Report of Four In a Row Project

Qiao Ren s1618067 software system

Part A Discussion of the Overall Design

1 & 2. Class diagrams and systematic overview
I made two java projects:
-TUI version Four In A Row: networked, but did not use observer pattern
-GUI version Four In A Row: un-networked, but used observer pattern
The reason is that I made a MVC first. I thought the structure of GUI is very clean. So I gave up my original TUI view and changed it to GUI. My next step is to implement the TCP protocol. I realize that it is easier to use TUI view for a TCP protocol. Because the protocol is complicated for me. Then I change the view back to TUI. Since at that moment, I did not have enough time to repeat the observer process, I kept both versions.

In general, the TUI-version project works as expected. It satisfies nearly all the protocol requirement, except for disconnection. What can be improved is the structure of the client's view. Another improvement could be that when the server cannot accept the client's input port number, it is better to show an error message. The GUI version project provides a good view pattern.

The systematic overview of TUI-version project can be seen in the attachment A. This flow chart is a good summary of my work. Client has a view (class ThreadOfServer and class ThreadOfClient) and controllers. Server has a view and controllers. Both the client and the server share the class "Controller". The controller communicates with the model, the server view and the client view. The systematic overview of GUI-version project can be seen in the attachment B. The attachment C shows the three views of the GUI-version project.

3. The use of the *Observer* and *Model-View-Controller* patterns. In the TUI-version project, view observes the model. The

model extends observable. The view implements observer. In the controller, model.addObserver (view). When the model takes any actions, the view should be informed. In both of my GUI project and TUI project, the controller communicates with both the view and the model. The model is responsible for making a movement, remembering the current model, checking the game status. The controller gets user’s input, checks whether the input is valid or not. Then it tells model what to do with the input. After that, it tells the view what messages should be shown. The view is responsible for showing players messages.

From a theoretical perspective, even though the view is changed, the controller and the model should not be influenced.

4. Formats for data storage and communication protocols.
Data storage:
Mode: means which kind of game

Mode=0 means PtoP: person vs person	or human vs human
Mode=1 means PtoE: person vs environment	or human vs AI
Mode=2 means EtoE: environemnt vs environment	or AI vs AI

Players in the following array lists are registered, but there are no new game for them. They are not ready to start a game.

humanVsHumanName :	ArrayList<String>() stores the incoming players’ names, for players who chose human vs human mode
humanVsAiName :	ArrayList<String>() stores the incoming players’ names, for players who chose human vs ai mode
aiVsAiName :	ArrayList<String>() stores the incoming players’ names, for players who chose ai vs ai mode
humanVsHumanSocket :	ArrayList<Socket>() stores the incoming players’ sockets, for players who chose human vs human mode
humanVsAiSocket :	ArrayList<Socket>() stores the incoming players’ sockets, for players who chose human vs ai mode
aiVsAiSocket :	ArrayList<Socket>() stores the incoming players’ sockets, for players who chose ai vs ai mode

Players in the “waitingPlayers” array lists are registered, not playing the game. They are abandoned by their opponent during the previous game. Because their opponents exit the game. They choose to wait for another opponent to start a new game.

waitingPlayers :	ArrayList<Player>() stores the incoming players’ name, for players who chose human vs human mode

Players in “playerList” array are ready to start a new game.
playerList: Player[2] an array of players stored in classes StarterHumanVsHuman, StarterHumanVsAi, StarterAiVsAi.

Players in following arrays are playing the game.
pToPArrayInterface : PlayerInterface[2]; an array of players stored in the model, for human vs human mode.
pToEArrayInterface : PlayerInterface[2]; an array of players stored in the model, for human vs AI mode.
eToEArrayInterface : PlayerInterface[2]; an array of players stored in the model, for AI vs AI mode.

int fieldJustMovedIn: monitors which field has been moved in just now
status:
status=5: red wins
status=6: blue wins
status=7: draw
status=2: a field has been moved successfully
wintype= 1,2,3,4,5,6,7,8,9,10,11,12,13
the index of a field= 0-63
x,y,z of a field: 1,2,3,4 Because I read the requirement from pdf first. It says x,y,z belongs to 1,2,3,4. I implemented it in the very beginning stage. Later, I did not have time to change it.

Part B. Discussion per Class

In the GUI version project:

- Player:

This class is used for creating a player. It implements the interface PlayerInterface. It stores player’s name, color, socket, mode (which kind of game the player wants to play) and clientAlive (connection status). All the variables can be got by the corresponding methods.

- PlayerClient:

1) ask the user to log in with ip address and port number.
2) create a socket for this client.
3) the main method creates a new object of PlayerClient, stores the socket of player, starts a thread which sends client's input to the server, and starts another thread which sends server's input to the client.

- PlayerDetails:

It stores 7 array lists. Each mode (PtoP or PtoE or EtoE) has two array lists:
-One array list stores player's name
-The other array list stores player's socket
When a player connects to the server, the server does the following:
1) asks the player which kind of game the user wants to play,
2) asks the player for its name.
3) registers the player's name and his socket into two different Array Lists.
4) To start a game, there should be 2 players. Therefore the server monitors the length of the array list. When the length reaches 2, the server notifies all, so that the object of class player will be created (in the class StarterHumanVsHuman, class StarterHumanVsAi, class StarterAiVsAi). This is the main idea of this class.
5) A special function is to check the waiting list. The purpose is to deal with this kind of situation: when two players (A and B) playing the game, one player (B) exit the game. The other player (A) can choose to exit or to stay and wait for a new player to start a new game. If the player (A) wants to stay, then the server put this player (A) into the array list called WaitingPlayer. When a new player (C) comes in, if there is a player (A) who is willing to play the same mode game, then the server matches A with C as a pair. So the old player (A) do not need to log in again.

- GameServer:

1) Ask the user to give a port number of server
2) Create a server socket for the server
3) Initialize the 7 array lists (

humanVsHumanName
humanVsAiName
aiVsAiName;
humanVsHumanSocket;
humanVsAiSocket;
aiVsAiSocket;
waitingPlayers;

which will be used in other classes to store players' names and sockets). They are all array lists. Because the length of array lists is dynamic. It is easy to remove or add an element from the array list. The length will automatically be updated. So it becomes easy to monitor how many players are there.

4) Starts three different kinds of games (StarterHumanVsHuman, StarterHumanVsAI, StarterAiVsAi) . These three kinds of games are started as threads. The aim is to realize that multiple games can run at the same time. The games that run in the same time can be either in the same mode or in different mode.

- Game starter classes:

There are three game starter classes: StarterHumanVsHuman, StarterHumanVsAI, StarterAiVsAi. All of them implements Runnable. So multiple games can run at the same time. The starter classes are called by the GameServer class. The functions of these three classes are the same:

1) Each game starter class has an access to the corresponding namelist (eg. array list humanVsHumanNameList) and socketlist (eg. array list humanVsHumanSocketList)
2) when the number of players<2, wait for another player to join.
3) when the number of players=2, create 2 objects of the Player class, start the controller.
4) One special thing is that the corresponding namelist (eg. array list humanVsHumanNameList) and socketlist (eg. array list humanVsHumanSocketList) will be cleared after the Player object are generated. The reason to make them clear is to make them available for new upcoming players.

To get more detailed explanation, please see the java doc in the class StarterHumanVsHuman.

- Model:

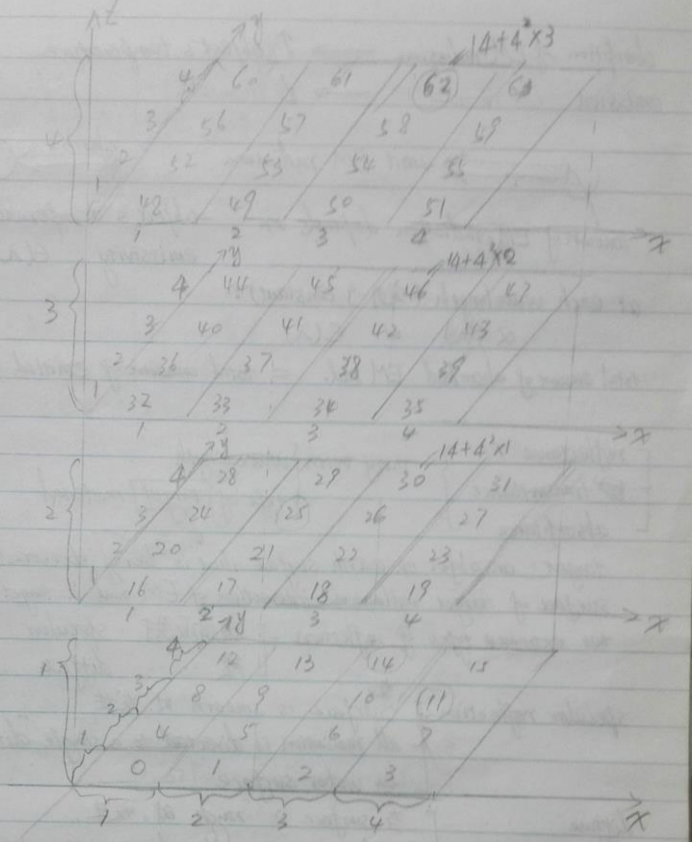
1) initialize the board (Color[] field).
2) Make a move on the board. The gravity is takken into account.
3) Check the result whether anybody win the game or there is a draw (no body wins) or the game is unfinished. Find the best field to move in.
4) Hint: when the user asks which place to move, the model gives the advice. The advice comes from the method bestFieldToMove(color)
5) AIMove: the model provides the best field to move in. The decision is made by the method bestFieldToMove(color). The AI sleeps for a certain amount of time before making a movement. The thinking time is a parameter. It can be defined by the user in the beginning of the game.
6) The Hint function and AI movement is based on the method bestFieldToMove(color). This method provides some artificial intelligence behavior: if the middle field of the board is not occupied, move to it. Otherwise, if the current player only need one piece to get a direct win, then move to that field. Otherwise, if the opponent has 3 pieces in a row, then defend it. Otherwise, choose a random field. This method has been tested. The defensive performance is very good. When AI uses it, it is not easy for a human player (opponent) to win.
7) The picture bellow shows how I made index to the 3D board. The other picture shows how I give numbers to the diagonals in various directions.

y \ x	1	2	3	4
1	0	1	2	3
2	4	5	6	7
3	8	9	10	11
4	12	13	14	15

y \ x	1	2	3	4
1	16	17	18	19
2	20	21	22	23
3	24	25	26	27
4	28	29	30	31

y \ x	1	2	3	4
1	32	33	34	35
2	36	37	38	39
3	40	41	42	43
4	44	45	46	47

y \ x	1	2	3	4
1	48	49	50	51
2	52	53	54	55
3	56	57	58	59
4	60	61	62	63



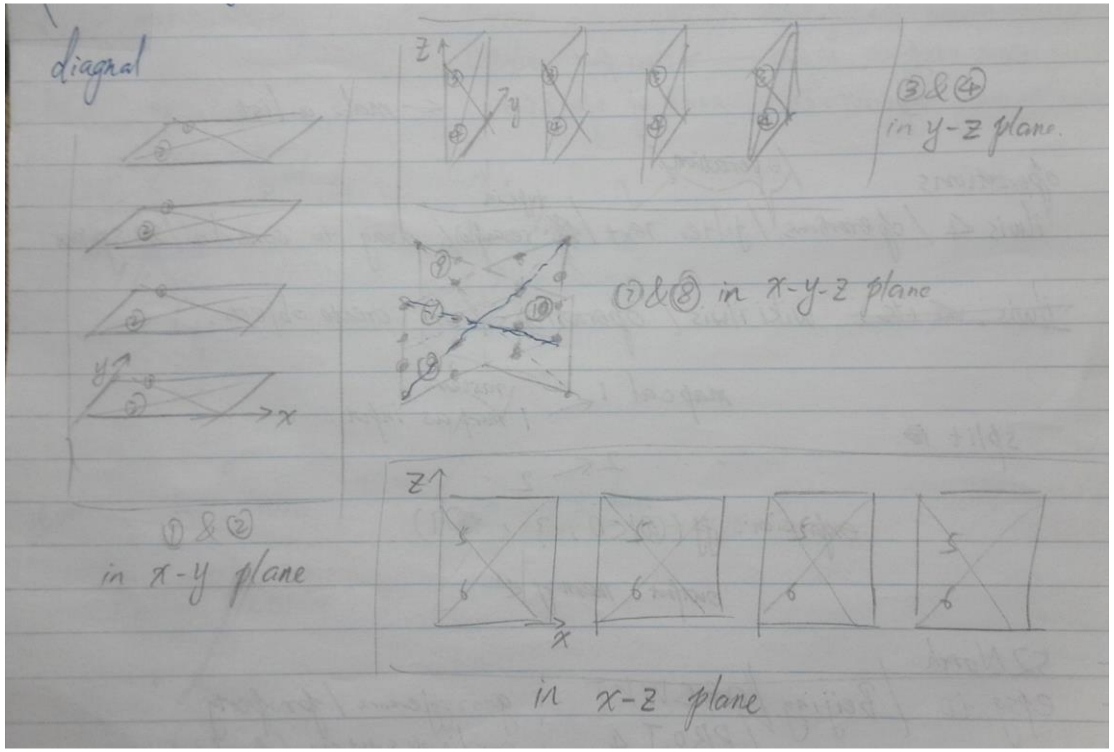
DIM= 4.

index	x	y	z
0	0	0	0
11	3	2	0

11÷4=2余3.

index∈[0,15] z=0

index x y z



- Controller:

The controller controls the game based on 3 cases:
mode ==0 (human <u>vs</u> human)|| mode==1 (human <u>vs</u> AI)|| mode==2 (AI <u>vs</u> AI)
checks whether the player's input is valid or not. If it is invalid, tells the player why it is invalid. It checks until the input is valid. Then it calls the methods in the model, and reacts to the player's input. The reactions include:

1) Check the turn of players, tell the player to move
2) Tells the board to be initialized (start a new game)

3) Drops a piece on a field
4) Gives a hint to the player
5) Reads thinking time from player and pass it to AI movement.
6) Tells AI to move
7) Tells the model to check the game result
8) Tells the viewer to display the board
9) Tells the viewer to show error messages when player's input is invalid
10) After a game is over, asks both players whether or not they want to restart the game. The game will restart only when both players are willing to
11) If any play wants to exit the game, clean the board, delete this player from the player list. Asks the other player whether he wants to exit or wait for another opponent to register in.

All three modes perform the above behaviors. To get more detailed explanation, please see the java doc in the class

- View of client:

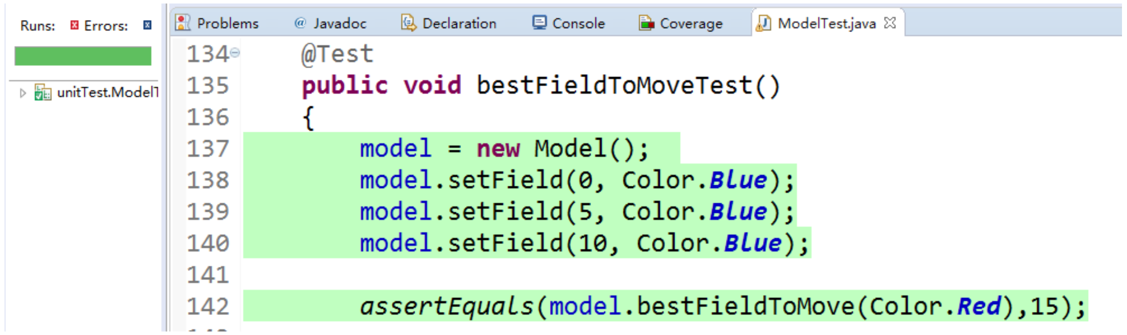
ThreadOfServer: read client's message (message from console), send it to the server. It extends Thread, because multiple players need to be communicated at the same time.
ThreadOfClient: read server's message (messages from controller), send it to the server. It extends Thread, because multiple players need to be communicated at the same time.
These two classes should have been put into the package view.

Part C. Test Report

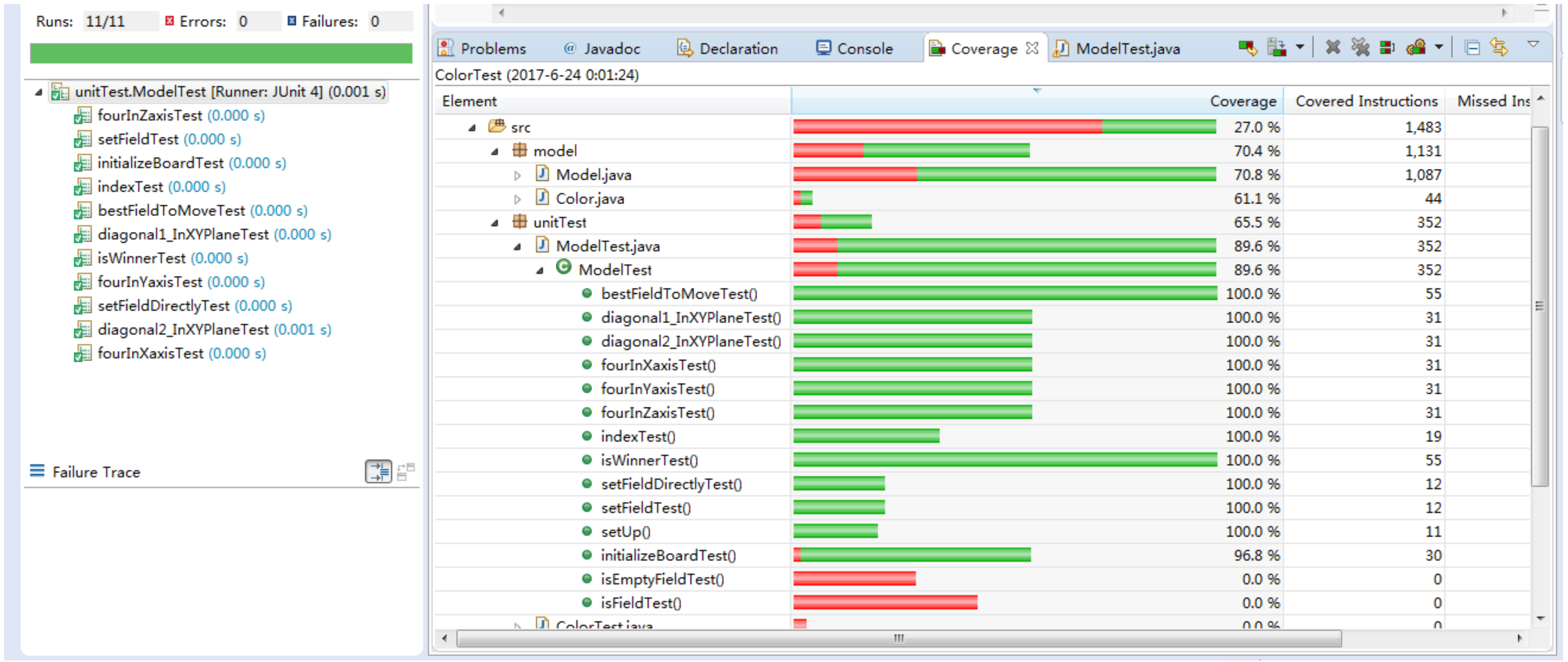
- *Unit Testing.*

I made a JUnit on the model class. The Junit testing file is called "ModelTest". In the "ModelTest" file, I create an object of Model first. This is in @Before. Then I wrote @test methods to test almost all the methods in the Model class. Especially, I wrote a case to test whether or not the method "bestFieldToMove()" can detect opponent's win. The result is the same as expected. After that, I ran the test by the coverage configuration.

The result of the coverage is 89.6%. It means that 89.6% of the source code has been tested. I checked the coverage of each method one by one. Most of the testing methods are in green. The green bar means tested. The red bar means untested. In the left window, it shows that 11 methods have been ran. 11 methods have been run successfully. There are no errors or failures during running.



```
    @Test
    public void bestFieldToMoveTest()
    {
        model = new Model();
        model.setField(0, Color.Blue);
        model.setField(5, Color.Blue);
        model.setField(10, Color.Blue);

        assertEquals(model.bestFieldToMove(Color.Red),15);
```



Runs: 11/11 Errors: 0 Failures: 0

```
unitTest.ModelTest [Runner: JUnit 4] (0.001 s)
    fourInZaxisTest (0.000 s)
    setFieldTest (0.000 s)
    initializeBoardTest (0.000 s)
    indexTest (0.000 s)
    bestFieldToMoveTest (0.000 s)
    diagonal1_InXYPlaneTest (0.000 s)
    isWinnerTest (0.000 s)
    fourInYaxisTest (0.000 s)
    setFieldDirectlyTest (0.000 s)
    diagonal2_InXYPlaneTest (0.001 s)
    fourInXaxisTest (0.000 s)
```

Failure Trace

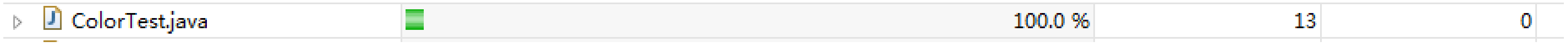
Problems | Javadoc | Declaration | Console | Coverage | ModelTest.java

ColorTest (2017-6-24 0:01:24)

Element	Coverage	Covered Instructions	Missed Ins
src	27.0 %	1,483	
model	70.4 %	1,131	
Model.java	70.8 %	1,087	
Color.java	61.1 %	44	
unitTest	65.5 %	352	
ModelTest.java	89.6 %	352	
ModelTest	89.6 %	352	
bestFieldToMoveTest()	100.0 %	55	
diagonal1_InXYPlaneTest()	100.0 %	31	
diagonal2_InXYPlaneTest()	100.0 %	31	
fourInXaxisTest()	100.0 %	31	
fourInYaxisTest()	100.0 %	31	
fourInZaxisTest()	100.0 %	31	
indexTest()	100.0 %	19	
isWinnerTest()	100.0 %	55	
setFieldDirectlyTest()	100.0 %	12	
setFieldTest()	100.0 %	12	
setUp()	100.0 %	11	
initializeBoardTest()	96.8 %	30	
isEmptyFieldTest()	0.0 %	0	
isFieldTest()	0.0 %	0	
ColorTest.java	0.0 %	0	

Although the coverage is high (89.6%), the coverage is not 100%. Because some methods in the model class were not tested, such as diagonal_InxyzSpace() and AIMove(). For diagonal in 3D spaces, I have tested them by system testing. It succeeded. So I didn't test it by Junit test. For AIMove(), it does not perform a lot of actions. Its action is very simple. So I did not write test for it. But I did system test on it. If the time is sufficient, I will write Junit test for these two methods. The methods that I have tested is the main part of the source code. Therefore, the result (89.6%) is reasonable.

By the same process, I made a Junit test for the class Color. The result is 100%, which is very good. It matches with my expectation very well.



ColorTest.java	100.0 %	13	0

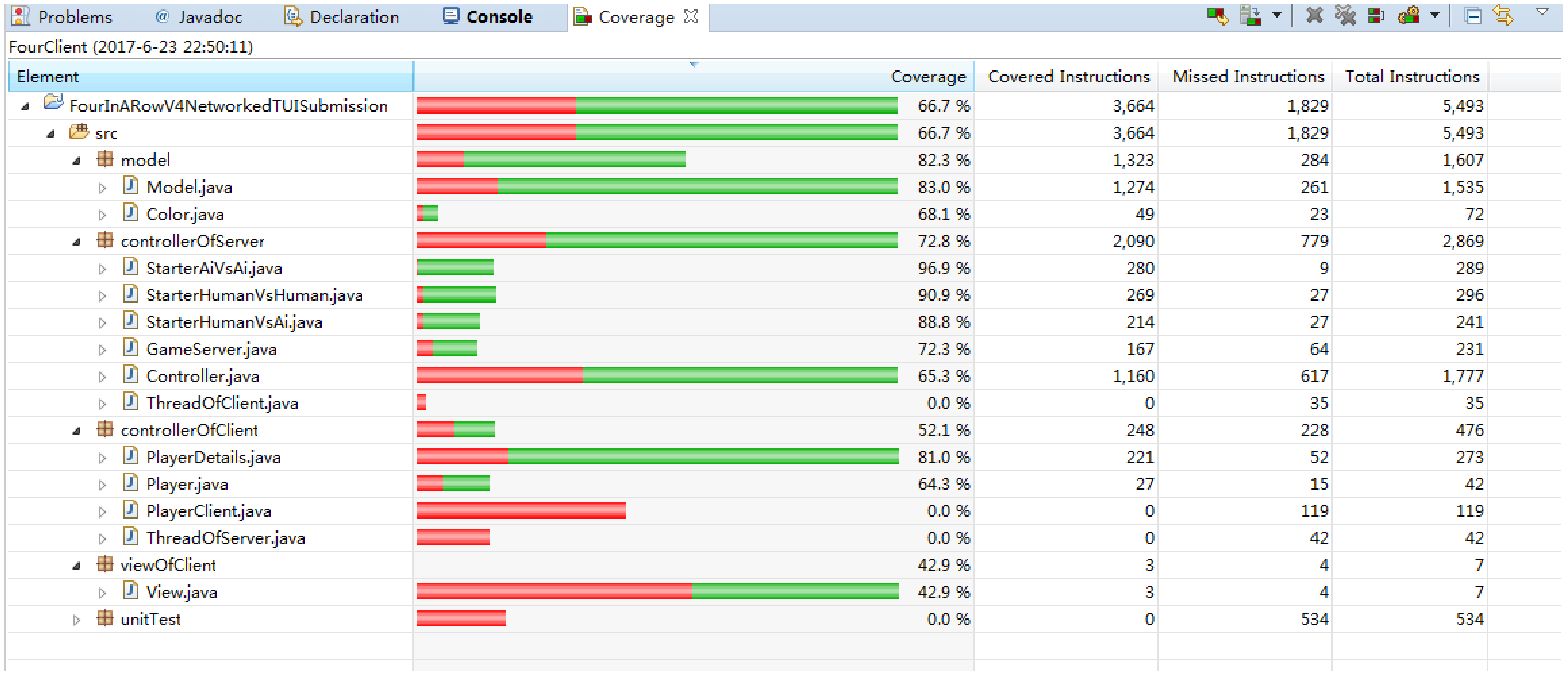
In the ControllerTest and PlayerTest, the coverage is low. The reason is that the socket cannot be successfully created. So the methods in those class could be called. In order to test these classes and other classes, my program need to connect to the internet. I wanted to do a telnet test. But because of the time, I could not implement it.

If the coverage is too low, it means that either the source code has problems or the test code has problems. The programmer should find out where the problem is. The purpose is to prevent that the program works but does not work as expected.

For most of the classes in this project (except for the model class), they cannot be tested isolated. For example, when testing the controller, the model should be initialized. When testing the PlayerDetails, the controller should be initialized.

Concerning which test technique has been applied, I used the Junit test. I used: assertTrue(boolean condition), assertFalse(boolean condition), assertEquals(java.lang.Object expected, java.lang.Object actual).

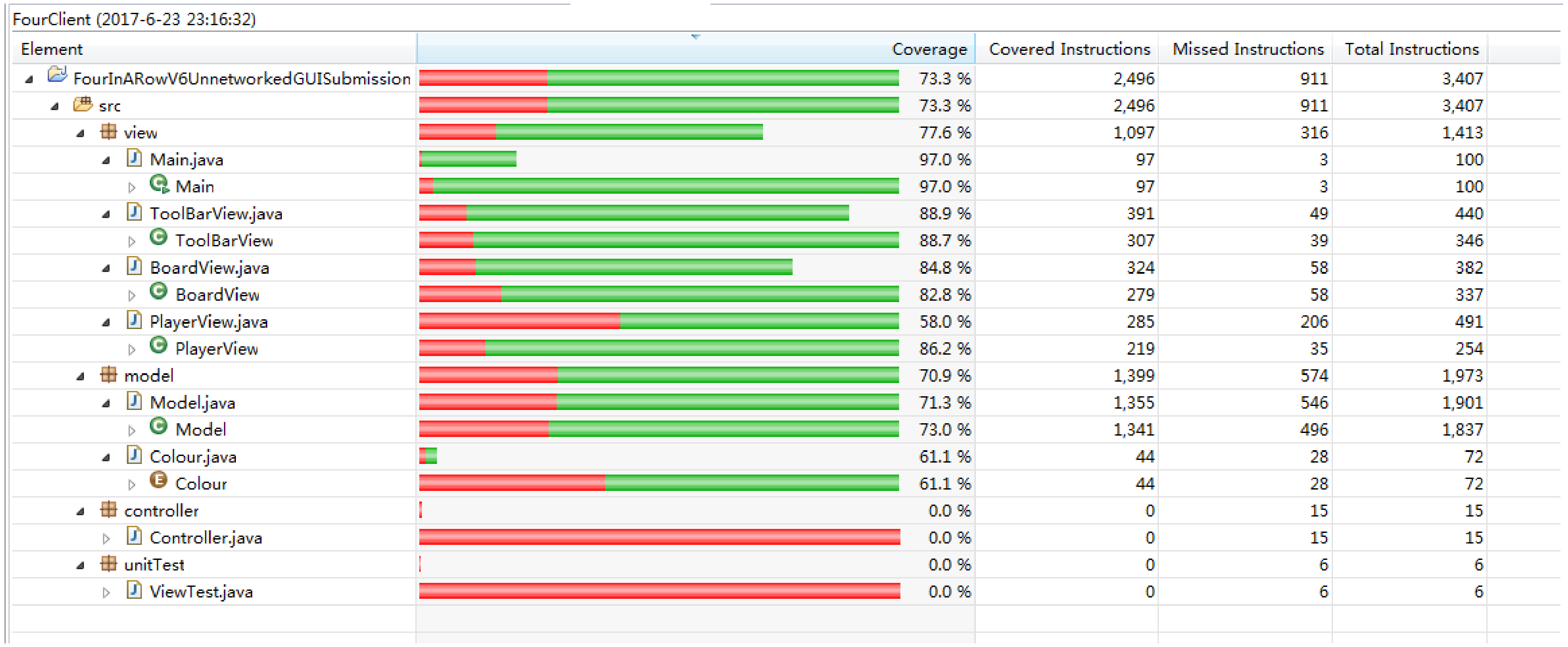
- Junit test for the whole project. I used the coverage to test my program. The result is positive. For the project in the TUI version, the coverage is 66.7%. For the project in the GUI version, the coverage is 73.3%. Both of them >50%.



FourClient (2017-6-23 22:50:11)

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
FourInARowV4NetworkedTUISubmission	66.7 %	3,664	1,829	5,493
src	66.7 %	3,664	1,829	5,493
model	82.3 %	1,323	284	1,607
Model.java	83.0 %	1,274	261	1,535
Color.java	68.1 %	49	23	72
controllerOfServer	72.8 %	2,090	779	2,869
StarterAiVsAi.java	96.9 %	280	9	289
StarterHumanVsHuman.java	90.9 %	269	27	296
StarterHumanVsAi.java	88.8 %	214	27	241
GameServer.java	72.3 %	167	64	231
Controller.java	65.3 %	1,160	617	1,777
ThreadOfClient.java	0.0 %	0	35	35
controllerOfClient	52.1 %	248	228	476
PlayerDetails.java	81.0 %	221	52	273
Player.java	64.3 %	27	15	42
PlayerClient.java	0.0 %	0	119	119
ThreadOfServer.java	0.0 %	0	42	42
viewOfClient	42.9 %	3	4	7
View.java	42.9 %	3	4	7
unitTest	0.0 %	0	534	534

Coverage of the project in the TUI version: 66.7%.



FourClient (2017-6-23 23:16:32)

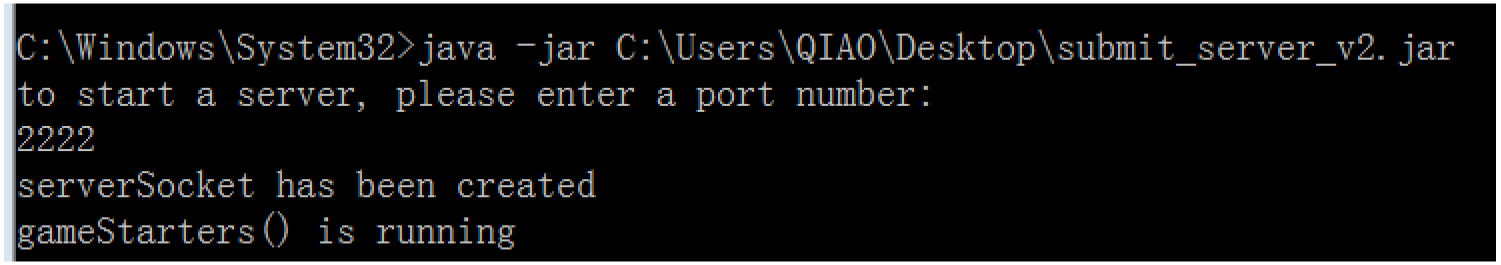
Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
FourInARowV6UnnetworkedGUISubmission	73.3 %	2,496	911	3,407
src	73.3 %	2,496	911	3,407
view	77.6 %	1,097	316	1,413
Main.java	97.0 %	97	3	100
Main	97.0 %	97	3	100
ToolBarView.java	88.9 %	391	49	440
ToolBarView	88.7 %	307	39	346
BoardView.java	84.8 %	324	58	382
BoardView	82.8 %	279	58	337
PlayerView.java	58.0 %	285	206	491
PlayerView	86.2 %	219	35	254
model	70.9 %	1,399	574	1,973
Model.java	71.3 %	1,355	546	1,901
Model	73.0 %	1,341	496	1,837
Colour.java	61.1 %	44	28	72
Colour	61.1 %	44	28	72
controller	0.0 %	0	15	15
Controller.java	0.0 %	0	15	15
unitTest	0.0 %	0	6	6
ViewTest.java	0.0 %	0	6	6

Coverage of the project in the GUI version: 73.3%

- Visual inspection of UI,

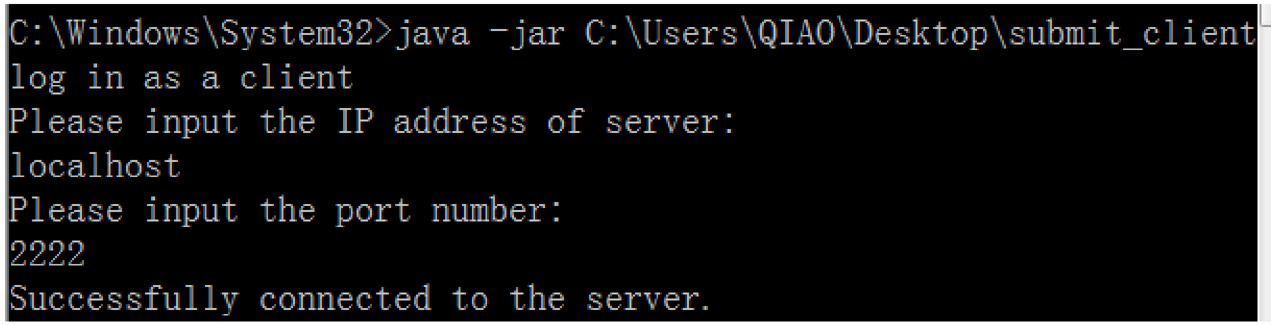
The TUI version-project:

To create a server



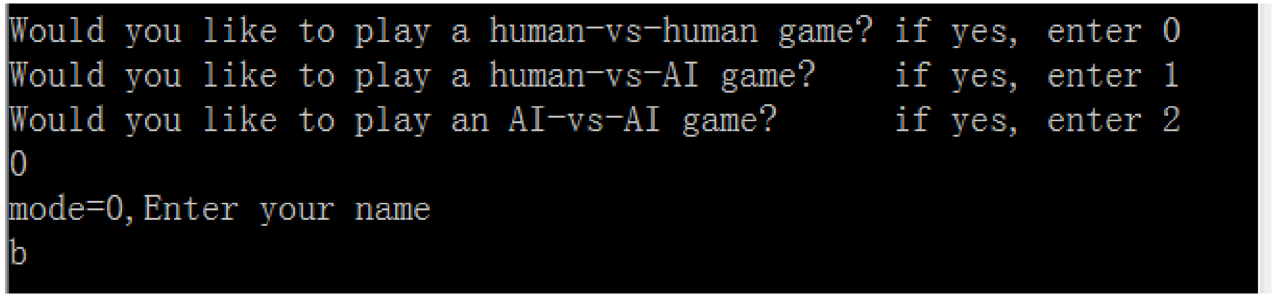
```
C:\Windows\System32>java -jar C:\Users\QIAO\Desktop\submit_server_v2.jar
to start a server, please enter a port number:
2222
serverSocket has been created
gameStarters() is running
```

To connect to the server:



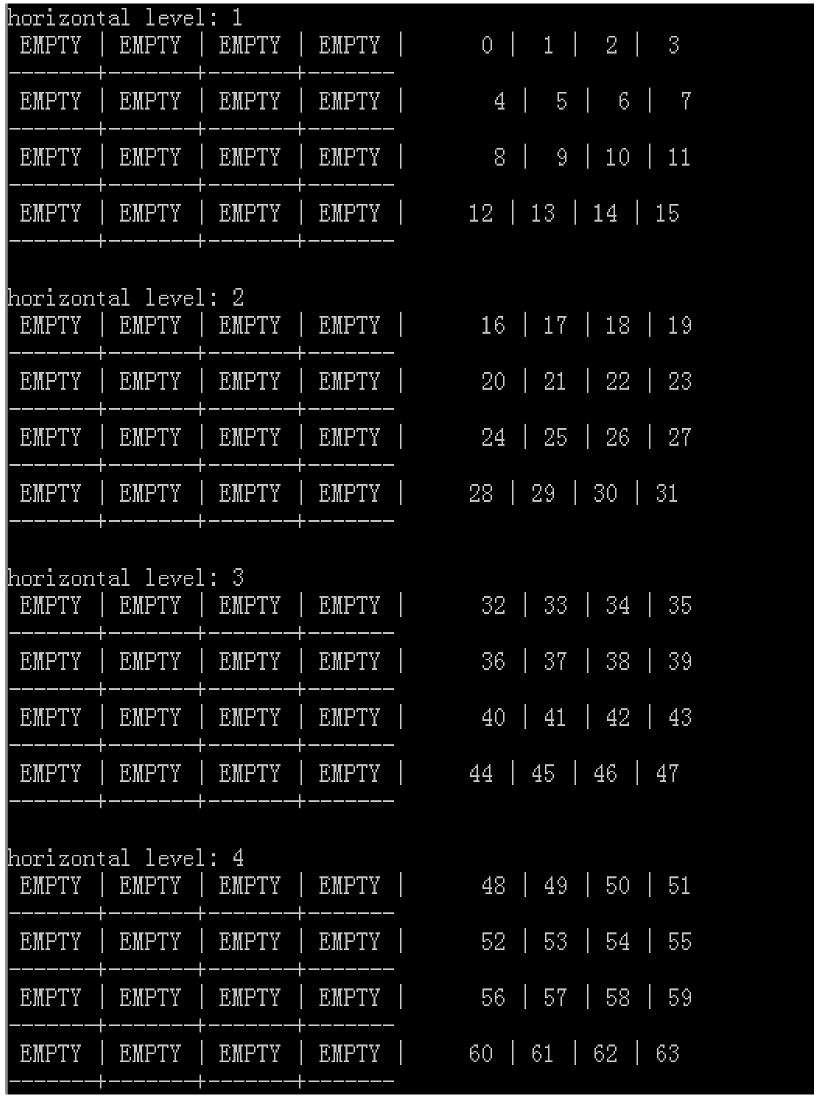
```
C:\Windows\System32>java -jar C:\Users\QIAO\Desktop\submit_client
log in as a client
Please input the IP address of server:
localhost
Please input the port number:
2222
Successfully connected to the server.
```

To register as a player:

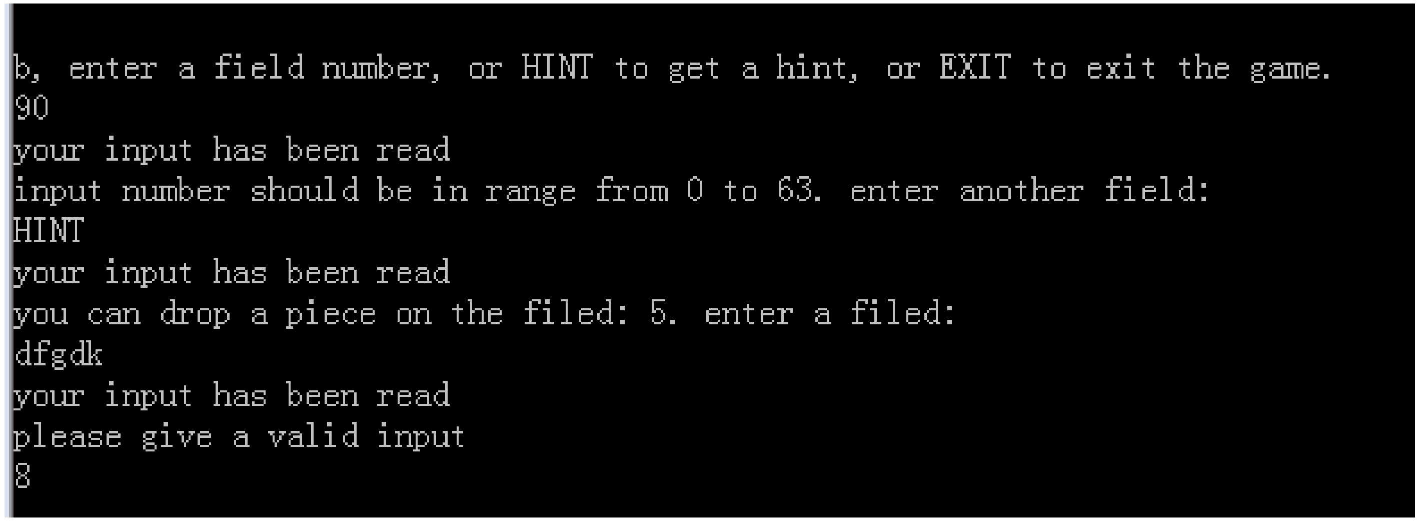


```
Would you like to play a human-vs-human game? if yes, enter 0
Would you like to play a human-vs-AI game?    if yes, enter 1
Would you like to play an AI-vs-AI game?      if yes, enter 2
0
mode=0,Enter your name
b
```

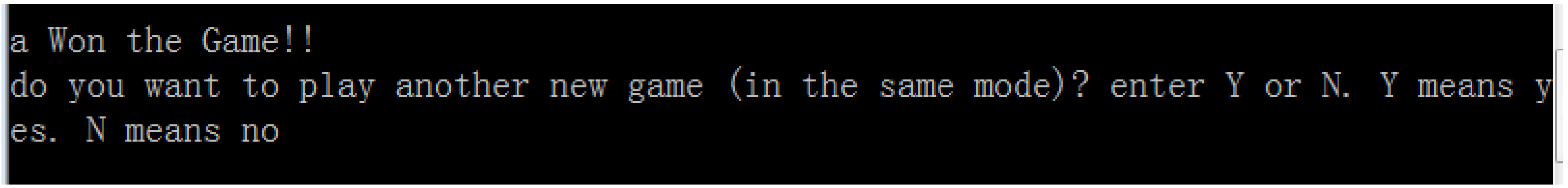
To display the board:


```
horizontal level: 1
 EMPTY | EMPTY | EMPTY | EMPTY |       0 |  1 |  2 |  3
-------+-------+-------+-------
 EMPTY | EMPTY | EMPTY | EMPTY |       4 |  5 |  6 |  7
-------+-------+-------+-------
 EMPTY | EMPTY | EMPTY | EMPTY |       8 |  9 | 10 | 11
-------+-------+-------+-------
 EMPTY | EMPTY | EMPTY | EMPTY |     12 | 13 | 14 | 15
-------+-------+-------+-------

horizontal level: 2
 EMPTY | EMPTY | EMPTY | EMPTY |      16 | 17 | 18 | 19
-------+-------+-------+-------
 EMPTY | EMPTY | EMPTY | EMPTY |      20 | 21 | 22 | 23
-------+-------+-------+-------
 EMPTY | EMPTY | EMPTY | EMPTY |      24 | 25 | 26 | 27
-------+-------+-------+-------
 EMPTY | EMPTY | EMPTY | EMPTY |     28 | 29 | 30 | 31
-------+-------+-------+-------

horizontal level: 3
 EMPTY | EMPTY | EMPTY | EMPTY |      32 | 33 | 34 | 35
-------+-------+-------+-------
 EMPTY | EMPTY | EMPTY | EMPTY |      36 | 37 | 38 | 39
-------+-------+-------+-------
 EMPTY | EMPTY | EMPTY | EMPTY |      40 | 41 | 42 | 43
-------+-------+-------+-------
 EMPTY | EMPTY | EMPTY | EMPTY |     44 | 45 | 46 | 47
-------+-------+-------+-------

horizontal level: 4
 EMPTY | EMPTY | EMPTY | EMPTY |      48 | 49 | 50 | 51
-------+-------+-------+-------
 EMPTY | EMPTY | EMPTY | EMPTY |      52 | 53 | 54 | 55
-------+-------+-------+-------
 EMPTY | EMPTY | EMPTY | EMPTY |      56 | 57 | 58 | 59
-------+-------+-------+-------
 EMPTY | EMPTY | EMPTY | EMPTY |     60 | 61 | 62 | 63
-------+-------+-------+-------
```

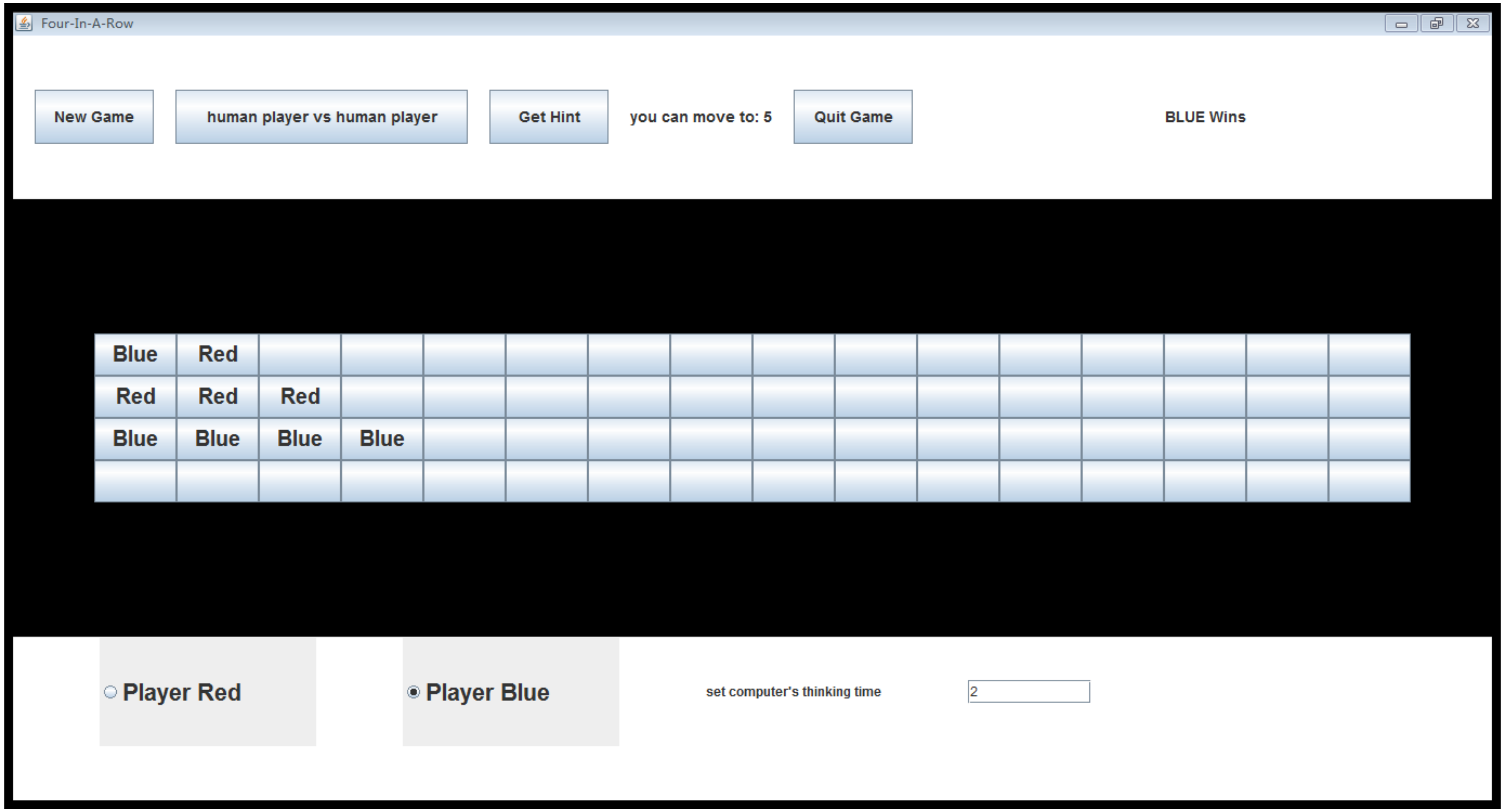
To read player’s input:


```
b, enter a field number, or HINT to get a hint, or EXIT to exit the game.
90
your input has been read
input number should be in range from 0 to 63. enter another field:
HINT
your input has been read
you can drop a piece on the filed: 5. enter a filed:
dfgdk
your input has been read
please give a valid input
8
```

To show the game result:


```
a Won the Game!!
do you want to play another new game (in the same mode)? enter Y or N. Y means y
es. N means no
```

The GUI version


Four-In-A-Row

New Game | human player vs human player | Get Hint | you can move to: 5 | Quit Game | BLUE Wins

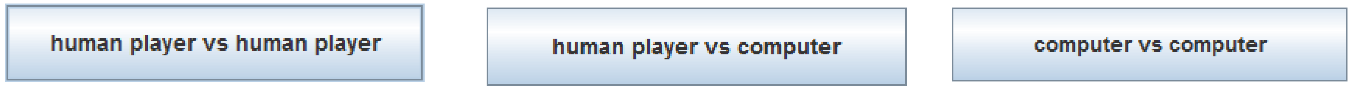
Blue	Red														
Red	Red	Red													
Blue	Blue	Blue	Blue												

○ Player Red   ● Player Blue   set computer's thinking time: 2

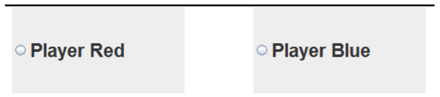
1) click "new game" button: initialize the board

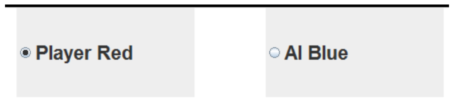
2) Choose a mode: by clicking the mode button. The mode button has 3 switches:

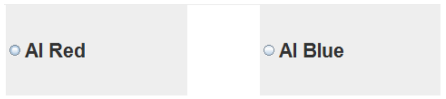
“human player vs human player” , “human player vs computer”, “computer vs computer”



3) Choose which color makes the first movement: by clicking the button “player/AI red” or “player/AI blue”





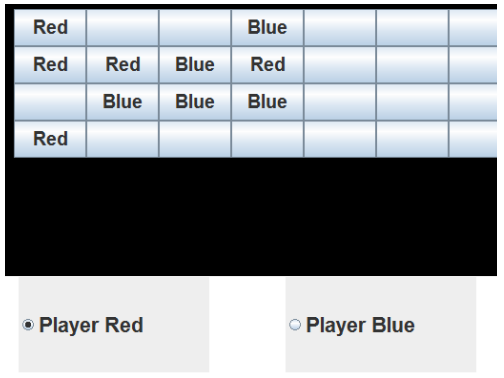


(In “human player vs human “ mode) (in “human player vs computer” mode) (in “computer vs computer” mode)

4) No matter which mode the game is, the user should always set the computer’s thinking time. Because the program need to read it. Otherwise, the program can not start a game. I know that this should be improved. In human vs computer mode, the larger the input number is, the longer the thinking time of computer is. So defining the thinking time works as expected.

set computer's thinking time 1000

5) The bottom bar shows who’s turn. When it is red player’s turn, the bar “Player/AI red” is automatically ticked. When it is blue player’s turn, the bar “Player/AI blue” is automatically ticked. The following is an example: in this example, now it is red’ turn.



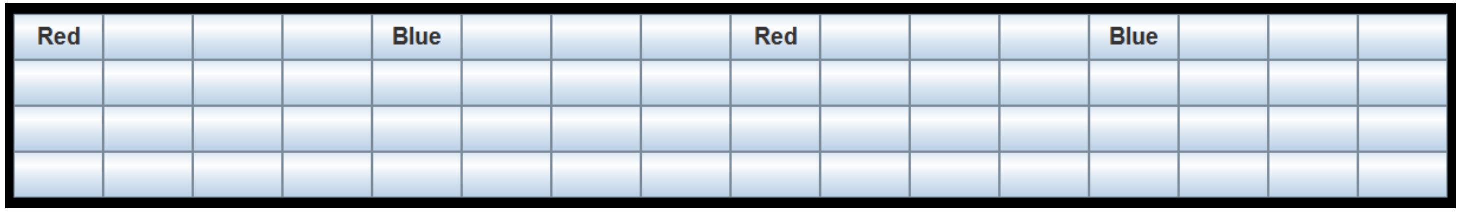
Red			Blue			
Red	Red	Blue	Red			
	Blue	Blue	Blue			
Red						

6) To drop a piece, the player only needs to click buttons in the board. The board is a 4*16 metrixs. It displays the 3D board into 2D. It means the following:

Horizontal Level 1				Horizontal Level 2				Horizontal Level 3				Horizontal Level 4			
0	1	2	3	16	17	18	19	32	33	34	35	48	49	50	51
4	5	6	7	20	21	22	23	36	37	38	39	52	53	54	55
8	9	10	11	24	25	26	27	40	41	42	43	56	57	58	59
12	13	14	15	28	29	30	31	44	45	46	47	60	61	62	63

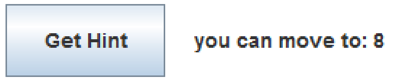
Each number means the number of a field. All the fields are buttons. When the player clicks the button, the button shows the color. It is better to split 4 horizontal levels. For example, add some spaces between levels, when displaying the buttons. So the players can understand the board easily.

If a field has been occupied, when the player wants to drop a piece on this field, due to gravity, the new piece will be droped automatically in the same x,y coordintor, but in a different z coordinator. For example:



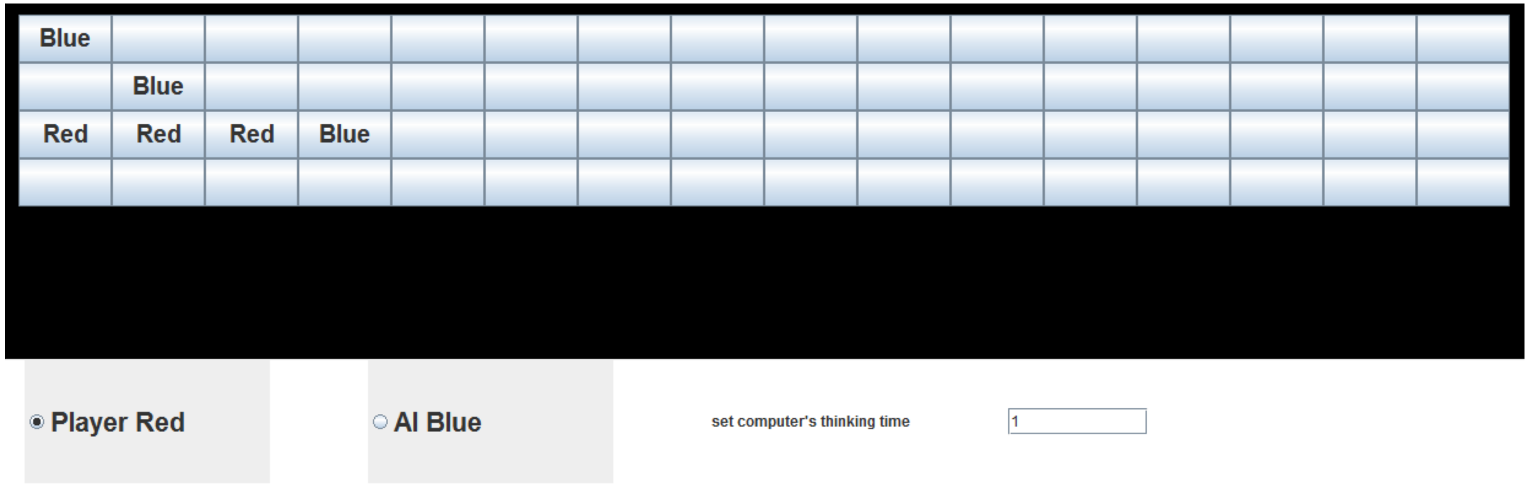
In this example, after the field 0 has been occupied by red, when the blue players clicks "0", blue will drop its piece to the field 6. Same goes for button 32 and 48.

7) During the game, click the "hint" button to get a hint. The hint message is shown right beside the hint button. The user has freedom to choose whether or not to follow the advice from hint.



you can move to: 8

8) The AI performs artificial behavior. The algorithms behind is same as in the TUI-version project. Here is an example, when 3 red colors are in a line, the AI detects it. To defend human palyer, AI drops its blue piece to the last empty field in red's line.



Blue															
	Blue														
Red	Red	Red	Blue												

● Player Red ○ AI Blue

set computer's thinking time 1

9) When a game is finished, the game result (somebody wins or draw) is shown in the right top message bar.

BLUE Wins

10) The button "new game" can be clicked at any time during the game. This button combines the funtions
  a) "start a new game in the beginning" "
  b) "restart a game, when the current game is unfinished"
  c) "restart a game, when the current game is finished".

11) If a player wants to quit the game, he/she can click the red cross button. Because the GUI-version project is unnetworked, it cannot inform the other players. But this function exists in the TUI-version project.

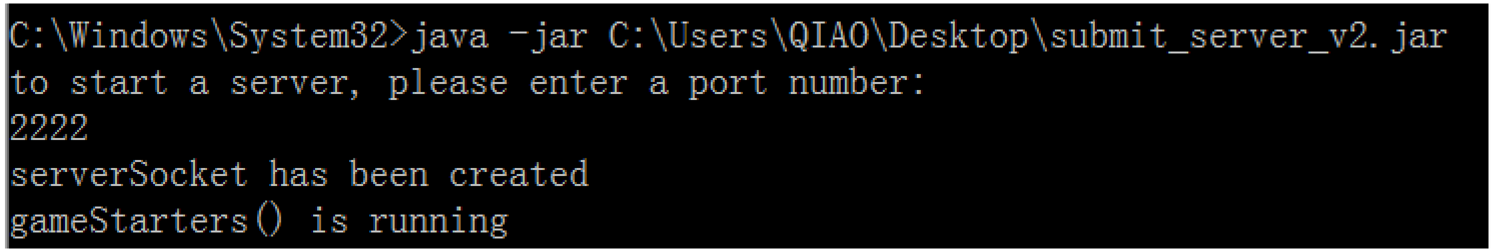
In the GUI version, the computer vs computer mode does not work well. I tried a lot ways to fix it. A good thing is that the computer vs computer mode works successfully in the TUI-version project.

- System testing

My program satisfies the following requirements:

I did the system testing in two environemnts: 1) in command.exe and 2) in the console of Eclipse. The test results are the same.

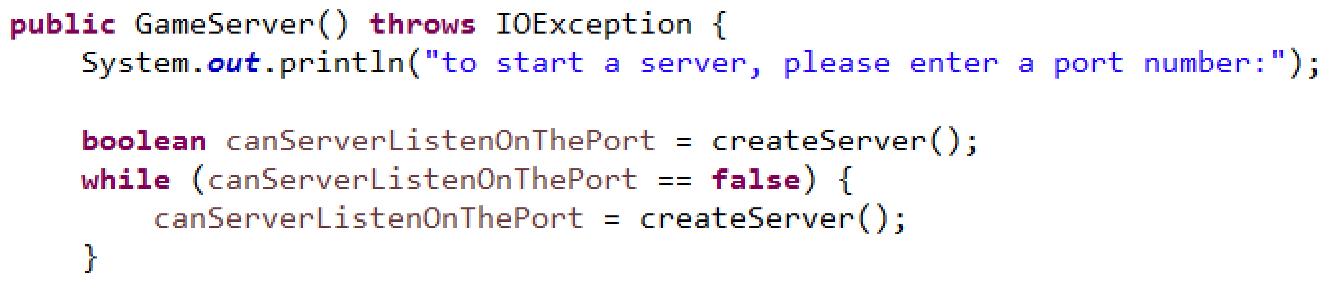
1. When the server is started, a port number should be entered that the server will listen to.



```
C:\Windows\System32>java -jar C:\Users\QIAO\Desktop\submit_server_v2.jar
to start a server, please enter a port number:
2222
serverSocket has been created
gameStarters() is running
```

2. If the port number already is in use, an appropriate error message is returned, and a new port number can be entered.

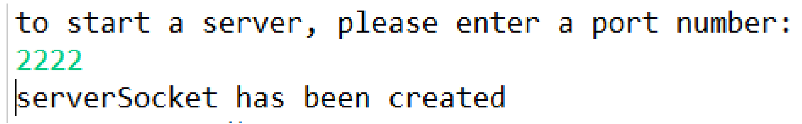
In the submitted project, the program is able to detect whether the input port is valid or not. To enter the new port number, the user need to restart a window. So I made some improvements. The improvements are shown in the pictures bellow. I wrote a method createServer() to check whether the port is available or not. This method returns true, if the port is available. It returns false, if the entered port has been being listened by other server. Here is the improved code:



```
public GameServer() throws IOException {
    System.out.println("to start a server, please enter a port number:");

    boolean canServerListenOnThePort = createServer();
    while (canServerListenOnThePort == false) {
        canServerListenOnThePort = createServer();
    }
```

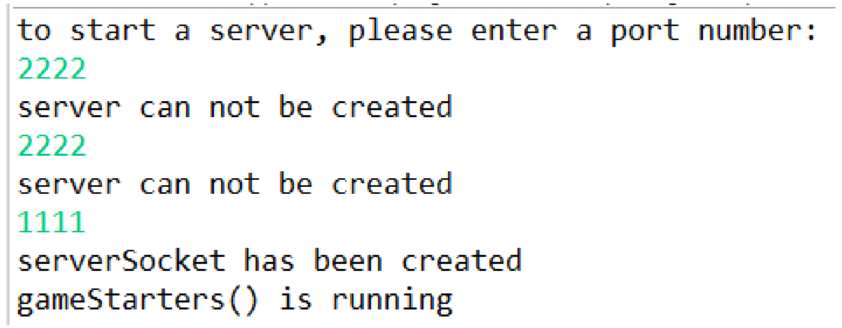
```java
public boolean createServer() {
    boolean boo = false;
    while (boo == false) {
        String str = read();
        int portInt = 0;
        try {
            portInt = Integer.parseInt(str);
            port = portInt;
            boo = true;
        } catch (NumberFormatException e) {
            boo = false;
            System.out.println("The port you entered was invalid, "
                    + "please input another port: ");
        }
    }
    try {
        this.serverSocket = new ServerSocket(port);
        System.out.println("serverSocket has been created ");
        return true;
    } catch (IOException e) {
        System.out.println("server can not be created");
        System.out.println("please enter a new port number");
        return false;
    }
}
```

The improved code works sucssfully. As you can see in the pictures bellow. The port 2222 is listened by the first server (picture 1). So the second potential server can not create a socket on port 2222. Then the program asks the second potential server to give a new port number. The new entered port number is 1111. It is available. So the port 1111 can be used (picture 2). The same port checking process goes for the third server (picture 3).



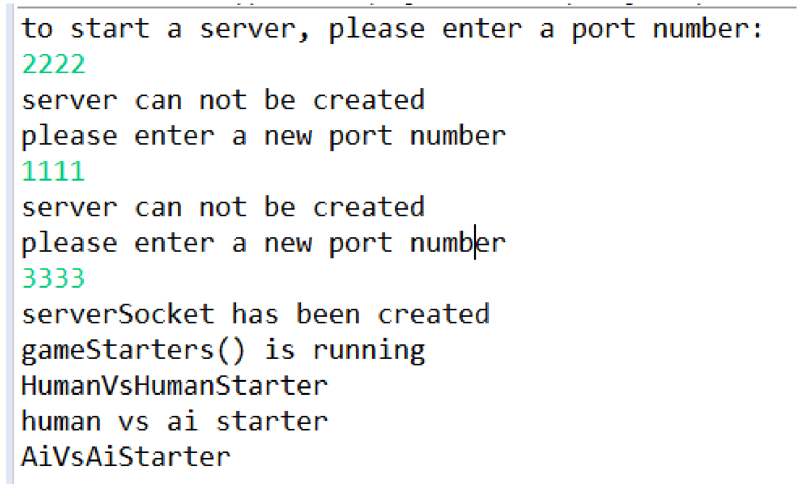
```
to start a server, please enter a port number:
2222
serverSocket has been created
```

Picture 1) The first server



```
to start a server, please enter a port number:
2222
server can not be created
2222
server can not be created
1111
serverSocket has been created
gameStarters() is running
```

Picture 2) the second server

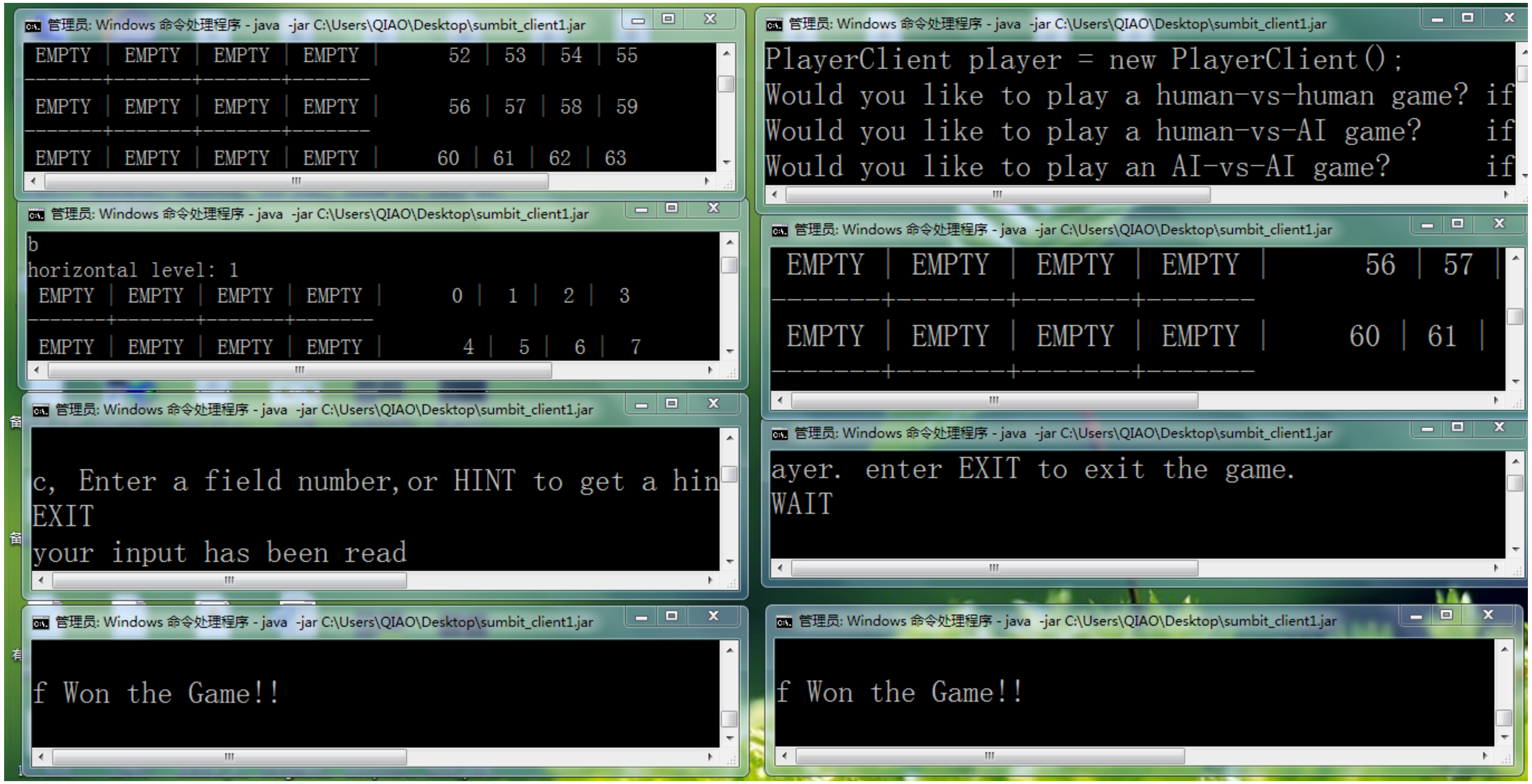


```
to start a server, please enter a port number:
2222
server can not be created
please enter a new port number
1111
server can not be created
please enter a new port number
3333
serverSocket has been created
gameStarters() is running
HumanVsHumanStarter
human vs ai starter
AiVsAiStarter
```

Picture 3) the third server

A problem is that the improved code is not in the submitted project. Because I forgot this requirement. I am sorry for this. But this requirement is satisfied now.

3. A server should be able to support multiple instances of the game that are played simultaneously by different clients.



```
管理员: Windows 命令处理程序 - java  -jar C:\Users\QIAO\Desktop\sumbit_client1.jar
 EMPTY | EMPTY | EMPTY | EMPTY |        52 | 53 | 54 | 55
-------+-------+-------+-------
 EMPTY | EMPTY | EMPTY | EMPTY |        56 | 57 | 58 | 59
-------+-------+-------+-------
 EMPTY | EMPTY | EMPTY | EMPTY |       60 | 61 | 62 | 63
```

```
管理员: Windows 命令处理程序 - java  -jar C:\Users\QIAO\Desktop\sumbit_client1.jar
b
horizontal level: 1
 EMPTY | EMPTY | EMPTY | EMPTY |        0 | 1 | 2 | 3
-------+-------+-------+-------
 EMPTY | EMPTY | EMPTY | EMPTY |        4 | 5 | 6 | 7
```

```
管理员: Windows 命令处理程序 - java  -jar C:\Users\QIAO\Desktop\sumbit_client1.jar

c, Enter a field number,or HINT to get a hin
EXIT
your input has been read
```

```
管理员: Windows 命令处理程序 - java  -jar C:\Users\QIAO\Desktop\sumbit_client1.jar

f Won the Game!!
```

```
管理员: Windows 命令处理程序 - java  -jar C:\Users\QIAO\Desktop\sumbit_client1.jar
PlayerClient player = new PlayerClient();
Would you like to play a human-vs-human game? if
Would you like to play a human-vs-AI game?    if
Would you like to play an AI-vs-AI game?      if
```

```
管理员: Windows 命令处理程序 - java  -jar C:\Users\QIAO\Desktop\sumbit_client1.jar
 EMPTY | EMPTY | EMPTY | EMPTY |       56 | 57 |
-------+-------+-------+-------
 EMPTY | EMPTY | EMPTY | EMPTY |     60 | 61 |
-------+-------+-------+-------
```

```
管理员: Windows 命令处理程序 - java  -jar C:\Users\QIAO\Desktop\sumbit_client1.jar
ayer. enter EXIT to exit the game.
WAIT
```

```
管理员: Windows 命令处理程序 - java  -jar C:\Users\QIAO\Desktop\sumbit_client1.jar

f Won the Game!!
```

The picture above shows that the program is able to support multiple games. Each CMD window is a player view. Here is my way of testing. I created a server on port 2222. I made all the players connect to localhost on port 2222. In the first case:

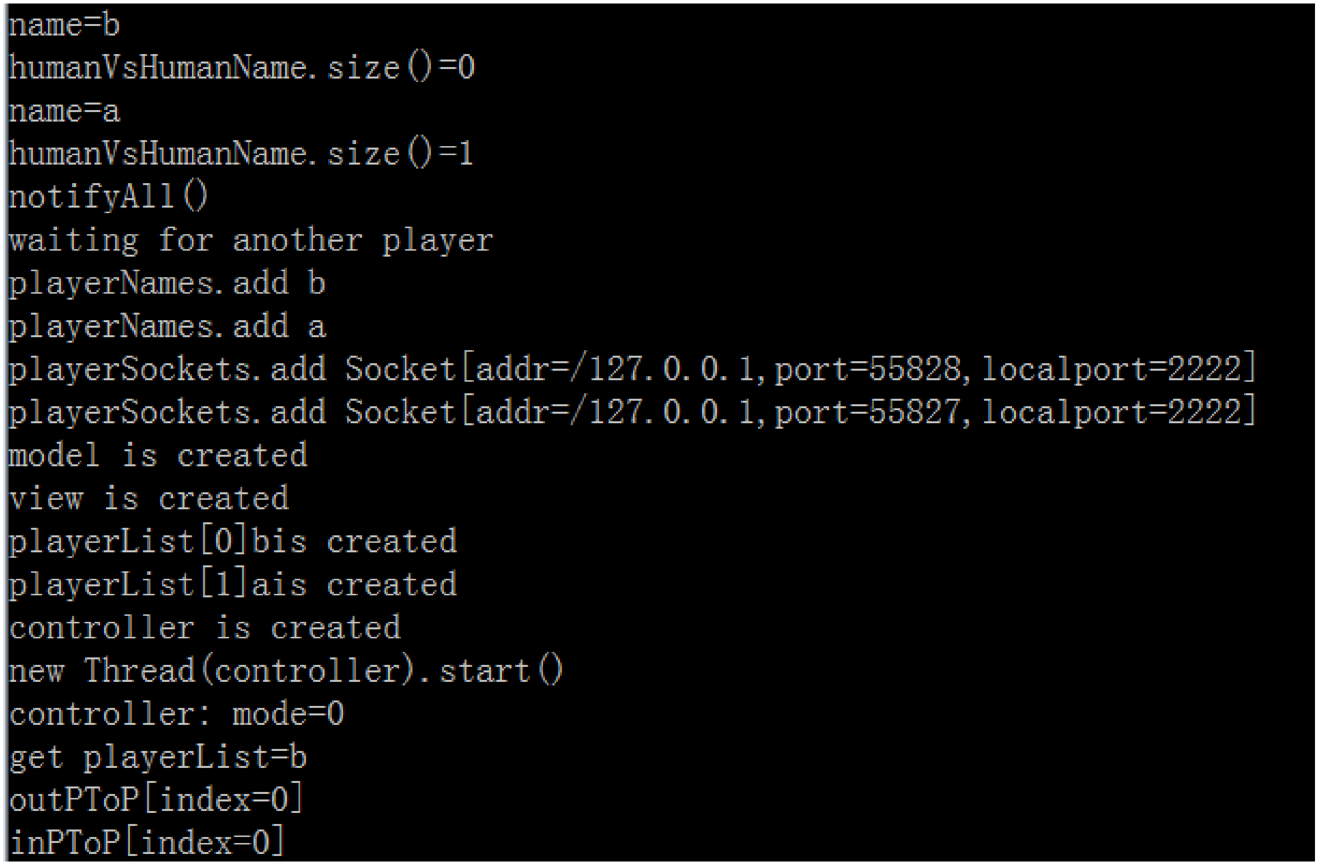
-player a and b: play mode 0 (human vs human)
-player c and d: play mode 0 (human vs human)
-player e and f: play mode 2 (AI vs AI)
-player g and h: play mode 1 (human vs AI)

When player a and b are in their game, player c and d are in their game as well. When player c and d are in their game, player e, f, g, h are also in their game.
After this test, I made a second case:

-player a and b: play mode 0 (human vs human)
-player c and d: play mode 0 (human vs human)
-player e and f: play mode 1 (human vs AI)
-player g and h: play mode 1 (human vs AI)
-player i and j: play mode 1 (human vs AI)
-player k and l: play mode 2 (AI vs AI)
-player m and n: play mode 2 (AI vs AI)

All the players (a, b, c, d, e, f, g, h, I, j, k, l, m, n) are in game. The input of any player only has an influence on its opponent. It does not have any influence on other games. By applying different cases to test, I am very sure that my program satisfies the requirement: Multiple games in the same mode can run at the same time. If the modes are different, games can also run at the same time.

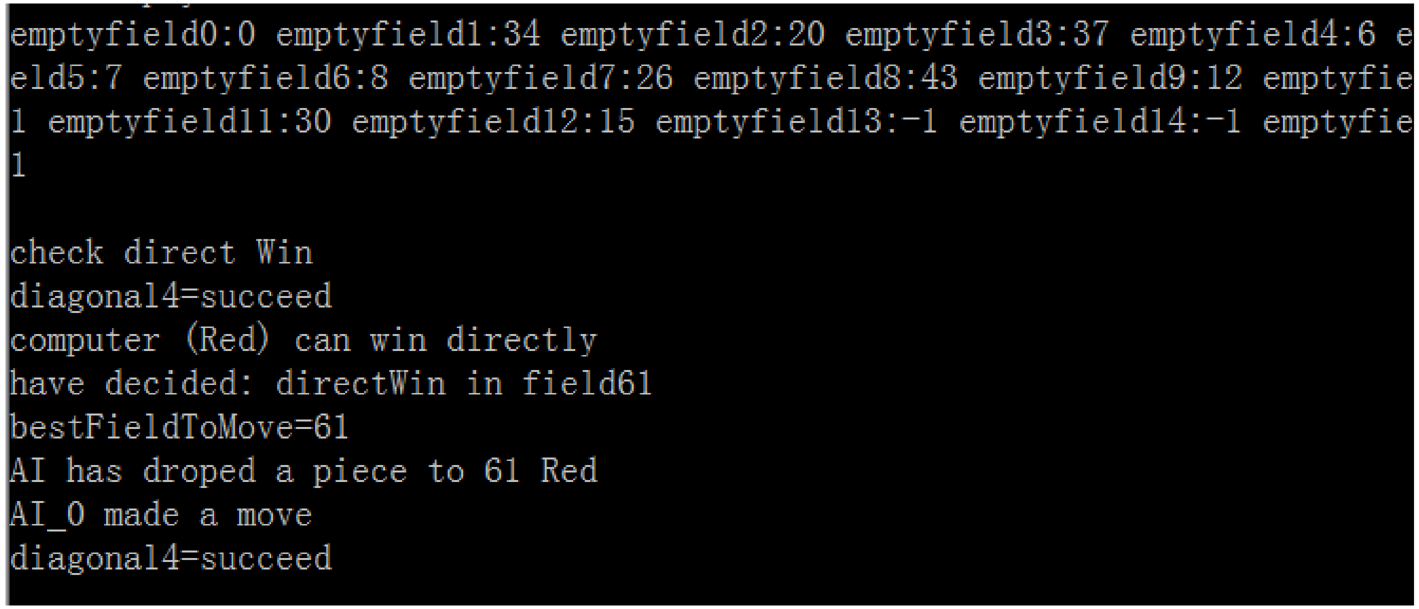
4. The server has a TUI that ensures that all communication messages are written to ***System.out***.
Please see the pictures bellow. The “Server TUI pic1)” shows that the incoming player a and player b has chosen mode 0 (human vs human). When the array list “HumanVsHumanName” contains 2 players, the players are stored in the “playerList”. The sockets of players are clearly shown on the server TUI. After that, model is created. View is created. Controller starts.



```
name=b
humanVsHumanName.size()=0
name=a
humanVsHumanName.size()=1
notifyAll()
waiting for another player
playerNames.add b
playerNames.add a
playerSockets.add Socket[addr=/127.0.0.1,port=55828,localport=2222]
playerSockets.add Socket[addr=/127.0.0.1,port=55827,localport=2222]
model is created
view is created
playerList[0]bis created
playerList[1]ais created
controller is created
new Thread(controller).start()
controller: mode=0
get playerList=b
outPToP[index=0]
inPToP[index=0]
```

Server TUI pic1)

The “Server TUI pic 2)” shows, when it is AI’s turn, how AI finds the best field field to move in. When AI knows if it drops a red piece on the field 61, it can win directly, it moves to field 61. The success is in the line diagonal 14.

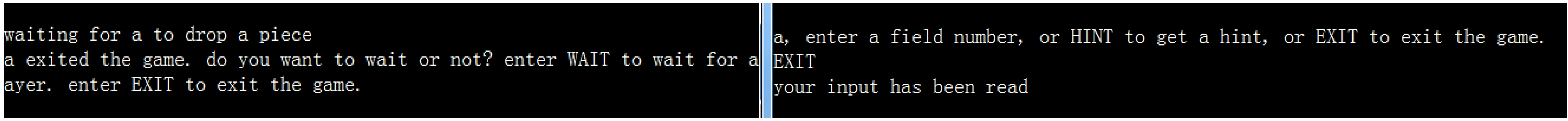


```
emptyfield0:0 emptyfield1:34 emptyfield2:20 emptyfield3:37 emptyfield4:6 e
eld5:7 emptyfield6:8 emptyfield7:26 emptyfield8:43 emptyfield9:12 emptyfie
1 emptyfield11:30 emptyfield12:15 emptyfield13:-1 emptyfield14:-1 emptyfie
1

check direct Win
diagonal4=succeed
computer (Red) can win directly
have decided: directWin in field61
bestFieldToMove=61
AI has droped a piece to 61 Red
AI_0 made a move
diagonal4=succeed
```

Server TUI pic 2)

This test was successful.

5. The server should respect the protocol as defined for the tutorial group during the project session in
My server is able to communicate with all other clients from the tutorial group. No matter a new This test was successful.

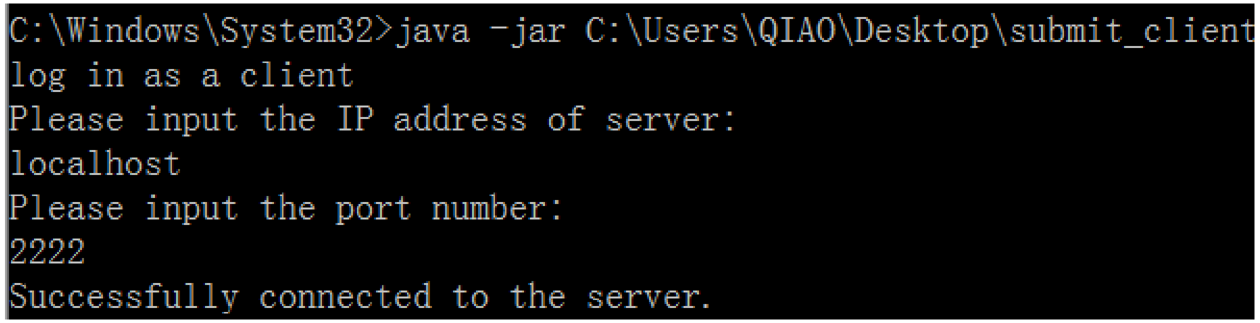


```
waiting for a to drop a piece
a exited the game. do you want to wait or not? enter WAIT to wait for a
ayer. enter EXIT to exit the game.
```

```
a, enter a field number, or HINT to get a hint, or EXIT to exit the game.
EXIT
your input has been read
```

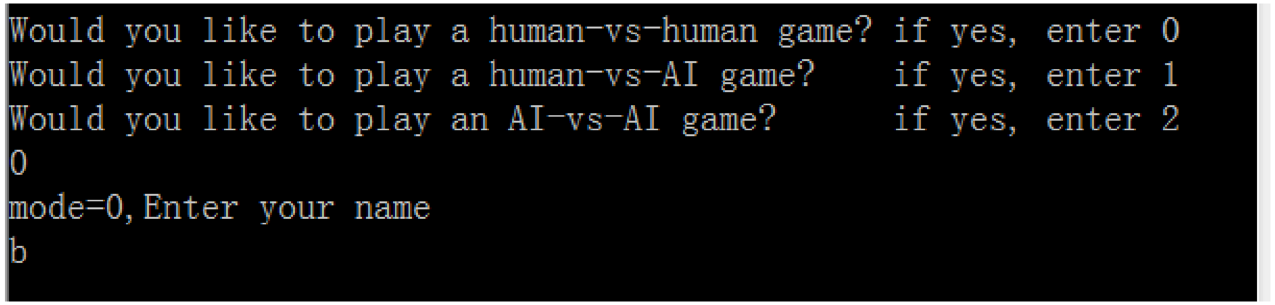
For the **client**, the following requirements should be implemented

1. The client should have a user-friendly TUI, which provides options to the user (**e.g.**, possibility to a enter port number and IP address) to request a game at the server.

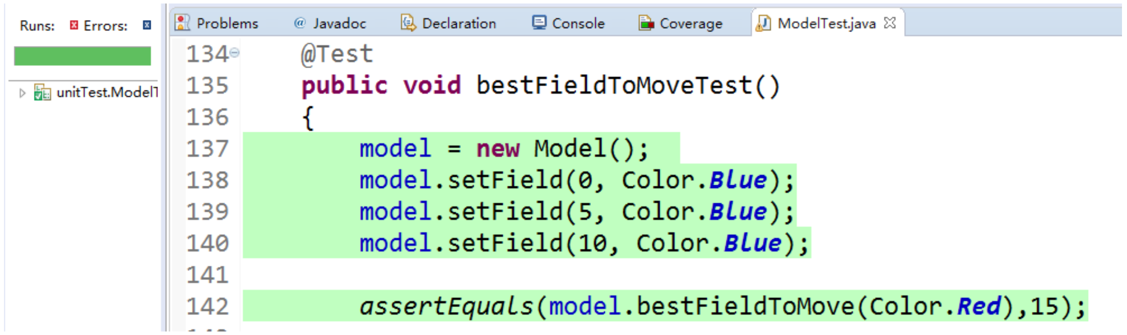


```
C:\Windows\System32>java -jar C:\Users\QIAO\Desktop\submit_client
log in as a client
Please input the IP address of server:
localhost
Please input the port number:
2222
Successfully connected to the server.
```

2. The client should support human players, and computer players with (some) artificially intelligent behaviour. The

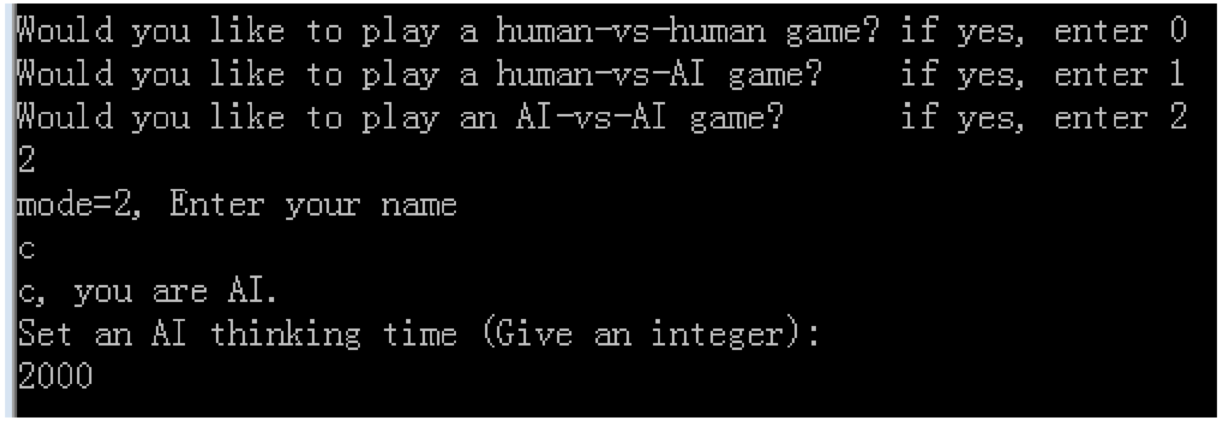


```
Would you like to play a human-vs-human game? if yes, enter 0
Would you like to play a human-vs-AI game?    if yes, enter 1
Would you like to play an AI-vs-AI game?      if yes, enter 2
0
mode=0,Enter your name
b
```



```
    @Test
    public void bestFieldToMoveTest()
    {
        model = new Model();
        model.setField(0, Color.Blue);
        model.setField(5, Color.Blue);
        model.setField(10, Color.Blue);

        assertEquals(model.bestFieldToMove(Color.Red),15);
```

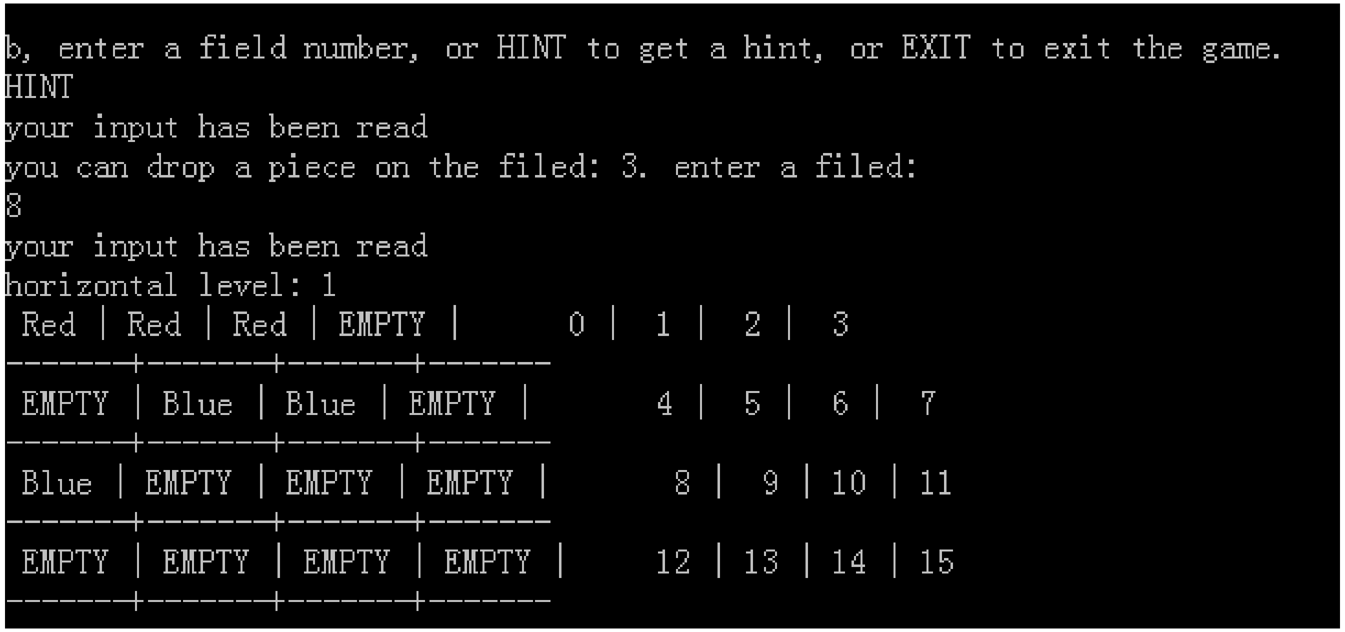
3. The thinking time of the computer player (and thus the power of the artificial intelligence) should be a parameter that can be changed via the client TUI.



```
Would you like to play a human-vs-human game? if yes, enter 0
Would you like to play a human-vs-AI game?    if yes, enter 1
Would you like to play an AI-vs-AI game?      if yes, enter 2
2
mode=2, Enter your name
c
c, you are AI.
Set an AI thinking time (Give an integer):
2000
```

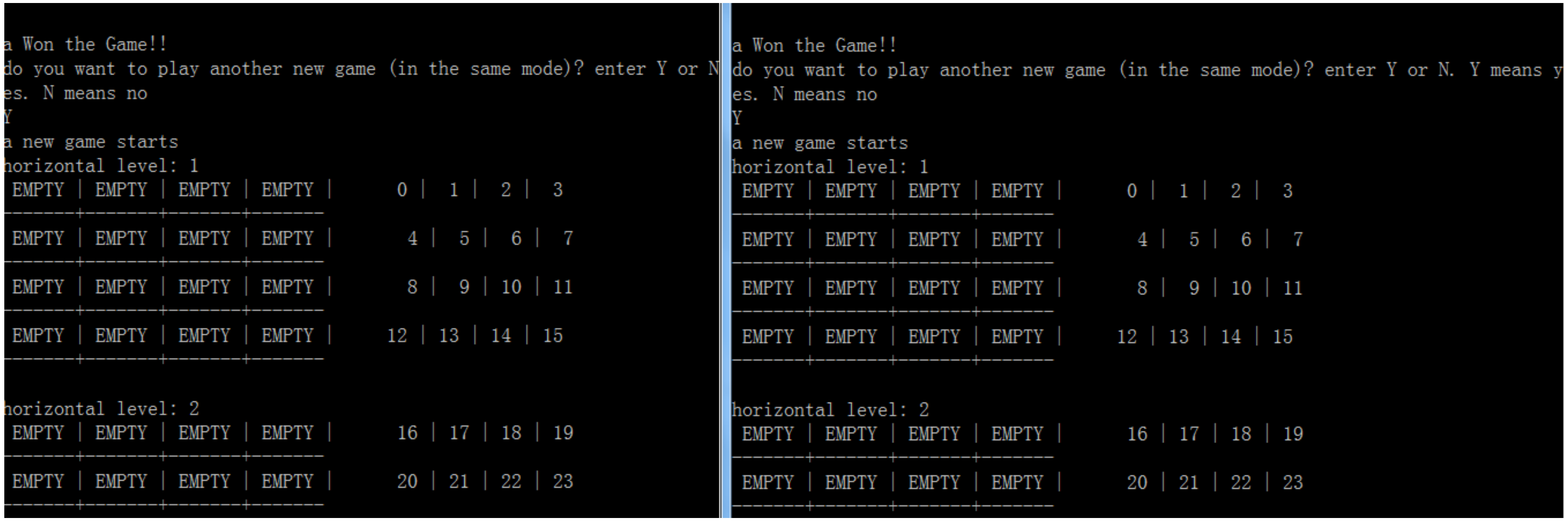
In human vs AI mode, the client who plays as AI needs to give an AI thinking time. In AI vs AI mode, both clients who play as AI needs to give an AI thinking time. The larger the input integer is, the longer the thinking time is. I have tried different cases. For example, AI_0’s thinking time is 1, AI_1’s thinking time is 2000. The result was: AI_0 moves very fast. AI_1 waits for a long time before making a movement. In the whole game, I saw the speed: fast, slow, fast, slow, fast, slow… This test was successful.

4. The client provides a **hint** functionality. This shows a human player a possible move, as indicated by the computer player. The move may only be proposed, the human player should have the possibility to decide whether to play this move, or make a different one.
This test is successful. Please see the picture bellow:



```
b, enter a field number, or HINT to get a hint, or EXIT to exit the game.
HINT
your input has been read
you can drop a piece on the filed: 3. enter a filed:
8
your input has been read
horizontal level: 1
 Red | Red | Red | EMPTY |      0 |  1 |  2 |  3
-------+-------+-------+-------
 EMPTY | Blue | Blue | EMPTY |       4 |  5 |  6 |  7
-------+-------+-------+-------
 Blue | EMPTY | EMPTY | EMPTY |       8 |  9 | 10 | 11
-------+-------+-------+-------
 EMPTY | EMPTY | EMPTY | EMPTY |     12 | 13 | 14 | 15
-------+-------+-------+-------
```

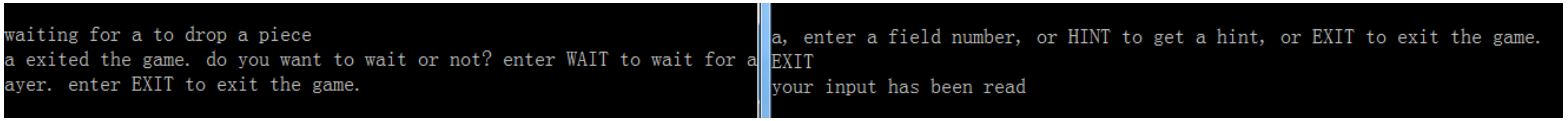
5. After a game is finished, the player should be able to start a new game.
This test is successful. Please see the picture bellow:



```
a Won the Game!!
do you want to play another new game (in the same mode)? enter Y or N
es. N means no
Y
a new game starts
horizontal level: 1
 EMPTY | EMPTY | EMPTY | EMPTY |       0 |  1 |  2 |  3
-------+-------+-------+-------
 EMPTY | EMPTY | EMPTY | EMPTY |       4 |  5 |  6 |  7
-------+-------+-------+-------
 EMPTY | EMPTY | EMPTY | EMPTY |       8 |  9 | 10 | 11
-------+-------+-------+-------
 EMPTY | EMPTY | EMPTY | EMPTY |      12 | 13 | 14 | 15
-------+-------+-------+-------

horizontal level: 2
 EMPTY | EMPTY | EMPTY | EMPTY |      16 | 17 | 18 | 19
-------+-------+-------+-------
 EMPTY | EMPTY | EMPTY | EMPTY |      20 | 21 | 22 | 23
-------+-------+-------+-------
```

```
a Won the Game!!
do you want to play another new game (in the same mode)? enter Y or N. Y means y
es. N means no
Y
a new game starts
horizontal level: 1
 EMPTY | EMPTY | EMPTY | EMPTY |       0 |  1 |  2 |  3
-------+-------+-------+-------
 EMPTY | EMPTY | EMPTY | EMPTY |       4 |  5 |  6 |  7
-------+-------+-------+-------
 EMPTY | EMPTY | EMPTY | EMPTY |       8 |  9 | 10 | 11
-------+-------+-------+-------
 EMPTY | EMPTY | EMPTY | EMPTY |      12 | 13 | 14 | 15
-------+-------+-------+-------

horizontal level: 2
 EMPTY | EMPTY | EMPTY | EMPTY |      16 | 17 | 18 | 19
-------+-------+-------+-------
 EMPTY | EMPTY | EMPTY | EMPTY |      20 | 21 | 22 | 23
-------+-------+-------+-------
```

6. If a player quits the game before it has finished, or the client crashes, the other player(s) should be informed, and the game should end cleanly. In this case, the other player(s) should be allowed to register again with the server for a new game.



```
waiting for a to drop a piece
a exited the game. do you want to wait or not? enter WAIT to wait for a
ayer. enter EXIT to exit the game.
```

```
a, enter a field number, or HINT to get a hint, or EXIT to exit the game.
EXIT
your input has been read
```

when two players (A and B) playing the game, one player (B) exit the game. The board is cleaned immediately. The game is finished. The other player (A) can choose to exit or to stay and wait for a new player to start a new game. If the player (A) wants to stay, then the server put this player (A) into the array list called WaitingPlayer. When a new player (C) comes in, if there is a player (A) who is willing to play the same mode game, then the server matches A with C as a pair. So a new game will start.
Each player has a private variable to store the connection status. The status can be get by other classes. My design is that if

the player disconnects, the other classes can know this player is disconnected by getting the status.

7. A server might at all times disconnect. The clients should react to this in a clean way, closing all open connections **etc.**
I learned two ways of checking this. One way is to send the urgent data before sending the actual data. When the server crashes, the urgent data will not be received. So the client knows that the server is crashed. But this way will slow down the speed of communication. The second way is to check whether br.read()!=-1. This only works for localhost, not on internet.
I wrote these methods in the class “Controller” and the class “GameServer”. The principle is correct. But I need more time to figure out how to insert them into the program. So they are not called by any classes.

8. The client should respect the protocol as defined for the tutorial group during the project session in Week 7.
The client is able to send messages to the server, receive the message to the server. The server broadcast client’s information to other clients.

Part D. Metrics report

WMC (weighted methods per class) shows the complexity of each class. It depends on the complexity of each method of the class. The number of methods and the complexity of methods is a direct predictor of how much time and effort it required to develop and maintain the class. I used WMC to test which classes are the most complex in the project. In my project, the most complex classes in the server are:
-class: Controller
-class: Model
-class: StarterHumanVsHuman

Part D. Reflection on Planning

My planning was very helpful. It makes me be alarmed when to finish what tasks. In the beginning two weeks, I almost followed my plan. I finished the MVC in time. My estimation in the networking stage was correct. I indeed spent a lot of time in implementing the TCP protocol. Because this part was hard for me.

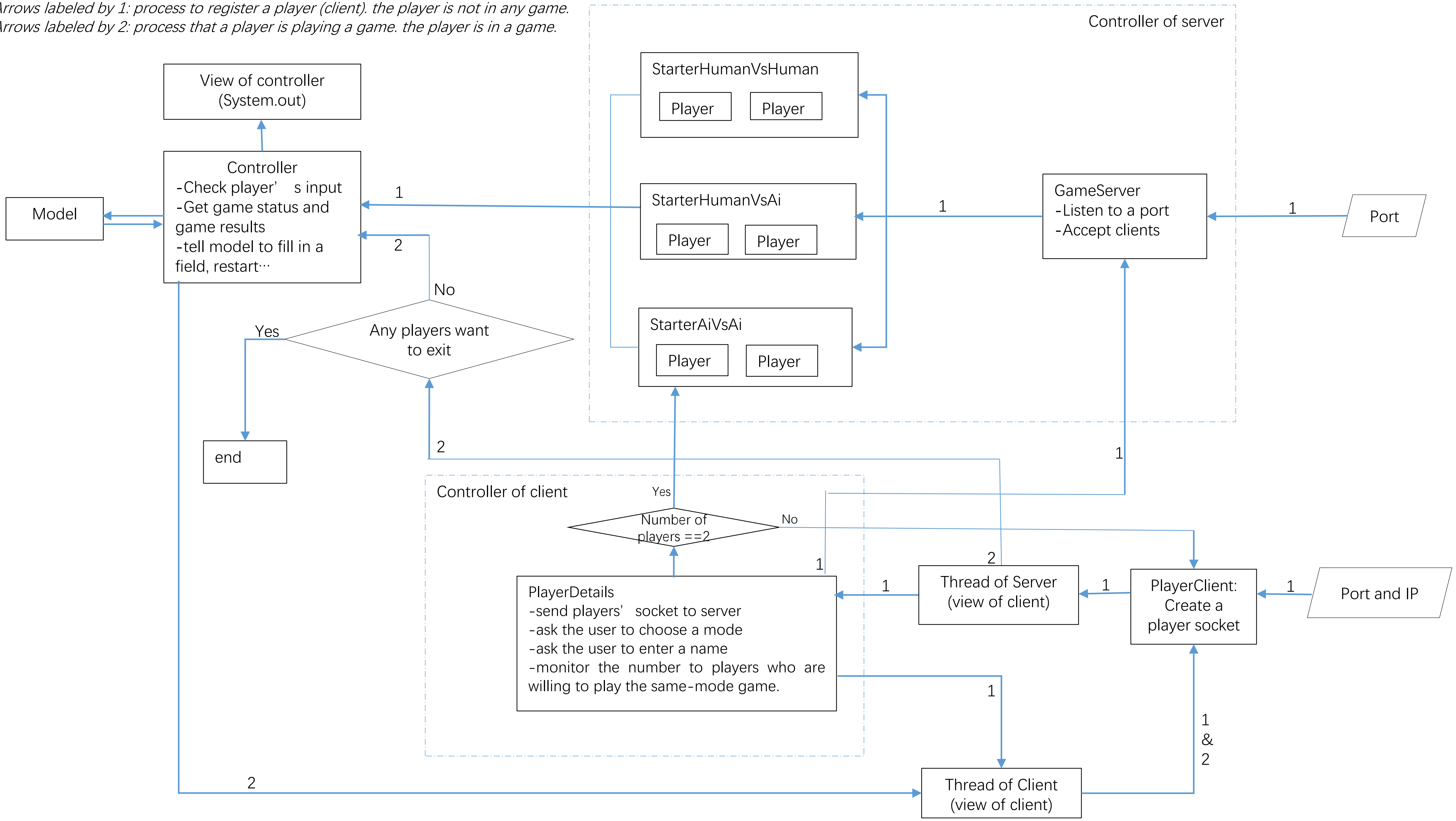
I made some adjustments in my plan during the project. Implementing the view took me more time than expected. I tried the TUI view in the beginning. Then changed it to GUI view. After that, I need to establish the TCP protocol. Because the protocol is complex, I feel that it might difficult to make buttons send information to the server. So I changed the view back to the TUI view. Then because of the time, I'd better move forward in solving networking problems, instead of repeating the view process. So I decided to leave the TUI view simple. I think it is good to take this countermeasure. Otherwise, I cannot finish the project in time.

An improvement can be that if I start from designing the whole structure of the project, I can save much time in changing the model and the controller. It is better to have a big picture in the beginning.

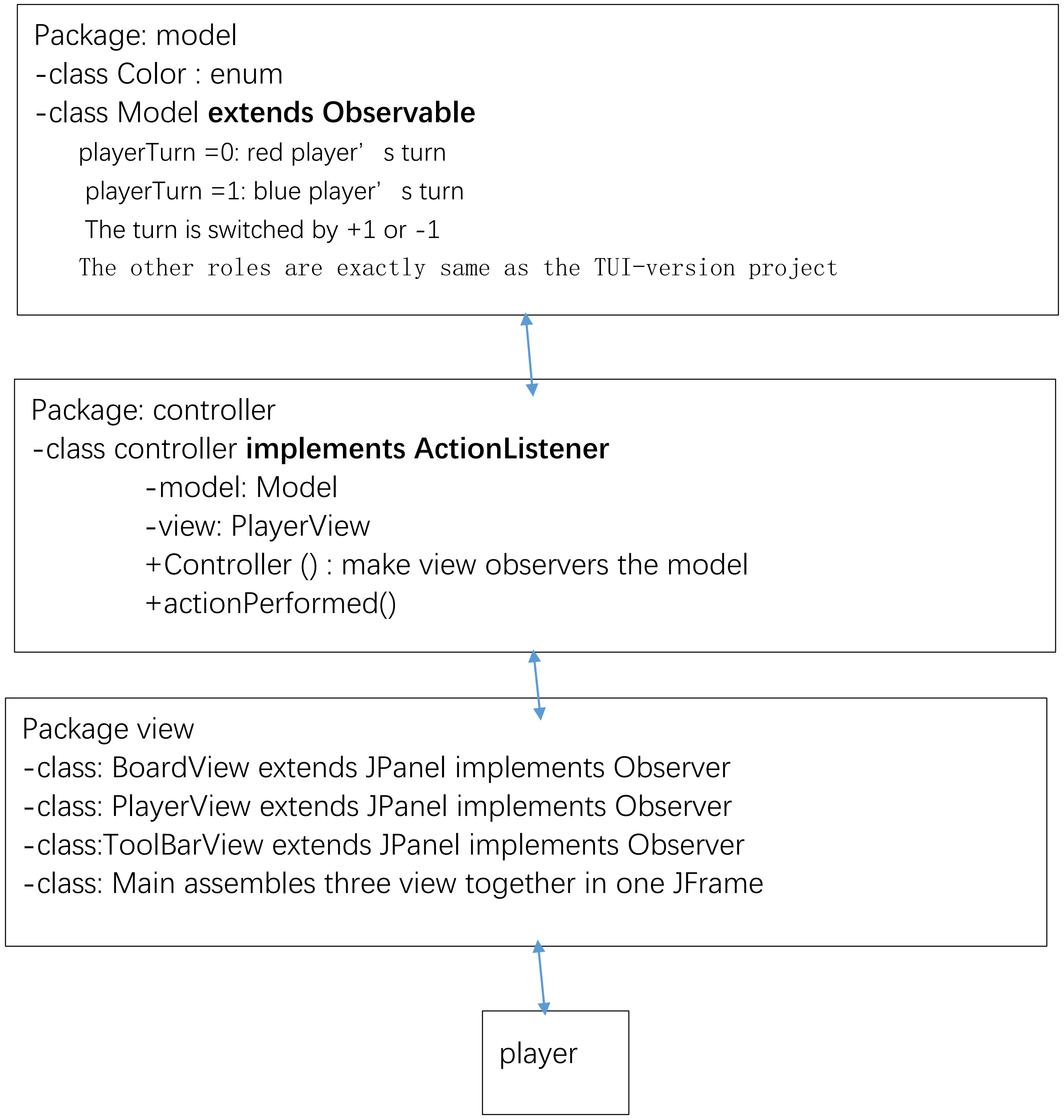
If I am next year's student assistant for this project. I would suggest that in the beginning of the project, draw a flow chart first. Having a big picture of the whole project is very important. It will make the implementation process highly efficient.

Appendix A: a systematic overview of the TUI-version project. Please see the nest page. *Arrows labeled by 1: process to register a player (client). the player is not in any game. Arrows labeled by 2: process that a player is playing a game. the player is in a game. The overview clearly shows the main role of the main classes and the relationship between different classes.*

Arrows labeled by 1: process to register a player (client). the player is not in any game.
Arrows labeled by 2: process that a player is playing a game. the player is in a game.



Appendix B: diagram of the GUI version project. Please see the nest page.



Package: model

-class Color : enum

-class Model **extends Observable**

playerTurn =0: red player’ s turn

playerTurn =1: blue player’ s turn

The turn is switched by +1 or -1

The other roles are exactly same as the TUI-version project

Package: controller

-class controller **implements ActionListener**

-model: Model

-view: PlayerView

+Controller () : make view observers the model

+actionPerformed()

Package view

-class: BoardView extends JPanel implements Observer

-class: PlayerView extends JPanel implements Observer

-class:ToolBarView extends JPanel implements Observer

-class: Main assembles three view together in one JFrame

Appendix C. The view in the GUI version project: there are three views. Their diagrams and roles are shown in the table below. the main class assembles three views together. The main class defines the layout of the frame, the location of each view, and the size of each view.   The class diagram of the view can be seen in the next page.

class: BoardView **extends JPanel implements Observer**

-tiles[][]: Jbuttons

-model: Model

-foreColoer: color

-backColor: Color

-playerView:Playerview

+BoarView():

create an array of buttons on the board

set the color of buttons before clicked and after clicked.

+registerControllers ():

add ActionListener() to each Jbuttons, each Jbutton will invokes a movement in the model

+update ()

shows the color on the button.

If this field is occupied by red player, the button shows " red " . Otherwise, the buttons shows "blue" .

if the game is over, shows which line has 4 pieces together.

class: TooBarView **extends JPanel implements Observer**

-newGameButtons: Jbutton

-chooseMode: Jbutton

-getHintButton: Jbutton

-guitGameButton: Jbutton

-label: Jlabel

-showHint: Jlabel

+ToolBarView():

create all the above buttons, set their texts, fonts, sizes, status

create all the labels, set their texts, fonts.

+registerControllers ():

add ActionListener() to each Jbuttons, each Jbutton will invokes a reaction in the model, such as start a new game, or change mode, or get hint.

+update ()

defines how the text on the mode button should be switched, after a click.

shows message to player what to do next.

shows whether anybody wins or it is a draw

class: PlayerView **extends JPanel implements Observer**

-player1: JRadiobutton

-player2: JRadiobutton

-group: ButtonGroup

-JTextField, Jlabel, and int for setting AI thinking time

+PlayerView():

set the location and size of the PlayerView

create two buttons. Set their size and status.

the button group makes two buttons contradict to each other

+registerControllers ():

add ActionListener() to each radio button. Set player' s turn based on which JRadioButton is clicked.

+update ()

change the player' s name according to the mode

if player wants to start a new game, clear the player turn

if game starts, update the player turn

set "AI thinking" text when AI is thinking