**1 Introduction**

This section introduces the motivation behind game application and gives a brief overview of its major functionalities.

* 1. **Motivation**

The motivation behind the project is to:

Work with multiple processes/threads

Become familiar with Inter-Process Communication

Gain experience with an Integrated Development Environment (IDE)

Gain experience with socket programming

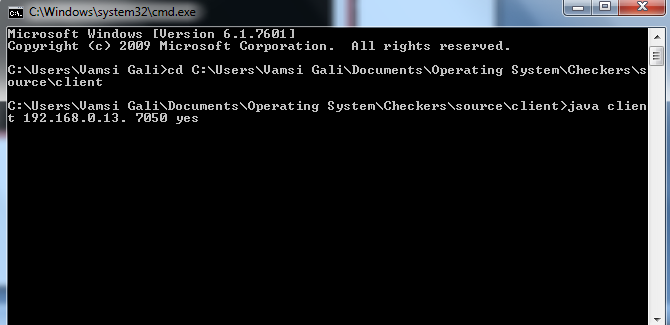
Gain experience with problem solving, solution design and implementation

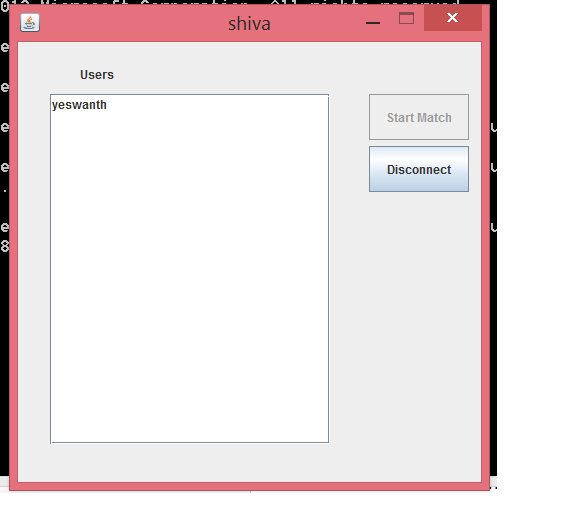
* 1. **Overview**

Developed a Multiplayer checkers game using socket programming in java. There will a single server socket and as many clients as possible. Once the server is created in the application, clients can connect to the server using the IP Address and the port number.

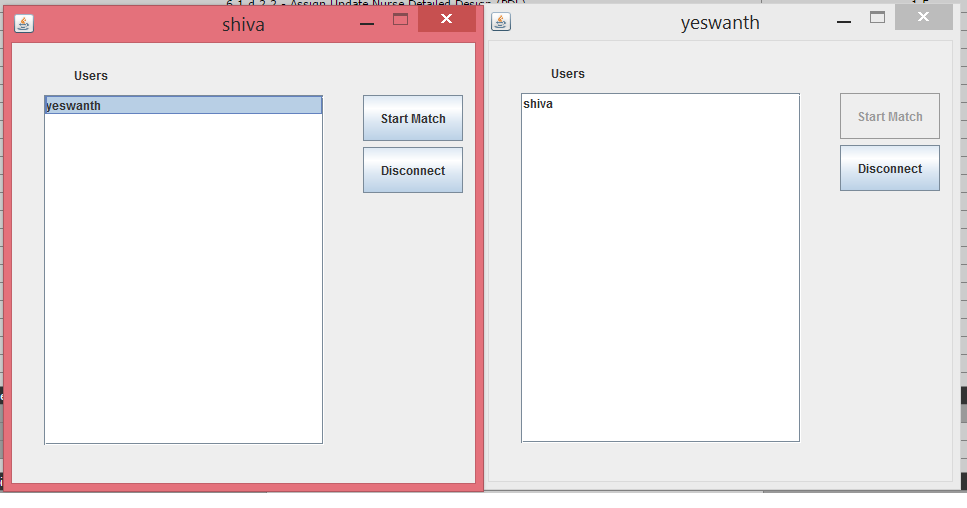
In the application, first the server socket is created. Once the server socket is created, Server socket will be ready to accept the client requests. For each client request, a window is opened where he can see the list of users who are connected to the same server.

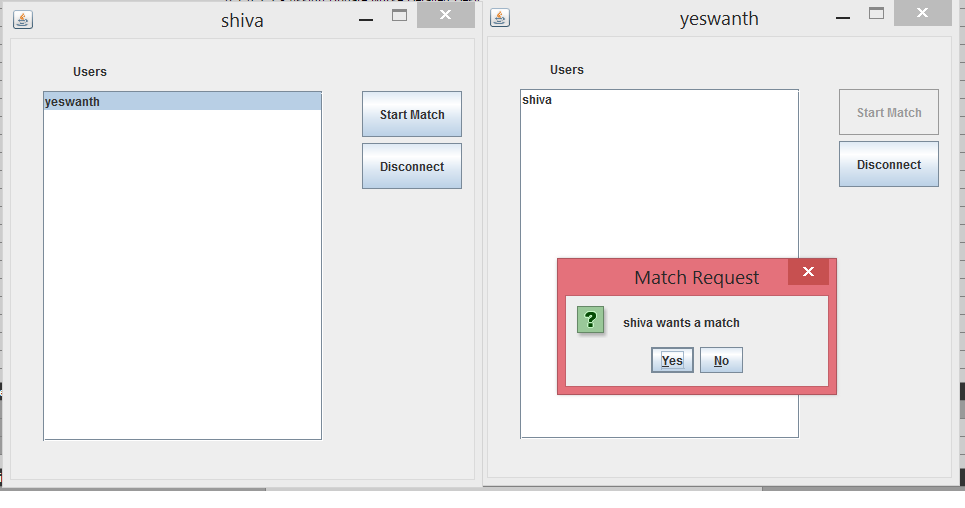
The client connects to the server by providing the IP address, port number and the username from the command line

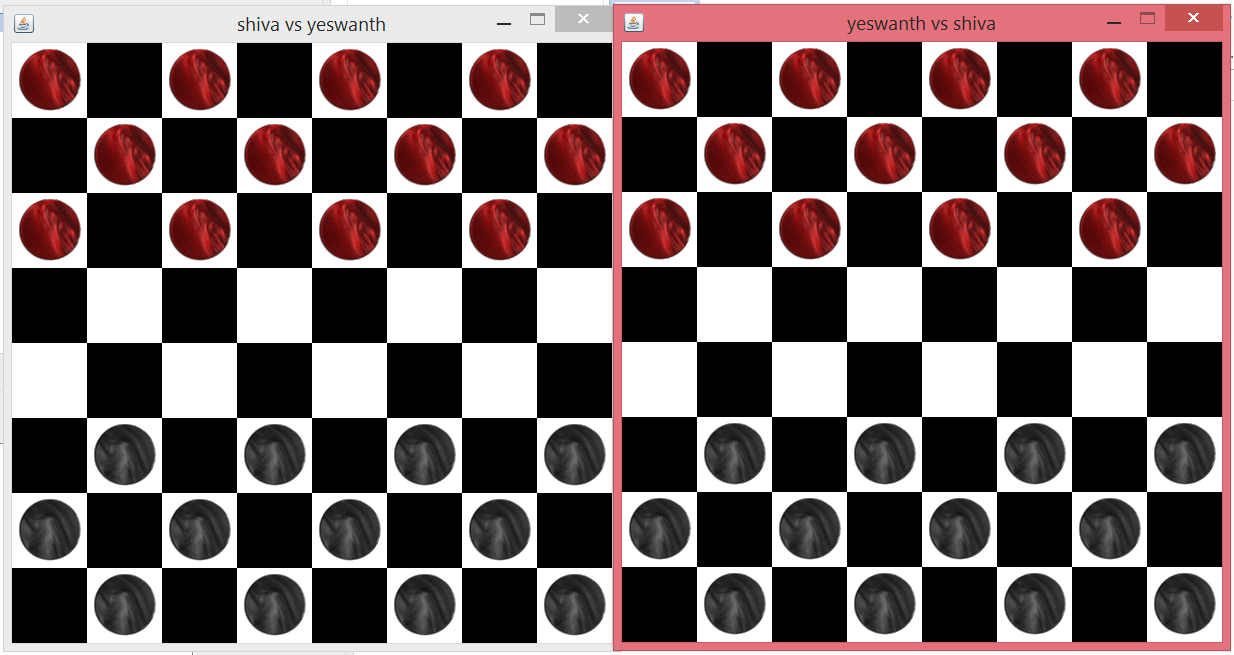




The client can select any of the client from the list and send client a game request. In this scenario, the client request to connect to another client is sent to server and the server issues the request to the particular client. If the requested client accepts the request, the server informs the client that game request is accepted and the server starts the game for the two users.

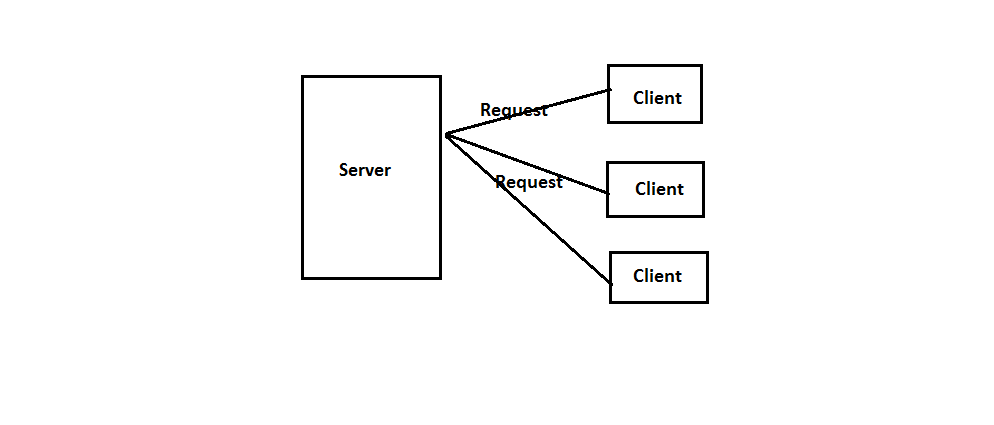


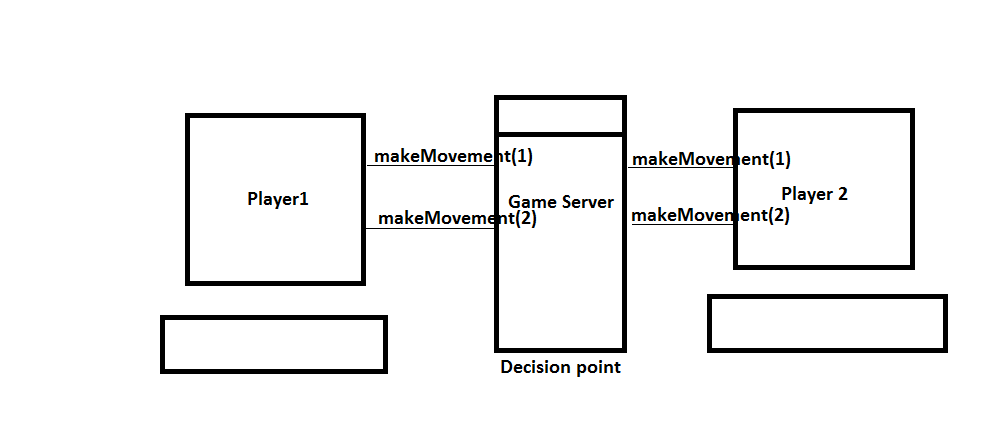




* 1. **Game Rules**
* Checkers is played by two players.
* Each player begins the game with 12 colored discs. (Typically, one set of pieces is black and the other red.)
* The board consists of 64 squares, alternating between 32 dark and 32 light squares. It is positioned so that each player has a light square on the right side corner closest to him or her.
* Each player places his or her pieces on the 12 dark squares closest to him or her.
* Black moves first. Players then alternate moves.
* Moves are allowed only on the dark squares, so pieces always move diagonally. Single pieces are always limited to forward moves (toward the opponent).
* A piece making a non-capturing move (not involving a jump) may move only one square.
* A piece making a capturing move (a jump) leaps over one of the opponent's pieces, landing in a straight diagonal line on the other side. Only one piece may be captured in a single jump; however, multiple jumps are allowed on a single turn.
* When a piece is captured, it is removed from the board.
* If a player is able to make a capture, there is no option -- the jump must be made. If more than one capture is available, the player is free to choose whichever he or she prefers.
* When a piece reaches the furthest row from the player who controls that piece, it is crowned and becomes a king. One of the pieces which had been captured is placed on top of the king so that it is twice as high as a single piece.
* Kings are limited to moving diagonally, but may move both forward and backward. (Remember that single pieces, i.e. non-kings, are always limited to forward moves.)
* Kings may combine jumps in several directions -- forward and backward -- on the same turn. Single pieces may shift direction diagonally during a multiple capture turn, but must always jump forward (toward the opponent).
* A player wins the game when the opponent cannot make a move. In most cases, this is because all of the opponent's pieces have been captured, but it could also be because all of his pieces are blocked in.

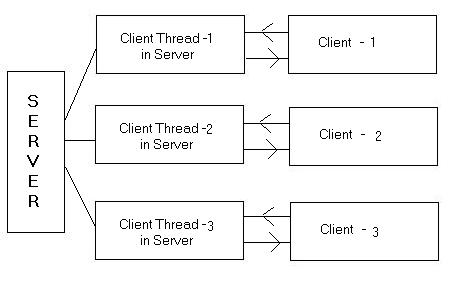
**2) Architecture Design**

****

****

In this application the master thread is the server and for every client request, a separate child thread is created. When the client1 thread requests for a game to other client2 thread, and the

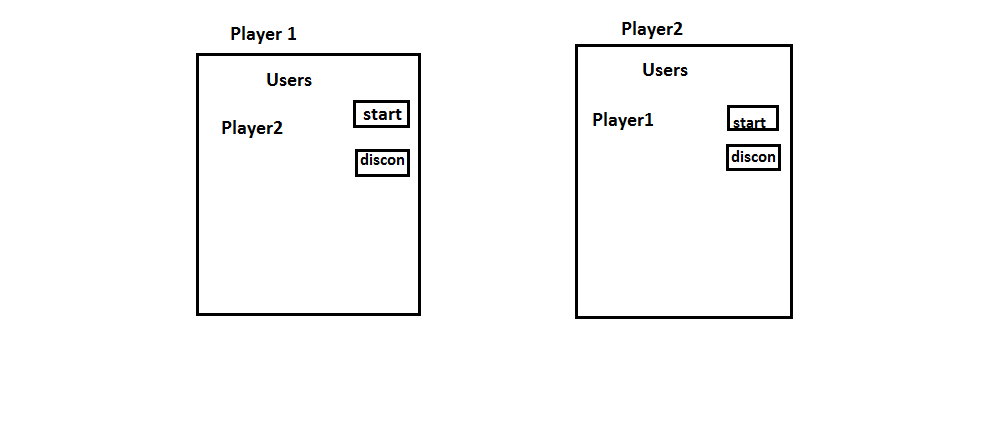
Game request is accepted, the server thread starts the game by creating a socket connection between the two clients and it is the server which manages the game and makes the decision of the game. When a move is made by the client1 thread, it is informed to the client2 by the server and vice versa. The communication between the threads is done by using inter thread communication in java.



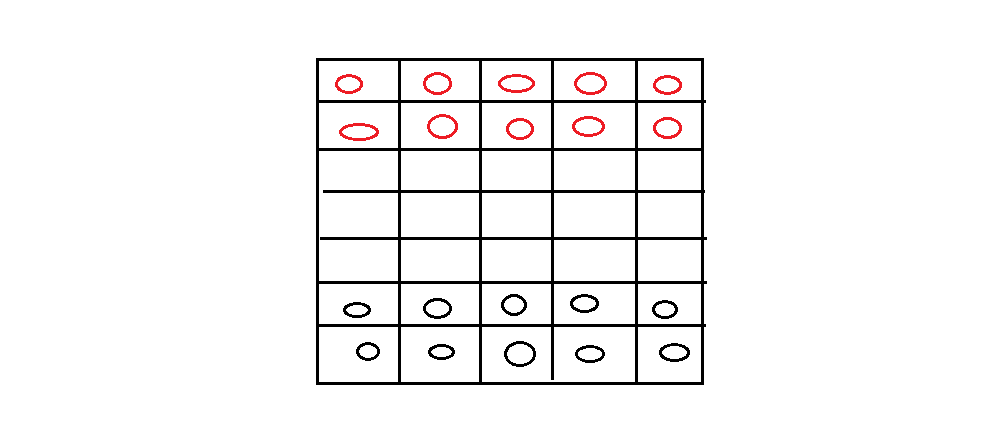
**3) Functional requirements**

1. In the Application the client can accept or deny the game request made by the other client.
2. The client can exit the game in the middle by clicking the disconnection button
3. Multiple clients can connect to the server and clients can see the list of other clients connected to the server.
4. Client can make game request of his choice. He can select the user from the list.
5. Clients can connect to the server from the different computers and play the game. It is not restricted to the local computer where it can behave as client and server.

**4) Graphical User Interface Design**

1) Client window

2) Checkers Board



**4) Implementation Environment**

Programming Language: JAVA (1.5 or higher version)

IDE Used : Eclipse

**5) Running the application.**

The application has two main programs. They are **client.java** and **server.java**

First server.java program should be executed by providing port no through run time arguments so that socket server is created.

Next client.java program should be executed by providing the server IP address, port and the name through run time arguments to connect to the server.

To create the multiple clients the client.java program should be executed multiple times.

The application can be executed from the command line.

The process for starting the server:

1. Open the command prompt
2. Go to the folder where the project is present
3. To run the server program go to server folder of the application and use the following command

cd source/server

javac server.java

java server<port no>

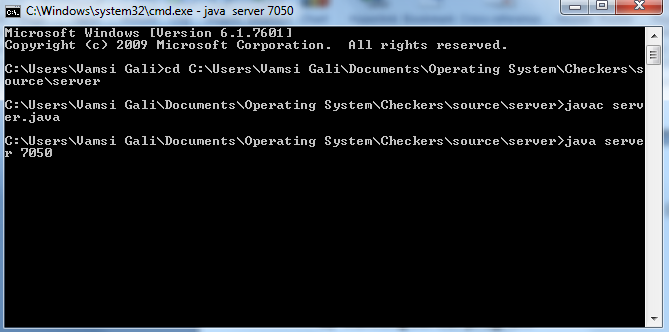
Process to run the client program

Cd source/client

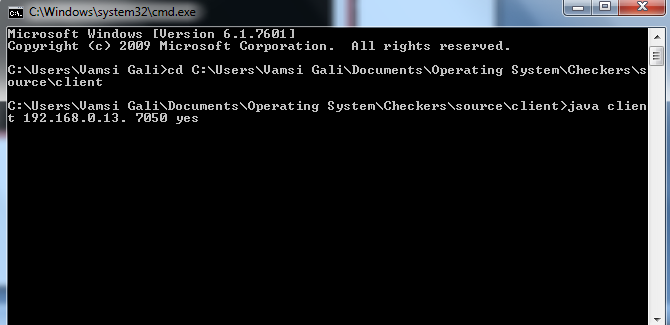
Javac client.java

Java client <IP Address><Port no><username>

1. **Running server program**



1. **Running client program**

****