

Project: Online Go5 Game

Hoang Trinh (hoang@cs.uchicago.edu)
Wonseok Chae (wchae@cs.uchicago.edu)

1. Document History

- 4.7 – This document is originated from the project proposal.
- 4.29 – Initial design document was prepared by Trinh and later combined into this document.
- ?? – Project scope was changed.
- 5.25 – Mobile section was added.

2. Overview (from proposal)

Go5 (a.k.a. FiveStone or GoMoKu), a variation of Go, is played by two persons on a board with black and white stones. As its name implies, whoever gets five stones in a line in any direction wins the game. For example, the white won the game in the Figure 1.

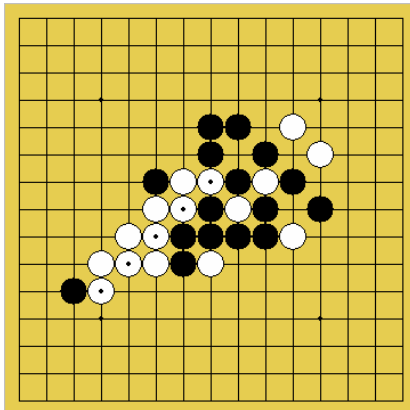


Figure 1



Figure 2

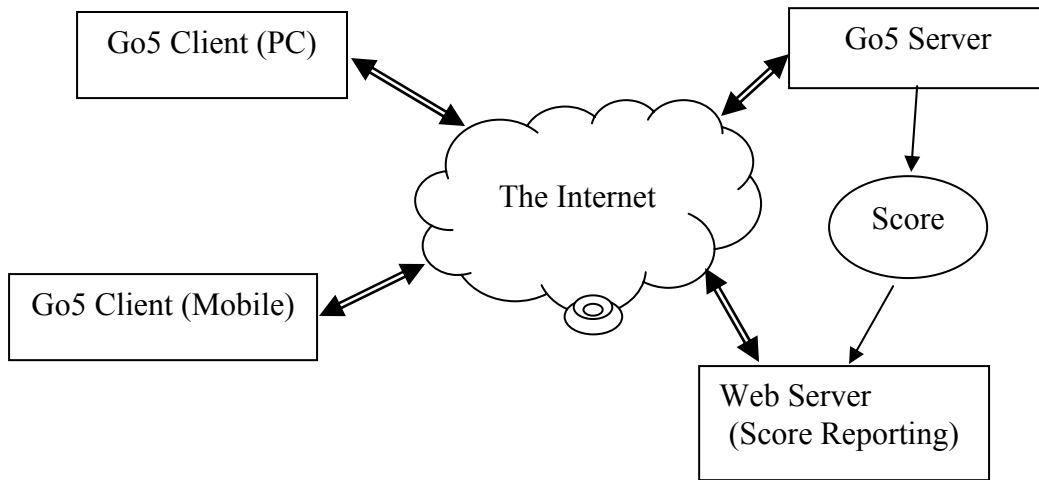
We are going to implement an online Go5 game with the following additional features:

- play games via the Internet
- support a ranking system which records the score of players
- support a coaching system which enables spectators to give advices to players
- provide AI players who plays against human players or another AI player

(Trinh, one of project members, had already implemented AI parts of this game. Therefore, we hope to integrate his AI program with this project.)

- support playing games with a mobile device
(We are investigating WIPI, a kind of MIDP (Mobile Information Device Profile) to emulate a mobile environment. For example, it would look like Figure 2.)

3. Scope of Project



There are five components in the project:

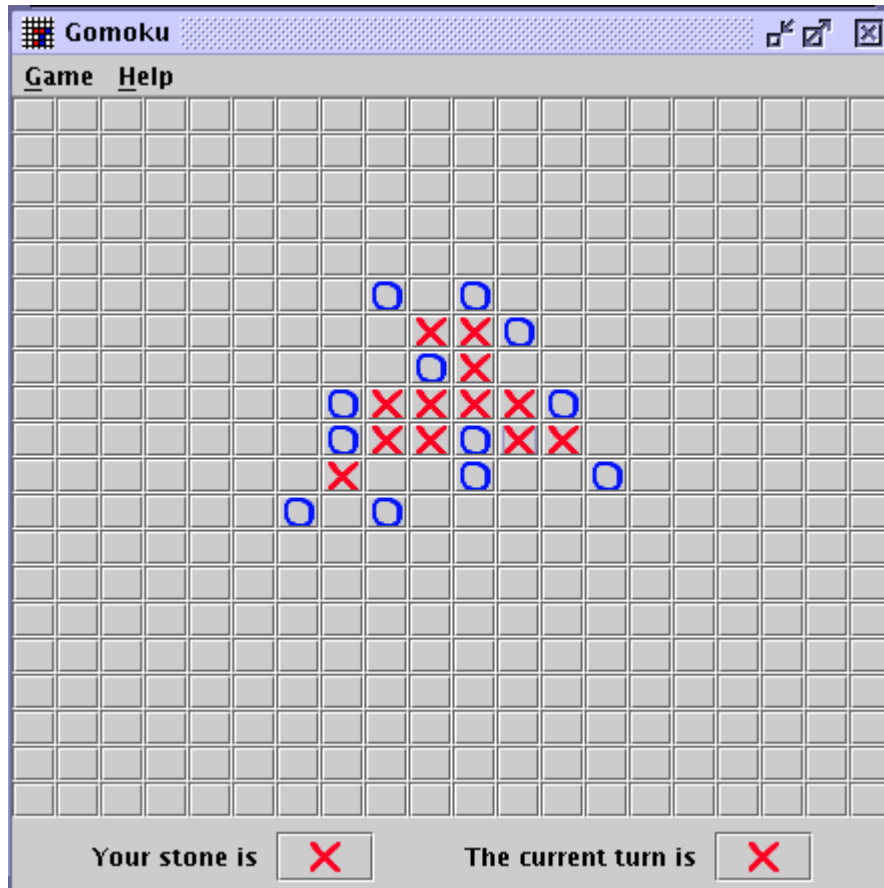
- Go5 Client (PC): provide reasonable user interfaces to the user using PC
- Go5 Client (Mobile): provide reasonable user interfaces to the user using a mobile
- Go5 Server: provide reasonable function that makes clients play games with each other and also support AI with which a client can play
- Web Server (Score Reporting): provide some mechanism to show score information. It will be some web pages
- Score: Text files or DB that contains score information

3.1. Modified Scope of Project

There have been several changes since we started the project:

- Instead of a coaching system, we decided to implement spectators to simplify the requirement.
- Using Trinh's AI implementation, we decided to use the open sources that implements Go5 AI games (because they also support fancy GUI, so we can only concentrate network functionality.)

4. Snapshot of Client (PC)



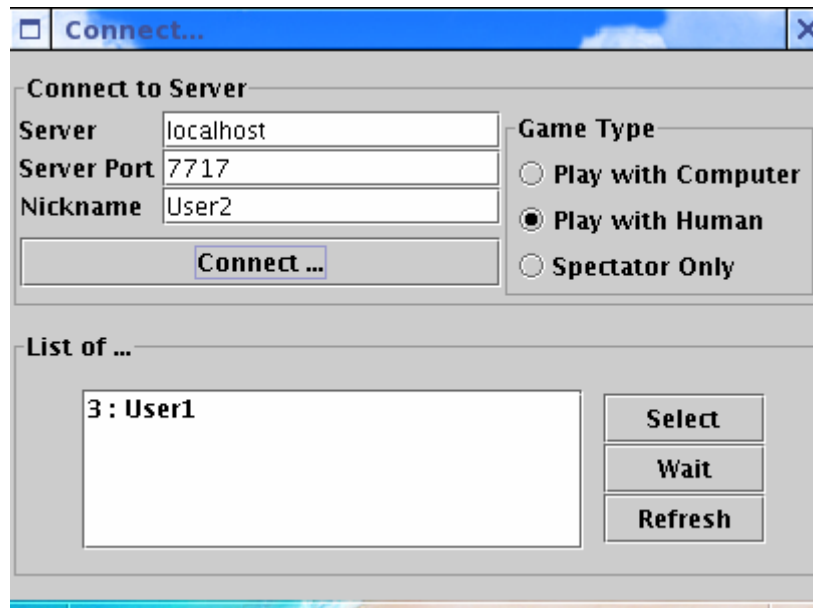
4.1. How to run it

- Server
 - `./gnomoku portnum`
- Client (PC)
 - `./make run`

After running it, you should set the configuration to connect the server by choosing "Connect to Server" in the Game menu.

 - Then, "Connect..." dialog will pop up. After filling "Server", "Server Port", "Nickname", and a proper game type, press "Connect..." button.
 - Based on the game type, the following will happen:
 - Play with Computer: the "Connect..." dialog will close and game with computer will start immediately.
 - Play with Human: A list of waiting players will be shown and you can select one with whom you want to play.

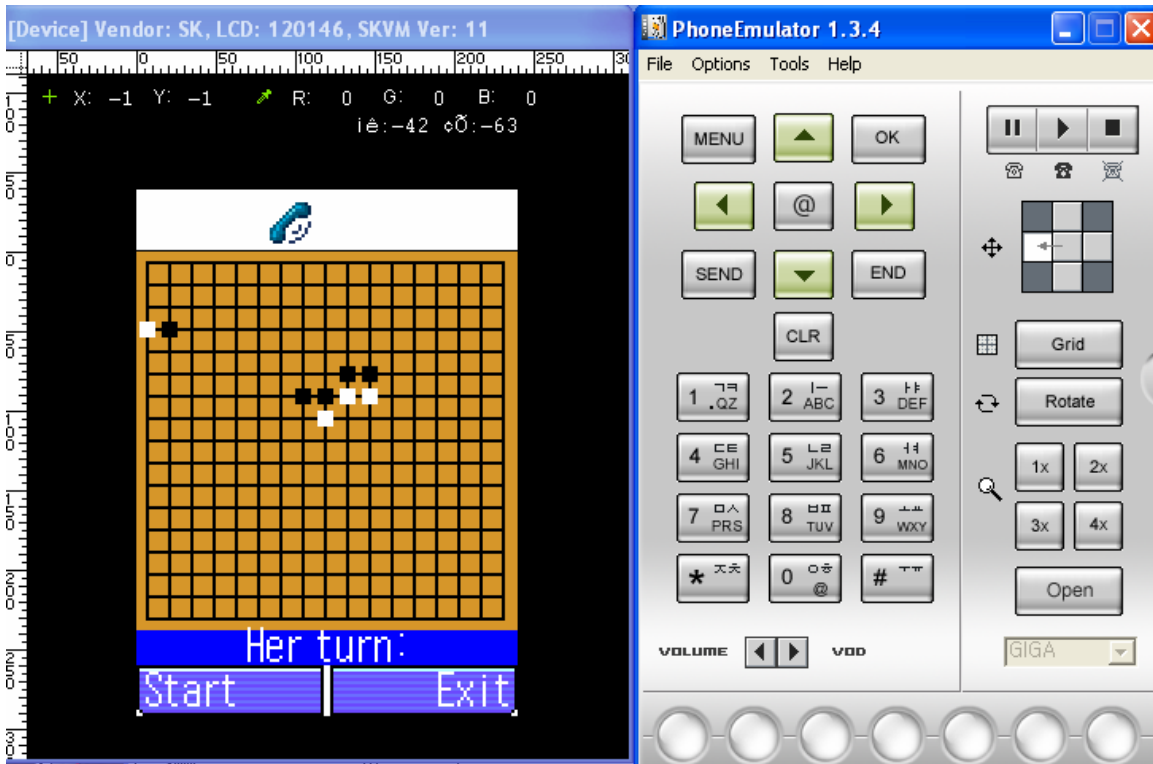
- Spectator Only: A list of playing games will be shown and you can select one that you want to watch.



- In the Game menu, there are several submenus:
 - Connect to Server : show the “Connect...” dialog
 - Disconnect : disconnect to server
 - New Game : start a new game with current players
 - Exit : quit this application



- Client (MOBILE)
 - We don't have any real mobile (which supports Java) around here, so we just used the emulator provided by XCE. You can get the emulator and manuals for it from their website (<http://dz.xce.co.kr/english/>).
 - Once launching the emulator (called PhoneEmulator 1.3.4), load Omok class and run it. Then, a Go5 board will be shown and a game will start by pressing "Menu" key on the virtual key pad. Arrow key will help you move your stone, and press "@" when you decide where to move.
 - The following figure shows how it looks like:



4.2. How to compile?

- The distributed file contains the following directories:

/server - contains files related to the server

- Makefile – make file
- src/gnomoku.c – modified to support network gaming
- src/robot.c – src for Go 5 AI and rarely modified

/client_pc – contains files related to the PC client

- Makefile – make file
 - make – compile
 - make run – run the client
 - make clean – clean all classes
- net/sourceforge/gomoku/
 - contains implementations of Go 5 game
 - the following files are the main files that we implemented or enhanced to support network playing
 - Gomoku.java – main file / handling messages
 - ListDialog.java = setting configuration
 - NetworkPlayer.java – virtual player for network

/client_mobile – contains files related to the mobile client

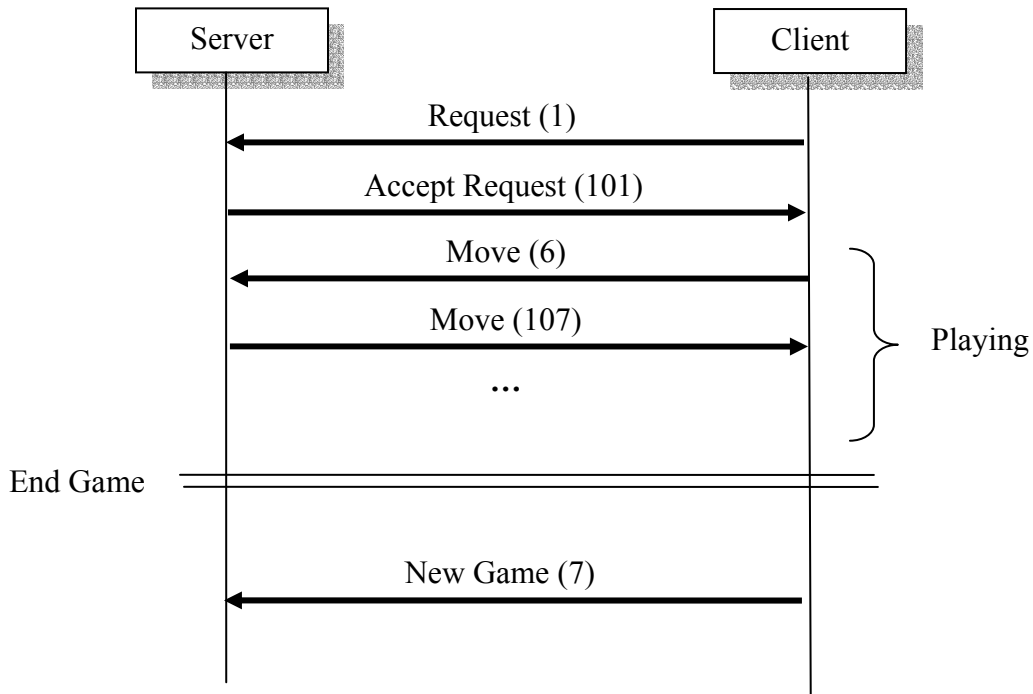
- to compile mobile code, you need to classes.jar provided by XCE. If available on “../classes/” directory, you can type like this:

```
javac -target 1.1 -bootclasspath ../classes/classes.jar *.java
```

- ServerConnector.java was modified to connect to the Go5 server.

5. Scenarios of the communications (Protocols)

5.1 A (PC) client connects to the server to play with the computer



1. When a user chooses a play with computer, “Request (1)” message will be sent to the server. The server will respond with “Accept Request (101)” message. Upon receiving “Accept Request (101)”, the game with computer starts with the first move of a user by sending “Move (6)” to the server. The server will respond with “Move (107)” containing computer’s movement.

1 byte	20 bytes	comments
Type = 1	player name	

1 byte	1	1	comments
Type = 101	User ID	Game ID	UserID and GameID is assigned to identify a user and her game.

1 byte	1	1	1	1	comments
Type = 6	User ID	Game ID	X	Y	Move(x,y)

1 byte	1	1	comments
Type = 107	X	Y	Move(x,y)

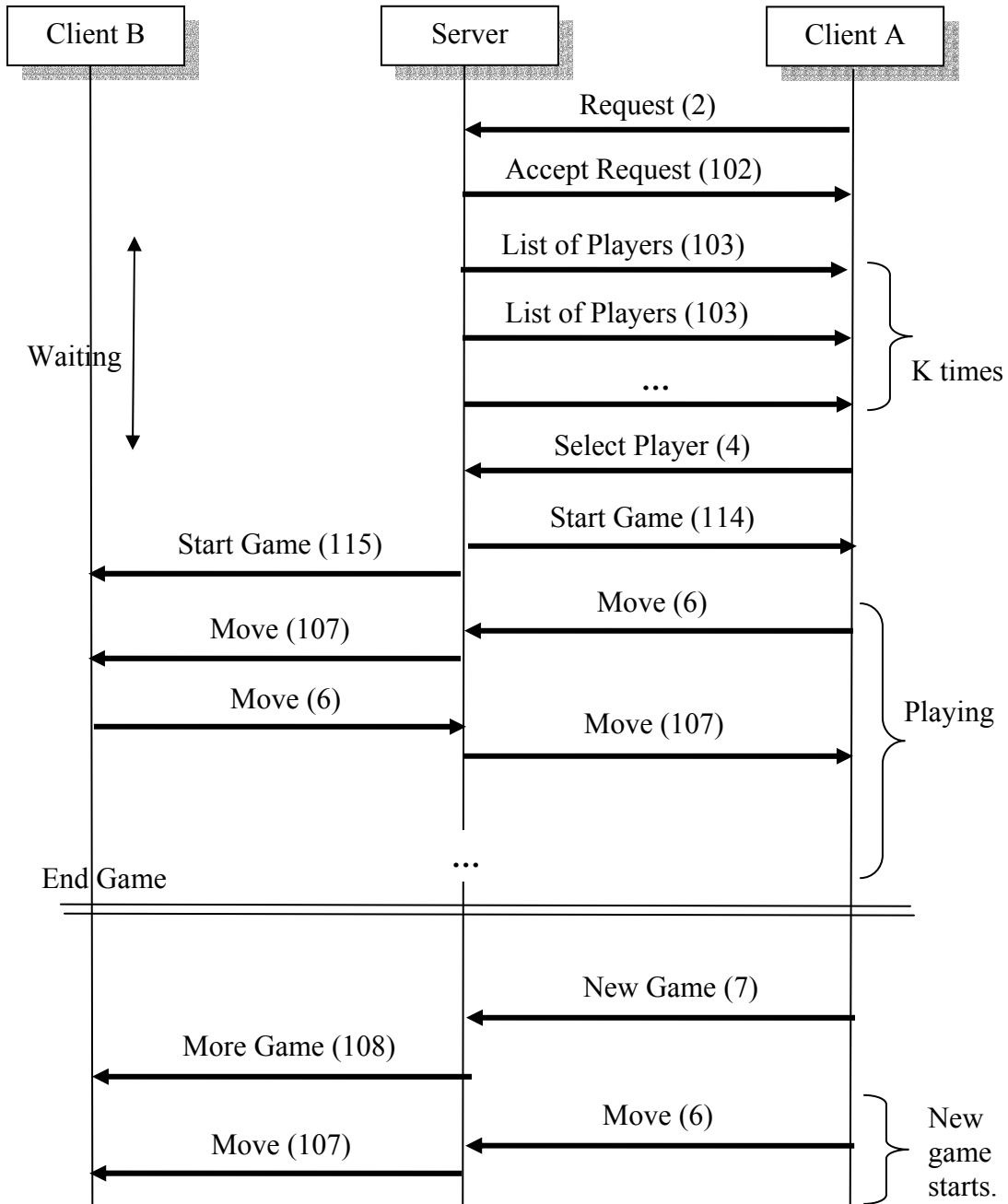
2. When the game is over, a user can start a new game with computer, choosing “New Game” in the Game menu. Then, “New Game (7)” message will be sent to the server. (Even during the game, a user can start a new game.)

1 byte	1	comments
Type = 7	User ID	

3. When a player wants to end a game, he can disconnect the connection to the server (even during a game) by choosing “Disconnect” in the Game menu. Then, “Request to Disconnect (8)” message will be sent to the server.

1 byte	1	comments
Type = 8	User ID	

5.2. A (PC) client connects to the server to play with human



1. When a user chooses a play with human, "Request (2)" message will be sent to the server. The server will respond with "Accept Request (102)" message followed by "List of Players (103)" messages.

1 byte	20 bytes	comments
Type = 2	player name	

1 byte	1	1	comments
Type = 102	User ID	# of waiting players (=K)	UserID is assigned to identify a user.

1 byte	1	20	comments
Type = 103	User ID	User Name	Information about players who is in the waiting list. There would be k messages.

2. (OPTIONAL) If there is no one in the waiting list, the server will send “No Player (113)” to the client and add this client in the waiting list automatically.

1 byte	1 bytes	comments
Type = 113	User ID	

3. Upon receiving “List of Players (103)”, a user can select the other player whom she wants to play with. Then, “Select Player (4)” message will be sent to the server.

1 byte	1	1	comments
Type = 4	User ID	User ID of player to play with	

4. (OPTIONAL) If the client does not like to play with the existing players and decide to wait for incoming players, she can send “Waiting (9)” for the server to add her into the list. While waiting, she can request “Refresh (10)” message to make the server resend the waiting list.

1 byte	1	comments
Type = 9	User ID	

1 byte	1	comments
Type = 10	User ID	

5. When the server receives “Select Player (4)” from the client A, it will send back “Start Game (114)” to the client A, and send “Start Game (115)” message to the chosen player (i.e, a client B) that a new game will be started with a client A. with the first move of the client A by sending “Move (6)”, the game will start. The server will send “Move (107)” to notify this move to the other player.

1 byte	1	comments
Type = 114	Game ID	

1 byte	1	comments
Type = 115	Game ID	

1 byte	1	1	1	1	comments
Type = 6	User ID	Game ID	X	Y	Move(x,y)

1 byte	1	1	comments
Type = 107	X	Y	Move(x,y)

6. When the game is over, a user can start a new game with the same player, choosing “New Game” in the Game menu. Then, “New Game (7)” message will be sent to the server. (Even during the game, a user can start a new game.) Then, the server will send “More Game (108)” message to the other player. The one who request the new game will move first.

1 byte	1	Comments
Type = 7	User ID	

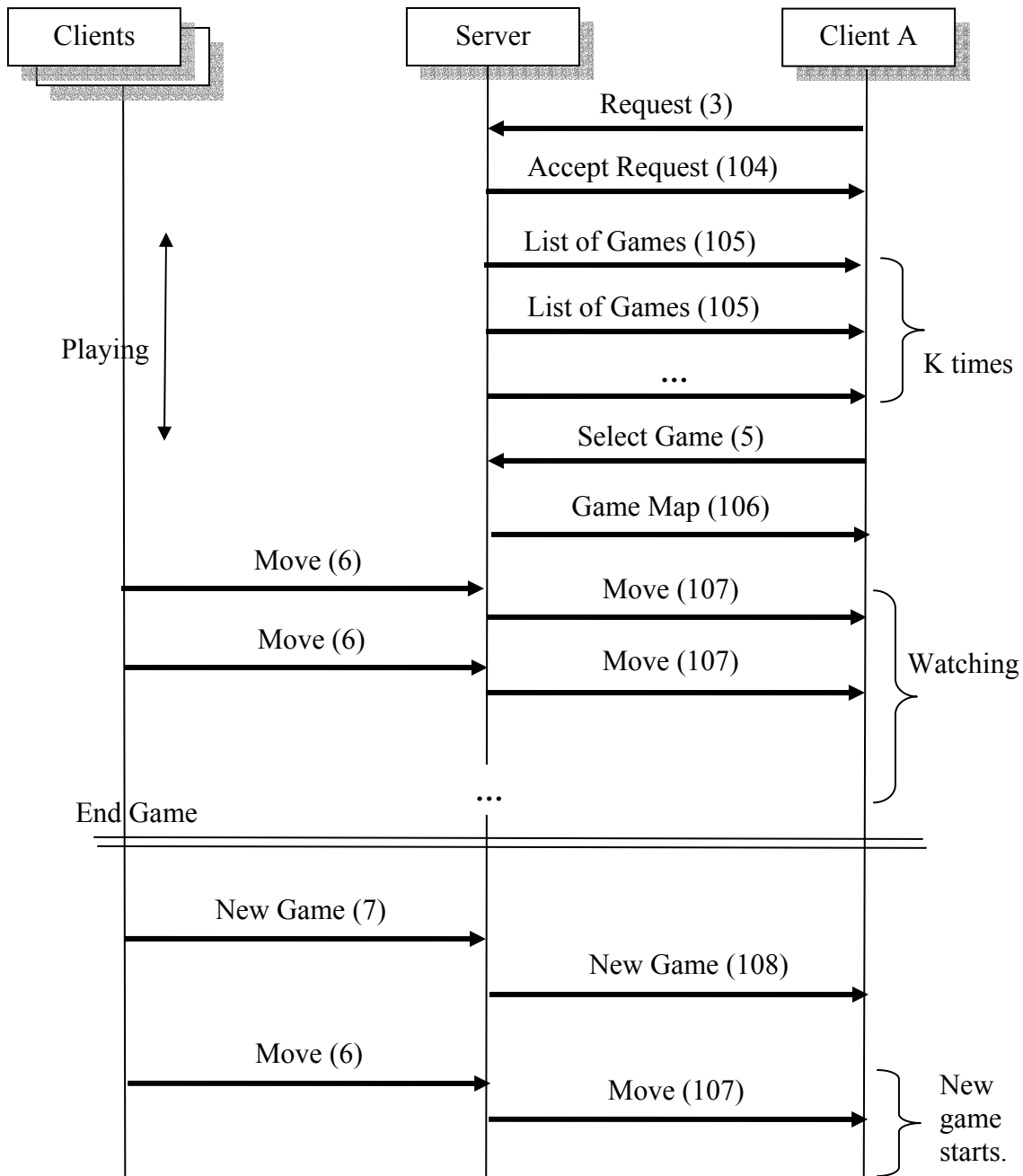
1 byte	Comments
Type = 108	The same Game ID will be used.

7. When a player wants to end a game, he can disconnect the connection to the server (even during a game) by choosing “Disconnect” in the Game menu. Then, “Request to Disconnect (8)” message will be sent to the server and server will notify this disconnection by sending “Disconnection (109) message to the other client including spectators. Even during a game, a user can choose “Disconnection” and the game will be finished immediately.

1 byte	1	comments
Type = 8	User ID	

1 byte	Comments
Type = 109	

5.3. A (PC) client connects to the server as a spectator



1. When a user chooses to be a spectator, “Request (3)” message will be sent to the server. The server will respond with “Accept Request (104)” message followed by “List of Games (105)” messages. Upon receiving “List of Games (105)”, a user can select the game that she wants to watch. Then, “Select Game (5)” message will be sent to the server.

1 byte	20 bytes	comments
Type = 3	player name	

1 byte	1	1	comments
Type = 104	User ID	# of current games (=K)	UserID is assigned to identify a user.

1 byte	1	20	20	comments
Type = 105	Game ID	User Name	User Name	Information about players in this game. There would be k messages.

1 byte	1	1	comments
Type = 5	User ID	Game ID to watch	

2. (OPTIONAL) If there is no game she wants to watch, then she can wait for a while. Then, she can request the server to resend the list of games later by sending “Refresh (11)” message.

1 byte	1	comments
Type = 11	User ID	

3. When the server receives “Select Game (5)” from the client, it will send back “Game Map (116)” which contains the current status of game and who is going to move next. After that, every move during a game will be reported via “Move (107)” message.

1 byte	1	1	17*17	comments
Type = 106	Game ID	Next Turn	Game Map	Next turn indicates who will move next.

1 byte	1	1	comments
Type = 107	X	Y	Move(x,y)

4. When the game is over, any user can start a new game. Then, “New Game (108)” message will be sent to the spectator.

1 byte	Comments		
Type = 108	The same Game ID will be used.		

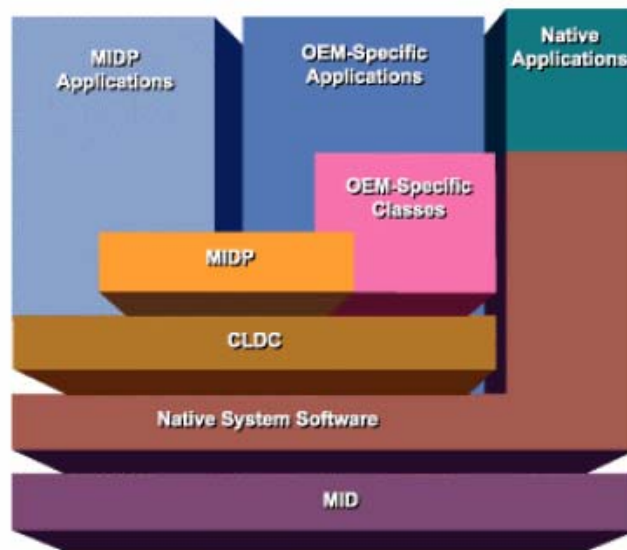
5. When any player ends a game, the server will notify this disconnection by sending “Disconnection (110)” message to the spectator.

1 byte	Comments
Type = 110	

6. Mobile

6.1. What is SK VM?

Sun Microsystems proposed J2ME (Java 2 Micro Edition) to enable Java application to be downloaded and executed on the mobile phone. Since then, many variation of J2ME have been implemented, and SK VM, one of them, has been developed by XCE to support SK wireless network in Korea. Because we have experience this platform (called SK VM) before, we chose it. We thought that minor modification enables our implementation to be reused on the other implementation of J2ME that follow the structure proposed by Sun Microsystems. In this figure, we are going to implement application on MIDP (Mobile Information Device Profile) which provide a kind of abstraction of a real hardware or an operating system.



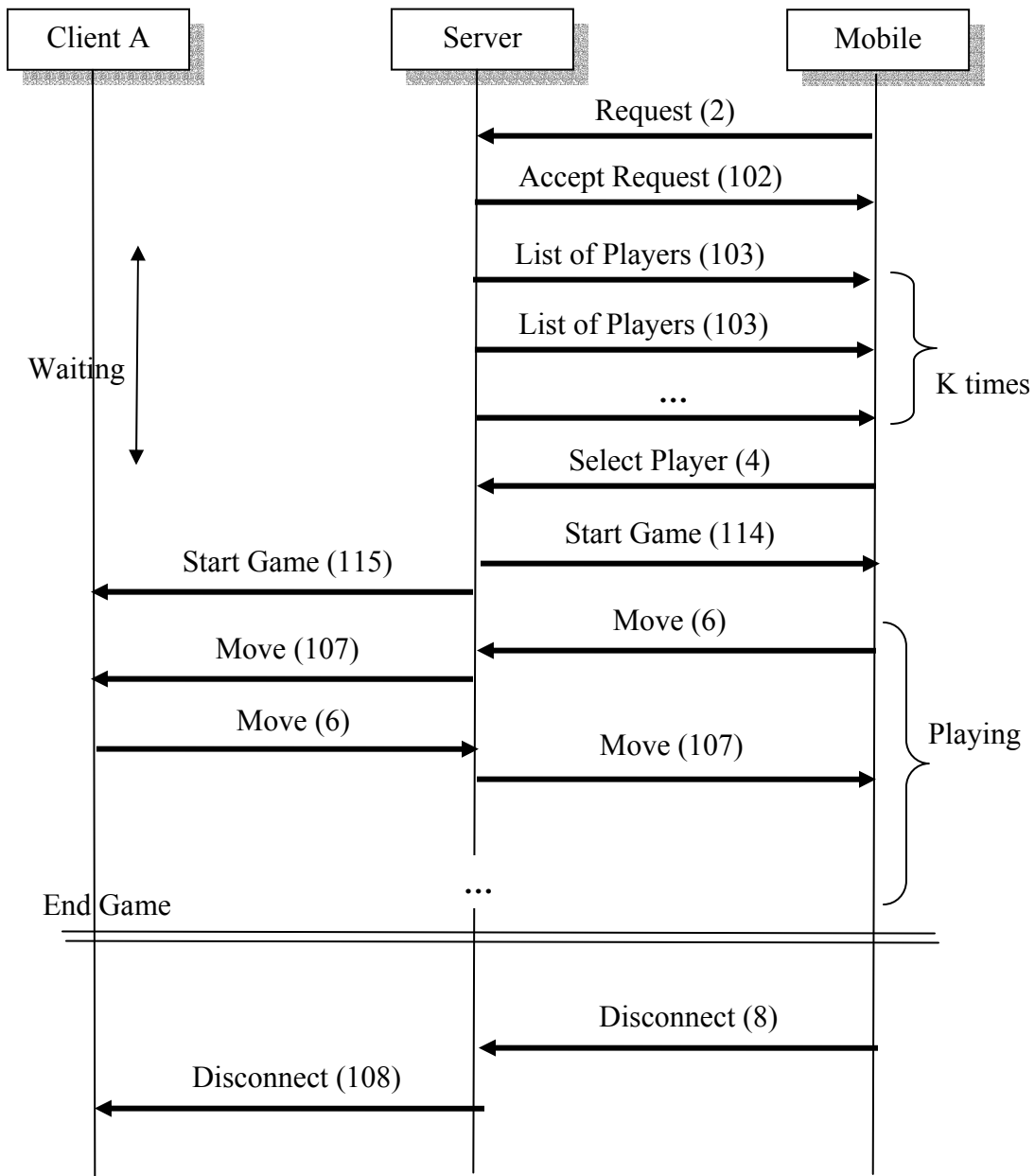
Structure of J2ME (from SK-VM's manual)

6.2. Design Assumption

To simplify implementation of this version of Go5 client, we made some assumptions:

- A mobile player always moves first (so, we don't have to implement second mover cases.)
- Information on server location, port number and nickname is assumed to be already known (so that we don't need to provide the fancy configuration dialog.)
- The "spectator only" case is not supported. When a mobile player tried to connect the server, the server tried to start a play with human if there is any one in the waiting list. Otherwise, the play with computer will start immediately. (Therefore, we don't need to provide a list of players or games, which requires a lot of efforts.)
- When a game is over, a mobile disconnects the connection to the server (so that we don't need to worry about the case where non-mobile clients try to start a new game with a mobile client.)

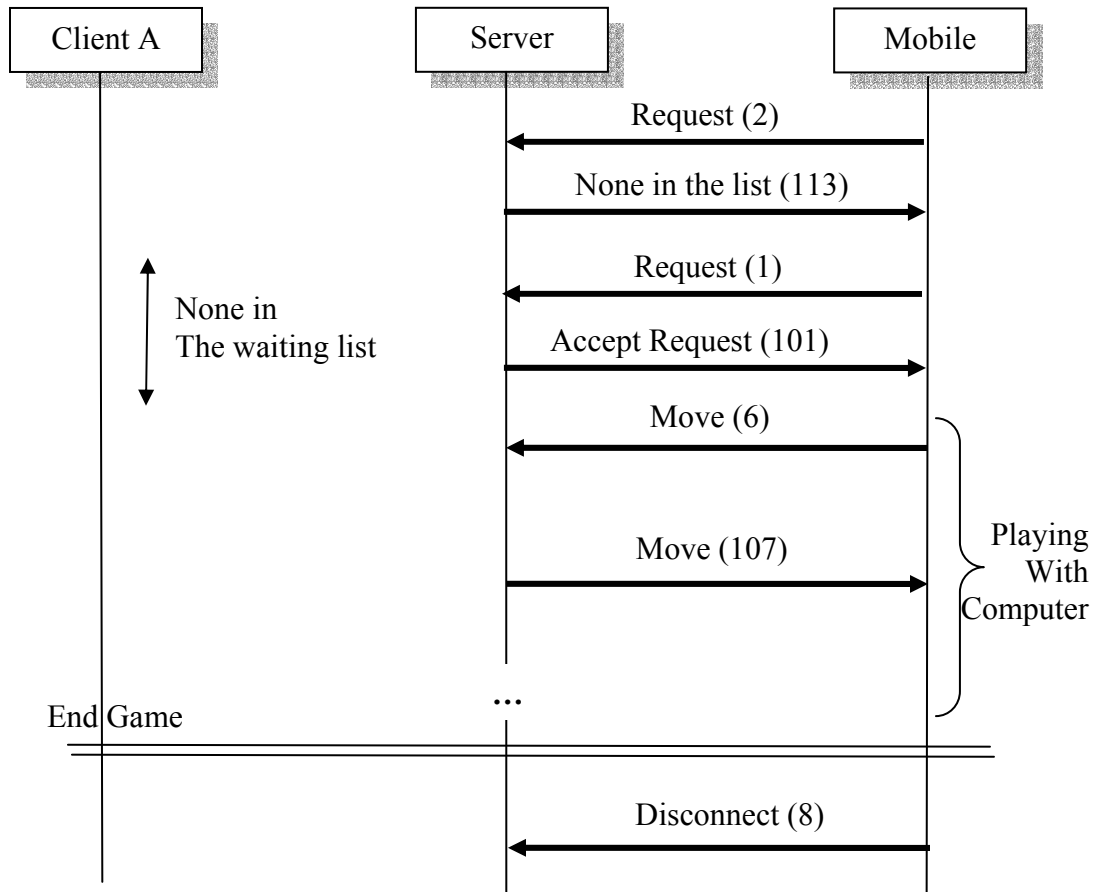
6.3. Simplified version of scenarios that a mobile connects to the server



- 1 When a mobile user connects to the server by sending “Request (2)”, the server will respond with “Accept Request (102)” message followed by “List of Players (103)” messages.

- 2 Upon receiving “List of Players (103)”, a mobile user chooses the first one in the list, and “Select Player (4)” message will be sent to the server.
- 3 When the server receives “Select Player (4)” from the mobile user, it will send back “Start Game (114)” to the mobile client, and send “Start Game (115)” message to the chosen player (i.e, a PC client that a new game will be started with the mobile user. with the first move of the mobile user by sending “Move (6)”, the game will start. The server will send “Move (107)” to notify this move to the other player.
- 4 When the game is over, a mobile user can start a new game by disconnecting the current connection. It comes from the design assumption we made to simplify the implementation. Therefore, “Request to Disconnect (8)” message will be sent to the server and server will notify this disconnection by sending “Disconnection (109) message to the other client including spectators.

6.4. Simplified version of scenarios that a mobile connects to the server (in case of none in the waiting list)



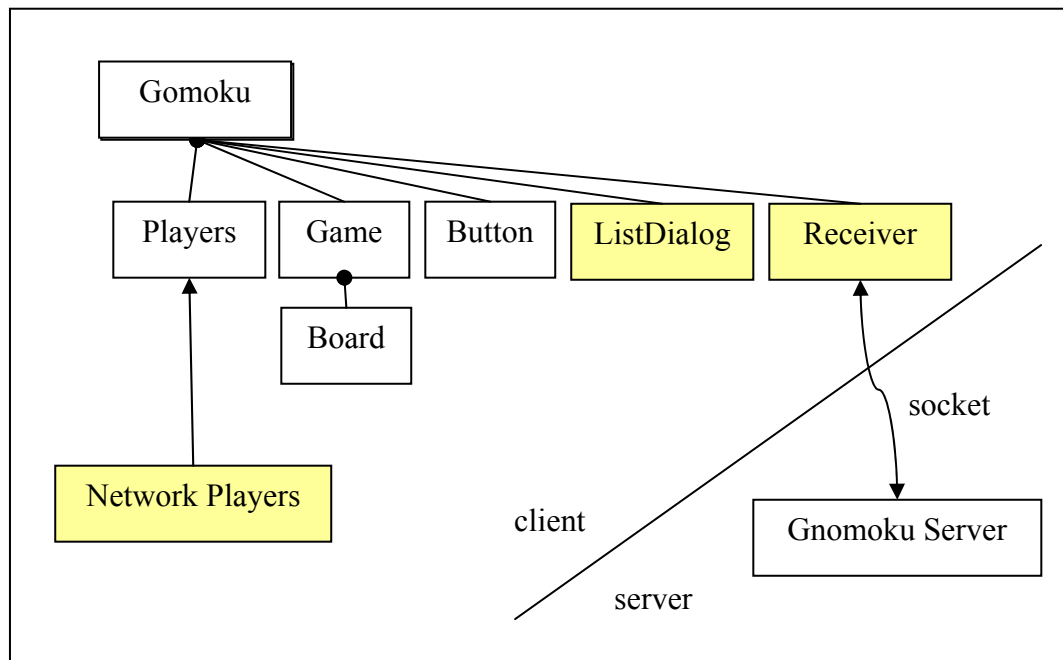
- 1 When a mobile user connects to the server by sending “Request (2)”, the server will respond with “Accept Request (102)” message followed by “List of Players (103)” messages. However, there is no one in the waiting list. So, the server will send “No Player (113)” to the client. Then, a mobile user sends “Request (1)” message to server so that she can play a game with computer avoiding waiting for incoming players.
- 2 When the server receives “Request (1)” from the mobile user, it will send back “Accept Request (101)” to the mobile client, and the game with compute will start with the first move of the mobile user.
- 3 When the game is over, a mobile user can start a new game by disconnecting the current connection. It comes from the design assumption we made to simplify the implementation. Therefore, “Request to Disconnect (8)” message will be sent to the server and server will notify

this disconnection by sending “Disconnection (109) message to the other client including spectators.

7. Implementation

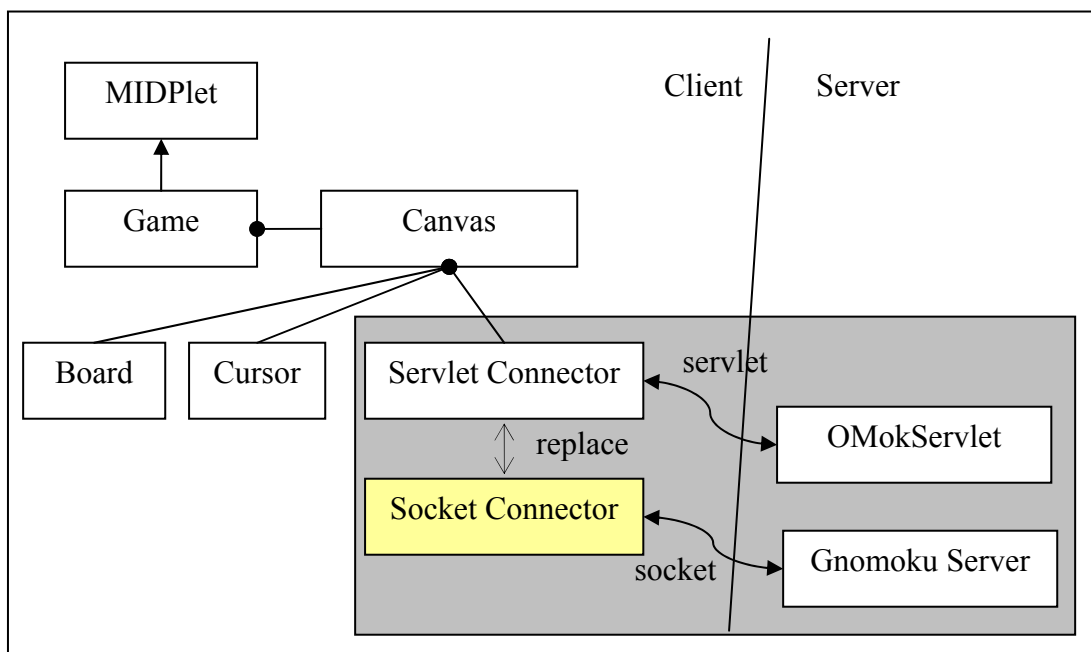
Actually, we reused the following implementation of Go5 and extended it to support on-line gaming.

- **Gnomoku:** A GTK+/GNOME version of Go5 game which was implemented and distributed by Robert V. Schipper under the GNU Public License. They provided AI functionality of this game. So, we have implemented the server side based on their AI function (called robot). It was version 0.5 and one of their future plans is to support two players' playing the game over the network and that is what we did.
- **Gomoku-1.0:** A Java version of Go5 game which was implemented and distributed by Douglas Ryan Richardson and Anton Safonov under the GNU Public License. It provided a fancy GUI written in Java that we needed, but does not support the playing over network. One of the reason that we chose it was that we simply could port it to the mobile version because we should implement socket communication in the mobile version by using Java. As shown in the following figure, "network player" class was introduced to handle the player which is either computer or the other player over network. "ListDialog" class collect configuration like server name, port number, nickname or various game configurations. "Receiver" class is a thread that handles incoming messages from server and parses them.



Structure of the client side: how to support playing over network

- **Omok:** A mobile version of Go5 game which was distributed anonymously in the site of www.mobilejava.co.kr where shares the technique of mobile programming in Java. Originally, this program support two playing over the internet, but through servlet technology (briefly, server side-applet programming), not socket. We replace all servlet parts with socket communication. Then, we made it communicate with the Go5 server that we just implemented. Now, our implementation (or enhancement) can support both playing with computer and playing with human). In the following diagram, we introduced “socket connector” class to support socket communication. That was our main task in this implementation even though there were several modifications to the existing classes.



Structure of the mobile client side: how to replace servlet with socket

8. Test Case

Configuration:

- Server: run on tti193.uchicago.edu:7717
- Client A (PC): run on hoang.uchicago.edu (User1)
- Client B (PC): run on 128.135.191.159 (User2)
- Client C (Mobile): run on 128.135.191.159 (Mobile)

ID	Description	Expect	Result
Play with computer			
1	Start the server.		
	Client "A" connects to the server with a play with computer.		
	When a game is over, try one more game		
	When a game is over, just stop by saying "no".		
	During playing, client "A" chooses "Disconnect"		
Play with a human			
2	Start the server.		
	Client "A" connects to the server with a play with human. Then, wait.		
	Client "B" connects to the server with a play with human.	"A" should be in the list.	
	Client "B" chooses "A" to play with.	Game will start with the first move of "A".	
	When a game is over, "B" chooses "New Game"	Game will start with the first move of "B"	
	During playing, client "A" chooses "Disconnect"		
Watch the game			
3	Start the server.		
	Client "A" connects to the server with a play with human. Then, wait.		
	Client "B" connects to the server with a play with human.	"A" should be in the list.	
	Client "B" chooses "A" to play with.	Game will start with the first move of "A".	
	Client "C" connects to the server with a play as a spectator.	Game list including b/w "A" and "B" will be shown.	
	Client "C" chooses the game b/w "A" and "C".	Game will be shown on the board of "C"	

	When a game is over, "A" chooses "New Game".	Game will start with the first move of "A".	
	During playing, client "C" chooses "Disconnect".		
(Mobile) Play with computer			
4	Start the server.		
	Client "C" connects to the server.	Game will start.	
	After a game is over, press "Enter" to start a new game.	Game will start.	
(Mobile) Play with a human			
5	Start the server.		
	Client "A" connects to the server with a play with human. Then, wait.		
	Client "C" connects to the server.	Game will start with the first move of "C".	
	A game is over.	"A" will receive "Disconnect" message from the server.	

9. Comments

- Difficulties with Java and C/C++: at the first stage, there were troubles to make “Gnomoku” server and “Gomoku” clients work together. What we did not expect was the fact that there are inconsistencies between Java and C/C++ implementation like the followings:
 - o Java can not use “ObjectInputStream” and “ObjectOutputStream”. Only “DataInputStream” and “DataOutputStream” are allowed to send and read data to (or from) socket that is implemented by C/C++.
 - o C/C++ may not send messages including “structure”. Java can not predict the size or layout of those messages. (It is true even between C/C++ in the different architectures.)
 - o Byte order should be considered. We just decided to send every data in the form of byte streams to avoid this problem.

10. Know Bugs or Future Works

- A (PC) client sometimes lost the opponent’s move. One of the reasons we guess is that after receiving message from server, we delivered it to the “network player” class via Java event mechanism. However, it might be lost very rare case. However, the result is very severe and the game seems just stuck. In this case, we might need to start a new game. This bug should be resolved to improve reliability.
- There is no checking mechanism so far. For example, what if one of players just disappears without any notice (like a disconnection message)? Definitely, “IsAlive”-like checking mechanism will be needed for more stable playing.
- The server currently records the scoring information in the form of a text file. However, we have not provided any fancy way to retrieve this information. Our intention was to provide web page for this, but we could not make it.