**BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI**

**MINI-PROJECT REPORT EC-2**

**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

**Problem Statement**

We are making a game "Warcraft". This will be a multiplayer real-time strategy action game with an implemented GUI.

The game is a 3 level difficulty game with a Live Scoreboard for comparing the scores.

This game is inspired from Pac-man and in Warcraft, there is a 2-D map containing swords, bugs, barriers, points, in which the aim is to collect maximum number of points in the given time limit while avoiding bugs and sword strikes from opponents and attacking opponents with swords. There will be a dynamic rating system with different game options based on varying difficulty.

**Status of Progress**

|  |  |
| --- | --- |
| Week 1 | Basic game has been developed so as to run between single server and a set of clients and it has been tested for 2 players.  Some basic game rules are implemented. |
| Week 2 | The GUI interface has been developed in lieu with the complete game and has been implemented at a basic level.  All game rules finalised and game extended for 5 players. |
| Week 3 | The game development for running multiple games on the same server between multiple clients is under progress.  Different game features like swords and bugs implemented. |
| Week 4 | Multithreading and thread synchronization implemented to handle different clients simultaneously. |

**Problems Faced**

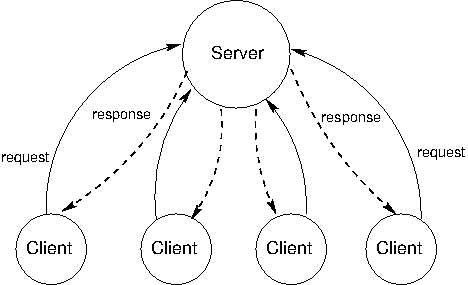
1. The GUI initially thought was to be implemented by the Skia or GTK library but for now we made it into an interactive console game with pseudo-animations as giving too much effort in GUI is not the prime objective of the project.
2. Making a backup server is proving extremely difficult for us as we have no idea on where to start and not getting proper references of the same on the internet
3. Certain rules of the game have been modified to make the game more enjoyable and given the time constraint we will be prioritizing the game and its gameplay instead of data portability between clients i.e. auto installation of game on clients etc.
4. Although we are creating a multiplayer game accurate testing with multiple players over many different sessions is not feasible hence this may cause some inconsistency in scaling.

**Design Rules**

This game(so far) has been implemented using a combination of multi-threading, OS system Calls, multiple Network interfaces and sockets and the game engine.

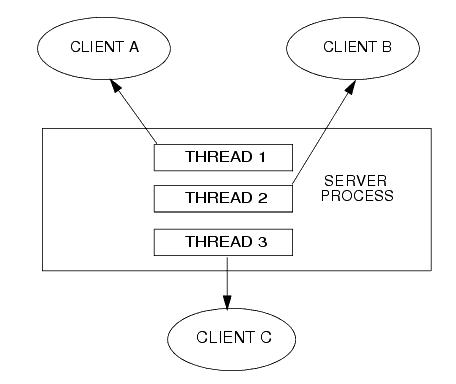
All network interfaces are being handled by TCP protocol as it’s reliable data transfer ensures that the game doesn’t get stuck(traffic congestion management) or has data arriving out of order which is important to this game. TCP sockets are being used for the same.

To explain the exact working and design of the game a 2 player model is considered on a single board (multiple boards also possible) which may be scaled up to 5 players(beyond that the board gets too crowded!)



Here a basic working is shown. 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 if it’s possible or not. The server will then periodically update the map and then send this updated map to the client.

**Server**



**Thread 3 and Thread C**

**Thread 2 and Thread B**

**Thread 1 and Thread A**

As shown in the above diagram the server runs 2 threads for every client.

There is a TCP socket setup at each thread responsible for the data transfer. One thread (Thread A/B/C) 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.

To ensure data consistency between the threads Mutex-locks have been used at various ends, e.g. If Thread1 is updating the map then Thread A won’t send the map until thread 1 is done and releases the lock.

For implementing multiple board games we will use OS command called the fork() command which will make multiple instances of the server and launch a board game in each instance of its own thus covering multiple clients. Right now this part of the game is under development and not tested yet.

**Client**



**Server**

**Client**

**Thread 1**

**Thread A**

**Input Thread**

**Print Thread**

The Client has 2 threads running both independent of each other.

One is responsible for printing the map that it receives from 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 no enter has to be pressed and data is automatically sent on mere press of a button.

Mutex locks have been used to ensure data consistency and all the connection are TCP based. The display output will use a combination of asci art and colours to accurately display the state of the board.

The communication between Server and client is carried out by using a string/buffer of fixed length conveying the game state in it.

---------------------------------END------------------------------------