CSIT691

Independent Project

Spring 2010

Real Time Multiplayer Online Facebook Game

Supervisor: Professor David Rossiter

Student: Yip Siu Hung, Penny (09964376)

1. Introduction

There are tremendous number of Facebook games available. Most of them are single player whilst some of multiplayer games like Restaurant City, Farm Villa offers a platform for different players to play together. However, there is no way for the players to interact with the others in real time. This trend motivated the project – Building a simple real time multiplayer Facebook game and to investigate the difficulties that drove the game developers away from developing real time multiplayer game and its possible solution. The game was built using Adobe Flex 3[1] together with an Open Source streaming server called Red5[2]. This report will first introduce the game play, followed by the technology used in the backend, difficulties faced and the resolutions.

2. Game Play

The game I developed is a Pictionary-liked game[3]. The system will send the one of the player a random picture, and this player has to draw the picture he saw on the painting area. The drew picture will be reflected on the other player's screen and he has to make the guess. If the guesser guesses it correctly, the system will generate another image for another guess. If the players made 5 correct guesses, the game ends and will record the time spent in the game. The top score will be shown on the score board.

The game has three main components:

1. Game Lobby
2. Game Room
3. Score Board

Game lobby is the entrance point of the game. When the player launches the application, he will see the lobby with different tables and seats. If the player clicks

the empty seat, it means he has joined that table. The lobby is synchronized which means the other players started the game knows which seats are occupied in real time.

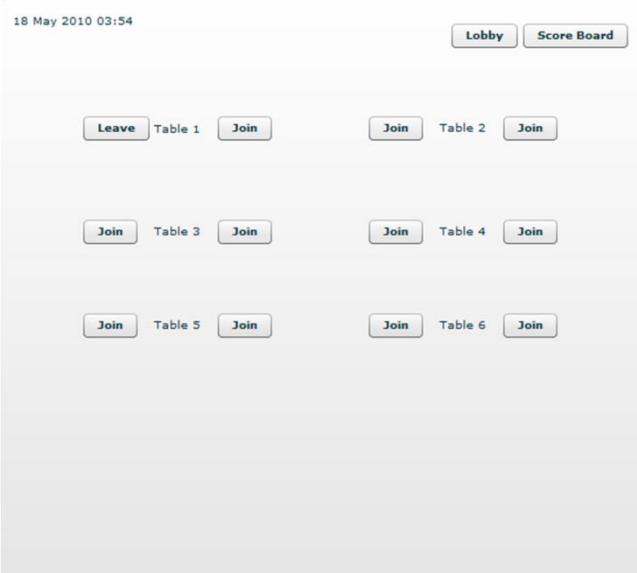


Figure1: Player 1 Joined the seat

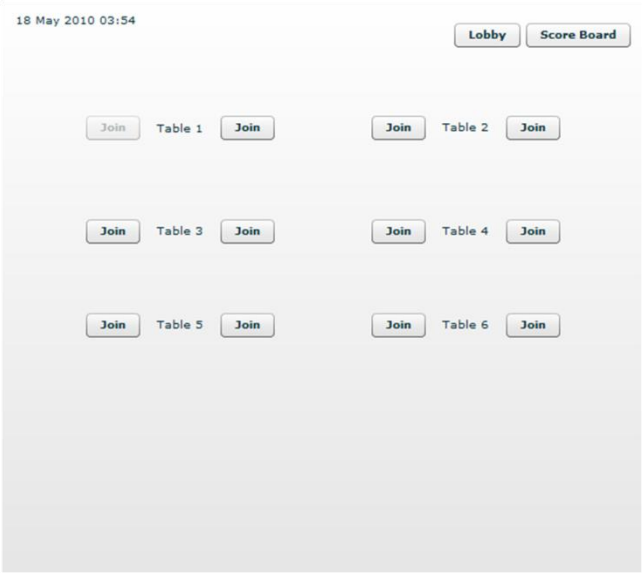


Figure 2: Player 2 sees the seat is occupied

When the table has 2 players joined, either of them can start the game by clicking the start button. The 2 players will be sent to a role selection page. In this page, the player can choose to be a drawer or a guesser. Of course, only one of them can be a drawer or guesser.

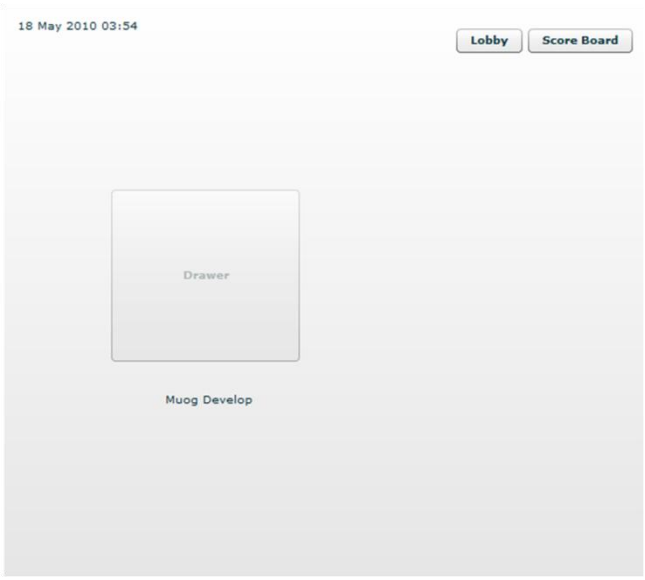


Figure 3: Player 1 as Drawer

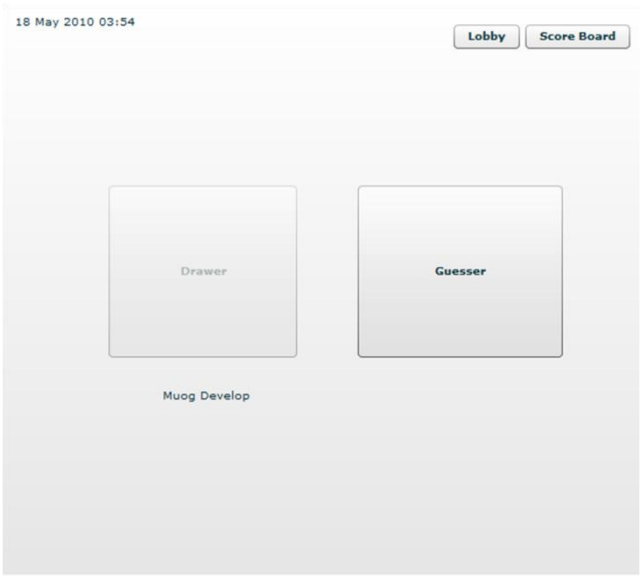


Figure 4: Player 2 can only choose as Guesser

After both of the players chose their role, they can start the game by pressing the "Start" button. The game will then prompt a timer from 3 to 0 letting the players to get ready. When the timer runs out, they will be sent to the game room. In the game room, the drawer can see a random picture generated by the server, and he can draw he saw on the drawing area and it will shows on the guesser's panel in real time. For the guesser, he will see a question mark and is not allowed to draw on the panel. What he can do is look at what the drawer drew and guess what he saw. The guesser can type what he guess in the text box, if it's correct, it will be counted and will go to another picture automatically.

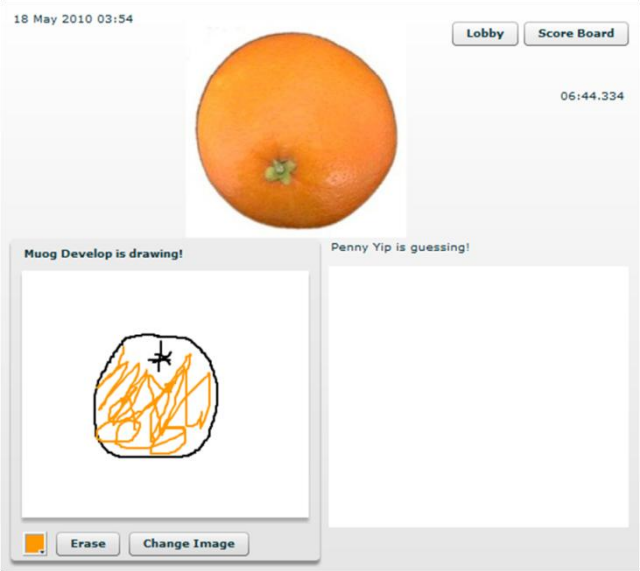


Figure 5: Scene of Drawer

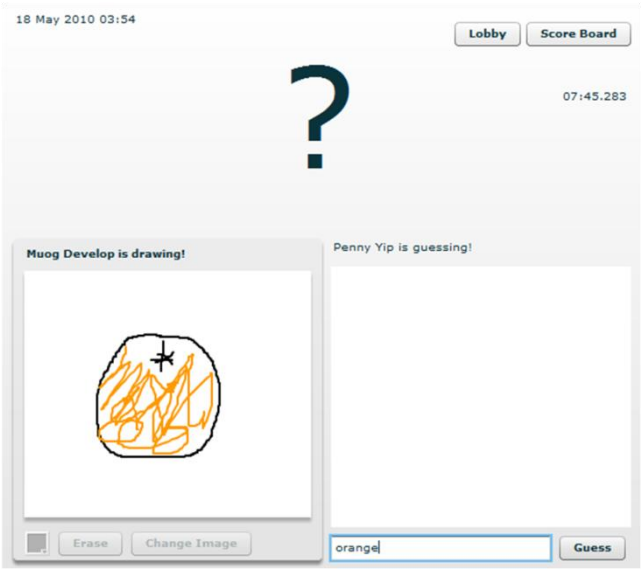
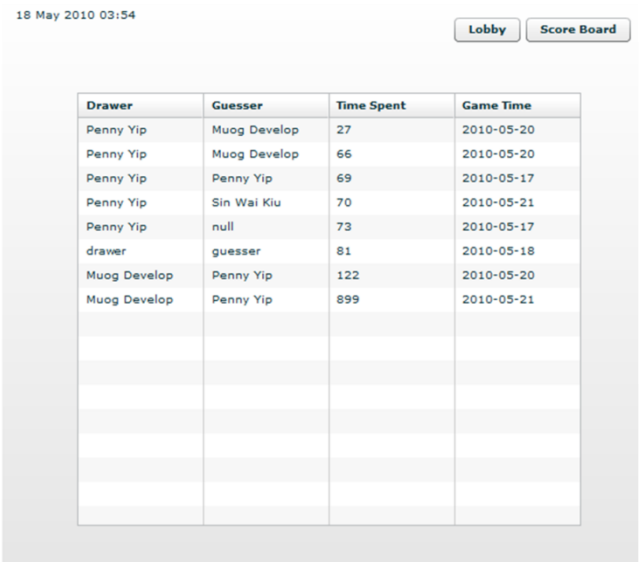


Figure 6: Scene of Guesser

The game goes on with 5 pictures for the players to guess and after the game is done, the game will ends and the time spent will be recorded on the database. Players can view the top 20 highest score by clicking the “Score Board” button.



Drawer	Guesser	Time Spent	Game Time
Penny Yip	Muog Develop	27	2010-05-20
Penny Yip	Muog Develop	66	2010-05-20
Penny Yip	Penny Yip	69	2010-05-17
Penny Yip	Sin Wai Kiu	70	2010-05-21
Penny Yip	null	73	2010-05-17
drawer	guesser	81	2010-05-18
Muog Develop	Penny Yip	122	2010-05-20
Muog Develop	Penny Yip	899	2010-05-21

Figure 7: The Score Board

This is all about the game play, very simple but demonstrated we can develop a real time multiplayer game on Facebook. It can even synchronize the drawing in real time with no issue. In the following technical part, I will talk about how it was done and the difficulties faced and the resolutions.

3. Technical Details

Before discussing the technical details in depth, let's have a look from a high level perspective.

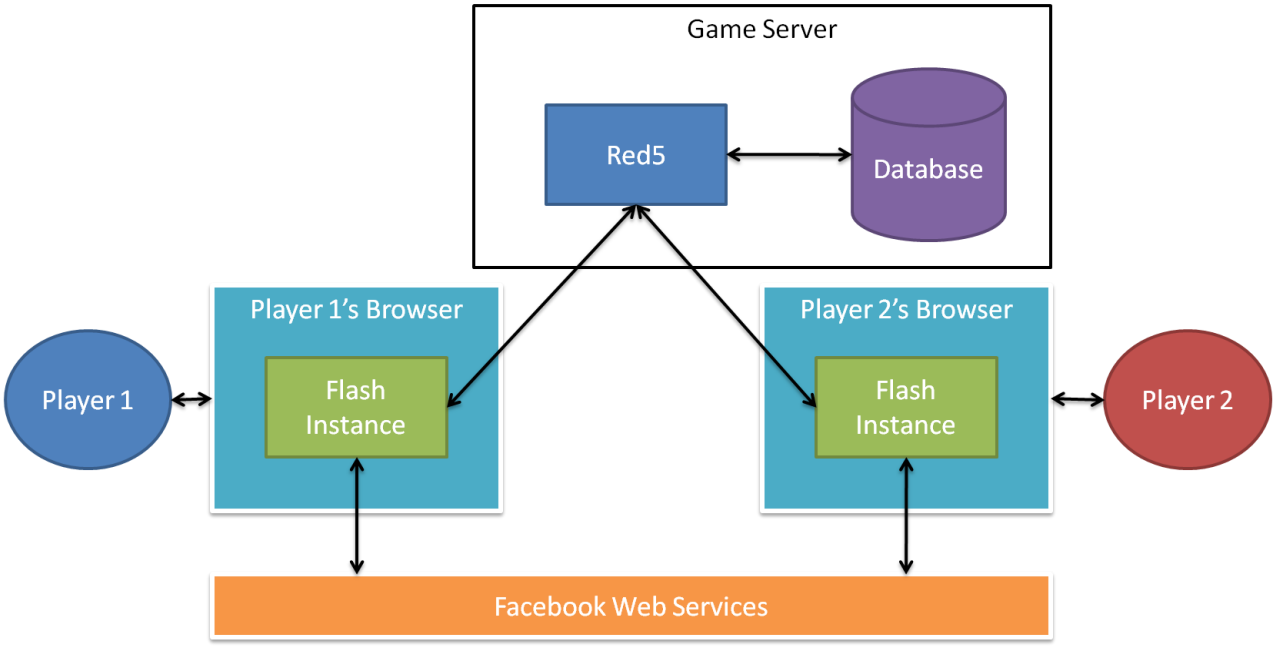


Figure 8: High Level Architecture of the Game

Figure 8 shows the high level architecture of the project. As you can see, all the flash instances are connected to the Red5 server and Facebook so they can synchronize with each other and retrieve Facebook's information via their API. Besides, in order to keep track of the game records and the status of the game, Red5 is connected with the database. The database connection remains on Red5 does not mean we cannot do that on the flash instance, it is actually a security measure and more will be discussed in the following part.

3.1 Communicating Flash with Red5

From the high level diagram, the Red5 and Flash connection is the most important component in the project as it provides the ability for the flash instances to

synchronize. The Red5 and Flash connections are made using RTMP[4] which was designed for real time message transmission based on TCP[5]. With the design of the protocol, it makes the synchronization much faster than using other protocols as it has a very small overhead in the TCP packet.

The way we used Flash to communicate with Red5 is very simple as Red5 is an Open Source clone of Adobe's Flash Media Server[6]. That is to say it has most of the native interfaces for Flash to make connections with the Red5 server and broadcast to other flash instances. It makes the development much simpler and easier.

One thing that worth mentioning is all flash instances are synchronized, which means all of them will see the same screen. In the project, I used multiple connections and the concept of scope to handle the problem.

For each flash instance connecting to the lobby, it uses one main connection which can be used to update the lobby's status and view the score board. Whenever the players start a game, it will make another private connection with a private scope. With the private connection, their game play will not affect the others whilst the rest shall not be able to distract them. Whenever the game ends, the private connection will be closed and released and the players will be redirected back to the lobby.

On the other hand, from the high level architecture of the project, you can see the database connection is connected to the Red5 server. As we mentioned before, it is not connected directly through the flash instances due to security issues. We know flash can be decompiled very easily. With the decompiled files, crackers can read the ActionScript very easily. Thus, if we put any secret strings like database hostname and password in the ActionScript, it equals to give the crackers the secrets. The

structure we now have is much more secure. Flash itself only handles a minimum number of operations; all the business logics and connections to other places are handles by the Red5 server and thus, Java. As Java is considered to be safe and secure, it makes the game much more secure.

3.2 Connection with Facebook

Connecting flash instances with Facebook is another important step in the project as it can share the power of the social network. For instance, retrieving friend list, user's profile picture, sending notifications, etc. However, in order to make flash communicate with Facebook is not as easy as it should be.

According to Adobe, there are two different ways to embed Flash into Facebook and retrieve the information from Facebook, they are iFrame and FBML[7]. For greater flexibility, I have used the iFrame for the project.

In order to make flash communicate with Facebook with iFrame, we can use the following steps (Graphical illustration in figure 9):

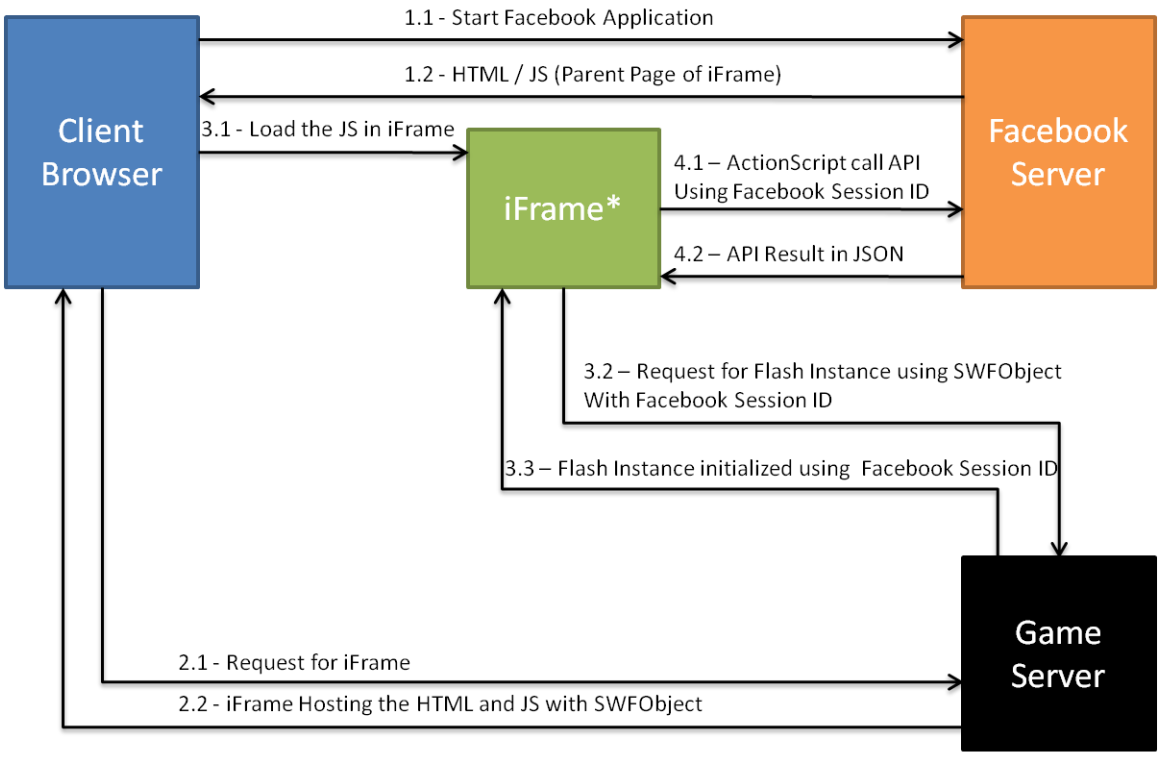
1.1 Player start the Facebook application using browser
1.2 Facebook Server response with the HTML page

2.1 Client browser request for the iFrame page specified in the HTML page received in step 1.2
2.2 Game server response with the iFrame page which includes the HTML, JavaScript with SWFObject

3.1 Client browser loads the JavaScripts in the iFrame
3.2 JavaScript in iFrame will send the Facebook session ID to the game server and request for the flash instance
3.3 Game server response with the flash instance which was initialized using the Facebook session ID

4.1 Client side's flash instance use ActionScript to call Facebook's API using the session ID

4.2 Facebook returns the query result in JSON format



*iFrame should be inside the Client Browser, it is moved out intentionally for showing the data flows

Figure 9: Mechanism for Calling Facebook API using Flash with SWFObject

The above approach requires the following components:

1. The Facebook API for Flash
2. An HTML page with JavaScript to communicate with SWFObject
3. SWFObject

When the client launches the application on Facebook, it will launch the iFrame hosted on our server. The iFrame has some JavaScripts to communicate with the SWFObject, that will pass the session key of the client on Facebook to the SWFObject. Afterwards, we can retrieve the session key from the flash instance. After getting the

session key of the client, we can then use the Facebook API to initialize the Facebook object and then retrieve the information from Facebook using AMF or XML and we finally has to extract the AMF and finally get the information and bind it to proper controls.

It sounds quite straight forward if you know the idea but it can be difficult if this is the first time deploying flash application on Facebook.

4. Difficulties and Resolutions

4.1 Red5 Responding Slowly

At the beginning of the implementation, when I was implementing the drawing panel, I found the latency was severe and it was not acceptable. At the time, I noticed the Red5 server has a 100% CPU usage and that caused the delay. At that time, I suspect there were too many connections fired at the same time. It is because when the player move the mouse for one pixel, it will fire a connection. Therefore, I tried to limit the number of connections to the server. However, it makes the drawing becomes very bad as most of them becomes straight lines and the guessers are not going to be able to recognize the drawings.

Intricately, I think the server should be able to handle the connections as the server can handle video streaming and should be capable for the drawing mechanism. Therefore, I dig into the source code of the Red5 server and found several solutions for the problem.

First, I figured how to make the server more efficient by making it a worker process [9] and by changing some Java codes. Besides, I managed to make the Red5 server

accepts concurrent connections. That makes the server can handle connects more efficiently by eliminating the waiting overhead when one connection is waiting for the other to be processed.

Second, I changed some of the Java code on Red5 such that it can use simple data structures to handle the received packets. By default, Red5 uses Map<String, Object> to handle the received connection. We know the Map container has a very big overhead and require several serializations and deserializations on the Object in the Map. I overrided the method and make it also accepts other data types such as Concurrent Hash Map, String, int, etc. With the modification, it saved a lot of processing power and can handle more connections at the same time. This method ultimately solved the problem. The server's CPU usage is around 5% if one player drawing continuously on a P4 machine with 1GB memory.

4.2 Stateless

Another challenge I had was Red5 is a streaming server, and it is not difficult to imagine it is stateless. It makes the lobby management very difficult as we do not know which room has been occupied or released.

In order to solve the problem, I have created a simple database modeling different rooms' status. Every time the room has updated, the database entry will be updated and then broadcast the latest status of the rooms to all the connected players. It is an easy and efficient way to handle the room status.

4.3 Insufficient EventHandlers

There is a problem with the existing design of ActionScript 3. There are two components in ActionScript to communicate with Red5, the NetConnection and

SharedObject, there is event handler for the program to know the connection has been made successfully, and then we can bind the SharedObject with the connection. However, if we wanted fire a message to the server asking for the room status right after the SharedObject is binded to the connection, there is no way with the existing design.

In order to make the player can retrieve the room information right after connection, I have to create the event handler on my own by extending the EventDispatcher and tell the event listeners the SharedObject has been binded to the connection.

4.4 Insufficient Documentation

Another difficulty I faced when developing the game was due to the insufficient Red5 documentation. Red5 is an Open Source project actively developed in year 2008, however, after 2 years time, most of the link on the official web site is not working. Even the JIRA issue tracker is broken as the project moved to the Google Code Project. Without sufficient documentation, it makes the development very difficult at the beginning. Fortunately, the project is Open Source and I can look into the code and see what it actually does and can therefore figure what approach is the best for the game.

4.5 Different Data Formats

Last but not least, the difficult goes to the communication of ActionScript and Java. Since we have all the business logic implemented on the Red5 server, what we need extract the information sent from the flash client and manipulate the data and send it back to the client. This is a difficult step if you do not know what data type matches to the others on the two platforms. For example, Object in ActionScript means Map<String, Object> in Java and ByteArray in ActionScript means byte[] in Java.

Besides trying the data structures one by one, I found a subtle yet very useful page [8] from Adobe's documentation. It specifies all the possible mapping with ActionScript and FMS. With the information, I managed to map any possible data structure with the others and make the development becomes much easier.

5. Possible Further Development

The basic structure of the game is done and there are many possibilities for further development. For instance, we can make fully utilities the power of the game with the enormous power offered by Facebook - Getting the players' mutual friends' picture and use them as the picture for guessing. Also, the drew images can be stored on the server temporarily and show the players what can they have drawn compare with the original image.

6. Conclusion

The successful deployment of the game marks the goal of the project has been accomplished. With the aid of the open source project Red5, I managed to develop a real time multiple player Facebook game. The hurdles I faced during development reflected the fact that not a lot of developers are going to develop a real time game on Facebook. However, with the successful deployment of the game, it means we have already set up a platform and with sufficient knowledge to build any kind of real time multiplayer Facebook game using the Red5 server.

7. Reference

[1] Open source framework, web application software development | Flex – Adobe. http://www.adobe.com/products/flex/. Retrieved on 28 May 2010.

[2] Red5. http://red5.org/. Retrieved on 28 May 2010.

[3] HASBRO – PICTIONARY. http://www.hasbro.com/shop/details.cfm?guid=96C13CEE-19B9-F369-D9E8-73D9C5517F50&product_id=25575&src=endeca. Retrieved on 28 May 2010.

[4] Real-Time Messaging Protocol (RTMP) specification. http://www.adobe.com/devnet/rtmp/. Retrieved on 28 May 2010.

[5] TCP, Transmission Control Protocol. http://www.networksorcery.com/enp/protocol/tcp.htm. Retrieved on 28 May 2010.

[6] Media server for streaming video | Adobe Flash Media Server family. http://www.adobe.com/products/flashmediaserver/. Retrieved on 28 May 2010.

[7] Comparing Flash iFrame and FBML Facebook applications. http://www.adobe.com/devnet/facebook/articles/iframe_fbml_flash_platform_comparison.html. Retrieved on 28 May 2010.

[8] Flex 3 - Using RemoteObject components. http://livedocs.adobe.com/flex/3/html/help.html?content=data_access_4.html. Retrieved on 29 May 2010.

[9] worker - Apache HTTP Server. http://httpd.apache.org/docs/2.2/mod/worker.html. Retrieved on 29 May 2010.