February 1, 2015

Analyzing WiFi P2P in the context of a hangman game

William L Honig, *Loyola University Chicago*

Asma Mehjabeen
Keerthana Aramudhan
Dr.William L. Honig

Loyola University Chicago
Department of Computer Science
28 February 2014

Analyzing Wi-Fi P2P in the context of a hangman game[1]

1. Introduction and Findings

Wi-Fi P2P[2], which complies with the Wi-Fi Alliance's Wi-Fi Direct™[3] certification program, is a relatively new addition to wireless communications systems. It is now supported in Android operating system (since version 4.0). In theory, Wi-Fi Direct offers advantages for ad hoc communications between mobile apps. A key goal of this project was to evaluate the ability of Wi-Fi P2P for interconnecting mobile apps by using a common game suitable to mobile screens and devices.

The application allows the user to interconnect two devices using Wi-Fi P2P and play the classic hangman game. The players search for devices that already have the game installed using Wi-Fi P2P service discovery and when it finds one, it can send a connection request to connect with the peer device. The peer device can choose to either accept or decline the connection request. When the two devices are connected, the game starts and the players can establish a socket connection using each of their device configurations and exchange messages with each other. The players take turns in guessing words. The player who initiates the connection goes first and gets to specify a word, which the second player has to guess, and a hint (displayed to player 2). The second player tries to guess the letters of the word and player 1 decides whether the letter is in the word or not. With every wrong guess, a part the man is hanged. The game ends if the second player guessed the word or if the hangman diagram is completed. The total score is displayed at the end of a game session. Any player can end the game session at any time.

This paper illustrates the software architecture, the method of adding Wi-Fi direct to your Android application, an understanding of a few pieces of code from the project and lastly learnings and future scope. A key finding of this project is that the application needs to run in the foreground in both devices while trying to establish a connection with a peer device.

[1] This work was partially supported by merit awards from The Graduate School, Loyola University Chicago
[2] Wi-Fi P2P, *Android Developers* - <http://developer.android.com/guide/topics/connectivity/wifip2p.html>
[3] Wi-Fi Direct™, *Wi-Fi Alliance*- <http://www.wi-fi.org/discover-wi-fi/wi-fi-direct>

2. Application Screenshots

Figure 2a shows the main screen that is displayed when the application is launched. When the user clicks on the NewGame button, then the Wi-Fi Direct activity is launched. This activity needs to be present in the foreground in both devices for the user's to see each others devices in the list of peers. When a user clicks on a peer device, a connection request is sent to the peer device as in Figure 2b and the second user gets 30 seconds to respond to the request.

Figure 2a

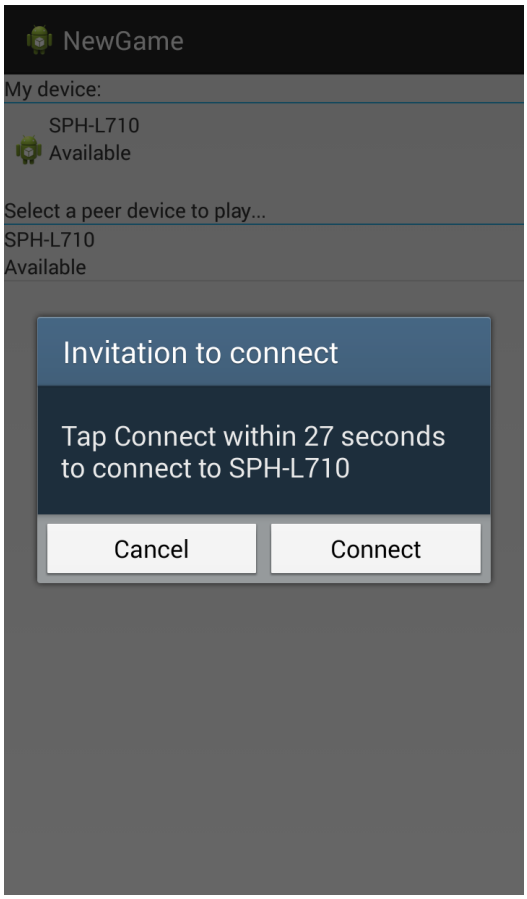

Figure 2b

Once the connection request is accepted by the second player, the player who

initiated the connection becomes player one and the other player becomes player 2. Figure 2c shows the screen where player 1 gets to specify the word and the hint. While player 1 is entering the word, player 2 sees a screen as shown in figure 2d that displays the appropriate message to the user.

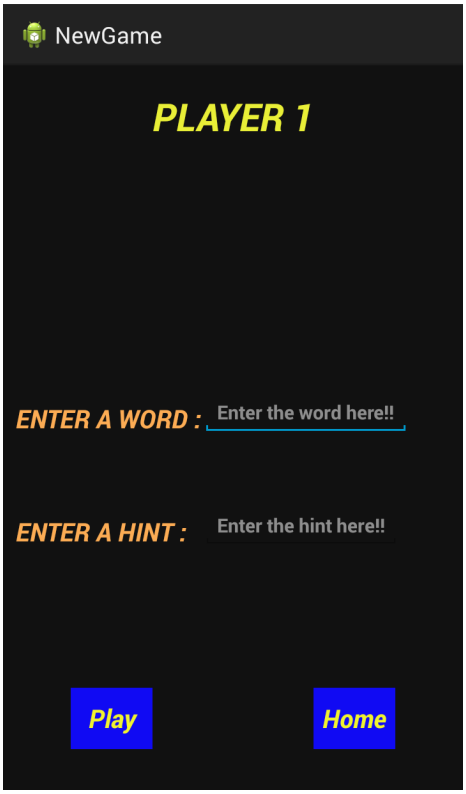

Figure 2c

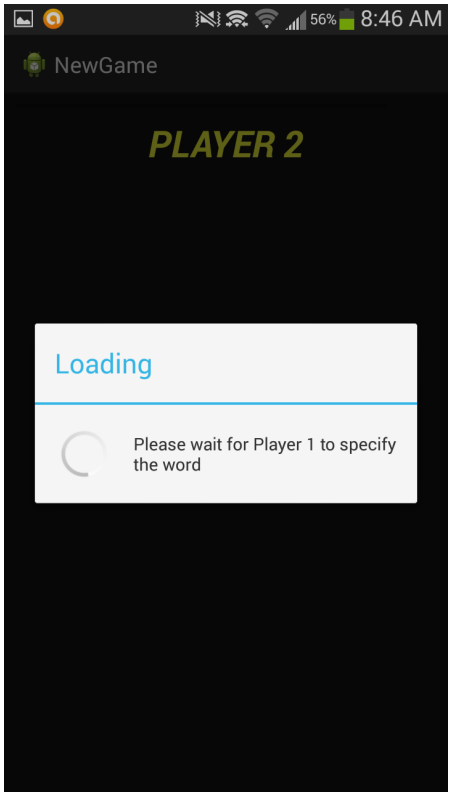

Figure 2d

As soon as player 1 clicks on the play button shown in Figure 2c, both devices will now display the screen as in Figure 2e. When Player 1 tries to click an alphabet button, the application displays a message to player one stating that its not his turn to guess the word. When the player 2 clicks on an alphabet button, if the letter is in the word then the word is updated in both devices; otherwise the hangman figure is updated. When the game ends, the application displays the score to both players as shown in Figure 2f. The players can choose to play the game again or exit the application.

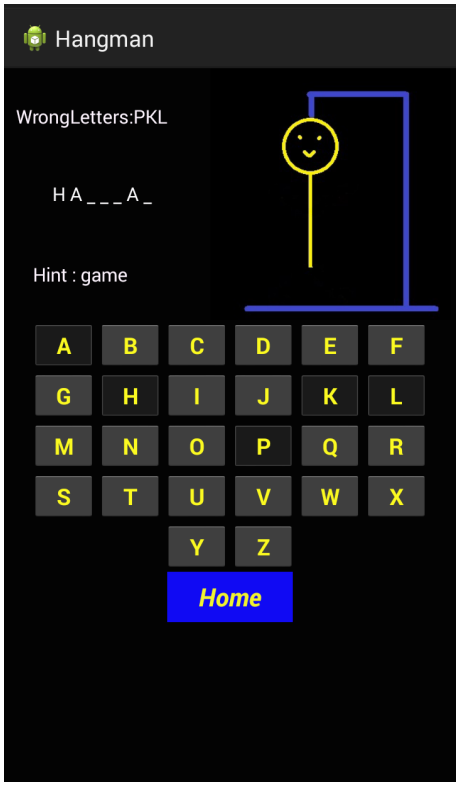

Figure 2e

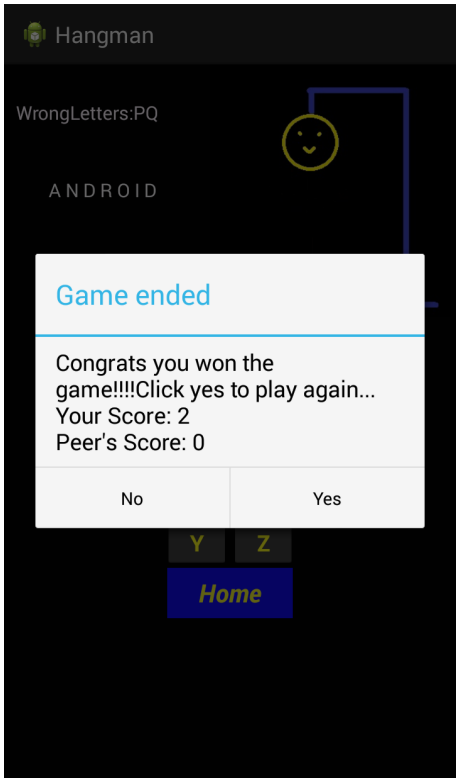

Figure 2f

3. Architecture of the software

The hangman application follows the peer to peer architecture of Wi-Fi P2P in Android as shown in Figure 3b, where players act as peers to connect to each other and exchange messages. Each player's device has the WiFiP2PManager[4] which is a part of the Android framework. The application depends on this class to connect to a peer device as shown in the figure below. Once the WiFiP2P connection is established, a socket connection is started between the two devices, by using the connection information from the Wi-Fi P2P connection; this socket is used to exchange messages. When the game ends or any player leaves the game, then both connections are closed. The application also makes use of Shared Preferences[5] to store data. Shared Preferences store data as key-value pairs in a file. That file is stored locally in the application's package in each device. The below code snippet in Fig 3a illustrates the usage of Shared Prefernces .

[4] Wi-Fi P2P Manager, *Android Developers, Google Inc.* - <http://developer.android.com/reference/android/net/wifi/p2p/WifiP2pManager.html>
[5] Android Shared Preferences, *Android Developers, Google Inc..* - <http://developer.android.com/reference/android/content/SharedPreferences.html>

```
/**
    Called when player two loses
     Get and set the details from and to the shared preferences
    */

   @Override
   public void playerLost() {
      startGame.setWinLoseStatus("Congrats you won the game!!!!");
      SharedPreferences sharedPref = startGame.getApplicationContext().
            getSharedPreferences(Constants.PREF_NAME, Context.MODE_PRIVATE);
      myScore = sharedPref.getInt(Constants.MY_SCORE,0);
      friendsScore = sharedPref.getInt(Constants.FRIENDS_SCORE,0);
      myScore += 1;
      startGame.setScores(myScore,friendsScore);
      SharedPreferences.Editor editor = sharedPref.edit();
      editor.putInt(Constants.MY_SCORE,myScore);
      editor.commit();
   }

    // when the game has ended switch the state of player 1 too player 2 using based o
the values stored in the shared preference
   @Override
   public void endGame(){
      //Change Players
      SharedPreferences sharedPreferences = startGame.getApplicationContext()
            .getSharedPreferences(Constants.PREF_NAME, Context.MODE_PRIVATE)
      SharedPreferences.Editor editor = sharedPreferences.edit();
      editor.putString(Constants.PLAYER,Constants.PLAYER_2);
      editor.commit();
   }
```

Figure 3a : Code snippet to illustrate the usage of shared preferences

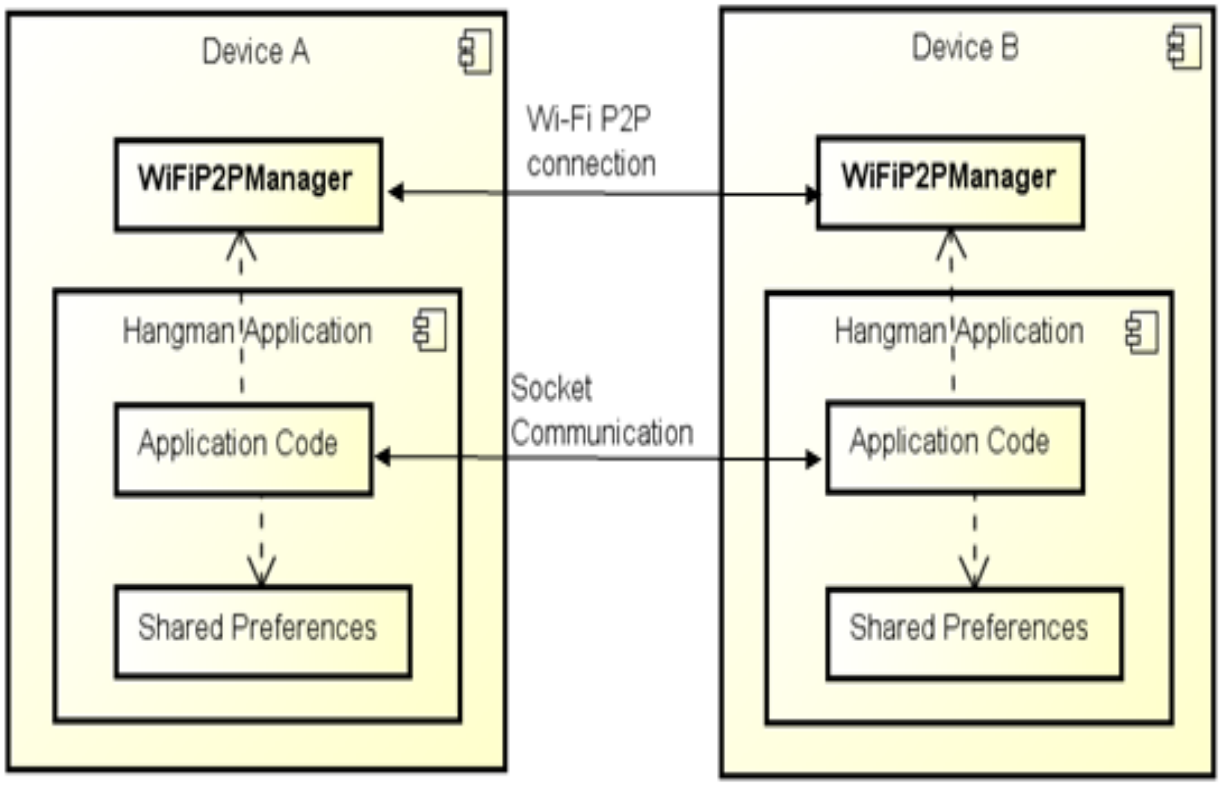

Figure 3b: Software architecture - Hangman

Also, the application code contains 3 packages – "gamelogic" that contains code related to the game, "wifidirect" that contains code related to Wi-Fi P2P and "common" that contains classes used by the other two packages. The WiFiDirectBridge class (a user defined class) shown in Figure 3c in the package "common" holds references to some of the game classes and the Wi-Fi direct class and so only a single instance of this class is required in the application. This class implements the Singleton pattern[6] to achieve the result. It also contains methods that the game classes use to send and receive messages. The main purpose for creating this class was to separate Wi-Fi direct logic from the application code. It allows the application to be extended to support different communication methods without hassle where the send and receive functions in this class have to be modified rather than modifying the code in the game logic.

[6] Singleton , *Design Patterns* , Web , Febuary 24, 2014 - <http://www.oodesign.com/singleton-pattern.html>

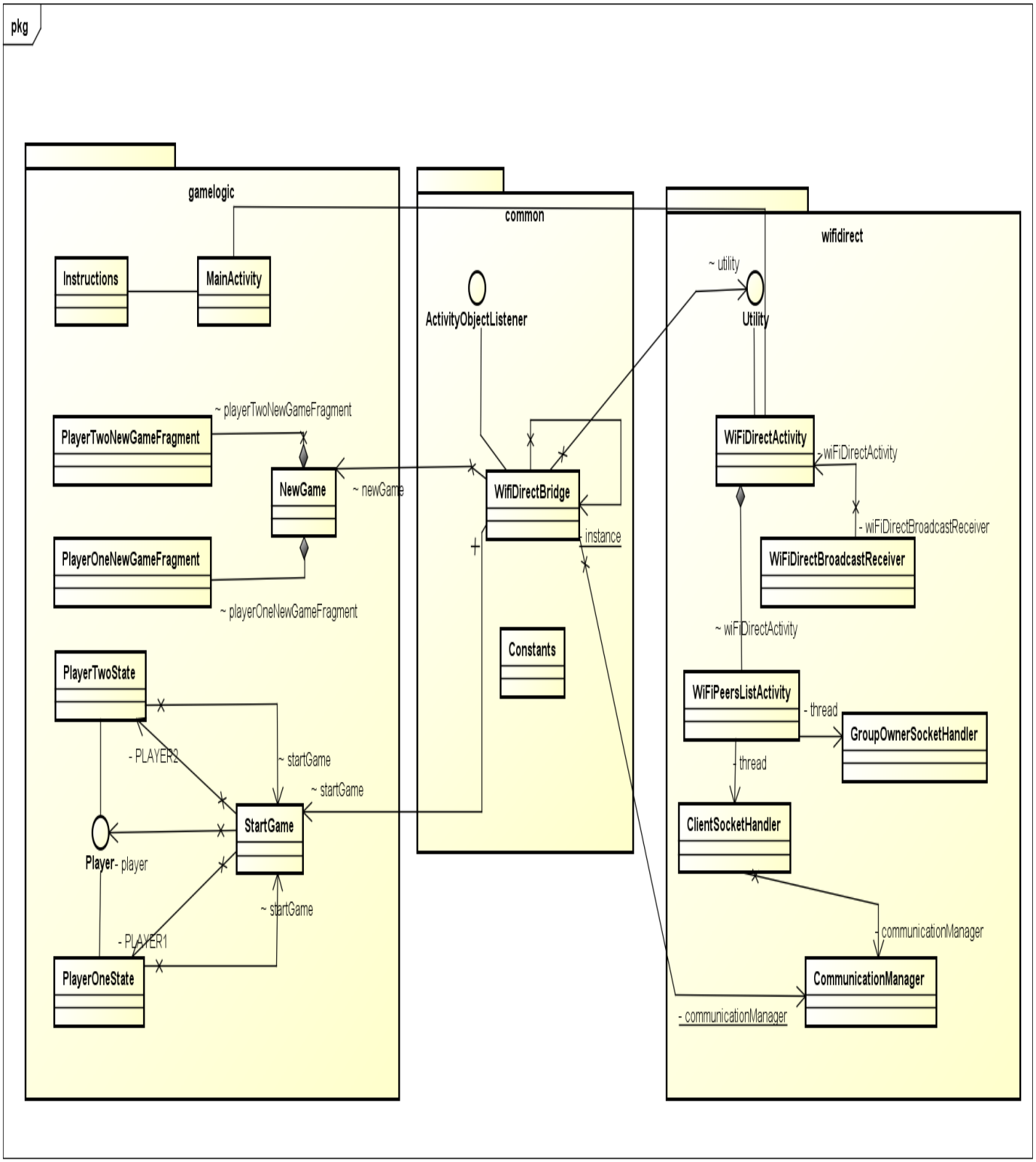

Figure 3c: The structure of hangman application

4. Adding WiFi Direct to your application

a. To include WiFi Direct in our application we need to add the following permissions in the manifest file:

```
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
<uses-permission android:name="android.permission.INTERNET" />
```

b. Then the application needs to set up the broadcast receiver that would listen to the four intents shown in the table below:

Intent Name	**Activity Performed**
WIFI_P2P_STATE_CHANGED_ACTION	Indicates whether Wi-Fi P2P is enabled
WIFI_P2P_PEERS_CHANGED_ACTION	Indicates that the available peer list has changed.
WIFI_P2P_CONNECTION_CHANGED_ACTION	Indicates the state of Wi-Fi P2P connectivity has changed.
WIFI_P2P_THIS_DEVICE_CHANGED_ACTION	Indicates this device's configuration details have changed.

The broadcast receiver can be registered in the onResume() method of the activity and unregistered in the onPause() method of the activity.

c. In the onCreate() method of our activity we need to call the initialize() method using WiFiP2PManager class which returns the channel used for connecting the application to the WiFiP2Pframework as seen in the code snippet in Figure 4a.

```
@Override
  protected void onCreate(Bundle savedInstanceState) {
     //set the content view
     /* register application with the WiFi P2P framework by calling initialize()
     returns a channel which is used to connect the application with the framework
      */
     manager = (WifiP2pManager) getSystemService(Context.WIFI_P2P_SERVICE);
     channel = manager.initialize(this, getMainLooper(), null);

  }
```

Figure 4a : Code snippet to illustrate the invoking of initialise() method of a Wi-Fi P2p manager

d. Then make a call to discoverServices() or discoverPeers() based on the need of the aplication. A call to discoverPeers() returns a list of peers in a WiFiP2PDeviceList object. But, a call to discoverServices() returns peers one at a time in a WiFiP2PDevice Object. The Figures 5b and 5c below, illustrate the sequence diagram and code snippet for discovering the peer devices

e. In order to connect to a peer device, a call to wifiP2Pmanager.connect() should be made which takes in the configuration of the peer device as a parameter.The Figures 5c and 5d below, illustrate the sequence diagram and code snippet for discovering the peer devices

f. For disconnecting, we can make a call to WifiP2Pmanager.disconnect().

g. WiFiP2P also contains other methods. An invitation that was sent to a peer device can be cancelled by calling the WiFiP2PManager.cancelConnect() method which cancels any connection invitations sent to peer devices. WiFiP2PManager.createGroup() can be called in order to set up a connection between multiple devices in which the caller of the method will be the group owner and to remove the group, a call to removeGroup() should be made. Also, if a peer device needs to find out who the current group owner is or any information about the group, then the device can make a call to requestGroupInfo() which returns the group information.

5. Finished project

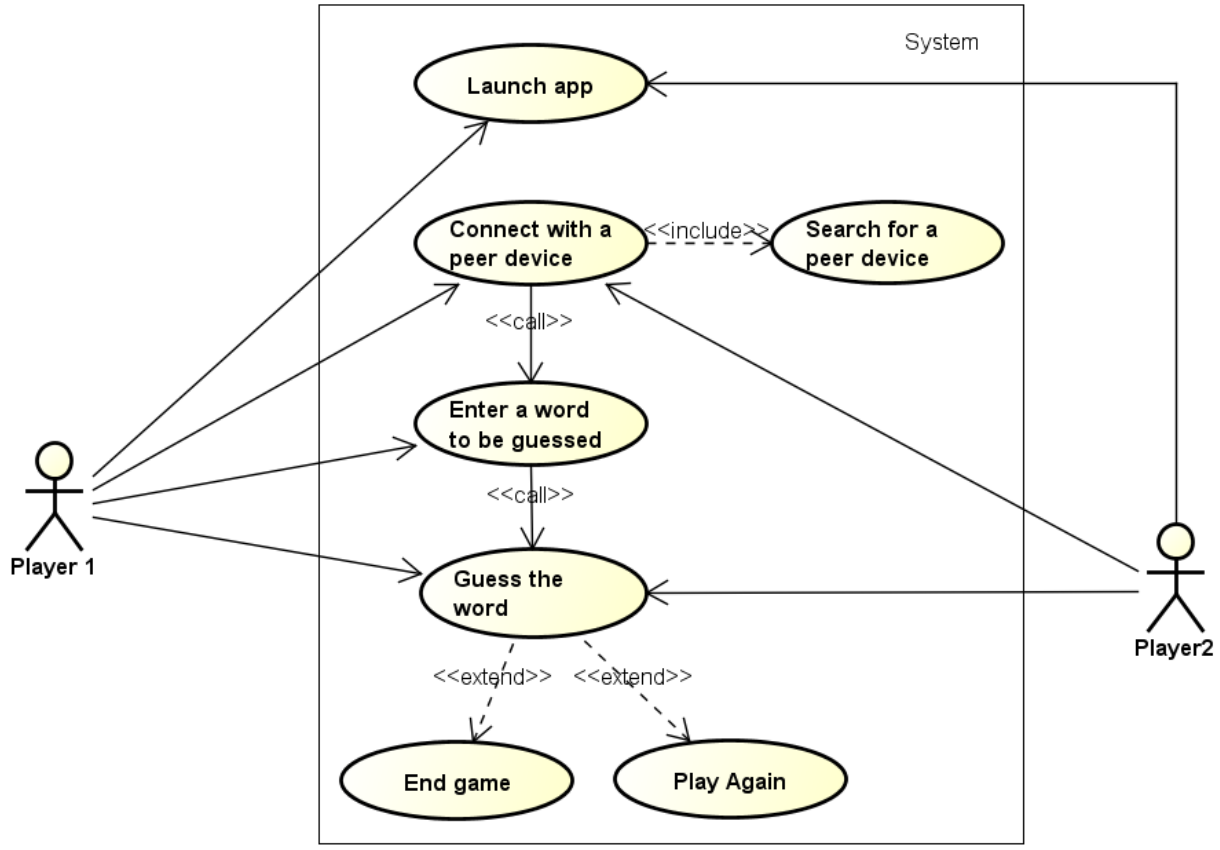

Figure 5a : Use- case diagram – Hangman

Discovering Services:

Use case: Search for a peer device

Actor: Any player.

Purpose: Allow the player to search for a peer device.

Overview: In order to discover devices that have the game installed, create a service request and make a call to discoverServices() which returns peers that advertise some service. Then select the peers that advertise the hangman game service and display them.

Preconditions: The player needs to have the application installed.

Postconditions: The user gets to view a list of available peers that have the hangman

application installed

Special Requirements: N/A

Flow of Events

Actor Action	System Response
Launch application	i. Register the application with the framework. It returns a channe via which the application can communicate with the framework Then register the broadcast receiver that would receive intents from the framework. ii. Register the broadcast receiver iii. Then create a service and advertise it using the framework and make a call to discoverServices() which returns peers that advertise services. iv. Select peers that advertise the same service as advertised by our application and display them.

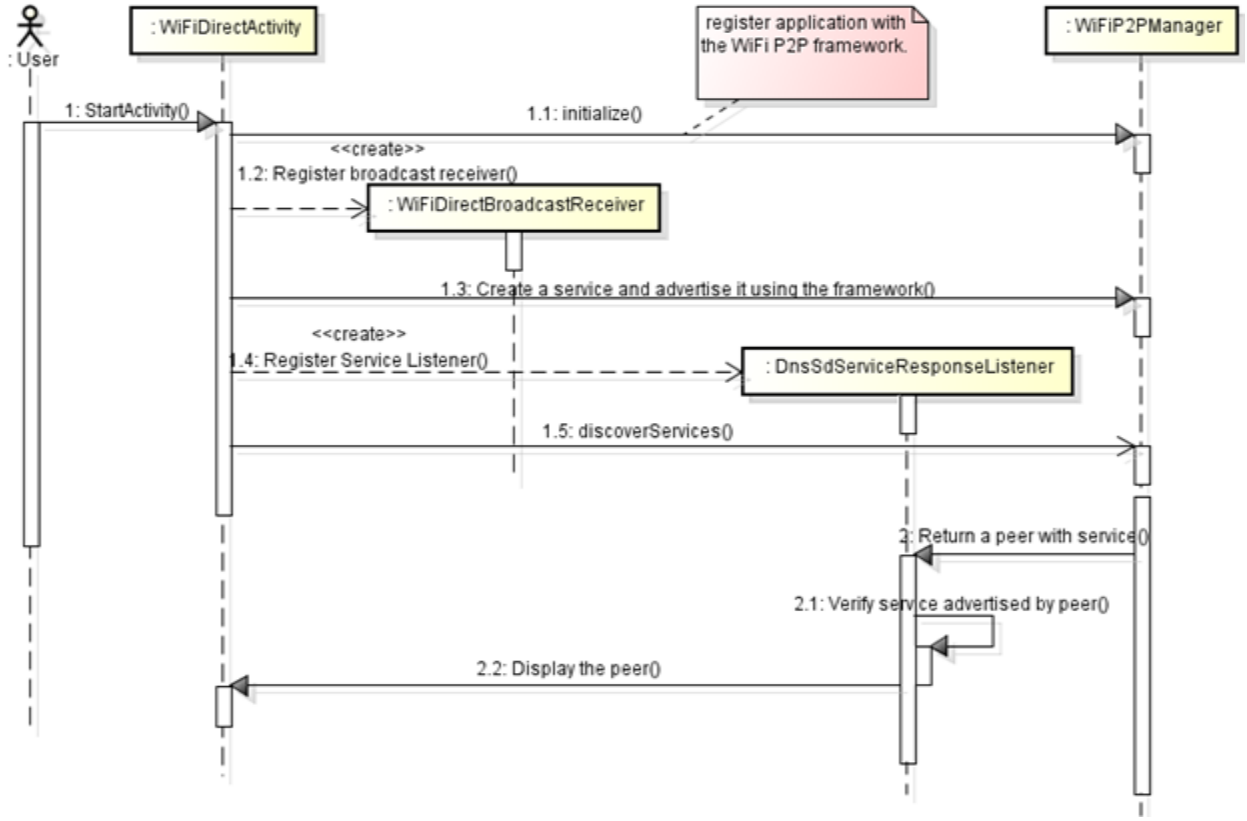

Figure 5a : A sequence diagram illustrating the "Search for a peer device" use case.

Below is the code fragment describing the searching/discovering of peer devices

```
/*
        Create a service.  Pass it an instance name, service type
_protocol._transportlayer ,
        and the map containing information other devices will want once they connect to
this one.
 */
WifiP2pDnsSdServiceInfo service = WifiP2pDnsSdServiceInfo.newInstance(
   Constants.SERVICE_INSTANCE, Constants.SERVICE_REG_TYPE, record);
manager.addLocalService(channel, service, new WifiP2pManager.ActionListener(){…})
//Create a listener
dnsSdServiceResponseListener =  (DnsSdServiceResponseListener)
(instanceName,registrationType,wifiP2pDevice) -> {
                //Check if the advertised service is the same as that advertised by our
application.
                if (instanceName.equalsIgnoreCase(Constants.SERVICE_INSTANCE)) {
                // update the UI and add the device to the list of available devices.
                WiFiPeersListActivity fragment = (WiFiPeersListActivity)
getFragmentManager()
                        .findFragmentById(R.id.frag_list);
                if (fragment != null) {
                  fragment.displayPeers(wifiP2pDevice);
                }
};
//Add the listeners to WiFi P2P manager.
manager.setDnsSdResponseListeners(channel, dnsSdServiceResponseListener,
                         dnsSdTxtRecordListener);
// After attaching listeners, create a service request.
 wifiP2pDnsSdServiceRequest = WifiP2pDnsSdServiceRequest.newInstance();
 manager.addServiceRequest(channel, wifiP2pDnsSdServiceRequest,
                new WifiP2pManager.ActionListener() {…});
manager.discoverServices(channel, new WifiP2pManager.ActionListener() {…});
```

Figure 5b : A code snippet illustrating the “Search for a peer device” use case.

Connecting to a peer device:

Use case: Connect to a peer

Actors: Player 1 and Player 2

Purpose: Connect the devices for player 1 and player 2 to start the hangman game

Overview: When Player 1 clicks on a peer device, a connection request is sent to the peer device(Player 2). The peer device can either accept or reject the connection and gets 30 seconds to respond. If the devices are connected successfully using Wi-Fi P2P, then a socket connection is established between the two to exchange messages.

Type: Essential Both the devices need to be connected during the entire game.

Preconditions: Both devices have the application installed and have displayed the list of available peers.

Postconditions: 1. If Player 2 accepted the connection request, then a socket connection is established between the two devices.
2. If Player 2 declines the request, then nothing changes.
Special Requirements: N/A

Flow of Events

Actor Action	**System Response**
1.Player 1 selects a device to connect	Send a connection request to the peer device by invoking connect() on the framework object
Player 2 responds to the connection request	i. If Player 2 accepted the connection request, the Wi-Fi framework in each device will broadcast the WIFI_P2P_CONNECTION_CHANGED_ACTION intent The application then requests the connection information When the connection information is received, a socket connection is established to exchange messages between the two devices. ii. If Player 2 declines the connection request, then Playe 1's device will be in the inviting state hence call cancelConnect() to go back to the available state.

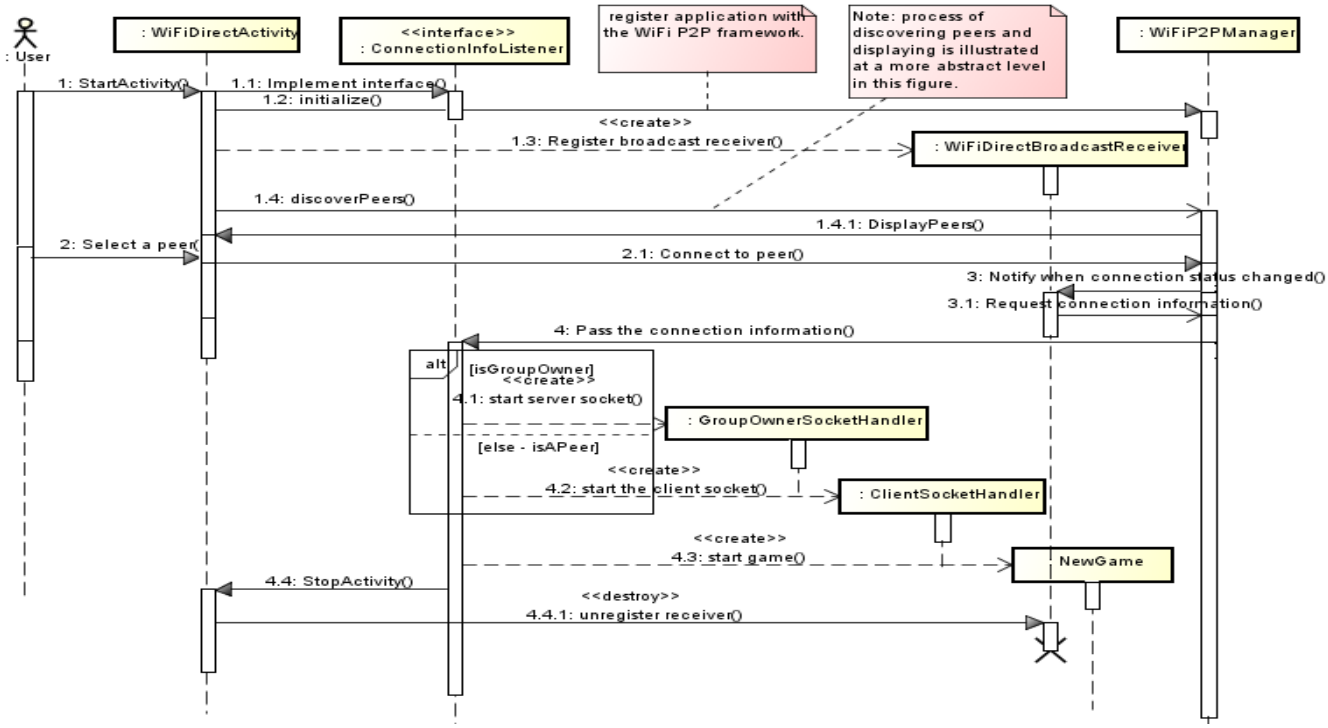

Figure 5c: Sequence diagram illustrating the “Connect to a peer” usecase. (This fig only depicts the sequence of messages in Player1 device and assumed that Player2 accepted the connection request).

```
WifiP2pConfig config = new WifiP2pConfig();
config.deviceAddress = device.deviceAddress;
//Wi-Fi Protected Setup(WPS) information.
config.wps.setup = WpsInfo.PBC;
manager.connect(channel, config, new WifiP2pManager.ActionListener() {…});
//Implement the WifiP2pManager.ConnectionInfoListener interface
@Override
public void onConnectionInfoAvailable(WifiP2pInfo wifiP2pInfo) {
   if (wifiP2pInfo.groupFormed && wifiP2pInfo.isGroupOwner) {
      //Start the server socket
   } else {
      //The device acts as the client. Start the client socket.
   }
}

In the receiver:
@Override
public void onReceive(Context context, Intent intent) {
String action = intent.getAction();
if (WifiP2pManager.WIFI_P2P_CONNECTION_CHANGED_ACTION.equals(action)) {
NetworkInfo networkInfo =
intent.getParcelableExtra(WifiP2pManager.EXTRA_NETWORK_INFO);
         if (networkInfo != null && networkInfo.isConnected()) {
                  manager.requestConnectionInfo(channel, connectionInfoListener);
         } else {
                  // It's a disconnect
                  wiFiDirectActivity.resetData();
                  //discover services again
                  wiFiDirectActivity.startRegistration();
         }
   }
}
```

Figure 5d : A code snippet illustrating the “Connecting of peer device” use case.

NOTE: The available WPS configurations in android are:
DISPLAY - Display pin method configuration - pin is generated and displayed on

device
Constant Value: 1 (0x00000001)
· INVALID - Invalid configuration
Constant Value: 4 (0x00000004)
· KEYPAD - Keypad pin method configuration - pin is entered on device
Constant Value: 2 (0x00000002)
· LABEL - Label pin method configuration - pin is labelled on device
Constant Value: 3 (0x00000003)
· PBC - Push button configuration
Constant Value: 0 (0x00000000)

Sending a guess and receiving a response:

Use case: Guess the word

Actors: Player 1, Player 2

Purpose: Allow Player 2 to guess a letter and receive the appropriate response from Player

Overview: Player 2 guesses a letter that is sent to player 1. The application checks if the letter is in the word or not and updates the mystery word or the hangman figure respectively. Also, while validating the guess, if the complete word is guessed or the hangman figure is completed, the game ends and players are notified whether they won or lost. Figure 5e and Tf illustrate this use case with a use case diagram and a code snippet respectively.

Preconditions: The devices have successfully established a connection to communicate with each other and the StartGame activity(which displays the main game) is in the foreground.

Postconditions: 1.Either the underscored mystery word or the hangman figure is updated in the players devices.
2. The alphabet button that was clicked by player 2 is now disabled in player 2's device.
3. If the word is guessed or the hangman figure is completed, the game ends.

Special Requirements: N/A

Flow of Events

Actor Action	**System Response**
Player 2 clicks an alphabet button	i. Send that letter to player 1. ii. Disable the button for that letter iii. Player 1 receives the letter iv. Check if the letter is in the word. v. If it is present then insert the letter in the proper position in the underscored word and send that word to player 2. Also, check if player 2 won and notify player 2. vi. If it is not present then increment the wrong letter count, append the letter to the string of wrong letters and send it to player 2. vi. Upon incrementing the wrong letter count in player 1's device, update the hangman figure and when player 2 receives the wrong letter count and the wrong letter word string, update the hangman figure in player 2's device. vii. If the wrong count reaches 7, notify the players that player 2 los and prompt them to either exit or play again.

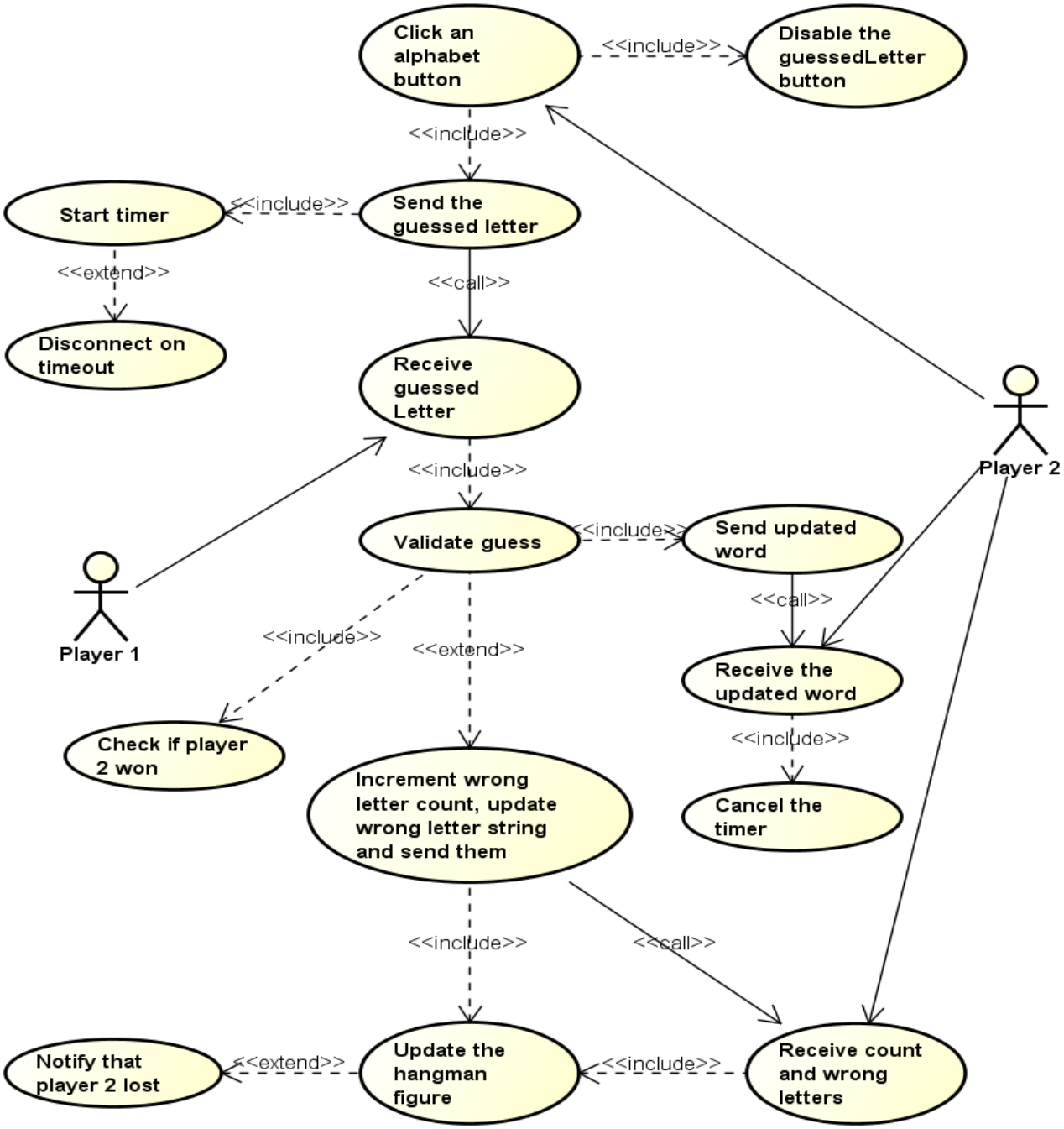

Figure 5e: A detailed use-case diagram illustrating the “Guess the word” use case.

```
In PlayerTwoState clickLetter method:
//view is the view id of the button clicked
startGame.send(Constants.COMMAND_LETTER + " " + letterPressed);
view.setClickable(false);
view.setEnabled(false);

/**
   Method is called by player 1 to validate the guess.
   @param guessedLetter  a char containing the alphabet whose button was pressed.
*/
public void validateGuess(char guessedLetter) {
   //validate the guess and send the underscored word
   …
   this.send(Constants.COMMAND_UNDERSCORED_WORD + " " + underscoreWord)
   …
   if (wrongGuess){
         …
            wrongLetterCount++;
           //Send the wrong letter count and the string containing the incorrect guesses.
           this.send(Constants.COMMAND_WRONG_COUNT+" "+wrongLetterCount+"
          "+StrWrongGuess);
            UpdateTheHangmanFigure(wrongLetterCount);
      }else{
         …
         if(tempString.equals(mysteryWordString)){
            playerWon();
         }
      }
   }
Receive in PlayerTwoState:
  @Override
   public void receive(String string) {
      if(string.startsWith(Constants.COMMAND_UNDERSCORED_WORD)){ …}
     else if(string.startsWith(Constants.COMMAND_WRONG_COUNT)) { …}
     else if(string.startsWith(Constants.COMMAND_WON)) { …}
   }
```

Figure 5f : A code snippet illustrating the “Guess the word” use case.

6. Findings

Based on the investigations and implementation work for this project we offer the following conclusions about WiFi P2P and other mobile interaction applications:

- We can opt to discover all the devices in range by making a call to discoverPeers() function. But if we intend to develop a game and want to discover only devices that have our application installed, then we need to register a service, advertise it and discover devices broadcasting the same service. Our device still discovers all the other devices in range but the listener(that is set up to listen for peer devices) can be used to filter the devices that the app should display to the user.

- For playing a game, both the devices should have the application running in the foreground of the application. Since the broadcast receiver was registered in onStart() and unregistered in onPause(), the application doesn't listen to the connection info even if the devices are connected when the activity is not in the foreground.

- Fragments can be added to an application statistically or dynamically. For adding static fragments, include them in the layout file in the <fragment> container. For using dynamic fragments include a <FrameLayout> container and then add your fragments dynamically in the activity by using FragmentTransaction class.

- Additional threads that are created in the application should be given a handler if they need to update the UI or send messages to the main thread. Only the main application thread should update the UI. Handlers can be used for passing messages between threads.

- For storing, Android provides many storage options such as storing in external storage or in an SQL Lite database or using Shared Preferences. If you need to store less amount of data then Shared Preferences could be an option where data is stored in key-value pairs and could be private to a specific application. Since the application only needed to store the player number and the scores of both player 1 and 2, we decided to use Shared Preferences.

7. Next steps, future work

- Different communication modules (Bluetooth and SMS) can be implemented and the user could be allowed to choose a method of communication with the other device.

- Game could also support single player version. The current implementation is structured to support this extension by providing a common interface for player which can be implemented to create a Single Player class.
- With the current set up we cannot extend the project to support the multiplayer version since the Wifi P2p code needs to be changed a bit. But since we created an interface for player, we can create a single player class and implement the Player interface and can provide the appropriate implementation of the methods in that interface.
- Finding if a Service can be implemented that runs in the background and listens for connection intents and starts the app on successful connection to peer device and registering the broadcast receiver in the manifest instead of registering in the activity.
- Remove the code that calls the new game class from the ConnectionInfoListener and let the user explicitly click a button to start the game when devices are connected. This could solve the problem where the game is not being launched on another device since the application was not running in the foreground