Software Documentation “Chess”

by Arwid (Arvydas) Bancewicz,

May 17, 2005

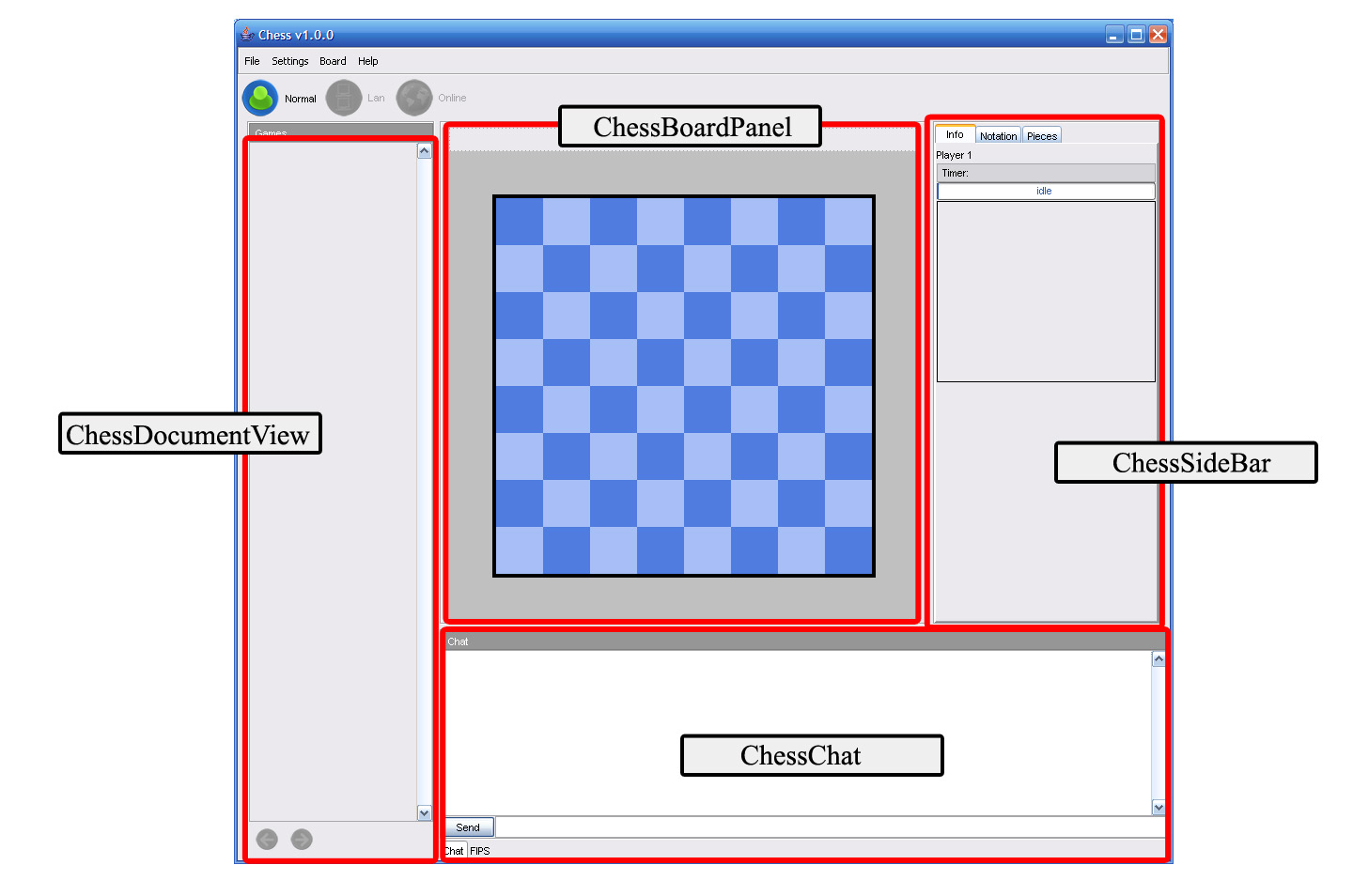
OBEA Computer Programming Contest, 2005

|  |  |
| --- | --- |
| **Tabled of Contents** |  |
| **Part 1: Graphical User Interface (GUI) Design** |  |
| **Part 2: Exploring the Move Algorithms**  Minimax explained  AlfaBeta explained  Move Algorithm Testing |  |
| **Part 3: Software Development Analysis**  Chess Applet  Chess Application |  |
|  |  |
| **Credits** |  |
|  |  |

**GUI (Graphical User Interface) Design**

The graphical user interface was designed with user friendliness and easiness in mind.

Although there are many panels, the four highlighted panels below are the main panels the user will work with.



The next pages contain information of each of these panels in the order as shown:

1. **The ChessBoardPanel**

2. **The ChessDocumentView**

3. **The ChessSideBar**

ChessChat is a simple panel so it does not need to be given attention.

**The ChessBoardPanel**

The **ChessBoardPanel** holds 3 components: 2 of which are **PlayerLabel**’s and a **ChessBoardMain** panel. There is a **PlayerLabel** for each of the two players, white and black. It displays the player’s name on the left and the player’s timer on the right. The **ChessBoardMain** represents the current selected game in the **ChessDocuments** list, which contains a list of all the present games. The board is highly interactive.

You can also access a context menu in the form of a popup by right clicking on the **ChessBoardMain.** The menu allows for quick access to commands, such as flip the board, zoom in and out, and switch to full mode. All of this is explained in the manual.

|  |
| --- |
|  |

**The DocumentView**

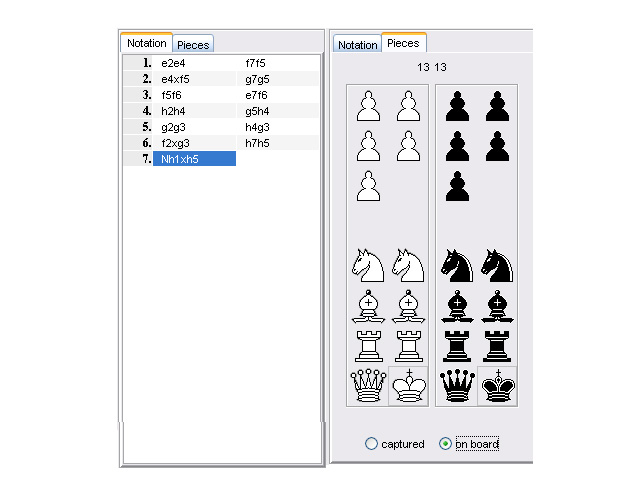
The **DocumentView** panel serves as to providing a visual representation in a **DocumentThumbnail** for each and every game. The **DocumentView** panel reads the **ChessDocuments** list which contains a list of all the present games and creates a **DocumentView** to visually represent it. Selecting a thumbnail will make the game it represents appear in the **ChessBoardPanel**.

The **DocumentThumbnail** has a nifty feature; you can minimize and maximize it as you please. The **DocumentThumbnail’s** can also be minimized or maximized all at once by right clicking on the header labelled “Games” to access a context popup menu.

|  |
| --- |
|  |

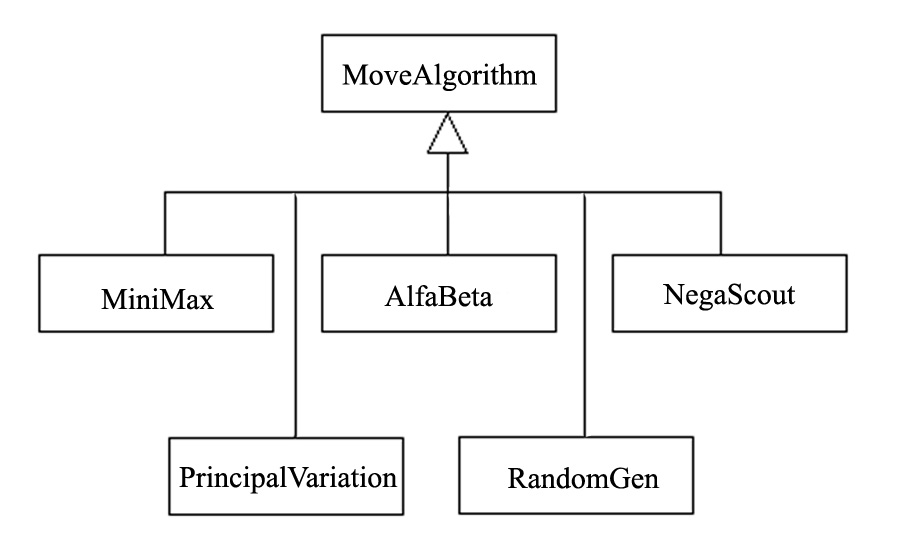
**The ChessSideBar**

The **ChessSideBar** contains two tabs: the **Notatation** tab and the **Pieces** tab. The **Notation** tab contains a table having the model **ChessTableModel.** The **ChessTableModel** contains a list of moves added by a **ChessGame** after moving a piece.The moves are then displayed in proper notation in **Notation’s** table. The **Pieces** tab shows all the pieces currently on the board or the pieces already captured. It gets the information about the number of pieces from a **PieceCounts** object. Every **Board** has a **PieceCounts** object.



**Exploring the Move Algorithms**

**Unified Modeling Language (UML) Diagram of MoveAlgorithm**



The UML diagram above is a class hierarchy for the MoveAlgorithm class and it’s subclasses.

It is recommended that the reader first consult the following java files before reading on.

These files are located in package chess.core (in folder “core”):

* Board
* Coord
* Move
* Piece

**Minimax explained**

Minimax is a method in decision theory for minimizing the expected maximum loss.

The minimax algorithm is a recursive algorithm for choosing the next move in a two player game such as chess. A value is associated with each position or state of the game. This value is computed by means of the position evaluation function located in the MoveAlgorithm class:

The value obtained from the estimate function indicates how good it would be for a player to reach that position.

*protected int estimate() {*

*stepcounter++;*

***int dc = getCost()-****est\_cost;*

***int dac = getAttackCost()-****est\_attackCost;*

***return dc\*10+dac;***

*}*

*private int getCost() {*

*int out = 0;*

*for(int c=0;c<8;c++)*

*for(int r=0;r<8;r++)*

***if(board.b[c][r]!=null)***

***out+=board.b[c][r].getCost();***

*return out;*

*}*

Note that **board.b** encapsulates an 8 by 8 array of **Piece** objects. The getCost function sums up the cost values for all the pieces currently on the board. A square containing no piece is a value of null in the **board.b** array. The cost for a piece is found in the **Constants** class:

***int[] cost = {1,5,5,10,40,100};***

The pawn is given the value of 1, 5 for the rook, 5 for the knight, 10 for the bishop, 40 for the queen, and 100 for the king. Notice that the values increase for the more valuable pieces. Also, if we look at the getCost method in the Piece class:

*public int getCost() {*

***return (white?1:-1)\*cost[type];***

*}*

The getCost function for each piece returns a positive value if the piece is white or a negative value if the piece is not white, but black. Thus, summing the cost values for all pieces in the estimate function returns a positive value if white is in favour or negative if black is in favour.

The getAttackCost function:

*private int getAttackCost() {*

*int out = 0;*

*for(byte c=0;c<8;c++)*

*for(byte r=0;r<8;r++)*

***if(board.b[c][r]!=null) {***

***Vector v = getRealAttacks(new Coord(c,r));***

***if(v!=null)***

*for(int k=0;k<v.size();k++) {*

*Move m = (Move)v.elementAt(k);*

***out += board.b[m.x2][m.y2].getCost();***

*}*

*}*

*return -out;*

*}*

The getAttackCost function is similar to getCost with the exception that it sums up the cost values for all the pieces that could be captured/killed. The function getAttackCost also returns the negative of the value because black pieces, which will return negative values, are good for the white player since white could capture these pieces. Thus the value returned is again positive if white is in favor or negative if black is in favor.

Now, going back to the return value of the estimate function:

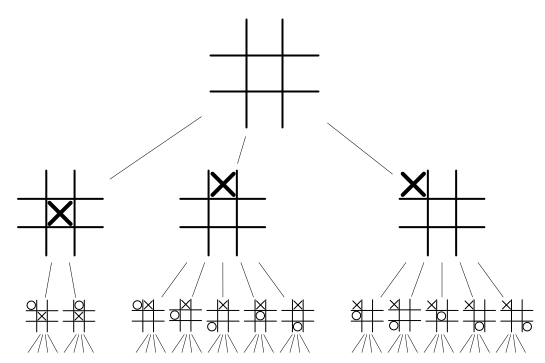
***return dc\*10+dac;***

This expression in words: *the cost x 10 + the attack cost*

Now that we know how a board position is evaluated, it makes sense that for the computer opponent to be effective, the algorithm will have to evaluate several board positions and even the board positions several moves ahead. The number of moves ahead is called the “look-ahead” or “**ply**”.

The minimax algorithm, as any algorithm, can be thought of as exploring the **nodes** of a **game tree**. The number of **nodes** to be explored usually increases exponentially with the number of **plies**. It is less than exponential if evaluating forced moves. For example, a white king checked by a black piece does not have many options but to get itself out of check.

A **game tree**, as mentioned earlier, is a tree of **nodes**. The example below is a **game tree** of the game tic-tac-toe. Notice the **game tree** contains all the possible **nodes** (positions in a game). It contains the number of possible different ways the game can be played. The tic-tac-toe **game tree** is a good example of a **game tree** since it has a very few number of possible games; 26 different nodes and 830 possible games to be exact. In contrast, the complete chess **game tree** has an immense number of **nodes;** not to even imagine the number of possible games**.**



Lets look at a chess situation where the white player has only three choices for moves. The white makes a move and now the black player has three choices for moves. This is illustrated in the example below:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Black chooses B1 | Black chooses B2 | Black chooses B3 |
| White chooses move A1 | +7 | -2 | +2 |
| White chooses move A2 | -1 | 1 | +4 |
| White chooses move A3 | 0 | -6 | +1 |

The values of the cells in-between are the board evaluation. For example, if white chooses A1 and black chooses B2 then A1 pays 2 to black (white loses). The simple minimax choice for white is A2 since the worst possible result is then having to pay 1. The none minimax choice for white is A3 since the worst possible result is to pay 6. This is basically how the minimax algorithm works.

**AlfaBeta explained**

The alfa-beta technique is called **alpha-beta pruning.** The technique reduces the number of **nodes** evaluated by the **minimax** algorithm. It prunes or cuts out parts of the **search tree** that are soo good for one player that the opponent will never allow them to be reached. Searching these **nodes** is obviously not necessary.

In sort, the **alpha-beta pruning** technique produces the same result as what minimax would, but without searching those unnecessary **nodes**. This vastly increases the actual time to search the **search tree.**

The algorithm maintains two values, alpha and beta, which represent the minimum score that the maximizing player is assured of and the maximum score that the minimizing player is assured of respectively. Initially alpha is minus infinity and beta is plus infinity.

The beta value will eventually become less than alpha, which means that the current position cannot be the result of best play by both players and hence does not need to be explored furthur. Thus this part of the tree is pruned/cut off.

The pseudocode of the alpha-beta (or alfa-beta) algorithm is given below. It is of that used by the **AlfaBeta** class.

NOTE: **“depth”** has the same meaning as “**ply”**

*evaluate (****depth****,* ***alpha****,* ***beta****)*

*if the* ***depth*** *is 0*

*return the value 0*

*get all the legal moves for this player*

*if the player is the minimizing node*

*if this player has some moves*

*for each of its moves*

***beta*** *= min (beta, evaluate (****depth-1****,* ***alpha****,* ***beta****))*

*if* ***beta*** *<=* ***alpha***

*return* ***alpha***

*return beta*

*if the player is the maximizing node*

*if this player has some moves*

*for each of its moves*

***alpha*** *= max (****alpha****, evaluate (****depth-1, alpha****,* ***beta****))*

*if* ***beta*** *<=* ***alpha***

*return* ***beta***

*return beta*

The other algorithms, Principal Variation and NegaScout, are very similar to alfa-beta while the RandomGen algorithm simply generates a random but legal move.

You can run a test of these algorithms by compiling and running **Tester**.java found in the package **chess.algorithms** (the algorithms folder). Follow the instructions in the file to see the algorithms in action.

A sample output is shown below.

\* Notice the large number of **nodes** and the relation between **# of nodes and time**. This is expected since a larger **search tree**, having more **nodes** than a smaller **search tree**, will take more time to search.

Move Algorithm Testing: Sample Output

White replies with AlfaBeta

reply => Bg1f3

time = seconds: 1 milliseconds: 625

depth # of nodes evaluation

5 13065 0

Black replies with AlfaBeta

reply => b7b6

time = seconds: 2 milliseconds: 187

depth # of nodes evaluation

5 29989 0

White replies with NegaScout

reply => Bb1a3

time = seconds: 1 milliseconds: 375

depth # of nodes evaluation

5 11798 0

Black replies with NegaScout

reply => c7c5

time = seconds: 1 milliseconds: 704

depth # of nodes evaluation

5 26079 0

White replies with PrincipalVariation

reply => Bb1c3

time = seconds: 1 milliseconds: 469

depth # of nodes evaluation

5 11893 0

Black replies with PrincipalVariation

reply => c7c5

time = seconds: 2 milliseconds: 578

depth # of nodes evaluation

5 31069 0

Here is data table containing the results of various tests made with these move algorithms. Notice that the **Minimax** replies and those of **AlfaBeta** are identical but the number of nodes are higher for **Minimax** and the time taken is lower for **AlfaBeta**. This is clearly seen in the highlighted rows below, although generating identical moves, **Minimax** goes through 19,7281 nodes white **AlfaBeta** goes through 998 nodes. This is reflected in the amount of time taken, which is a difference of approximately 16 seconds. This gives us a clear understanding why **AlfaBeta** is a much favoured technique over **Minimax.** You can also notice that the other two algorithms produce very similar results

in the number of nodes and time.

Move Algorithm Testing: Data

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Algorithm | Depth | White | Reply  (Move) | Number of nodes | Time (s) |
| Minimax | 1 | true | Bb1a3 | 20 | 0.150 |
| false | h7h5 | 20 | 0.000 |
| 2 | true | d2d4 | 400 | 0.790 |
| false | c7c6 | 400 | 0.310 |
| 3 | true | d2d3 | 8902 | 0.687 |
| false | c7c5 | 8902 | 0.610 |
| 4 | true | f2f4 | 197281 | 17.125 |
| false | g7g5 | 197281 | 16.703 |
|  |  |  |  |  |
| AlfaBeta | 1 | true | Bb1a3 | 21 | 0.031 |
| false | h7h5 | 21 | 0.031 |
| 2 | true | d2d4 | 40 | 0.047 |
| false | c7c6 | 40 | 0.031 |
| 3 | true | d2d3 | 467 | 0.078 |
| false | c7c5 | 522 | 0.063 |
| 4 | true | f2f4 | 986 | 0.422 |
| false | g7g5 | 998 | 0.312 |
| 5 | true | c2c4 | 14710 | 1.891 |
| false | g7g6 | 15714 | 2.016 |
| 6 | true | f2f3 | 159048 | 37.843 |
| false | d7d6 | 45304 | 12.610 |
|  |  |  |  |  |
| NegaScout | 1 | true | Bb1c3 | 20 | 0.031 |
| false | b7b5 | 20 | 0.000 |
| 2 | true | f2f3 | 39 | 0.032 |
| false | d7d6 | 39 | 0.015 |
| 3 | true | Bg1f3 | 466 | 0.062 |
| false | Bg8h6 | 466 | 0.062 |
| 4 | true | a2a4 | 929 | 0.297 |
| false | g7g5 | 995 | 0.281 |
| 5 | true | c2c4 | 11858 | 1.500 |
| false | c7c5 | 15360 | 2.109 |
| 6 | true | b2b3 | 142302 | 27.031 |
| false | f7f5 | 29571 | 9.016 |
|  |  |  |  |  |
| PrincipalVariation | 1 | true | g2g4 | 20 | 0.015 |
| false | d7d5 | 20 | 0.000 |
| 2 | true | a2a4 | 39 | 0.015 |
| false | Bg8h6 | 39 | 0.031 |
| 3 | true | Bb1a3 | 466 | 0.062 |
| false | b7b5 | 466 | 0.063 |
| 4 | true | e2e4 | 932 | 0.328 |
| false | Bg8h6 | 995 | 0.297 |
| 5 | true | g2g3 | 11757 | 1.406 |
| false | e7e6 | 15976 | 2.203 |
| 6 | true | h2h4 | 143049 | 26.609 |
| false | Bb8c6 | 29637 | 9.422 |
|  |  |  |  |  |

**Software Development Analysis**

Software development duration:

**Chess Applet**: Oct 22 2004 – Jan 21 2005 approx. 3 months

**Chess Application**: Nov 15 2004 – May 16 2005 approx. 6 months

This Software Development Analysis contains both an analysis for the **Chess Applet** development and an analysis for the **Chess Application** development.

This development analysis is meant to give a broad understanding of the type of files involved in each and the scale of each program. The data for each program type was obtained in the date specified below each program heading in the next pages. The pages are ordered as follows:

1. **Software Development Analysis:** Chess Applet
2. **Software Development Analysis:** Chess Application

**Software Development Analysis:**  Chess Applet

NOTE: the data tables were accumulated with the data of Jan 23 2004

**File Count**

|  |  |
| --- | --- |
| package | # of .java files |
| ICS4M1\_ChessProject | 17 |
| total | 17 |

**File Information**

|  |  |  |  |
| --- | --- | --- | --- |
| **.java file** | **Type** | **Extends** | **Implements** |
| Chess | class | Applet |  |
| ChessBoard | class | JPanel | Constants, MouseListener,  MouseMotionListener, ActionListener, Runnable |
| ChessBottom | class | Applet | Runnable |
| ChessConsole | class |  | Constants |
| ChessGame | class |  | Constants |
| ChessMinimax | class |  |  |
| ChessPlayer | class |  |  |
| ChessSideBar | class | JPanel | ActionListener, Runnable |
| ChessTools | class |  | Constants |
| Constants | interface |  |  |
| Coord | class |  | Constants |
| ImageManipulator | class |  | RGBImageFilter |
| ListModel | class |  | AbstractListModel |
| Move | class |  | Constants |
| Piece | class |  | Constants |
| ThreadApp | class | Thread | Constants |
| timer | class |  |  |

**Status Reports**

|  |  |  |
| --- | --- | --- |
| **Date** | **Time Spent (approximates)** | **Brief** |
| Leading up to Fri. Oct. 22 2004 | ? | - ideas, discussion |
| Fri. Oct. 22 2004 | > 10 hrs | - began coding structure |
| Sat. Oct. 23 2004 | - 4 hrs - | --- |
| Sun. Oct. 24 2004 | - ? - | ChessApplet.java v0.1 complete |
| Sat. Nov. 06 2004 | - 2 hrs - | - implemented player turns, fixed bugs – board now resizes + double click error |
| Mon. Nov. 08 2004 | - 3 hrs - | - generating legal moves + fixed bugs. Big change: the user’s moves that were defined in the ‘mouseReleased’ method are now defined in a new method – used by both the user and will be by the AI which is not yet implemented.  Next step for AI: calculate for each piece whether it’s protected, if so, by how many pieces – and is the piece targeted by enemy pieces, if so – how many |
| Thurs. Nov. 11 2004 | - class time - | - made space on the side for game info |
| Fri. Nov. 12 2004 | - 5 hours - | - pixel graphics (color changes) + display inactive pieces on the side + fixed bugs and began 3d pixelling |
| Sat. Nov. 13 2004 | - 2 hours - | - fixed bug, although new problem not noticed before: while dragging a piece and then with a second mouse click on a different piece the dragging piece disappears  - spacing for the board |
| Tue. Nov. 24 2004 | - 30 min - | - problem – when releasing a piece on the edges of the board, the piece moves but also reappears in it’s original location  - fixed – although a piece can not be dropped on the furthest left edge now |

|  |  |  |
| --- | --- | --- |
| **Date** | **Time Spent (approximates)** | **Brief** |
| Wed. Nov. 25 2004 | - class time - | TODO: 1-Allow moves by not only piece dragging, but also by clicking on the original square and then the square to move to.  2- Once a piece is clicked on, show all the possible moves for the piece  3- Animate darkening of the board with bright center box with changeable settings.  4- Computer – don’t take the AI learning approach, instead, give it a set of rules for prioritizing some locations over others  5. Create clickable tab regions to display different game info  6- Log piece movement history in one of the tabs. |
| Wed. Dec. 01 2004 | - 3 hrs - | - a “dots” game AI |
| Sat. Dec. 04 2004 | - 5 hrs - | - split the code into classes (now ChessConsole holds all the piece information and settings)  - organize the code  - the last move is indicated on the chess board (highlighted) |
| Sun. Dec. 05 2004 | - ? - | - tabs are switchable and working  - the board’s width and height stay proportional when the frame is resized |
| Mon. Dec. 06 2004 | - 5 hrs - | - new pieces + new board |
| Thurs. Dec. 30 2004 | - 2 hrs - | - split the code furthur into classes and objects + new panel layout  - the console is abstract |

|  |  |  |
| --- | --- | --- |
| **Date** | **Time Spent (approximates)** | **Brief** |
| Dec. 21 2004 –  Fri. Jan 21 2005 | - ? - | - reformatted all classes.  - AI complete  - modifying different opponent strengths  - new piece images + sound  - new layout (now 3 panels)  - board and sideboard now split up  - move notation reformated  - sliding pieces now during game play  - optional features such as show legal moves with arrows  - main menu, settings menu, new game menu |
| Sun. Jan. 23 2005 | - ? - | ChessApplet v1.0.0 complete |

**Software Development Analysis:**  Chess Application

NOTE: the data tables were accumulated with the data of May 16 2005

**File Count**

|  |  |
| --- | --- |
| package | # of .java files |
| chess | 1 |
| chess.algorithms | 7 |
| chess.core | 24 |
| chess.gui | 1 |
| chess.gui.board | 6 |
| chess.gui.lookAndFeel | 11 |
| chess.gui.panels | 7 |
| chess.gui.window | 11 |
| chess.media | 3 |
| chess.network | 8 |
| chess.online | 4 |
| chess.properties | 10 |
| total | 93 |

**File Information Ascending With Date Created**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **.java file** | **Package** | **Type** | **Extends** | **Implements** | **Date Created** |
| ChessGame | chess.core | class |  | Constants, Serializable | Nov 15 2004 |
| ImageManipulator | chess.core | class |  | RGBImageFilter | Nov 20 2004 |
| ChessBoard | chess.gui.board | class | JPanel | Constants | Dec 02 2004 |
| Constants | chess.core | interface |  |  | Dec 02 2004 |
| Coord | chess.core | class |  | Constants, Serializable | Dec 10 2004 |
| Move | chess.core | class |  | Constants, Serializable | Dec 11 2004 |
| ChessListModel | chess.core | class | AbstractListModel | Serializable | Dec 16 2004 |
| Piece | chess.core | class |  | Constants, Serializable | Dec 24 2004 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **.java file** | **Package** | **Type** | **Extends** | **Implements** | **Date Created** |
| ChessChat | chess.network |  |  | ActionListener | Feb 10  2005 |
| ChessClient | chess.network |  | NetworkEnd |  | Feb 10  2005 |
| ChessNetPlay | chess.network |  |  |  | Feb 10  2005 |
| ChessServer | chess.network |  | NetworkEnd |  | Feb 10  2005 |
| MsgRedirector | chess.network |  |  | Runnable | Feb 10  2005 |
| NetworkEnd | chess.network |  |  |  | Feb 10  2005 |
| TestClient | chess.network |  | JFrame |  | Feb 10  2005 |
| TestServer | chess.network |  | JFrame | ActionListener | Feb 10  2005 |
| ChessComponent | chess.gui.panels | class | JPanel |  | Feb 22  2005 |
| Chess | chess.gui | class | JFrame | ActionListener | Mar 02 2005 |
| ChessPreferences | chess.properties |  |  |  | Mar 05 2005 |
| PreferencesDialog | chess.gui.window | class | JDialog |  | Mar 16 2005 |
| MoveAlgorithm | chess.algorithms | class |  | Constants | Mar 16 2005 |
| AlfaBeta | chess.algorithms | class | MoveAlgorithm |  | Mar 16 2005 |
| NegaScout | chess.algorithms | class | MoveAlgorithm |  | Mar 16 2005 |
| PrincipalVaration | chess.algorithms | class | MoveAlgorithm |  | Mar 16  2005 |
| ChessPreferences | chess.properties | class |  |  | Mar 16 2005 |
| ChessColors | chess.properties | interface |  |  | Mar 20 2005 |
| ChessBoardMain | chess.gui.board | class | ChessBoard |  | Mar 20 2005 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **.java file** | **Package** | **Type** | **Extends** | **Implements** | **Date Created** |
| ChessBoardSmallView | chess.gui.board | class | ChessBoard |  | Mar 20 2005 |
| ChessBoardVirtual | chess.gui.board | class |  | Constants,  MouseListener | Mar 20 2005 |
| AboutDialog | chess.gui.window | class | JDialog | ActionListener | Mar 20 2005 |
| ChessSetDialog | chess.gui.window | class | JDialog | ActionListener | Mar 20 2005 |
| ExitDialog | chess.gui.window | class | JDialog | ActionListener | Mar 20 2005 |
| ChessDocuments | chess.core | class | AbstractListModel |  | Mar 21 2005 |
| ChessDocumentView | chess.gui.panels | class | JPanel |  | Mar 21 2005 |
| SplashScreen | chess.gui.window | class | JDialog |  | Mar 23  2005 |
| PeerConnectDialog | chess.gui.window | class | JDialog |  | Mar 23  2005 |
| Parameters | chess.properties | class |  |  | Mar 24 2005 |
| GameParameters | chess.properties | class | Parameters | Constants, Serializable | Mar 24 2005 |
| PlayerParameters | chess.properties | class | Parameters | Serializable | Mar 24 2005 |
| State | chess.properties | class |  | Serializable | Mar 24 2005 |
| FlatButton | chess.core | class | JPanel |  | Mar 25 2005 |
| ComponentList | chess.core | class | JList |  | Mar 27 2005 |
| DocumentThumbnail | chess.core | class | JPanel |  | Mar 27 2005 |
| ChessBoardSetup | chess.gui.board | class | ChessBoard |  | Mar 27 2005 |
| PiecesPanel | chess.gui.panels | class | JPanel | Constants, ActionListener | Mar 27 2005 |
| ChessSetupDialog | chess.gui.window | class | JDialog | ActionListener | Mar 27 2005 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **.java file** | **Package** | **Type** | **Extends** | **Implements** | **Date Created** |
| BoardMedia | chess.media | class |  | Constants | Mar 27 2005 |
| MyLookAndFeel | chess.gui.lookAndFeel | class | MetalLookAndFeel |  | Mar 27 2005 |
| BoardParameters | chess.properties | class | Parameters | Constants | Mar 28 2005 |
| ResumeStateDialog | chess.gui.window | class | JPanel | ActionListener | Mar 30 2005 |
| ButtonUI | chess.gui.lookAndFeel | class | BasicButtonUI |  | Mar 30 2005 |
| ChessFileFilter | chess.core | class | FileFilter |  | Mar 31 2005 |
| ScrollBarUI | chess.gui.lookAndFeel | class | MetalScrollBarUI |  | Apr 01  2005 |
| ToggleButtonUI | chess.gui.lookAndFeel | class | MetalToggleButtonUI |  | Apr 01  2005 |
| DefaultTheme | chess.gui.lookAndFeel | class | DefaultMetalTheme |  | Apr 02  2005 |
| FastGradientPaint | chess.gui.lookAndFeel | class |  | PaintContext | Apr 02  2005 |
| GrayTheme | chess.gui.lookAndFeel | class | DefaultMetalTheme |  | Apr 02  2005 |
| MenuBarUI | chess.gui.lookAndFeel | class | BasicMenuBarUI |  | Apr 02  2005 |
| TabbedPaneUI | chess.gui.lookAndFeel | class | BasicTabbedPaneUI |  | Apr 02  2005 |
| ToolBarUI | chess.gui.lookAndFeel | class | BasicToolBarUI |  | Apr 02  2005 |
| MiniMax | chess.algorithms | class | MoveAlgorithm |  | Apr 02  2005 |
| RandomGen | chess.algorithms | class | MoveAlgorithm |  | Apr 02  2005 |
| Tester | chess.algorithms | class |  |  | Apr 02  2005 |
| SpringUtilities | chess.gui.window | class |  |  | Apr 02  2005 |
| JTableCellRenderer | chess.core | class | TableCellRenderer |  | Apr 03  2005 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **.java file** | **Package** | **Type** | **Extends** | **Implements** | **Date Created** |
| CustomCellRenderer | chess.core | class | JLabel | TableCellRenderer | Apr 03  2005 |
| ChessTableModel | chess.core | class | AbstractTableModel |  | Apr 03  2005 |
| HelloNet | chess.online | class | JFrame | ActionListener, Runnable | Apr 09  2005 |
| FICSProtocol | chess.online | class | JFrame | Runnable | Apr 14  2005 |
| FICSProtocolHandler | chess.online | class | ICSProtocolHandler |  | Apr 14  2005 |
| ICSProtocolHandler | chess.online | class |  | Runnable | Apr 14  2005 |
| ClockPanel | chess.gui.panels | class | JPanel |  | Apr 19  2005 |
| VirtualClock | chess.core | class |  | Constants, Serializable | Apr 19  2005 |
| VisualClock | chess.core | class | JApplet |  | Apr 19  2005 |
| PlayerLabel | chess.core | class | JPanel |  | Apr 22  2005 |
| PGN | chess.core | class |  |  | Apr 27  2005 |
| ChessFile | chess.core | class |  |  | May 03 2005 |
| Main | chess | class |  |  | May 04  2005 |
| ErrorMessage | chess.gui.window | class | JDialog |  | May 04  2005 |
| InfoLabel | chess.core | class | JLabel |  | May 05  2005 |
| ImageUtilities | chess.media | class |  |  | May 08  2005 |
| Surface | chess.media | class |  |  | May 08  2005 |
| Board | chess.core | class |  | Constants | May 10 2005 |
| ChessBoardSample | chess.gui.board | class | JPanel | Constants | May 10 2005 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Computer | chess.core | class | Thread |  | May 13 2005 |
| PieceCounts | chess.core | class |  | Constants | May 13 2005 |
| GameDetailsDialog | chess.gui.window | class | JDialog |  | May 15 2005 |
| ChessSmallMode | chess.gui.panels | class |  |  | May 15 2005 |
| Utilities | chess.properties | class |  |  | May 15 2005 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **.java file** | **Package** | **Type** | **Extends** | **Implements** | **Date Created** |

**Credits**

The Chess project was created by Arvydas Bancewicz, including the contents described below.

All of the images, other then, the piece images were created by the author. The original images can be found in this software package in the photoshop document (.psd) file format. These images are in their raw form and can be opened via Adobe Photoshop.

All end-user and technical documentation was created by the author, with the author’s wording.