# CSCE 606 PROJECT REPORT

## MONKEYCHAT – A STARCRAFT 2 UNIT ONLINE QUERYING SYSTEM

2013 FALL


**Shuyu Wu (921005246)**

16.6%. He wrote part of the code, wrote part of the report.

**Qili Huang (322008251)**

16.6%. He wrote part of the code and wrote part of the database entries.

**Ting Chen (921004583)**

16.6%. She wrote part of the code and wrote part of the database entries.

**Xiaohui Shen (822009069)**

16.6%. He wrote part of the code, wrote part of the report.

**Yang Wang (321004893)**

16.6%. He wrote part of the code and wrote part of the database entries.

**Yang Zhou (822009072)**

16.6%. She wrote part of the code and wrote part of the database entries.

# I. INTRODUCTION

## 1.1 Topic

We implemented a StarCraft 2 Units Query Answering System, named “MonkeyChat”. By following its “public account” in WeChat (a chatting app available on most popular mobile operating systems), users can type in commands to get a detailed description of the StarCraft2 unit he wants to search for.

## 1.2 Programming Language

Python is selected as the programming language used in this project as:

1. It’s suitable for rapid software development, and
2. System performance is not a big concern in this project.

## 1.3 Programming Environment

The system relies on the WeChat Official Account Platform and the Platform as a Service (PaaS) systems provided by Appfog and MongoLab.

The WeChat Official Account Platform allows developers to create “public accounts” that can interact with users automatically. For example, WeChat user can subscribe to some public account which provides weather information and, by sending weather query information, the public account will reply with the weather information the user needs. Another example is that a user can subscribe to the public account provided by “Evernote” and ask it to save some information to the user’s Evernote account.

PaaS is a service model of cloud computing which provides a computing platform and a solution stack as a service. Normally development teams who want to create web applications need to setup hosting machines and install servers and databases before they can start to implement those web applications. This means that development teams not only need to know how to write software, but they also need to know how to setup and maintain a stable running environment for those web applications, which can be quite troublesome. PaaS providers ease the application deployment processes for the development teams by providing the running environment of web applications so that development teams can focus on software development and start immediately; plus, they just need to pay for the computing resources they use.

Both Appfog and MongoLab are PaaS providers. For MonkeyChat, the web server running environment is provided by Appfog, and the database is provided by MongoLab.

## 1.4 Methodology

For this project we followed a software development methodology that is similar to Extreme Programming.

## 1.5 Expect to Learn from the Project

1. Gain more experience with the Python Programming Language
2. Gain some experience with PaaS platform
3. Gain some experience with NoSQL database
4. Gain some experience with Extreme Programming

# II. WHAT WE DID IN THE PROJECT

## 2.1 Function Demonstration

Here is a set of screenshots of the MonkeyChat.

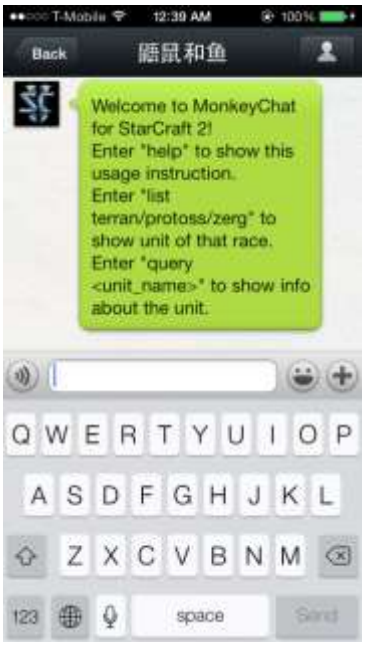


**Figure 1 Welcome message**

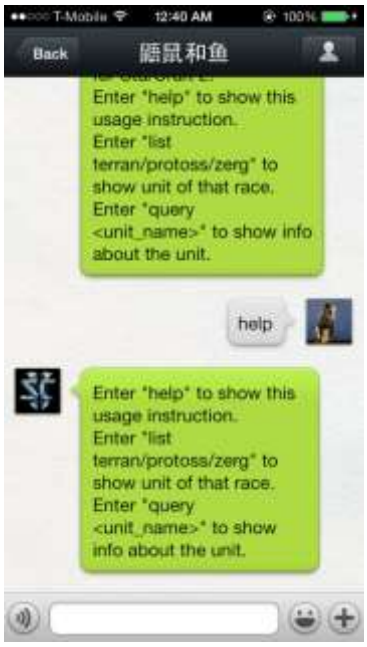


**Figure 2 Enter "help" to get usage instructions**


**Figure 3 Enter "list terran" to get a list of units that belong to the race "Terran"**


**Figure 4 Enter "query banshee" to query the specifications of the unit "Banshee"**


**Figure 5 A snapshot of the detailed information page (part 1)**


**Figure 6 A snapshot of the detailed information page (part 2)**

Here is a quick explanation of the screenshots:

- The Figure 1 shows that when a user subscribe to the MonkeyChat public account, he will receive a welcome message which shows how to interact with MonkeyChat.
- The Figure 2 shows that when the user enters “help”, he will get a reply with the full usage instructions.
- The Figure 3 shows that when the user enters “list terran”, he will get a full list of units that belong to the race “Terran”.
- The Figure 4 shows that when the user enters “query banshee”, he will get a card which contains a picture of the unit “Banshee”, a short description of it and a button that leads the user to a page containing a full specification of the unit.
- The Figure 5 and the Figure 6 are parts of the snapshots of the page containing the detailed information of the unit “Banshee”.

The MonkeyChat will also reply with specific error notices when the input is not correct – for example, when a user enters a wrong command.

## 2.2 Working Mechanism Overview

### 2.2.1 When the MonkeyChat Server Registers Itself to the WeChat Server

1. The WeChat server will send a HTTP POST message to the MonkeyChat server’s root URL, which contains some validation information – “signature”, “timestamp”, “nonce” and “echostr”
2. The MonkeyChat sever will do a specific kind of hashing based on the “timestamp”, “nonce” and a token agreed by both the WeChat server and the MonkeyChat server. Then it will compare the hash result with “signature”:
   a. If they matches, the MonkeyChat server will reply with the “echostr”
   b. Otherwise reply with anything other than “echostr”
3. If the WeChat server correctly receives the “echostr”, then the MonkeyChat server will register itself successfully to the WeChat server

### 2.2.2 When a User Enters a Command or a User Subscribes to the MonkeyChat’s Public Account

1. The WeChat app will generate a WeChat message and sends it to the WeChat server
2. The WeChat server will forward the message through HTTP GET method to the MonkeyChat server
3. The MonkeyChat server will check if the incoming message is sent by the MonkeyChat server:
   a. If it is, then MonkeyChat will generate a reply message containing the information needed by the user (or an error notice), and sends it back to the WeChat server
   b. If it is not, then MonkeyChat will reply with an error message
4. The WeChat server receives the reply message and checks If it is within 5 seconds since it forwards the message to the MonkeyChat server:
   a. If it is, it will forward the reply message back to the user
   b. Otherwise no message will be forward back to the user

### 2.2.3 When a User Wants to View the Detailed Information of a Unit

1. The WeChat server will receives an HTTP GET message asking for that page from the user directly
2. The WeChat server will then query the database, looking for relevant information:

a. If it successfully finds that information, it will render the page based on an existing HTML template with that information and return the page to the user directly
b. If not, then returns an HTTP ERROR 404 page back to the user directly

## 2.3 Structure of MonkeyChat

MonkeyChat has a very simple and neat structure. From Figure 7 we can see that there are 5 modules.

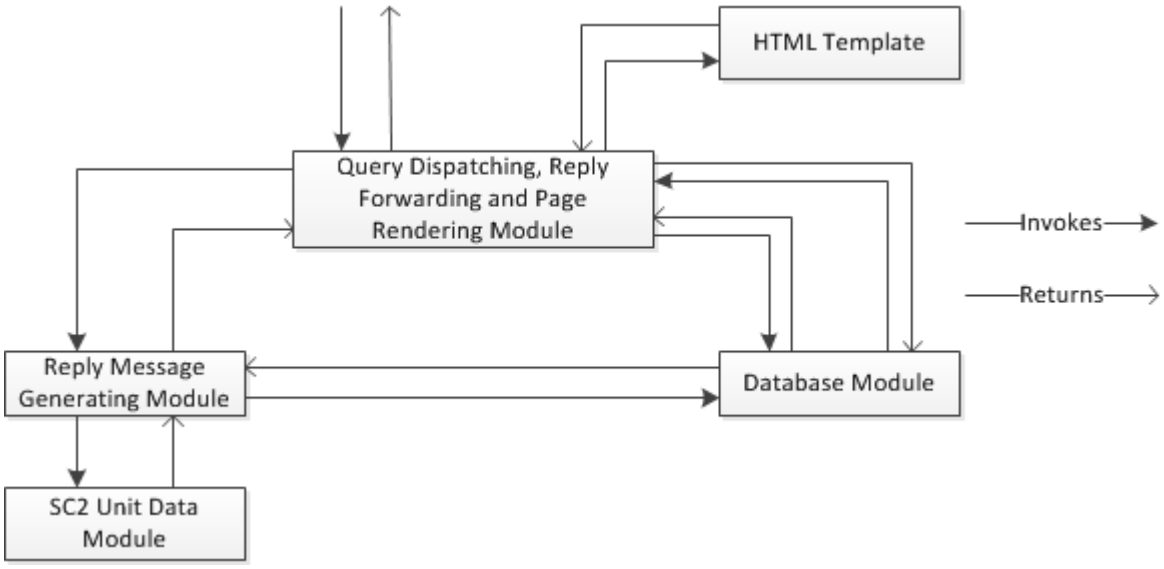


**Figure 7 Structure of MonkeyChat**

Here is a summary of each of the modules:

1. The *Query Dispatching, Reply Forwarding and Page Rendering Module* can be regarded as the server. It's the interface which communicates with the user/WeChat server.
2. The *Reply Message Generating Module* is the core module for MonkeyChat. It determines the reply message to be returned, based on the incoming message.
3. The *Database Module* contains interfaces to the database, as well as some helper functions and constants
4. The *SC2 Unit Data Module* contains all details of each of the StarCraft 2 units.
5. The *HTML Template* is a template that can be used whenever the details of a StarCraft 2 unit need to be displayed to the user in HTML page.

### 2.3.1 When the MonkeyChat Server Registers Itself to the WeChat Server

As mentioned above, the WeChat server will send an HTTP POST message to the MonkeyChat server. In this case the message will be handled solely by the *Query Dispatching, Reply Forwarding and Page Rendering Module*.

### 2.3.2 When a User Enters a Command or a User Subscribes to the MonkeyChat's Public Account

As mentioned above, the MonkeyChat server will receive an HTTP GET message containing the WeChat message.

1. This message will be first validated by the *Query Dispatching, Reply Forwarding and Page Rendering Module* to make sure it is from the WeChat server. If it is not, then an error page will be returned.

2. If it is, then that module will ask the *Reply Message Generating Module* to generate the actual content to be returned to the WeChat server. Based on the "MsgType" tag within the WeChat message, different handlers will be invoked to generate the reply message:
    a. If "MsgType" is "event", then the incoming WeChat message indicates that there is a user subscribing or unsubscribing event. The event handler will be invoked to deal with this message.
    b. If "MsgType" is "text", then the incoming WeChat message contains some user input. The text handler will be invoked to deal with the message:
        1) If the input is "help", returns usage instructions.
        2) If the input is "list <race>", invokes the *SC2 Unit Data Module* and looks for that race and returns unit list of that race (or an error message if the race does not exist).
        3) If the input is "query <unit_name>", invokes the *Database module*, searches the database and returns data of that unit (or an error message if that unit does not exist).

### 2.3.3 When a User Wants to View the Detailed Information of a Unit

As mentioned above, the MonkeyChat server will receive an HTTP GET message directly from the user.

1. The *Query Dispatching, Reply Forwarding and Page Rendering Module* will extract the name of the unit from the HTTP GET message first, and then invoke *Database Module* to query the database. If there is no such unit, an HTTP ERROR 404 page will be returned to the user.
2. If the unit is found, the data of the unit is retrieved from the database, and then the *Query Dispatching, Reply Forwarding and Page Rendering Module* will invoke the HTML template to render the data in a pre-defined format.

# III. THE SOFTWARE DEVELOPMENT METHOLODOGY

## 3.1 Frequent Iterations

For this project we have followed a software development methodology that is similar to Extreme Programming: we have used frequent small iterations, each of which implements a small set of features, to develop the software:

1. In the 1st iteration, we spent some time understanding the messaging mechanism of WeChat, such as how to register MonkeyChat server to the WeChat server, the format of the WeChat message. After that we spent some time figuring out how to create a web app on Appfog. Then a skeleton was created on Appfog such that it could register itself to the WeChat server.
2. In the 2nd iteration, we wrote some code to make sure MonkeyChat could reply to the most basic command, i.e. "help". Also we wrote some code to make sure it could handle error cases. We then refactored the code such that there were two modules, i.e. the *Query Dispatching, Reply Forwarding and Page Rendering Module* and the *Reply Message Generating Module*.
3. In the 3rd iteration, we spent some time understanding the basics of MongoDB, how to use the Python Driver of MongoDB, and the usage of the MongoDB-as-a-Service provided by MongoLab. We then created the *Database Module* and implemented the module such that the *Reply Message Generating Module* could query the database through this module. At this time, we inserted only a few of database entries (known as "documents" in MongoDB) just to make sure the *Database Module* could work correctly.
4. In the 4th iteration, we wrote code to make sure all commands were supported. After testing that all these commands had been supported correctly, we refactored the code to keep a good module design.

In this iteration, we also wrote a very basic HTML template so that we could test if the "query <unit>" command worked correctly.

5. In the 5th iteration, we inserted all the database entries and improved the style of the HTML template.

## 3.2 Practices

### 3.2.1 User Stories

User stories are written by customers as features that the system needs to have. Each user story is described by only 3 to 5 sentences and only needs to be detailed enough to make low risk, reasonable estimate on how long the user story will take to be implemented.

For this project, at first four user stories were written: 1) "help" command 2) "list <race>" command, 3) "query <unit>" command 4) "random" command. The first 3 out of the 4 are of high priority, and we implemented MonkeyChat in a feature-by-feature manner, high priority first.

Note that due to the time limitations the 4th command is not implemented. However the first 3 commands work pretty well. This shows that developing software in a feature-by-feature manner has its advantage as no time is wasted on features that are less-important should the time is not enough to implement those less-important features.

### 3.2.2 Release Planning Meeting

Release planning meeting is used to specify which user stories should be implemented in which release, and the schedule (i.e. the deadlines of all the releases).

For this project, we held a release planning meeting to determine the priority of each of the user stories, in which iteration each of the user stories should be implemented and the length of each iteration. See section 3.1 for more details.

### 3.3.3 Acceptance Tests

For each user story, just before it was going to be implemented, we determined the test cases for that user story. After each user story was implemented, all previous test cases, include the test cases for the current user story, were executed to make sure that there were no regressions.

### 3.3.4 Refactoring

We refactor our code frequently to keep the logic clean and easy to understand to make sure it's easy to maintain. Through this process we also try to make sure modules have low coupling and high cohesion.

### 3.3.5 Coding Style

Python has an advantage that the language itself forces programmers to write code in similar styles. Furthermore, by adhering to the PEP8 (style guide for Python code) and constantly running our code through a style checker named "pep8", we have made the code written by different team members follow same coding style.

# IV. RESULTS, DISCUSSIONS AND KEY LEARNINGS

## 4.1 Results

See section 2.1 for function demonstrations.

## 4.2 Discussions and Key Learnings

### 4.2.1 The Last 20% of the Work Often Take 80% of Time

It is not difficult to write the first 80% of the code; however it might take longer than expected to deal with the rest 20% of the code. This is because that for the first 80% percent of the code we are usually trying to implement features, which often are relative easy. However for the rest of our work we are usually trying to test, debug and fix the code, refactor the code and improve its stability and scalability.

For our project, even though the code testing, refactoring and improving process does have taken a lot of time, though not as much as 80%. It is worth noticing that much time is spent on code refactoring – it's easy to implement commands such as "help, "list <race>" and "query <unit>" as there are so many ways to do it, but to make sure the code is easy to maintain is not that easy: we need to make sure that the modular design has low coupling and high cohesion, the name of variables and functions are easy to understand and straightforward, etc.

### 4.2.2 Choose Features to be Implemented First Wisely

A research in 2002 carried by The Standish Group shows that roughly 80% of the users only use 20% of the features of a software system. Based on the fact that delayed delivery and budget overrun are common, this indicates that when applying Agile Methods to our software development activities, we often should first focus on implementing those software features that are most frequently used.

For our project, apparently the most frequently used feature is the "query <unit>" command. To guide users on how to use it, we need to implement both the "help" command and the "list <race>" command as well. The rest features are optional and can be implemented later.

### 4.2.3 Team Cooperation Is Not Easy

The amount of work done by six people is not equal to six times the amount of work done by one people. Different people have different backgrounds, and it needs time for people to learn until they are capable enough to handle the work on their own, in most cases. Also, different people have different coding styles. Though in our project, Python is selected as the main coding language, which prevents people from writing code in dramatically different coding styles, an agreement on coding style still needs to be reached to make the project more maintainable.

# V. Appendix

## 5.1 Source Code

The source code can be found on Github:

https://github.com/shuyuwu/monkeychat.git

## 5.2 How to Use

1. Download WeChat, install it on your Android/iOS mobile phone
2. Follow the MonkeyChat account: add by searching for ID “squirrel-and-fish”
3. Follow the usage instructions in the welcome message sent by the MonkeyChat public account