CS F303 : COMPUTER NETWORKS

MINI PROJECT EC-3

on

“Simulation of multilevel card game (Hearts) over TCP network”

FINAL REPORT

By

Group 6

Pranjal Gupta	2013B4A7470P
Rishabh Jain	2013B3A7473P
Neel Kasat	2013B3A7670P
Akshit Bhatia	2013B3A7722P

https://github.com/g31pranjal/online-hearts

April 19, 2017

1. Problem Statement

The software will be the Conceptualization, Specification, Design and Implementation of the famous card game called Hearts. This game can be played among a group of 4 users(clients) and will be co-ordinated by a server over a TCP network. The software also extends the basic version of the game with increasing difficulty levels which introduces subtle changes to the game rules with its progress. The game can be played by 4 players which will connect to a host server. The host server synchronizes and alternates between the user(client) activities. Each client should communicate to the server with the current state of its player. The client will prompt its user to take an action (e.g. throw a card) and will communicate the user action to the server. The server should check if the move from a user (through client) is feasible and legit, and consequently continues the same with other clients. The server should also keep scores and communicate the actions of other client to a client. The client shall have a basic user interface which will display scores and actions of other users, as required.

1. Scope of work defined:

S.No.	DESCRIPTION	MODULE	OWNERSHIP
1.	BLUEPRINT OF ALL COMMUNICATION BETWEEN SERVER AND CLIENT	1	**Akshit, Neel**
2.	CLIENT-SERVER ARCHITECTURE : BASIC BUILD UP (WITH SERVER HANDLING MULTIPLE CLIENTS)	2	**Akshit, Rishabh**
3.	SPECIFIC FUNCTIONS FOR CLIENT AND SERVER FOR IMPLEMENTATION OF GAME LOGIC	3	**Pranjal, Neel**
4.	ACTION SYNCHRONIZATION BETWEEN CLIENTS AND SERVER AND SHARED MEMORY	4	**Pranjal**
END SEMESTER DEMO AND SUBMISSION			

3. Perceived Challenges (in Design and Implementation)

- **Handling Multi-client concurrent connections:** In the game, we had to connect 4 clients to one server and connect them to play one game. So, we created a child process as soon as we get a group of 4 clients connected to the server. So now, for every group of 4 clients, a new game room (child process spawned off the main server process) will get initialized. The parent server process continues to run the background.
- **Problem in consecutive send/receive between same server-client pair:** We were having data inconsistency issues on sending two consecutive messages or receiving two or more consecutive messages to a specific client. This was specifically because the TCP protocol does not sends/receives data based on application discretion. Any data that comes to the TCP socket, is stored in specific buffers and transferred accordingly. In order to solve it, we added a sleep of 1 time unit between any two consecutive send/receive commands between a pair of client-server, this delay ensure that data in the buffer is emptied before consequent data is received in it.
- **Action synchronization between client and server:** To maintain the smooth flow of the action sequence to be taken between the server client pair, we were having a problem synchronizing the flow of data, signals and commands. So, instead of sending the data to the client to communicate the state of the game at each specific interval, we resorted to keeping a copy of required data with the client, and sending signals to the client. And, as per the signals sent, the client copy of the data is manipulated.
- **Information synchronization using temporary buffers:** In Level 2, we need to transfer two cards to the other player. We will store the cards in a temporary buffer at the server side. A random number, say n, between 1-4 is then generated. Now, every player transfers 2 cards to the nth player in the sequence beginning from that player.

4. Limitations:

- We need exactly 4 players to begin a game room. If there are 3 players, then they will have to wait for the 4th player. And to begin a new game room after first set of 4 players have joined, we will again need players in multiples of 4.
- Textual User Interface: Although we have put up unicode symbols of card deck, the textual user interface still provides a cluttered view of the game. A graphical user interface could present the game in an interesting manner.
- We have currently not employed any mechanism for ensuring the “presence” of a client in the game i.e. if the player leaves the game abruptly, the game room continues to execute and gets hanged at the moment of no response by the left client. This situation needs to be handled to ensure the proper functioning of the client.

5. Architectural Design

Game Room Formation:

Every client will have to connect to the particular host IP and port in which the server is hosted. Server waits for 4 clients to get connected, in the event of which, a child process is spawned and it forms a "game room". The game logic and further server-client interactions and synchronization is handled in the game room.

If more clients get connected, the parent process of the server again waits for clients in multiples of 4 and spawns child processes for that group to run the game room.

The game room further facilitates the playing of the main Hearts game. Every player will get a server generated player ID and a randomly allotted player name. Server makes use of the allotted player ID to cater to communication with all the clients

Server's sequence of actions :

The server's communication in the game room has been implemented as a sequence of signals to each of the client based on the state of play. With each signal, data is either received or sent based on the pre-defined action requirement. Each signal corresponds to a specific action in the game, which depends upon the action of other clients in the game play. The table of signals, along with their corresponding action and data that needs to be sent or received, are as listed :

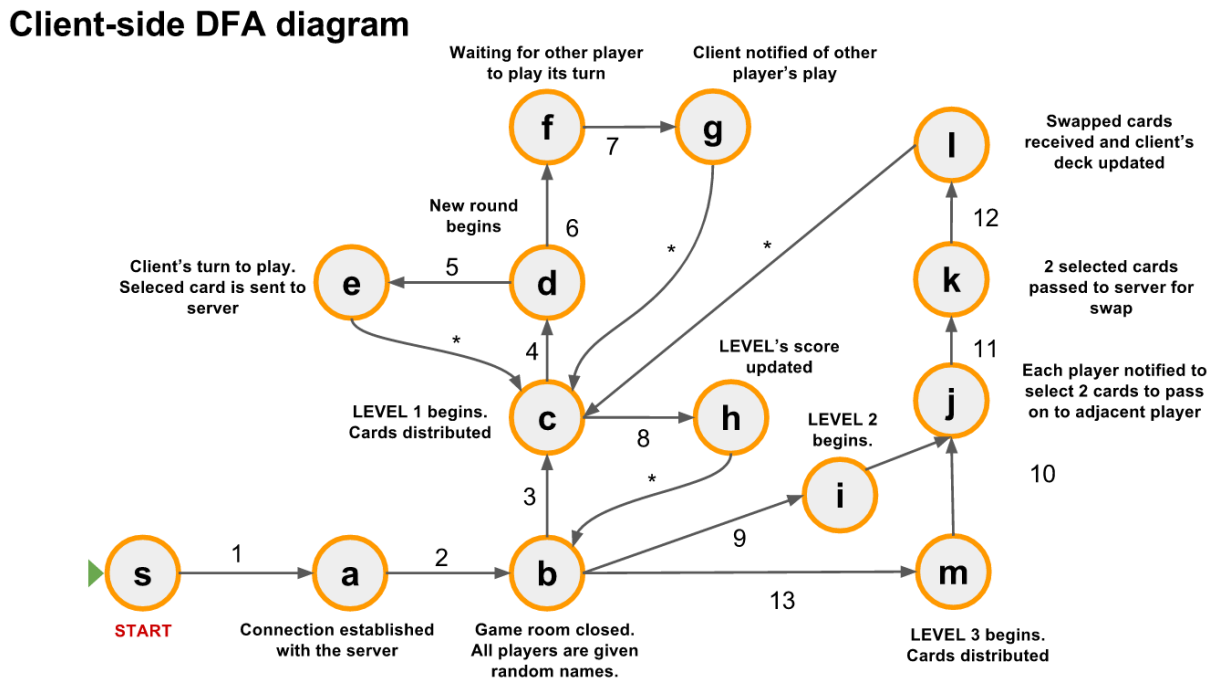
SIGNAL	ACTION	DATA (SENT/RCVD)
1	Connection established successfully over TCP	-
2	Game room closed. All players are allotted random names.	**SENT :** Sequence of player names in the room and id of the client.
3	Level 1 begins. Cards are distributed to all the players in the room	**SENT :** 13 cards for each player
4	Beginning a round of play	**SENT :** Round number
5	Prompts the client to play the card.	**RCVD :** The index number of the card played by a particular client
6	Wait while the other player plays his turn	**SENT :** id of the player currently playing his turn
7	Prompts that the current player has played his turn	**SENT :** Index of the card played by the other player
8	Updates score after each round	**SENT :** Scores of all the players in the

		game
9	Level 2 begins. Cards are distributed	**SENT :** 13 cards for each player
10	Prompts the player to select 2 cards that the client wants to swap with other player	-
11	Requests to send the selected cards	**RCVD :** Comma-separated index of 2 cards to be swapped.
12	Sends the swapped cards to the user.	**SENT :** Comma-separated index of 2 cards.
13	Level 3 begins. Cards are distributed	**SENT :** 13 cards for each player
14	Game ends.	-

Client Model :

The client's action sequence has been modelled in the form of Deterministic Finite Automata (DFA), wherein each state represents a specific progress in the gameplay. The state transitions are only possible in the event of a "valid" signal from the client. Any other "invalid" signal is treated as an unwanted deviation from the normal play, and is reported as an error. Each state refers to a specific situation in the game wherein a user is either 1). notified about an update in the game (possibly as the result of action of the other players) or 2). prompted to perform an action.

A DFA diagram with state descriptions are as depicted below :



6. Structural Design and Implementation Details

The game will require proper synchronisation between the turns of players, handling connection drops. Highlights of the game are as follows :

a) Running of Server and Client files

i) First the server file needs to be hosted. The server binds to its assigned network IP using a free port (Any number greater than 1024).

ii) The clients now need to connect to the same IP and port, which needs to be communicated to them beforehand.

b) 4 users have to play at a time on different machines.

i) 4 players will have to connect to the server. If more than 4 players request for a connection, a new game room will get created for other players, in multiples of 4. All the users should be connected to a common network.

c) Each player is distributed cards randomly.

i) The cards have been numbered from 1 to 52, where first set of 13 cards being of Clubs, followed by Diamond, Hearts, and Spade.

ii) The cards are printed along with the unicode symbols of the deck.

d) Score of each user, each player move, handling game levels

i) Score of each user is maintained on the server. The card set is also stored on the server as well as on the respective client. This has been done to ensure fair gameplay on the side of server.

ii) As one level ends, the user gets upgraded to play the next level. At the end of the three levels, the overall winner is declared.

e) Each user has to wait for the user to complete his turn

i) As per the signalling, each player has to wait until his chance comes. He will be updated as per whose chance is going on, and what card he/she has played. This signaling is well=handled using the DFA (explained in Architectural Design).

f) Levels and Rounds

i) There are 3 levels, with increasing level of difficulty, with each level having 13 rounds. The rounds progress sequentially, with each player waiting for 3 other players to play their turn and playing its own turn in a pre-defined order. At the end of each round, the server update points scored by all the players in a particular round, also total cumulative score till that round.

ii) Scores are maintained at server for each round and levels consequently. The scores are sent to clients at the end of each round.

iii) The server also ensures fair play in game. The client checks if the card played by the player is a valid move based of the previous cards played by other players. This is cross checked at server. The server also notes the sequence of cards played and the collection of cards at hand for each player.

iv) Description of Levels:

1) **Level I**: Vanilla version of Hearts. No special rules are there in this level. Scoring rules are mentioned below.
2) **Level II**: In this level, each player has to pass two of its cards to the adjacent player in the beginning of the round. Hence, the player can pass its "risky" cards to its opponent. Post this change in hands, the normal gameplay continues.
3) **Level III:** In addition to the rules added in Level II, three new rules are added here:
   (a) The first card of a round can't be of Hearts deck, until some points have been scored by any player.
   (b) The Queen of Club, which earlier had no points, now has (-10) points. This reduces the player points and hence, must be sought for.
   (c) If a player gets both, The Queen of Hearts and The Queen of Spades, then the total points are nullified, i.e., -14 points are awarded.

g) Error Handling

i) The reliability of a software is purely dependant upon its ability to handle unforeseen situations efficiently.

ii) The server responds to connection drops with clients and consequently signals other players about the drop. In this case, the client closes its connection to the server. This response to situation is a major limitation in our implementation.

iii) Provisions are also made to cross out invalid signals while communicating, so as to ensure proper sequencing of events in the game play. We have also tried to minimize the errors that may arise that to stray data in the socket buffer.

Gameplay Rules:

In this game, the goal is to set the *Lowest Score Possible* by avoiding cards that are worth points. Each player devices the strategy to shell out their card, given that they avoid receiving cards that carry points.

Points Table:

Card	Points
Hearts	1
Queen of Spades	13
Rest	0
Queen of Clubs (For Level 3 only)	-10

1. Each player is given 13 cards.
2. The player having The 2 of Clubs plays first in every level.

3. Once a card is led, all remaining players must play a card of the same suit, if they are able to. This case is handled as error in the client and user is prompted to re-enter the card.
4. If you do not have any cards of the lead suit, you can play any other card. Usually this is a good time to start dumping any card of Hearts you may have.
5. The player with the highest ranking card of the lead suit, collects all 4 cards. Also known as *Taking the trick.*
6. Whoever took the last trick, leads the next one and the round follows circularly.

7. Major Learnings

- Working of TCP and Socket Programming: Defining the socket by IP and Port number. We preferred using TCP over UDP because of the reliability offered by the former.
- Process synchronization between client and server: If we were sending consecutive messages to a same client, then there was a synchronization problem. So, putting a sleep of 1 time unit escaped the problem.
- Use of colors and unicode symbols in C. As we were limited to using Textual user Interface, we tried finding ways on how we can make the best of the this. So, we got to know the use of colors and unicode symbols in C/C++ programming.
- Implementation of DFA instead of else-if ladder: When we started with the project, the client code had long chains of if-else conditions. It became difficult for us to comprehend our code. So, we changed it into a DFA structure, using switch-case statements.

8. Conclusions

In the course of the project, we came across a lot of new things. In order to generate multiple game rooms, we used forking where a new process is spawned out of the parent process each time a group of 4 players connect to the server. This has helped us in modulating our code structure and also separate the server functionalities from the game logic. The knowledge of “Sockets” was fundamental in the project. This has greatly helped us in understanding the design principles of TCP in terms of behavioral and functional aspects. In our project, the game room basically holds the sockets to each of the 4 clients. Also, sending two messages to the same client-server consecutively can lead to synchronization issues, so we corrected that using sleep in between the send commands on the server. We have implemented our client model as a DFA instead of a if-else ladder, which has greatly reduced the implementation difficulties. The DFA is driven by the signals from the server which performs specific operations on the client side.

An important aspect of networking is error and ambiguity handling so as to ensure a clean and correct flow of execution. We have employed a basic error management that handles certain situations in which error arises. However, our error handling can be further improved if we can employ logics for error recovery as well. The basic error handling mechanism takes care of connection breaks, invalid transmissions over TCP and network issues.

9. Possible Future Extension

- Flexibility in number of players per game: As we are limited to a 4-player game room, we can further extend it to support a 3 or 5 player game room.
- A possible extension to the project can be to support robust error handling. A lot of unpredictable situations can arise as the result of which unforeseen consequences can be encountered in both the server and client processes. Handling of errors can only be full-proof iteratively during the development of the game.
- The protocol of communication between the server and client, that is currently based on the sequence of signals and data transfers, can be further improved to reduce communication requirements over the network. Hence, improving the existing protocol can definitely help to reduce network usage and simplify the gameplay.
- Creating a computerized player. If only a single person wants to play the game, then other computer players can be generated to play with the player. Further adding artificial intelligence capabilities to the “computerized player”

APPENDIX 1 : REFERENCES

[1] http://www.dreamincode.net/forums/topic/313244-multiple-clients-single-server-fork-sockets/

[2] Description of traditional Hearts game : https://en.wikipedia.or g/wiki/Hearts

[3] cgui.sourcefor ge.net/

[4] Network libraries in C : https://www.gnu.org/software/libc/manual/ html_node/Sockets.html