**DraughtsModel**

**com.ipl.training.induction.draughts.model.DraughtsModel**

**Class Specification**

**Project Reference 999/170**

**Document Reference DraughtsModel\_cs**

**29 July 2013**

**Issue 7**

**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 |
| 10/06/2010 | 1 | Brian Bassil | Issued |
| 28/09/2010 | 2 | C Harrison | Corrections after first run of the course |
| 08/11/2010 | 3 | C Harrison | Corrected test plans to match order of firePropertyChanges in setClick method |
| 02/12/2011 | 4 | Phil Lewis, C Harrison | Updated to reflect new Java standard, minor typo corrections, including diagrams. |
| 13/06/2013 | 5 | C Harrison | Updated processing logic to explain what’s going on. |
| 25/07/2013 | 6 | C Harrison | Corrected setClick() – it returns a Boolean. |
| 29/07/2013 | 7 | C Harrison | Corrected setClick() – make sure oldHints is set up correctly. |

Contents

[1 Package Identity 1](#_Toc358907758)

[2 Description 1](#_Toc358907759)

[3 Interface 1](#_Toc358907760)

[4 Structure 1](#_Toc358907761)

[5 Element Descriptions 2](#_Toc358907762)

[5.1 Public Methods 2](#_Toc358907763)

[5.1.1 addPropertyChangeListener 2](#_Toc358907764)

[5.1.2 createModel 2](#_Toc358907765)

[5.1.3 errorMsg 3](#_Toc358907766)

[5.1.4 export 3](#_Toc358907767)

[5.1.5 getCurrentPlayer 4](#_Toc358907768)

[5.1.6 newGame 4](#_Toc358907769)

[5.1.7 setClick 6](#_Toc358907770)

[5.1.8 undo 8](#_Toc358907771)

[5.2 Package Access Methods 10](#_Toc358907772)

[5.2.1 getCurrentLayout 10](#_Toc358907773)

[5.3 Protected Methods 10](#_Toc358907774)

[5.4 Private Methods 10](#_Toc358907775)

[5.4.1 DraughtsModel 10](#_Toc358907776)

[5.4.2 doMove 11](#_Toc358907777)

[5.4.3 findPossibleMoves 12](#_Toc358907778)

[5.4.4 changeTurn 13](#_Toc358907779)

[5.4.5 checkSquare 13](#_Toc358907780)

[5.4.6 firePropertyChange 14](#_Toc358907781)

[5.4.7 loadPlayer 15](#_Toc358907782)

[5.4.8 writePDNBody 16](#_Toc358907783)

[5.4.9 writePDNHeader 16](#_Toc358907784)

[5.5 Public Fields 18](#_Toc358907785)

[5.6 Package Access Fields 18](#_Toc358907786)

[5.7 Protected Fields 18](#_Toc358907787)

[5.8 Private Fields 18](#_Toc358907788)

[6 Resource Requirements 18](#_Toc358907789)

[7 Test Plan 18](#_Toc358907790)

[7.1 setUp 19](#_Toc358907791)

[7.2 tearDown 19](#_Toc358907792)

[7.3 testErrorMsg 19](#_Toc358907793)

[7.4 testNewGame 19](#_Toc358907794)

[7.5 testNewGameWithFEN 19](#_Toc358907795)

[7.6 testNewRemoteGame 20](#_Toc358907796)

[7.7 testExport 20](#_Toc358907797)

[7.8 testInvalidClick 21](#_Toc358907798)

[7.9 testValidClicks 21](#_Toc358907799)

[7.10 testNewGameFollowingSelection 22](#_Toc358907800)

[7.11 testMoves 22](#_Toc358907801)

[7.12 testCapturingMove 23](#_Toc358907802)

[7.13 testUndoNoHistory 24](#_Toc358907803)

[7.14 testUndo 25](#_Toc358907804)

[7.15 testUndoFinishedGame 26](#_Toc358907805)

[8 Scenarios 26](#_Toc358907806)

[9 Supporting Documentation 26](#_Toc358907807)

# Package Identity

Class Name - DraughtsModel  
Class Identity - com.ipl.training.induction.draughts.model.DraughtsModel

# Description

# Interface

|  |  |
| --- | --- |
| Attribute | Value |
| **Visibility** | **Public** |
| **Modifiers** | **Final** |
| **Extends** | **-** |
| **Implements** | **IDraughtsModel** |

# Structure



# Element Descriptions

## Public Methods

### addPropertyChangeListener

Registers a property change listener with the DraughtsModel. All registered listeners will be notified about property change events.

|  |  |
| --- | --- |
| Attribute | Value |
| Modifiers | - |
| Overrides | IDraughtsModel.addProperyChangeListener() |
| Throws | - |

**Inputs**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| listener | PropertyChangeListener | The property change listener to register |

**Outputs**

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

**Processing Logic**

Call propertyChangeSupport.addProperyChangeListener with listener.

### createModel

Creates a new instance of DraughtsModel via a call to the private constructor but only returns the interface. This hides the implementation details from the caller.

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

**Inputs**

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

**Outputs**

|  |  |
| --- | --- |
| Type | Description |
| IDraughtsModel | A newly constructed instance of DraughtsModel. |

**Processing Logic**

* Construct a new DraughtsModel via the private constructor and return it.

### errorMsg

Used to inform the model that an error has occurred. This causes property change events to be fired, one to display an error message and another to trigger a new game.

|  |  |
| --- | --- |
| Attribute | Value |
| Modifiers | - |
| Overrides | IDraughtsModel.errorMsg() |
| Throws | - |

**Inputs**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| errorMsg | String | The error message to display |

**Outputs**

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

**Processing Logic**

* firePropertyChange(IDraughtsController.ERROR, null, errorMsg);
* firePropertyChange( IDraughtsController.NEW\_GAME, null, 1 );

### export

Writes a PDN file containing the current board layout and the game history to the Writer specified by writer.

|  |  |
| --- | --- |
| Attribute | Value |
| Modifiers | - |
| Overrides | IDraughtsModel.export() |
| Throws | - |

**Inputs**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| writer | java.io.Writer | The writer to export the PDN to. |

**Outputs**

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

**Processing Logic**

* If writer is null
  + return
* Otherwise
  + Construct a new PrinterWriter with writer
  + Call writePDNHeader
  + Call writePDNBody
  + Close the PrintWriter

### getCurrentPlayer

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

**Inputs**

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

**Outputs**

|  |  |
| --- | --- |
| Type | Description |
| PlayerColor | The current player. |

**Processing Logic**

* Return boardLayout.getCurrentPlayer().

### newGame

Sets up the DraughtsModel according to the supplied GameData object. If the GameData object specifies a FEN tag then this is used to layout the board, otherwise the default layout is used. Any existing computer players must be stopped and discarded. New computer players are created and started if required.

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

**Inputs**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| newGameData | GameData | A GameData object describing the new game. |

**Outputs**

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

**Processing Logic**

First – if we are in the middle of network game then disconnect before we start again:

* First – if we are in the middle of network game then disconnect before we start again:  
  if gameData is not null and if gameData is a network game then call firePropertyChange(IDraughtsController.DISCONNECT, null, “Disconnected” ).

Tidy up all old data and remove any computer players of the old game:

* Reset turnHistory
* Store gameData in local variable, oldData – we use this later when in firePropertyChange()
* Set gameData to newGameData
* Iterate through computerPlayers from the old game, interrupting and removing each one

Now tell the controller if a REMOTE game has been requested:

* If newGameData.isNetworkGame() fire call firePropertyChange( IDraughtsController.REMOTE, oldData, gameData )

Set up the display and any computer players for the new game:

* If currentlySelectedSquare != 0, tidy up the display before resetting the board to the new layout
  + Use boardLayout to get the possible moves for the current player from the currently selected square. Store this in a local variable moveData.
  + Call firePropertyChange(IDraughtsController.SET\_HINTS, moveData.getMoves(), new HashSet<Integer>() )
  + firePropertyChange(IDraughtsController.SET\_SELECTED, currentlySelectedSquare, 0 );
  + set currentlySelectedSquare to 0.
* Store boardLayout in a localVariable, old
* Construct a new BoardLayout with newGameData.getFEN and assign to boardLayout. If an IllegalArgumentException is thrown report it via errorMsg() and return.
* Loop over the return from BoardLayout.diff(old, boardLayout) – this resets the board to the new game layout required.
  + Call firePropertyChange( IDraughtsController.SET\_SQUARE, null, change ) where change is the loop element.
* If the newGameData specifies any players are of type Computer then create the required players using loadPlayer() and add them to computerPlayers.
* firePropertyChange( IDraughtsController.GAME\_DATA\_UPDATED, oldData, gameData );
* firePropertyChange( IDraughtsController.PLAYER, null, getCurrentPlayer() )
* firePropertyChange(IDraughtsController.STATUS, null, "Ready")
* Loop over computerPlayers starting each one.

### setClick

This method is used to indicate to the model that a square has been ‘clicked’ by the user.

|  |  |
| --- | --- |
| Attribute | Value |
| Modifiers | synchronized |
| Overrides | IDraughtsModel.setClick() |
| Throws | - |

**Inputs**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| squareID | Integer | The square that has been clicked |
| propagate | boolean | Whether or not a SEND\_CLICK event should be fired. |

**Outputs**

|  |  |
| --- | --- |
| Type | Description |
| Boolean | true on success, false otherwise |

**Processing Logic**

State transitions:



**Processing Logic**

* If checkSquare(squareID) returns false
  + Return false – i.e. ignore any invalid clicks
* Create local variable – call it oldHints – of type Set<Integer> and initialise it with a new HashSet.
* Call getCurrentPlayer and store the return in a final local variable called initialPlayer
* Store currentlySelectedSquare in a final variable called oldSelection
* If currentlySelectedSquare is 0 (i.e. no square is currently selected)
  + Set currentlySelectedSquare to squareID
* Otherwise this should be the destination square for the currently selected piece:
  + Call boardLayout.getMoves(initialPlayer).get(oldSelection) and store the return in a local variable possibleMoves.
  + If possibleMoves is non null (i.e. it’s possible to move from the currently selected square)
    - Add the return from possibleMoves.getMoves to oldHints (i.e. store those possible moves in oldHints – we’ll need to remove the hint highlights from these squares when we make the move)
  + If oldHints.contains(squareID) (i.e. if the square clicked is a possible move – do the move)
    - Call doMove(oldSelection, squareID). If an InvalidMoveException is thrown return false.
  + Otherwise if boardLayout.canSelect(squareID) (i.e. if it’s selectable – change the selection to this new square)
    - currentlySelectedSquare = squareID
  + Otherwise
    - firePropertyChange( IDraughtsController.STATUS, null, "Cannot select " + squareID );
    - return false;
* Call findPossibleMoves(initialPlayer) and store the return in a local variable newHints.
* Call firePropertyChange( IDraughtsController.SET\_HINTS, oldHints, newHints );
* Call firePropertyChange( IDraughtsController.SET\_SELECTED, oldSelection, currentlySelectedSquare );
* Call firePropertyChange( IDraughtsController.PLAYER, initialPlayer, getCurrentPlayer() );
* If propagate is true
  + Call firePropertyChange( IDraughtsController.SEND\_CLICK, -1, squareID );
* If initialPlayer is not the same as the currentPlayer
  + Call changeTurn()
* Return true

### undo

Makes the model attempt to undo back to the last human move. Note that if there are only human players then only one turn will be undone. If there is a computer player then up to two turns may be undone – the current turn and if the previous turn was a computer turn, then that turn as well.

|  |  |
| --- | --- |
| Attribute | Value |
| Modifiers | synchronized |
| Overrides | IDraughtsModel.undo() |
| Throws | - |

**Inputs**

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

**Outputs**

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

**Processing Logic**

* if boardLayout.getGameState != GameState.IN\_PROGRESS
  + firePropertyChange( IDraughtsController.STATUS, null, "Cannot undo a finished game" );
* otherwise if turnHistory.isEmpty
  + firePropertyChange( IDraughtsController.STATUS, null, "No move to undo" );
* otherwise if computerPlayers.size() == 2
  + firePropertyChange( IDraughtsController.STATUS, null, "Cannot undo when only using computer players" );
* Otherwise
  + Create a new local variable called poppedTurns of type Stack<TurnData> and initialise it with a new Stack
  + Create a local variable of type PlayerColor called computerPlayer. Initalise it to null.
  + If computerPlayers is not empty set computerPlayer to the color of the computer player
  + Create a local variable called initialLayout of type BoardLayout and initialise it with boardLayout
  + Try
    - Do
      * Call turnHistory.pop() and assign the return to a new variable called pop
      * Push pop onto poppedTurns
      * Set boardLayout to pop.getBoardLayout()
    - While computerPlayer != null and boardLayout.getCurrentPlayer() equals computerPlayer
  + Catch EmptyStackException
    - While poppedTurns is not empty
      * Push the return from poppedTurns.pop() onto turnHistory.
    - Set boardLayout to initialLayout.
    - firePropertyChange( IDraughtsController.STATUS, null, "No move to undo" );
  + If currentlySelectedSquare != 0
    - Use initialLayout to get the possible moves for the selected square. Store this in a local variable moveData.
    - Call firePropertyChange(IDraughtsController.SET\_HINTS, moveData.getMoves(), new HashSet<Integer>() )
    - firePropertyChange(IDraughtsController.SET\_SELECTED, currentlySelectedSquare, 0 );
    - Set currentlySelectedSquare to 0.
  + firePropertyChange( IDraughtsController.REMOVE\_HISTORY, null, poppedTurns.size() );
  + Loop over the return from BoardLayout.diff(initialLayout, boardLayout) with loop variable sq
    - Calling firePropertyChange( IDraughtsController.SET\_SQUARE, null, sq );
  + firePropertyChange( IDraughtsController.PLAYER, initialLayout.getCurrentPlayer(), boardLayout.getCurrentPlayer() )
  + If initialLayout.getCurrentPlayer != boardLayout.getCurrentPlayer()
    - Call notifyAll().

## Package Access Methods

### getCurrentLayout

Returns a BoardLayout representing the board state.

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

**Inputs**

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

**Outputs**

|  |  |
| --- | --- |
| Type | Description |
| BoardLayout | The current board layout |

**Processing Logic**

* return layout

## Protected Methods

None

## Private Methods

### DraughtsModel

This is an empty private constructor to prevent use outside of this class.

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

**Inputs**

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

**Outputs**

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

**Processing Logic**

This method does nothing.

### doMove

This method performs a move and fires any related events.

|  |  |
| --- | --- |
| Attribute | Value |
| Modifiers | synchronized |
| Overrides | - |
| Throws | InvalidMoveException |

**Inputs**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| start | int | The start point for a move |
| end | Integer | The end of the move |

**Outputs**

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

**Processing Logic**

* Create a final BoardLayout called initialLayout and assign boardLayout to it
* Create a new Set<SquareData> called changes and initialise it with a new HashSet.
* Create a new BoardLayout using initialLayout, start and end and assign it to boardLayout.
* Call BoardLayout.diff(initialLayout, boardLayout) and add the return to changes.
* Iterator over changes with loop variable d
  + Call firePropertyChange(IDraughtsController.SET\_SQUARE, null, d)
* If currentTurnData is null or currentTurnData.getCurrentPosition != start
  + Construct a new TurnData with start and intialLayout and assign it to currentTurnData
* If boardLayout.getCaptured is not null
  + Add a capturing move to currentTurnData using end and boardLayout.getCaptured()
  + If the current player for boardLayout is the same as the one for intialLayout
    - Set currentlySelectedSquare to end
  + Otherwise
    - Set currentlySelectedSquare to 0
* Otherwise
  + Add end to currentTurnData as a normal move
  + Set currentlySelectedSquare to 0

### findPossibleMoves

Find all moves for a player on the currently selected square

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

**Inputs**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| player | PlayerColor | The player to find moves for |

**Outputs**

|  |  |
| --- | --- |
| Type | Description |
| Set<Integer> | A set of possible moves from the currently selected square, if no moves are possible the set will be empty. |

**Processing Logic**

* Create a new local variable possibleMoves of type Set<Integer> and initialise with a new HashSet.
* If getCurrentPlayer == player and currentlySelectedSquare is not 0
  + Call boardLayout.getMoves(player) and assign to a new local variable moves.
  + Call moves.get(currentlySelectedSquare) and assign to a new local variable moveData.
  + If moveData is not null
    - Add all of the moves from MoveData to possibleMoves.
* return possibleMoves.

### changeTurn

Handles processing related to the turn changing

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

**Inputs**

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

**Outputs**

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

**Processing Logic**

* Push currentTurnData onto turnHistory
* Call firePropertyChange with IDraughtsController.ADD\_HISTORY, null and currentTurnData.toString.
* Set currentTurnData to null
* Call boadLayout.getGameState and assign to a new local variable state.
* If state is not GameState.IN\_PROGRESS
  + Loop over computerPlayers interrupting each one and removing them
  + Call firePropertyChange(IDraughtsController.STATUS, null, “Game Over – “ + state).
  + Call firePropertyChange(IDraughtsController.GAME\_END, null, “Game Over”).
* Call this.notifyAll() to wake up all waiting threads.

### checkSquare

Returns true if the specified square can be selected, otherwise returns false

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

**Inputs**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| squareID | int | The square ID to check |

**Outputs**

|  |  |
| --- | --- |
| Type | Description |
| boolean | True if the squareID can be selected, otherwise false |

**Processing Logic**

* if squareID is the same as currentlySelectedSquare
  + return false
* if the boardLayout.getGameState is not GameState.IN\_PROGRESS
  + return false
* if currentlySelectedSquare is 0 and !boardLayout.canSelect(squareID)
  + Call firePropertyChange with IDraughtsController.STATUS, null, "Cannot select " + squareID.
  + Return false
* Return true

### firePropertyChange

Causes a propertyChangeEvent to be fired. This method makes use of propertyChangeSupport and if newVal is equal to oldVal then no event will be fired.

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

**Inputs**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| name | String | The name of the property that has changed |
| oldVal | Object | The old value of the property |
| newVal | Object | The new value of the property |

**Outputs**

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

**Processing Logic**

* Construct a new DraughtsChangeEvent with this, name, oldVal and newVal
* Call propertyChangeSupport.firePropertyChange() with the newly constructed DraughtsChangeEvent.

### loadPlayer

This method handles the dynamic loading of AbstractComputerPlayer objects based on system properties. The system properties used are IDraughtsModel.BLACK\_COMPUTER\_PLAYER and IDraughtsModel.WHITE\_COMPUTER\_PLAYER.

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

**Inputs**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| color | PlayerColor | The color that the new AbstractComputerPlayer will represent. |

**Outputs**

|  |  |
| --- | --- |
| Type | Description |
| AbstractComputerPlayer | A new AbstractComputerPlayer representing color. |

**Processing Logic**

* Declare a final String className but do not initialise it.
* Declare a final String packageName and initialise it to "com.ipl.training.induction.draughts.model.".
* if color is PlayerColor.BLACK
  + If the System property BLACK\_COMPUTER\_PLAYER is null
    - Throw an IllegalArgumentException with an appropriate message
  + Otherwise
    - Set the local variable className to packageName + the property value.
* if color is PlayerColor.WHITE
  + If the System property WHITE\_COMPUTER\_PLAYER is null
    - Throw an IllegalArgumentException with an appropriate message
  + Otherwise
    - Set the local variable className to packageName + the property value.
* try
  + Use this.getClass().getClassLoader().loadClass(className) to load the specified class and assign it to a local variable clazz.
  + Obtain the constructor using clazz.getConstructor(PlayerColor.class, DraughtsModel.class) and assign it to a local variable ctor.
  + Call ctor.newInstance(color, this) and assign the return to a variable newInstance.
  + If newInstance is an instanceof AbstractComputerPlayer return it, otherwise throw an IllegalArgumentException with an appropriate error message.
* catch IllegalArgumentException
  + rethrow the exception
* catch Exception
  + throw a new IllegalArgumentException with a suitable error message detailing the source.

### writePDNBody

Writes the turn history to the supplied PrintWriter.

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

**Inputs**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| writer | PrintWriter | The PrintWriter to use when writing the body. |

**Outputs**

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

**Processing Logic**

* Loop over the entire turn history from index 0.
  + If the index is divisible by 2
    - Print a newline
    - Print index/2+1
    - Print “.”.
  + Print the current element from the turn history
  + Print a space.

(The test on divisibility ensures that black and white moves are paired.)

### writePDNHeader

Writes out the header to the PDN file. This comprises the following as defined in the SRS

[Event "Induction"]

[Site "IPL"]

[Date "2010.04.08"]

[Round "1"]

[White "White"]

[Black "Black"]

[Result "0-2"]

[SetUp "1"]

[FEN "W:WK3,K8,K13,K15,28,K32:BK14,19,K22,K23,K24"]

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

**Inputs**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| writer | PrintWriter | The PrintWriter to use when writing the body. |

**Outputs**

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

**Processing Logic**

* Each tag must be on a new line
* The following tags are constants
  + Event
  + Site
  + Round
  + SetUp
* The month and day parts of the Date tag must be preceded with ‘0’ so they are always two characters
* The FEN tag must be immediately preceded by the SetUp tag
* The contents of the FEN tag should be created by calling boardLayout.toString().
* The result should be determined by using boardLayout.getGameState(), the relevant string are shown below
  + GameState.IN\_PROGRESS – “\*”
  + GameState.BLACK\_WIN – “1-0”
  + GameState.WHITE\_WIN – “0-1”
  + GameState.DRAW – “1/2-1/2”

## Public Fields

None

## Package Access Fields

None

## Protected Fields

None

## Private Fields

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Type | Attributes | Value | Description |
| boardLayout | BoardLayout | - | BoardLayout.EMPTY\_LAYOUT | The current board layout. |
| computerPlayers | Set  <AbstractComputerPlayer> | - | new HashSet  <AbstractComputerPlayer>() | Set of currently registered computer players |
| currentlySelectedSquare | int | - | 0 | The currently selected square – 0 for no square selected |
| currentTurnData | TurnData | - | null |  |
| gameData | GameData | - | - |  |
| propertyChangeSupport | PropertyChangeSupport | - | new PropertyChangeSupport( this ); |  |
| turnHistory | Stack<TurnData> | - | - | Game history |

# Resource Requirements

None

# Test Plan

DraughtsModel cannot be tested in isolation from AbstractComputerPlayer, BoardLayout, GameData or PlayerData. Because of this it is essential that these classes have been fully tested before attempting to test DraughtsModel.

Due to issues using JMock when unit testing code that makes use of Threads methods that requires the use of AbstractComputerPlayer will not be unit tested. This means that 100% code coverage will not be achieved.

## setUp

Create a DraughtsModel

Create a mock PropertyChangeListener and add it to the DraughtsModel

## tearDown

Clear the system property IDraughtsModel.BLACK\_COMPUTER\_PLAYER.

Clear the system property IDraughtsModel. WHITE\_COMPUTER\_PLAYER.

## testErrorMsg

This test checks that an ERROR event and a NEW\_GAME event are triggered by calls to IDraughtsModel.errorMsg.

Call errorMsg() with a known String (errorText) and verify that following property change events are fired

* IDraughtsController.ERROR with old value null and new value errorText
* IDraughtsController.NEW\_GAME with old value null and new value 1.

## testNewGame

This test checks that required events to set up a default layout are fired for a null FEN tag.

Create a new GameData object with two human players and a null FEN tag.

Call newGame with the GameData object

Verify the following events:

* The correct sequence of IDraughtsController.SET\_SQUARE for a default board layout
* IDraughtsController.GAME\_DATA\_UPDATED event is fired with the supplied GameData
* IDraughtsController.PLAYER with PlayerColor.BLACK
* IDraughtsController.STATUS with String “Ready”

## testNewGameWithFEN

This test checks that required events to set up a specific layout are fired for a non-null FEN tag.

Create a new GameData object with two human players and a null FEN tag.

Call newGame with the GameData object

Create a second GameData object with two human players and "W:B1,2,3:WK4,5,6" as the FEN tag.

Call newGame with the second GameData object

Verify the following events:

* The correct sequence of IDraughtsController.SET\_SQUARE for a default board layout
* An IDraughtsController.GAME\_DATA\_UPDATED event is fired with the supplied GameData
* IDraughtsController.PLAYER with PlayerColor.BLACK
* An IDraughtsController.STATUS with text “Ready”
* The correct sequence of IDraughtsController.SET\_SQUARE for the specified FEN tag
* An IDraughtsController.GAME\_DATA\_UPDATED event is fired with the supplied GameData
* IDraughtsController.PLAYER with PlayerColor.WHITE
* An IDraughtsController.STATUS with text “Ready”

## testNewRemoteGame

This test checks that the expected IDraughtsController.REMOTE event is fired when a new game is stated with a remote player.

Create a new GameData object with one human player and one remote player and a null FEN tag.

Call newGame with the GameData object

Verify the following events:

* IDraughtsController.REMOTE with the supplied GameData.
* The correct sequence of IDraughtsController.SET\_SQUARE for a default board layout
* IDraughtsController.GAME\_DATA\_UPDATED event is fired with the supplied GameData
* IDraughtsController.PLAYER with PlayerColor.BLACK
* IDraughtsController.STATUS with String “Ready”

## testExport

This test checks the export() method produces a valid PDN file.

Create a new GameData object with two human players (use “Bill” for BLACK and “Ben” for WHITE) and a null FEN tag.

Create a BoardLayout called defaultLayout with the FEN tag from the GameData object

Call newGame with the GameData object.

Create a StringWriter named writer

Call export(writer).

Create a StringReader named reader and initialise it with new StringReader(writer.toString)

Create a second GameData with two human players and reader.

Create a BoardLayout with the FEN tag from the second GameData

Assert that the new board layout is equal to defaultLayout.

Check that writer.toString() contains each of the following lines:

* [Event "Induction"]
* [Site "IPL"]
* [Date "<DATE>"] where <DATE> is the current date.
* [Round "1"]
* [White "Ben"]
* [Black " Bill "]
* [Result "\*"]
* [SetUp "1"]

Verify the following events:

* The correct sequence of IDraughtsController.SET\_SQUARE for a default board layout
* IDraughtsController.GAME\_DATA\_UPDATED event is fired with the supplied GameData
* IDraughtsController.PLAYER with PlayerColor.BLACK
* IDraughtsController.STATUS with String “Ready”

## testInvalidClick

This test checks that when it is the black players turn clicks on a white piece, and empty square or a black piece that cannot move sends a STATUS message saying it is invalid.

Create a new GameData object with two human players and a null FEN tag.

Call newGame with the GameData object

Call setClick on 32 with propagate as false (white piece), assert return is false.

Call setClick on 22 with propagate as false (empty square), assert return is false.

Call setClick on 6 with propagate as false (black piece that cannot move), assert return is false.

Verify the following events:

* The correct sequence of IDraughtsController.SET\_SQUARE for a default board layout
* IDraughtsController.GAME\_DATA\_UPDATED event is fired with the supplied GameData
* IDraughtsController.PLAYER with PlayerColor.BLACK
* IDraughtsController.STATUS with String “Ready”
* IDraughtsController.STATUS with “Cannot select 32”
* IDraughtsController.STATUS with “Cannot select 22”
* IDraughtsController.STATUS with “Cannot select 6”

## testValidClicks

Create a new GameData object with two human players and a null FEN tag.

Call newGame with the GameData object

Call setClick on 10 with propagate as false, assert return is true.

Call setClick on 6 with propagate as false, assert return is false.

Call setClick on 12 with propagate as false, assert return is true.

Verify the following events:

* The correct sequence of IDraughtsController.SET\_SQUARE for a default board layout
* IDraughtsController.GAME\_DATA\_UPDATED event is fired with the supplied GameData
* IDraughtsController.PLAYER with PlayerColor.BLACK
* IDraughtsController.STATUS with String “Ready”
* IDraughtsController.SET\_HINTS with old value empty set and new value a set containing 14 and 15
* IDraughtsController.SET\_SELECTED with 10
* IDraughtsController.STATUS with “Cannot select 6”
* IDraughtsController.SET\_HINTS with old value a set containing 14 and 15 and new value a set containing 16
* IDraughtsController.SET\_SELECTED with 12

## testNewGameFollowingSelection

Create a new GameData object with two human players and a null FEN tag.

Call newGame with the GameData object

Call setClick on 10 with propagate as false, assert return is true.

Call newGame with the GameData object

Verify the following events:

* The correct sequence of IDraughtsController.SET\_SQUARE for a default board layout
* IDraughtsController.GAME\_DATA\_UPDATED event is fired with the supplied GameData
* IDraughtsController.PLAYER with PlayerColor.BLACK
* IDraughtsController.STATUS with String “Ready”
* IDraughtsController.SET\_HINTS with old value empty set and new value a set containing 14 and 15
* IDraughtsController.SET\_SELECTED with 10
* IDraughtsController.SET\_HINTS with old value a set containing 14 and 15 and new value an empty set
* IDraughtsController.SET\_SELECTED with old value 10 and new value 0
* IDraughtsController.PLAYER with PlayerColor.BLACK
* IDraughtsController.STATUS with String “Ready”

## testMoves

This test confirms that clicking on a sequence of squares moves one black piece and one white piece. The BoardLayout is checked at the end to ensure it is as expected and the history is exported.

Create a new GameData object with two human players and a null FEN tag.

Create a BoardLayout using the FEN tag from the new GameData object and assign it to defaultLayout.

Create a new layout called layoutAfterMove1 using defaultLayout, 10 and 14.

Create a new layout called layoutAfterMove2 using layoutAfterMove2, 22 and 18.

If an InvalidMoveException is thrown call fail() and return.

Call newGame with the GameData object

Call setClick(10, false) and assert the return is true.

Call setClick(14, false) and assert the return is true.

Call setClick(22, false) and assert the return is true.

Call setClick(18, false) and assert the return is true.

Verify that layoutAfterMove2 is the same as the return from getCurrentLayout.

Create a new StringWriter called writer.

Call export(writer)

Create a StringReader named reader and initialise it with new StringReader(writer.toString)

Create a second GameData with two human players and reader.

Create a BoardLayout with the FEN tag from the second GameData

Assert that the new board layout is equal to layoutAfterMove2.

Check that writer.toString() contains the following turn history line:

* 1.10-14 22-18

Verify the following events:

* following newGame
  + The correct sequence of IDraughtsController.SET\_SQUARE for a default board layout
  + IDraughtsController.GAME\_DATA\_UPDATED event is fired with the supplied GameData
  + IDraughtsController.PLAYER with PlayerColor.BLACK
  + IDraughtsController.STATUS with String “Ready”
* following Click 10
  + IDraughtsController.SET\_HINTS with no old hints new hints 14 and 15
  + IDraughtsController.SET\_SELECTED, selecting square 10
* following Click 14
  + IDraughtsController.SET\_SQUARE setting square 10 empty
  + IDraughtsController.SET\_SQUARE setting square 14 as a BLACK\_PIECE
  + IDraughtsController.SET\_HINTS with old hints 14 and 15 and no new hints
  + IDraughtsController.SET\_SELECTED, selecting square 0
  + IDraughtsController.PLAYER with old PlayerColor.BLACK and new PlayerColor.WHITE
  + IDraughtsController.ADD\_HISTORY “10-14”
* following Click 22
  + IDraughtsController.SET\_HINTS with no old hints new hints 17 and 18
  + IDraughtsController.SET\_SELECTED, selecting square 22
* Following 18
  + IDraughtsController.SET\_SQUARE setting square 22 empty
  + IDraughtsController.SET\_SQUARE setting square 18 as a WHITE\_PIECE
  + IDraughtsController.SET\_HINTS with old hints 17 and 18 and no new hints
  + IDraughtsController.SET\_SELECTED, selecting square 0
  + IDraughtsController.PLAYER with old PlayerColor.WHITE and new PlayerColor.BLACK
  + IDraughtsController.ADD\_HISTORY “22-18”

## testCapturingMove

This test confirms that a capturing move fires the expected events.

Create a new GameData object with two human players and "B:BK15:W19"as the FEN tag..

Call newGame with the GameData object

Call setClick(15, false)

Call setClick(24, false)

Create a new StringWriter called writer.

Call export(writer)

Check that writer.toString() contains the following turn history line:

* 1.15x24

Verify the following events:

* following newGame
  + The correct sequence of IDraughtsController.SET\_SQUARE for a default board layout
  + IDraughtsController.GAME\_DATA\_UPDATED event is fired with the supplied GameData
  + IDraughtsController.PLAYER with PlayerColor.BLACK
  + IDraughtsController.STATUS with String “Ready”
* following Click 15
  + IDraughtsController.SET\_HINTS with no old hints new hints 24
  + IDraughtsController.SET\_SELECTED, selecting square 15
* following Click 24
  + IDraughtsController.SET\_SQUARE setting square 15 empty
  + IDraughtsController.SET\_SQUARE setting square 19 empty
  + IDraughtsController.SET\_SQUARE setting square 24 as a BLACK\_KING
  + IDraughtsController.SET\_HINTS with old hints 24 and no new hints
  + IDraughtsController.SET\_SELECTED, selecting square 0
  + IDraughtsController.PLAYER with old PlayerColor.BLACK and new PlayerColor.WHITE
  + IDraughtsController.ADD\_HISTORY “15x24”
  + IDraughtsController.STATUS “Game Over – BLACK\_WIN”
  + IDraughtsController.GAME\_END “Game Over”

## testUndoNoHistory

This test confirms that undo will send a status update if there is no game history.

Create a new GameData object with two human players and a null FEN tag.

Call newGame with the GameData object

Call undo()

Verify the following events:

* following newGame
  + The correct sequence of IDraughtsController.SET\_SQUARE for a default board layout
  + IDraughtsController.GAME\_DATA\_UPDATED event is fired with the supplied GameData
  + IDraughtsController.PLAYER with PlayerColor.BLACK
  + IDraughtsController.STATUS with String “Ready”
* Following undo
  + IDraughtsController.STATUS with String "No move to undo"

## testUndo

This test confirms that undo will revert a move.

Create a new GameData object with two human players and a null FEN tag.

Call newGame with the GameData object

Call setClick(10, false) and assert the return is true.

Call setClick(14, false) and assert the return is true.

Call undo()

Verify the following events:

* following newGame
  + The correct sequence of IDraughtsController.SET\_SQUARE for a default board layout
  + IDraughtsController.GAME\_DATA\_UPDATED event is fired with the supplied GameData
  + IDraughtsController.PLAYER with PlayerColor.BLACK
  + IDraughtsController.STATUS with String “Ready”
* following Click 10
  + IDraughtsController.SET\_HINTS with no old hints new hints 14 and 15
  + IDraughtsController.SET\_SELECTED, selecting square 10
* following Click 14
  + IDraughtsController.SET\_SQUARE setting square 10 empty
  + IDraughtsController.SET\_SQUARE setting square 14 as a BLACK\_PIECE
  + IDraughtsController.SET\_HINTS with old hints 14 and 15 and no new hints
  + IDraughtsController.SET\_SELECTED, selecting square 0
  + IDraughtsController.PLAYER with old PlayerColor.BLACK and new PlayerColor.WHITE
  + IDraughtsController.ADD\_HISTORY “10-14”
* Following undo
  + IDraughtsController.SET\_SQUARE setting square 10 as a BLACK\_PIECE
  + IDraughtsController.SET\_SQUARE setting square 14 as EMPTY
  + IDraughtsController.REMOVE\_HISTORY with 1
  + IDraughtsController.PLAYER, with new as PlayerColor.BLACK

## testUndoFinishedGame

This test verifies that undo correctly reports that undo is not possible once a game has been completed.

Create a new GameData object with two human players and "B:BK15:W19"as the FEN tag..

Call newGame with the GameData object

Call setClick(15, false)

Call setClick(24, false)

Call undo()

Verify the following events:

* following newGame
  + The correct sequence of IDraughtsController.SET\_SQUARE for a default board layout
  + IDraughtsController.GAME\_DATA\_UPDATED event is fired with the supplied GameData
  + IDraughtsController.PLAYER with PlayerColor.BLACK
  + IDraughtsController.STATUS with String “Ready”
* following Click 15
  + IDraughtsController.SET\_HINTS with no old hints new hints 24
  + IDraughtsController.SET\_SELECTED, selecting square 15
* following Click 24
  + IDraughtsController.SET\_SQUARE setting square 15 empty
  + IDraughtsController.SET\_SQUARE setting square 19 empty
  + IDraughtsController.SET\_SQUARE setting square 24 as a BLACK\_KING
  + IDraughtsController.SET\_HINTS with old hints 24 and no new hints
  + IDraughtsController.SET\_SELECTED, selecting square 0
  + IDraughtsController.PLAYER with old PlayerColor.BLACK and new PlayerColor.WHITE
  + IDraughtsController.ADD\_HISTORY “15x24”
  + IDraughtsController.STATUS “Game Over – BLACK\_WIN”
  + IDraughtsController.GAME\_END “Game Over”
* Following call to undo
  + IDraughtsController.STATUS "Cannot undo a finished game"

# Scenarios

# Supporting Documentation

<Add any further useful information for this package.>

**- End of Document -**