COP 5611

Project Report:

Implementation and Study of a “Term” based Role Playing Game using Client Server Paradigm.

Author:

Vaithiyanathan Sundaram

V3138147
Abstract

This project aims at using the client server paradigm to build a multiplayer game. A central server controls the action of players (multiple clients) and it updates the states of the players. Each player will have its own Graphical User Interface (GUI) for choosing their attacking patterns. Every attack is term based and initiated by the player. This project also ensures concurrency between player attacks. Eventually the winner and loser are determined, and their appropriate experience points are awarded. This project is implemented using Java and Java Swing, and run under Netbeans 7.0.1 Integrated Development Environment. This project also provides few performance results on the behavior of server at different occasions and suggests a measurement profile for efficient load balancing of servers.

I. Introduction

The client server model is a computing model that acts as distributed application which partitions tasks or workloads between providers of a resource or service called servers, and service requesters called clients [1]. The client/server characteristic describes the relationship of cooperating programs in applications. The server provides service to one or many clients. Similarly, in this project the server acts as the driving force and controller for multiple clients who play the game.

Typically, a server keeps listening to incoming connections by opening a virtual connection called socket. An open listening socket can be visualized as an open pipe into which any number of people can fill water in their bucket provided they sign an agreement with the server. Similarly, the server listens to multiple clients and the clients establish a connection to connecting to that socket. In java the listening socket is called the Server-socket and the connecting socket is called Client-socket. Every socket should be connected to a specific port number. In this project the port number 1234 is used. On the other hand, every client should connect to the server disguising with a unique name called host name. Since, this project is implemented to run in the same machine at different instances the host name is “localhost”.

Once the connection is established, clients and server communicate in a duplex manner. All the messages are passed using a buffer which flows through the socket. The buffer could be a string or a character or an integer or even a set of data. Java uses a class called BufferedReader and PrintWriter to read and write in the socket buffer respectively. In this project, the client selects a particular pattern of attack and sends it to
the server. The server on receiving the computed attack rate computes the amount of health lost by the opponent and refreshes the state in the opponent.

**Scope of this project:**

This project aids the learning of the following:

1. The working of client server paradigm.
2. TCP/IP application on creating a client server.
3. Usage of Java to implement sockets and establish connection.
4. Usage of Java Swing to implement GUI for the client to play the game.
5. Real time Role Playing Game mechanics and multiplayer environment.

**II. Implementation details:**

**Platform:**

This project is implemented in Java using the Netbeans IDE [2, 3]. Netbeans is a very popular IDE for Java. Any IDE reduces the workload of the programmer by easily identifying the errors and recommending suggestions for the programmer. This project uses the following packages:

1. Java.awt.Component
2. Java.lang
3. Java.io
4. Java.net
5. Java.util.Random
6. Javax.swing

This project also uses Java swing which is an excellent GUI creator. Java swing has multiple methods and classes to help the programmer to design his/her own User Interface.
Figure 1 shows the User Interface provided for each player. The UI is user friendly, where the player has to type his/her name in the text box provided at top. The health bar refers to the health available for the player and the Opponent bar refers to the health available to the player’s opponent. The Mana bar refers to the amount of power the player will be able to use. The Mana is controlled by the slider given under the Mana bar. More attack will be caused if more Mana is used during attack. It should be noted that the player will lose the game if all his Mana gets over even before his health depletes. The combo box with Machine Gun as its text gives a supply of different weapons for the player. Depending on the weapon the player’s attack rate changes. The details about weapon is given in the game mechanics section.

The player has to click Fight to initiate the attack on the opponent. When the Fight button is clicked the server will trigger a refresh in the opponent’s screen. When the opponent refreshes its state, it will be able to see his/her remaining life. The opponent has to take his/her decision depending on the remaining health and Mana. In reality, in all multiplayer games the players’ states will be automatically refreshed at regular intervals. To create that real environment, this project devised a mouse movement capture to detect the movement of the opponent and refreshes the state when it detects it.

Once the player loses its health it loses the game and the corresponding opponent wins the game. The player can choose to play again by clicking the Revenge button which will add some experience points to the winners. After reaching a particular experience point, the player will become a level higher than before.
**Backbone:**

The following classes are used to implement the backbone framework of this project in Java:

1. `Serversocket` – used for creating listening socket
2. `Clientsocket` – used for creating client connection socket
3. `PrintWriter` – used for writing in the socket buffer
4. `BufferedReader` – used for reading from the socket buffer
5. `JFrame` – used for extending the attributes of the GUI frame
6. `Thread` – used for handling multiple clients

When the client initiates the attack by clicking the Fight button, it computes the attack rate and sends it to the server through the socket buffer. Once the server receives, it will compute the opponent’s health loss and refreshes the opponent’s GUI to reflect its new health and cuts of the chance for the first player. Now, the opponent initiates its own pattern of attack and the cycle continuous.

**Game mechanics:**

A Role Playing Game (RPG) is a game in which player assumes the roles of characters in a fictional setting [4]. Every player grows along the game. For instance, the players will be awarded experience points which will take them to the higher levels. When the level of the player grows; the attack rate, health and the other attributes of the player advances. In this project, each player is given a class which defines its own attack rate and health benefits. Then the player will get an option to select from the list of the following weapons:

**Table 1: Game mechanics**

<table>
<thead>
<tr>
<th>Weapon name</th>
<th>Attack rate</th>
<th>Critical rate</th>
<th>Mana penalty</th>
<th>Health penalty</th>
</tr>
</thead>
<tbody>
<tr>
<td>Machine Gun</td>
<td>20</td>
<td>80</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Magical Sword</td>
<td>25</td>
<td>60</td>
<td>10%</td>
<td>0</td>
</tr>
<tr>
<td>Rocket Launcher</td>
<td>35</td>
<td>60</td>
<td>5%</td>
<td>10%</td>
</tr>
<tr>
<td>Assault Rifle</td>
<td>40</td>
<td>60</td>
<td>15%</td>
<td>0</td>
</tr>
<tr>
<td>Heal</td>
<td>Add life with a random probability of critical rate and Mana</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Grand power of death</td>
<td>70</td>
<td>Random</td>
<td>20%</td>
<td>25%</td>
</tr>
</tbody>
</table>
The Table 1 shows the different kinds of weapons involved to choose in the game by any player. For instance, player 1 chooses to use the Machine Gun to attack the opponent with high amount of Mana. It is obvious that the opponent would lose health proportional to the Mana used. Since the opponent has full Mana, the opponent should opt between bravery and defense. If the opponent wants to defend he will be choosing Machine Gun (which does not take any penalty) or will be choosing to heal. If the opponent wants to be brave he will choose a weapon with penalty to bring down the player 1’s health faster. This cycle repeats until one of them wins. It should be noted that all the players will be unaware of the weapon the attacker is going to choose. In that way, this game emphasizes on taking chances stochastically.

### III. Some live screen shots:

![Image of game interface showing player names and options for attack and defense]

**Figure 2: When both the player joins and begins to fight**
Figure 3: When player 1 attacks during his/her turn with the Machine gun and 25% of Mana power. The server refreshes the Player 2’s GUI and its remaining health is displayed.

Figure 4: When player 1 chooses to go for a brave attack over the opponent. It should pray to get more critical hit, though.
Figure 5: Player 1 succeeds during the Grand Power of Death attack and the state is reflected in Player 2’s GUI.

Figure 6: Both the players wish to take revenge on each other. Winner (Player 1) receives 200 experience points and the loser (Player 2) receives 0 experience points.
IV. Performance results:

This project also aims at studying the performance of the server during this multiplayer game. This project eventually shows how the server gets overloaded when there are more clients using the server in a large scale. Eventually this project suggests ways to improve the load balancing in the server.

The following steps are followed to measure the performance result:

1. Assuming that the server has the system configuration four times lesser than that of the tested system, the computation resources consumed (CPU usage) is measured using “task manager” [5] for each and every weapon used by a player to attack the opponent.
2. After collecting the measurements, they are applied in a simulated environment to study the load in the server with respect to computation usage or CPU usage.

![Figure 7: Real time performance testing for the server with a PAIR of players](image)

Figure 7 shows the behavior of the server when a pair of players chooses the weapons in random, play the game. It can be seen that the CPU usage is also random and has lots of peaks compared to a pair of players who understands the game. This proves that the server has enough resources to accommodate
more players who are good rather than noob. This is also an inference to decide which server to allocate for a new player to have efficient load balancing.

![CPU usage in percentage over time slots]

**Figure 8:** Simulated performance testing when 1-10 pairs of players join and leave the server randomly.

Figure 8 shows the large scale performance testing. This studies the impact of server load when the number of players increases. In addition to study the randomness in the new players joining and leaving the server, in this test a stochastic arrival and exit of players is simulated. It can be seen that in almost 50% of the time the server becomes overloaded (over 100% CPU usage). This study infers the requirement of studying the client arrival rate before building the server.
Finally, Figure 9 shows the behavior of the server when a pair of lethargic players play. A lethargic player is defined as the player who is either not interested in the game or not focused in the game, thinks a lot and initiates the attack after a long time. This will not only annoy a barbaric opponent who is ready to fight the next instant but also the server which has to sacrifice the arrival of good players. This study infers that the server should decide upon the players with lethargic gameplay.

All these performance results emphasizes on the need to study the following before deploying the servers:

1. Type of gameplay (lethargic or barbaric)
2. Type of player (Random or intelligent)
3. Number of dynamic join and leave.

When these aspects are studied the server can be able to have the optimized load balancing.

V. Challenges faced and summary of the project:

There were few challenges faced during the implementation. The challenges are:

1. Implementation of a real time refreshing of the state of the opponent using a standalone Java Swing GUI.
2. Since the Operating system used in Windows 7, it was difficult to measure the CPU usage from the “task manager”. In Linux it is fairly simple and accurate to measure the CPU usage at regular instances.

Thus this project aims at accomplishing the following:

1. Implement a working client server paradigm
2. Implement multiple client instances and maintain concurrency at server
3. Implement a real time multiplayer gaming prototype
4. Study the requirements of the server before choosing the right load balancing technique before deploying them in large scale.

VI. References: