BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI

MINI-PROJECT REPORT FINAL REPORT

TITLE: Multiplayer Game Warcraft

Course Title- Computer Networks

Course No. - CS F303

BY

GROUP - 9

PIYUSH GOYAL - 2014A7PS029P

RAGHAV BHARTIA - 2014A7PS742P

RAHUL BANERJI - 2014A7PS082P

AVEEPSHITO MITRA - 2014A7PS120P

DOS:- 19th April 2017

Github Link - https://github.com/raghavbhartia/NetworkMiniProject

Problem Statement

The aim of this project is to develop a Pac man-inspired multiplayer real-time strategy action game, complete with GUI, named 'Warcraft'. There are 3 different difficulty levels, a Live Scoreboard for purposes of recording the score and contains a 2-D map, consisting of bugs, swords and points. The player's objective is to gain the maximum number of points he can in the given time limit, while avoiding bugs and sword strikes (by the opponents). He can also try and strike the other player with the sword, causing a reduction in his score. At most five players can simultaneously play the game. A player has to login or sign up in order to play the game, by providing his user id and password.

Scope of Work

The work done covers the design and implementation of the 'Warcraft' game. A conscious attempt has been made to focus on the networking and synchronization aspects, while also trying to ensure that the GUI and other facets of the game are up to standard. Along with the various features of the game, multiple games can be run on the server (by use of the fork () function, creating replicas of the server). Multiple network interfaces and sockets have been used. All four partners have been involved in the programming and testing of the game; in addition, all members were equally involved in making this project possible with work equally divided among the four and each taking turns on module implementations.

Challenges and Limitations

1) In spite of our earlier decision, we were unable to implement a backup server, as we could not find much information about it.
2) Although we had planned to use GTK+ for GUI, we eventually made an interactive console game, as it appeared difficult to use a widget toolkit like GTK+ and integrate it with our project. The main problem was that the GUI was heavy weight and was leading to a lot of time lag. Also use of external libraries was discouraged.
3) We have not been able to test the scalability of the game to the extent we would have wished to, although a certain amount of scalability is ensured as the number of PCs at our disposal was limited.
4) General challenges in programming and debugging were faced, but we hope that we have managed to circumvent such issues. For instance, debugging the problem of the server not receiving the client address, seemingly at random, was rather time-consuming.
5) Maintaining a balance between ease of programming, game complexity and time constraints was rather tough. We had to make do without a few features we had earlier wished to include (for instance, player tables and relegation, more complex GUI and rules), primarily based on the fact that the focus was to be on networking.

Basic Game explained with rules

1. The game of Warcraft has a few elements – A hero(player), treasure chests, monster bugs, and the monster killing sword 'The goliath'
2. Our hero while slaying monsters, is free to steal treasure chests or steal form other heroes.
3. The hero needs 'The Goliath' which is spread out all over the 'Warzone' to kill monsters or steal treasure from other heroes.
4. Stealing treasure chests gets 1 point while killing monsters yields 2 and stealing from other heroes yields 3 points.
5. Basic W,A,S,D for movement is all that's needed!!!
6. Stay away from bugs if you don't have the sword else you get hurt and loose points to heal yourself.
7. 'The Goliath' is a very heavy sword so the hero can carry only one sword. He can defend himself from other heroes if he has his sword.
8. This sword is only 1 time usable and again has to be picked in the game once used.
9. Get the maximum points before the timer runs out and become the legendary Hero!!!

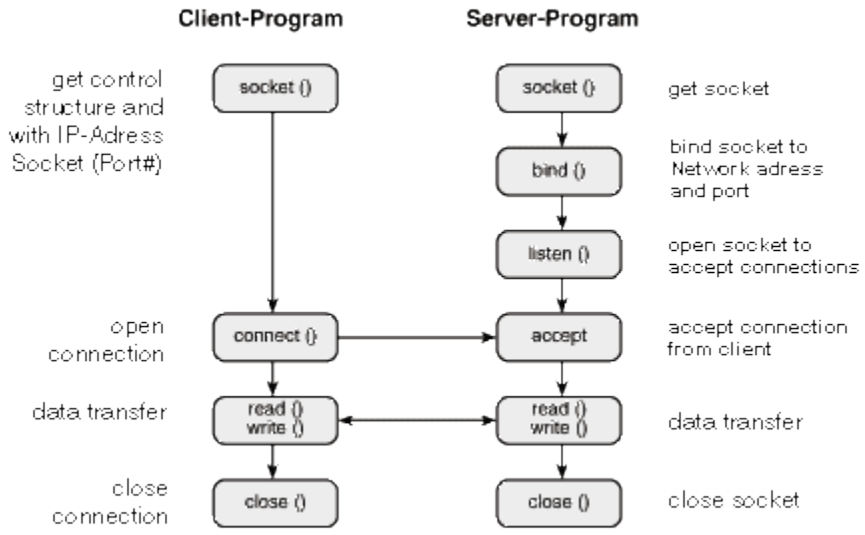
p.s : We made the game to be a very thrilling and enjoyable experience hence designed a completely new game setup to intrigue the new players.

Design and Implementation Details

Basic Working

TCP sockets have been used for all network interfaces, to ensure the management of traffic congestion or of data arriving out of order does not arise, using IP addresses for exchange of messages. The server and client all have a local map. Once a player makes a move the client sends the move to the server. The server checks the validity of the move and whether it is possible or not. The server will then periodically update the map and then send this updated map to the client. There is a TCP socket setup at each thread responsible for the data transfer. One thread is responsible to update the map of clients at regular intervals simultaneously while the other thread is tasked with receiving user input from client and adding the move or updating the server table. Then later the other thread takes care of the update to client.

The Client has two threads running independently of each other. One is responsible for printing the map that it receives from the server. This is done by maintaining a local map which is updated timely by the server. The thread then proceeds to print this map at fixed intervals in range of a few milliseconds. The second tread is responsible for getting the user input from the terminal and passing that input to the server. This has been implemented such that 'Enter' key need not be pressed and data is automatically sent on at the press of a button.



To ensure data consistency between the threads, Mutex-locks (a program object that allows multiple program threads to share the same resource, such as file access, but not simultaneously, to ensure exclusive access) have been used at various ends, i.e. If Thread 1 is updating the map then Thread 2 won't send the map until thread 1 is done and releases the lock.

A very detailed description of the above was done in writing and via diagrams in component EC-2 hence we are skipping it to prevent repetitions. Link is there in GitHub.

Features and Implementation Details

There are many network and optimization features applied in this game.

1. We made parallel connections as opposed to iterative ones as this ensures optimal and equal server utilization for multiple players. The iterative solution was painstakingly slow and was having a lag time of 600ms - 700ms. On making parallel connections this was reduced significantly (tens of milliseconds on same subnet).
2. Scaling of Games across multiple instances was enabled. This was done by making a copy of the server instance at the moment the number of players were enough to make a new board and the game was started on that new instance while the old instance waits for more players to join the game and this goes on. The TCP connection is maintained across all the instances by using an array of sockets to individually control every instance.
3. Reconnection of user back to the game has been implemented on disconnection. This has been implemented on a different way. Upon discussion with team members it was decided to reconnect the user to a new game and is started as it makes no point to join back to the exact old game. The reason being this game is very fast paced so disconnection and reconnection will take more time and the player will lose his or her chance in that game making it very unfair. So for better implementation this feature was added.

   This can also be seen as a pseudo pause feature in the sense the player can exit the game at any time and reconnect to a new game at the very next moment.
4. We used a random generator using some basic probability heuristics to randomly generate bots and bugs in the game and bug distance is being updated based on the distance to nearest player at level 3 difficulty.
5. Basic DP algorithms were implemented like local updating in Map (which is a 2d array) and updating only the needed fields to improve performance.
6. Whiteboard feature was implemented. This means that the very same screen will be seen by all the players at the very same time. This was done by setting some bits to help synchronize all the instances. A very basic and intuitive greedy approach was used for the implementation
7. A login system is enabled to check access of users and allow registered users to establish connection with the server.
8. Multiple Difficulty levels has been implemented based on the bug movement. Level 1 has static bugs, level 2 has moving bugs who move at random location, and in level 3 the bugs run to the nearest player.

NO EXTERNAL LIBRARIES WERE USED IN THIS PROJECT

Functions in brief

In the server code, the program works with a new socket being accepted on the 'accept' function call, and stored in the 'array_of_sockets'. These correspond to the different players, who are connected via the server. The 'input_thread' function checks for the correctness of the sent move, and, if correct, updates the position of the game board (and also updates in the 2-D position array player_positions). The new_map function, on the other hand, generates a new map, while the send_to_all function, as the name indicates, sends the map to the client. Id and password are checked (option = 2) in case of login, but not for signing up (option = 1), using the buffers buffer2 and buffer3. The two pthreads with mutex locks are used to maintain proper synchronization.

On the client side, the player_input function takes the input for sending to the server, while the print_board function is used to print the new board received from the server, and the players' points. This function has an input as the character array buffer, sent from the server (send_to_all function). The 'endgame' function concludes the game and declares the results.

Learnings and Conclusions

This project helped us gain in-depth knowledge of and experience in network programming. The project is able to fulfil its main requirement, that of designing a multiplayer game that can be played across multiple terminals over a connected network. The dynamic functioning of the client and server modules help achieve the objective of the project.

We learnt a great deal about how exactly sockets function and work in real life scenarios including multiple connections. Also integration of Network programming with basic OS calls and semaphores was a mammoth task and helped us gain info of integration of topics learnt in Computer Science across many subjects. Got a good feel about latencies in network how they can be reduced to improve performance with the right approach.

Future Extension

This project could be made to be far better if an efficient and interactive GUI could be implemented. Also the board can be made to add more accurate heuristics for bot and bug generation in the map. With an added auto installation script this could be turned into a complete game package to be enjoyed by all hopefully in the future!!! Also extra features may include an added backup server thus helping increase reliability and data portability.

A final word: In case of any problems with the game a readme file is available at GitHub and any of the team members may be contacted