**RemoteView**

**com.ipl.training.induction.draughts.view.remote.RemoteView**

**Class Specification**

**Project Reference 999/170**

**Document Reference RemoteView\_cs**

**12 May 2014**

**Issue 11**

**Copyright Notice**

This document has been prepared by IPL Information Processing Limited (“IPL”) and, subject to any existing rights of other parties, IPL is the owner of the copyright of this document. No part of this document may be copied, reproduced, stored in a retrieval system, disclosed to a third party or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior written permission of IPL.

**© IPL Information Processing Limited 2013**

**Document History**

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Issue | Author | Comments |
| 30/03/2010 | 1 | B. Bassil |  |
| 08/11/2010 | 2 | C Harrison | Corrections after first year of the course |
| 29/11/2010 | 3 | R Ferrario | Updated to fix issue with connecting after a game had finished |
| 10/12/2010 | 4 | C Harrison | Diagrams updated with methods introduced in issue 3 |
| 02/12/2011 | 5 | Phil Lewis, C Harrison | Updated to reflect new Java standard, including diagrams. |
| 16/10/2012 | 6 | Andrew Jenkins | Fixed minor typo in Section 7.10 and updated diagram in section 3 – gameEndedNotify() was missing. |
| 21/05/2013 | 7 | C Harrison | Added information about GameRequestData |
| 14/06/2013 | 8 | C Harrison | Removed unnecessary casting from createRegistry and updated some processing logic to explain what’s going on. |
| 18/07/2013 | 9 | C Harrison | Fixed error in modelPropertyChange method – need getNewEvent() method. And removed ambiguity in processing logic for RemoteView(). |
| 16/09/13 | 10 | C Harrison | Minor clarifications to createRegistry(). |
| 12/05/14 | 11 | B Bassil | Updates for Java 7 multiple exception handler and udates to createRegistry. |
| 04/08/2014 | 12 | L Gilbraith | Update getType() to getViewType() as per the code refactor. |

Contents

[1 Class Identity 1](#_Toc393715357)

[2 Description 1](#_Toc393715358)

[3 Interface 2](#_Toc393715359)

[4 Structure 3](#_Toc393715360)

[5 Element Descriptions and Interfaces 4](#_Toc393715361)

[5.1 Public Methods 4](#_Toc393715362)

[5.1.1 createView 4](#_Toc393715363)

[5.1.2 gameRequest 4](#_Toc393715364)

[5.1.3 getType 6](#_Toc393715365)

[5.1.4 modelPropertyChange 7](#_Toc393715366)

[5.1.5 registerController 8](#_Toc393715367)

[5.1.6 sendClick 8](#_Toc393715368)

[5.1.7 disconnectNotify 9](#_Toc393715369)

[5.1.8 gameEndedNotify 9](#_Toc393715370)

[5.2 Private Methods 11](#_Toc393715371)

[5.2.1 connect 11](#_Toc393715372)

[5.2.2 RemoteView 11](#_Toc393715373)

[5.2.3 createRegistry 12](#_Toc393715374)

[5.2.4 disconnect 13](#_Toc393715375)

[5.2.5 gameEnded 13](#_Toc393715376)

[5.2.6 processRemote 14](#_Toc393715377)

[5.3 Public Fields 15](#_Toc393715378)

[5.4 Package Access Fields 15](#_Toc393715379)

[5.5 Private Fields 15](#_Toc393715380)

[6 Resource Requirements 16](#_Toc393715381)

[7 Test Plan 16](#_Toc393715382)

[7.1 setUp 16](#_Toc393715383)

[7.2 tearDown 16](#_Toc393715384)

[7.3 testResetLocalBlack 16](#_Toc393715385)

[7.4 testResetLocalWhite 17](#_Toc393715386)

[7.5 testSendClickOK 17](#_Toc393715387)

[7.6 testSendClickFail 17](#_Toc393715388)

[7.7 testConnectionFailed 17](#_Toc393715389)

[7.8 testErrorEvent 18](#_Toc393715390)

[7.9 testSendClickEvent 18](#_Toc393715391)

[7.10 testSendInvalidClick 18](#_Toc393715392)

[7.11 testDisconnectNotify 19](#_Toc393715393)

[7.12 testGameEndedNotify 19](#_Toc393715394)

[8 Scenarios 19](#_Toc393715395)

[8.1 Game Initiation 19](#_Toc393715396)

[8.2 Remote Game invocation 21](#_Toc393715397)

[8.3 Event handling 22](#_Toc393715398)

# Class Identity

Class Name - RemoteView

Class Identity - com.ipl.training.induction.draughts.view.remote.RemoteView

# Description

There is one remote view object of this class in each instance of the draughts game. Its purpose is to communicate with a remote instance of the game. It acts as both an RMI server and client.

As an RMI client, the local remote view object consumes events produced by the local controller and communicates the required information to its counterpart remote view object in the remote instance of the game.

As an RMI server, the local remote view object’s ISharedObject methods are invoked by its counterpart in the remote instance of the game in response to events produced by the remote controller. It then invokes methods on the local controller to handle those events in the same way as a local view would do.

# Interface

|  |  |
| --- | --- |
| Attribute | Value |
| Visibility | Public |
| Modifiers | Final |
| Extends | - |
| Implements | IDraughtsView, ISharedObject, Serializable |

The remoteView class implements the IDraughtsView and ISharedObject interfaces. The only other public method it contains is the static factory method createView().



Figure Class Diagram of RemoteView

# Structure



Figure Structure diagram of RemoteView

Note:

1. The methods in the ISharedObject interface are called by the RemoteView object in the **remote** instance of the draughts game.
2. The methods in the IDraughtsView interface are called by the controller object in the **local** instance of the draughts game.

# Element Descriptions and Interfaces

Provides a view that is rendered on a remote game instance. This class must be created through the static factory createView().

## Public Methods

### createView

Creates a new instance of RemoteView as an IDraughtsView.

|  |  |
| --- | --- |
| Attribute | Value |
| Modifiers | static |
| Overrides | - |
| Throws | UnknownHostException when the name of the local host cannot be determined  RemoteException is passed up from the call to createRegistry |

**Inputs**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| - | - | - |

**Outputs**

|  |  |
| --- | --- |
| Type | Description |
| IDraughtsView | A new RemoteView |

**Processing Logic**

Return a new RemoteView.

### gameRequest

Called from a remote system indicating that it wants to play.

Connects to the RMI Registry, uses AcceptGameDialog to allow the user to specify the name and type of the local player. Constructs a GameData object from the received data plus local player and sets up the controller with this data.

|  |  |
| --- | --- |
| Attribute | Value |
| Modifiers | - |
| Overrides | ISharedObject.gameRequest |
| Throws | RemoteException |

**Inputs**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| request | GameRequestData | The name and colour of the player requesting the game |
| hostName | String | The host name to connect to |
| port | int | The remote port to connect to |

The GameRequestData is a static class defined in ISharedObject. The constructor is public and has the following interface:

**public** GameRequestData(**final** String playerName, // player requesting game

**final** PlayerColor player, // player requesting game

**final** String fen // initial board layout

**Outputs**

|  |  |
| --- | --- |
| Type | Description |
| boolean | Returns true on success |

**Processing Logic**

Locate the registry using LocateRegistry.getRegistry with the hostname and port parameters.

Set acceptingGame to true (this prevents an infinite loop when resetBoard is called).

Locate the SystemTest property using System.getProperty with the string “SystemTest” and check that it is defined.

If SystemTest is defined

* create a PlayerData object for the local player with name localHostName and type PlayerType.Computer.

If SystemTest isn’t defined

* create an AcceptGameDialog object using the name and colour from request and set this new object to be visible.
* If the dialog is ok then get the player data from the AcceptGameDialog and store it as a PlayerData object, otherwise return false.
* Call disconnect to disconnect from any previous games.

Create another PlayerData object with name hostname and type of Remote.

Lookup REMOTE\_ID in the registry and store this in the remoteObject field. If this throws a NotBoundException then return false.

Use request to determine the colour of the remote player and the FEN. Call resetBoard on the local controller, passing in a GameData object that contains this information.

At the very end of the method, use the “finally” keyword to put a block of code that resets acceptingGame to false, this requires a corresponding “try” at the very start of the function (make sure this is outside of any try/catch blocks).



Figure 3 gameRequest activity diagram

### getViewType

Called from the controller in the local instance of the Draughts game. Always returns Type.Remote. Used to indicate that this view is a remote view.

|  |  |
| --- | --- |
| Attribute | Value |
| Modifiers | Final |
| Overrides | IDraughtsView.getViewType |
| Throws | - |

**Inputs**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| - | - | - |

**Outputs**

|  |  |
| --- | --- |
| Type | Description |
| Type | Always returns Type.Remote |

This view will always be a remote view so Type.Remote will always be returned.

**Processing Logic**

Return Type.Remote.

### modelPropertyChange

Called by the controller in the local instance of the Draughts game when it needs to pass along a property change from the model. There are five different events that are acted upon by RemoteView, all others are ignored as they are for the local UI.

|  |  |
| --- | --- |
| Attribute | Value |
| Modifiers | - |
| Overrides | IDraughtsView.modelPropertyChange |
| Throws | - |

**Inputs**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| event | PropertyChangeEvent | The property change event, the name of which is defined within IDraughtsController |

**Outputs**

|  |  |
| --- | --- |
| Type | Description |
| - | - |

**Processing Logic**

Perform the following processing dependent upon the value of event.getPropertyName().

***IDraughtsController.REMOTE:***

Call processRemote passing in event.getNewValue cast to GameData.

***IDraughtsController.DISCONNECT:***

If we are not accepting a game then call disconnect.

***IDraughtsController.GAME\_END:***

Call gameEnded.

***IDraughtsController.ERROR:***

Call disconnect.

***IDraughtsController.SEND\_CLICK:***

If remoteObject isn’t null then call sendClick on remoteObject with getNewValue from event cast to an Integer.

If a RemoteException occurs then call error on the controller with a string “Connection Lost”. Set remoteObject to null and call connection failed on controller.

### registerController

Called by the controller in the local draughts game to register this view with the controller. This view will thus be sent view events from the controller.

|  |  |
| --- | --- |
| Attribute | Value |
| Modifiers | - |
| Overrides | IDraughtsView.registerController |
| Throws | - |

**Inputs**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| cont | IDraughtsController | The controller that this view is registered with |

**Outputs**

|  |  |
| --- | --- |
| Type | Description |
| - | - |

**Processing Logic**

Set controller to be cont.

### sendClick

Called from the remote system to inform the local draughts game that a square has been clicked. If the remoteView indicates the move is invalid a fatal error is displayed and a RemoteException thrown.

|  |  |
| --- | --- |
| Attribute | Value |
| Modifiers | - |
| Overrides | ISharedObject.sendClick |
| Throws | RemoteException |

**Inputs**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| squareID | Integer | The square that was clicked |

**Outputs**

|  |  |
| --- | --- |
| Type | Description |
| - | - |

**Processing Logic**

Call squareClicked on controller passing in **this** and the squareID parameter.

If squareClicked returns false then call error on the controller with the string “Remote host rejected move as invalid: “ and squareID as a string.

### disconnectNotify

Called from the remote system to inform the local draughts game that the remote player has disconnected. It is used to ensure that if a player disconnects whilst in the middle of a remote game then the other player is notified.

|  |  |
| --- | --- |
| Attribute | Value |
| Modifiers | - |
| Overrides | ISharedObject.disconnectNotify |
| Throws | RemoteException |

**Inputs**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| - | - | - |

**Outputs**

|  |  |
| --- | --- |
| Type | Description |
| - | - |

**Processing Logic**

Set remoteObject to null.

Call connectionFailed on controller.

### gameEndedNotify

Called from the remote system to inform the local draughts game that the game has ended, this ensures that no unexpected messages are sent after the game has ended.

|  |  |
| --- | --- |
| Attribute | Value |
| Modifiers | - |
| Overrides | ISharedObject.gameEndedNotify |
| Throws | - |

**Inputs**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| - | - | - |

**Outputs**

|  |  |
| --- | --- |
| Type | Description |
| - | - |

**Processing Logic**

Set remoteObject to null.

## Private Methods

### connect

Connect to an RMI registry specified remote host and port, storing the remote object stub from lookup() in the remoteObject field.. If the connection fails for any reason false is returned to indicate that the remote system could not be contacted.

|  |  |
| --- | --- |
| Attribute | Value |
| Modifiers | - |
| Overrides | - |
| Throws | - |

**Inputs**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| hostname | String | The host to connect to |
| port | int | The port number to connect to |

**Outputs**

|  |  |
| --- | --- |
| Type | Description |
| boolean | True if the connect is successful, otherwise false. |

**Processing Logic**

If connected, i.e. remoteObject isn’t null, then call disconnect.

Use LocateRegistry.getRegistry to get the registry for hostname and port. Call disconnect if a RemoteException is raised.

Call lookup on the registry to find the object identified as REMOTE\_ID. Set remoteObject to this value. Call disconnect if any exceptions are raised.

Return true if remoteObject isn’t null;

### RemoteView

Constructs a RemoteView object.

Determines whether to use PORT\_ONE or PORT\_TWO for the object being constructed. It will use PORT\_ONE unless PORT\_ONE is already in use by another instance of the Draughts game, in which case it will use PORT\_TWO. Then creates a Registry for this object.

|  |  |
| --- | --- |
| Attribute | Value |
| Modifiers | - |
| Overrides | - |
| Throws | UnknownHostException. RemoteException |

**Inputs**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| - | - | - |

**Outputs**

|  |  |
| --- | --- |
| Type | Description |
| - | - |

**Processing Logic**

Call InetAddress.getLocalHost().getHostName() and store result in localHostName.

Call LocateRegistry.getRegistry for PORT\_ONE to see if PORT\_ONE is already in use.

If a registry is found, this means that PORT\_ONE is possibly already in use. If so, call lookup(REMOTE\_ID) on it to determine if a connection has been made by another instance of the Draughts game. A connection has been made by another instance if none of the method’s exceptions are thrown.

If a connection has been made, this means that PORT\_ONE is definitely already in use.

If PORT\_ONE is not already in use set myPort to PORT\_ONE else set myPort to PORT\_TWO.

Call createRegistry with myPort. Do not catch the exception thrown by createRegistry as the constructor throws this to indicate a failure has occurred.

### createRegistry

Creates a local RMI registry, using port on the local host. Exports **this** (as an ISharedObject) and binds the exported remote object to the registry created.

|  |  |
| --- | --- |
| Attribute | Value |
| Modifiers | - |
| Overrides | - |
| Throws | RemoteException when getRegistry fails after a failed createRegistry. |

**Inputs**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| port | int | The port number to connect to |

**Outputs**

|  |  |
| --- | --- |
| Type | Description |
| - | - |

**Processing Logic**

Call LocateRegistry.createRegistry with port, storing the result in a localRegistry.

If the create fails then call LocateRegistry.getRegistry with port, storing the result in a localRegistry. If this call fails do not catch the RemoteException but llow it to be passed out of the module.

If the create succeeds, or if you manage to get one already createad, call UnicastRemoteObject.exportObject passing in **this** and port. If this call fails do not catch the RemoteException but llow it to be passed out of the module. Store the result as a local Remote object.

Call localRegistry.rebind passing in REMOTE\_ID as the name and the stored remote object.

### disconnect

Disconnects from the remote host. This method is called from a button press in the UI. Setting up a new thread for the disconnectNotify() call keeps the UI responsive.

|  |  |
| --- | --- |
| Attribute | Value |
| Modifiers | - |
| Overrides | - |
| Throws | - |

**Inputs**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| - | - | - |

**Outputs**

|  |  |
| --- | --- |
| Type | Description |
| - | - |

**Processing Logic**

Store remoteObject in a local variable.

Set remoteObject to null.

Create a new thread and override the run() method of this thread by typing:

*new Thread() {*

*public void run() {*

Inside the run method, if the local variable of type ISharedObject is not null then call disconnectNotify on it. Ignore any remote exception raised.

Finally, call start() on the new thread by typing “.start()” after the closing bracket.

### gameEnded

Tells the remote host not to send any more messages. This method is called from the game itself.

|  |  |
| --- | --- |
| Attribute | Value |
| Modifiers | - |
| Overrides | - |
| Throws | - |

**Inputs**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| - | - | - |

**Outputs**

|  |  |
| --- | --- |
| Type | Description |
| - | - |

**Processing Logic**

Store remoteObject in a local variable.

Set remoteObject to null.

If the local variable of type ISharedObject is not null then call gameEndedNotify on it. If gameEndedNotify throws a RemoteException, catch and ignore it.

### processRemote

Called by modelPropertyChange when a Remote game request is received. processRemote will locate which player is to be played by a remote instance and create a GameRequest.

|  |  |
| --- | --- |
| Attribute | Value |
| Modifiers | - |
| Overrides | - |
| Throws | - |

**Inputs**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| newValue | GameData | The GameData received as part of the remote request. |

**Outputs**

|  |  |
| --- | --- |
| Type | Description |
| - | - |

**Processing Logic**

If we are already accepting a game then perform no processing, otherwise determine which host and port to connect to. The host is defined in newValue – use the getHost() method. The port is PORT\_TWO if this instance is already using PORT\_ONE as stored in myPort and if the remote host is this machine (hint: you will need to use the equalsIgnoreCase method on the host), otherwise use PORT\_ONE.

Call the private connect() method to connect to the remote host. If the connection fails call connectionFailed on controller.

If the connection succeeds then use newValue to determine the type of each player. Create a GameRequestData object using the local player’s name, local player’s colour and FEN from newValue.

Call gameRequest on the remote object with the GameRequestData, localHostName and myPort. Store the result in a local variable.

If the game request returned false then set remoteObject to null and call newGame on controller.

If a RemoteException occurs call connectionFailed on controller.

## Public Fields

None.

## Package Access Fields

None.

## Private Fields

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Type | Attributes | Value | Description |
| serialVersionUID | long | static final | -7195469229687381137L | Class version number |
| PORT\_ONE | int | static final | 1889 | The default port that will be used for the RMI registry |
| PORT\_TWO | int | static final | PORT\_ONE +1 | The port used for the RMI Registry if PORT\_ONE is in use. This occurs when this is the second instance on 1 machine. |
| REMOTE\_ID | String | static final | IPLDraughts | The identifier of the remote object in the registry |
| myPort | int | - | -1 | The port in use for this view.  Will be PORT\_ONE or PORT\_TWO |
| controller | IDraughtsController | transient | - | The controller |
| remoteObject | ISharedObject | - | null | The remote draughts view, will be null if not connected |
| localHostName | String | - | - | The name of the local host |
| acceptingGame | boolean | - | false | This is true if this instance is accepting a game request. |

# Resource Requirements

None.

# Test Plan

## setUp

For all tests create a mock of IDraughtsController

Set the system property “SystemTest”

Set up a host string with InetAddress.getLocalHost().getHostName()

Create a test view.

## tearDown

At the end of each test unbind “IPLDraughts” from registries on ports 1889 and 1890. Ignore any exceptions.

## testResetLocalBlack

This test checks that requesting a game with a remote BlackPlayer causes resetBoard to be called with the remote player marked a the black player.

Create a PlayerData with the host set to the local machine and the type being REMOTE.

Create a PlayerData with the host set to the local machine and the type being COMPUTER.

Create an expected GameData with the remote player as black and computer player as white. Setup some PDN.

Call registerController on the test view.

Call gameRequest on the test view passing in the game request, the host as the local machine and port as 1889.

Check that the testView reports its type as REMOTE.

Check resetBoard is called on the controller. Use GameDataMatcher to check the gamedata is as per the expected game data.

## testResetLocalWhite

This test checks that requesting a game with a remote WhitePlayer causes resetBoard to be called with the remote player marked a the white player.

Create a PlayerData with the host set to the local machine and the type being REMOTE.

Create a PlayerData with the host set to the local machine and the type being COMPUTER.

Create an expected GameData with the computer player as black and remote player as white. Setup the PDN to be (String)null.

Call registerController on the test view.

Call gameRequest on the test view passing in the game request, the host as the local machine and port as 1889.

Check that the testView reports its type as REMOTE.

Check resetBoard is called on the controller. Use GameDataMatcher to check the gamedata is as per the expected game data.

## testSendClickOK

This test ensures that when sendClick is called the value is passed to squareClicked in the controller.

Call registerController on the test view.

Call sendClick(5) on the test view.

Check squareClicked is called with parameters of the testView and 5. Return true.

## testSendClickFail

This test ensures that when sendClick is called the value is passed to squareClicked in the controller and the controller indicates this move is invalid that error is called.

Call registerController on the test view.

Call sendClick(6) on the test view.

Check squareClicked is called with parameters of the testView and 6. Return false.

Check error is called on the controller with message "Remote host rejected move as invalid: 6"

## testConnectionFailed

This test checks that a connection failed error occurs when a REMOTE event is received and the other view does not exist.

Create a PropertyChangeEvent for property REMOTE. Set the newValue to be a GameData with a black computer player and white remote player.

Call registerController on the test view.

Call modelPropertyChange with the event from above.

Check that connectionFailed is called on the controller.

## testErrorEvent

This test checks the remote view responds correctly to an ERROR event

Create a PropertyChangeEvent for property ERROR. Set the newValue to be a GameData with a black computer player and white remote player.

Call registerController on the test view.

Call modelPropertyChange with the event from above.

## testSendClickEvent

This test ensures that a when a view is connected to a remote view that a SEND\_CLICK event results in the remote view calling squareClicked on the controller.

Create a PlayerData with the host set to the local machine and the type being REMOTE.

Create a PlayerData with the host set to the local machine and the type being COMPUTER.

Create an input GameData with the remote player as black and computer player as white. Setup some PDN.

Create an expected GameData with the computer player as black and remote player as white. Setup some as above

Create a second RemoteView and register it with the controller.

Call registerController on the test view and second RemoteView.

Create a PropertyChangeEvent for property REMOTE. Set the newValue to be a input game data.

Create a PropertyChangeEvent for property SEND\_CLICK. Set the newValue to be 7.

Call modelPropertyChange with the REMOTE event.

Call modelPropertyChange with the SEND\_CLICK event.

Check resetBoard is called on the controller. Use GameDataMatcher to check the gamedata is as per the expected game data.

Check squareClicked is called on the controller with the view being the second view and the square being 7. Return true.

## testSendInvalidClick

This test ensures that a when a view is connected to a remote view that an SEND\_CLICK event results in the remote view raising an error.

Create a PlayerData with the host set to the local machine and the type being REMOTE.

Create a PlayerData with the host set to the local machine and the type being COMPUTER.

Create an input GameData with the remote player as black and computer player as white. Setup some PDN.

Create an expected GameData with the computer player as black and remote player as white. Setup some as above

Create a second RemoteView and register it with the controller.

Call registerController on the test view and second RemoteView.

Create a PropertyChangeEvent for property REMOTE. Set the newValue to be a input game data.

Create a PropertyChangeEvent for property SEND\_CLICK. Set the newValue to be 7.

Call modelPropertyChange with the REMOTE event.

Call modelPropertyChange with the SEND\_CLICK event.

Check resetBoard is called on the controller. Use GameDataMatcher to check the gamedata is as per the expected game data.

Check squareClicked is called on the controller with the view being the second view and the square being 7. Return false.

Check error is called on the controller with message "Remote host rejected move as invalid: 7"

## testDisconnectNotify

This test ensures that calling disconnectNotify correctly disconnects and does not leave any references.

Call registerController on the test view.

Call disconnectNotify on the test view.

Check connectionFailed is called.

## testGameEndedNotify

This test ensures that calling gameEndedNotify correctly disconnects and does not leave any references.

Call registerController on the test view.

Call gameEndedNotify on the test view.

Check connectionFailed is not called.

# Scenarios

## Game Initiation

Shows the process that creates and registers a remote view.



Figure Creation of views

## Remote Game invocation

The following diagrams show the interaction that connects two remote games. The first diagram shows the creation of a gameRequest message to the remote game instance. The second shows the acceptance of a game request from a remote game instance.



Figure 5Send Game Request to Remote Game



Figure Accept Game Request from Remote Game

## Event handling

All events received by the controller are passed using modelPropertyChange.



Figure Event Received from Controller

**- End of Document -**