TÉCNICO
LISBOA

Instituto Superior Técnico

Engenharia Eletrotécnica e de Computadores

Redes Móveis e Sem Fios

2015/2016 2nd Semester

Preliminary Report

Development of Wi-Fi based multiplayer game as Android App

João Pedro Gonçalves Lages no. 75286

Vassiliki Poulaki no. 85404

Table of Contents

Brief description of the Android application

For the project, we decided to develop an Android game, based in a card game called “Agony”, that will be explained ahead.

In this report, we will do a quick explanation on how the flow of the game will be implemented. Note that we are still developing the application, so maybe changes on these thoughts will still be made. Then, we will describe how the game is played and the rules of it.

Agony is a Greek card game which is played by 2 to 4 players (or even more, but with more than one set of cards required, according to the number of players), and is similar to the well-known game “UNO”. In the beginning of the game, seven cards are dealt to each player and one card is placed face up on the table consisting a pile, while the rest of the cards are placed face down in a pile on the table. The main purpose of this game is to end up with as few cards as possible and the game is over when a player, who is the winner, runs out of cards. The players play in turn, one card at a time, placing it on top of the face up card creating a pile, according to the general rule that the card must be either the same color or the same suit with the top face up card. Players are allowed to pick one or more cards from the face down pile, and add them to their cards. If that pile runs out of cards, then all the cards of the face up pile, except for the top card, are mixed and placed as a new face down pale.The game has the following rules:

- A player can pass (without playing a card), only after picking at least one card from the face down card pile.
- When a player uses a card of number 8, has to play one more card according to the game rules.
- When a player uses a card of number 9 the next player lose their turn.
- When a player uses a card of number 7, the next player has to pick up 2 cards from the face down card pile. If the next player plays also a card of number 7, then the next player has to pick up 4 cards, and so on, until one player has no card of number 7. Even if a player has picked up 2 (or 4 or 6 e.t.c) cards because the previous player/players used a seven, but still has no proper card to play, they must pick up at least one card from the pile before passing. If a player has picked up cards because the previous player/players used a seven, and one of the cards he picked is a seven, then it counts as the first seven in a row.
- If a player uses an ace, he/she can redefine the “current color” of the face up pile (regardless of the ace’s color). An ace card cannot be played after another ace card has just been played. A player cannot finish a game (play his/her last card) with an ace.

In the end, the points are calculated for each player by adding the values of the cards that they end up with. The least points a player has, the highest rating he/she takes.

Introduction

This project consists of 3 android activities: the MainActivity, the PlayGame and the Winner, with the corresponding xml layout files (activity_main, activity_play_game, activity_winner correspondingly).

There is also an additional xml file, the simple_list_item, which is used to describe a ListView’s items that are programmatically added to the list.

Finally, the project contains 14 more java classes, some of them extending native java classes or implementing existing interfaces, which are used to implement several network facilities, to assist to the logical implementation of the game, and provide encapsulation and Object Oriented programming Style to the project.

The java classes are the following: Game, PlayerState, Card, MyPeerListListener, MyBroadcastReceiver, MyConnectionInfoListener, ServerAsyncTask, ClientAsyncTask, AsyncTask3, AsyncTask3Server, ServerSendsThread, ClientSendsThread, ClientThread , BroadcastToAClient.

A sort description of the most important activities and classes is following. The rest will be addressed later.

Card.java

This class extends the ImageView class (android) and has only one additional field, the id, which is a String that describes every card of a typical 52 cards set, and contains an English letter for the suit followed by a number for the card's number.

For example, an ace of spades will get "s1" as id, a jack of diamonds will get "d11", and so on. Card's instances are used to depict every card in the user interface with a functional way that can also manipulate these objects.

Game.java

A fundamental class for this project. Contains fields for all the elements that constitute an 'agony' game, and several methods that process them. Some of these fields are: a list with all players' names, a list with all players' current number of cards, stacks for the face down and the face up piles, an integer value that indicates whose turn is it, and many more.

At the creation of a new Game object, among the other elements that are being initialized, a (shuffled) new face down pile is being created (containing a full 52-card set), and its first element is being popped from the stack in order to be pushed into the face up pile stack.

Furthermore, the class includes a method that deals the cards to each player, by popping them from the face down pile, and another method that picks a new card for a player (from the face-down pile) by popping the card from the stack and returning it, while if that face-down pile runs out of cards, the method takes all the cards from the face-up pile, except for the upper one which consists now the new face-up pile, shuffles them and adds them to the face –down pile (according to the official rules of the game). All the card related lists are of String type, as the cards are represented by Strings as described before (and only when they are supposed to be displayed on the user interface Card objects are created).

The Game class implements the Serializable interface, so that its objects are exchangeable via the sockets. Every time a new game is being started, a new game object is created in order to represent it, and it is supposed to be public to and common for all players, and, consequently, it is must be shared among all the android devices that participate that round of playing. At every 'turn' of the game, the player that just finished their moves is supposed to broadcast the updated game object to the rest of the players, who are assigning this value to their game variable.

PlayerState.java

This Class's objects are private for each android device, and represent the elements of the game that are unique for each player, like their personal card set. In this class there are several methods that facilitate some operations related to each players cards, like adding a new card to it, removing a card from it (because the player played that card), search for a specific card at it, and so on.

The cards here, in the personal card list, are also represented by the same type of String values as described before. A PlayerState object is created for each player in the beginning of a game.

MainActivity

The beginning activity of the application.

PlayGame

The activity where the game takes place. It depicts all the public and private game elements, as the face up pile and the personal card set, and updates them accordingly, using the real values of these elements as they are stared in the appropriate class objects mentioned before.

The PlayGame activity has a Game object, and a PlayState object as class variables, and initializes them in its onCreate method. It also contains user interface (Views) variables that are being connected with the corresponding xml file in the creation of the activity, and are used to upda.

The activity includes several methods that update the user interface when needed (every time, for example, that a player makes a move, like picking a new card, or when a player receives a wifi message from another player that has completed their turn).

Wifi Peer-to-Peer

Small description

The Wi-Fi peer-to-peer (P2P), known as Wi-Fi Direct as well, allows applications to connect to nearby devices without needing to connect to a network or hotspot.

Wi-Fi Direct negotiates the link with a Wi-Fi Protected Setup system that assigns each device a limited wireless access point. The "pairing" of Wi-Fi Direct devices can be set up to require the proximity of a near field communication, a Bluetooth signal, or a button press on one or all the devices.

This is useful for our multiplayer application in order to make the connection between multiple devices, as we are going to explain below.

Set Up a Broadcast Receiver and Peer-to-Peer Manager

When we start the game, the first thing to do is to set up our broadcast receiver, by this means, create an IntentFilter and add the same intents that our receiver checks for.

Secondly, an instance of WifiP2pManager is obtained and our application is registered in the Wi-Fi P2P framework by calling the initialize function in our manager.

Then, we are able to detect available peers that are in range by calling the discoverPeers method (asynchronous) in our manager.

Peers Discovery

After discovering new peers, we created a list of the Peers, peerListListener, which provides information about the peers that the Wi-Fi P2P has detected. This way, every time there’s a change in our list of available devices, a function inside our list, onPeersAvailable, is called asynchronously and it updates the list of peers in our MainActivity, by calling updatePeerList. Besides updating the list of peers available, this method also changes the interface in order for the user to see what peers are available at the moment.

Having this done, the user is ready to see who is available and to start a group with how many devices he wants to.

Group creation

CreateGroup method

The creation of the group was made in a specific way for this application. A group can be created in WiFi P2P by calling the method createGroup on our manager. By creating a group, it means that this device is accepting multiple connections to him. However, it can still connect to other devices.

Connecting to the Group

The Wi-Fi direct has a system that, when a connection is established, it calls back a method named onConnectionInfoAvailable, from WifiP2pManager.ConnectionInfoListener. This function divides the user’s code in two parts: one for the server and another for the client.

This server is chose by Wi-Fi direct automatically, by reading the groupOwnerIntent integer (0 to 15) from both devices. Every device has its own default value although it can be modified when connecting to a group, by this means, only when connecting to a device that has made a group.

To have our multiplayer game functional, we had to find a way of connecting multiple devices and they all had to have the same server.

Therefore, the clients had to connect with the server because only the device who is connecting is able to set its groupOwnerIntent (to 0 in this case because we want it to be a client).

When we tried to do the opposite, we realized that the server who wanted to connect with an owner intent of 15 could connect with a device that has 15 as a default value too (they both wanted to become group owners owners) and the connection would crash in this way.

Since no device has a default value of zero, we establish the connections from client to server. So, the user that wants to create a group chooses how many players it wants at the beginning.

There was only one catch. How do the clients know that a group was made by another user?

Service Discovery

All the users know who is available in the network, as we described before, but they don’t know which ones created a group already and they can’t connect randomly.

In order to solve this issue, we used another Wi-Fi P2P feature called Service Discovery.

Service Discovery allows us to discover the services of nearby devices directly, without being connected to a network. We can also advertise the services running on your device. These capabilities help us communicate between apps, even when no local network or hotspot is available.

By implementing this feature, we are able to advertise our group made, without being connected to any device.

Step-by-step

Consequently, we followed these steps for the group creation:

- When the button is pressed, a group is formed right away, with the right number of players expected to connect;
- After that, a local service it’s created and added to our manager;
- Since this Service Discovery has not an asynchronous call, all the users have a Refresh button, that updates the available services in the network for them. This available services are then written in another list with the connections available, the same way we did for the available peers;
- In order to connect with a group Owner, the user only has to press one of the available services in that list and a connection is established.
-

ConnectionInfoAvailable method

Looking at the onConnectionInfoAvailable method more carefully, we see that it receives an argument, WifiP2pInfo info, that contains information about the connection that was just established (group owner address, if the device is the group owner, etc).

Using this information, we divide the code in two parts (client and server), as it was said previously.

Lets look at both sides at the same time:

Server/GroupOwner

- If the function is called from a working connection and not from a createdGroup callback, increase the number of Players that has joined the group by one;
- If the number of connected players equals the number of players that the server wanted, launch a thread. If not, just exit this function;
- This thread makes a server socket and accepts connections from a number of devices equal to the number of players that it chose (minus 1 because he doesn't accept himself of course);
- Sends the game object that it previously created to all the clients, each one with a different priority. This priority represents the turn to play that each client has;
- Begins new activity PlayGame that starts the game itself.

Client

- If this is the first time connecting to someone, launch a thread. If it isn't, just exit the function. It means that another connection is already going on;
- This thread creates a socket and tries to connect with a server socket, in the other end (blocks);
- Connection accepted;
- Waits to read a game object (blocks);
- Receives the game object;
- Begins new activity PlayGame that starts the game itself.

PlayGame

The PlayGame activity is being used by both the server and the clients (as the game activity), and implements all the operations and actions that must be done by a player of agony, while in some places the actions are separated to client's and server's actions.

At the creation of the PlayGame activity, the class variables **game** and **priority** are being initialized by retrieving their value from the bundle that was used by the AsycTasc thread to transferred data to that activity. The priority of the server is always 0.

After that, the class variable state (unique for each player) is being created using the existing game and priority of the player, and an array of Socket objects is being created. When the PlayGame activity (the game) is launched, all the previously created sockets, as well as their input and output streams, are already closed. For that reason, a thread is being launched by both the clients and server in order to settle their communication sockets and streams:

The server is creating a new Serversocket object, two arraylists for the input and output streams for all the clients, and an arraylist with Socket objects for all the clients. Then, a for loop 'accepts' the client sockets and connects them with the streams. Every client creates a new Socket object, connects it with the server (using the same port and the group owner address that the activity has received from the bundle, and passed to the Thread object as arguments of the constructor) and settles input and output streams for it. Every client thread launches a new reading thread (for that client) that waits for the server to write the new game object.

All the socket and input-output variables (or lists variables) are global and static, so they can be used from all the different threads that operate network activities, and are being used through the whole game. The sockets and input/output stream objects are only closed at the destruction of the PlayGame activity (onDistroy() method).

After these threads die, the rest of the elements are being initialized (user interface) and the game rules are applied for the first time by a call to the function applyTheRules(), which is called every time a player begins their turn (before the player plays). That method checks if there are any necessary operations to be executed before a player makes their move (for example if the upper card is a seven, so the player has to pick 2 cards unless he also has a seven, if the upper card is a 9 so that player immediately loses their turn, and so on).

The rest of the game actions that must be implemented (e.g. game variables that must assigned to a specific value), are taking place right after a player finishes their move. Furthermore, when a player makes their move, a function called finish_playing() is being called, and the last thing that happens is that a thread is being started (different for the server and the clients). The server starts a new sending thread, while the clients start their own sending thread and a reading thread right after that. The client-server communication during the game is described in more details later.

The server- client communication

The server is always assigned with the priority 0, so is always the first one to play. In two words, very time the server has to play it starts a new sending thread for each client. That thread blocks waiting for reading only for one client, the client that plays next, waiting for a new game object input. After the server receives that input, the server broadcasts the new object to all the clients by starting different sending threads for each one, and if the next player (the next turn) is another client, it creates a new reading thread for that client, otherwise terminates the current thread. Each client starts at the beginning of the game a new reading thread, and receives a new game object from the server after every round, checks it its turn has come, where the current reading thread terminates, or goes on with the reading loop. After completing playing, each client starts a new writing-to-server thread, and a new reading thread right after that, and the same process goes on, until there is a winner.

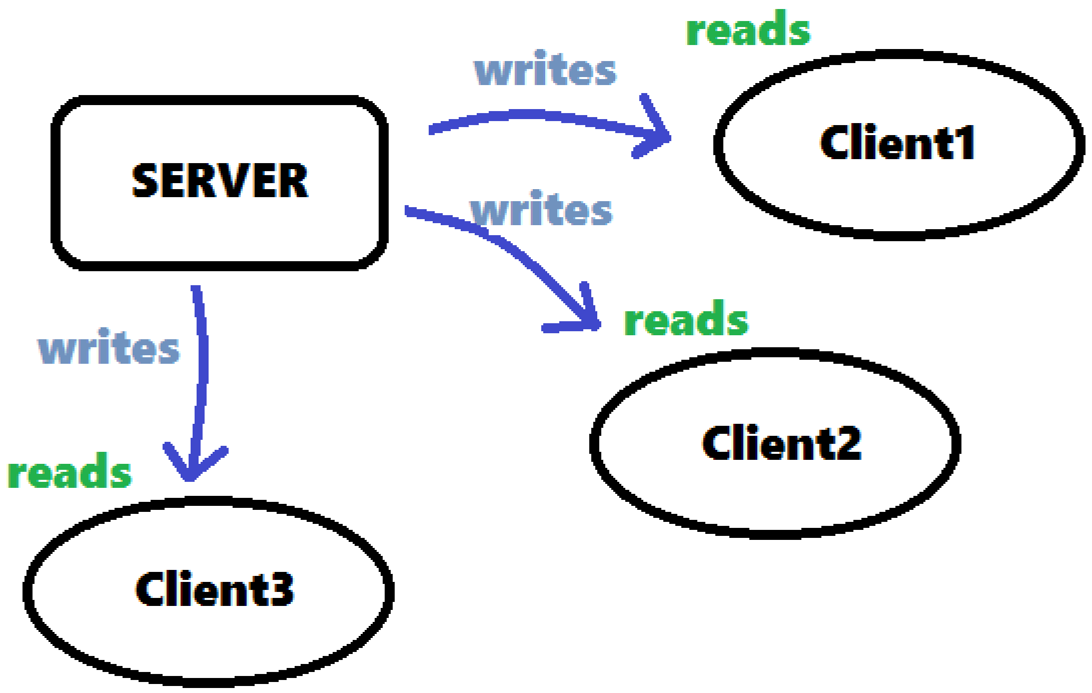
The process is described in the table below.

	CLIENT 1	*CLIENT 2*	*CLIENT 3*
SERVER	Write read	Write read	Write read
	Read write	read	read
	Write read	Write read	Write read
	read	Read write	read

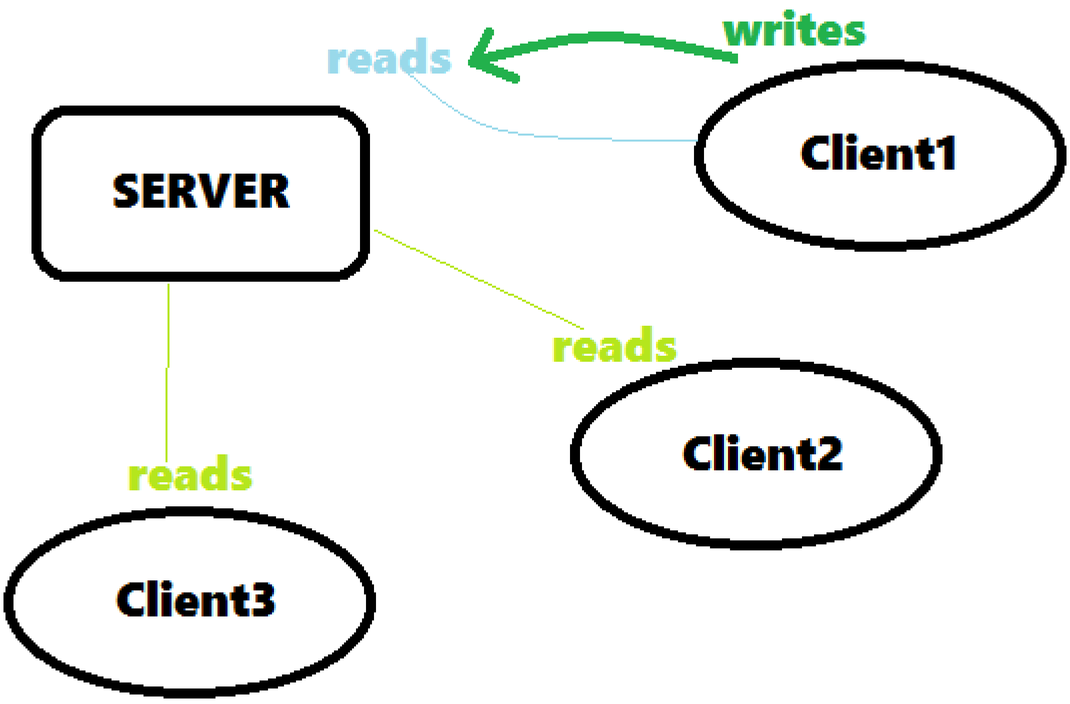
	Write read	Write read	Write read
	read	read	Read write
	Write read	Write read	Write read
	Write read	Write read	Write read

With more details:

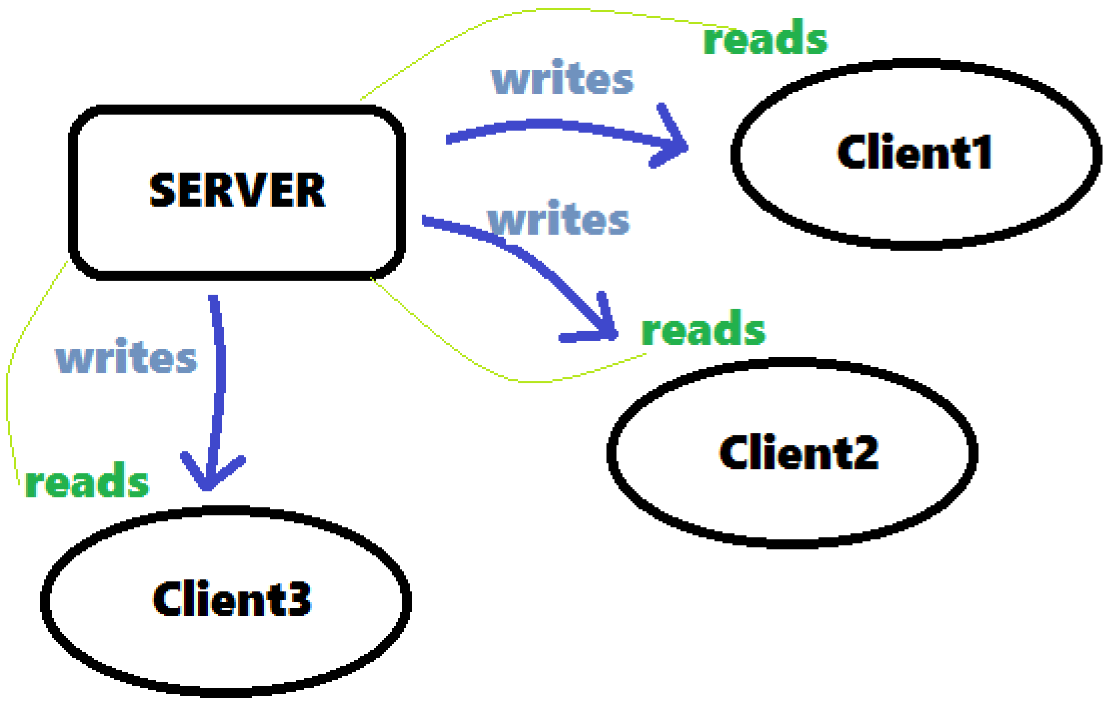
The clients start a new thread at the creation of the PlayGame activity, waiting to read from the server, while the server starts a thread for each client only after he (the player) makes their move. The server's threads writes to each client the game object. (They all update their game activity)

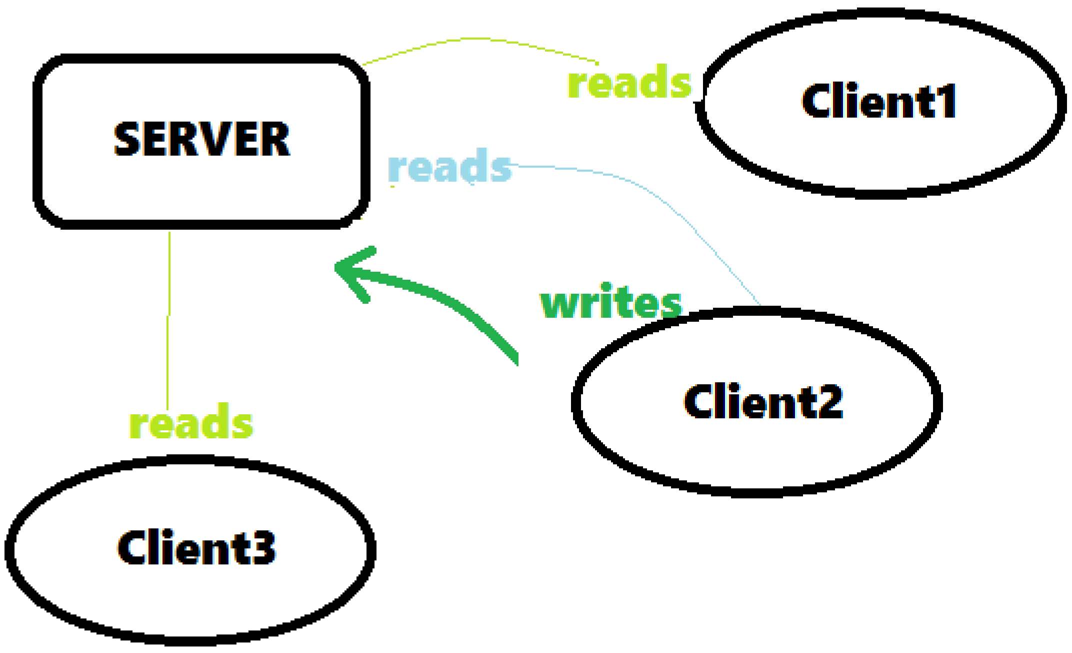


The server's thread that communicates with the client that is supposed to play next blocks, and the server waits to read from that client. The rest of the server threads terminate. The clients check if the next turn is theirs. If not, they remain in their (reading) loop, in order to read from the server again. The client that plays next terminates its thread and begins a new thread, which writes the game object to the server, only after the player makes their move.



After the clients finishes its move and writes to the server, it begins immediately a new reading thread. The server, from the reading thread, starts a new writing thread for every client, and checks if it its turn. In that case it terminates the current thread, so the player can make the new move. In the opposite case, it starts a new reading thread for the client that plays next. (Again, everyone updates their play activity)

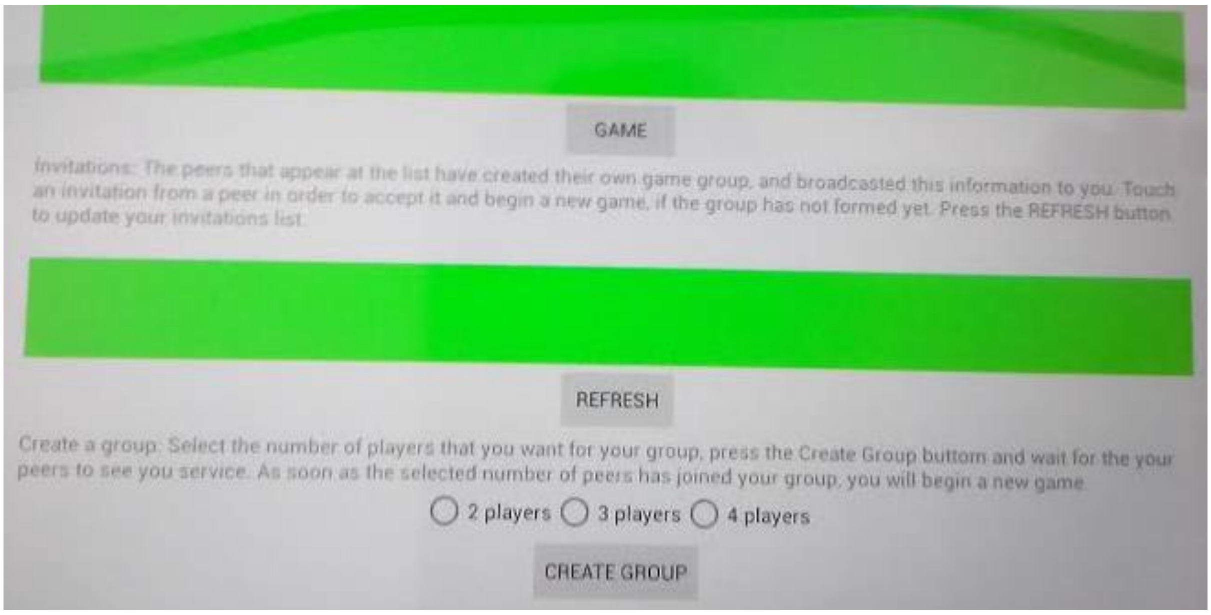




The same process repeats for all the clients in turn, until the server checks and finds out that its turn has come, so it starts no other reading thread for any client. The player of the server device then, can make their move, the server begins a writing thread for every client after that move, and the process starts from the beginning.

User manual

In the picture below we can see how we use our interface to create or enter a game.



Invitations: The peers that appear at the list have created their own game group, and broadcasted this information to you. Touch an invitation from a peer in order to accept it and begin a new game, if the group has not formed yet. Press the REFRESH button to update your invitations list.

Create a group: Select the number of players that you want for your group, press the Create Group buttom and wait for the your peers to see you service. As soon as the selected number of peers has joined your group, you will begin a new game.

We can see the available players in the first green screen and the available groups in the seconds green screen.

As a user, we have two options: either create a group or enter a group.

If we want to create our own group, we should select how many players we want and then click the "Create Group" button. We can't and aren't allowed to perform any more actions afterwards. Messages will appear when someone joins your group.

To enter a group, firstly we have to press the refresh button in order to see the available groups. Keep in mind that this process takes time for some android devices and you might be able to see the group only after a minute or two.

However, if you can't see any group, <u>but you know that the group exists</u>, you can press the group owner in the first green screen and then press the game button to enter it's game. But this <u>is not convenient</u>, since the Wi-Fi Direct has its own problems so only use this button in emergencies! Before using the button, make sure that the group really exists or it won't work!