User Guide of FragMe

Introduction

FragMe is a Java framework which is used to develop peer-to-peer applications. It provides the management of the peers and their shared objects. Message transmissions and peer discovery are provided by a stripped down version of JGroups. JGroups provides peer discovery using multicast protocol and the message between peers can be sent using multicast or unicast.

FragMe can be used for Android and Desktop application development. The applications could be multiplayer games and any kind of social networking applications.

FragMe API

The FragMe framework provides an interface to develop applications. Basically it includes two parts: one is shared object and one is peer connection setup.

1. Build shared objects.

When starting an application, peers need to create objects or get shared objects from other peers. FragMe framework provides management of the objects and provides *FMeObject* abstract class. We need to import three related FragMe classes:

```
import org.nzdis.fragme.factory.FactoryObject;

import org.nzdis.fragme.factory.FragMeFactory;

import org.nzdis.fragme.objects.FMeObject;
```

The shared objects in the application need to extend the *FMeObject* and implement the abstract method. For example:

```
public class TicTacToeModel extends FMeObject {

}
```

Also the constructor must not have any parameters to be conforming to the implementation of the factory pattern. If an initialisation with parameters is needed for this object, use a separate initialisation method. A private constructor used could prevent normal instantiation. This is because FragMe needs to control all objects. For example:

```
private TicTacToeModel() {

}
```

Besides, the object class needs its own private FragMeFactory subclass to enable FragMe factory support. This factory enables access to all of the functionality of the Factory. For example,

```
private static class Factory extends FragMeFactory {

        protected FactoryObject create() {

                return new TicTacToeModel();

        }

}
```

Next the private factory has to be registered with the single FragMeFactory for this application.

This is done by using a static constructor:

```
static {

        FragMeFactory.addFactory(new Factory(), TicTacToeModel.class);

}
```

Since the object extends the *FMeObject* class, four inherited methods need to be implemented, *changed(FMeObject object), delegatedOwnership(FMeObject object), deleted(FMeObject object),* and *deserialize(FMeObject serObject).* For example:

```
@Override

public void changed(FMeObject object) {

}


@Override

public void delegatedOwnership(FMeObject object) {

}


@Override

public void deleted(FMeObject object) {

}
```

```
public void deserialize(FMeObject serObject) {

        this.positions = ((TicTacToeModel) serObject).getPositions();

        this.lastMove = ((TicTacToeModel) serObject).getLastMove();

}
```

The method *changed(FMeObject object)* is called when the object is changed and the method *deleted(FMeObject object)* is called when the object is deleted. The method *deserialize()* is responsible for deserialising your data. It is called whenever a serialized object containing the data of this *FMeObject* is needed. This is done in the FragMe framework by calling the change() method of the FragMe Object. For example:

```
tModel.change();
```

If only one field is changed, the name of the field can be passed into the change() method and only changes to that field will be sent to other peers. For example:

```
tModel.change(tModel.getClass().getField("positions").getName());
```

When this method is called it serializes the current object using the serialized version and then sends it to the other peers where it is deserialized and the new values of the fields are copied over to the FragMe object.

2. Set up connection of peers.

After we have shared objects built up, we can use that for the connection setup between peers. The related FragMe classes are:

```
import org.nzdis.fragme.ControlCenter;

import org.nzdis.fragme.helpers.StartupWaitForObjects;

import org.nzdis.fragme.util.NetworkUtils;
```

2.1 Set up local network address used to connect to other peers

The address is a string type. For example:

```
String address="127.0.0.1"
```

If you want to pick up a local non-loopback address, you can call *NetworkUtils.* For example:

```
String address = NetworkUtils
        .getNonLoopBackAddressByProtocol(NetworkUtils.IPV4);
```

This will get a local IPV4 type address.

2.2 Set up peer name

A peer name is needed to set up a connection. It is a string type. For example,

```
String user =System.getProperty("user.name");
```

2.3 Set up connection

FragMe provides ControlCenter class to help setup the connection of the peers. Developers can use it without the need to worry about any underlying code in the framework. To set up a connection, we need to use

ControlCenter.setUpConnections(String groupName, String peerName, String bindAddress)

When peers start up at the same time, we might need to use

```
ControlCenter.setUpConnectionsWithHelper("TicTacToe", user, address,
        new StartupWaitForObjects(1));
```

Class *StartupWaitForObjects* provides a method to wait for other peers setting up object storage so that new peer can get the shared objects.

The *ControlCenter.setUpConnections()* method contains all the setup code to get a FragMe application running. It creates the network connection and notifies the others a new peer has joined. It also takes care of setting up the initial objects. The example above is from the TicTacToe application and therefore the group it joins to is “tictactoe” with its peer name and local bind address.

2.4 Create and obtain shared objects

After the connection is setup, if there is a peer already there, we need to receive all the shared objects currently in the application from that peer. This is done by calling the *getAllObjects()* method.

```
Vector shareObjects = ControlCenter.getAllObjects();
```

This method then returns a vector of all the FragMe objects in the game.

If there is no other peer, we need to create objects. New FragMe objects cannot just be created by using the "new" keyword. Instead we must use the *ControlCenter* method *createNewObject*. This is because the FragMe framework uses the Factory Pattern in order to have absolute control over instantiating of relevant FragMe objects and to encapsulate object creation in one place. This then allows for sophisticated memory management, as the user cannot simply build instances of these objects, but must use the factory class. The following is an example from the TicTacToe application of how to create a new FragMe object.

```
tModel = (TicTacToeModel) ControlCenter
        .createNewObject(TicTacToeModel.class);
```

Android Application Development

We are now using the simple game TicTacToe as an example to explain how to develop an Android application using FragMe. TicTacToe is a traditional pencil-and-paper game for two players, O and X. The player who succeeds in placing three of their marks in a horizontal, vertical, or diagonal row wins the game. **Note:** The entire **source code** for that game (including configuration files referred to in this tutorial) can be found under https://github.com/NZDIS/fragme-apps.git in the subfolder TicTacToe (with further subfolders for the Android (*android*) and desktop version (*desktop*). Code used in both versions is located in the subfolder *shared.).*

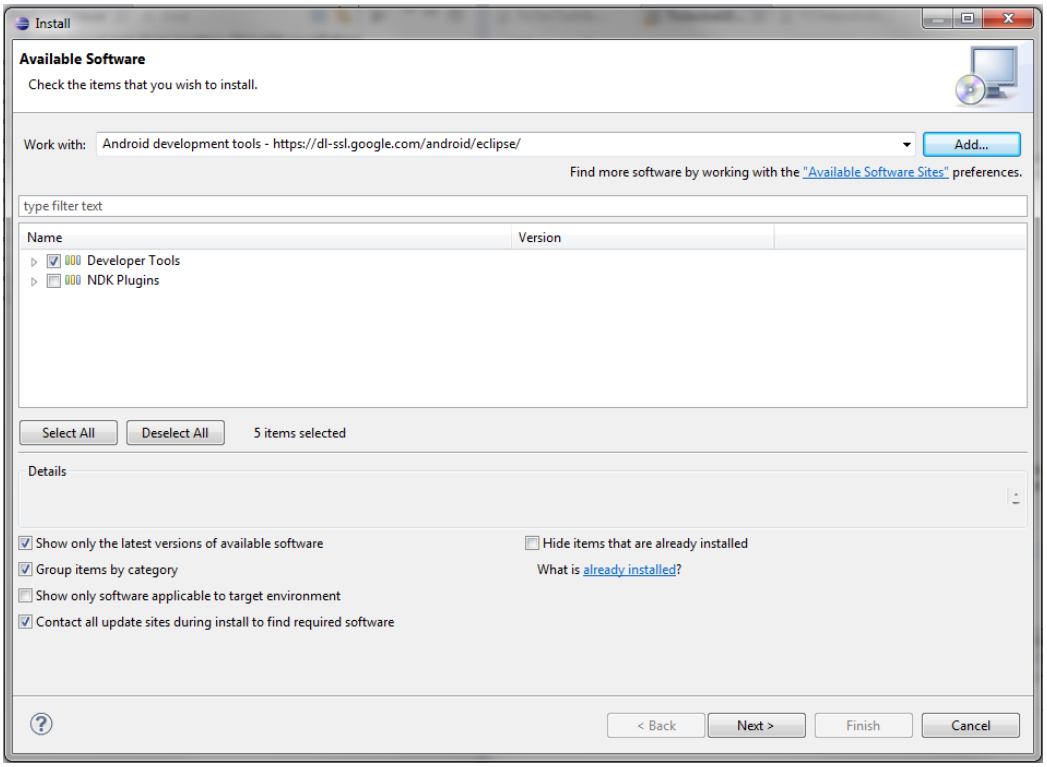
1. Install Eclipse and Android SDK

1.1. Install Eclipse

- Download Eclipse from http://eclipse.org/downloads/ and choose Eclipse IDE for Java Developers package.
- Install Eclipse on local machine. You might need to install the latest Java JDK and JRE first.

1.2. Install Android SDK plugins

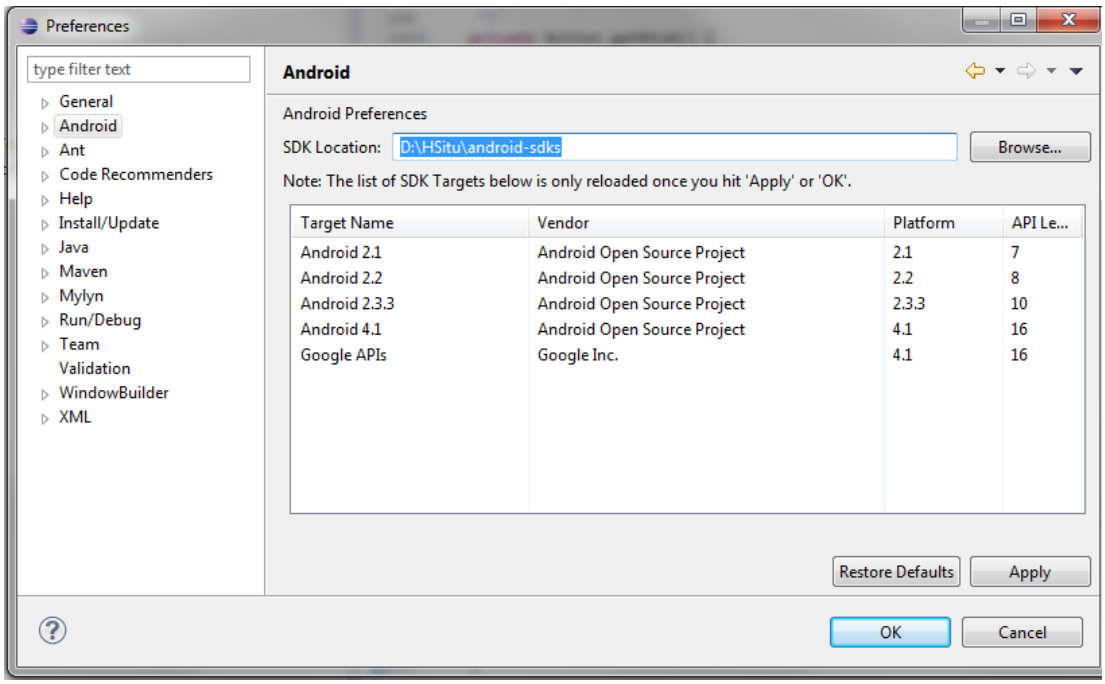
- Run Eclipse and choose *Help->Install New Software*.
- Click *Add* in the Available Software window.
- Enter *Android Development Tools* in the *Name* field, and https://dl-ssl.google.com/android/eclipse/ in the *Location* field.
- Click *OK* and check *Developer Tools* in the list of available software. This will install the Android Development Tools and DDMS, Android's debugging tool.



1.3 Click *Next* and *Finish* to install the plugin. You'll need to restart Eclipse once everything is installed.

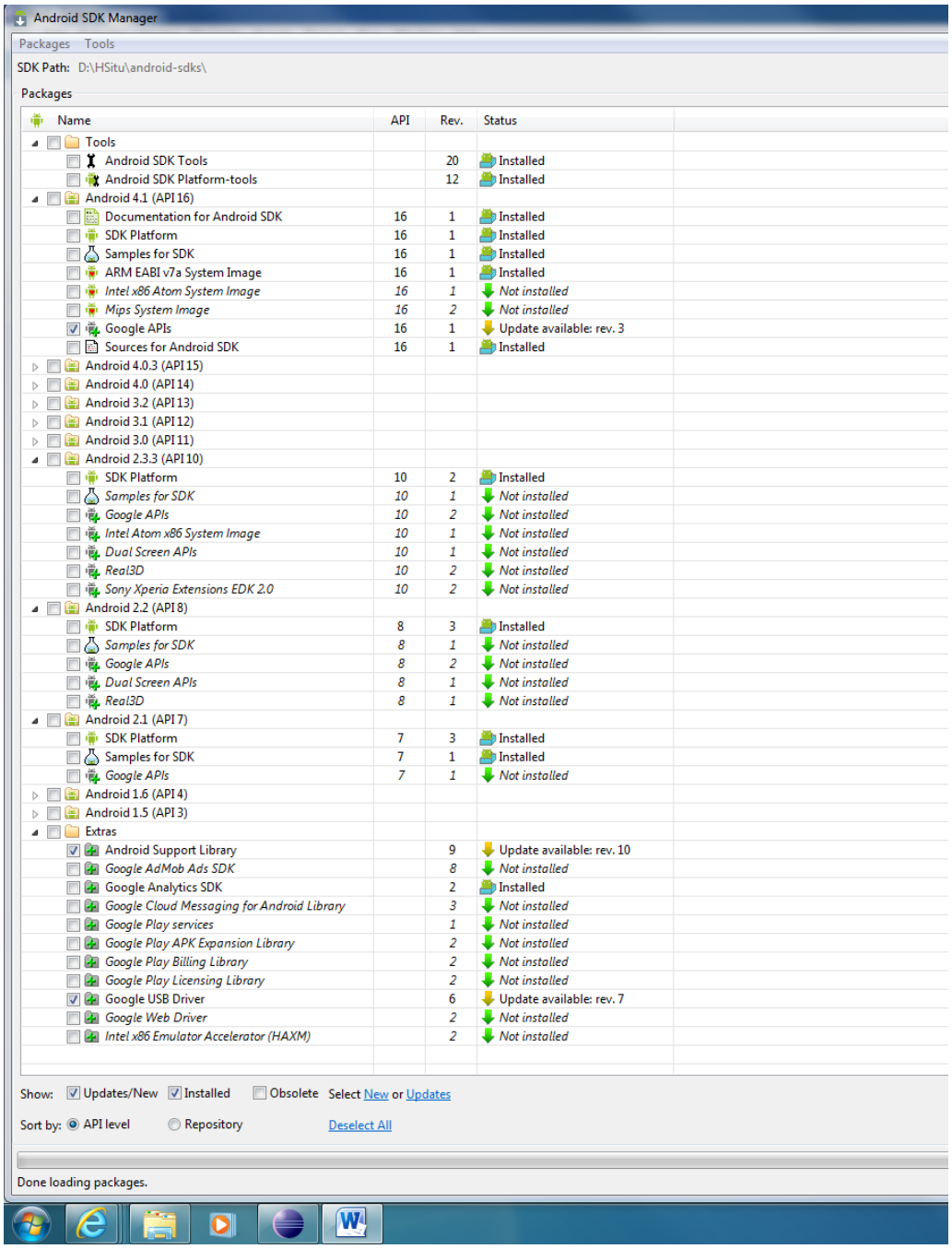
1.4 When Eclipse restarts, choose *Window->Preferences* and you should see *Android* listed in the categories.

1.5 You now need to tell Eclipse where you've installed the Android SDK. Click *Android* and then *Browse* to select the location where you extracted the SDK files. For example, *D:\HSitu\android-sdks*.



1.6 Click *OK* to save the Android SDK location.

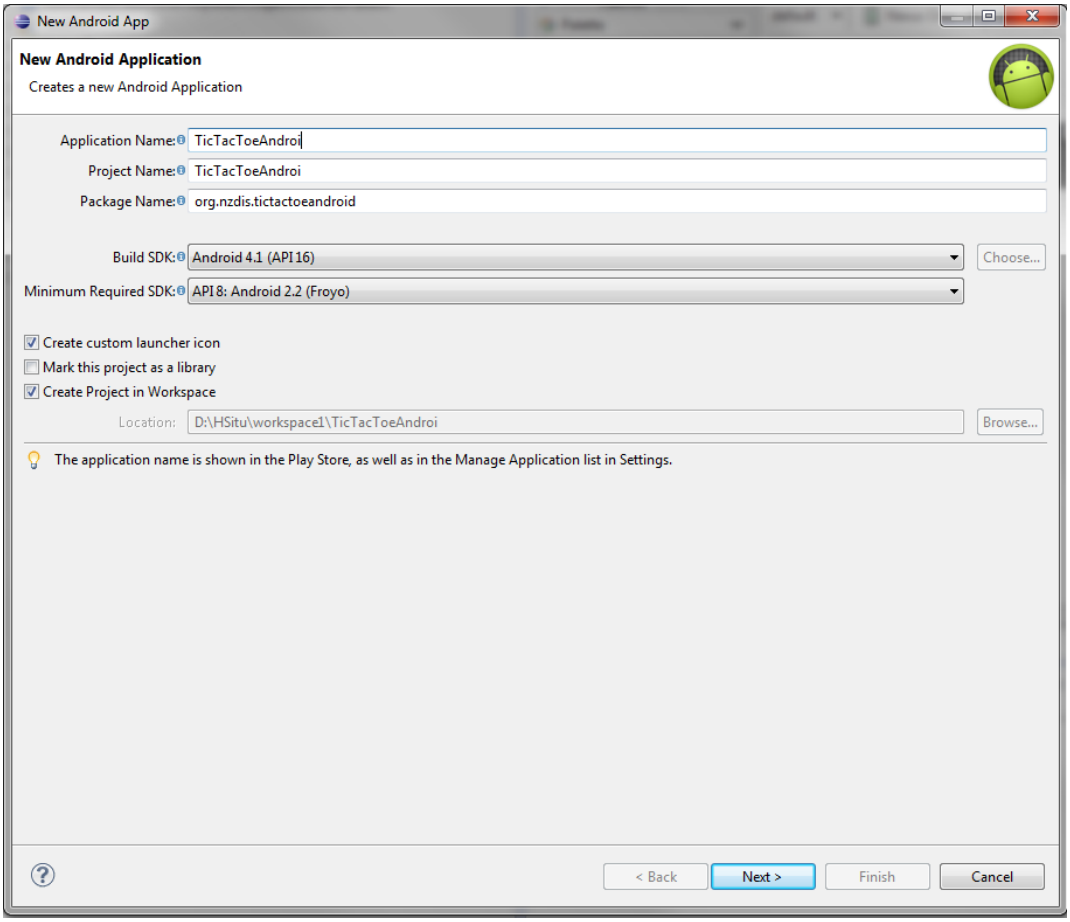
1.7 Choose *Window->Preferences->Android SDK Manager* to install specific Android API version, such as Android 2.2 (API 8) and Android 4.1 (API 16).



2. Create an Android project

2.1. Choose *File->New->Android Application Project*.

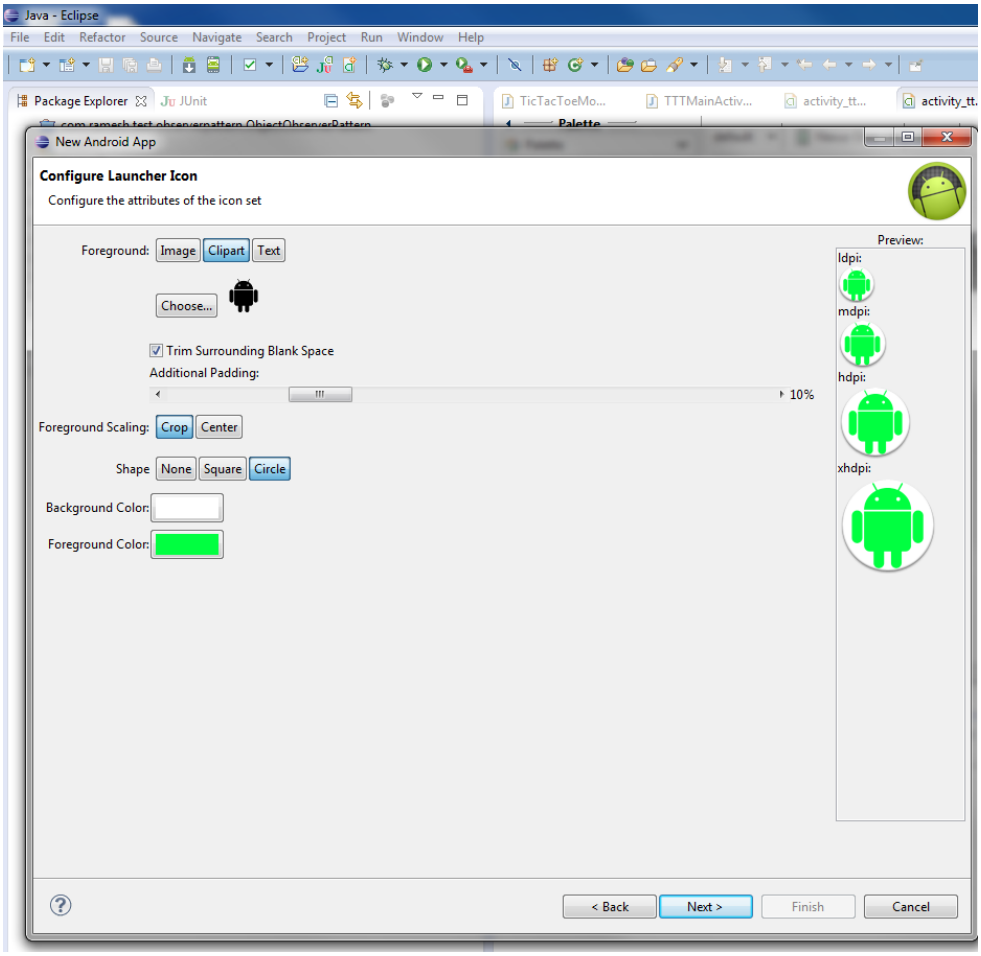
- Input names in the fields of *Application Name, Project Name and Package Name.*
- Choose the Android SDK version. The *Minimum Required SDK* means the earliest version of Android on which your application will run.



- Tick *Create custom launcher icon* and *Create Project in Workspace*
- Click *Next* to configure the application icon.

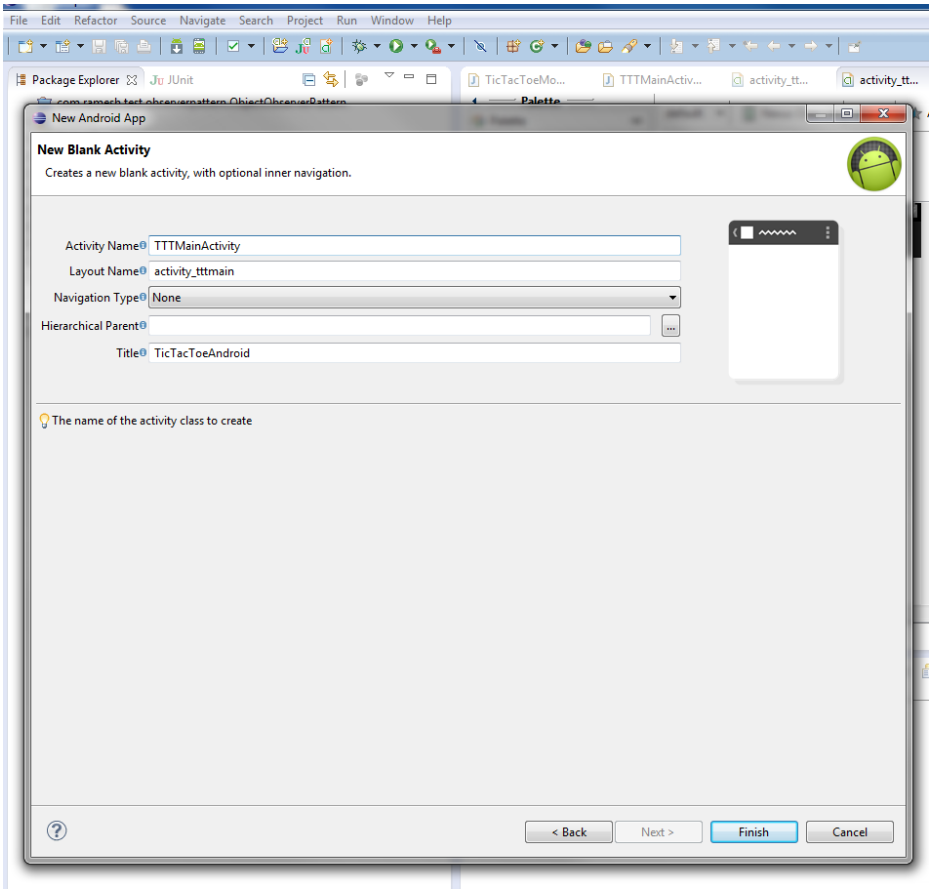
2.2 Configure launcher Icon

- Choose and design your own icon for your application.
- Click *Next* to create an activity



2.3 Create an Activity

- Choose *BlankActivity*
- Input names in the field of *Activity Name, Layout name* and *Title*.
    - *Activity name* is the class name of the activity
    - *Layout name* is the file name of layout saved under */res/layout*.
    - *Title* is the name of the application displayed on the screen.
- Click *finish* to finish setting up the Android project.



3. Build Path

3.1 To develop a FragMe application, we need to add FragMe and JGroups3 source in the build Path. First we need to download source from GitHub and then link the source of FragMe and JGroups3 to the project or add jar file to the /libs directory of the project.

The source code of FragMe is at https://github.com/NZDIS/fragme.git and the JGroups3 is at https://github.com/NZDIS/jgroups3.git. To download source from GitHub, you can refer to http://git-scm.com/book.

JGroups3 requires that the files bsh.jar and log4j.jar are included in your projects. They are located in JGroup3's lib folder and can be added by right clicking your project in Eclipse and selecting Build Path->Add External Archives...

3.2 Choose Android version for the application.
Right click the project and choose *Build path->Configure build path->Android* then choose one Android version. We use Android 4.1 for this example.

4. Create an Activity class

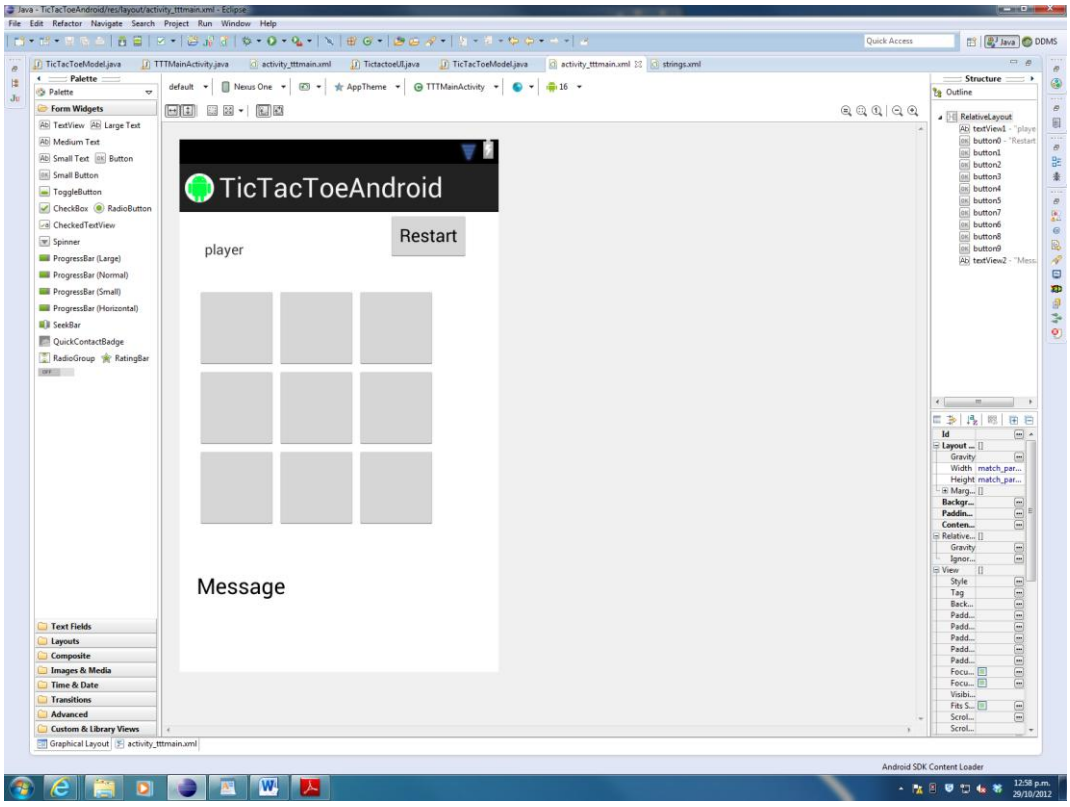
First we need to create a game interface displayed on the screens of the phones allowing users to interact with the screens. Then we need a class to handle the user input on the screen.

4.1 Create the TicTacToe game interface

Open and edit the file *activity_tttmain.xml* we created before under *res/layout/* of the project. You can choose to use Graphical layout to drag and edit buttons and text views or directly edit the .xml file.

For this example, we need

- A TextView to display the player information.
- A button for restart the game and start a new game. For this button, we need to set a method name for this button. When this button is clicked, the method will be invoked. The method name is given "onRestart". In the XML file, you can see An*droid:onClick="onRestart"* under this button.
- Nine buttons for Tictactoe game board. We set an *onClick* method name for each button from *onPosition1* to *onPosition9.*
- A TextView for displaying message
- The default name of each view can be defined in the *values/String.xml*
- The following picture shows the graphic layout of the TicTacToe interface.



4.2 Create an Activity class which is used to initial application and handle users' inputs.

Open and edit TTTMainActivity.java we created when we created the project. It is under the *src/org.nzdis.tictactoeAndroid/* of the project.

4.2.1 Activity class structure

In Android application, each activity class needs to extend *Activity* super class and implement callback methods that the system calls when the activity transitions between various states of its lifecycle. The callback methods are such as *onCreate(),* and *onDestroy().*

onCreate(): The system calls this method to create your activity. You need to initialise the essential components of your activity.

onDestroy(): The system calls this method when finishing your activity. You need to release some resources here.

For the TicTacToe application, we need to implement FragMe's Change*Observer* interface, because we want to be notified when an FMObject class changed. The *changed(FMeObject object), delegatedOwnership(FMeObject object)* and *deleted(FMeObject object)* methods of ChangeObserver need to be implemented. Calls to *changed* and *deleted* can both be handled in the same way, so we create a common method *updated*. *delegatedOwnership* is not used in this application so it is left empty.

The basic structure of the class is shown as follows:

```
public class TTTMainActivity extends Activity implements Observer{

  @Override

  public void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);

    setContentView(R.layout.activity_tttmain);

    ……

  }


  @Override

   public void changed(FMeObject object) {

            this.updated(object);

    }


  @Override
```

```
    public void delegatedOwnership(FMeObject object) {
    }

  @Override
   public void deleted(FMeObject object) {
            this.updated(object);
    }

   public void updated(FMeObject object) {
            ……..
   }

  @Override
   public void onDestroy(){
    super.onDestroy();
    ……
   }
 }
```

4.2.2 Link the layout with code

- Define same type of variables as they are in the *activity_tttmain.xml*.
- Use *findViewById()* to store them to the variables. Each item in the xml file has an id (@+id attribute in xml file).

```
   public class TTTMainActivity extends Activity implements ChangeObserver{
            private TextView player;
            private Button restart;
            private Button bn1;
            private Button bn2;
```

```
        private Button bn3;
    ....
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_tttmain);
     //get items of the activity
    player=(TextView)findViewById(R.id.textView1);
    restart=(Button)findViewById(R.id.button0);
    bn1=(Button)findViewById(R.id.button1);
    bn2=(Button)findViewById(R.id.button2);
    bn3=(Button)findViewById(R.id.button3);
....
    bn9=(Button)findViewById(R.id.button9);
    messinfo=(TextView)findViewById(R.id.textView2);
    ....
        }
....
}
```

- Implement *onClick* method for each button.
  For each button, it will call *buttonClick(position)* method. For example, when the first button is clicked, it will call *buttonClick(1)* with the position number 1 as parameter.

```
    public void onPosition1(View view) {
            buttonClick(1);
    }
```

  For each click, the method will check whether it is legal and record this move through calling *TicTacToeModel* class[1] methods. If it is legal, we can trigger

[1] The file TicTacToeModel **source code** can be found in the ***shared*** **subfolder** of the game source code as it is shared between the desktop and Android version of the game (see description under Android Application Development).

FragMe framework to change the whole object or only the changed fields of the object by calling *change()* or *change(field name).*

```
private void buttonClick(int position){
        boolean legal=false;
        if(playerNum==1){
                legal=tModel.OPlays(position);
                tModel.setLastMove("O");
        }else{
                legal=tModel.XPlays(position);
                tModel.setLastMove("X");
        }
        if(legal){
                /*transfer the whole object. We don't use it here because
                we only need to transfer two changed variables. It can save
                network traffic when the object is very big.*/
                //tModel.change();
                //transfer only changed fields
                try{

                    tModel.change(tModel.getClass().getField("positions")
                    .getName());


                    tModel.change(tModel.getClass().getField("lastMove")
                    .getName());
                }catch (Exception e) {
                    Log.e("TicTacToe","No such field");
                    e.printStackTrace();
                }
        }else{
                //print illegal move info
```

```
                    Toast.makeText(this, "Illegal move! Wait until the other peer
moves.",         Toast.LENGTH_LONG).show();

                    Log.e("Tictactoe","Illegal move!");


                }

                this.update(tModel, null);

            }
```

4.2.3 Implement *update()* method

When the peer gets notified the object has changed, this peer will update the game interface via *changed* and *deleted*. To display the changes in the Activity main thread, we can use *Handler* class of Android. Using method *handler.post (Runnable)* is shown as follows. The thread will be queued in the main thread. For example:

```
        public void update(FMeObject object) {


            final TicTacToeModel tttm=(TicTacToeModel) object;

            hdlr.post(new Runnable() {

                public void run(){

                    updateInterface(tttm);}

            });


        }
```

The method *updateInterface()* is the main method to update the game interface.

```
        //update interface

        private void updateInterface(TicTacToeModel model){

            String[] positions=model.getPositions();

            if(model.isNew_game()){

                messinfo.setText("");

                restart.setText("Restart");
```

```
        }
        for (int i=1;i<=9;i++){
            if(positions[i].equals("X")){
                buttons[i].setText("X");
                buttons[i].setEnabled(false);
            }else if (positions[i].equals("O")){
                buttons[i].setText("O");
                buttons[i].setEnabled(false);
            }else {
                buttons[i].setText(" ");
                buttons[i].setEnabled(true);
            }
            if (tModel.isGameOver()){
                buttons[i].setEnabled(false);
                if(tModel.hasWon("X")){
                    messinfo.setText("Player 2 won!");
                }else if (tModel.hasWon("O")){
                    messinfo.setText("Player 1 won!");
                }else {
                    messinfo.setText("The game is tied");
                }
                restart.setText("New Game");
            }
        }
    }
```

4.2.4 Initial the application.

To play the game TicTacToe, two peers will start to connect to a group “TicTacToe” and find whether there is already a peer there. If there is no other peer, this peer will be the game starter and create the TicTacToe game. If there is already a peer there, this peer will get the copy of the TicTacToe game object from the starter. After the peer creates or gets the game object, it will update its game interface. When this sets up, it will wait for the click events and start the interactive game.

To connect to a group, the peer first needs to set its local address used to connect to the network and the peer name. The peer name could be set by users. Here we just simply get the smartphone model as the peer name.

Since FragMe is a framework using multicast protocol, before the connection we need to unlock multicast of the smartphone. To access and change network state, we must edit *AndroidManifest.xml* file under the project and add those uses permissions.

```
private void initialConnection() {

        // set local address

        String address = NetworkUtils

                .getNonLoopBackAddressByProtocol(NetworkUtils.IPV4);

        if (address == null) {

            Log.e("tictactoe", "Could not find a local ip address");

            return;

        }


        // set the peer name

        String user = "test" + Build.MODEL;


        // unlock multicast of the devices

        WifiManager wifi = (WifiManager)
getSystemService(Context.WIFI_SERVICE);

        mcLock = wifi.createMulticastLock("mylock");

        mcLock.acquire();


        //set up connection
```

```
ControlCenter.setUpConnectionsWithHelper("TicTacToe", user, address,
                        new StartupWaitForObjects(1));

// get shared objects
Vector shareObs = ControlCenter.getAllObjects();

if (shareObs.size() == 0) {
        tModel = new TicTacToeModel();
        tModel = (TicTacToeModel) ControlCenter
                        .createNewObject(TicTacToeModel.class);
        Log.i("TicTacToe", "Create a new TicTacToe object");
        playerNum = 1;
        player.setText("Player1 (O)");

} else {
        tModel = (TicTacToeModel) shareObs.get(0);
        Log.i("TicTacToe", "Join and share the TicTacToe object");
        playerNum = 2;
        player.setText("Player2 (X)");
}
// set observer of the object and update the game interface
tModel.register(this);
this.update(tModel);

}
```

4.2.5 Finish the application

When the application is finished, we need to close the connection and release the resource of the devices. This will be done in the *onDestroy()* method.

```
public void onDestroy(){
    super.onDestroy();
    ControlCenter.closeUpConnections();// to close the connection
    if (mcLock.isHeld()) {
                mcLock.release();// to release the lock for the multicast
    }
    Toast.makeText(this, "Activity destroied",      Toast.LENGTH_LONG).show();// to
display the information
  }
```

5. Create a shared object

The TicTacToe object includes all the game logic and variables. To create a new class in Eclipse, you can right click on the package name and choose *new->class*. Type class name *TicTacToeModel*. This is the shared object between peers and we want it to be transferred by the FragMe framework, so we need to extend *FMeObject* class when we create it and implement the abstract methods of superclass *FMeObject*.

5.1 The basic structure of the TicTacToe object is shown as follows:

```
public class TicTacToeModel extends FMeObject {

        private static final long serialVersionUID = 2443727758382046169L;

        // The positions of user move
        public String[] positions = { "*", "1", "2", "3", "4", "5", "6", "7", "8",
                        "9" };

        private boolean new_game;
        // last move of peers
        public String lastMove = "";

        public TicTacToeModel() {
```

```
}

……

@Override

 public void changed(FMeObject object) {

        //System.out.println("Received a change notification!");

}


@Override

public void delegatedOwnership(FMeObject object) {

}


@Override

public void deleted(FMeObject object) {

        //System.out.println("Received a delete notification!");

}


@Override

public void deserialize(FMeObject serObject) {

        this.positions = ((TicTacToeModel) serObject).getPositions();

        this.lastMove = ((TicTacToeModel) serObject).getLastMove();

}


private static class Factory extends FragMeFactory {

        protected FactoryObject create() {

                return new TicTacToeModel();

        }

}
```

```
        static {
            FragMeFactory.addFactory(new Factory(), TicTacToeModel.class);
        }
    }
```

5.2 The logic of the game

The logic of the game includes deciding which player wins, whether a move is legal, and initialising variables for a new game. The methods are shown as follows:

```
    //Decide which player ("O" or "X") wins
    public boolean hasWon(String anXorO) {

        // check the rows for XXX or OOO
        if ((positions[1].equals(anXorO)) && (positions[2].equals(anXorO))
                && positions[3].equals(anXorO))
            return true;
        else if ((positions[4].equals(anXorO)) && (positions[5].equals(anXorO))
                && positions[6].equals(anXorO))
            return true;
        else if ((positions[7].equals(anXorO)) && (positions[8].equals(anXorO))
                && positions[9].equals(anXorO))
            return true;
        // check the columns for XXX or OOO
        else if ((positions[1].equals(anXorO)) && (positions[4].equals(anXorO))
                && positions[7].equals(anXorO))
            return true;
        else if ((positions[2].equals(anXorO)) && (positions[5].equals(anXorO))
                && positions[8].equals(anXorO))
            return true;
```

```
            else if ((positions[3].equals(anXorO)) && (positions[6].equals(anXorO))
                        && positions[9].equals(anXorO))
                return true;
            // check the diagonals for XXX or OOO
            else if ((positions[1].equals(anXorO)) && (positions[5].equals(anXorO))
                        && positions[9].equals(anXorO))
                return true;
            else if ((positions[3].equals(anXorO)) && (positions[5].equals(anXorO))
                        && positions[7].equals(anXorO))
                return true;
            return false;
    }

    // Decide whether the game is over. The game is over if either "X" or "O" have a
winning position or there are no more free positions
    public boolean isGameOver() {

            if (!hasFreePositions()) {
                lastMove = "";
                return true;
            }
            if (hasWon("X")) {
                lastMove = "";
                return true;
            }
            if (hasWon("O")) {
                lastMove = "";
                return true;
```

```
            }
            return false;
        }

        // Decide whether there are free places to play on
        private boolean hasFreePositions() {

            boolean hasFreePlacesToPlay = false;
            for (int i = 1; i < positions.length; i++)
                if (isFree(i))
                    hasFreePlacesToPlay = true;
            return hasFreePlacesToPlay;
        }

        //Decide whether the position is free to play on
        private boolean isFree(int aPosition) {

            if ((aPosition < 1) || (aPosition > 9))
                return false; // positions are out of range
            return !((positions[aPosition].equals("X")) || (positions[aPosition]
                    .equals("O")));
        }

        //Make a move for "X" player if it is a free place and not the same player as last
move
        public boolean XPlays(int aPosition) {

            if (isFree(aPosition) && (!lastMove.equals("X"))) {
```

```
                positions[aPosition] = "X";
                return true;
            }
            return false;
        }

        //Make a move for "O" player if it is a free place and not the same player as last move
        public boolean OPlays(int aPosition) {

            if (isFree(aPosition) && (!lastMove.equals("O"))) {
                positions[aPosition] = "O";
                return true;
            }
            return false;
        }

        //initial a new game
        public void restart() {
            positions[0] = "*";
            for (int i = 1; i <= 9; i++) {
                positions[i] = String.valueOf(i);
            }
            lastMove = "";
        }

        // getters and setters of all variables
        public String[] getPositions() {
```

```
        return positions;
    }

    public void setPositions(String[] positions) {
        this.positions = positions;
    }

    public String getLastMove() {
        return lastMove;
    }

    public void setLastMove(String lastMove) {
        this.lastMove = lastMove;
    }

    public boolean isNew_game() {
        return this.new_game;
    }
    public void setNew_game(boolean new_game) {
        this.new_game = new_game;
    }
```

6. Deploy and run the application

See the section of **Deploy and Run the application** of this document.

Desktop Application Development

Developers can use the same FragMe API to develop desktop applications as well. We are now developing a desktop version of TicTacToe application as Android one talked above.[2]

1. Install Eclipse

1.1 Download Eclipse from http://eclipse.org/downloads/ and choose Eclipse IDE for Java Developers package.

1.2 Install Eclipse on local machine. You might need to install the latest Java JDK and JRE first.

2. Create an Java project

Choose *File->New->Java Project*.

- Input names in the *Project Name.*
- Choose the JRE version.

Click *Next* then *Finish*

3. Build Path

To develop a FragMe application, we need to add FragMe and JGroups3 source in the build Path. First we need to download source from GitHub and then link the source of FragMe and JGroups3 to the project or add jar file to the /libs directory of the project.
The source code of FragMe is at https://github.com/NZDIS/fragme.git and the JGroups3 is at https://github.com/NZDIS/jgroups3.git. To download source from GitHub, you can refer to http://git-scm.com/book.
JGroups3 requires that the files bsh.jar and log4j.jar are included in your projects. They are located in JGroup3's lib folder and can be added by right clicking your project in Eclipse and selecting Build Path->Add External Archives...

4. Create a main class

First right click the project name to create a new package and then create a new class. For this example we create a class name *TictactoreUI.java*.

This class creates a game interface, sets up buttons and their *onClick* method to handle the users' clicks, initialises the application and implements *changed(FMeObject object), delegatedOwnership(FMeObject object), deleted(FMeObject object),* and *deserialize(FMeObject serObject)* methods of FragMe's *ChangeObserver* interface.

The main structure of the class is shown as follows:

public class *TictactoeUI* ***extends*** *Frame* ***implements*** *ChangeObserver {*

[2] To obtain the full source code for the desktop version of the TicTacToe game used in this tutorial, please refer to the description at the beginning of the chapter 'Android Application Development'.

```
    public TictactoeUI() {
        //initialise interface
        initialize();
        //initialise connection
        initialConnection();
    }
    @Override
    public void changed(FMeObject object) {
        this.updated(object);
    }

    @Override
    public void delegatedOwnership(FMeObject object) {
    }

    @Override
    public void deleted(FMeObject object) {
        this.updated(object);
    }

    public void updated(FMeObject object) {
        ……..
    }

    public static void main(String[] args) throws IOException {
        new TictactoeUI().setVisible(true);
    }
}
```

The class uses *java.awt* to create interface including labels, buttons and buttons' onClick methods. Those are the same elements in Android version but using *java.awt* classes.

initialConnection() is used to initialise the connection and application. It is the same as it in Android version.

updated() is called by *changed()* and *deleted()* so that the game interface can be notified when the shared *FMeObject* changed. Updating the game interface uses *java.awt classes*.

5. Create a shared object

The TicTacToe object includes all the game logic and variables. It is the same as it in Android version. You simply can copy the class TicTacToeModel.java to your Java project or just link the source from your Android project to your Java project.

6. Deploy and run the application

See the section of **Deploy and Run the application** of this document.

Deploy and Run the application

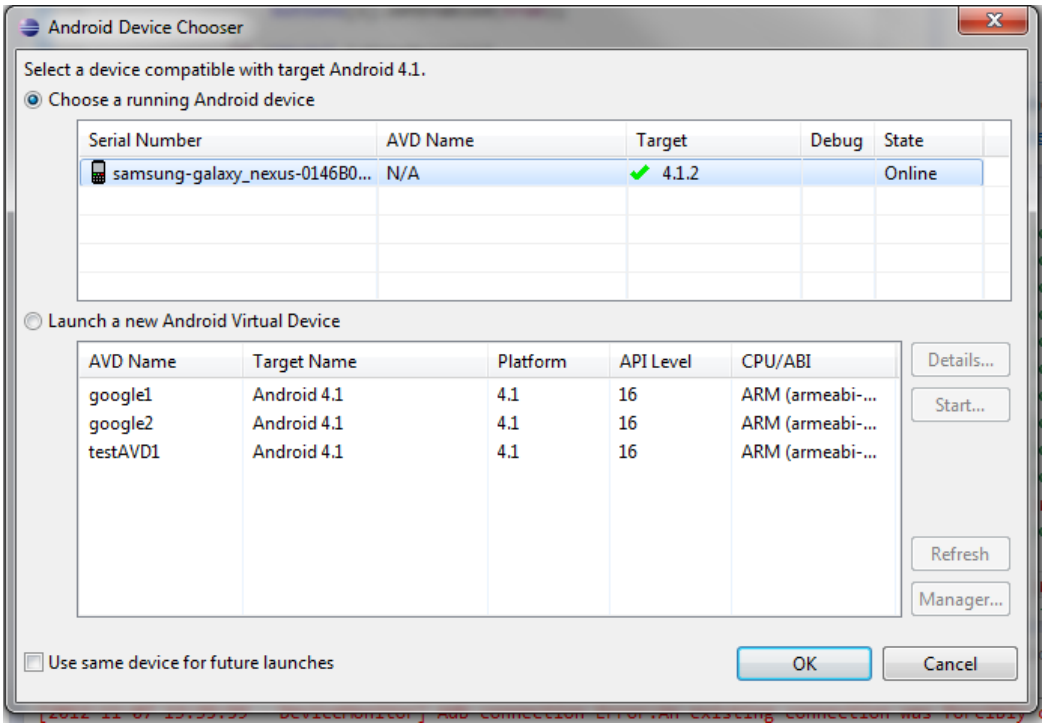
1. On Android phone

1.1. Start a wireless router and make sure the smartphones use the same wireless.

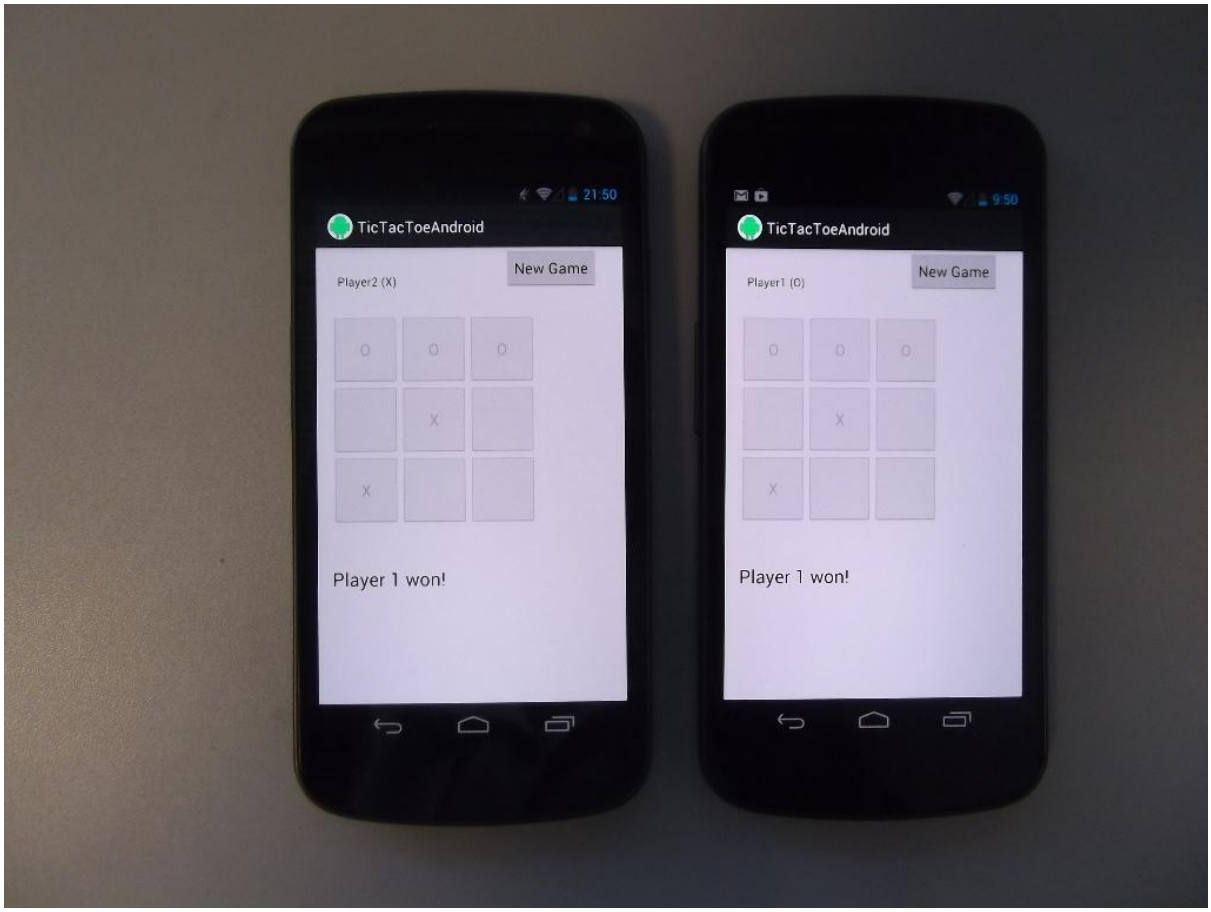
1.2. Connect smartphones to the computer with USB cables.

1.3. In Eclipse, click *Run->Run configurations*. Right click *Android Application->New*. Choose *Android* tab and click *Browse* to select project TicTacToeAndroid. Choose *Target* tab, tick *Always prompt to pick device*.

1.4. Click *Run* at the bottom and then choose a smartphone from the list.

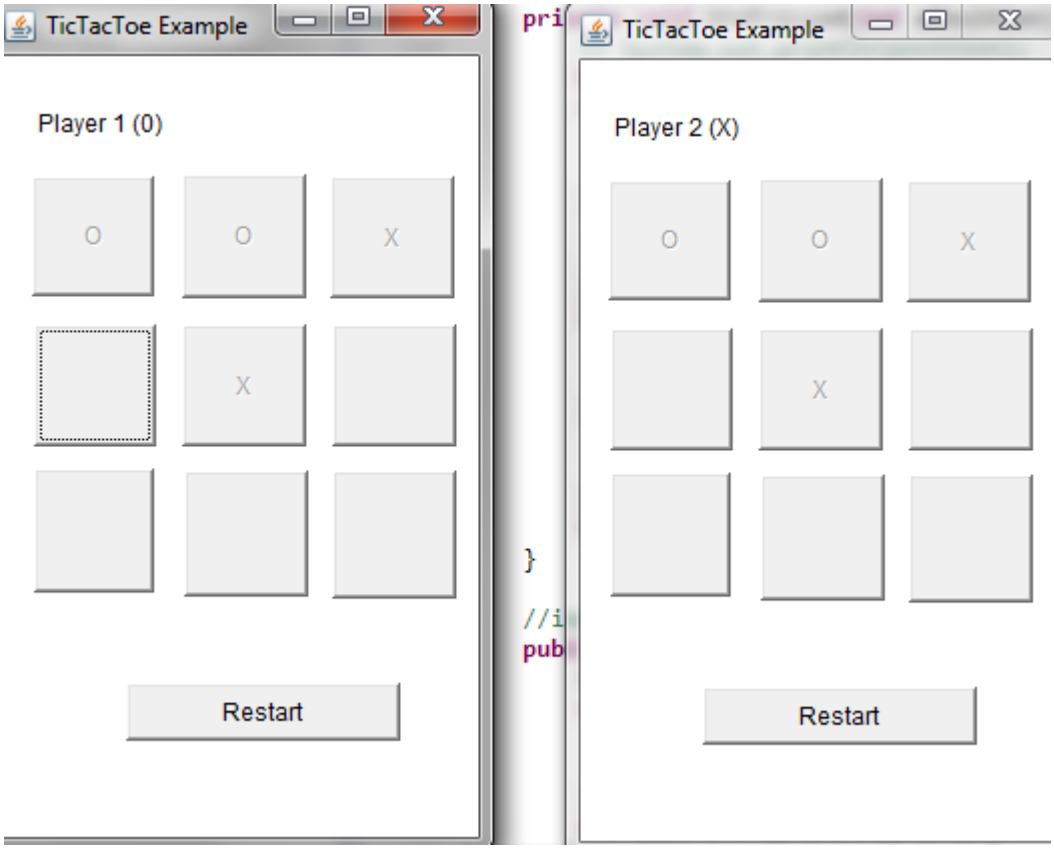


1.5. Click *OK* at the bottom. Now you can see a TicTacToe game board shown on the smartphone.
1.6. After installing the application to the Android phone, you can unplug cable and play it. The picture shows two Samsung NEXUS phones playing the TicTacToe game.



2. On Desktop

Right click the project name and choose *Run as -> Java application*. You can deploy it on two PCs or you can start two instances at the local machine. When you start the application then you can play with it. The picture shows the TicTacToe game running on the same machines.



3. Run the application on PCs and smartphones at the same time

We can deploy the application on a laptop and an Android phone, player can play game each other.

3.1 Start a wireless router and set up the laptop and the Android phone using the same wireless.
3.2 Run the TicTacToe desktop version on the laptop.
3.3 Run the TicTacToe Android version on the Android phone.
After two peers find each other they can start to play game.