begin their journey.

**!HELP** – This command will display an image to the user listing all of the possible commands.

**!INVENTORY** – This command will show the player an image of their current equipped items as well as what they currently have equipped

**!EQUIP (1 -9)** – This command will equip the item that the user specifies for example **!EQUIP 1** will equip the item in the first inventory slot

**!BATTLE (1-5)** – This command will start a battle between the player and a randomly selected monster, the number followed defines how difficult the monster will be.

## Moscow List

|      |           |             |          |            |
|------|-----------|-------------|----------|------------|
| Key: | Usability | Performance | Features | Processing |
|------|-----------|-------------|----------|------------|

### Functional Requirements

| <b>MUST</b> |                                                                                 |
|-------------|---------------------------------------------------------------------------------|
| Features    | The application must give a tutorial on how it is used                          |
| Features    | The application must Allow for user input.                                      |
| Usability   | The application must provide user feedback.                                     |
| Processing  | The application must be able to Add users to the database                       |
| Processing  | The application must be able to Access user information from the database       |
| Processing  | The application should be able to access monster information from the database. |
| Processing  | The application must allow for users to battle monsters                         |
| Processing  | The application must be able to receive tweets                                  |
| Processing  | The application must be able to respond to tweets                               |
| Features    | The application must allow the users to make moves                              |
| Processing  | The application must generate responses depending on system input               |
| Processing  | The application must follow a well-defined set of rules                         |

Table 1 - Functional Requirements MUST

| <b>SHOULD</b> |                                                               |
|---------------|---------------------------------------------------------------|
| Usability     | The application should provide visual feedback                |
| Performance   | The application should be user friendly                       |
| Processing    | The application should run on all twitter clients             |
| Processing    | The application should be able to generate images of players  |
| Processing    | The application should be able to generate images of items    |
| Processing    | The application should be able to generate images of monsters |
| Usability     | The application should be able to attach media to tweets      |

Table 2 - Functional Requirements SHOULD

| <b>COULD</b> |                                                                               |
|--------------|-------------------------------------------------------------------------------|
| Features     | The application could show examples of usage                                  |
| Features     | The application could allow users to view their inventory                     |
| Features     | The application could allow users to equip items from their inventory         |
| Features     | The application could allow users to drop / delete items from their inventory |

Table 3 - Functional Requirements COULD

| <b>WON'T</b> |                                                                                          |
|--------------|------------------------------------------------------------------------------------------|
| Performance  | The application will not have any personal information held (Excluding Twitter Username) |
| Performance  | The application will not have a levelling system                                         |
| Performance  | The application will not allow for PVP combat                                            |

Table 4 - Functional Requirements WON'T

**Non - Functional Requirements**

|                                                                                   | <b>WON'T</b>                                                             |
|-----------------------------------------------------------------------------------|--------------------------------------------------------------------------|
|  | The system won't be available 24/7, Only available when demo is running. |

Table 5 - Non-Functional Requirements *WONT*

## **Time boxes**

### **Timebox 1**

Date – 2<sup>nd</sup> January 2015

Duration – 5 Days

Completion Date – 7<sup>th</sup> January

### ***Features***

- Create Server Twitter Account
- Create Database
- Send / Receive Tweet from Twitter

## Design and Implementation

### Send Tweet Use Case Description

|                                                                                                                                                                                                          |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Use Case Name: Send Tweet                                                                                                                                                                                |
| Stakeholders & Interests: Server and client                                                                                                                                                              |
| Brief description: Server must be able to communicate and send data to twitter                                                                                                                           |
| Trigger: The Server is required to send a tweet.                                                                                                                                                         |
| Normal flow of events:<br><ol style="list-style-type: none"><li>1. Server is informed that it needs to tweet.</li><li>2. Server connects to twitter.</li><li>3. Server sends tweet to twitter.</li></ol> |
| Alternative / Exceptional flow of events:<br>S-1a:<br><ol style="list-style-type: none"><li>1. Server has sent too many tweets recently.</li><li>2. Server has no internet connection.</li></ol>         |

Table 6 - Send Tweet Use Case

### Use Case Diagram

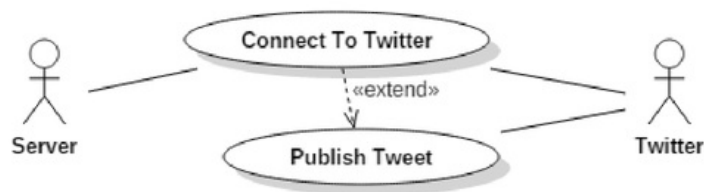


Figure 1 - Send Tweet

#### Receive Tweet Use Case Description

|                                                                                                                                                                                               |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Use Case Name: Receive Tweet                                                                                                                                                                  |
| Stakeholders & Interests: Server and client                                                                                                                                                   |
| Brief description: Server must be able to communicate and receive data from twitter                                                                                                           |
| Trigger: The Server is required to receive a tweet from twitter.                                                                                                                              |
| Normal flow of events: <ol style="list-style-type: none"> <li>1. Server Connects to Twitter</li> <li>2. Server Continuously listens for tweets</li> <li>3. Server interprets tweet</li> </ol> |
| Alternative / Exceptional flow of events:<br>S-1a: <ol style="list-style-type: none"> <li>1. Server has not internet connection</li> <li>2. Server cannot connect to twitter.</li> </ol>      |

Table 7- Receive Tweet Use Case

#### Use Case Diagram

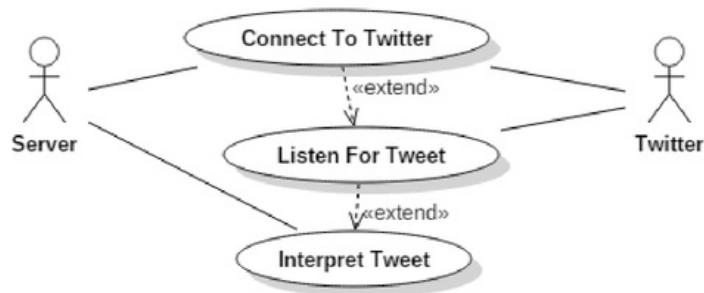


Figure 2 - Receive Tweet

Once the system was able to both send and receive tweets it means that the system would now start to be able to do something with the data being received and also respond appropriately. Since the main focus of the project was going to be an RPG style game this means that there would be some form of database required. The tables of the database will be created using SQL server. To test the functionality of the program a series of tests were carried out to ensure that the functionality were working as expected. To be able to test that the system is able to send a tweet to twitter, the value was simply hardcoded and run. However testing the receiving of a tweet required a MessageBox to be implemented so that we can check that the tweet had been received as well as checking various other technical details such as being able to check the user who sent the tweet and other various pieces of information.

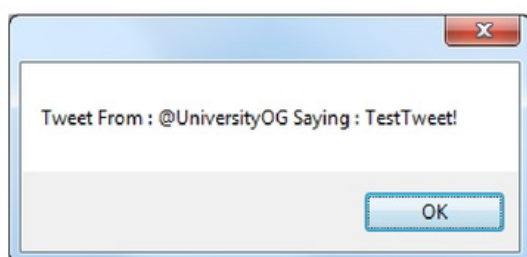


Figure 3- Testing receive a tweet

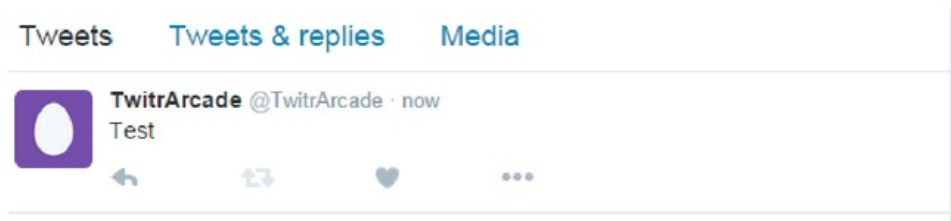


Figure 4 - Testing send a tweet

### *Testing*

| Task ID | Task Name                             | Days expected | Actual Days | Planned Results                                                              | Actual Results | Problems | Comments |
|---------|---------------------------------------|---------------|-------------|------------------------------------------------------------------------------|----------------|----------|----------|
| 1       | Create server<br>Twitter              | 1 days        | 1 days      | Twitter account created                                                      | As Expected    | N/A      | N/A      |
| 2       | Create Database                       | 3 days        | 2 days      | Database designed and implemented to house all planned features of the game. | As Expected    | N/A      | N/A      |
| 3       | Test Send and Receive Tweet Functions | 1 days        | 1 days      | Relevant message box's would show                                            | As Expected    | N/A      | N/A      |

*Table 8 - Timebox 1 Testing*

***Evaluation***

All tasks were finished within the planned time period with all of the tasks being completed early and without any issues. As this is the first time box, it can be expected to be marginally incorrect by the amount of days needed to complete each task. Progression and completion of each timebox will allow the time scope to be managed efficiently and realistically.

***Requirements Completed***

- The server must have it's own twitter account.
- Create a database to store information
- Send / Receive Tweets to and from Twitter

**Timebox 2**

Date – 14<sup>th</sup> January 2015

Duration – 4 Days

Completion Date –18<sup>th</sup> January

***Features***

- Create Twitter application (Apps.twitter.com)
- Integrate database functionality into send and receive functions

## Design and Implementation

### Database Integration – Receive Tweet Use Case Description

|                                                                                                                                                                |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Use Case Name: User Register                                                                                                                                   |
| Stakeholders & Interests: Server and client                                                                                                                    |
| Brief description: Server must be able to receive a tweet from twitter, Interpret the data and then register the user into the database.                       |
| Trigger: The server receives a tweet that contains a register command.                                                                                         |
| Normal flow of events:<br>4. Server receives tweet<br>5. Server Interprets tweet<br>6. Server checks if user is in database<br>7. Server adds user to database |
| Alternative / Exceptional flow of events:<br>S-1a:<br>3. User is already in database<br>4. Server can't connect to database.                                   |

Table 9 - Receive Tweet With Database Intergration

### Use Case Diagram

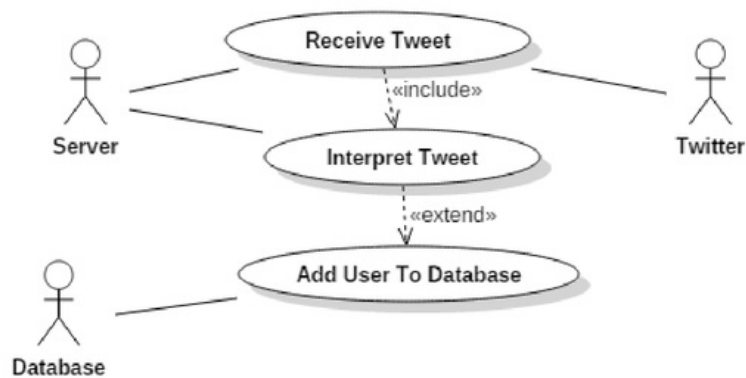


Figure 5 - Register use case

Once the functions has been updated to work with the database it would make sense now to test the functions to ensure that everything is working correctly; however before this is possible it is required that a twitter app account is setup. This can be done by going to the [apps.twitter.com](https://apps.twitter.com/) website where you're able to sign up and create an application. Once an application has been created it means that the TweetInvi API can be used to gather information such as time that the tweet was sent and various other details.

## TwtrArcade

Test OAuth

DetailsSettingsKeys and Access TokensPermissions

TwitterArcade  
<https://apps.twitter.com/>

### Organization

Information about the organization or company associated with your application. This information is optional.

|                      |      |
|----------------------|------|
| Organization         | None |
| Organization website | None |

### Application Settings

Your application's Consumer Key and Secret are used to [authenticate](#) requests to the Twitter Platform.

|                         |                                                                                                       |
|-------------------------|-------------------------------------------------------------------------------------------------------|
| Access level            | Read, write, and direct messages ( <a href="#">modify app permissions</a> )                           |
| Consumer Key (API Key)  | mmrh6rdLW8o4vewcCNkapaAN ( <a href="#">manage keys and access tokens</a> )                            |
| Callback URL            | None                                                                                                  |
| Callback URL Locked     | No                                                                                                    |
| Sign in with Twitter    | Yes                                                                                                   |
| App-only authentication | <a href="https://api.twitter.com/oauth2/token">https://api.twitter.com/oauth2/token</a>               |
| Request token URL       | <a href="https://api.twitter.com/oauth/request_token">https://api.twitter.com/oauth/request_token</a> |
| Authorize URL           | <a href="https://api.twitter.com/oauth/authorize">https://api.twitter.com/oauth/authorize</a>         |
| Access token URL        | <a href="https://api.twitter.com/oauth/access_token">https://api.twitter.com/oauth/access_token</a>   |

Figure 6 - Twitter APP details

The APP defines what different permissions can be carried out such as only being able to read tweets, read and write or even send direct messages to people. Our APP has full access meaning it can do anything a regular user would be able to do.

Next we are able to test the extended send and receive functions to ensure that they are working as expected.

To test the receiving tweet function, the same debugging messagebox was used but instead of this time using hardcoded values to separate the username from the message, this time it's done through the use of the TweetInvi API and Twitter APP that was setup.

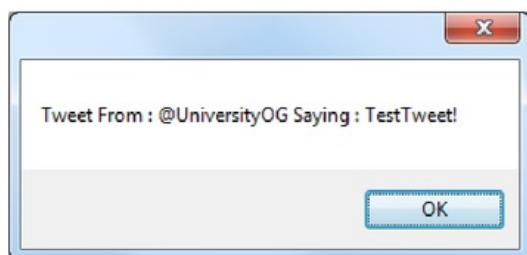


*Figure 7 - Testing new receive function*

To test that the extended sending function worked as correctly, it was setup so that upon successful registration the user who sent the registration tweet would get a response automatically.



Figure 8 - Testing new send tweet function

### *Testing*

| Task ID | Task Name                             | Days expected | Actual Days | Planned Results                                               | Actual Results | Problems                                                | Comments                                      |
|---------|---------------------------------------|---------------|-------------|---------------------------------------------------------------|----------------|---------------------------------------------------------|-----------------------------------------------|
| 1       | Create Twitter Application            | 1 days        | 1 days      | Twitter Application Created                                   | As Expected    | N/A                                                     | N/A                                           |
| 2       | Extend Send and receive Functions     | 1 days        | 2 days      | Send and receive functions extended to work with the database | As Expected    | Originally the server would not send tweets to twitter. | Problem has been fixed, typo in security key. |
| 3       | Test Send and Receive Tweet Functions | 1 days        | 1 days      | Relevant message box's would show                             | As Expected    | N/A                                                     | N/A                                           |

*Table 10 - Timebox 2 Testing*

The issue with the send and receive functions were down to human error where there was a typo in the access key. Without a correct access key the software was not able to send tweets.

### ***Evaluation***

All the tasks were completed, this time to a more accurate time period with the exception of extending the methods; however this is due to a small error which if did not occur would have meant the deadlines were reached.

### ***Requirements Completed***

- Server can register users.
- Send / Receive Tweets to and from Twitter (Redone)

**Timebox 3**

Date – 22<sup>nd</sup> January 2015

Duration – 6.5 Days

Completion Date – 28<sup>th</sup> January

***Features***

- **Create Character generation (Register Extension)**
- Create Inventory management

## Design and Implementation

### Character Generation Use Case

|                                                                                                                                                            |
|------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Use Case Name: Character Generation                                                                                                                        |
| Stakeholders & Interests: Server and client                                                                                                                |
| Brief description: Server generates a new players starting character                                                                                       |
| Trigger: The server receives a tweet that contains a register command.                                                                                     |
| Normal flow of events:<br>8. Server receives tweet<br>9. Server Interprets tweet<br>10. Sets user Armour to be Starting Armour<br>11. Add user to database |
| Alternative / Exceptional flow of events:<br>S-1a:<br>5. User is already in database<br>6. Server can't connect to database.                               |

Table 11 - Character Generation Use Case

### Use Case Diagram

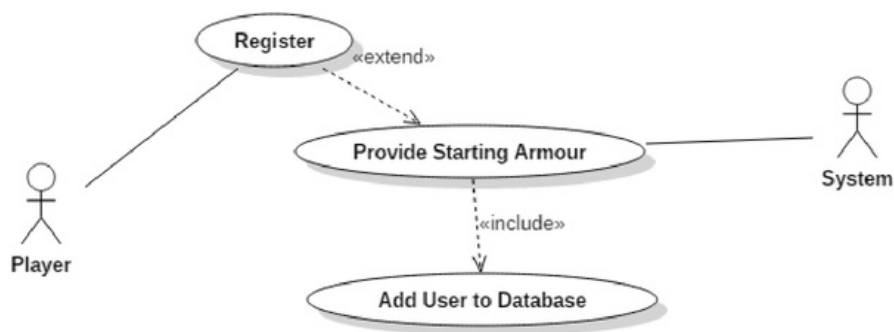


Figure 9 - Updated Register usecase

The Character generation is a replacement to the original register user case, this is because originally the register use case was only dealing with non-specific details but instead stored personal information about the users such as twitter username, however now that the system has expanded the register / character generation functions now need to account for the users in game player too.

### Inventory Management Use Case

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Use Case Name: Inventory Management                                                                                                                                                                                                                                                                                                                                                                                                                            |
| Stakeholders & Interests: Server and client                                                                                                                                                                                                                                                                                                                                                                                                                    |
| Brief description: Server handles the swapping of items between inventory and equipped items                                                                                                                                                                                                                                                                                                                                                                   |
| Trigger: The server receives a tweet that contains an equip command.                                                                                                                                                                                                                                                                                                                                                                                           |
| Normal flow of events: <ol style="list-style-type: none"> <li>1. Server receives tweet</li> <li>2. Server interprets tweet</li> <li>3. Gets the item from the slot provided in tweet</li> <li>4. Gets the type of current equipped item</li> <li>5. Gets the current equipped item of that type</li> <li>6. Swap the current equipped item with the one that was specified</li> <li>7. Update Database</li> <li>8. Send Response Confirming Changes</li> </ol> |
| Alternative / Exceptional flow of events:<br>S-1a: <ol style="list-style-type: none"> <li>1. Invalid Slot ID</li> <li>2. Can't Connect to Database</li> </ol>                                                                                                                                                                                                                                                                                                  |

Table 12 - Inventory Management Use Case

### Use Case Diagram

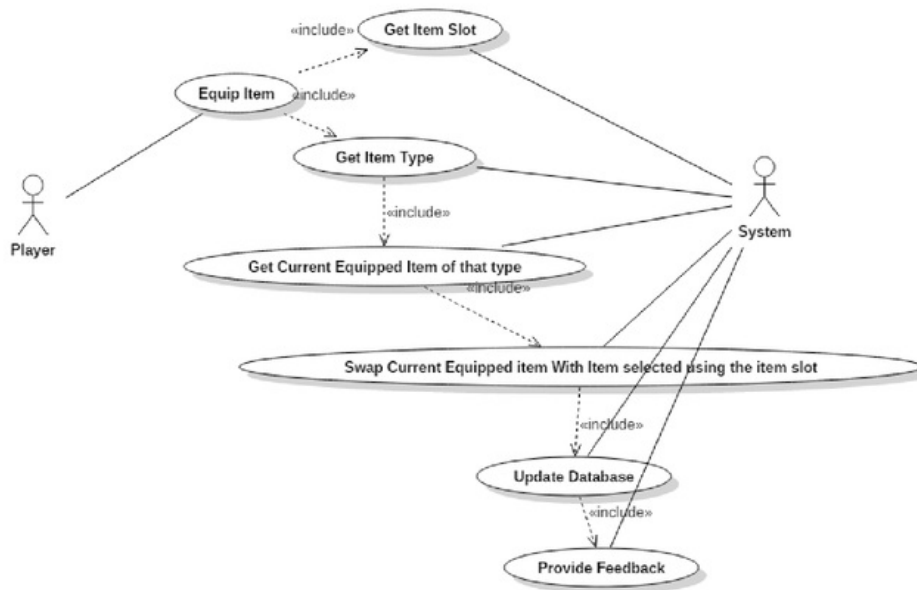


Figure 10 - Equip item usecase

To be able to test the newly implemented functions, all users from the database were deleted so that there wasn't a need to create a new twitter account each time.

### ***Testing***

| Task ID | Task Name                                                | Days expected | Actual Days | Planned Results                                                                                                                           | Actual Results | Problems | Comments |
|---------|----------------------------------------------------------|---------------|-------------|-------------------------------------------------------------------------------------------------------------------------------------------|----------------|----------|----------|
| 1       | Extend Register Function                                 | 2 days        | 1.5 days    | Function Extended to now also handle the creation of player characters too (set their starting armour and inventory)                      | As Expected    | N/A      | N/A      |
| 2       | Create Inventory Management Function                     | 5 days        | 4 days      | Inventory management function created, users can now equip items from their inventory.                                                    | As Expected    | N/A      | N/A      |
| 3       | Test Inventory Management and Extended Register Function | 0.5 days      | 1 days      | Users Registered as well as their starting armour and inventory created, players able to swap items between inventory and equipped items. | As Expected    | N/A      | N/A      |

*Table 13 - Timebox 3 Testing*

### ***Evaluation***

All the tasks were completed; most tasks were completed on time with the exception of testing the changes which took slightly longer than expected.

### ***Requirements Completed***

- Server can register users (Redone).
- Users can swap items between inventory and equipped.

**Timebox 4**

Date – 5<sup>th</sup> February 2015

Duration – 13 Days

Completion Date – 18<sup>th</sup> February

***Features***

- Create Battle System
- Create Loot System

## Design and Implementation

### Player Battle Use Case

|                                                                                                                                                                                                                                                                      |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Use Case Name: Character Generation                                                                                                                                                                                                                                  |
| Stakeholders & Interests: Server and client                                                                                                                                                                                                                          |
| Brief description: Server Initiates a battle between a player and monster                                                                                                                                                                                            |
| Trigger: The server receives a tweet that contains a battle command.                                                                                                                                                                                                 |
| Normal flow of events:<br>12. Server receives tweet<br>13. Server Interprets tweet<br>14. Select / Generate Monster To Battle<br>15. Simulate a Turn Based Battle Between Monster / Player<br>16. Calculate Result<br>17. Generate Loot<br>18. Post Result of Battle |
| Alternative / Exceptional flow of events:<br>S-1a:<br>7. Invalid Difficulty Supplied<br>8. Server can't connect to database.                                                                                                                                         |

Table 14 - Player Battle Use Case

### Use Case Diagram

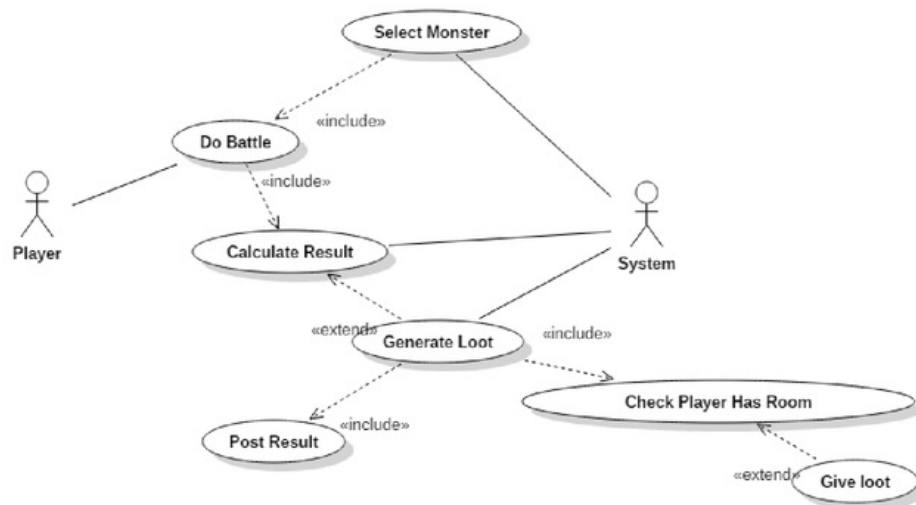


Figure 11 - Battle use case

#### Loot Generation Use Case

|                                                                                                                                                                                                                                                                                                                              |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Use Case Name: Loot Generation                                                                                                                                                                                                                                                                                               |
| Stakeholders & Interests: Server and client                                                                                                                                                                                                                                                                                  |
| Brief description: Server generates loot to give to the player after successfully beating a monster.                                                                                                                                                                                                                         |
| Trigger: The server has just finished a battle simulation in which the player won                                                                                                                                                                                                                                            |
| Normal flow of events:<br>9. Simulation Ends with Player Winning<br>10. Checks if user has inventory space free.<br>11. Randomly Selects item type to generate<br>12. Randomly Selects item rarity to generate (based on monster difficulty)<br>13. Generate Loot<br>14. Add item to Player Inventory<br>15. Update Database |
| Alternative / Exceptional flow of events:<br>S-1a:<br>3. Player Inventory Full<br>4. Can't Connect to Database                                                                                                                                                                                                               |

Table 15 - Loot Generation Use Case

The battle system implemented was rather complex taking into account things such as player armour to mitigate monster damage as well as calculating maximum hits based on equipped items. Due to this it means that the testing process for this was rather complex and required the modification of pre-existing code. By modifying existing code it allowed for hardcoded values such as which monster to fight instead of a new monster; by doing this it allowed to check that all player stats were correctly being taken into account and that the battle algorithm was working as expected.

### Testing

| Task ID | Task Name                          | Days expected | Actual Days | Planned Results                                                                                                             | Actual Results | Problems                                           | Comments                                               |
|---------|------------------------------------|---------------|-------------|-----------------------------------------------------------------------------------------------------------------------------|----------------|----------------------------------------------------|--------------------------------------------------------|
| 1       | Create battle system               | 7 days        | 7 days      | Create a complex battle system that has turned based combat that the server simulates to determine the winner.              | As Expected    | N/A                                                | N/A                                                    |
| 2       | Create Loot system                 | 2 days        | 2 days      | Upon winning a battle, the system should reward the player.                                                                 | As Expected    | Originally Forgot to account for full inventories. | N/A                                                    |
| 3       | Test Battle system and loot system | 3 days        | 4 days      | Users able to battle monsters and in the event that they win and do not have a full inventory should be rewarded with loot. | As Expected    | Balancing Issues.                                  | Extended time is due to tweaking the combat algorithm. |

Table 16 - Timebox 4 Testing

Although the original combat system worked without any bugs, it was completely unbalanced in the way that the monsters were too strong. Once the algorithm was tweaked it needed to be re-tested to ensure it still worked as expected.

### ***Evaluation***

All the tasks were completed; most tasks were completed on time with the exception of testing of the battle algorithm this is because it seemed that the monsters would overpower lower levelled players easily so the algorithm had to be tweaked slightly and re-tested.

### ***Requirements Completed***

- Users can battle monsters and receive loot.

## Implementation

This section of the report is going to outline exactly how each part of the system was implemented.

The first section of code that will be explained is the Main method, this is run when the server / application is started.

```
static void Main()
{
    Application.EnableVisualStyles();
    Application.SetCompatibleTextRenderingDefault(false);

    Tweetinvi.Auth.SetUserCredentials("www", "xxxx", "yyyy", "zzzz");
    GETTWEET();
}
```

The first thing the server does is setup the various different user credentials, to ensure security the code has been modified slightly. The arguments are Public key, Secret Key, Access Token and Access Secret; all of which are available through the twitter applications website.

Once the server has setup the user credentials it next calls a function called GETTWEET().

```
static private void GETTWEET()
{
    var userStream = Tweetinvi.Stream.CreateUserStream();
    userStream.TweetCreatedByAnyone += (sender, args) =>
    {
        tweets.Add(args.Tweet.Id);
        CheckTweet(args.Tweet.CreatedBy.IdStr, "" +
args.Tweet.CreatedBy.ScreenName, "" + args.Tweet.Text, args.Tweet.CreatedBy.Id,
args.Tweet.Id);
    };
    userStream.StartStream();
}
```

The GETTWEET Method is quite a simple method. The GETTWEET method makes use of the Tweetinvi API to create a stream, by doing this it allows the method to be called each time the server receives a new tweet.

Upon receiving a new tweet it then adds the tweetID to an array so that it can be handled later; it also then passes the arguments of the tweet (The user who created the tweet, the text that the tweet contained as well as the ID of the user who created it) to a new method called CheckTweet.

The CheckTweet method is what handles what the system does depending on the input received.

```
static void CheckTweet(string sid, string sname, string text, long id, long
tweetID)
{
    string tweet = text.Remove(0, 13);

    if (tweet == "!REGISTER")
    {
        registerUser(sid, sname, id, tweetID);
    } else if(tweet == "!INVENTORY")
    {
        drawPlayer(id);
        pushTweet(id, tweetID);
    }
    string[] ssize = text.Split(null);

    if(ssize[1] == "!INSPECT")
    {
        int idtoview = int.Parse(ssize[2]);
        drawInventorySlot(idtoview, id, tweetID);
    } else if(ssize[1] == "!EQUIP")
    {
        int invslot = int.Parse(ssize[2]);

        EquipItem(id, invslot, tweetID);
    } else if(ssize[1] == "!BATTLE")
    {
        int difficulty = int.Parse(ssize[2]);
        int monsterID = selectMonster(difficulty);
        DoBattle(id, monsterID, tweetID);
    }
}
```

Before the method is able to interpret the text, it must trim un-needed information (in this case it would be the first characters of the tweet that says @TwitrArcade), once it has done this then a sequence of if statements determine what to do next.

If the tweet contains a register command then the server will call the registerUser method passing in various arguments such as the id of the user, the username and the id of the tweet that called the register command.

The registerUser method is slightly more complicated than the previous methods looked at so far, this is because there is database interaction.

The first thing the registerUser method does is setup a string that can be used to connect to the database. Once the connection string is setup then a connection to the database will try to be established.

```
static void registerUser(string sid, string sname, long id, long tweetID)
{
    string myConnection =
"datasource=localhost;port=3306;username=usernamehere";
    MySqlConnection myConn = new MySqlConnection(myConnection);
    MySqlDataAdapter myDataAdapter = new MySqlDataAdapter();
    try
    {
        myConn.Open();

        myDataAdapter.InsertCommand = new MySqlCommand("INSERT INTO
twitterarcade.users VALUES (NULL, " + sid + ", '" + sname + "', 4422, 4421, 4423,
4424, 0, 0, 0, 0, 0, 0, 0)", myConn);
        myDataAdapter.InsertCommand.ExecuteNonQuery();
        myConn.Close();

        List<Byte[]> files = new List<byte[]>();
        byte[] file =
File.ReadAllBytes(@"C:\Users\Andrew\Desktop\TwitterArcade\RPGBATTLE\welcome.bmp");

        files.Add(file);

        var tweetToReplyTo = Tweet.GetTweet(tweetID);

        var tweet2 = Tweet.CreatePublishTweetOptionalParameters();
        tweet2.MediaBinaries = files;
        tweet2.InReplyToTweet = tweetToReplyTo;
        var tweet = Tweet.PublishTweet(string.Format("@{0} {1}",
tweetToReplyTo.CreatedBy.ScreenName, "Thanks for Registering!"), tweet2);

    }
    catch (Exception ex)
    {
    }
}
```

If the connection is established then an insert command is created. The insert command is used to insert the user into the appropriate table (twitterarcade.users) and set the appropriate values. The values in the insert command sets the values of the players inventory and armour so that they have starting armour. Once the database has been successfully updated, then the server creates a byte array which will store the information required to upload a tweet with an image attached.

Once the image has been read into the array then the server will send a reply to the userID that was provided by the CHECKTWEET method earlier.

if instead of the tweet containing a register command it instead contains a inventory command then the server will execute two different methods.

The first method is the drawPlayer method, due to the complexity and reputation of this code some bits will be snipped out. This method starts off by creating a new bitmap that is 1380 x 1600, once it has created this it then sets the background of the bitmap by loading a saved image and applying it.

Next it gets the players information, such as what armour they are wearing and what they have in their inventory.

```
static void drawPlayer(long UID)
{
    Bitmap bitmap = new Bitmap(1380, 1600);
    Graphics graphics = Graphics.FromImage(bitmap);
    Image inputImage =
Bitmap.FromFile(@"C:\Users\Andrew\Desktop\TwitterArcade\RPGBATTLE\playerbackground
.png");

    int head = getPlayerEquipment(UID, 0);
    int body = getPlayerEquipment(UID, 1);
    int gloves = getPlayerEquipment(UID, 2);
    int legs = getPlayerEquipment(UID, 3);

    int inv1 = getPlayerEquipment(UID, 4);
    int inv2 = getPlayerEquipment(UID, 5);
    int inv3 = getPlayerEquipment(UID, 6);
    int inv4 = getPlayerEquipment(UID, 7);
    int inv5 = getPlayerEquipment(UID, 8);
    int inv6 = getPlayerEquipment(UID, 9);
    int inv7 = getPlayerEquipment(UID, 10);
    int inv8 = getPlayerEquipment(UID, 11);
    int inv9 = getPlayerEquipment(UID, 12);

-SNIP! -
```

This is done by using the getPlayerEquipment method although this is a fairly self-explanatory method since it is used widely through out the application it would make sense to show how it works.

The getPlayerEquipment method works by connection to the database and then searching the users table for the player ID that is provided in the arguments as well as the equipment slot that is required.

The slot that is required is then passed into a switch statement that each returns the information from the appropriate field in the database.

```
static int getPlayerEquipment(long UID, int type)
{
    try
    {
        string myConnection =
        "datasource=localhost;port=3306;username=root";
        MySqlConnection myConn = new MySqlConnection(myConnection);
        MySqlDataAdapter myDataAdapter = new MySqlDataAdapter();

        myDataAdapter.SelectCommand = new MySqlCommand("SELECT * from
        twitterarcade.users WHERE UID = " + UID, myConn);
        myConn.Open();
        DataSet ds = new DataSet();
        myDataAdapter.Fill(ds);

        myConn.Close();
        string text = ds.Tables[0].Rows[0].ItemArray[3].ToString();

        switch (type)
        {
            case 0:
                return Int32.Parse(text);

            case 1:
                text = ds.Tables[0].Rows[0].ItemArray[4].ToString();
                return Int32.Parse(text);

            case 2:
                text = ds.Tables[0].Rows[0].ItemArray[5].ToString();
                return Int32.Parse(text);

            case 3:
                text = ds.Tables[0].Rows[0].ItemArray[6].ToString();
                return Int32.Parse(text);

            case 4:
                text = ds.Tables[0].Rows[0].ItemArray[7].ToString();
                return Int32.Parse(text);
        }
    }
    catch (Exception ex)
    {
        return 0;
    }
}
```



Once all the information has been gathered then each one is checked with if statements to set the appropriate image to be displayed.

```
if(inv1Type == "LEGS")
{
    inv1Image =
    Bitmap.FromFile(@"C:\Users\Andrew\Desktop\TwitterArcade\RPGBATTLE\A
rmour\legs" + getItemRarity(inv1) + ".png");
} else if(inv1Type == "BODY")
{
    inv1Image =
    Bitmap.FromFile(@"C:\Users\Andrew\Desktop\TwitterArcade\RPGBATTLE\A
rmour\body" + getItemRarity(inv1) + ".png");
} else if(inv1Type == "HELMET")
{
    inv1Image =
    Bitmap.FromFile(@"C:\Users\Andrew\Desktop\TwitterArcade\RPGBATTLE\A
rmour\helmet" + getItemRarity(inv1) + ".png");
} else if(inv1Type == "GLOVES")
{
    inv1Image =
    Bitmap.FromFile(@"C:\Users\Andrew\Desktop\TwitterArcade\RPGBATTLE\A
rmour\gloves" + getItemRarity(inv1) + ".png");
}
-SNIP!-
```

Once the correct image for the item type and rarity has been set then the next step of the method is to actually draw the image onto the bitmap that was created at the start of the method.

Before doing this, the method first checks that each item actually exists, this is done by checking that the item is not = 0 with would be invalid.

```
        if (inv1 != 0)
        {
            graphics.DrawImage(inv1Image, new Point(650, 950));
        }
        if (inv2 != 0)
        {
            graphics.DrawImage(inv2Image, new Point(870, 950));
        }
        if (inv3 != 0)
        {
            graphics.DrawImage(inv3Image, new Point(1070,
950));
        }
        if (inv4 != 0)
        {
```

```

        graphics.DrawImage(inv4Image, new Point(650,
1165));
    }

```

**-SNIP!-**

Once the equipment has been drawn onto the bitmap the next thing the method does is draw the player stats. There are 3 methods that are used to get the attack stats, the defence stats and the health stats of the player.

```

        RectangleF AttackMod = new RectangleF(900, 280, 400,
400);
        RectangleF DefMod = new RectangleF(1000, 430, 400, 400);
        RectangleF HPMod = new RectangleF(1000, 570, 400, 400);

        graphics.DrawString("" + getEquippedAttackMod(UID), new
Font("Tahoma", 50), Brushes.Black, AttackMod);
        graphics.DrawString("" + getEquippedDefMod(UID), new
Font("Tahoma", 50), Brushes.Black, DefMod);
        graphics.DrawString("" + getEquippedHPMod(UID), new
Font("Tahoma", 50), Brushes.Black, HPMod);
-SNIP!-

```

As the methods for getting the stats are all almost identical, the following example will be for getting the players attack stats.

The method first gets the items that the player has equipped.

```

static int getEquippedAttackMod(long UID)
{
    int head = getPlayerEquipment(UID, 0);
    int body = getPlayerEquipment(UID, 1);
    int gloves = getPlayerEquipment(UID, 2);
    int legs = getPlayerEquipment(UID, 3);
    int attackModifier = 0;
    attackModifier = attackModifier + getAttackMod(head);
    attackModifier = attackModifier + getAttackMod(body);
    attackModifier = attackModifier + getAttackMod(legs);
    attackModifier = attackModifier + getAttackMod(gloves);
    return attackModifier;
}

```

Once the items have all been gathered, the method will next get the attack stats of this item; this is done by using another method called `getAttackMod`.

```
static int getAttackMod(int id)
{
    try
    {
        string myConnection =
"datasource=localhost;port=3306;username=root";
        MySqlConnection myConn = new MySqlConnection(myConnection);
        MySqlDataAdapter myDataAdapter = new MySqlDataAdapter();

        myDataAdapter.SelectCommand = new MySqlCommand("SELECT * from
twitterarcade.items WHERE itemID = " + id, myConn);
        myConn.Open();
        DataSet ds = new DataSet();
        myDataAdapter.Fill(ds);
        myConn.Close();

        string text = ds.Tables[0].Rows[0].ItemArray[3].ToString();
        int retid = Int32.Parse(text);

        return retid;
    }
    catch (Exception ex)
    {
        return 0;
    }
}
```

This method simply connects to the database, this time however it checks the items table in search for the players currently equipped item and then returns the attack stat. The `getEquippedAttackMod` method will do this for each item the user has equipped adding the numbers together each time until a total is reached. This process is also repeated for the defense and health stats too.

Once the stats have been gathered they are then drawn onto the image in the correct locations before being saved. The bitmap is saved in a specific player folder following a specific naming convention (PLAYER- followed by the players ID). By doing this allows for them to easily be tracked and accessed.

```
bitmap.Save(@"C:\Users\Andrew\Desktop\TwitterArcade\RPGBATTLE\Players\PLAYER-" + UID + ".bmp");

        graphics.Dispose();
        bitmap.Dispose();
        inputImage.Dispose();

        Image resized =
resizeImage(Image.FromFile(@"C:\Users\Andrew\Desktop\TwitterArcade\RPGBATTLE\Players\PLAYER-" + UID + ".bmp", false), 890, 1000);

resized.Save(@"C:\Users\Andrew\Desktop\TwitterArcade\RPGBATTLE\Players\PLAYER-" + UID + "-RESIZED.png");
    }
```

However this isn't the last step, once the image has been saved it is then reloaded and resized (this was a change implemented after the original code).

Once the drawplayer method has finished, the next method that is run is the pushTweet method. The pushTweet method is a simple method that loads the image that was just created and then sends it as a tweet to the correct user.

```
static void pushTweet(long id, long tweetID)
{
    List<Byte[]> files = new List<byte[]>();
    byte[] file =
File.ReadAllBytes(@"C:\Users\Andrew\Desktop\TwitterArcade\RPGBATTLE\Players\PLAYER-" + id + "-RESIZED.png");
    files.Add(file);
    var tweetToReplyTo = Tweet.GetTweet(tweetID);
    var tweet2 = Tweet.CreatePublishTweetOptionalParameters();
    tweet2.MediaBinaries = files;
    tweet2.InReplyToTweet = tweetToReplyTo;
    var tweet = Tweet.PublishTweet(string.Format("@{0} {1}",
tweetToReplyTo.CreatedBy.ScreenName, "Inventory:"), tweet2);
}
```

If instead the tweet contains a inspect command then the drawInventorySlot method is run passing in the number the user specified as an argument.

The first thing that the drawInventorySlot method does is check if the number the user entered is between 0 & 12, these are the valid options.

if the user does enter a valid option then the first thing the method does it get a reference to the item in the slot specified, this is done by using the getPlayerEquipment method mentioned earlier.

```
static void drawInventorySlot(int slotID, long id, long tweetid)
{
    if(slotID >= 0 && slotID <= 12)
    {
        int item = getPlayerEquipment(id, slotID);
        if (item != 0)
        {
            drawLoot(item);
            List<Byte[]> rewards = new List<byte[]>();
            byte[] rewardImage =
File.ReadAllBytes(@"C:\Users\Andrew\Desktop\TwitterArcade\RPGBATTLE\Armour\ITEM-"
+ item + ".bmp");
            rewards.Add(rewardImage);
            var tweetToReplyTo = Tweet.GetTweet(tweetid);
            var tweet3 = Tweet.CreatePublishTweetOptionalParameters();
            tweet3.InReplyToTweet = tweetToReplyTo;
            tweet3.MediaBinaries = rewards;
            var tweet5 = Tweet.PublishTweet(string.Format("@{0} {1}",
tweetToReplyTo.CreatedBy.ScreenName, "In Inventory Slot " + slotID + " You are
wearing: "), tweet3);
        }
    }
}
```

Once it has a reference to the item, the next thing that the method does it to call another method called drawLoot while passing in the item as an argument.

The drawLoot method is relatively simple, it works similar to other methods previously looked at. The first thing that this particular method does is check the items database for the provided itemID; once the item has been found it gets the type of the item (gloves, boots ect) as well as the rarity.

Once it has the item type and rarity it can then generate an image for that particular item. The way it does this is rather clever, since the items are setup using a naming convention (ITEMTYPE followed by the rarity ID) it allows for any item type of any rarity to be generated with ease.

```
static void drawLoot(int itemID)
{
    string type = "NONE";
    string rarity = "0";

    try
    {
        string myConnection =
"datasource=localhost;port=3306;username=root";
        MySqlConnection myConn = new MySqlConnection(myConnection);
        MySqlDataAdapter myDataAdapter = new MySqlDataAdapter();

        myDataAdapter.SelectCommand = new MySqlCommand("SELECT * from
twitterarcade.items WHERE itemID =" + itemID, myConn);
        myConn.Open();
        DataSet ds = new DataSet();
        myDataAdapter.Fill(ds);
        myConn.Close();

        type = ds.Tables[0].Rows[0].ItemArray[1].ToString();
        rarity = ds.Tables[0].Rows[0].ItemArray[2].ToString();
    }
    catch (Exception ex)
    {
    }

    Bitmap bitmap = new Bitmap(800, 800);
    Graphics graphics = Graphics.FromImage(bitmap);
    Image inputImage =
Bitmap.FromFile(@"C:\Users\Andrew\Desktop\TwitterArcade\RPGBATTLE\Armour\itemStats
.png");

    Image armourImage =
Bitmap.FromFile(@"C:\Users\Andrew\Desktop\TwitterArcade\RPGBATTLE\Armour\" +
type.ToLower() + rarity + ".png");

    graphics.DrawImage(inputImage, new Point(0, 0));
    graphics.DrawImage(armourImage, new Point(320, 50));

    -SNIP!-
```

Once the item has been generated and drawn, the attack, defence and health stats of the item are obtained and drawn in the same way as the drawplayer method. When the item

image has been generated and saved, the next thing `drawinventoryslot` image does is simply load the image and send it as a tweet.

If the tweet instead receives an equip command, the EquipItem method is called with the equipment slot passed in as an argument.

```
static void EquipItem(long id, int inventorySlot, long tweetID)
{
    try
    {
        int itemID = getPlayerEquipment(id, inventorySlot);
        string ITEMTYPE = getItemType(itemID);
        string myConnection =
        "datasource=localhost;port=3306;username=root";
        MySqlConnection myConn = new MySqlConnection(myConnection);
        MySqlDataAdapter myDataAdapter = new MySqlDataAdapter();
        MySqlDataAdapter myDataAdapter2 = new MySqlDataAdapter();
        int curEquipped = 0;
        myConn.Open();
        int slotToEquip = inventorySlot - 3;

        if (ITEMTYPE == "LEGS")
        {
            curEquipped = getPlayerEquipment(id, 3);
            myDataAdapter2.InsertCommand = new MySqlCommand("UPDATE
twitterarcade.users SET legs=" + itemID + " WHERE UID=" + id, myConn);
            myDataAdapter2.InsertCommand.ExecuteNonQuery();

        }
        else if (ITEMTYPE == "BODY")
        {
            curEquipped = getPlayerEquipment(id, 1);
            myDataAdapter2.InsertCommand = new MySqlCommand("UPDATE
twitterarcade.users SET body=" + itemID + " WHERE UID=" + id, myConn);
            myDataAdapter2.InsertCommand.ExecuteNonQuery();
        }
    }
    catch { }
}
-SNIP!-
```

The first thing that the EquipItem method does is get a reference to the item that the player wants to equip, it does this using the getPlayerEquipment method.

Once it has the reference to the item, It then gets the type of the item using the `getItemType` method.

```
static string getItemType(int itemID)
{
    try
    {
        string myConnection =
"datasource=localhost;port=3306;username=root";
        MySqlConnection myConn = new MySqlConnection(myConnection);
        MySqlDataAdapter myDataAdapter = new MySqlDataAdapter();

        myDataAdapter.SelectCommand = new MySqlCommand("SELECT * from
twitterarcade.items WHERE itemID =" + itemID, myConn);
        myConn.Open();
        DataSet ds = new DataSet();
        myDataAdapter.Fill(ds);
        myConn.Close();
        return ds.Tables[0].Rows[0].ItemArray[1].ToString();
    }
    catch
    {
        return "Error";
    }
}
```

The `getItemType` method simply connects to the database and searches the item table for the specified ID, it then returns the type of the item from the correct field.

The next thing that the `EquipItem` method does is swap the current equipped item with the item that the player wants to equip.

```
myDataAdapter.InsertCommand = new MySqlCommand("UPDATE twitterarcade.users SET
inventorySlot" + slotToEquip + "=" + curEquipped + " WHERE UID=" + id, myConn);
myDataAdapter.InsertCommand.ExecuteNonQuery();
myConn.Close();
} catch (Exception ex)
{
}
drawPlayer(id);

List<Byte[]> rewards = new List<byte[]>();
byte[] rewardImage =
File.ReadAllBytes(@"C:\Users\Andrew\Desktop\TwitterArcade\RPGBATTLE\Players\PLAYER
-" + id + "-RESIZED.png");
rewards.Add(rewardImage);
var tweetToReplyTo = Tweet.GetTweet(tweetID);
var tweet3 = Tweet.CreatePublishTweetOptionalParameters();
tweet3.InReplyToTweet = tweetToReplyTo;
tweet3.MediaBinaries = rewards;
var tweet5 = Tweet.PublishTweet(string.Format("@{0} {1}",
tweetToReplyTo.CreatedBy.ScreenName, "Item Equiped : "), tweet3);
}
```

Finally the method will call the necessary methods to regenerate the image players inventory image and then upload that image in a tweet.

The final possible command that the server could receive is the battle command, this is by far the most complex out of all the commands.

The first thing that happens when a battle command is received is that a monster is selected on the difficulty that the user has entered, this is done by passing the number they entered as an argument into the selectMonster method.

The actual selection of a monster is relatively simple, it simply connects to the database and then gets a list of all monsters with the selected difficulty. From there a random monster is selected and returned.

```
static int selectMonster(int difficulty)
{
    int monsterID = 0;
    try
    {
        string myConnection =
"datasource=localhost;port=3306;username=root";
        MySqlConnection myConn = new MySqlConnection(myConnection);
        MySqlDataAdapter myDataAdapter = new MySqlDataAdapter();

        myDataAdapter.SelectCommand = new MySqlCommand("SELECT * from
twitterarcade.monsters WHERE Difficulty =" + difficulty, myConn);
        myConn.Open();
        DataSet ds = new DataSet();
        myDataAdapter.Fill(ds);
        myConn.Close();
        int ammount = ds.Tables[0].Rows.Count;
        Random rnd = new Random();
        string text = ds.Tables[0].Rows[rnd.Next(0,
ammount)].ItemArray[0].ToString();
        monsterID = Int32.Parse(text);
    }
    catch (Exception ex)
    {
    }
    return monsterID;
}
```

Once a monster has been selected the next thing that happens is the actual DoBattle method is called.

The DoBattle method handles everything from simulating the battle to reporting the outcome. The first thing the method does is get the stats for the player, this is done by the previously mentioned getEquippedAttackMod, getEquippedHPMod and getEquippedDefMod methods. However to be able to simulate the fight, it is required to know the stats of the monster too; this is done using the GetMonsterSTAT method.

```
static void DoBattle(long playerID, int MonsterID, long tweetID)
{
    int playerHealth = getEquippedHPMod(playerID);
    int playerAttack = getEquippedAttackMod(playerID);
    int playerDefense = getEquippedDefMod(playerID);
    int monsterHP = GetMonsterSTAT(MonsterID, 1);
    int monsterAttack = GetMonsterSTAT(MonsterID, 2);
    double playerNegate = (playerDefense * 10) / 100;
    double monsterMaxHit = monsterAttack - playerNegate;
    drawMonster(MonsterID);
    List<Byte[]> files = new List<byte[]>();
    byte[] file =
File.ReadAllBytes(@"C:\Users\Andrew\Desktop\TwitterArcade\RPGBATTLE\monsterFight-"
+ MonsterID + ".bmp");
    files.Add(file);
    var tweetToReplyTo = Tweet.GetTweet(tweetID);
    var tweet2 = Tweet.CreatePublishTweetOptionalParameters();
    tweet2.MediaBinaries = files;
    tweet2.InReplyToTweet = tweetToReplyTo;
    var tweet = Tweet.PublishTweet(string.Format("@{0} {1}",
tweetToReplyTo.CreatedBy.ScreenName, "Opponent has been Selected!"), tweet2);

    Random rnd = new Random();

    while(monsterHP > 0 && playerHealth > 0)
    {
        int playerHit = rnd.Next(playerAttack/2, playerAttack);
        int monsterHit = rnd.Next((int)monsterMaxHit/2,
(int)monsterMaxHit);
        //Player turn
        monsterHP = monsterHP - playerHit;
        if(monsterHP > 0)
        {
            playerHealth = playerHealth - monsterHit;
        }
    }

    if(playerHealth <= 0)
    {
        var tweet3 = Tweet.CreatePublishTweetOptionalParameters();
        tweet3.InReplyToTweet = tweetToReplyTo;
        var tweet4 = Tweet.PublishTweet(string.Format("@{0} {1}",
tweetToReplyTo.CreatedBy.ScreenName, "The Monster beat you!"), tweet3);
    }
}

-SNIP!-
```

The GetMonsterSTAT method is simple in that all it does is connect to the database and get the relevant stat which is listed in the arguments for the particular monster in question.

```
static int GetMonsterSTAT(int MonsterID, int stat)
{
    try
    {
        string myConnection =
"datasource=localhost;port=3306;username=root";
        MySqlConnection myConn = new MySqlConnection(myConnection);
        MySqlDataAdapter myDataAdapter = new MySqlDataAdapter();

        myDataAdapter.SelectCommand = new MySqlCommand("SELECT * from
twitterarcade.monsters WHERE MonsterID = " + MonsterID, myConn);
        myConn.Open();
        DataSet ds = new DataSet();
        myDataAdapter.Fill(ds);

        myConn.Close();
        string text = ds.Tables[0].Rows[0].ItemArray[1].ToString();

        switch (stat)
        {
            case 1:
                return Int32.Parse(text);
            case 2:
                text = ds.Tables[0].Rows[0].ItemArray[2].ToString();
                return Int32.Parse(text);
            case 3:
                text = ds.Tables[0].Rows[0].ItemArray[3].ToString();
                return Int32.Parse(text);

            default:
                return 0;
        }
    }
    catch (Exception ex)
    {
        return 0;
    }
}
```

Before the simulation of the battle is actually carried out, an image of the monster is generated so the player knows what monster they are about to battle, this is done using the drawMonster method.

The drawMonster method is pretty similar to most of the other draw methods in such that all it does is gather relevant information and then draw an image based on the information.

```
static void drawMonster(int MonsterID)
{
    Bitmap bitmap = new Bitmap(1380, 1600);
    Graphics graphics = Graphics.FromImage(bitmap);
    Image inputImage =
Bitmap.FromFile(@"C:\Users\Andrew\Desktop\TwitterArcade\RPGBATTLE\fightMonster.png
");
    Image monsterImage =
Bitmap.FromFile(@"C:\Users\Andrew\Desktop\TwitterArcade\RPGBATTLE\monster" +
GetMonsterIcon(MonsterID) + ".jpg");
    RectangleF monsterHP = new RectangleF(550, 1000, 400, 400);
    RectangleF loot = new RectangleF(450, 1200, 600, 400);
    RectangleF chance = new RectangleF(450, 1400, 600, 400);
    graphics.DrawImage(inputImage, new Point(0, 0));
    graphics.DrawImage(monsterImage, new Point(550, 400));
    int health = GetMonsterSTAT(MonsterID, 1);
    graphics.DrawString("" + health, new Font("Tahoma", 50),
Brushes.White, monsterHP);

bitmap.Save(@"C:\Users\Andrew\Desktop\TwitterArcade\RPGBATTLE\monsterFight-" +
MonsterID + ".bmp");
    graphics.Dispose();
    bitmap.Dispose();
    inputImage.Dispose();
}
```

Once the monster is drawn, it is then uploaded as a tweet to the user so they know what monster they are facing. The next thing the DoBattle method does is actually simulate the fight between the player and the monster.

The way that the simulation works is that the player and monster both take turns attacking until one of them is dead; the damage that the player or monster does is random based and can be anywhere between their attack damage divided by 2 and their attack damage.

The players max hit is calculated based on their attack damage stats from their equipment, the monsters however is calculated on their hardcoded attack damages with player armour mitigation taken into account. A player can mitigate 10% of their total armour for example if a monster has a max hit of 500 and a player has 1000 armour then the maximum that the monster would ever hit on that particular player is 400.

If the battle ends and the player has lost then the DoBattle method will send the user a tweet telling the user that they lost.

However, if the player has beat the monster then loot needs to be generated and given.

This is done using a few methods, the first is the GenerateLoot method.

```
else if (monsterHP <= 0)
{
    int reward = GenerateLoot(GetMonsterSTAT(MonsterID, 3));
    drawLoot(reward);

    List<Byte[]> rewards = new List<byte[]>();
    byte[] rewardImage =
File.ReadAllBytes(@"C:\Users\Andrew\Desktop\TwitterArcade\RPGBATTLE\Armour\ITEM-"
+ reward + ".bmp");

    rewards.Add(rewardImage);

    if (GivePlayer(playerID, reward))
    {
        var tweet3 = Tweet.CreatePublishTweetOptionalParameters();
        tweet3.InReplyToTweet = tweetToReplyTo;
        tweet3.MediaBinaries = rewards;
        var tweet5 = Tweet.PublishTweet(string.Format("@{0} {1}",
tweetToReplyTo.CreatedBy.ScreenName, "You have Defeated the Monster! And
Received:"), tweet3);
    }
    else
    {
        var tweet3 = Tweet.CreatePublishTweetOptionalParameters();
        tweet3.InReplyToTweet = tweetToReplyTo;
        var tweet5 = Tweet.PublishTweet(string.Format("@{0} {1}",
tweetToReplyTo.CreatedBy.ScreenName, "You have Defeated the Monster! However your
inventory was too full to gain loot!"), tweet3);
    }
}
}
```

The generate loot method is simple in how it works. First it randomly selects what type of item to reward the player (legs, helmet, etc) as well as the rarity. The rarity of the item will depend on the difficulty of the monster battled, an example of this could be that a difficulty 1 monster will only allow loot rarity of tier 0 – tier 1 where as a difficulty 4 monster will allow rarity of tier 0 – tier 4.

```
static int GenerateLoot(int MaxLevel)
{
    Random rnd = new Random();
    int rarity = rnd.Next(0, MaxLevel);
    int type = rnd.Next(0, 3);
    string itemType = "NONE";

    switch(type)
    {
        case 0:
            itemType = "LEGS";
            break;
        case 1:
            itemType = "HELMET";
            break;
        case 2:
            itemType = "BODY";
            break;
        case 3:
            itemType = "GLOVES";
            break;

        default:
            itemType = "LEGS";
            break;
    }
    try
    {
        string myConnection =
"datasource=localhost;port=3306;username=root";
        MySqlConnection myConn = new MySqlConnection(myConnection);
        MySqlDataAdapter myDataAdapter = new MySqlDataAdapter();

        myDataAdapter.SelectCommand = new MySqlCommand("SELECT * from
twitterarcade.items WHERE itemType = '" + itemType + "' AND ItemRarity = " +
rarity + "'", myConn);
        myConn.Open();
        DataSet ds = new DataSet();
        myDataAdapter.Fill(ds);
        int maxNo = ds.Tables[0].Rows.Count;

        myConn.Close();
        string text = ds.Tables[0].Rows[rnd.Next(0,
maxNo)].ItemArray[0].ToString();
        return Int32.Parse(text);
    }
    catch
    {
        return 0;
    }
}
```

Once the item type and rarity has been defined, the method simply randomly selects an item from the items table in the database that matches the criteria.

Now that a reward has been generated, the next step in the process is to actually award the player if possible, this is done using the GivePlayer method.

The GivePlayer method simply connects to the database and checks each inventory slot of the player to check if they have a free inventory space; if this is the case then it will add the item to the inventory and return true, otherwise it will return false.

```
static bool GivePlayer(long UID, int itemID)
{
    string myConnection = "datasource=localhost;port=3306;username=root";
    MySqlConnection myConn = new MySqlConnection(myConnection);
    MySqlDataAdapter myDataAdapter = new MySqlDataAdapter();
    try
    {
        myConn.Open();
        if(getPlayerEquipment(UID, 4) == 0)
        {
            myDataAdapter.InsertCommand = new MySqlCommand("UPDATE
twitterarcade.users SET inventorySlot1=" + itemID + " WHERE UID=" + UID, myConn);
            myDataAdapter.InsertCommand.ExecuteNonQuery();
            myConn.Close();
            return true;
        }
        else if (getPlayerEquipment(UID, 5) == 0)
        {
            myDataAdapter.InsertCommand = new MySqlCommand("UPDATE
twitterarcade.users SET inventorySlot2=" + itemID + " WHERE UID=" + UID, myConn);
            myDataAdapter.InsertCommand.ExecuteNonQuery();
            myConn.Close();
            return true;
        }
        -SNIP!-
        else
        {
            return false;
        }
    }
    catch (Exception ex)
    {
        return false;
    }
}
```

If the item has been added to the inventory then the DoBattle method will send a tweet to the user showing their reward, however if the inventory was full it will notify the user that they did not have enough inventory space to claim their reward.

**Problems and successes**

Throughout the development process of the project there were a few obstacles or problems that slowed progress. The first issue that was encountered is that there is a bug with WAMP server that would on occasion stop the server from being started; this issue would not be resolved until the WAMP server is re-installed.

Another issue that was encountered during the development of the project is that the documentation for the API being used (TweetInvi) was out of date and didn't show how to use the current version of TweetInvi.

The final issue that caused a problem during the development of the system is the slow internet connection in the house, this made debugging twitter code difficult as it was hard to differentiate the difference between defective code and internet latency.

Regardless of all the above issues, they were able to be overcome to create a successful project.

Unfortunately it wasn't possible to overcome the WAMP server issue and if the server breaks again then it will be required to be reinstalled. One issue that was able to overcome is the outdated API; although there was no current documentation or examples there was documentation and examples for previous versions which could be used and modified slightly to make work with the updated version.

## **Testing**

Now that the software has been completed and refined, various aspects of it can now be tested to ensure both usability to see how easy people find it to use as well as acceptance testing.

The first type of testing carried out is usability testing, this particular type of testing is where the user is asked to perform various different tasks that would be commonly done in the system to see if they are able to complete the task without any help.

### **Usability Testing**

A number of users were asked to participate in this testing and were informed how to use the !REGISTER command as this is required to start the process.

**User 1**

| Task asked to perform                 | Expected user process                                                                               | Actual User process                                                               | Comments / Findings                                                                                                   |
|---------------------------------------|-----------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------|
| Register with the service             | User sends a tweet to the server that says !REGISTER                                                | As Expected                                                                       | No Findings                                                                                                           |
| Use the help command                  | User sends a tweet to the server that says !HELP                                                    | As Expected                                                                       | No Findings                                                                                                           |
| Do battle with a difficulty 1 monster | User sends a tweet to the server that has the !BATTLE command followed by the difficulty 1 argument | User sends a tweet containing the battle command but not the difficulty argument. | The user did not know they were able to / required to specify the difficulty, expansion on the help command required. |
| View the Inventory                    | User sends a tweet to the server that contains the !INVENTORY command                               | As Expected                                                                       | No Findings                                                                                                           |
| Equip item from inventory slot 1      | User send a tweet to the server that contains the !EQUIP command followed by the number 1           | User didn't know the slot numbers for each inventory slot.                        | Make inventory interface easier to use, this could be done by possibly drawing the slot ids on each slot.             |

*Table 17 - Usability Testing User 1*

From looking at the results that the first user produced, it is quite apparent that there were some things overlooked. This particular user tried to battle a monster by just using the Battle command without specifying a difficulty; this could be for a few different reasons. The most likely for this is that they didn't know how to specify the difficulty. Another possibility is that they assumed the battle would default to difficulty 1 if something was not specified.

Something else that has been learnt from these particular users results is that the user interface graphic isn't very informative when it comes to helping the user equip items. This particular user didn't know the number range for the different inventory slots.

## ***User 2***

| Task asked to perform                 | Expected user process                                                                               | Actual User process | Comments / Findings |
|---------------------------------------|-----------------------------------------------------------------------------------------------------|---------------------|---------------------|
| Register with the service             | User sends a tweet to the server that says !REGISTER                                                | As Expected         | No Findings         |
| Use the help command                  | User sends a tweet to the server that says !HELP                                                    | As Expected         | No Findings         |
| Do battle with a difficulty 3 monster | User sends a tweet to the server that has the !BATTLE command followed by the difficulty 3 argument | As Expected         | No Findings         |
| View the Inventory                    | User sends a tweet to the server that contains the !INVENTORY command                               | As Expected         | No Findings         |
| Equip item from inventory slot 2      | User send a tweet to the server that contains the !EQUIP command followed by the number 2           | As Expected         | No Findings         |

*Table 18 - Usability Testing User 2*

This particular user didn't have any issue whilst using the system.

### *User 3*

| Task asked to perform                 | Expected user process                                                                                                      | Actual User process                                                                            | Comments / Findings                                                                                      |
|---------------------------------------|----------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------|
| Register with the service             | User sends a tweet to the server that says !REGISTER                                                                       | User sends a tweet that says register but wasn't capitalised so the server didn't recognise it | Server needs to account for un-capitalised variations of commands.                                       |
| Use the help command                  | User sends a tweet to the server that says !HELP                                                                           | As Expected                                                                                    | No Findings                                                                                              |
| Do battle with a difficulty 2 monster | User sends a tweet to the server that has the !BATTLE command followed by the difficulty 2 argument                        | User tried to send a tweet saying !BATTLE2 to battle a difficulty 2 monster.                   | User didn't know how to correctly specify the battle difficulty, help command needs to be expanded upon. |
| View the Inventory                    | User sends a tweet to the server that contains the !INVENTORY command                                                      | As Expected                                                                                    | No Findings                                                                                              |
| Equip item from inventory             | User send a tweet to the server that contains the !EQUIP command followed by the inventory slot number they wish to equip. | As Expected                                                                                    | No Findings                                                                                              |

*Table 19 - Usability Testing User 3*

This particular user pointed out something that was overlooked in the initial development of the system, when trying to type commands for the service this user didn't use capitalisation and in return the system would not recognise the commands. As well as pointing out the capitalization issue, this user also confirmed that there was not enough clarity with how the battle system worked by also having trouble without using the commands.

Upon completion of the usability testing, it is clear that there were some issues that were overlooked due to knowledge of how the system works. From the tests it's clear that the help command can be expanded upon to better explain how the battle command works.

It has also become apparent that the server needs to be modified slightly to allow the use of commands that aren't capitalised as it might not be common sense to capitalise the commands.

The inventory interface could be modified slightly too to help the users better understand the id of each item slot to make equipping items a lot easier.

Although there were some issues identified in the Usability testing phase, they were relatively small and could be overcome with proper explanation on how to use the system. The next phase of testing is acceptance testing.

### **Acceptance Testing**

The acceptance testing phase is where I would gather feedback from a wider variety of people on all aspects of the service instead of a select few people on specific actions. To be able to do this efficiently I made a thread post on an internet forum explaining the purpose of the project as well as how to use it; once this had been done the service was then made available publicly for 24 hours.

At this time there were approximately 100 users that signed up to the service however not everyone left feedback about the service. Out of the 100 users that signed up only around 15 people left feedback.

Since the feedback didn't follow a particular template, users were free to voice their opinions on all the various aspects of the system.

The most common feedback received was that the quality of graphics were not the best quality. Although this is reasonable feedback, this wasn't helpful as the graphics used were placeholders.

Apart from feedback concerning graphics, there were a few users that had done a few monster battles and pointed out that there was no actual way to drop or delete items from the inventory; this was a slight oversight because without a way to do so, once the user had filled their inventory there would be no way for them to ever progress.

Unlike in the usability testing, there wasn't any feedback concerning the syntax of the commands, this could be due to the fact that the people who tested it were from a technological website and have more knowledge / experience with using commands the importance of being exact in what you type.

## Evaluation

To evaluate, this project was an overall success in that a system was successfully created that allowed users to play an RPG game through the use of twitter only. All of the information that was gathered in the literature review was backed up by the success of this project in particular it confirmed what makes a game playable on twitter.

Overall the objectives were met but only to a certain extent. Although they were all implemented, some of the implementations were basic and if not improved upon would become repetitive and the users would quickly get bored; an example of this is the battle system where although there are varieties of monsters that have different stats, the icons for the monster only have 2 variations.

The main aim of this project was to create a fully working game that can be played through twitter without the need of any special client or website and this was achieved.

Overall the answers obtained from this project are slightly different from what I expected. I thought that there was potential for fun and enjoyable games that can be played on twitter, however based on the findings on this project there are too many restrictions and the platform is simply not designed to support games.

If the project was to be done again it would most likely be done on a different social media platform such as Facebook as there is much more freedom with what can be done.

However if the project had to for whatever reason be on the twitter social networking platform then a better documented API would be used to make things easier in the earlier stages in development.

## **Conclusions and Recommendations**

To summarise this project, it was broken down into sections to give an overview of how the project was carried out. The sections were Literature review, research and development and design and implementation.

The literature review sections covered areas such as the background of twitter and how twitter has progressed over the previous years. It also outlined pre-existing twitter games and what common features each of them shared that made them playable on twitter.

The research section outlined what the chosen research and development methodologies were and how they were going to be carried out.

The design and implementation section detailed the design of the project by creating a list of requirements that the project must meet to be considered complete. This section also contained various different diagrams to help illustrate the process of the project and how progress was made.

As outlined in the evaluation, twitter in the traditional sense is not a suitable platform for social gaming, this being said a possible avenue of research is how twitter could be integrated into games that aren't hosted on the twitter platform.

## **Reflection**

Throughout the project a huge amount of knowledge has been gained. The first thing that stands out to me is the importance of checking things all the way through before making a decision. This became apparent when I had decided on using the TweetInvi API that I thought was well documented but turned out the documentation was out of date and not very useful.

Although I mentioned that TweetInvi was not well documented, I am still happy that this is the API that I chose as it was written in C# which is the language that was most commonly taught during my time at Swansea Trinity St David University.

Overall this project has given me more confidence in both my programming ability as well as problem solving, even though there was help available from both my lecturers and supervisor alike I feel that as I was able to solve the issues encountered without any help that this project is truly mine.

## References

- [1] "Twitter Help Centre" [Online]  
Available: <https://support.twitter.com/articles/215585>  
[Accessed 10/12/2015]
- [2] "The Financial Times Limited - An abridged #history of @Twitter" Oct 5, 2015
- [3] "Founders Twitter" [Online]  
Available: <https://twitter.com/jack/status/20>  
[Accessed 10/12/2015]
- [4] "Twitter Milestones" [Online]  
Available: <https://about.twitter.com/company/press/milestones>  
[Accessed 10/12/2015]
- [5] "Chris Messina Twitter" [Online]  
Available: <https://twitter.com/chrismessina/status/223115412>  
[Accessed 10/12/2015]
- [6] "Why didn't the creator of Hashtag patent the concept" [Online]  
Available: <https://www.quora.com/Why-didnt-the-creator-of-Hashtag-patent-the-concept>  
[Accessed 11/12/2015]
- [7] "Twitter Taps Dick Costolo as COO" [Online]  
Available: <https://gigaom.com/2009/09/02/twitter-taps-dick-costolo-as-coo/>  
[Accessed 11/12/2015]
- [8] "Promoted Tweets" [Online]  
Available: <https://blog.twitter.com/2010/hello-world>  
[Accessed 11/12/2015]
- [9] "Tweet Preservation" [Online]  
Available: <https://blog.twitter.com/2010/tweet-preservation>  
[Accessed 11/12/2015]
- [10] "Promoted Trends"  
Available: <http://www.pcmag.com/article2/0,2817,2365233,00.asp>  
[Accessed 11/12/2015]

[11] "Promoted Accounts"

Available: <https://blog.twitter.com/2010/promoted-promotions>

[Accessed 11/12/2015]

[12] "Casual Gaming"

Available: <http://www.computerhope.com/jargon/c/casual-gaming.htm>

[Accessed 13/12/2015]

[13] "Relationship between social games and casual games" [Online]

Available: <https://www.quora.com/What-is-the-relationship-between-social-games-and-casual-games>

[Accessed 13/12/2015]

[14] "10 Real time twitter games" [Online]

Available: <http://www.makeuseof.com/tag/10-real-time-twitter-games-enjoy-tweet/>

[Accessed 14/12/2015]

[15] "Facebook Apps Leaderboard" [Online]

Available: [http://www.appdata.com/leaderboard/apps?action=apps&cat\\_id=400&controller=leaderboard&facebook\\_metric\\_id=81&fanbase=0&list\\_select=apps&metric\\_select=mau&p=61](http://www.appdata.com/leaderboard/apps?action=apps&cat_id=400&controller=leaderboard&facebook_metric_id=81&fanbase=0&list_select=apps&metric_select=mau&p=61)

[Accessed 14/12/2015]

[16] "Twitter Developer Terms of Service" [Online]

Available: <https://dev.twitter.com/overview/terms>

[Accessed 14/12/2015]

[17] "Twitter Gambling Policy" [Online]

Available: <https://support.twitter.com/articles/20170426>

[Accessed 14/12/2015]

[18] "TweetInvi Website" [Online]

Available: <https://tweetinvi.codeplex.com/>

[Accessed 15/12/2015]

[19] "TweetInvi API Documentation" [Online]

Available: <https://github.com/linvi/tweetinvi/wiki/Introduction>

[Accessed 15/12/2015]

[20] "Deductive Approach" [Online]

Available: <http://research-methodology.net/research-methodology/research-approach/deductive-approach-2/>

[Accessed 20/12/2015]

[21] "Research Approach" [Online]

Available: <http://research-methodology.net/research-methodology/research-approach/>

[Accessed 20/12/2015]

## **Appendix**

### **Installation**

The installation process a little complicated as the system was mainly designed to be run on a specific machine; however it is possible to run it on any system with a bit of effort.

#### **Setting up the database**

The following steps will walk through the changes required to successfully setup the database to be used with the system.

1. Install WAMP Server
2. Copy the Files from the folder marked Database to the "C:\wamp\bin\mysql\mysql(version)\data" directory.
3. Now that the database is ready to be used you must next modify the server to connect to the database; by default the WAMP server will use the username root without a password, if this is left as default the system should work but encase you need to modify the code you can search for "myConnection" in the code to modify database connection strings.

#### **General System Setup**

Following the below steps will ensure that the system will work on any machine.

1. Copy the Folder called Images to somewhere on your machine (Make a note of this path)
2. Search the code for all references to "C:\Users\Andrew" and then modify each path to the corresponding folders.
3. To run the system simply open in visual study and press run (Note: there is no user interface).