University of Liverpool

Solving Nine Men's Morris

Final Year Project

Xiaoyi Cai

2012/5/9

Content

Abstract

This project is a problem-solving project supervised by Dr Sven Schewe. The aim of this project is to develop a piece of software that is capable of playing the game of Nine Men's Morris.

I decided to take Nine Men's Morris as my final year project because I think it is a challenge for me to do. I had never developed a computer game such as this. Moreover, it also provides a good opportunity for me to learn more about Distribution and game itself in particular. In second year study, just some theory of distribution has been taught and less practice for me to use it in real life. However, this project provides me a chance to apply what I learned in an actual game. These all attracted me to select this as my final year project.

The whole project is divided into four parts:

1. Background research
2. Planning and Design
3. Implementation and Realization
4. Evaluation

The project lasts for a whole year, and the final completed report deadline is 10th of May 2012. During the two semesters I handed in three reports, and gave two presentations about my report.

- The first report is about the specification. In the specification, I gave a general description of the project. This report also gave me a chance to describe the blueprint of the whole software. The expected features and desirable features were defined.
- The second report is about the design. During this part, the detail of the game was designed. It includes how to implement the game rules and distribution in my software. In addition, the game screen also designed, and some important parts of codes had been generated.
  After handing in the Design document, I did my design presentation. I present my design to my supervisor and marker face to face. They gave me a number of useful advises to improve my design. As a consequence, I had a rather wholesome preparation to the implementation part.
- The third report is Progress Report. This report just provided a brief description of my progress up to the third week of the second semester. It offered a wonderful opportunity for me to discuss with my supervisor about my project and the problem I met.
- Before the final report, a demonstration is needed. I was required to demonstrate my entire software to my supervisor and marker. I present all the features have been achieved. And my supervisor and marker provided some suggestions.

Introduction

Project Description

The project is a final year project supervised by Dr Sven Schewe, and it is about to develop an ancient board game named Nine Men's Morris. In the finished software, three game modes were implemented, which includes local game, online game and one player game. All of these three modes will be explained in detail in this report.

Aims and Objectives

The aim of this project is to design a computer version of this traditional game. The idea is to implement a program that allows playing the game on a computer. This has three aspects. The simplest one is that, using the program, a computer can be used as a board. In this most basic version, both players would sit in front of the same computer and play, just as they would play on a normal board. There would be two extensions of this basic version: First, I would implement a distributed version, where two different computers are used by the players, instead of using just one. This enables players to play from different places. Second, I implement an artificial opponent that will enable a player to play alone.

Challenges

There are two major challenges in this project. They are the connection of distributed machines and the development of an artificial intelligence for single player. I have some limited experience in the first, and no experience in the latter. My decision is to manage the risk involved by giving priority to the distribution.

At the end of the project, the distribution problem has been solved successfully. Two dislocated computers can be connected effectively in the game, although there still exist some bugs in online game mode.

Due to the time limitation, a wholesome artificial intelligence for single player mode was not developed. In the contrary, a very easy artificial intelligence which randomly choosing positions and counters has been produced. While, this also ensure that one player can play the game as well, and improve the playability.

Proposed Solution

I used the waterfall model, as it is the model anticipated by the generic description of final year projects. Also, it is a suitable design technique for projects of this size. (Some 200 to 250 hours on the project work, not counting presentations and their preparation and writing the dissertation.)

I used Java as the main programming language, and used Eclipse as my development tool. I chose Java, because I know it best among all suitable programming languages. Also, as a Java Virtual Machine is embedded in every browser, this allows using the developed software platform independent.

For the development of the GUI, I followed COMP106 module's note and found the number of source code in this area. Because I never did the GUI before, it took me a long time to be familiar with the method in GUI area.

For the development of the distribution, I reread the material of COMP211 and COMP212. Furthermore, a large amount of source code had been downloaded. Both of these gave me significant hint to do the distribution in my game. It took me about a whole week to write the code for distribution. After failed for a number of times, finally, a workable version was produced.

For the artificial intelligence, originally, I considered to research more on text book, and implement a heuristic evaluation of situations (the current state of a board) and use it to inform an A* based search. However, GUI and distribution parts took me much of time. As a result, not enough time was left to do a sturdy artificial intelligence. An easy intelligence was established, just randomly choosing the positions and counters.

Effectiveness of Solution

My software was effective in following ways:

- A workable two player in same machine version has been produced.
- The game allows two players to play on dislocated machines.
- One player can also play the game against computer, which equips with very weak artificial intelligence.

My software was ineffective in following points:

- The game does not have a wonderful interface.
- There still exist some bugs in distribution mode.
- The artificial intelligence is extremely weak in my software.

Background

Nine Men's Morris's History

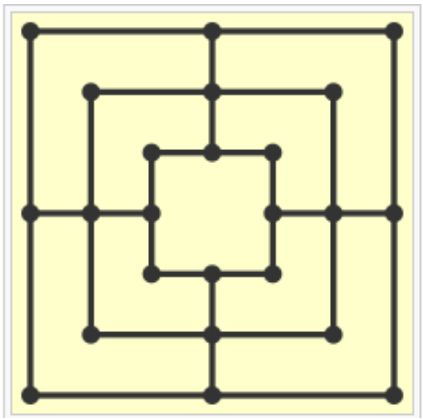
Nine Men's Morris is an abstract strategy board game for two players that emerged from the Roman Empire. The game is also known as Nine Man Morris, Mill, Mills, Merels, Merelles, and Merrills in English.

The knowledge is got from Wikipedia.

Game Rules

Nine Men's Morris is a game played by two players. Each player has nine pieces, or "men", which move among the board's twenty-four spots. The object of the game is to leave the opposing player with fewer than three pieces or, as in checkers, no legal moves.

A standard layout of Nine Men's Morris board is shown as follow:



The game begins with an empty board. Players take turns placing their pieces on empty spots. If a player is able to form a straight row of three pieces along one of the board's lines (i.e. not diagonally), he has a "mill" and may remove one of his opponent's pieces from the board; removed pieces may not be placed again. Players must remove any other pieces first before removing a piece from a formed mill. Once all eighteen pieces have been used, players take turns moving.
To move, a player slides one of his pieces along a board line to an empty adjacent spot. If he cannot do so, he has lost the game. As in the placement stage, a player who aligns three of his pieces on a board line has a mill and may remove one of his opponent's pieces, avoiding the removal of pieces in mills if at all possible. Any

player reduced to two pieces is unable to remove any more opposing pieces and thus loses the game.

The knowledge is got from Wikipedia.

Existing Game

I found an existing game on the internet at http://merrelles.com/English.html. In order to be familiar with the game, I played a number of times on this website. The game on the website has a strong artificial intelligence, which is lack in my software. The game set the level of difficulties, one is the default level and seven is the highest level. However, this version does not support the distribution game. In other words, it only allows the player to play against their computer, no way for two persons to play against each other. While, the inspiration of the game board and status bar in my game is come from this web game. In general, this version of web game provides me a lot of helps, my game screen.

Java

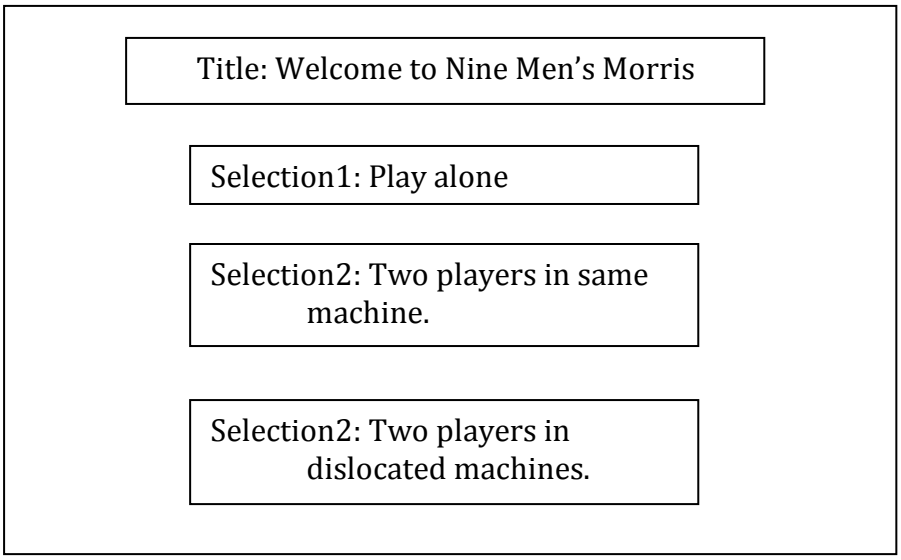
In the specification, I pointed that java will be the main language which will be used in this project. The reason of choosing java is that java is the language which I used most. Moreover, java is much more suitable for Object Oriented design which I used in my project. Java programming is a kind of Object Oriented programming, and it is used by much software.
Although I have used java for two years, there still exist some advanced aspects of the Java language has to be learned. Such as to design and produce a multifarious graphical user interface (GUI) and a connectable distribution system. I have no previous experience with both of these. As a result, I research deeply on GUI and Distribution, totally it cost me half a month to generate a simple GUI and workable distribution system.

Design

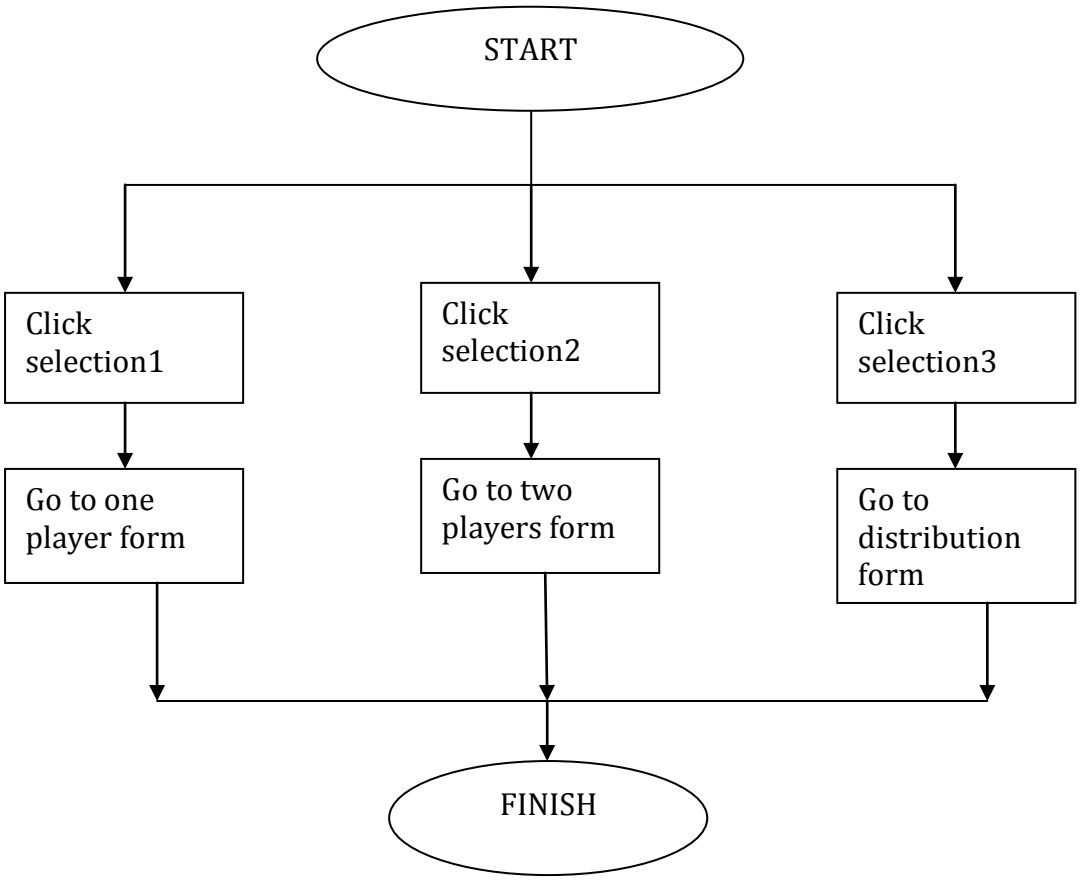
Initial Screen:

This is the first thing the user will see when he enters the game. According to the project aims and objectives, three selections will be shown on the screen. They are

"Play against with the computer", "Play with people in same computer" and "Play with people in dislocated machines"



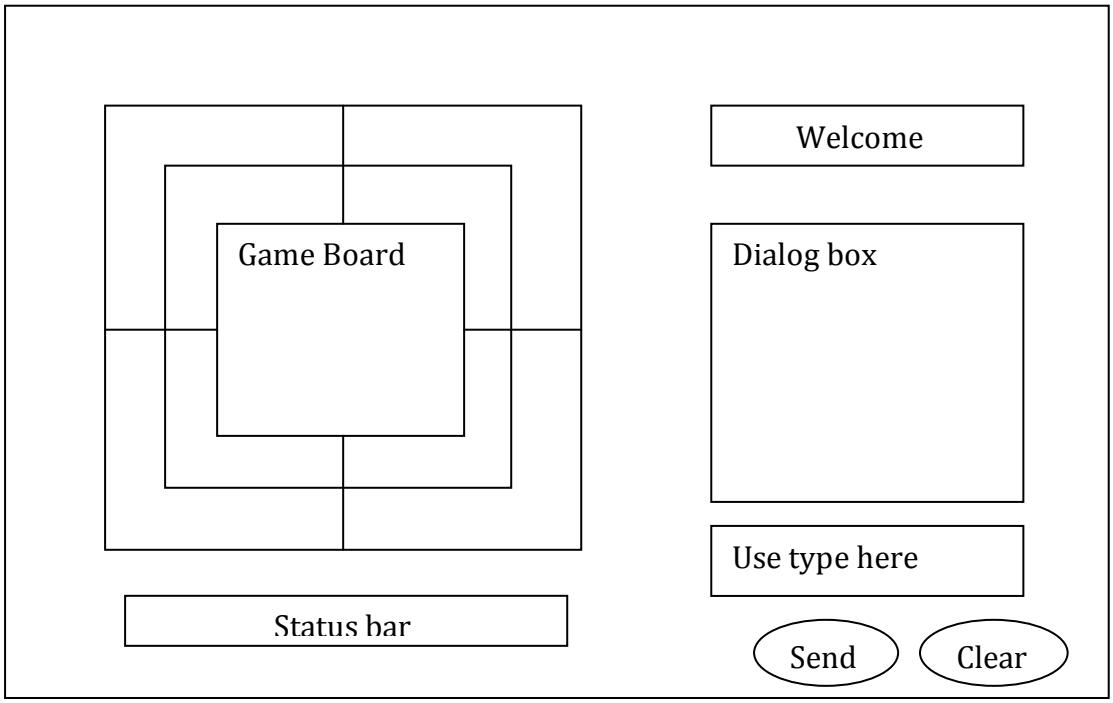
This is the flow chart to show what will happen, when the user clicks the button



Main Game Screen:

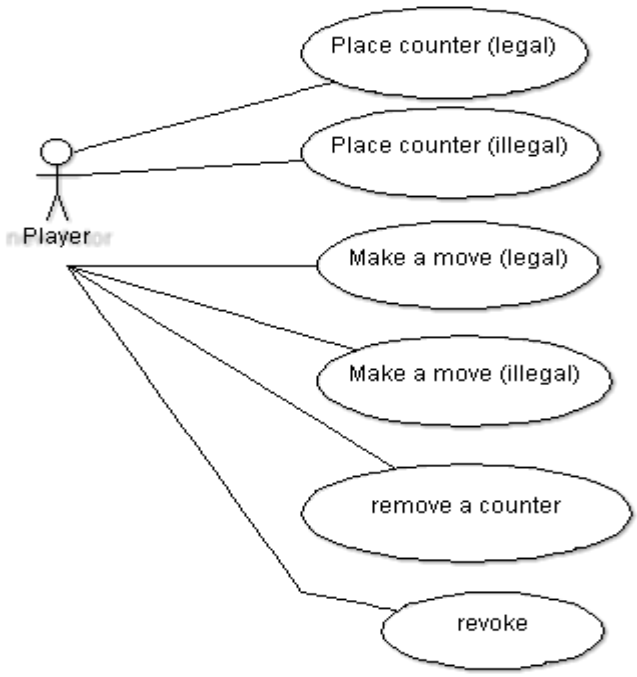
This is the main game screen of the game. After choosing the game model, this game screen will appear. In the local game, two players sit together and they can determine who takes the first step. In the one player model, human being player always places the counter first. In the distribution model, the player who enters the room first can take the first step. During the game, the status bar will update in order to tell the players what to do next. And if one player forms a mill or he/she do some illegal things, a message window will jump out to warn the player.

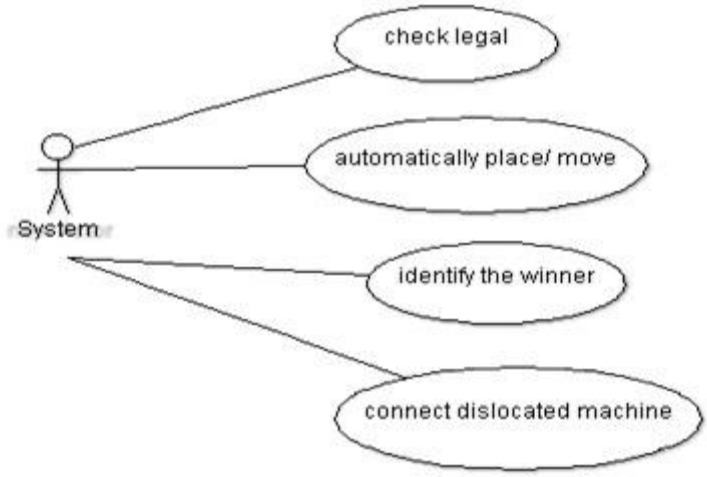
In the distribution game, two players locate in different places. So they can use the chat system to chat with each other. Player can type what they want to say in the input area, if player presses the button send, the message will appear on the dialog box, the other players who are in the room can see the message in dialog box. If the player realizes that he/she is writing some wrong message, he/she can clear what they input by using clear button.



Use case view:

The use case diagram is used to capture the requirements of the proposed solution. It shows what the users could do in the game. The diagram bellow gives us a generally view of the functions that the program will provided.





Use case1: Place a counter (legal)
Actor: Player
Pre-condition: The game started and there are counter left
Main scenario: The game begins with an empty board. Players should place the counters in turn. Players could only place the counter to empty place by clicking one of 24 points on the board. The program checks the counter placing is legal, and displays the counter in the screen.

Use case2: Place a counter (illegal)
Actor: Player
Pre-condition: The game started and there are counter left
Main scenario: The game begins with an empty board. Players should place the counters in turn. Players could only place the counter to empty place by clicking one of 24 points on the board. The program checks the counter placing is illegal, and returns an error message to player to tell him he needs to place the counter again.

Use case3: Make a move (legal)
Actor: Player
Pre-condition: All the counter have been placed correctly and the game is not over
Main scenario: Once all eighteen pieces have been used, players take turns moving. The program checks the moving is legal, and displays the counter in the screen.

Use case4: Make a move (illegal)
Actor: Player
Pre-condition: All the counter have been placed correctly and the game is not over
Main scenario: Once all eighteen pieces have been used, players take turns moving. The program checks the moving is illegal, and returns an error message to player.

Use case5: Remove a counter (legal)
Actor: Player
Pre-condition: The player has a mill
Main scenario: If a player is able to form a straight row of three pieces along one of the board's lines, he can remove one of his opponent's pieces from the board.

Use case6: Revoke
Actor: Player
Pre-condition: The play did something
Main scenario: If a player wants to replace a counter or redo a move, the player can click the revoke button to revoke one of his moves.

Use case7: Check legal
Actor: System
Pre-condition: Player did something
Main scenario: After the placement or the move which is made by player. The system has the duty to check if it is a legal placement or a legal move. Due to different situation, the system will return different messages.

Use case8: Automatically place/ move
Actor: System

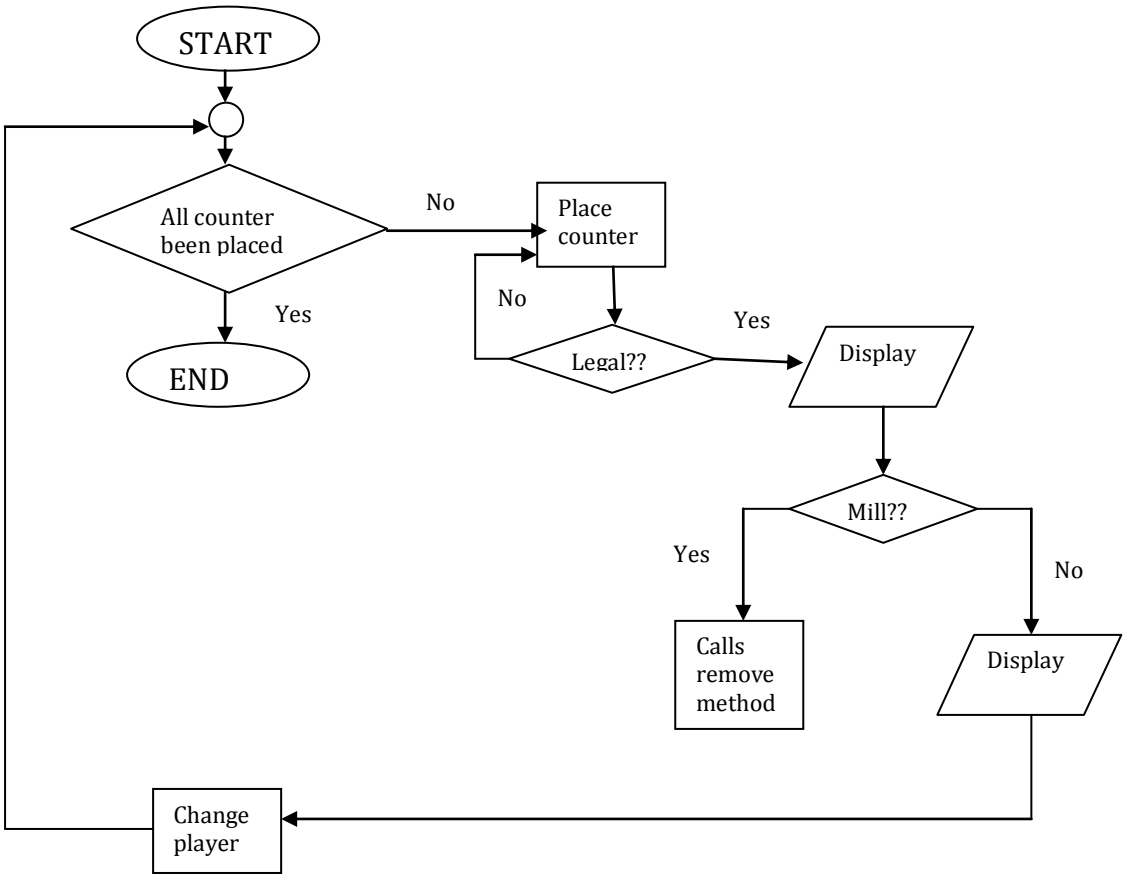
Pre-condition: Player choose one player game model
Main scenario: Because there is only one player, the computer needs to play with the player. The system will automatically place the counter and move the counter in the game.

Use case9: Identify the winner
Actor: System
Pre-condition: The game is in processing
Main scenario: When one player cannot move the counter any more, or just has 2 counters left, the system should send a congratulation message to the winner.

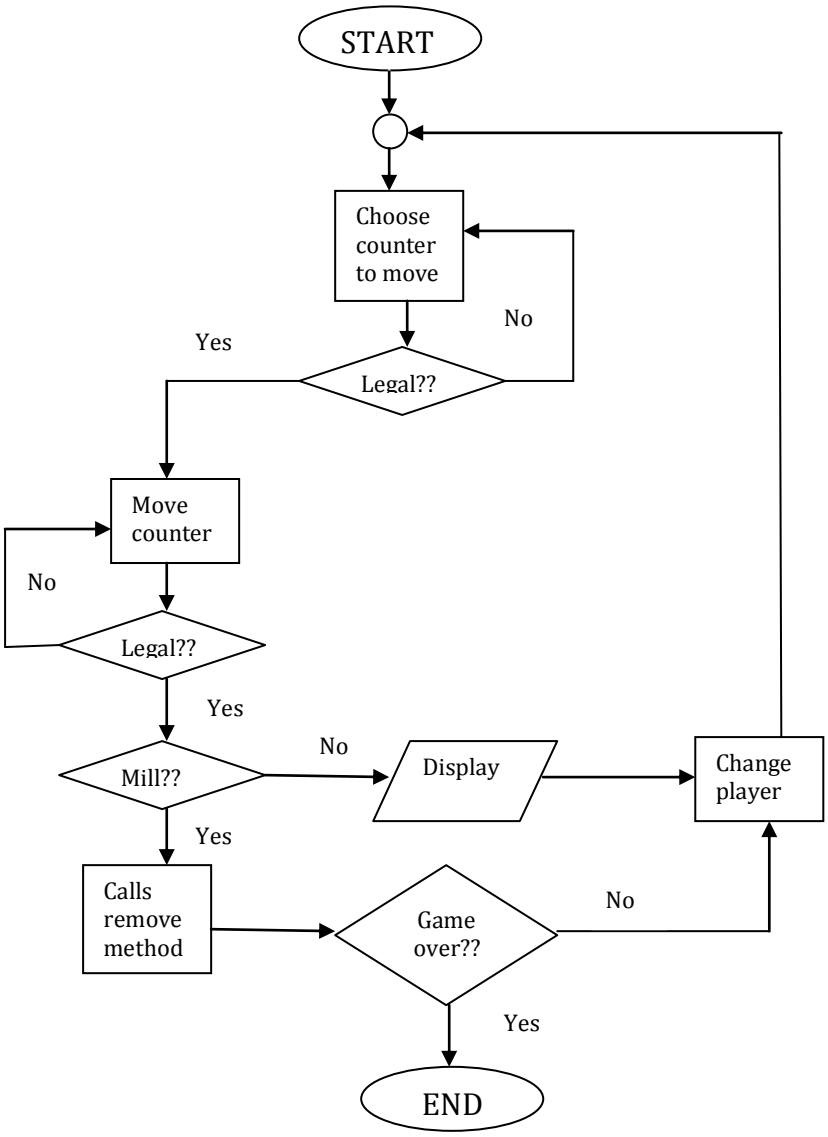
Use case10: Connect dislocated machines
Actor: System
Pre-condition: The player choose distribution game model
Main scenario: When a player wants to play with another player who has another computer, system needs to connect these two computers and let them play the game use different machines.

Flow Chart:

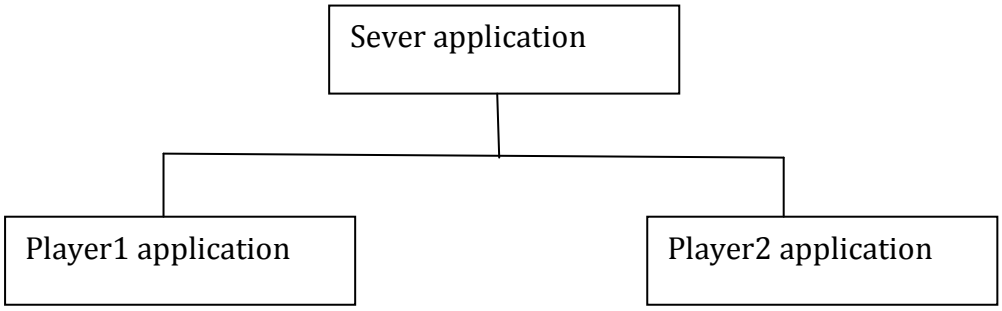
In this part, several flow charts will be shown. Flow chart is the diagram that represents the process. This diagram can give a step-by-step explanation to a given problem. Process operations are also represented in these boxes.
The fist chart will show the process of counter placing. Firstly, system will check if all the counters have been placed. If not, lets the player continue to place, and identifies whether there is a mill. If yes, call the remove method.



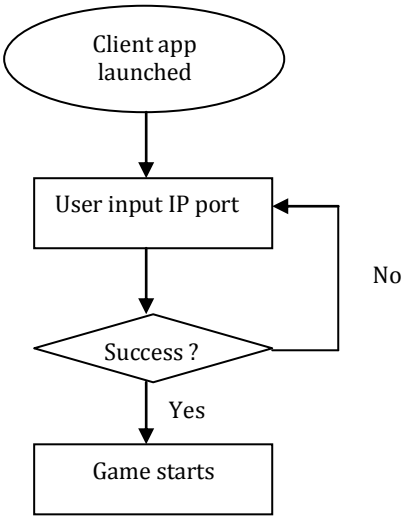
The second chart is about the moves. When all the counters are on the board, the player starts to move the counter with the aim of achieving a mill. In this phase, according to the rules, system will check if the movement is legal. Moreover, to find a mill is another task for the system.



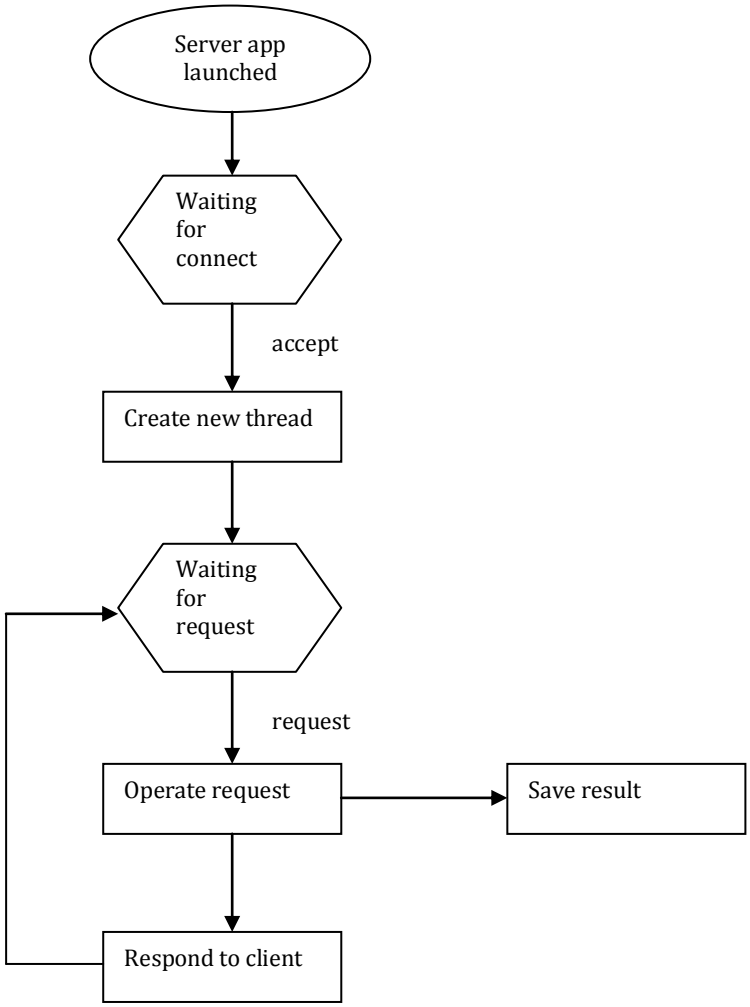
Next few paragraphs will talk about the distribution. Due to the fact that two machines are dislocated, a server application needs to be set. Any kinds of data from clients' machines need to go through the server. For example, if player1 wants to make a legal move, the move will first sends to server. And then, sever sends the move of player1 to player2's computer. There is no directly connection between player1's computer and player2's computer.
The diagram below presents the organization of the network, how do these three applications connect.



The flow chart below gives a brief view of what player needs to do before the game starts in client side.

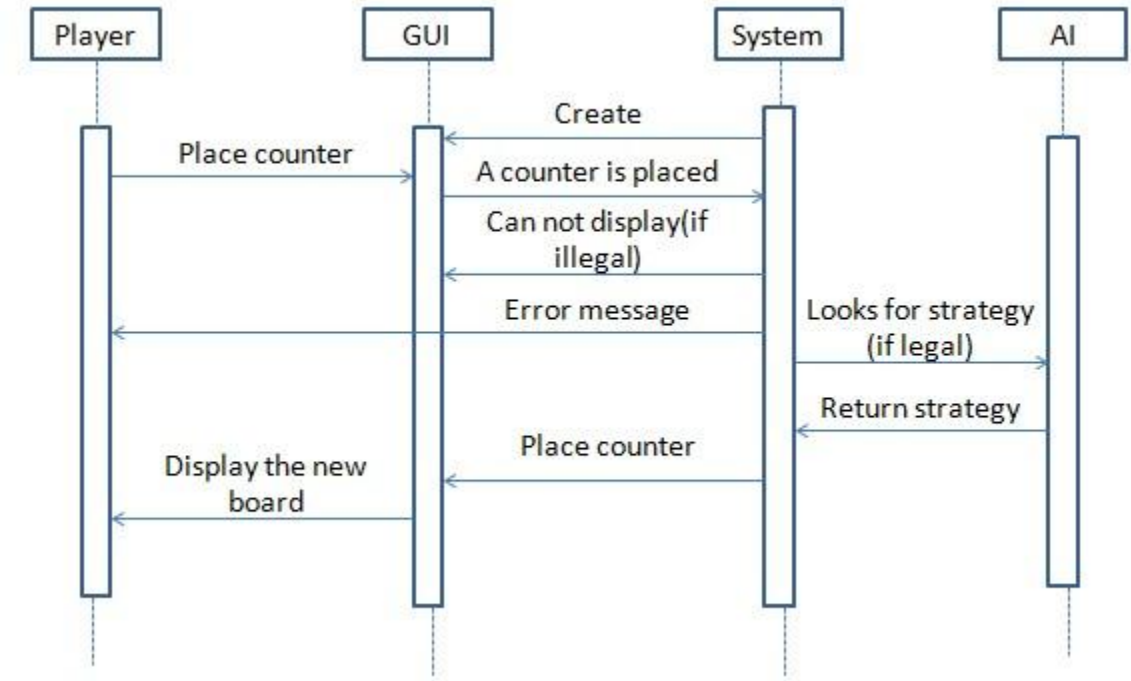


Server side is a little more complicated than client side. If there is a new connection between two clients, a new thread will be created. This thread will operate the request from the client. After the operation, sever saves the result. Finally, responds the result to the client and waits another request.

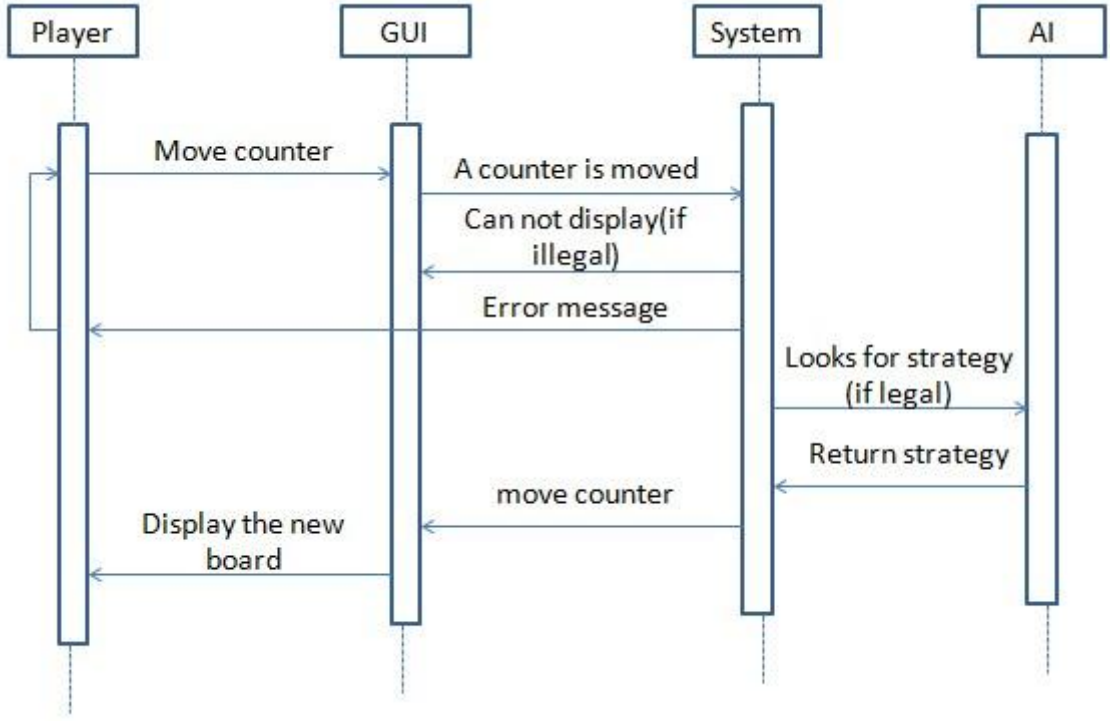


Sequence Diagram

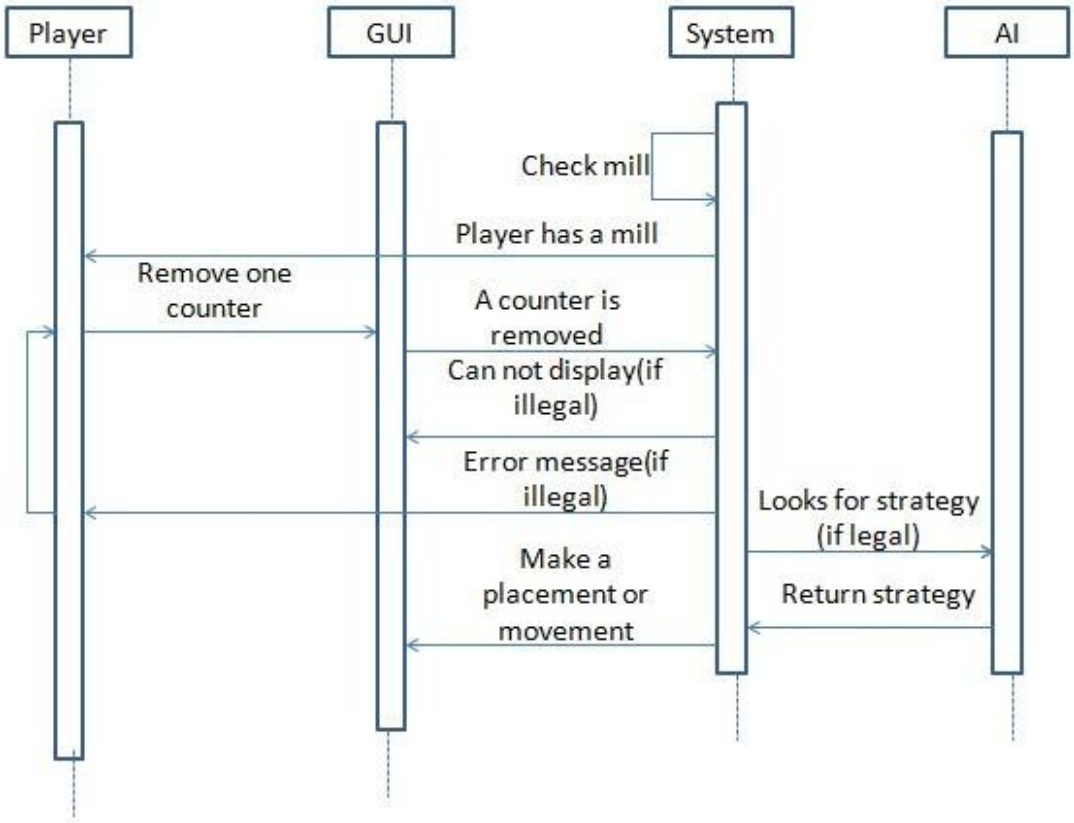
The first one talks about the placement procedure in one player game model.



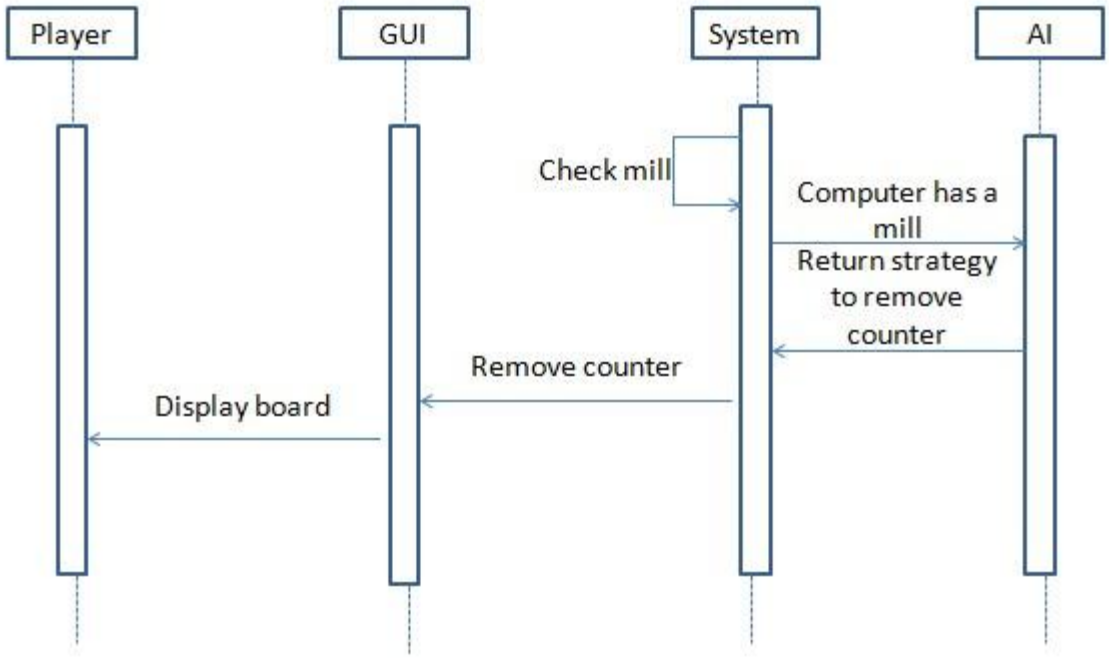
The second one is about the move procedure in one player model.



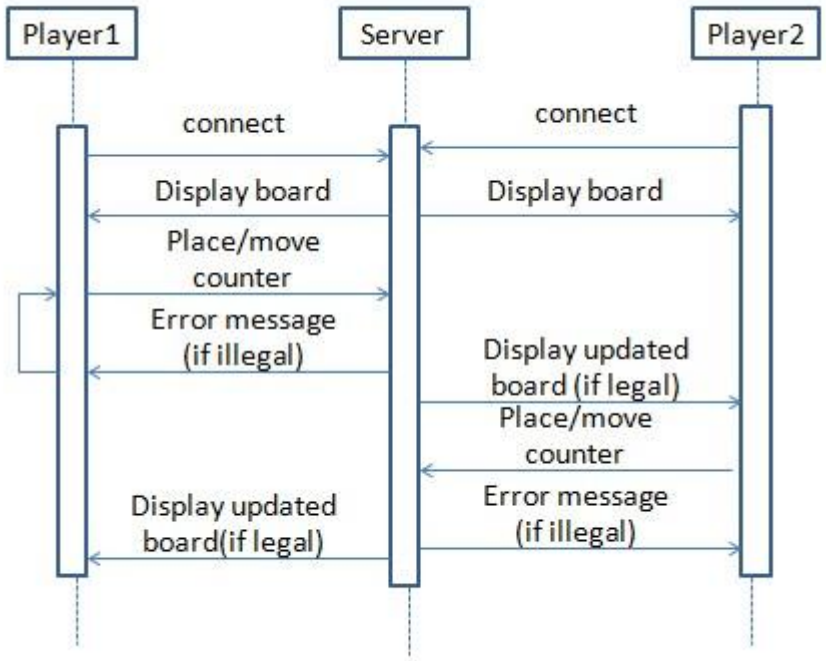
This one explains the situation about what will happen when player has a mill in one player game model.



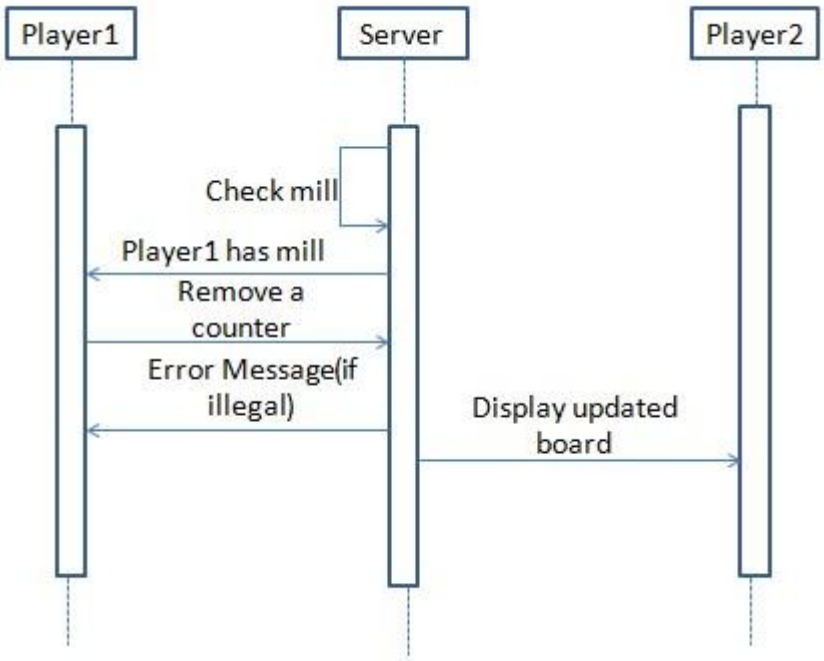
This one explains the situation about what will happen when computer has a mill in one player game model.



This diagram relates to the connection and the movement/placement in distribution model.

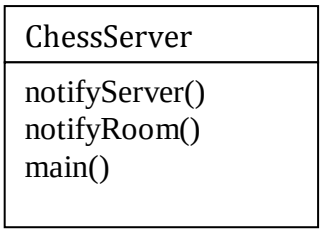


When one of the players has a mill, the program will do as follow diagram.



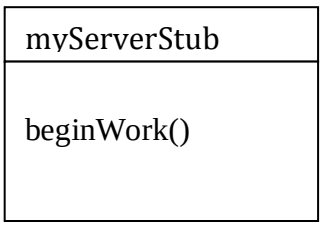
Class diagram

Distribution is one of the most challenge parts in this project. To solve this, I plan to create two packages in the programming procedure, one is sever, and the other is client. Package Server is used to start the server, create the new server socket and establish the working thread. Here I list some important classes and some significant method which I wish to use in programming.



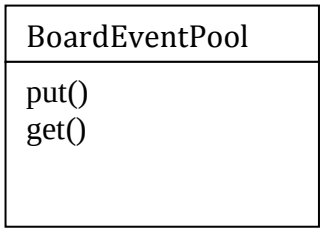
A chess server has 10 rooms, and each room has two players and some number of watchers. This class contains the room information and the players in it.

- notifyRoom() method is used to send messages to the players who are in the same rooms, so the game screen of all the players in this room can be exactly synchronized.
- notifyServer() method is used to send messages to the server.
- main() method first creates a new chessServer instance and a new server Socket with the default port number, for example ServerSocket(8888). Then use the chessServer instance and new ServerSocket to invoke the beginWork method which will be explained later to start the whole server.



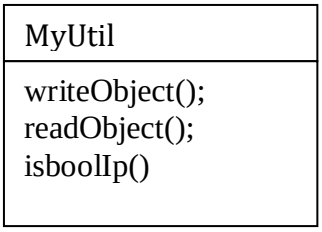
This class is used to send and receive information with client. The information means something like play event, register event, and etc.

- beginWork() method which mentioned before is the most significant method in this class or even in the whole server side. It may be just a few lines of code, but it starts two threads, which are writeThread and readThread. It can be known from the name, writeThread is used to write event to the players and readThread is used to read event.



The distribution of the project uses the Multi-Thread pattern. To be much more exactly, it should be a producer-customer pattern. As a result, an event pool is necessary. All events need to be sent to the pool or received from the event pool.

- Put() method is used to put an event to the pool, and notify the consumer that there is something to do. For example: In our program, the system put an event to the pool, and tells the chessServer to send this event to the client to process.
- Get() method is used to get an event from the pool to deal with, if there is no event there, wait for it until someone notify it (call function put). For example: In our program, the system wait for an event to arrive, if there is no event here, it waits. If there is an event, it picks it up, and sends it to the client.



This class is a class with common tools. It contains some methods which are frequently used in the other class.

- writeObject method is used to write the object to the OutputStream, this is used in java socket to send message to other ends.
- readObject method is used to read the object from the InputStream, this is also used in java socket, but it is to receive message from other ends
- isboolIp() method is to check if the string which is typed by customer is a valid IP address.

The four diagrams above are just four important classes in Server Package. Besides these four, it is certain that there will be more classes in the package. Something likes PlayerInfo class, Command class and etc… Due to the limited time, I only analyze the most challenge part in the Server package. Next, I will go into the client Package.

BoardClient

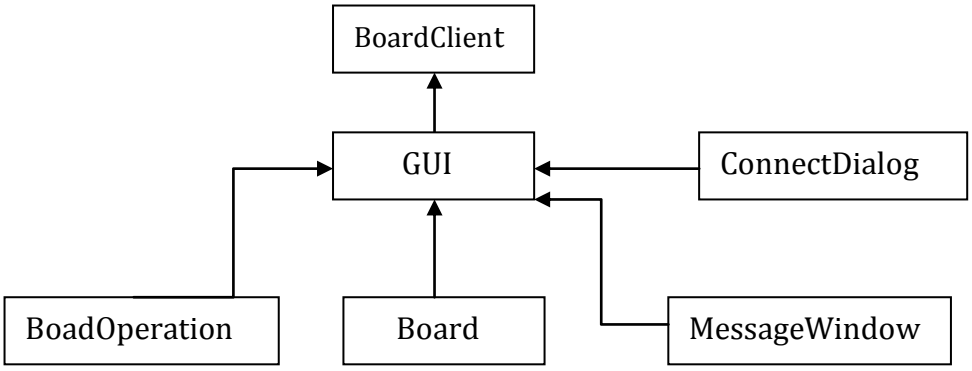
Main();
connect();
disconnect();

This class is used to start graphic user interface, connect to the server side and process some events.

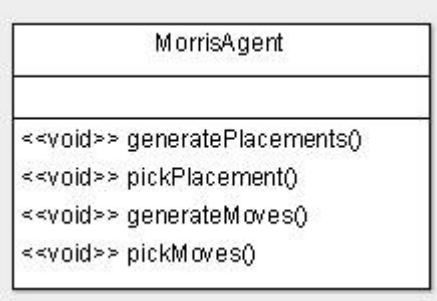
- Main() method which is needed in order to get the program run. It creates a new boardClient instance and uses this instance to invoke the method in GUI class, so the game window can be generated.
- Connect() method is used to connect to the server. Same as Server side, in Client side, there is also two threads, readThread and writeThread. And these two threads have the same utility as the two threads in server side.
- Disconnect() method is used to close the connection.

Same as the server package I just analyze the distribution part in client side, and a few methods about the distribution. In the client side, it still needs something like GUI, board, boardOperation, messageWindow and etc…

The relationship between the classes in the client side should like the following chart. The boardClient class has main method, so every class serves for it. GUI class is the core of the client side. It processes all the events in the game, because this class has the responsibility to determines what is going to be presented in the screen. Board class is to show the game board and the counters on it. The counter information is got from the GUI class. And GUI gets the counter information from the BoardOperation class. All counters on the board store to two vectors which is created in boardOperation class. The two vectors belong to two different players. MessageWindow class is used to produce several message windows due to different situation. For example, when the player makes an illegal placement and the system detects that, as a result, a message window about illegal placement should appear on the screen in order to warn the player, thus the player can replace his/her counter. The connectDialog class is used to generate a new window when the player chooses to play the game online. The dialog should contain the server IP address, port number, room number and the user name which will be used in the chat system in online game model.



For another most challenge part AI, I design to generate a class called MorrisAgent. This allows the computer can produce its own moves/placements and using appropriate heuristic function to choose the best.



- generateMoves() method generates all legal moves. And stores all the moves.
- pickMoves() method uses heuristic function to analyze the states generated by the method above. Moreover, it chooses the best one to show on board.
- generatePlacement() method generates all legal placements. And stores all the moves.
- pickPlacement() method uses heuristic function to analyze the states generated by the method above. Moreover, it chooses the best one to show on board.

Because there is a limitation on time for this project and distribution is a huge part in the whole program. I considered that if there is no enough time for building a strong artificial intelligence, I will give up to this thought, and to program this part in the future. In such situation, to be instead, a especially weak artificial intelligence will be produced by using random algorithm. All the placement and movement will be randomly generated by computer.

Realization

General Realization

In both specification and design documents, a gantt chart was produced. It is a clear and reasonable project plan which I need to follow. All works should to be done in time, which is clearly formulated in the gantt chart by milestone.

At first, all work was preceded in an orderly way systematically. The background research, specification document and design document were completed on time or even earlier than the time ruled in the gantt chart. However, when come to implementation stage, the difficulty of generate a workable game totally exceeds what I estimated before. A number of problems were occurred during the programming.

Owing to the problems was encountered in coding procedure, which will explained in detail in this section, the software was not accomplished by week 7 in the second semester which is planned in design document. Software was not completed until the week 9. In other words, the project is finished just before the project demonstration. As a result, I did the entire testing and evaluation after the demonstration.

Change to design

In original design for distribution part, the game rules would be programmed in server side. In coding procedure, I completed local game mode first. When I want to start distribution part, I used local game mode as the client side and found that it is too complicated to move the game rules part to the server side. If I did that, the code would be much more redundant. As a result, I made a decision to modify the original design, and keep game rules part in the client side.

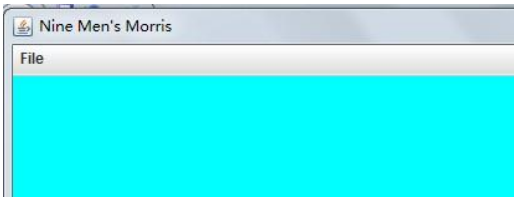
The second modification is to delete the initial screen. I clarified three game modes in the menu bar. Consequently, there is no need to program an initial screen. So in the actual software, there is no screen like the initial screen which is designed in design part, the game board is printed directly.

Software Realization

For the implementation, I split the entire project into four sections. They are user interface, board, game rules, distribution and artificial intelligence. All these sections will be given details in the following part.

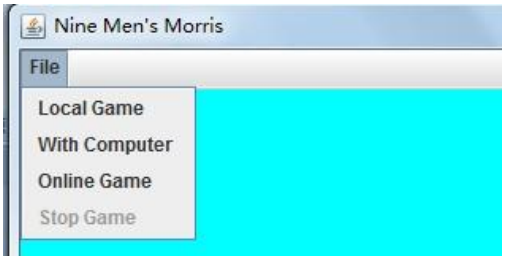
User interface

When the user enters the game, it should be an empty game window with a menu bar. Just one item named file is in the menu bar.

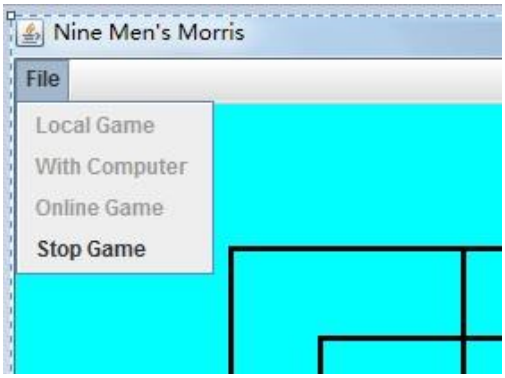


In the file menu, there are four selections, local game, with computer, online game and stop game. First three selections correspond to the three different game modes in the project. And the last selections are used to quit the game. When player chooses one of the first three options, the main game screen will appear including the board,

status bar and the chat system. When player chooses the last option, the game window will closed directly. In my software, the quit game option only can be used when the player enters one of the game mode. It is can not be selected when the player just enters the game without choosing the game mode.



After entering a game mode, the only thing that the player can do in the file menu is to stop the game. All the options except that can not be chosen. Just like the following picture.



This is the first time I need to produce a user interface like this. A lot of problems have been come across. At the beginning, I really do not know what to do first. I searched how to do a graphic user interface on the internet. Several documents and the number of source code were read. After the preparation, I create the empty window first. Secondly, using JMenuBar creates a new menu bar. Thirdly, adds four JMenuItem which correspond to the four options that mentioned before. Finally, adds action listeners to all of the options. Therefore the system can have exact response after player choosing different options. The code of this part should like followings.

```
public void CreateWindow(){

            this.setSize(680, 600);
            this.setResizable(false);

            cp.setLayout(null);
            cp.setBackground(Color.cyan);
            fileMenu.add(playLocal);
            fileMenu.add(playComputer);
            fileMenu.add(playOnline);
            fileMenu.add(stopGame);
            bar.add(fileMenu);
            setJMenuBar(bar);
```

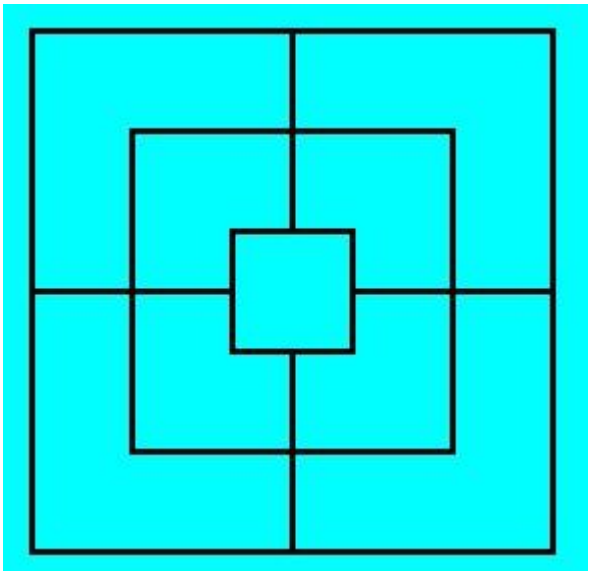
```
            this.stopGame.setEnabled(false);

            playLocal.addActionListener(this);
            playComputer.addActionListener(this);
            playOnline.addActionListener(this);
            stopGame.addActionListener(this);
            board.addMouseListener(this);


    }
```

Board

Compare to other tasks, to print a game board on the game screen is a relatively uncomplicated thing. In the second year's study, I have already learned how to draw lines by using language Objective-c. Both of Java and Objective-c use the same theory and some similar method, although there are some differences between these two languages. The game board that used in the game is like the following.



paintComponent() method was used to print the board, and by using repaint() method to update the game board when player has some actions like placing a counter and moving a counter. To draw the line on board is the most complicated task in this part. Firstly, I need to get the exact coordinate of the start and the end of the line. It is really a boring thing. In order to find the most suitable coordinate, several attempts have been made. In my software, the length of the lines in outer square are 260, lines in middle square are 160, and lines in inner square are 60.

```
g.drawLine(10, 10, 270, 10);
```

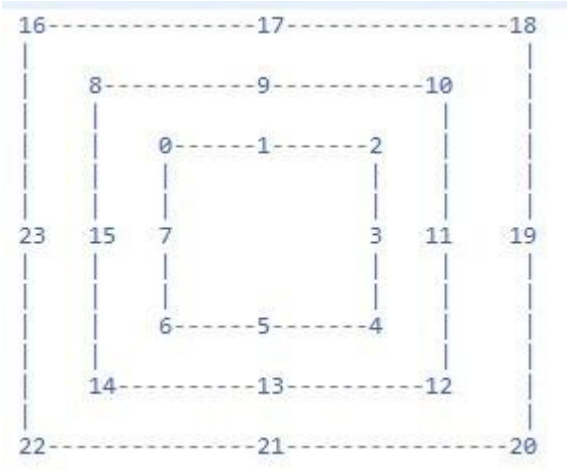
The method above is the code which I used to draw the line. The arguments in that method are the coordinate of the start point and the end point of the line.

```
Graphics2D g2d=(Graphics2D)g;
Stroke stroke=new BasicStroke(3.0f);
```

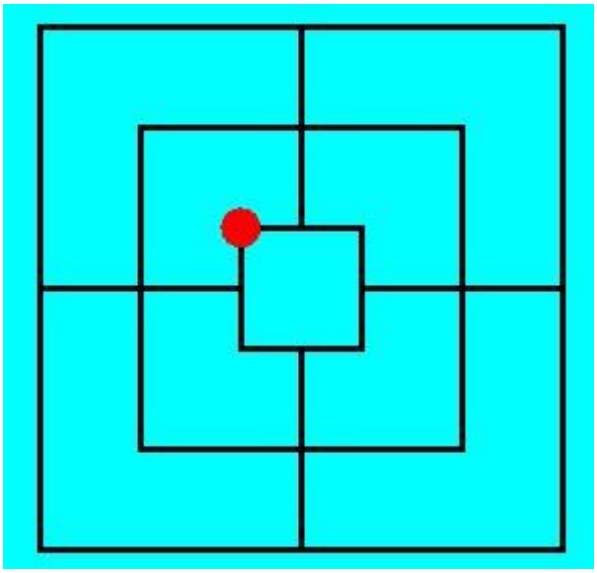
```
g2d.setStroke(stroke);
g2d.setColor(Color.black);
```

The first three sentences are found from the internet. Its function is to change the thickness of the lines. The arguments 3.0f in BasicStroke() is to determine the degree of thickness of the lines. If the number is larger, the lines will be thicker.
The last sentence is used to set the color of the lines. In Java, a number of colors can be chosen, including yellow, blue, white and etc… In my software, I chose cyan as my background color and black as my game board color.

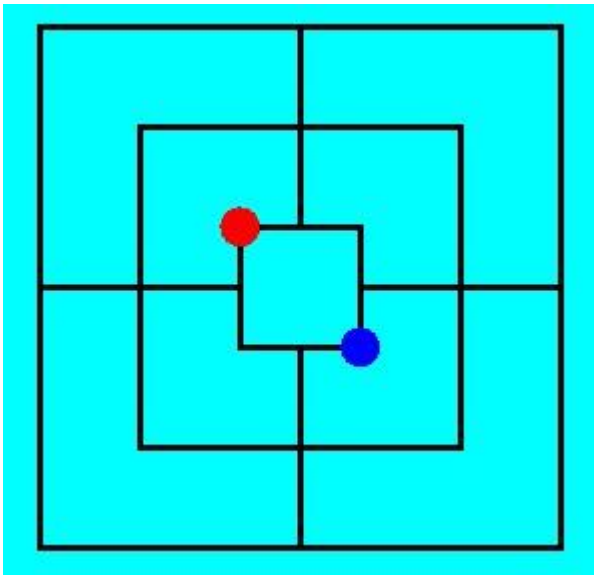
Another difficult task in this part is to print the counters on the board after the player has some actions. The game board need to keep updating during the game, it need to present the right action corresponds to the players. In the game, there are two kinds of counters, one is red counter and the other is blue counter. So, I use a set method in the board class, which can be invoked in other class to set two vectors. These two vectors are used to contain the two different kinds of counters' position. There are twenty-four points on which counters can be placed, so the position of each point is defined below.



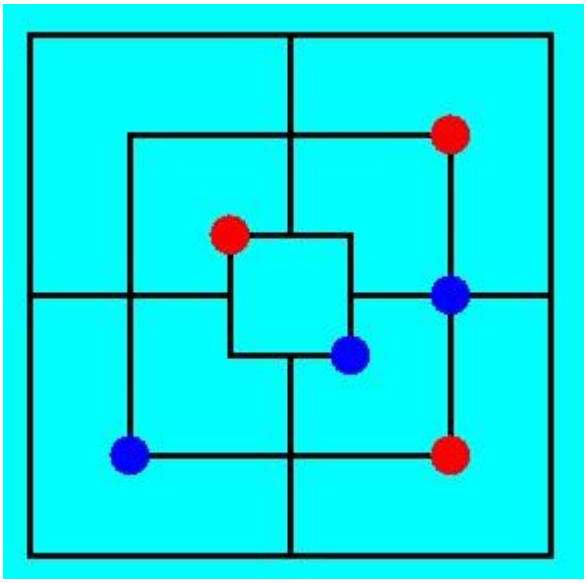
Due to the different operation by the player, the content of the vector of each player will be changed. For instance, at the beginning of the game, both of two vectors are empty, and after some placement, the vector will updated.



```
red is [0]
blue is []
```



```
red is [0]
blue is [4]
```



```
red is [0, 10, 12]
blue is [4, 14, 11]
```

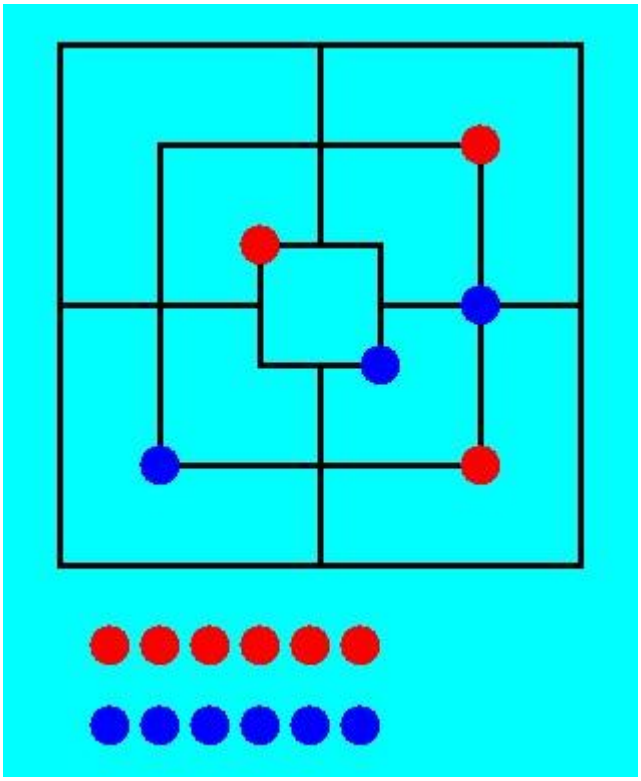
Because the board keeps updating, and need to print the correct counter at correct position, in each updating, each point needs to check whether a counter has been placed on it.

```
if(redStateStore.elementAt(i)==0){
	g.setColor(Color.red);
	g.fillOval(100, 100, 20, 20);
}
```

The code like above is used in my project. In this example, the system checks the red vector which named redStateStore in the project. If there is a red counter on the position 0, then a counter will be printed out at the right coordinate by using fillOval() method. Moreover, the left counters are also printed on the screen. After each placement, the left counters will be reduced. When there is no left counter on the screen, that means all counter has been placed, the player can start to move the counter. While, due to the rules of the game, the counter which is removed by mill can not be placed on board again. So the left counter will not be added when one counter has been removed. The code and screen shot is like following.

```
  for(int m=1;m<=redCounter; m++){
	total = (25*m);
	g.setColor(Color.red);
	g.fillOval(total, 300, 20, 20);
}

for(int m=1;m<=blueCounter; m++){
	total = (25*m);
	g.setColor(Color.blue);
	g.fillOval(total, 340, 20, 20);
}
```



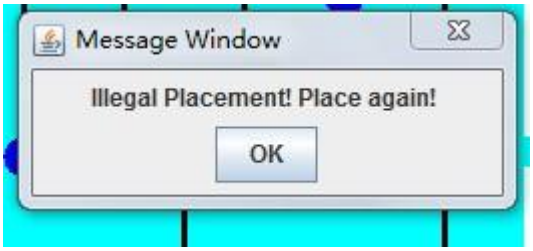
Game rules

In the Nine Men's Morris game, according to my analysis, there are four main steps. First is the placement procedure, two players need to place the counter on the board in order. Second is the movement procedure, when all the counters are placed on the board, the player starts to move the counter in order to have a mill and beat the opponents. Third is the flying procedure. When one player has only three counter left on the board, the counters belong to that player can be moved to any empty place on the board. Fourth is the removing procedure. When one player is able to form a mill, that means that player can remove one of his/her opponent's counters. And the removed counter can not be placed to board again. Moreover, when one player has only two counters left on board, that player lose the game. Consequently, first three steps are all aiming to structure a mill. In this part, the four steps stated above will be clarified one by one.

Placement

At the beginning of the game, red player needs to place the counter. When player choose a point to put the counter, he/she should click that point by using mouse. Firstly, according to the design, system should check whether this placement is

legal or not. If it is an illegal placement, a message window like below will appear to tell the player that replacement is needed.



The system will not let the other player to place the counter until the current player has a legal placement. The check legal method mainly checks two things. It has the responsibility to check if the chosen position already has a counter on it. For this situation, it needs to check the red state store and blue state store which are mentioned before. Moreover, it needs to check if the player clicks in the allowed area. Because, in my project, only the area near the cross point of the board lines can be clicked. Any click outside theses areas are illegal. The code is like below.

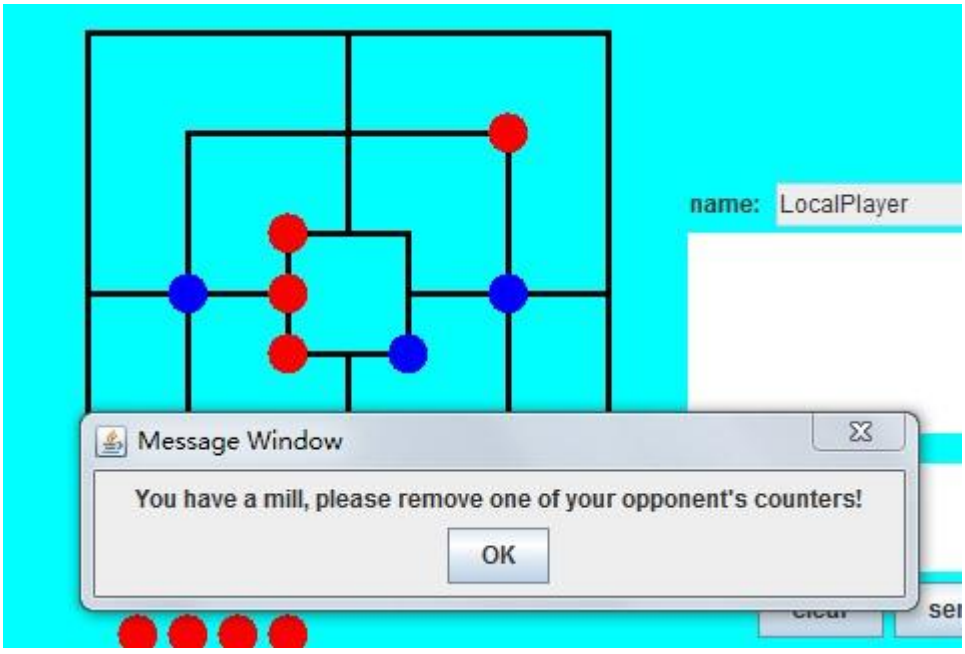
```
public boolean checkLegal(int p){
	for(int i=0; i<redStateStore.size();i++){
		if(p==redStateStore.elementAt(i)){
			return false;
		}
	}
	for(int j=0; j<blueStateStore.size(); j++){
		if(p==blueStateStore.elementAt(j)){
			return false;
		}
	}
	if(p==1000){
		return false;
	}
	return true;
}
```

When the placement passes the first inspection of the system, then system will check who put this new counter to determine whether a blue counter or a red counter should be appeared on board. The system adds the new position to corresponding vector, and reprint the board. Subsequently there will be another inspection which called mill check in my software. In this inspection, system will check all counters on board to confirm that if there is a new mill has been formed. This is one of the most complex detects procedure in the whole project. Some especial points were picked up to check if the points on the same line have been occupied by the counters from same player. The code for check mill is like below.

```
if(a.elementAt(i)==1){
	for(int j=0; j<a.size();j++){
		if(a.elementAt(j)==0){
			count1++;
		}
		if(a.elementAt(j)==2){
			count1++;
```

```
                }
            }
            if(count1==2){
                return true;
            }
            if(count1%2!=0){
                count1=0;
            }
        }
```

These pieces of code just used to check if there is a new mill on the upper row line of the inner square. If there is a new counter has been placed on position 1, the system will check position 0 and position 2 automatically, to see if there are same color counter on them. The count1 used here initialize as 0 at the start of this class. When the count1 equals to 2, it means there are same color counters on position 0 and position 2. In other words, there form a new mill. While, in the next check this mill will also be detected. But due to the rules of the game, the old mill can not be used unless it was destroyed once. And in my code, once the formed mill in this example has been destroyed, count1 will equal to 0 again. So if and only if count1 equals to 2, the system can returns true, then the system will give a message window like the picture below to tell both players that there is a new mill. If there is no new mill, the other player can place a new counter and have the same process like above.



In this part, the entire code of player who takes the red counters should like this.

```
if(step==1){
        if(bo.checkLegal(position)==true){
                if(player==1){
                        bo.addStateRed(position);
                        leftRed=leftRed-1;
                        board.set(leftRed,leftBlue);
                        if(bo.checkMill(bo.getRedVector())==false){
                                player++;
```

```
                    }
                    else{
                        board.set(bo.getRedVector(), bo.getBlueVector());
                        addGameBoard()
                        MessageWindow mill = new MessageWindow(null);
                        mill.millInfo();
                        mill.setLocation(500, 250);
                        mill.show();
                        step=2;
                    }
                }
            }
            else{
                MessageWindow illegalPlace = new MessageWindow(null);
                illegalPlace.illegalPlace();
                illegalPlace.setLocation(500,250);
                illegalPlace.show();
            }

}
```

Movement

When all counters have been placed on board, players start to move their counters in turn. In this part, firstly, the system will check how many red counters and blue counters left on board. For example, if this is the red turn, then system checks the total number of the red counters on board. If there is only three red counters on board, according to the game rules, the red player can fly one of his/her counters. The player needs to choose which counter he/she wants to move, and then he/she can choose any empty position on board. In this part of code, firstly, an inspection method to check whether the selection of the counter is legal is programmed.

```
public boolean detectSelectedRedCounter(int p){
            for(int i=0;i<redStateStore.size();i++){
                if(p==redStateStore.elementAt(i)){
                    red++;
                }
            }
            if(red!=1){
                red=0;
                return false;
            }
            else{
                red=0;
                return true;
            }
        }
```

The method above returns a Boolean value, through which can be seen that whether the selection is legal or not. An integer argument is used in this method. It is the position which the player has chosen. In the example above, the system checks every position that stored in red state store to find out that if the chosen position belongs to

the red player. If there is a position in the store same as the argument, it means it is a legal selection, or it should be an illegal selection, and current player needs to reselect.

If it is a legal selection, then next click should be the point where the current player wants to fly to. System also needs to check whether it is an empty position. When the system confirms that it is able to fly to the chosen position, the corresponding vector will delete the original position and add the new position. Then change to the other player, and repeat the same procedure. The code just likes below.

```
bo.removeStateRed(bo.getChooseCounter());
bo.addStateRed(position);
countRed=1;
player++;
```

Because one player needs to click the board twice and have different purpose, I used an integer countRed to control which procedure should be entered after each click. If countRed equals to one, it should be the choosing counter procedure. If countRed equals to two, it should be the moving counter procedure.

While in the situation of that there are more than three red counters or blue counters left on board, there are a little bit differences. Same as the situation above, the selection needs to be checked by using the same method. While, in the moving procedure it has different check method. Due to the rules, the counter can only be moved to the empty position which is near to its original position. The check method used here should be much more complicated than others. Firstly, the system needs to check if the position which the player wants to move to is empty. If it is not empty return a warning window to tell the player to move again.

```
for(int i=0;i<redStateStore.size();i++){
	if(q==redStateStore.elementAt(i)){
		return false;
	}
}
for(int j=0;j<blueStateStore.size();j++){
	if(q==blueStateStore.elementAt(j)){
		return false;
	}
}
```

If it is empty, the system will also need to check if the position that the player has clicked is near the counter's original position. This inspection is a little difficult, system need to compare two positions to see whether the move between these two positions is against the rules.

```
if(q==p+1||q==p-1){
	return true;
}
if(p==0&&q==7){
	return true;
}
```

In the second part of this inspection, there are two situations. In the first situation, it is much more general. Just calculate if the moving position is the former number or the

latter number of the chosen position. While in the second position, several special numbers need to be checked. Because there are four lines cross the outer, middle and inner squares. The counters can be moved on these four lines and the position numbers on these lines are not continuous. Moreover, the start position of each square is also not continuous with the end of end position of each square. In the example above, 0 is the start position of inner square, 7 is the end position of inner square. And counter can be moved from position 0 to position 7.

When the movement passes two inspections above, if it is legal, update the corresponding vector. After the updating, it also needs to check whether there forms a new mill by using the method which is mentioned in placement part. If there is a mill go to the mill procedure, if not change a new player.

```
if(bo.detectIllegalMove(bo.getChooseCounter(), position)==true){

	bo.removeStateRed(bo.getChooseCounter());
	bo.addStateRed(position);

	if(bo.checkMill(bo.getRedVector())==false){

		player++;
		countRed=1;
	}
	else{
		board.set(bo.getRedVector(), bo.getBlueVector());
		addGameBoard();

		MessageWindow mill = new MessageWindow(null);
		mill.millInfo();
		mill.setLocation(500, 250);
		mill.show();
		sign=3;
		step=2;
		countRed=1;
	}
}
```

The integer sign here is used to let the system go back to movement procedure, after existing from the mill procedure. Due to the fact that if there detects a mill, the system must go to mill procedure, no matter it is originally in placement procedure or movement procedure. It is important to ensure that the system can go back to correct procedure, so an integer is designed to control this problem. If sign equals to three it should go back to movement procedure, else it needs to go back to placement procedure.

Mill

This part may be the most important part in the game itself. Because if this part can not work successfully, there will be no ways to produce a winner of the game. Both of placement procedure and movement procedure have chance to use this part of code. This part is mainly about removing opponent's counters. The only inspection used

here is to check whether the removing is legal according to the rules. The code for this inspection is quite easier.

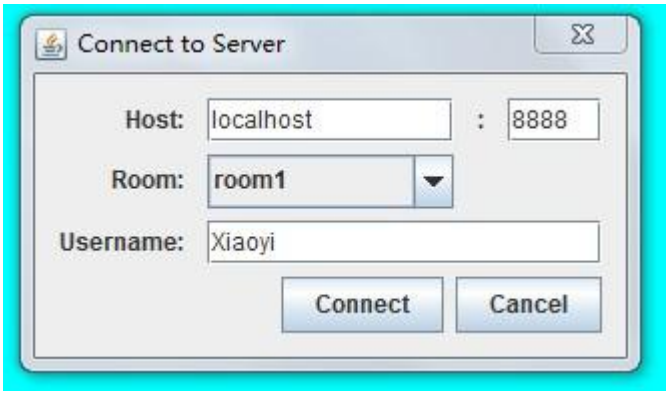
```
public boolean checkRedRemove(int p){
	for(int i=0;i<redStateStore.size();i++){
		if(p==redStateStore.elementAt(i)){
			return false;
		}
	}
	return true;
}
```

In the example above, the blue player has a mill and a red counter can be removed. Blue player should select a counter which he/she want to removed. Then, the system called this method to check if the selected counter is in the red state store. If the counter is removable then update the corresponding vector and go back to the original procedure, if it does not pass the inspection, the current player need to remove opponent's counter again.

Distribution

This is the most challenged part in the whole project. And this part divided into two parts, client side and server side. Both of these will be explained in this part.

Firstly I will talk about client side. When the player choose online game mode, a connection dialog will jump out. Just like the bellowing picture.



The player needs to type the IP address and port number of the server machine. Furthermore, the room also needs to be chosen, the server in my project provides ten rooms, room1 to room10. The players can decide with their friends and choose anyone of them. The last text field is about the username, which will be used in chat system. When the player in put something, the name will present before the typed sentences, like following.

When the player typed all the information, player should click the connect button which is at the bottom of the dialog.

```
bCancel.setText("Cancel");
  bCancel.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(java.awt.event.ActionEvent evt) {
          bCancelActionPerformed(evt);
      }
  });

  bConnect.setText("Connect");
  bConnect.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(java.awt.event.ActionEvent evt) {
          bConnectActionPerformed(evt);
      }
  });
```

Once connect button has been clicked, bConnectActionPerformed() method will be called, this method used to check if every information has been typed correctly. If yes, it will store the typed information.

```
cif = new ConnectInfo(host, port, room, username);
```

This sentence is used in bConnectActionPerformed() method, to store the information that typed by the player. And then use these information create a new instance called connectInfo in my project.

```
this.cif = dialog.getConnectInfo();
try {
    this.conn.connect(cif);
} catch (Exception e1) {
    e1.printStackTrace();
}
```

After that, this new instance will be used as argument to invoke the connect method. In the connect method, using the server's IP address and port number connect to the server, and create a new input stream and a new output stream.

```
server = new Socket(InetAddress.getByName(cif.getHost()), cif.getPort());
System.out.println(String.format("Connection to server %s:%s successfully!", cif.getHost(), cif.getPort()));
final InputStream in = server.getInputStream();
final OutputStream out = server.getOutputStream();
```

For distribution mode, in the client side, every position that clicked by player is sent to the other players in the same room by using the following code. Then, the other player can operate the coordinate with its own algorithm. So this action can synchronize the game screen of both players.

```
public void mouseClicked(MouseEvent e) {
    int x = e.getX();
    int y = e.getY();
    this.doEvent(x, y);
    //send the event to the other side
    if(otherPlayer!=null){
        msg.setText("<HTML><FONT COLOR='RED' SIZE='5'>Hi, " + cif.getUsername() + ". Enjoy it~</FONT><HTML>");
        otherPlayer.doEvent(PlayInfo.buildPlayEvent(cif.getRoom(), cif.getUsername(), x, y));
    }

}
```

While, it can not work successfully without the read thread and write thread. These two threads are used to send and receive event from or to other players. All the events need to go through the server. The code of these two part is like following.

```
writeThread = new Thread(new Runnable() {
    @Override
    public void run() {
        while (true) {
            try {
                MyUtil.writeObject(out, outEventPool.get());
            } catch (Exception e) {
                e.printStackTrace();
                break;
            }
        }
    }

});

// Receive the event from other players;
readThread = new Thread(new Runnable() {
    @Override
    public void run() {
        while (true) {
            try {
                processResult((PlayInfo)MyUtil.readObject(in));
            } catch (Exception e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
                break;
            }
        }
    }

});
```

Next, the server side will be explained. If the player wants to play the online game, he/she needs to confirm that the server application is currently running. The server can be started by using the bellowing code.

```
System.out.println("Server Starting...");
ChessServer chessServer = new ChessServer();
ServerSocket socketServer = new ServerSocket(DEFAULT_PORT);
while (true) {
    final Socket socket = socketServer.accept();
    System.out.println("Accepted a client " + MyUtil.getClientInfo(socket) );
    new MyServerStub(socket, chessServer).beginWork();

}
```

To make the server running is by creating a new server socket with a port number. And then, an infinite loop is used to ensure that the server is running and working all the time. Inside the loop, a method called beginWork() is invoked. This method is used to start two working threads. Same as the client side, these two threads are read thread and write thread.

```
public void beginWork() {
    new Thread(this.writeThread).start();
    new Thread(this.readThread).start();
}
```

These two threads have the same working theory as the threads in client side. So, that piece of code will not present here, the full details can be found in the appendix. Here I will just explain one method used in thread called put() which is used in bellowing sentence.

```
MyUtil.writeObject(out, eventPool.get());
```

eventPool is an instance of class boardEventPool, which is generally used in producer-customer problem. The distribution problem likes this pattern more or less. That is why a events pool is created here. In the events pool there are two main method called set and get. Set method is used to put an event in the pool. In opponent, get method return an event object, which is needed by server or clients. The code of this part is like below.

```
public synchronized void put(T event) throws InterruptedException {
    while (count > eventPool.size()) {
        wait();
    }
    eventPool.add(tail, event);
    tail = (tail + 1) % eventPool.size();
    count++;
    notifyAll();
}

public synchronized T get() throws InterruptedException {
    while (count <= 0) {
        wait();
    }
    T event = eventPool.get(head);
    head = (head + 1) % eventPool.size();
    count--;
    notifyAll();
    return event;
}
```

These two methods are clarified as synchronized method, in order to avoid the deadlock, which is a common problem in producer-customer pattern. The integer count used here is to control the synchronization. When count is larger than the size of events pool, it means the put method is being used by other event. The new event needs to wait until it finish. However, when count is equal zero, it means the events pool is empty. Nothing can be got from the pool.

While, building distribution is a rather large and complicated project. I can not explain every step as what I did in game rules part. Nevertheless, all the rather important methods were shown and clarified in this realization part. Besides, the full code with comments will be listed in the appendix. More details about realization of distribution can be found through that.

Artificial intelligence

In my project, only a very weak artificial intelligence was produced. In the one player mode, the human player always places the counter first. And then, system will generate a legal placement by random algorithm by using the following method.

```
public boolean randomSelect(){
    random=(int) Math.round(Math.random()*23);
    for(int i=0;i<redStateStore.size();i++){
        if(random==redStateStore.elementAt(i)){
            return false;
        }
    }
    for(int j=0;j<blueStateStore.size();j++){
        if(random==blueStateStore.elementAt(j)){
            return false;
        }
    }
    addStateBlue(random);
    return true;
}
```

In this method, firstly, a random number will be created. Then system will check if this number is a legal position number. If not, this method needs to be invoked again.

```
if(bo.checkMill(bo.getRedVector())==false){
    while(bo.randomSelect()==false){
    }
    leftBlue=leftBlue-1;
    board.set(leftRed,leftBlue);
    if(bo.checkMill(bo.getBlueVector())==true){
        while(bo.randomRemove()==false){

        }
    }
```

The first loop in the above code is to invoke the random select method until a legal position is generated and added to the corresponding vector. Second loop is about the mill, if there is a mill formed by computer player, system will remove one of the

human player's counters by using the randomRemove method, which is shown in the following.

```
public boolean randomRemove(){
    random=(int) Math.round(Math.random()*23);
    for(int i=0;i<redStateStore.size();i++){
        if(random==redStateStore.elementAt(i)){
            removeStateRed(random);
            return true;
        }
    }
    return false;
}
```

Same theory has been used as randomSelect() method. Firstly, generate a number randomly, then check if it is legal.

Like two methods above, in movement procedure there also has a randomMove() method and using the same theory.

```
public boolean randomMove(){
    random=(int) Math.round(Math.random()*23);
    for(int j=0;j<blueStateStore.size();j++){
        if(random==blueStateStore.elementAt(j)){
            for(int i=0; i<24;i++){
                if(detectIllegalMove(random,i)==true){
                    removeStateBlue(random);
                    addStateBlue(i);
                    return true;
                }
            }

        }
    }
    return false;
}
```

The full code lists and detail screen shot are in the appendix. If there is anything unclear about realization, please check the appendix.

Testing

Test case 1: Placement test
Expected result: Counters can be placed on board correctly.
Actual result: Same as expected result.

Test case 2: Movement test
Expected result: Counters can be moved on board without problem.
Actual result: Same as expected result.

Test case 3: Detect mill
Expected result: When there is a mill, the system will give a message window

Actual result: When one player has an old mill on board, sometimes the system will give a message window about the old mill.

Test case 4: Flying
Expected result: When one player only has three counters on board, the player can fly his/her counters.
Actual result: Same as expected result.

Test case 5: Wrong Placement
Expected result: a message window with corresponding text will jump out.
Actual result: Same as expected result.

Test case 6: Wrong movement
Expected result: a message window with corresponding text will jump out.
Actual result: Same as expected result.

Test case 7: Wrong remove
Expected result: a message window with corresponding text will jump out.
Actual result: Same as expected result.

Test case 8: Remove
Excepted result: When one player has a new mill, he/she can remove one of his/her opponent's counters
Actual result: Same as expected result.

Test case 9: Connection
Excepted result: Players can connect to the server without problem.
Actual result: Same as expected result.

Test case 10: Synchronization
Excepted result: The game screen of two dislocated players' should be synchronized.
Actual result: Same as expected result.

Test case 11: control in distribution mode
Excepted result: The system should control the actions of two players to check if it is a legal action.
Actual result: my software did not have control in this game mode.

Test case 12: one player mode
Excepted result: The system could run without problem in this game mode.
Actual result: Same as expected result.

Test case 13: hint for the end of placement
Excepted result: there will be a message window, when all counters have been placed.
Actual result: Same as expected result.

Test case 14: hint for the end of game
Excepted result: there will be a message window, when one of the players win the game.
Actual result: Same as expected result.

Test case 15: chat system
Excepted result: The chat system can be worked very well in distribution mode.
Actual result: Same as expected result.

The details of testing are in the appendix, where a lot of screen shots can be found.

Evaluation

My software was tested in every stage during the programming. Three main methods have been used.

One method of evaluation is that of how many expected features and desirable feathers which are mentioned in Specification document I have achieved at the end of the project. I finished all the expected features except two points. Firstly, the game can not be restarted at any time. If the player wants to start a new game, he/she needs to close the game first. Second point is that in one player mode, the player can not revoke on of his/her moves. On the other side, because of limited time, the entire desirable feathers have not achieved.

A second aspect is the feedback I received from my supervisor and marker, as well as of a few friends who have volunteered to try out the tool. The latter was based on the questionnaire. According to the result of the questionnaire, I did some changes to my program.

Thirdly, black box testing and white box testing methods will be used to test my game at every stage.

Black box testing

- It was used to test my game's GUI, to check the button event, mouse event and their function are appropriately. Expected result for this is that every button has the exact reaction when it is pressed. Actually, every button in the game has its own responsibility and works without errors.
- The connection is another thing need to test in this way. To verify the connection between dislocated computers is actually established. The excepted result is that the two players can use different computers to play the game without any problems. While the actually situation is that the game screen can be synchronized between two dislocated player, but less control in the game mode.

White box testing

- This mainly used to test the rules. To check if the system can detect illegal moves and illegal placement. What is more, system also has the responsibility to perceive the mill and the winner. The excepted result is that, the system can realize anything illegal, and then return the error message to player. Moreover, it can remind player to remove one of the opponent's counter when player has a mill. In addition, it can also finalize the game, while there is a winner, and then

congratulates to the winner. In the real life, the game passed this test, but there is a small bug in mill part.

I summarize the weaknesses and strengths of the game which will be presented later. These are collected from the feedbacks of supervisor, marker and my friends who tried this software.

Weakness

- The game designed in this project does not have a nice user interface. The background of the game window is just a single color, no other decoration.
- It is too simple in the menu bar. Just three game modes and a stop game option were designed. If there is enough time, I would like to add some more options like restart, revoke, help and etc…
- The game board and chess are simply structured. I used java to draw the board and print the chess. It should be better if I can find some board pictures and chess pictures on internet. Then, my game screen should be better.
- In one player mode, the artificial intelligence is extremely weak. In desirable software, artificial intelligence should be strong and has different levels.
- In distribution model, just the game screen has been synchronized, no control between two players. It means if there is only one player in the room, he/she can also start the game and place the counters.
- In game rules side, there exists a bug in mill part, which still can not be found.

Strength

- During the game, when one of the players did something illegal, there will jump out a message window with corresponding message to warn the player.
- There is a status bar under the game board. The message on it will keep updating, according to the events on game board.
- A chat system is designed. In distribution mode, two players in different places can use such system to chat with each other.
- Game rules are programmed without huge error. The game can aware when the placement procedure is end and give a message window. Moreover, system can detect if a new mill was formed, if yes, system will go to mill procedure automatically. Furthermore, system can realize when the players can fly their counters.
- The game can be played fluently by two people in the same machine.
- The game can be played by two people in dislocated machine.
- The game can also be played by one people, to play against the computer.

Despite there are a number of weaknesses, it can not be described as a failing project. My software is best evidence. It can be successfully worked in each game mode. With a little amount of time, it is certain that a perfect distribution and artificial intelligence could be implemented.

Learning point

General Learning Point

I have never handled a large project such as this by myself before. Therefore, a large amount of problems have been encountered, through which a lot of skills and the number of knowledge has been acquired.

In this project, three official documents need to be handed in. Totally, about fifteen thousand words have been produced. And this practiced my writing skill. How to produce a nice report is a useful skill that I learned in this project.

My project is a problem solving project. Thus, skill in planning and researching is another important aspect that I learned. Through the project, I know how to plan a whole project and how to have a reasonable time management. Besides, in every stages of the project, I need to do the research including the background and the programming problems. Currently, I have a great deal of experience on research.

Nine Men's Morris Game

Because my project aim is to develop a Nine Men's Morris game, I did research on this game for several months. In order to be familiar with the rules, I played this game online for several times. Until now, I have at least four different strategies which can be used in the game.

Software engineering

I studied software engineering for two semesters. Enough theories has been equipped with my, and this project provides me a chance to practice what I learnt in the class in the real life. In this project, I used the object-oriented design method for the first time. Before, I know how to draw a sequence diagram and user case diagram, but I have never used it in a real project. After the valuable experience of this project, I am much more familiar with software engineering.

Java

As the main programming language, Java learning is an extremely significant point in the whole project. Although I have get in touch with Java for three years, there still exist a lot of things that I do not know. To do this project, the number of source code was downloaded from the internet. I run the source code on my computer, so I can know how the code working, then use it in my own project.

How to make a distribution is the most important part that I learnt through this project. It took me totally a week to find an effectiveness way to synchronize the game screen of two dislocated computer. About 1000 lines of code were generated for this part. I have confidence that if there is enough time, there will be no problem with the distribution.

Professional issue

The British Computer Society is a society that represents the practitioners in United Kingdom and internationally.

Code of Practice

The code of Practice of the British Computer Society is a series of rules established to guarantee that Information System (IS) Practitioners constantly preserve a high standard of practice. It states that Information System Practitioners shall seek to upgrade their professional knowledge and skill. This project is a great challenge and chance for me. Through this project, the number of knowledge and a lot of skills has been upgraded, such as presentation skill, writing skill, and knowledge about distribution and artificial intelligence.

Code of Conduct

The code of conduct is a set of professional standards required by British Computer Society. These standards are used as a condition of membership. There are four section and totally seventeen points in the code of conduct. I will illustrate how my software relates to these sections.

The Public Interest

The relevant authority is in my project is the University of Liverpool. I believe that my project complies with the rules of University and Computer Science Department. My software is just a board chess game. Obviously, it has no danger to the public health, safety and environment. Every people can play the software, there is no message in my software with discrimination against clients and colleagues.

Duty to Relevant Authority

I do not think I will have a conflict of interest with the university. My software is just a small game. No profit can be got from it. I will do my best to follow the rules in this section.

Duty to the Profession

As a Computer Science student, I do the best to upload the reputation and good standing of the BCS in particular. Through this project, my professional standard is obviously improved. Furthermore, during the project, I am certain that I act with integrity in my relationship with my supervisor, my second marker and my friends. Moreover, all the materials that used in this report are listed in the reference.

Professional Competence and Integrity

In order to finish the project, I need to keep upgrading my professional knowledge and skill. If I just depend on the knowledge learnt from the classes and text books, the project will not be completed. I have produced this software in my level of competence. I can explain everything in my code, which you have already seen from the realization part. While, to conclude, I can take full responsibility of the software that has been produced in this project.

Bibliography:

Background research:

Websites:

- http://en.wikipedia.org/wiki/Nine_Men%27s_Morris
- http://library.msri.org/books/Book29/files/gasser.pdf
- http://merrelles.com/English.html

Software Design:

Books:

- Ian Sommerville. Software engineering, Addison Wesley, (2004)
- Stevens, Perdita. Using UML : software engineering with objects and components, Addison-Wesley, (2006)
- Bennett, Simon. Schaum's outline of UML, McGraw-Hill, (2005)
- Alhir, Sinan Si. Learning UML, O'Reilly, (2003)
- Robert Adam Lovelock. Honours Project, University of Liverpool, (2001)
- Stephen Devereux. Project-Software solution to Nine Mans Morris, University of Liverpool, (2001)
- Ryan Price. Third Year Honour Project, University of Liverpool,(2001)

Software Implementation:

Books:

- Morelli, Ralph. Java, Java, Java : object-oriented problem solving, Pearson/Prentice Hall, (2006)
- Boger, Marko. Java in distributed systems : concurrency, distribution, and persistence, Wiley,(2001)
- Daum, Berthold. Professional Eclipse 3 for Java developers, Wiley, (2005)
- Van der Linden, Peter. Just Java 2, Sun Microsystems Press, (2004)
- Kurose, James F.. Computer Networking: A Top-Down Approach, Pearson, (2010)
- Tanenbaum, Andrew Stuart. Distributed systems : principles and paradigms, Prentice Hall, (2008)
- Callan, Robert. Artificial intelligence, Palgrave Macmillan, (2003)
- Rothlauf, Franz. Design of Modern Heuristics : Principles and Application, Springer-Verlag Berlin Heidelberg, (2011)
- Topham, Alivia. Heuristics for move selection, University of Liverpool, (2003)

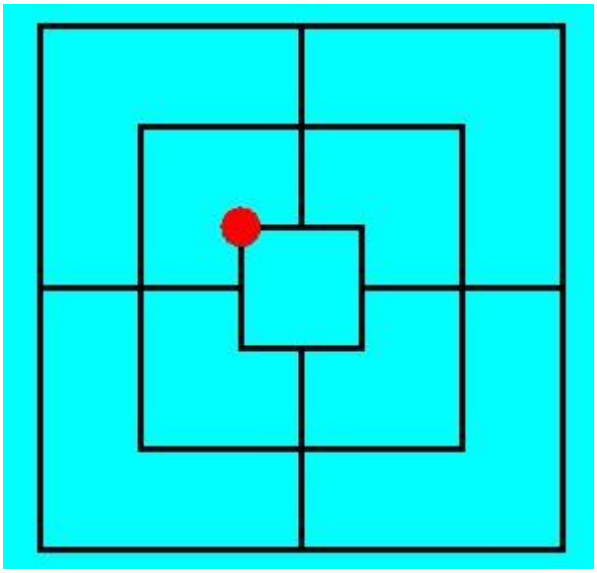
Appendix

Code list

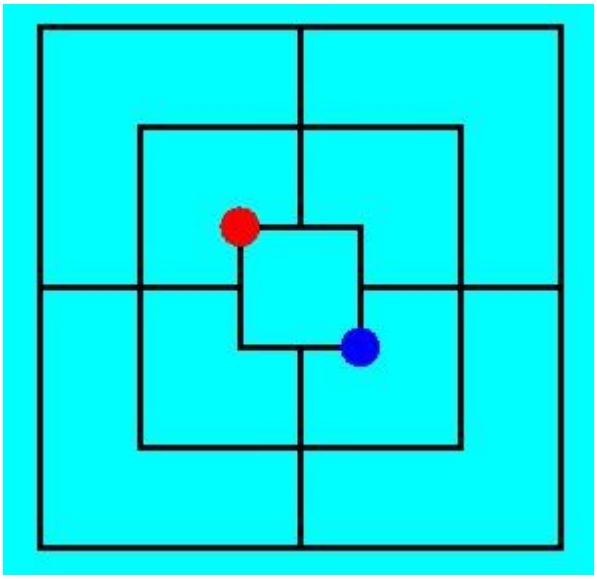
Totally 16 files and over 3000 lines of code were generated. It is too large to paste in this document. So, all the codes are in the disc.

Detail of test data

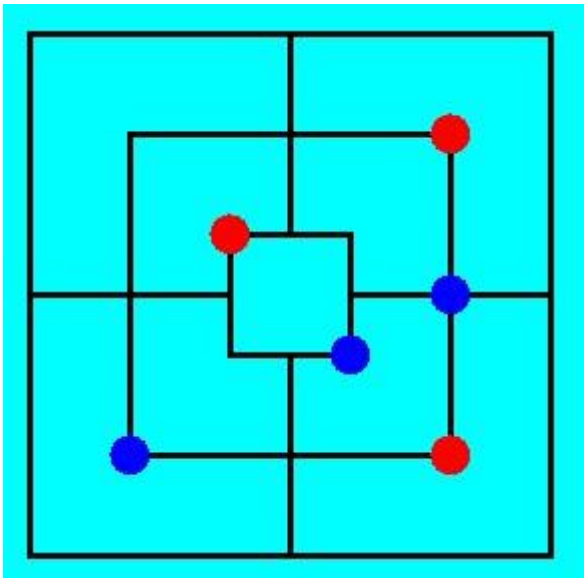
Test case 1:



```
red is [0]
blue is []
```

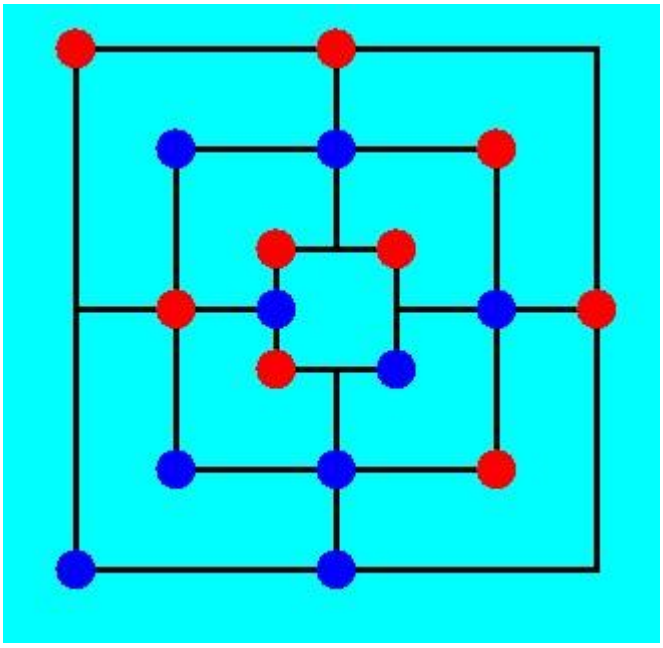


```
red is [0]
blue is [4]
```

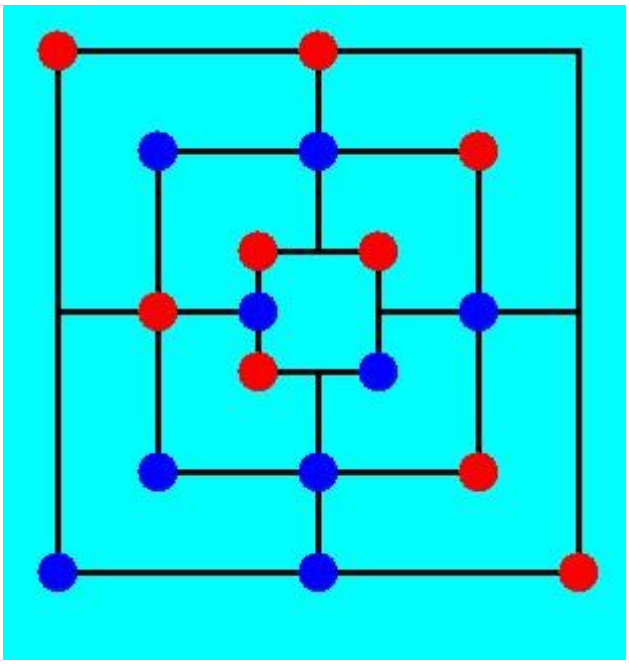


```
red is [0, 10, 12]
blue is [4, 14, 11]
```

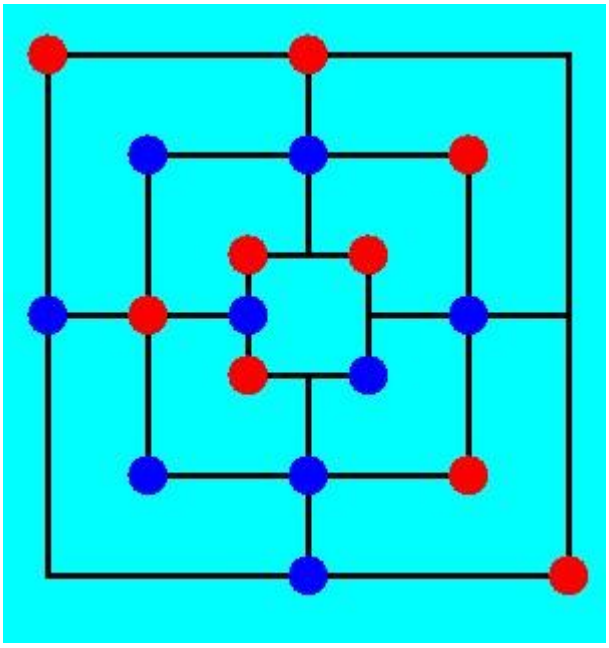
Test case 2:



```
red is [0, 10, 12, 17, 19, 16, 15, 6, 2]
blue is [4, 14, 8, 21, 11, 22, 13, 7, 9]
```

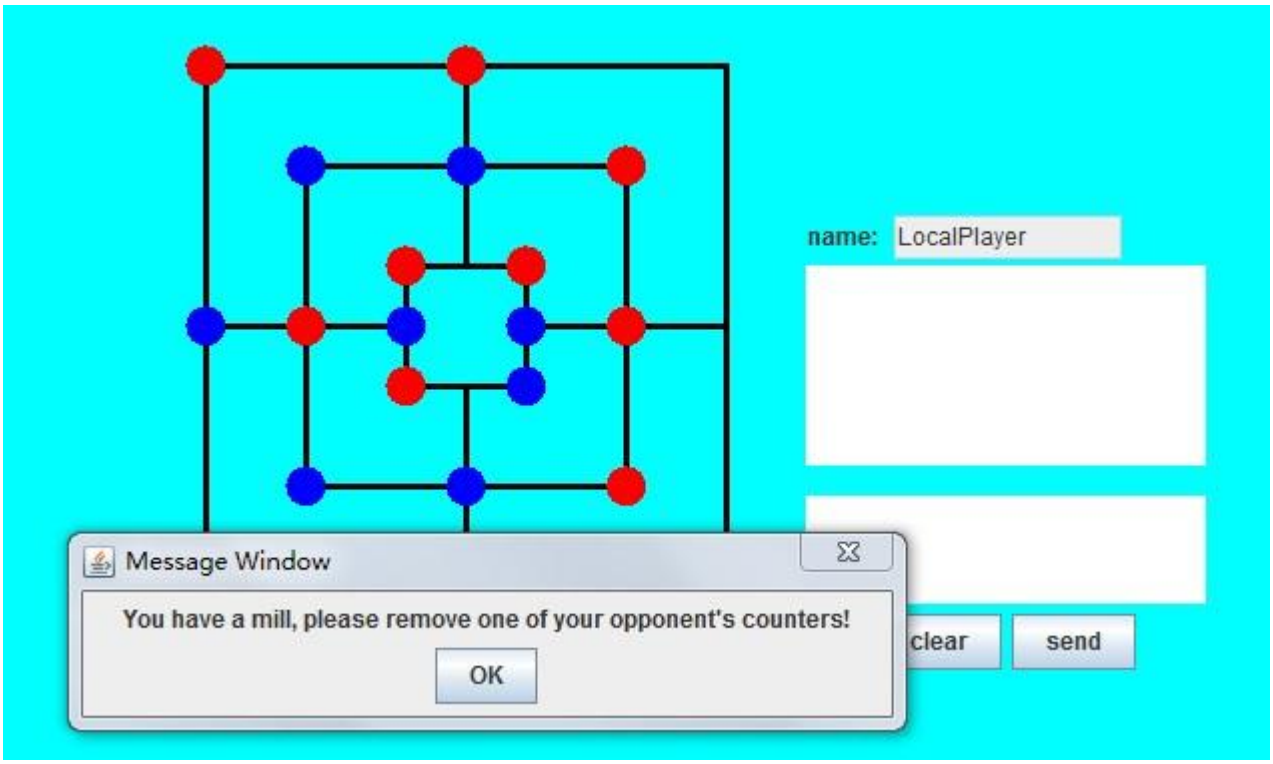


```
red is [0, 10, 12, 17, 16, 15, 6, 2, 20]
blue is [4, 14, 8, 21, 11, 22, 13, 7, 9]
```



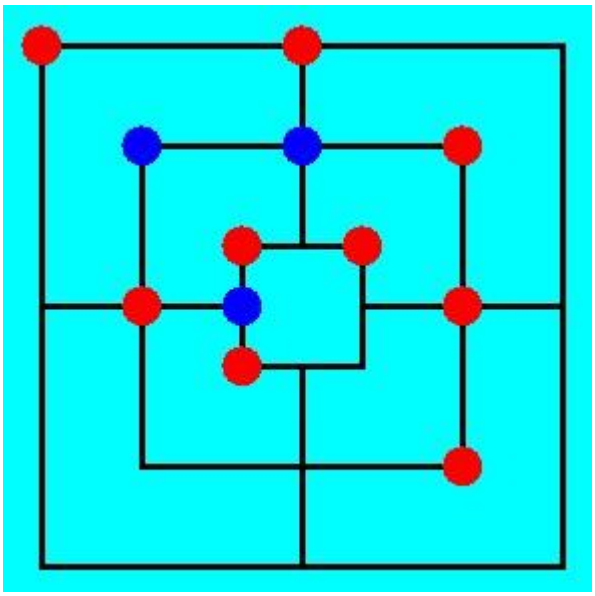
```
red is [0, 10, 12, 17, 16, 15, 6, 2, 20]
blue is [4, 14, 8, 21, 11, 13, 7, 9, 23]
```

Test case 3:

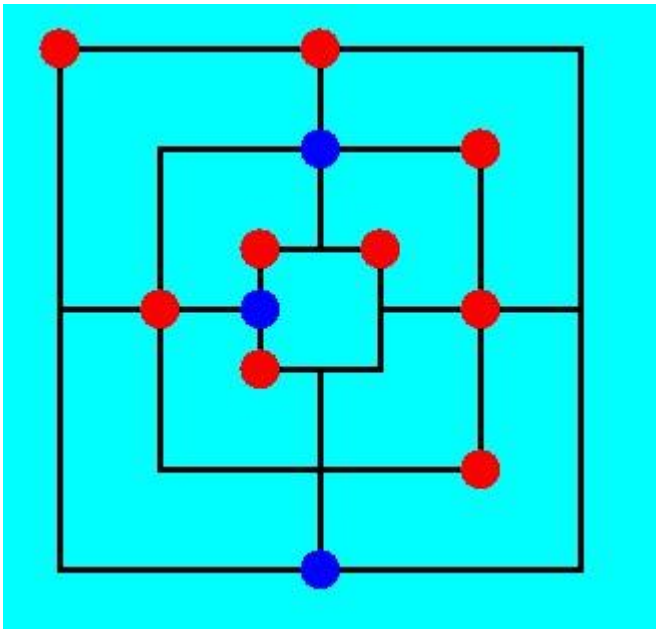


```
red is [0, 10, 12, 17, 16, 15, 6, 2, 11]
blue is [4, 14, 8, 21, 13, 7, 9, 23, 3]
```

Test case 4



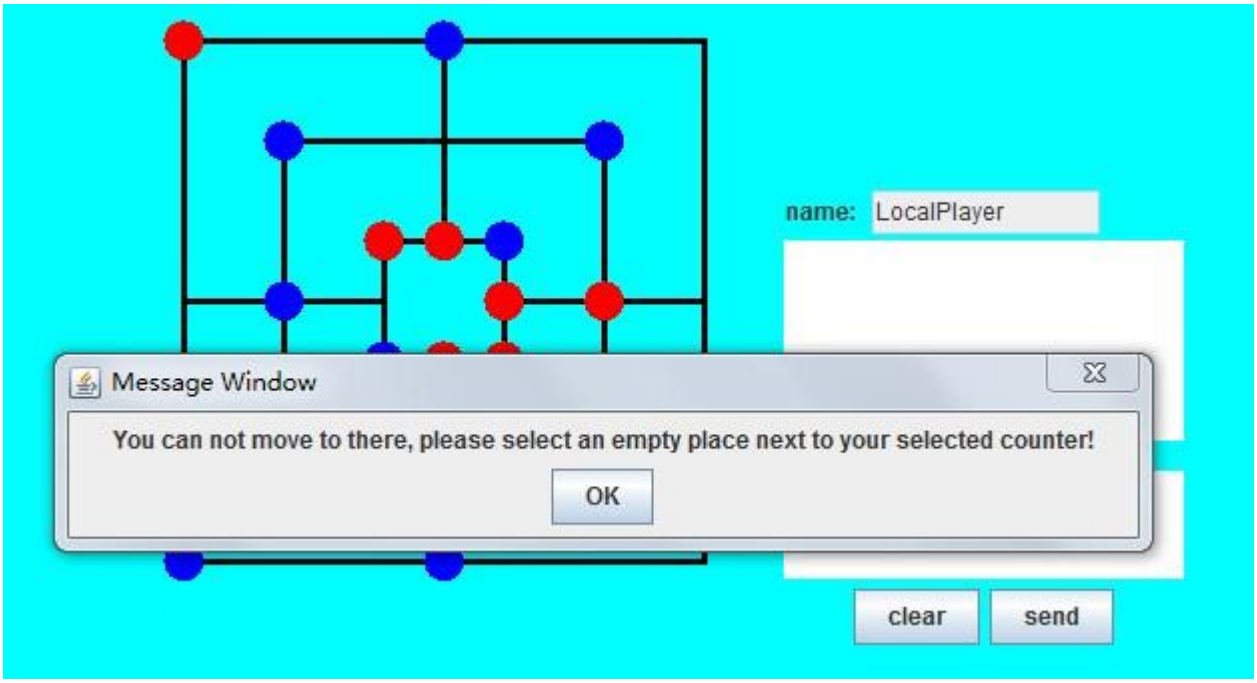
```
red is [0, 10, 12, 17, 16, 15, 6, 2, 11]
blue is [8, 7, 9, 5]
```

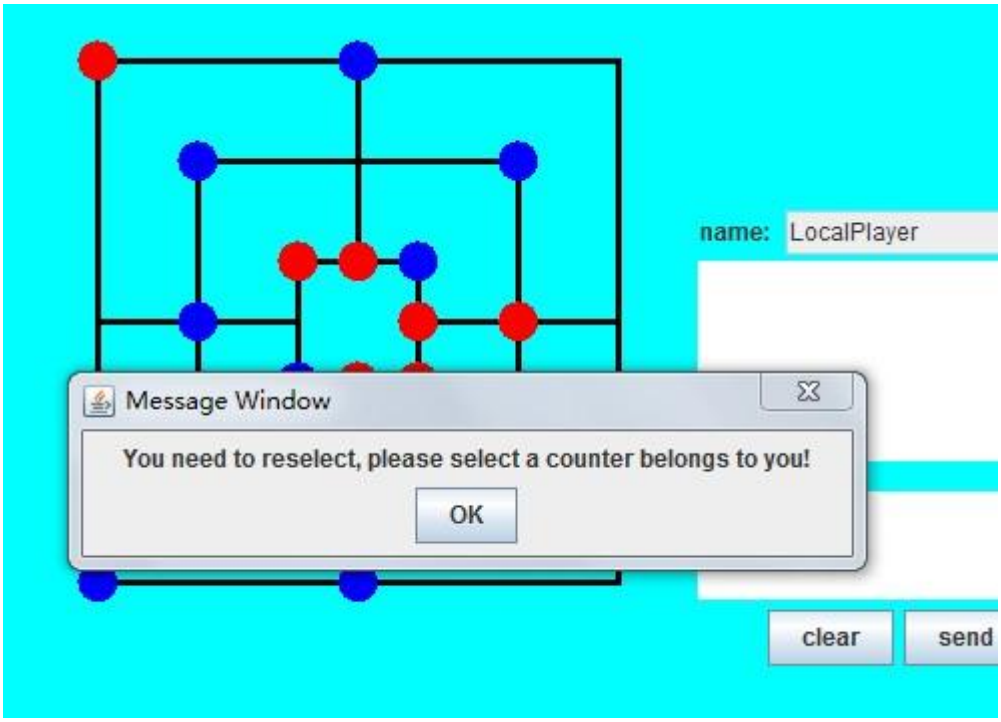


Test case 5:

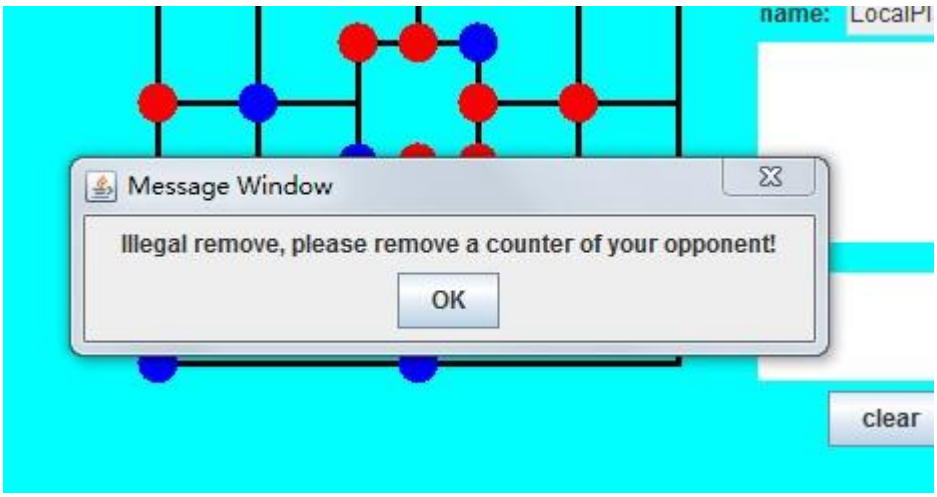

name:
Message Window
Illegal Placement! Place again!
OK

Test case 6:

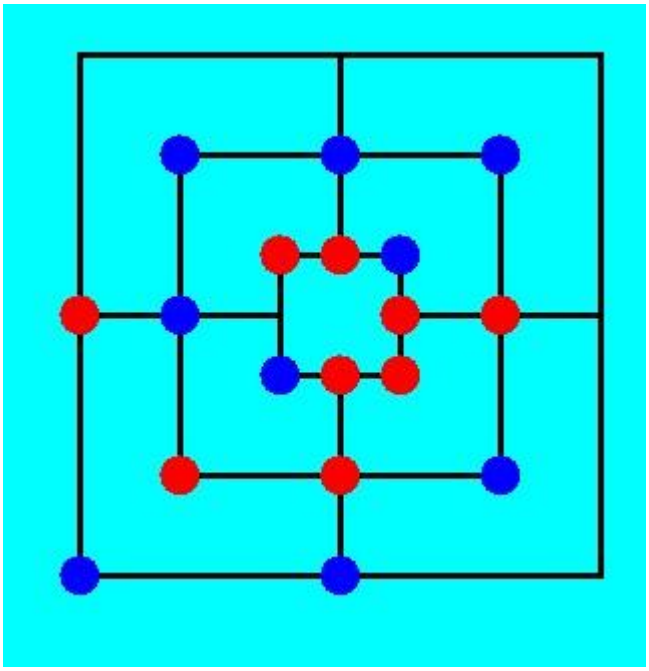

name: LocalPlayer
Message Window
You can not move to there, please select an empty place next to your selected counter!
OK
clear
send

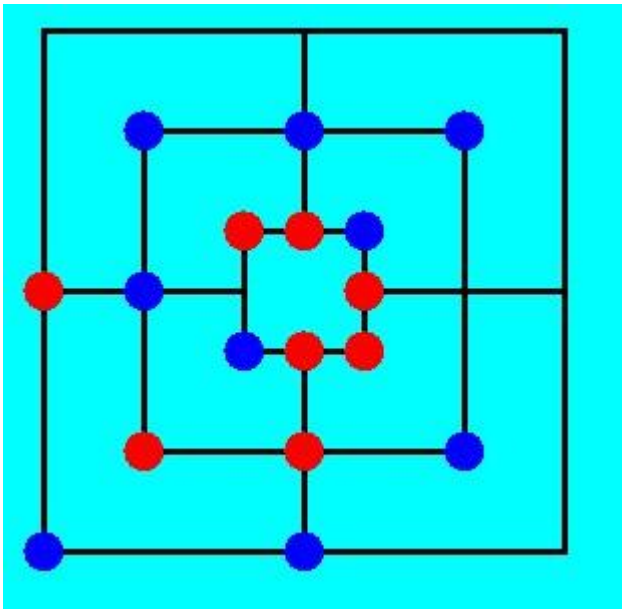


Test case 7:

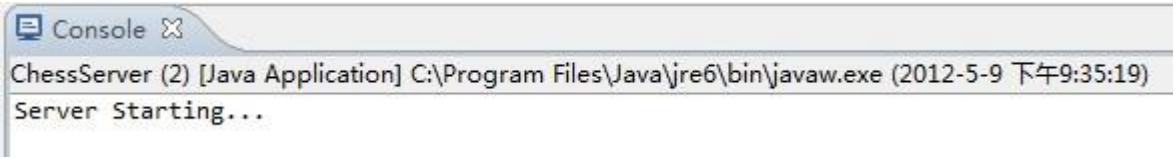


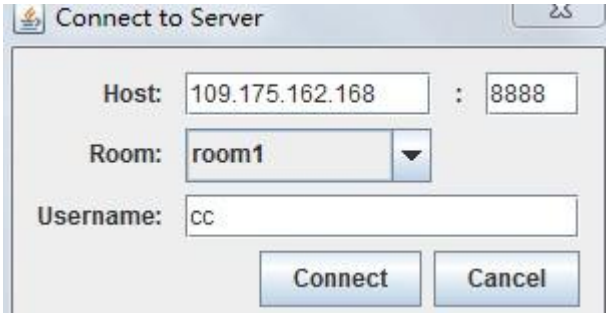
Test case 8

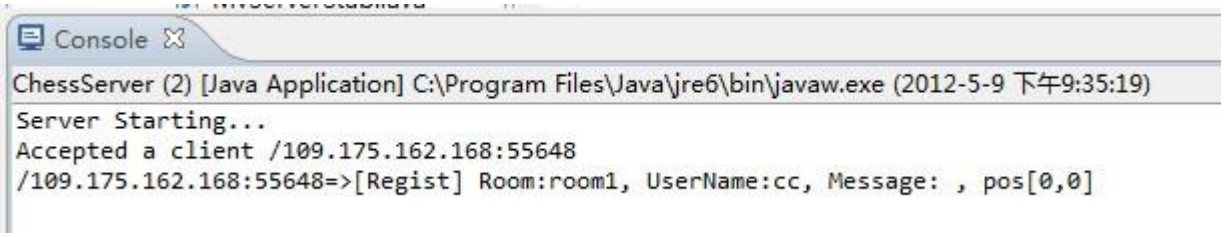




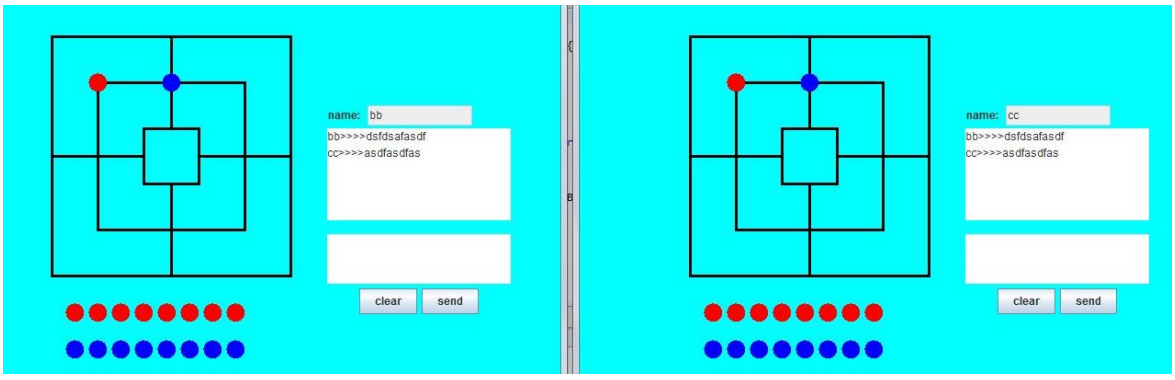
Test case 9:


Console
ChessServer (2) [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (2012-5-9 下午9:35:19)
Server Starting...





Test case 10& Test case 15

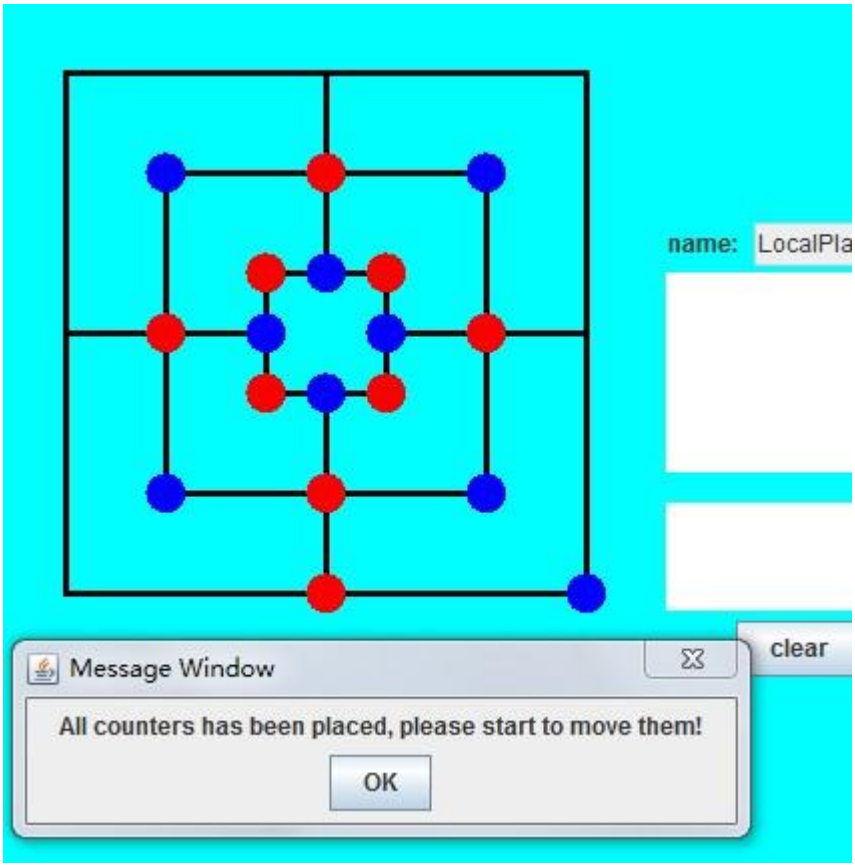



```
Accepted a client /109.175.162.168:55661
/109.175.162.168:55661=>[Regist] Room:room1, UserName:cc, Message: , pos[0,0]
Accepted a client /127.0.0.1:55662
/127.0.0.1:55662=>[Regist] Room:room1, UserName:bb, Message: , pos[0,0]
/127.0.0.1:55662=>[Play] Room:room1, UserName:bb, Message: , pos[65,61]
Play: [Play] Room:room1, UserName:bb, Message: , pos[65,61]
/109.175.162.168:55661=>[Play] Room:room1, UserName:cc, Message: , pos[140,59]
Play: [Play] Room:room1, UserName:cc, Message: , pos[140,59]
/127.0.0.1:55662=>[Message] Room:room1, UserName:bb, Message: dsfdsafasdf, pos[0,0]
/109.175.162.168:55661=>[Message] Room:room1, UserName:cc, Message: asdfasdfas, pos[0,0]
```

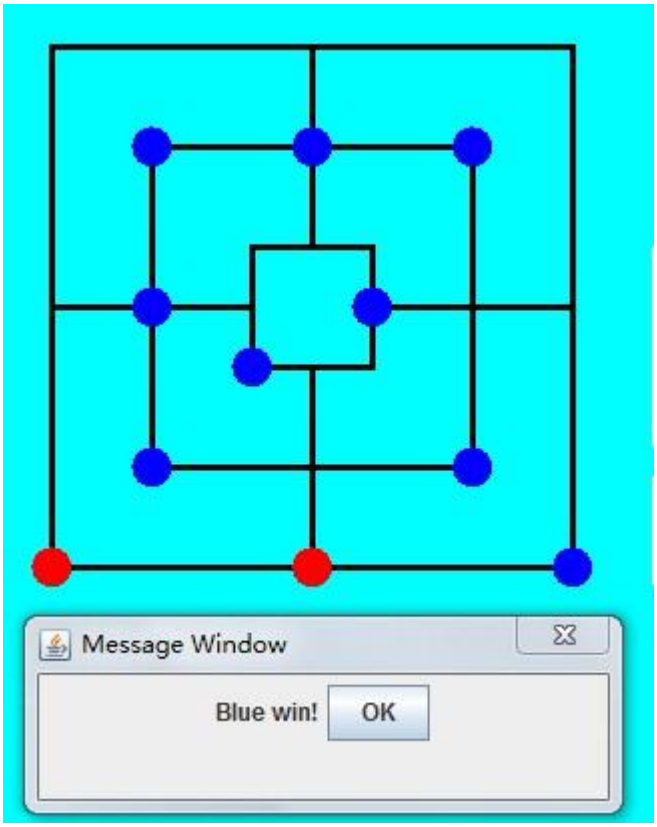
Test case 11: No successful test

Test case 12: Please run the program yourself

Test case 13:


name:
LocalPla
clear
Message Window
All counters has been placed, please start to move them!
OK

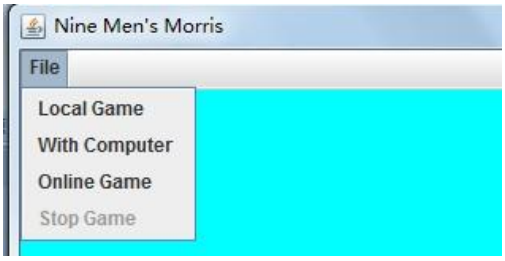
Test case 14:


Message Window
Blue win!
OK

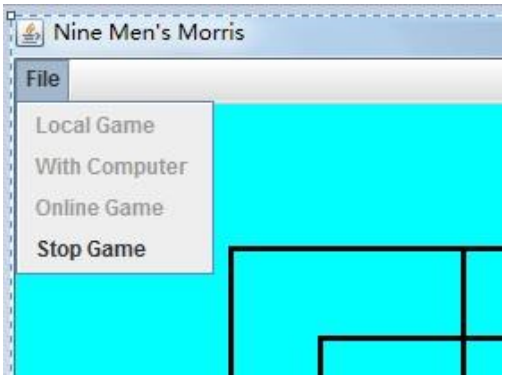
User guide

Local game mode and One player mode

If you want to play local game or play with the computer. You just need to run the BoardClient.java file in command window or in the eclipse. Then the main game screen will be presented. You should click the menu bar to choose the game mode.

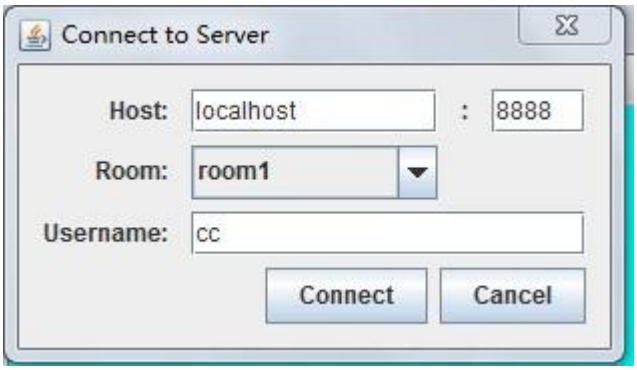


After you entered the game, when you want to stop game, you need to click the menu bar again to and choose the stop game option.

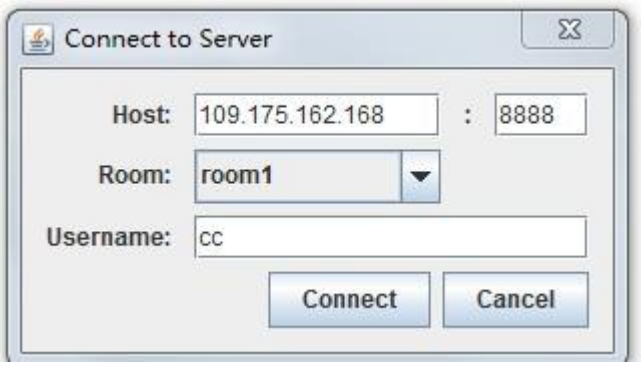


Distribution mode

If you want to play in this mode, you need to run the chessServer.java file first, which is in the board.server package. When the server has been started, you run the boardClient.java file, and then click the online game option in the menu bar. After that, a connect dialog will be presented.



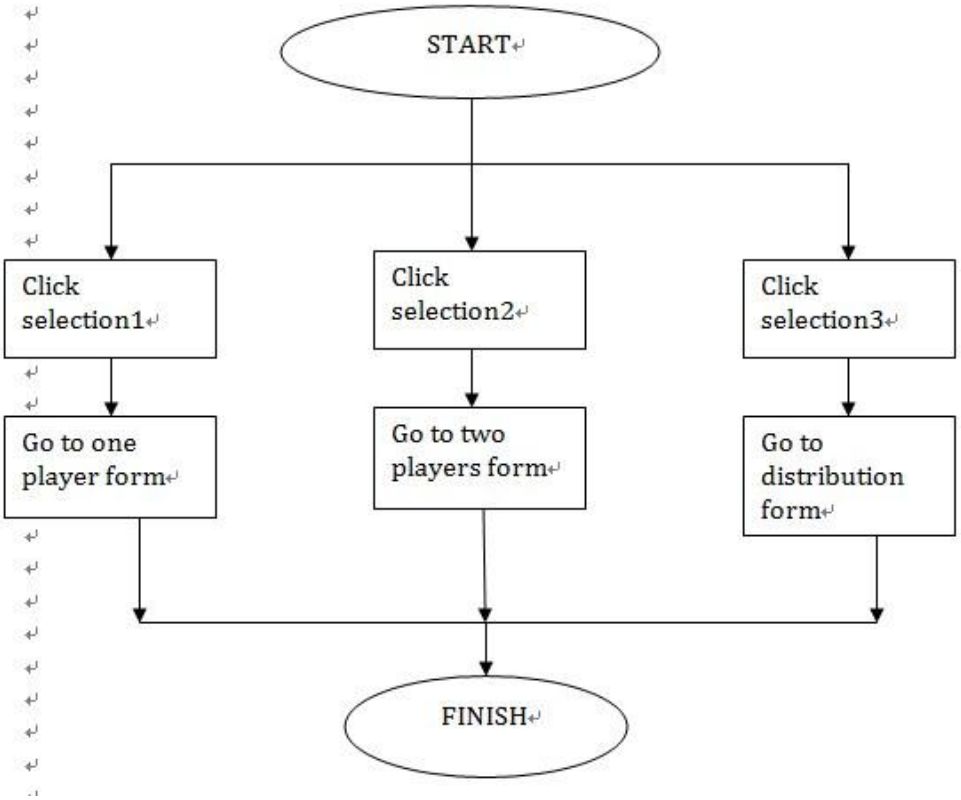
If you just want to test the game on the same machine, no need to change anything that presented on this dialog, just input the username. If you want to connect to another machine, you need to type the IP address of the server machine and the port number. In this first text filed. It can be learnt from following picture.

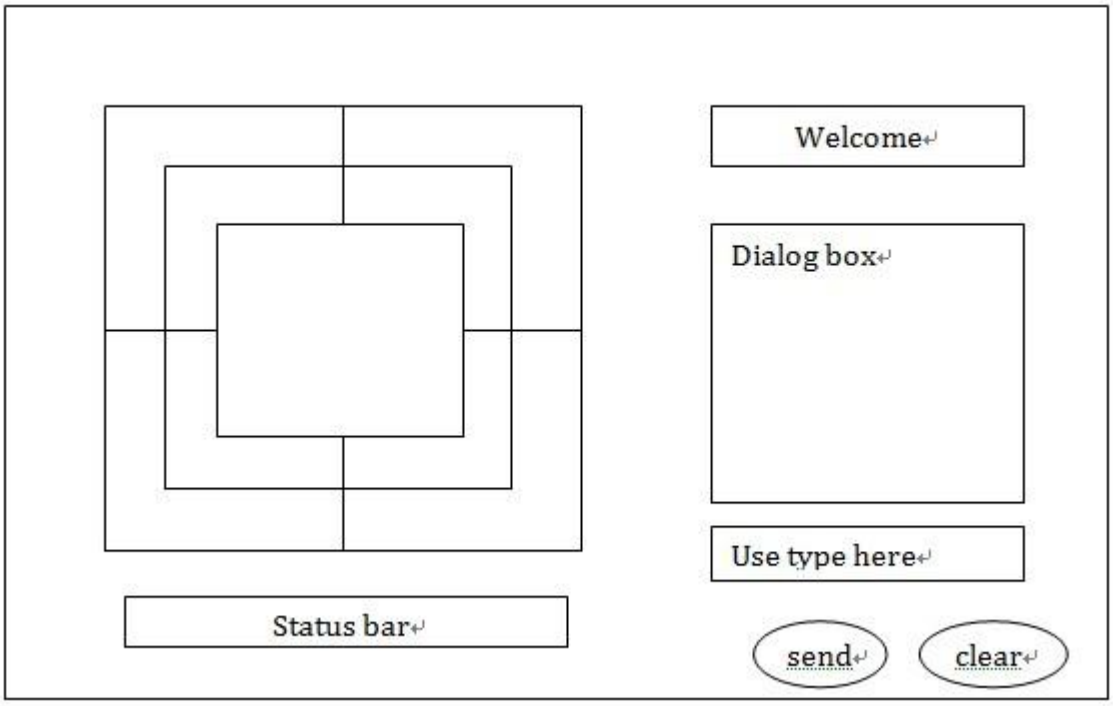


Then you should click the connect button. When there are two players in the room, you can play the game.

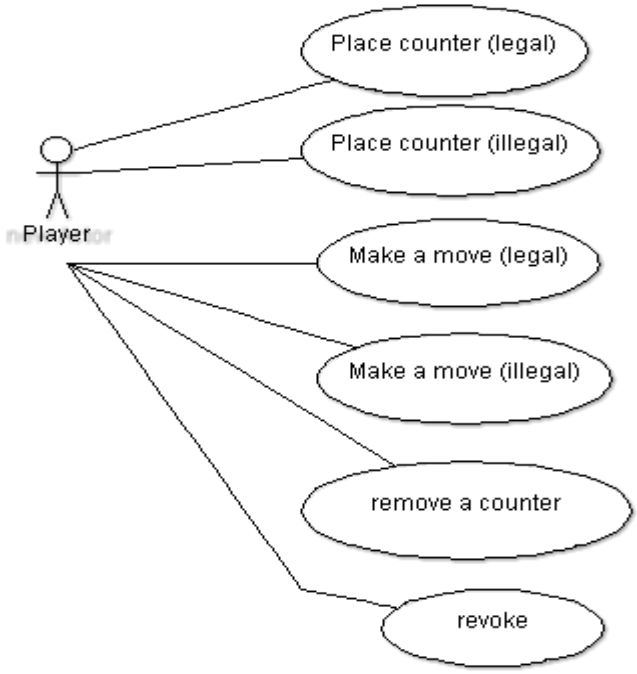
Full design diagram

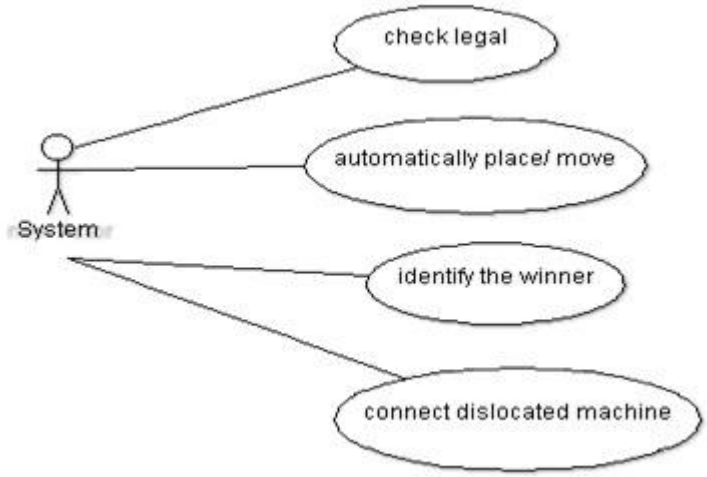
Game screen



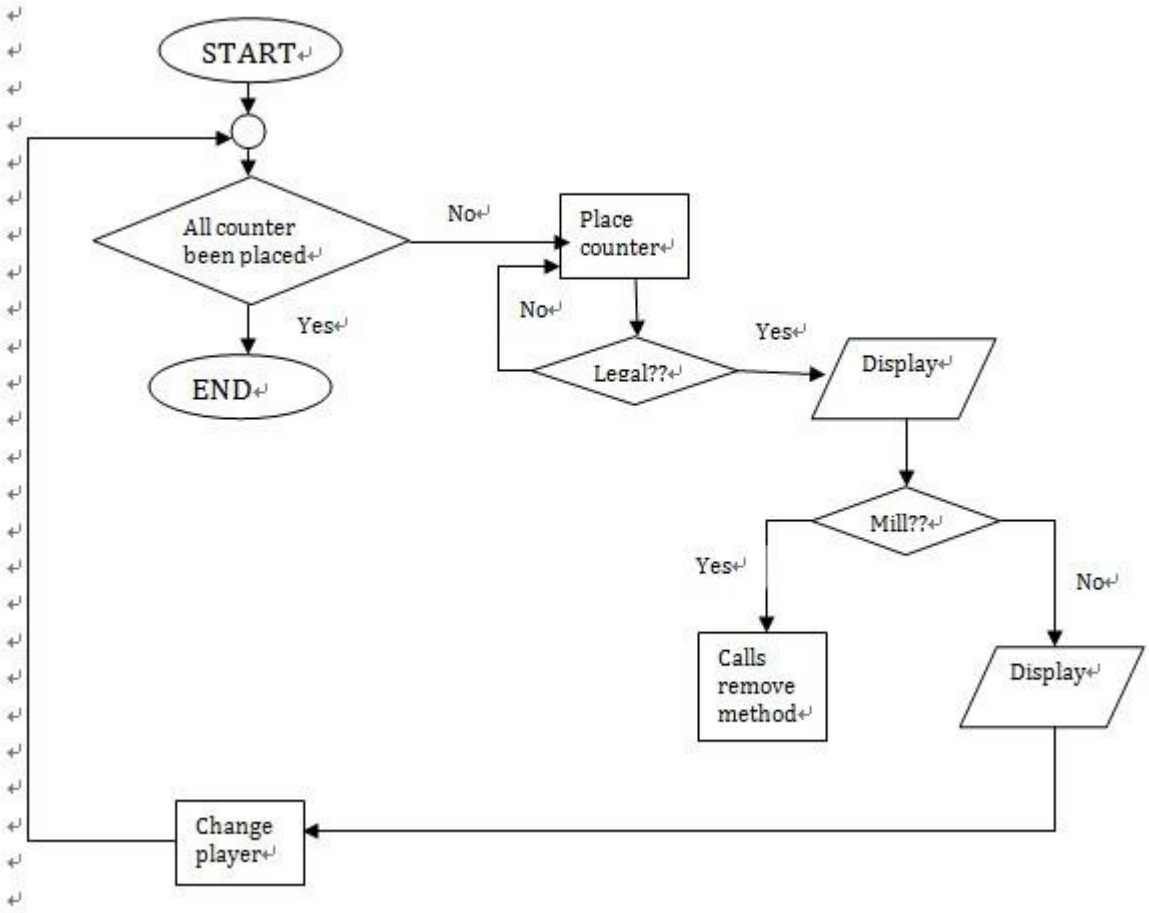

Welcome
Dialog box
Use type here
Status bar
send
clear

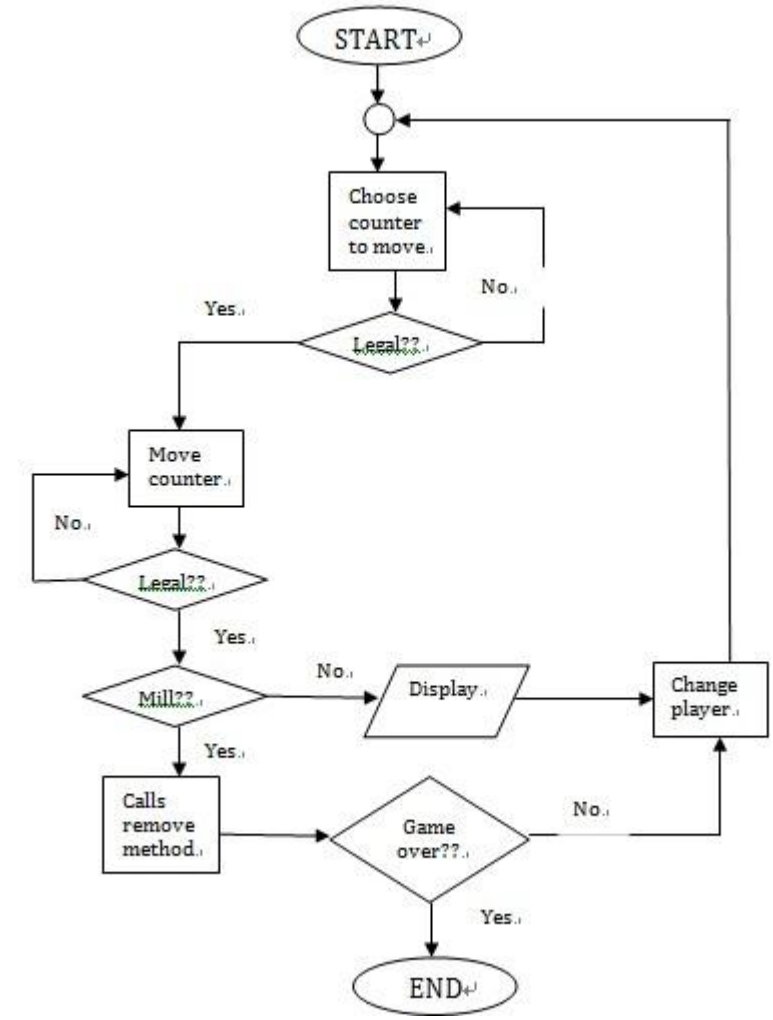
Use Case View

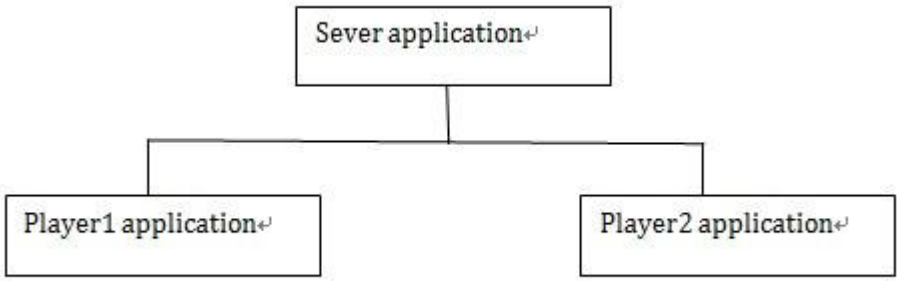


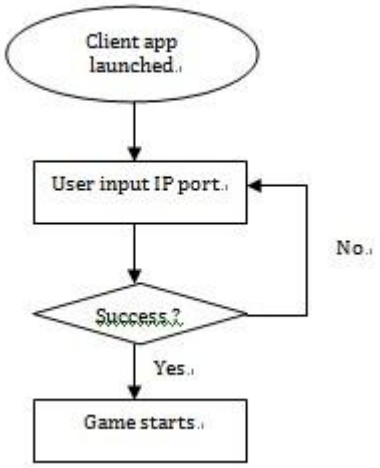


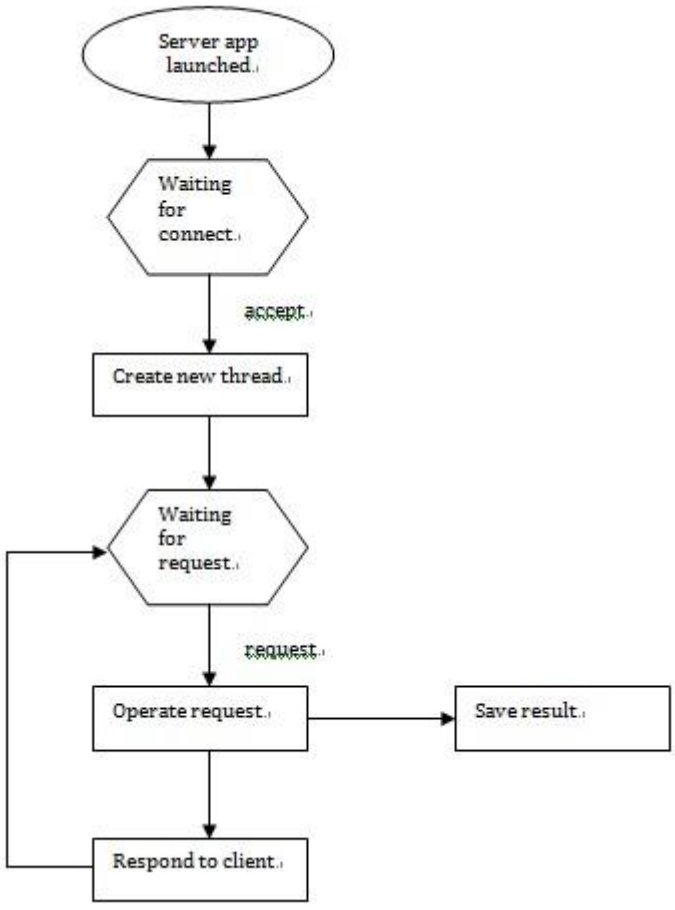
Flow chart


START
All counter been placed
No
Place counter
Yes
END
No
Legal??
Yes
Display
Mill??
Yes
No
Calls remove method
Display
Change player

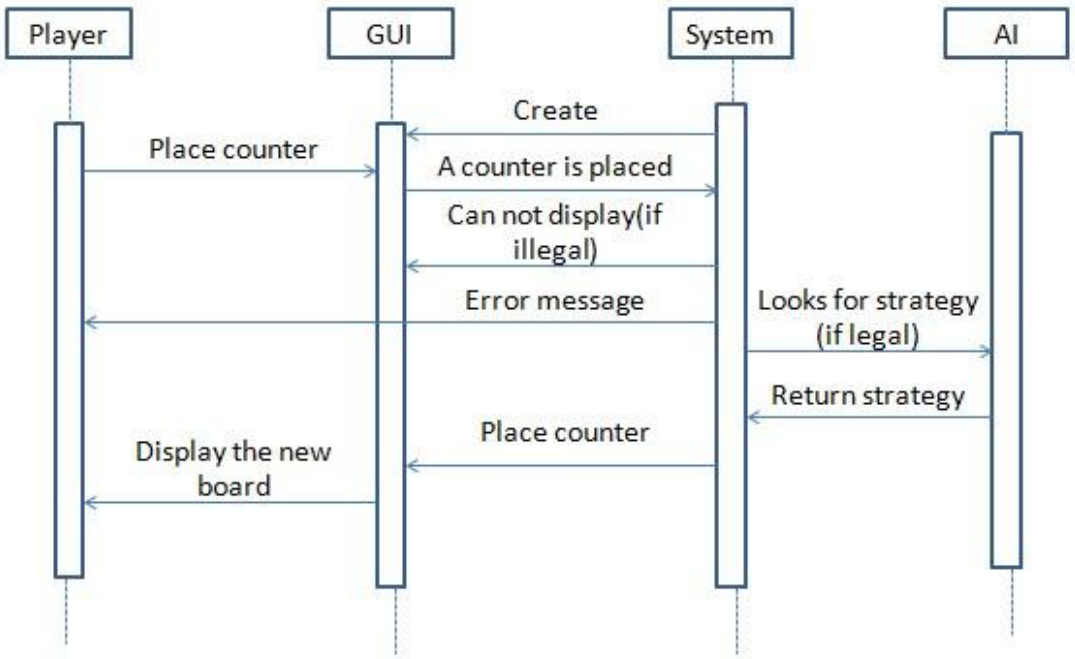

START
Choose counter to move.
No.
Yes.
Legal??
Move counter.
No.
Legal??
Yes.
Mill??
No.
Display.
Change player.
Yes.
Calls remove method.
Game over??
No.
Yes.
END

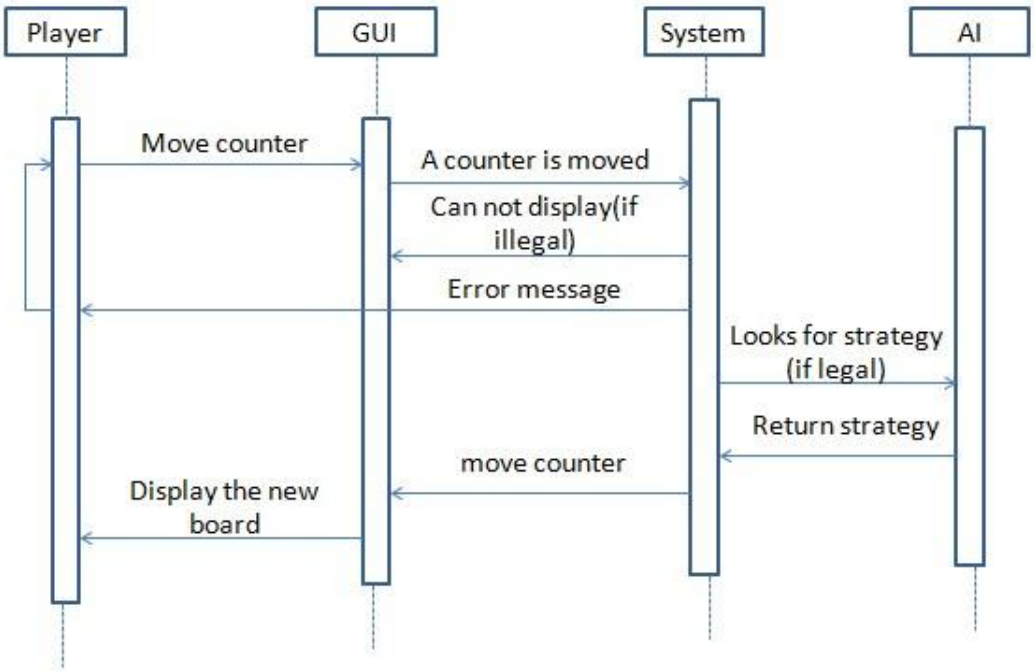

Sever application
Player1 application
Player2 application

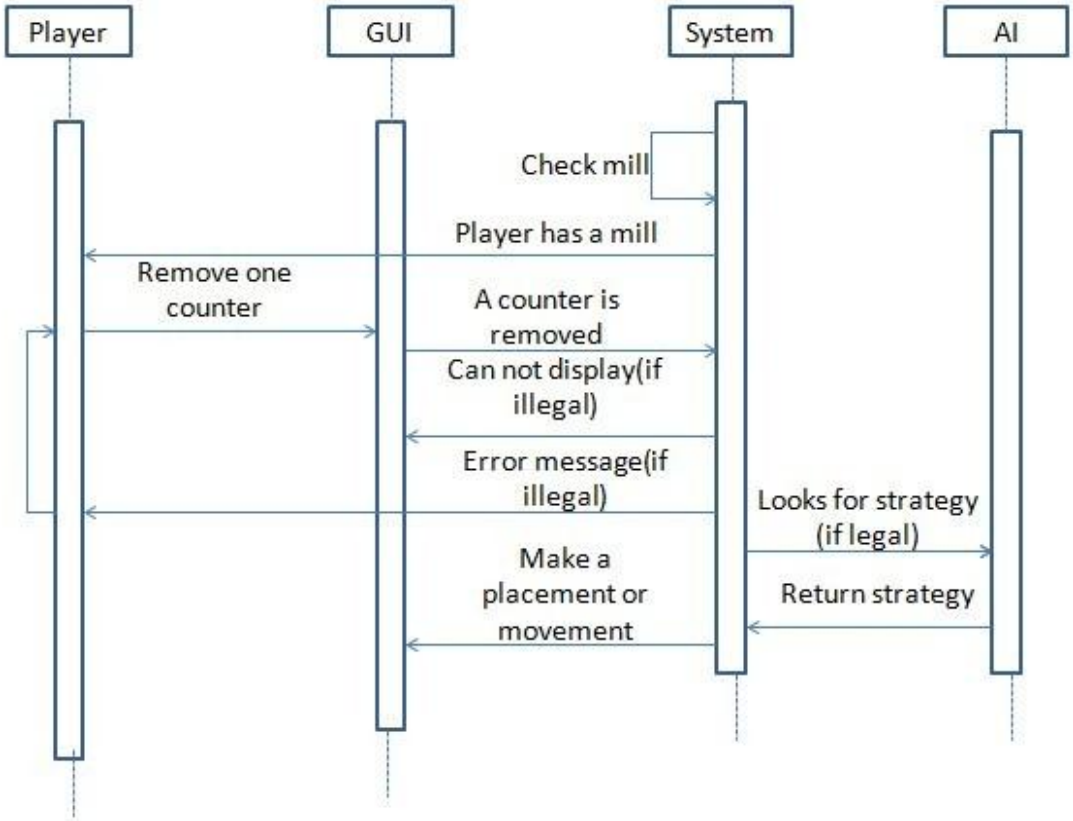

Client app launched
User input IP port
No
Success ?
Yes
Game starts

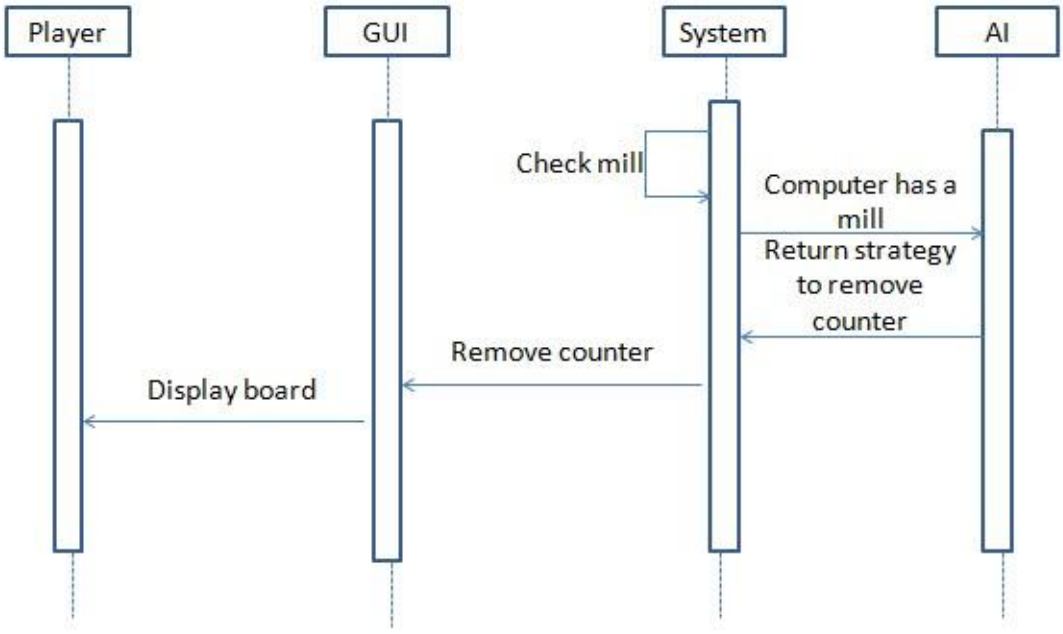

Server app launched
Waiting for connect
accept
Create new thread
Waiting for request
request
Operate request
Save result
Respond to client

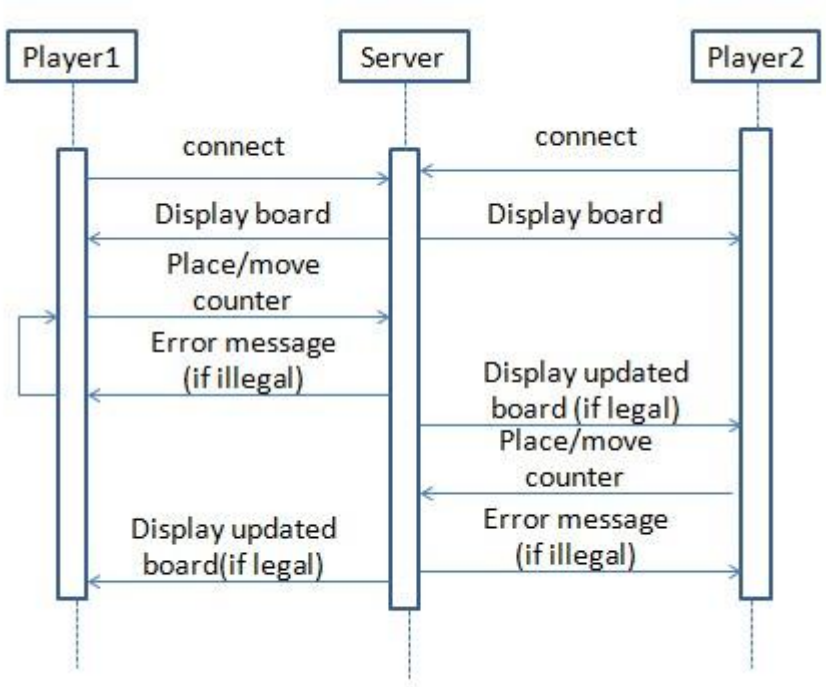
Sequence diagram

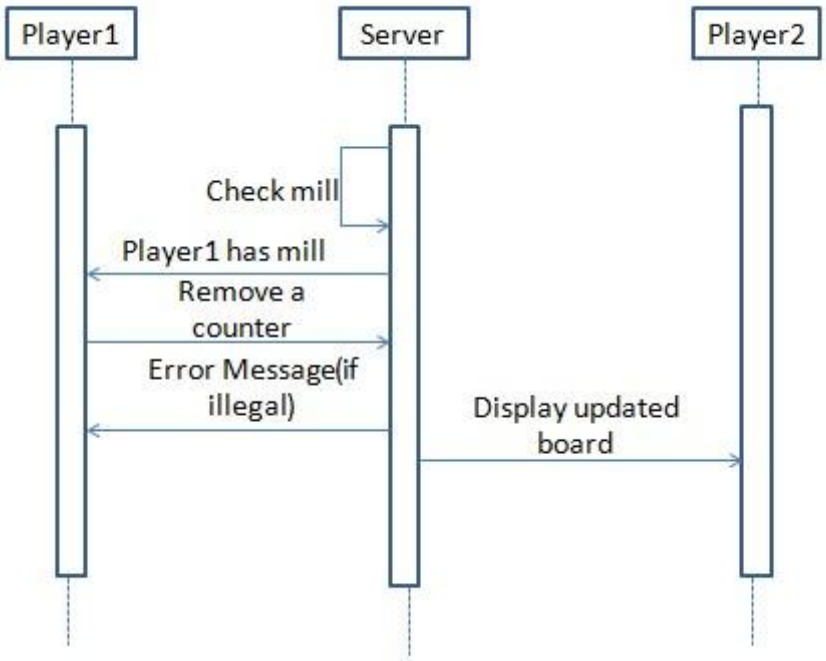





Player
GUI
System
AI
Check mill
Player has a mill
Remove one counter
A counter is removed
Can not display(if illegal)
Error message(if illegal)
Looks for strategy (if legal)
Make a placement or movement
Return strategy


Player
GUI
System
AI
Check mill
Computer has a mill
Return strategy to remove counter
Remove counter
Display board


Player1
Server
Player2
connect
connect
Display board
Display board
Place/move counter
Error message (if illegal)
Display updated board (if legal)
Place/move counter
Error message (if illegal)
Display updated board(if legal)


Player1
Server
Player2
Check mill
Player1 has mill
Remove a counter
Error Message(if illegal)
Display updated board