**AbstractComputerPlayer**

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

**Class Specification**

**Project Reference 999/170**

**Document Reference AbstractComputerPlayer\_cs**

**15 May 2014**

**Issue 8**

**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 |
| 10/6/2010 | 1 | Brian Bassil |  |
| 28/09/2010 | 2 | C Harrison | Corrections after first run of the course |
| 08/11/2010 | 3 | C Harrison | Section 5.2.1 2nd bullet point corrected |
| 24/11/2011 | 4 | Phil Lewis, C Harrison | Updated to reflect new Java standard, including diagrams. |
| 06/06/2013 | 5 | C Harrison | Updated to clarify some processing logic. |
| 22/07/2013 | 6 | C Harrison | And more clarifications |
| 16/09/2013 | 7 | C Harrison | More clarifications |
| 15/05/2014 | 8 | B Bassil | Reworked run method to make processing clearer. |
| 04/08/2014 | 9 | L Gilbraith | Correction to logic in TakeFirstLegalMove(). |

Contents

[1 Class Identity 1](#_Toc309909056)

[2 Description 1](#_Toc309909057)

[3 Interface 2](#_Toc309909058)

[4 Structure 2](#_Toc309909059)

[5 Element Descriptions 2](#_Toc309909060)

[5.1 Public Methods 2](#_Toc309909061)

[5.1.1 run 2](#_Toc309909062)

[5.2 Package Access Methods 5](#_Toc309909063)

[5.2.1 AbstractComputerPlayer 5](#_Toc309909064)

[5.2.2 determineMoves 5](#_Toc309909065)

[5.2.3 getPlayerColor 6](#_Toc309909066)

[5.3 Protected Methods 6](#_Toc309909067)

[5.3.1 getTimeRemaining 6](#_Toc309909068)

[5.3.2 stopPlaying 7](#_Toc309909069)

[5.3.3 storeClick 7](#_Toc309909070)

[5.4 Private Methods 8](#_Toc309909071)

[5.4.1 takeFirstLegalMove 8](#_Toc309909072)

[5.5 Public Fields 9](#_Toc309909073)

[5.6 Package Access Fields 9](#_Toc309909074)

[5.7 Protected Fields 9](#_Toc309909075)

[5.8 Private Fields 9](#_Toc309909076)

[6 Resource Requirements 9](#_Toc309909077)

[7 Test Plan 9](#_Toc309909078)

[8 Scenarios 9](#_Toc309909079)

[9 Supporting Documentation 9](#_Toc309909080)

# Class Identity

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

# Description

This is an abstract base class handling the processing common to all computer players.

The functionality includes:

* Waiting until it is the computer player’s turn.
* Timing out after 10 seconds.

The possible states are described by the following diagram:



# Interface

|  |  |
| --- | --- |
| Attribute | Value |
| **Visibility** | **Package** |
| **Modifiers** | **Abstract** |
| **Extends** | **Thread** |
| **Implements** | **-** |

# Structure



# Element Descriptions

## Public Methods

### run

This method handles waiting until it is the computer player’s turn and then calls the abstract method determineMoves() to take the move. If the computer calculates a move within the timeout period, 10 seconds then those moves are made. Otherwise the first legal moves will be made ignoring any choice by the algorithm

Once a move have been made we then wait for the model..

|  |  |
| --- | --- |
| Attribute | Value |
| Modifiers | Final |
| Overrides | Thread.run() |
| Throws | - |

**Inputs**

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

**Outputs**

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

**Processing Logic**

See the diagram on the next page.

Explanation of some decisions:

* [!stop] – while the model is playing and hasn’t stopped the computer player.
* [playerColor == model.getCurrentPlayer()] – if it is this computer player’s turn, then we do something, otherwise we wait for the model to tell us to try again (when the player turn changes).
* [getTimeRemaining < 0] – if we don’t have time to do a proper move (i.e. determineMoves() took too long), then we make sure the clickQueue is cleared so we don’t go round the following while loop. takeFirstLegalMove() is called which will play the first possible move.
* [!clickQueue.isEmpty()] – while there are some clicks to process (i.e. determineMoves() successfully returned some, within the required time limit), then we process those clicks in turn by calling setClick() on the model. Note that we go round this loop as many times as required to process all clicks for the move. When all clicks have been processed we come out of the loop. Note – optionally a short sleep can be added here so that the computer moves can be observed. See note below.

A try-catch block is required just inside the synchronised code to ensure that the model can interrupt when necessary.

**Note that if you get as far as playing a game with your code, you may want to add ‘sleep(500);’ to the code inside the while loop. This adds a half second delay between clicks and makes it easier to see the moves made by the computer player.**



## Package Access Methods

### AbstractComputerPlayer

This is the only constructor for AbstractComputerPlayer.

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

**Inputs**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| color | PlayerColor | The player that this AbstractComputerPlayer represents |
| model | DraughtsModel | The DraughtsModel that represents the game. |

**Outputs**

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

**Processing Logic**

* Store model in this.model
* Store color in this.playerColor
* Initialise this.clickQueue with a new ArrayDeque of type Integer
* Invoke super.setName() with this.getClass().getName() + " " + color.

Invoking super.setName() in the constructor ensures that the Thread has a useful name. This will aid debugging, making it clear which player the AbstractComputerPlayer represents and which implementation of AbstractComputerPlayer is running.

### determineMoves

An abstract method that must be overridden by sub-classes to determine which moves to take. Moves are stored by calling storeMove().

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

**Inputs**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| layout | BoardLayout | the current BoardLayout |

**Outputs**

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

### getPlayerColor

This method simply returns the colour of the player that the AbstractComputerPlayer represents.

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

**Inputs**

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

**Outputs**

|  |  |
| --- | --- |
| Type | Description |
| PlayerColor | The colour that this AbstractComputerPlayer represents |

**Processing Logic**

* return playerColor.

## Protected Methods

### getTimeRemaining

A method to query the time left before timeout. This can be used by subclasses to determine if they have sufficient time to do more processing.

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

**Inputs**

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

**Outputs**

|  |  |
| --- | --- |
| Type | Description |
| long | The time left for this move, in milliseconds. |

**Processing Logic**

* return TIME\_LIMIT – (System.currentTimeMillis() – startTime)

### stopPlaying

Used to stop the looping in the run() method.

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

**Inputs**

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

**Outputs**

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

**Processing Logic**

* Set stop to true.

### storeClick

Should be called from determineMoves() to store the square that should be clicked to perform this move.

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

**Inputs**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| squareID | Integer | The square that should be clicked on as part of this move. |

**Outputs**

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

**Processing Logic**

* Add squareID to clickQueue.

## Private Methods

### takeFirstLegalMove

Calls model.setClick() to peform the first available moves until the end of the computer player’s turn.

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

**Inputs**

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

**Outputs**

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

**Processing Logic**

In the situation where the computer player runs out of processing time or when there is no move selection algorithm implemented, takeFirstLegalMove will simply select the first move from the list of all available legal moves.

* While it is still this player’s turn according to model and stop is false
  + Get the possible moves from model.getCurrentLayout().getMoves(playerColor)
  + Loop through the set of possibleMoves.entrySet, use **entry** as the loop variable
    - If this entry shows that it’s possible to move from this square
      * Get the list of moves for each value in this **entry** and loop through each one in turn.
        + If any **move** is non zero then

Use model to call setClick(entry.getKey(), true)

Use model to call setClick(**move**, true)

Call **break** to stop both loops – you will need a label.

## Public Fields

None

## Package Access Fields

None

## Protected Fields

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Type | Attributes | Value | Description |
| playerColor | IDraughtsModel.PlayerColor | Final | - | The Player that is represented by this instance of AbstractComputerPlayer. |
| model | DraughtsModel | Final | - | The model that is being played against |

## Private Fields

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Type | Attributes | Value | Description |
| TIME\_LIMIT | int | Static, Final | 10000 | The time limit in milliseconds |
| clickQueue | Deque<Integer> | Final | - | Queue used in run() to hold the clicks that will be performed. |
| startTime | long | - | - | The time that the current turn began. |
| stop | boolean | - | false | Flag used to tell the run() method to stop looping. |

# Resource Requirements

None

# Test Plan

Due to the close ties between AbstractComputerPlayer and DraughtsModel and the limitations of JMock we are unable to unit test AbstractComputerPlayer.

AbstractComputerPlayer will be verified by inspection of the code and integration testing.

# Scenarios

# Supporting Documentation

None

**- End of Document -**