Course No. - CS F303
Course Title - Computer Networks
Group No. - G34
Title of Project - Canadian Fish Card game (Server - Client based)

1. **Sugam Garg - 2014A7PS092P**
2. **Vishal Arya - 2014A7PS073P**
3. **Pranav Sood - 2014A7PS155P**
4. **Divish Dayal - 2014A7PS132P**

Evaluation component - 3
Github Link - https://github.com/sugam11/Litt_network_implement
Date of Submission - 25 April, 2017

Problem Statement:-

To design and implement a server and client procedure to successfully play Canadian Fish Card game.

The server should be able to connect 8 clients at the same time and divide them into two teams of four. After that server should be able to facilitate game playing between the clients with Whiteboard and Live Score Display and Update feature. The server should eliminate the worst performing client from each team after every Three rounds. The server should decide the winning team at the end of the game. The server should also store a history of last 10 moves. At the Client side the software should be able to facilitate communications with the server. The software should interact with the user and receive due inputs from him/her. Also, it should be capable of receiving messages from server and display it to the user. The server should also tell the players whether a given card is present with the user or not.

Scope of Work:

1. Game play implementation on server and client side : Sugam & Vishal
2. Move history implementation, misc. : Pranav
3. Score tally : Divish
4. User input restraints,warnings,etc. : Pranav
5. Game level management : Divish

Structural Design Details(Final):

1. Maximum No. of clients expected in first round : 6 or 8.
2. 3 separate structures for teams, cards and players.
3. The structure for the team has the following fields:
   Number of players, team score, and a player array.
   ```
   struct team
   {
           int num_of_players;
           int team_score;
           struct player arr[];
   }
   ```
4. The structure for the player has the following fields:
   Number of cards, player score, char team, username and a cards array.
   ```
   struct player
   {
           int num_of_cards;
           int player_score;
           struct cards array[];
           char team;
           char username[];
   }
   ```
5. The structure for the cards has the following fields:
   Value of card and the suite of card.
   ```
   enum value
   {
           ace = 1, two, three, four, five, six, seven, eight, nine, ten, jack, queen, king
   }
   enum suite
   {
           clubs = 1, spades, diamonds, hearts
   }

   struct cards
   {
   ```

```
        value cardVal;
        suite cardSuite;
}
```

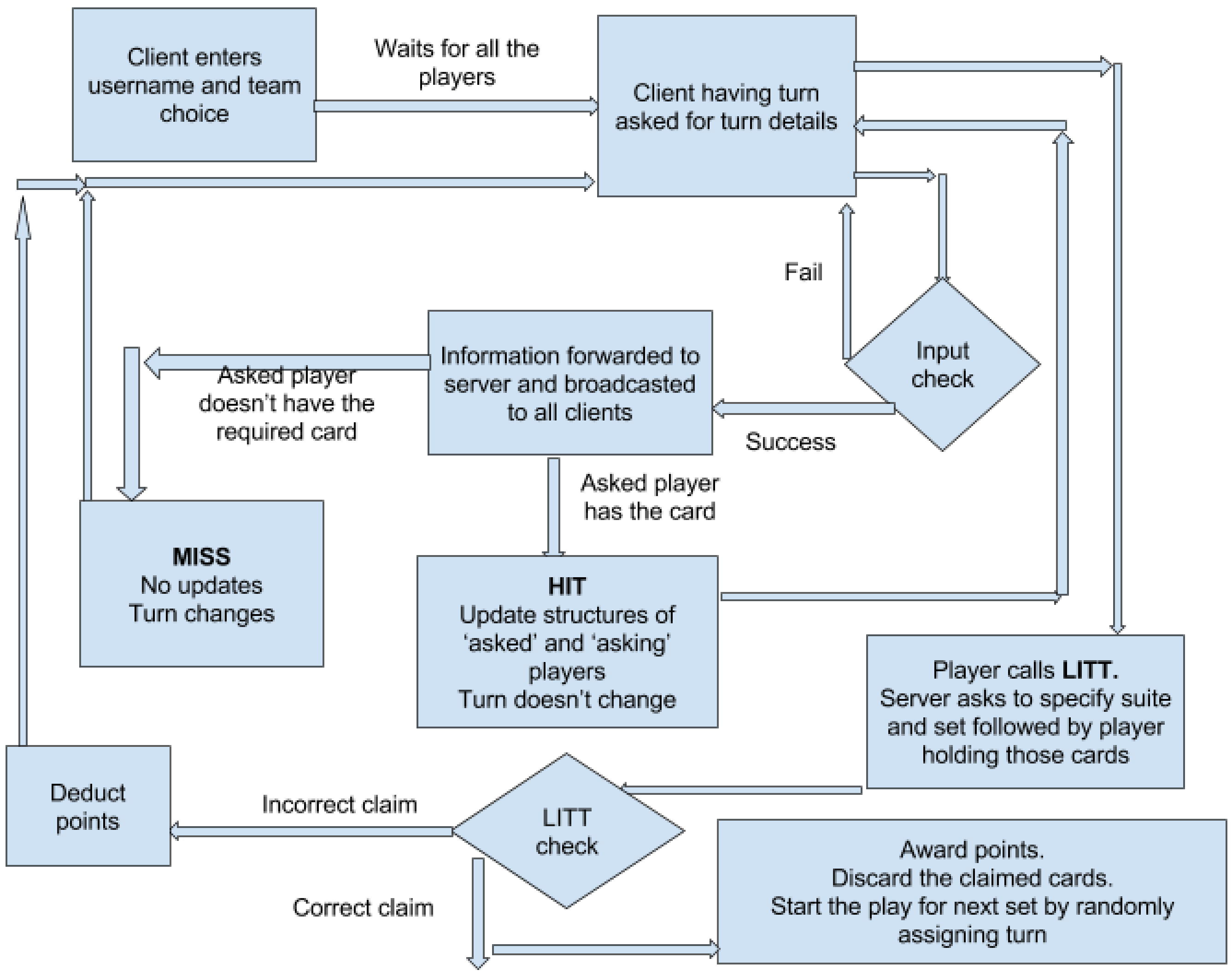
6. A queue of history of turns, with a turn having the following fields:
   Player asking, player asked, card asked.

```
struct turn
{
        player asking;
        player asked;
        card asked;
}
```

Implementation Details(Final):

1. Every client after connecting to the server will be asked to provide his/her player name along with the choice for its team.
2. After which the Player waits till all the remaining players are connected.
3. After all the 8 players have connected, the server will randomly assign the turn to one of the individuals.
4. The server will require the card asked and player asked to store the details of a turn, which will be provided by the player currently having the turn. Then if input check gives success, input forwarded to server.
5. Server broadcasts the turn details to every other player. After which, the server checks the cards of the "asked" player and if found it is considered a HIT and that card is removed from "asked" player by the server and added to "asking" player's cards, respective clients update the other concerned fields.
6. In case, the card is not found its a MISS and no updates are necessary.
7. Also, if its a HIT, the turn continues for the same player else it goes to "asked" player.
8. This process continues, till a player claims "LITT". If a player claims LITT, the server asks him/her to choose the suite and set 2-7 or set 9-ACE. Depending on which, the server asks the player to provide a teammate name having the card for each card.
9. If all the cards are claimed correctly, then the team is awarded that set else the LITT is denied.
10. After every three set claims , two players with lowest scores are kicked from the game and their holding cards redistributed randomly and the level is up.
11. Points are awarded as follows:
    1. For every HIT : +1 to the player
    2. For every MISS: 0 to the player
    3. For every Set scored: +5 added to each player of the team
    4. For every correct LITT claim:+3 to the player
    5. For every wrong LITT claim: -3 to the player

Architectural Design (Final):


Client enters username and team choice
Waits for all the players
Client having turn asked for turn details
Fail
Input check
Success
Information forwarded to server and broadcasted to all clients
Asked player doesn't have the required card
Asked player has the card
MISS No updates Turn changes
HIT Update structures of 'asked' and 'asking' players Turn doesn't change
Player calls LITT. Server asks to specify suite and set followed by player holding those cards
Deduct points
Incorrect claim
LITT check
Correct claim
Award points. Discard the claimed cards. Start the play for next set by randomly assigning turn

Design/ Implementation challenges:

The game requires 8 players to begin. So every time we needed to test our code , we had to run 8 clients which took a lot of time.

Limitations of the project:

1. The game requires at first level 8 players , at second level 6 players and at third level 4 players to continue. So, if any player leaves the game in between , the game has to be shut down.
2. UI provided is simple print statements on console which maybe tough to comprehend at times given the input constraints.
3. Connection needs to be stable throughout the gameplay otherwise the game may lead to unexpected results.

4. The random function of C is used to generate random values , which may not lead to absolute random values at times.

Major Learning from the project:

1. Establishing a multi client single process TCP server.
2. Sending broadcast messages in a TCP connection.
3. Introduction to using TCP sockets for communicating b/w server & client.

Conclusion (Summary of submitted work):

The game implemented is based on client - server architecture. To play the game , 8 clients are needed who each have unique username and are divided into two teams , each of size 4 as per user's choice. After every connection, only connection fd of the socket is stored and IP address of the client is displayed. For every send/receive the same connection fd of every user(player) is used. The game starts by randomly assigning deck of cards to every player which is being displayed on every client(player)'s console. The starting turn is randomly assigned to a player after which the game starts giving a player two options :

- Demand a Card
- Claim Litt

After every three set claims , two players with lowest scores are kicked from the game and their holding cards redistributed randomly, and the game goes a level up. The team which has the maximum sets at the end of the Game wins!

Future extension :

1. The server can provide an option to create game rooms so that more than 8 players can play at a time.
2. Also, when the players are waiting for 8 player constraint to complete, it can allow user to chat.
3. The GUI can be updated in order to display cards and small animations.