**SATRANÇ TAHTASI ÜZERİNDE**

**AT İLE BÜTÜN ALANLARIN**

**DOLAŞILMASI**

İçindekiler

[A. Problemin Tanımı 3](#_Toc353240063)

[B. Problemin Algoritmik Tanımı 3](#_Toc353240064)

[1. Genel Çözümün Algoritmik Tanımı: 3](#_Toc353240065)

[2. Warnsdorff Kuralının Algoritmik Tanımı: 3](#_Toc353240066)

[C. UML Diyagram 5](#_Toc353240067)

[D. Akış Diyagramı 6](#_Toc353240068)

[E. Ekran Görüntüleri 7](#_Toc353240069)

[1. Açılış Görüntüsü 7](#_Toc353240070)

[2. Fare ile Tahta Üzerinde Herhangi Bir Yer Tespitinden Sonra 7](#_Toc353240071)

[3. Çözümün Bulunması 8](#_Toc353240072)

[4. Sağ-Sol Oklarıyla Çözümünde Herhangi Bir Adımın İncelenmesi 8](#_Toc353240073)

[F. Sınıflar 10](#_Toc353240074)

[5. TileState.java 10](#_Toc353240075)

[6. Tile.java 10](#_Toc353240076)

[7. Knight.java 10](#_Toc353240077)

[8. Board.java 11](#_Toc353240078)

[9. ChessDiscoverer.java 13](#_Toc353240079)

[10. Algorithm.java 13](#_Toc353240080)

[11. UI.java (Ayrıca dahili sınıflar: JComboboxItem, DrawingHelper) 14](#_Toc353240081)

[12. Start.java 20](#_Toc353240082)

[G. Kaynakça 20](#_Toc353240083)

# Problemin Tanımı

“Bir At Turu” olarak bilinen problem bir satranç tahtası üzerinde her kereye bir kere uğramak koşuluyla atın bütün kareleri atın hareket koşullarına uygun olarak dolaşması olarak nitelendirilebilir.

Problem 9. yüzyılda ortaya atılmış olup, atın bu tam turu çeşitli yöntemlerle çözülmüş ve matematiksel bir problem olduğu kanıtlanmıştır.

Çizge kuramındaki yönsüz Hamilton probleminin temel konusu niteliğindedir. Problemin çözümüne en yatkın teorem olarak kabul edilen teorem ise “Hamilton Yolu” teoremidir.

Yapılan bir çok araştırma ve geliştirmelerden sonra 1990’larda Warnsdorff adında bir araştırmacı tarafından “Warnsdorff Kuralı” olarak tarihe geçmiş bir bilgili arama (heuristik ve ~O(n) karmaşıklığında) ortaya atılmış olup günümüzde de popülerliğini korumaktadır.

# Problemin Algoritmik Tanımı

## Genel Çözümün Algoritmik Tanımı:

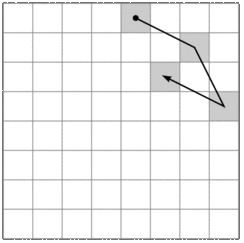
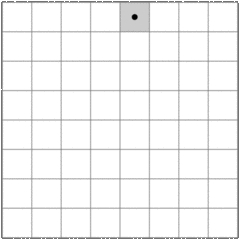
1. At tahta üzerinde istenilen herhangi bir kareye konur.
2. Atın hareket kurallarının el verdiği ölçüde hareket edebildiği ve daha önce hiç uğramadığı karelerden birine hareket ettirilir ve bulunulan kare “şu anki” olarak belirlenir.
3. Atın önceden bulunduğu konum “uğradı” olarak işaretlenir.
4. Eğer atın gidebileceği herhangi bir kare yok ise daha önceden “uğradı” olarak işaretlenen kareye geri dönülür ve durumu “şu anki” olarak değiştirilir. Geri dönülen yol “cevapsız” olduğu için bu yola bir daha uğramamak kaydıyla terk edilir.
5. Oyun tahtasındaki bütün alanlar ziyaret edilmişse 6. adıma; aksi takdirde 2. adıma tekrar gidilir.
6. Tahta üzerindeki bütün alanlar ziyaret edilmiş olacağı için atın hareketi sona erer.

## Warnsdorff Kuralının Algoritmik Tanımı:

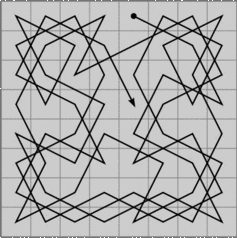
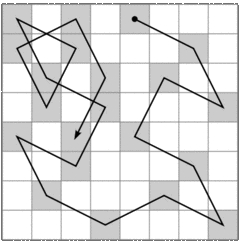
1. At tahta üzerinde istenilen herhangi bir kareye konur.
2. Atın hareket kurallarının el verdiği ölçüde hareket edebildiği ve daha önce hiç uğramadığı karelerden **gidildiği zaman en az alternatifi veren kareye hareket** ettirilir ve bulunulan kare “şu anki” olarak belirlenir.
3. Atın önceden bulunduğu konum “uğradı” olarak işaretlenir.
4. Eğer atın gidebileceği herhangi bir kare yok ise daha önceden “uğradı” olarak işaretlenen kareye geri dönülür ve durumu “şu anki” olarak değiştirilir. Geri dönülen yol “cevapsız” olduğu için bu yola bir daha uğramamak kaydıyla terk edilir.
5. Oyun tahtasındaki bütün alanlar ziyaret edilmişse 6. adıma; aksi takdirde 2. adıma tekrar gidilir.
6. Tahta üzerindeki bütün alanlar ziyaret edilmiş olacağı için atın hareketi sona erer.

**(Warnsdorff algoritmasının normal algoritmadan farkı yukarıda koyu ve altı çizgili olarak belirtilmiştir.)**

Aşağıdaki şekilde örnek bir Warnsdorff algoritmasının çözümü gösterilmiştir:

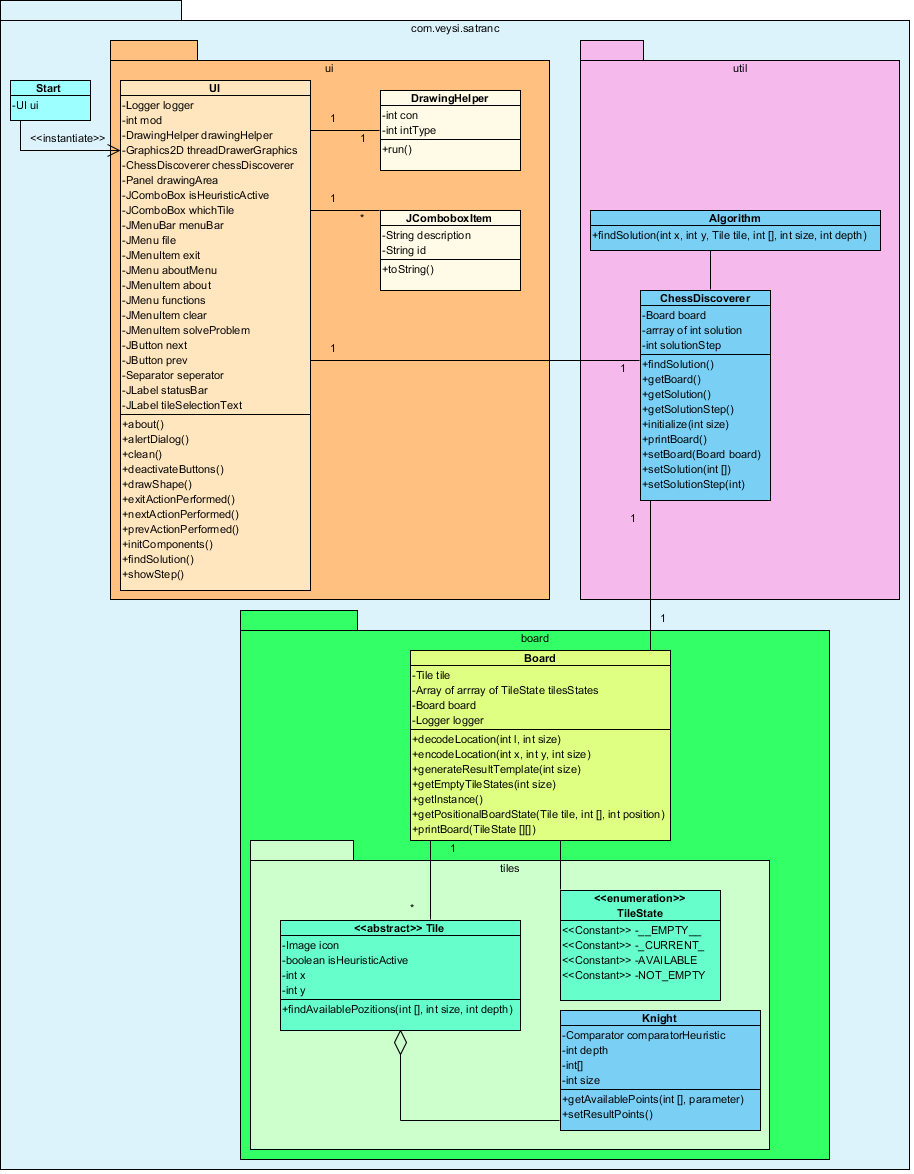


a) b)

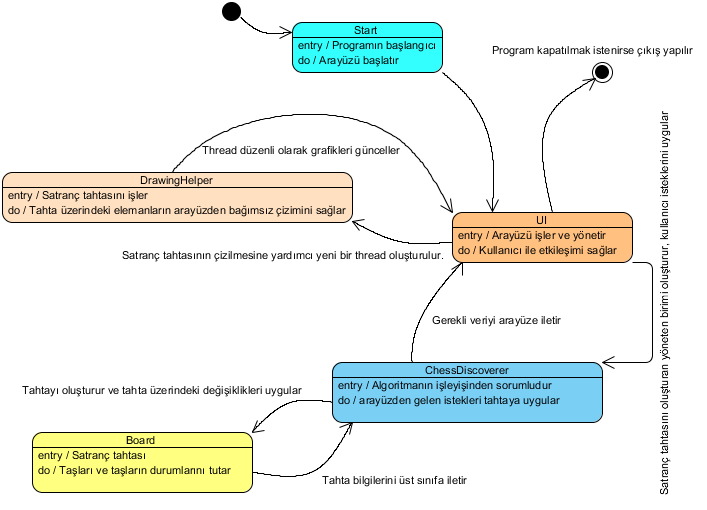


c) d)

# UML Diyagram

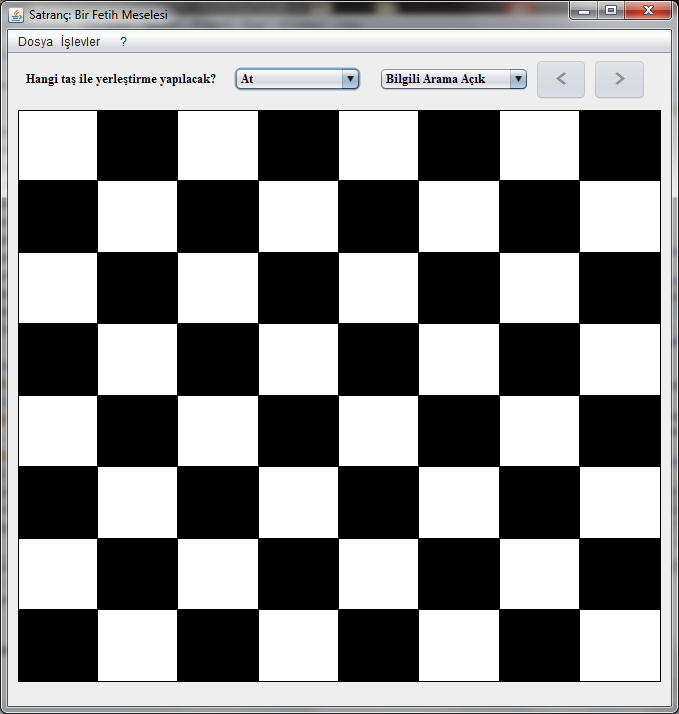


# Akış Diyagramı

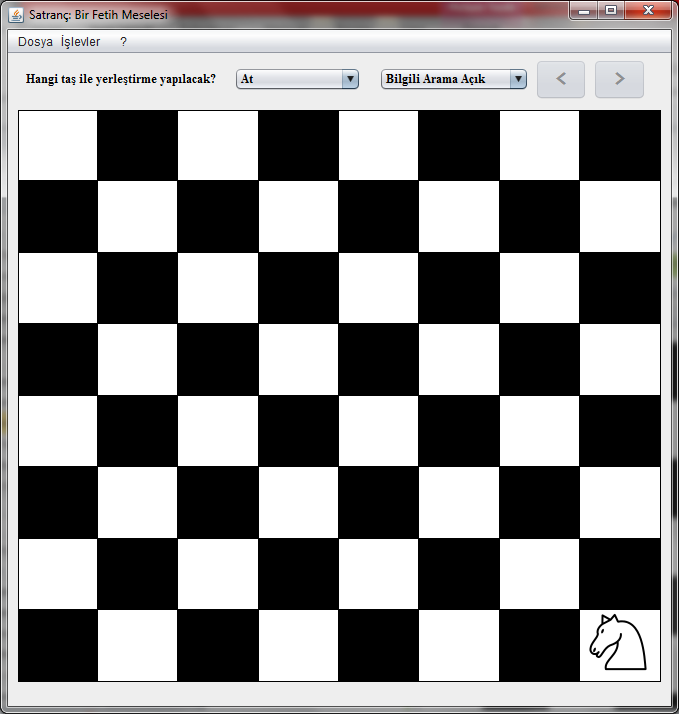


# Ekran Görüntüleri

## Açılış Görüntüsü



## Fare ile Tahta Üzerinde Herhangi Bir Yer Tespitinden Sonra

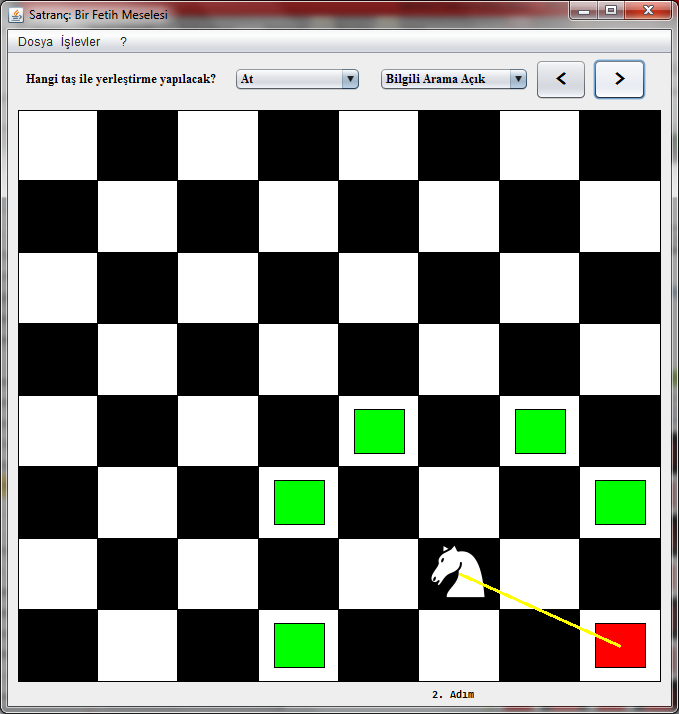


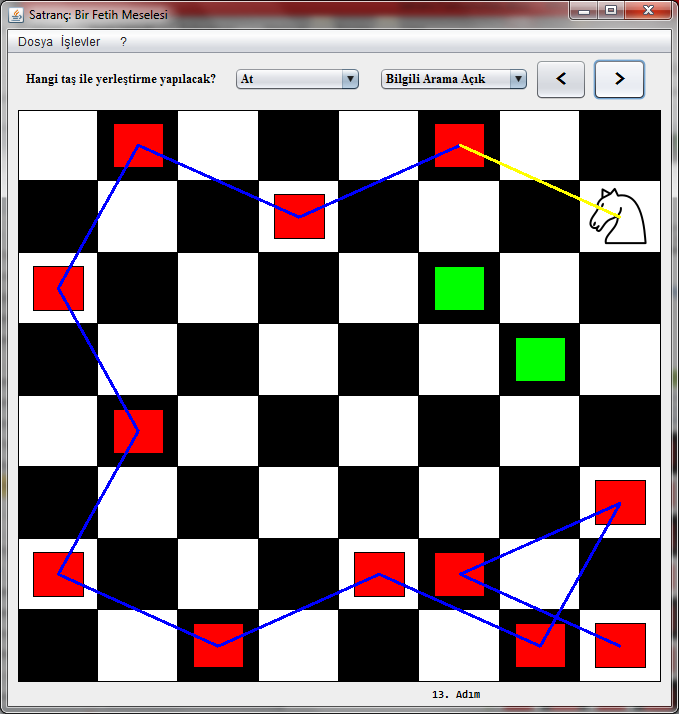
## Çözümün Bulunması

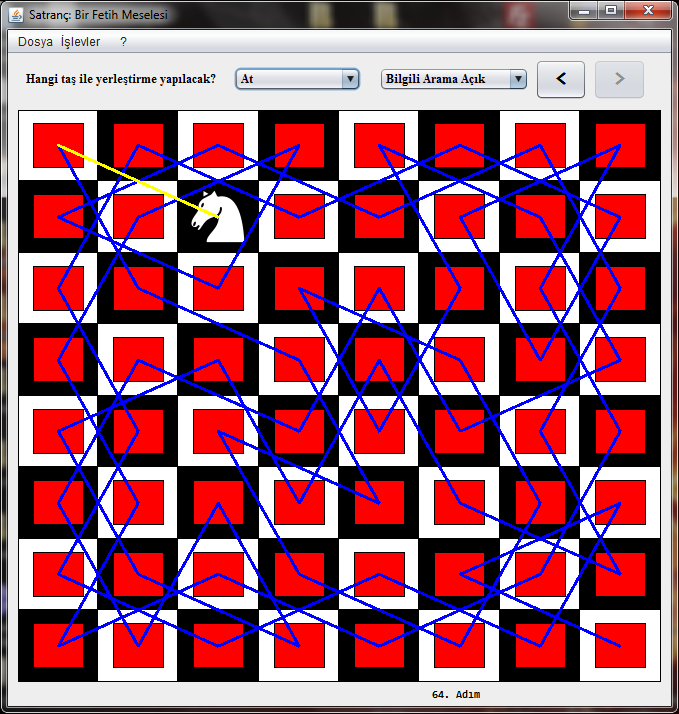
* “İşlevler->Çözüm Bul” menüsü yardımıyla veya
* Ctrl+F ile çözümün bulunması istenebilir.



## Sağ-Sol Oklarıyla Çözümünde Herhangi Bir Adımın İncelenmesi







# Sınıflar

## TileState.java

|  |
| --- |
| **package** com.veysi.satranc.board.tiles;  /\*\*  \* **@author** veysiertekin  \*/  **public** **enum** TileState {  \_\_EMPTY\_\_, \_CURRENT\_, AVAILABLE, NOT\_EMPTY;  } |

## Tile.java

|  |
| --- |
| **package** com.veysi.satranc.board.tiles;  **import** java.awt.Image;  **import** java.util.List;  /\*\*  \*  \* **@author** veysiertekin  \*/  **public** **abstract** **class** Tile {  **private** Image icon;  **public** **boolean** isHeuristicActive = **false**;  **private** **int** x;  **private** **int** y;  **public** **abstract** List<**int**[]> findAvailablePozitions(**int**[] result, **int** size, **int** depth);  **public** Image getIcon() {  **return** icon;  }  **public** **int** getX() {  **return** x;  }  **public** **int** getY() {  **return** y;  }  **public** **void** setIcon(Image icon) {  **this**.icon = icon;  }  **public** Tile setPosition(**int** x, **int** y) {  setX(x);  setY(y);  **return** **this**;  }  **public** **void** setX(**int** x) {  **this**.x = x;  }  **public** **void** setY(**int** y) {  **this**.y = y;  }  } |

## Knight.java

|  |
| --- |
| **package** com.veysi.satranc.board.tiles;  **import** java.io.IOException;  **import** java.util.ArrayList;  **import** java.util.Collections;  **import** java.util.Comparator;  **import** java.util.List;  **import** javax.imageio.ImageIO;  **import** com.veysi.satranc.Start;  **import** com.veysi.satranc.board.Board;  /\*\*  \*  \* **@author** veysiertekin  \*/  **public** **class** Knight **extends** Tile {  **private** Comparator<**int**[]> comparatorHeuristic = **new** Comparator<**int**[]>() {  /\*\*  \* Warnsdorff algorithm  \*  \* **@param** o1  \* **@param** o2  \* **@return**  \*/  @Override  **public** **int** compare(**int**[] o1, **int**[] o2) {  **int** x = getX();  **int** y = getY();  setX(o1[0]);  setY(o1[1]);  result[depth + 1] = Board.encodeLocation(o1[0], o1[1], size);  **int** a = getAvailablePoints(result, size).size();  setX(o2[0]);  setY(o2[1]);  result[depth + 1] = Board.encodeLocation(o2[0], o2[1], size);  **int** b = getAvailablePoints(result, size).size();  setX(x);  setY(y);  result[depth + 1] = -1;  **return** a == b ? 0 : (a < b ? -1 : 1);  }  };  **int** depth;  **int**[] result;  **int** size;  **public** Knight() {  **try** {  setIcon(ImageIO.read(Start.**class**.getResource("img/png/at.png")));  }  **catch** (IOException e) {  e.printStackTrace();  }  }  @Override  **public** List<**int**[]> findAvailablePozitions(**int**[] result, **int** size, **int** depth) {  **this**.result = result;  **this**.depth = depth;  **this**.size = size;  List<**int**[]> resultPoints = getAvailablePoints(result, size);  **if** (isHeuristicActive)  Collections.sort(resultPoints, comparatorHeuristic);  **return** resultPoints;  }  **public** List<**int**[]> getAvailablePoints(**int**[] result, **int** size) {  List<**int**[]> resultPoints = **new** ArrayList<**int**[]>();  **int** x = getX();  **int** y = getY();  **int** tmpX;  **int** tmpY;  **if** ((tmpX = x - 2) >= 0 && (tmpY = y - 1) >= 0) {  setResultPoints(resultPoints, result, size, tmpX, tmpY);  }  **if** ((tmpX = x - 1) >= 0 && (tmpY = y - 2) >= 0) {  setResultPoints(resultPoints, result, size, tmpX, tmpY);  }  **if** ((tmpX = x + 1) >= 0 && (tmpY = y - 2) >= 0) {  setResultPoints(resultPoints, result, size, tmpX, tmpY);  }  **if** ((tmpX = x + 2) >= 0 && (tmpY = y - 1) >= 0) {  setResultPoints(resultPoints, result, size, tmpX, tmpY);  }  **if** ((tmpX = x + 2) >= 0 && (tmpY = y + 1) >= 0) {  setResultPoints(resultPoints, result, size, tmpX, tmpY);  }  **if** ((tmpX = x + 1) >= 0 && (tmpY = y + 2) >= 0) {  setResultPoints(resultPoints, result, size, tmpX, tmpY);  }  **if** ((tmpX = x - 1) >= 0 && (tmpY = y + 2) >= 0) {  setResultPoints(resultPoints, result, size, tmpX, tmpY);  }  **if** ((tmpX = x - 2) >= 0 && (tmpY = y + 1) >= 0) {  setResultPoints(resultPoints, result, size, tmpX, tmpY);  }  **return** resultPoints;  }  **public** **void** setResultPoints(List<**int**[]> resultPoints, **int**[] result, **int** size, **int** tmpX, **final** **int** tmpY) {  **for** (**int** index = 0; index < result.length; index++) {  **if** (Board.encodeLocation(tmpX, tmpY, size) == result[index])  **return**;  }  **if** ((tmpX < size && tmpY < size)) {  resultPoints.add(**new** **int**[] { tmpX, tmpY });  }  }  } |

## Board.java

|  |
| --- |
| **package** com.veysi.satranc.board;  **import** com.veysi.satranc.board.tiles.Tile;  **import** com.veysi.satranc.board.tiles.TileState;  **import** java.util.Arrays;  **import** java.util.List;  **import** java.util.logging.Logger;  /\*\*  \* **@author** veysiertekin  \*/  **public** **class** Board {  **private** Tile tile;  **private** TileState[][] tilesStates;  **private** **static** Board board;  **private** **static** Logger logger = Logger.getLogger(Board.**class**.getName());  **public** **static** **int**[] decodeLocation(**int** l, **int** size) {  **if** (l < 0)  **return** **null**;  **int** x = l % size, y = l / size;  **return** **new** **int**[] { x, y };  }  **public** **static** **int** encodeLocation(**int** x, **int** y, **int** size) {  **return** x + (y \* size);  }  **public** **static** **int**[] generateResultTemplate(**int** size) {  **int**[] result = **new** **int**[size \* size];  Arrays.fill(result, -1);  **return** result;  }  **public** **static** TileState[][] getEmptyTileStates(**int** size) {  TileState[][] tileStates = **new** TileState[size][size];  **for** (TileState[] tileState : tileStates) {  Arrays.fill(tileState, TileState.\_\_EMPTY\_\_);  }  **return** tileStates;  }  **public** **static** Board getInstance() {  **if** (board == **null**) {  board = **new** Board(8);  }  **return** board;  }  **public** **static** Board getInstance(**int** size) {  board = **new** Board(size);  **return** board;  }  **public** **static** TileState[][] getPositionalBoardState(Tile tile, **int**[] result, **int** position) {  **int** size = (**int**) Math.sqrt(result.length);  **int**[] resultTMP = Board.generateResultTemplate(size);  TileState[][] states = Board.getEmptyTileStates(size);  **for** (**int** j = 0; j <= position; j++) {  // (++++++++ Clean up board  **for** (TileState[] statesTMP : states) {  **for** (**int** i = 0; i < statesTMP.length; i++) {  **if** (statesTMP[i] == TileState.\_CURRENT\_) {  statesTMP[i] = TileState.NOT\_EMPTY;  }  **else** **if** (statesTMP[i] != TileState.NOT\_EMPTY) {  statesTMP[i] = TileState.\_\_EMPTY\_\_;  }  }  }  // Clean up board ---------)  **if** (result[j] == -1)  **return** **null**;  resultTMP[j] = result[j];  **int**[] point = decodeLocation(result[j], size);  tile.setPosition(point[0], point[1]);  states[point[1]][point[0]] = TileState.\_CURRENT\_;  List<**int**[]> availablePositions = tile.findAvailablePozitions(resultTMP, size, j);  **for** (**int**[] pointTMP : availablePositions) {  states[pointTMP[1]][pointTMP[0]] = TileState.AVAILABLE;  }  }  printBoard(states);  **return** states;  }  **public** **static** **void** printBoard(TileState[][] states) {  logger.info(Arrays.deepToString(states).replace("[[", "\n").replace("]]", "\n").replaceAll("\\], \\[", "\n"));  }  **private** Board(**int** size) {  setTilesStates(getEmptyTileStates(size));  }  **public** Tile getTile() {  **return** tile;  }  **public** TileState[][] getTilesStates() {  **return** tilesStates;  }  **public** **void** setTile(Tile tile) {  **this**.tile = tile;  }  **public** **void** setTilesStates(TileState[][] tilesStates) {  **this**.tilesStates = tilesStates;  }  } |

## ChessDiscoverer.java

|  |
| --- |
| **package** com.veysi.satranc.util;  **import** com.veysi.satranc.board.Board;  /\*\*  \* **@author** veysiertekin  \*/  **public** **class** ChessDiscoverer {  **private** Board board;  **private** **int**[] solution;  **private** **int** solutionStep;  **public** ChessDiscoverer() {  initialize(8);  }  **public** ChessDiscoverer(**int** size) {  initialize(size);  }  **public** **boolean** findSolution() {  **int**[] resultTMP = Board.generateResultTemplate(board.getTilesStates().length);  solution = Algorithm.findSolution(board.getTile().getX(), board.getTile().getY(), board.getTile(), resultTMP, board.getTilesStates().length, 0);  **return** solution[solution.length - 1] != -1;  }  **public** Board getBoard() {  **return** board;  }  **public** **int**[] getSolution() {  **return** solution;  }  **public** **int** getSolutionStep() {  **return** solutionStep;  }  **private** **void** initialize(**int** size) {  board = Board.getInstance(size);  setSolution(Board.generateResultTemplate(board.getTilesStates().length));  }  **public** **void** printBoard() {  Board.printBoard(board.getTilesStates());  }  **public** **void** setBoard(Board board) {  **this**.board = board;  }  **public** **void** setSolution(**int**[] solution) {  **this**.solution = solution;  }  **public** **void** setSolutionStep(**int** solutionStep) {  **this**.solutionStep = solutionStep;  }  } |

## Algorithm.java

|  |
| --- |
| **package** com.veysi.satranc.util;  **import** java.util.List;  **import** com.veysi.satranc.board.Board;  **import** com.veysi.satranc.board.tiles.Tile;  /\*\*  \*  \* **@author** veysiertekin  \*/  **public** **class** Algorithm {  // private static Logger logger = Logger.getLogger(Board.class.getName());  **public** **static** **int**[] findSolution(**int** x, **int** y, Tile tile, **int**[] result, **int** size, **int** depth) {  tile.setPosition(x, y);  result[depth] = Board.encodeLocation(x, y, (**int**) Math.sqrt(result.length));  List<**int**[]> list = tile.findAvailablePozitions(result, size, depth);  **if** (result[result.length - 1] != -1) {  **return** result;  }  **int** i, j;  **for** (**int**[] point : list) {  i = point[0];  j = point[1];  **int**[] resultTMP = **new** **int**[result.length];  System.arraycopy(result, 0, resultTMP, 0, result.length);  resultTMP = findSolution(i, j, tile, resultTMP, size, depth + 1);  **if** (resultTMP[resultTMP.length - 1] != -1) {  **return** resultTMP;  }  }  **return** result;  }  } |

## UI.java (Ayrıca dahili sınıflar: JComboboxItem, DrawingHelper)

|  |
| --- |
| **package** com.veysi.satranc.ui;  **import** java.awt.Color;  **import** java.awt.Graphics;  **import** java.awt.Graphics2D;  **import** java.awt.Image;  **import** java.awt.event.ActionEvent;  **import** java.awt.event.ActionListener;  **import** java.awt.image.BufferedImage;  **import** java.util.Arrays;  **import** java.util.Vector;  **import** java.util.logging.Logger;  **import** javax.swing.JButton;  **import** javax.swing.JDialog;  **import** javax.swing.JFrame;  **import** javax.swing.JLabel;  **import** javax.swing.JPanel;  **import** javax.swing.UIManager;  **import** javax.swing.WindowConstants;  **import** com.veysi.satranc.board.Board;  **import** com.veysi.satranc.board.tiles.Knight;  **import** com.veysi.satranc.board.tiles.Tile;  **import** com.veysi.satranc.board.tiles.TileState;  **import** com.veysi.satranc.util.ChessDiscoverer;  /\*\*  \*  \* **@author** veysiertekin  \*/  **public** **class** UI **extends** JFrame {  **private** **static** Logger logger = Logger.getLogger(UI.**class**.getName());  **private** **int** mod;  **private** DrawingHelper drawingHelper;  **private** Graphics2D threadDrawerGraphics;  **public** ChessDiscoverer chessDiscoverer;  **private** java.awt.Panel drawingArea;  **private** javax.swing.JComboBox isHeuristicActive;  **private** javax.swing.JComboBox whichTile;  **private** javax.swing.JMenuBar menuBar;  **private** javax.swing.JMenu file;  **private** javax.swing.JMenuItem exit;  **private** javax.swing.JMenu aboutMenu;  **private** javax.swing.JMenuItem about;  **private** javax.swing.JMenu functions;  **private** javax.swing.JMenuItem clear;  **private** javax.swing.JMenuItem solveProblem;  **private** javax.swing.JButton next;  **private** javax.swing.JButton prev;  **private** javax.swing.JPopupMenu.Separator seperator;  **private** javax.swing.JLabel statusBar;  **private** javax.swing.JLabel tileSelectionText;  **private** **static** **final** **long** serialVersionUID = 2331602266948345139L;  **public** UI(**int** boyut) {  **try** {  UIManager.setLookAndFeel("com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel");  }  **catch** (Exception e) {}  initComponents();  threadDrawerGraphics = (Graphics2D) drawingArea.getGraphics();  deactiveButtons(0);  chessDiscoverer = **new** ChessDiscoverer(boyut);  drawingHelper = **new** DrawingHelper();  drawingHelper.setPriority(10);  drawingHelper.start();  }  **private** **void** about(java.awt.event.ActionEvent evt) {  alertDialog("Veysi Ertekin - Mart / 2013");  }  **private** **void** alertDialog(String string) {  setEnabled(**false**);  **final** JDialog a\_ = **new** JDialog();  JPanel a = **new** JPanel();  JLabel aa = **new** JLabel(string);  JButton aaa = **new** JButton("Tamam");  aaa.setBounds(50, 50, 100, 50);  a.add(aa);  a.add(aaa);  aaa.addMouseListener(**new** java.awt.event.MouseAdapter() {  @Override  **public** **void** mouseClicked(java.awt.event.MouseEvent evt) {  setEnabled(**true**);  a\_.dispose();  }  });  a\_.add(a);  a\_.setSize(350, 100);  a\_.setVisible(**true**);  a\_.setLocation(**this**.getLocation().x - 100 + **this**.getWidth() / 2, **this**.getLocation().y + 100);  a\_.setResizable(**false**);  a\_.setDefaultCloseOperation(WindowConstants.DO\_NOTHING\_ON\_CLOSE);  }  **public** **void** clean() {  mod = 0;  deactiveButtons(0);  threadDrawerGraphics.clearRect(0, 0, drawingArea.getSize().width, drawingArea.getSize().height);  chessDiscoverer = **new** ChessDiscoverer(chessDiscoverer.getBoard().getTilesStates().length);  statusBar.setText("");  }  **private** **void** clean(java.awt.event.ActionEvent evt) {  clean();  }  **public** **void** deactiveButtons(**int** i) {  **switch** (i) {  **case** 0:  prev.setEnabled(**false**);  next.setEnabled(**false**);  **break**;  **case** 1:  prev.setEnabled(**true**);  next.setEnabled(**true**);  **break**;  **case** 2:  prev.setEnabled(**false**);  next.setEnabled(**true**);  **break**;  **case** 3:  prev.setEnabled(**true**);  next.setEnabled(**false**);  **break**;  }  }  @Override  **public** **void** dispose() {  **try** {  drawingHelper.interrupt();  drawingHelper.join();  **super**.dispose();  }  **catch** (InterruptedException ex) {}  }  @SuppressWarnings({ "unchecked", "rawtypes" })  **private** **void** drawShape(java.awt.event.MouseEvent evt) {  **int** x = evt.getX();  **int** y = evt.getY();  **double** YGenislik = drawingArea.getSize().height / (**double**) chessDiscoverer.getBoard().getTilesStates().length;  **double** XGenislik = drawingArea.getSize().width / (**double**) chessDiscoverer.getBoard().getTilesStates().length;  x = (**int**) ((**int**) x / XGenislik);  y = (**int**) ((**int**) y / YGenislik);  **if** (chessDiscoverer.getBoard().getTilesStates()[y][x] == TileState.\_\_EMPTY\_\_ || mod == 10) {  **switch** (mod) {  **case** -1:  **break**;  **default**:  **try** {  chessDiscoverer.getBoard().setTilesStates(Board.getEmptyTileStates(chessDiscoverer.getBoard().getTilesStates().length));  chessDiscoverer.getBoard().getTilesStates()[y][x] = TileState.\_CURRENT\_;  Tile tile = (Tile) ((Class) Class.forName(((JComboboxItem) whichTile.getSelectedItem()).getId())).getConstructor().newInstance();  tile.setPosition(x, y);  tile.isHeuristicActive = Boolean.parseBoolean(((JComboboxItem) isHeuristicActive.getSelectedItem()).getId());  chessDiscoverer.getBoard().setTile(tile);  }  **catch** (Exception e) {  e.printStackTrace();  }  }  }  }  **private** **void** exitActionPerformed(java.awt.event.ActionEvent evt) {  **this**.dispose();  }  **private** **void** findSolution(java.awt.event.ActionEvent evt) {  **if** (mod == -1 || chessDiscoverer.getBoard().getTile() == **null**) { // çözüm bulunduysa  **return**;  }  **try** {  mod = -1;  showStep();  **final** JDialog dialog = **new** JDialog();  **final** Thread th = **new** Thread() {  @Override  **public** **void** run() {  **long** start, end;  start = System.currentTimeMillis();  **if** (chessDiscoverer.findSolution()) {  end = System.currentTimeMillis();  System.gc();  logger.info("Solution: " + Arrays.toString(chessDiscoverer.getSolution()));  logger.info("Time Estimation: " + (end - start) + " millisecond(s)");  alertDialog("Çözüm " + (end - start) + " milisaniyede bulundu.");  showStep();  }  **else** {  deactiveButtons(0);  logger.warning("There is no solution!");  alertDialog("Çözüm Bulunamadı!");  }  dialog.dispose();  }  };  JPanel a = **new** JPanel();  JButton aaa = **new** JButton("Durdur");  aaa.setBounds(50, 50, 200, 50);  a.add(aaa);  aaa.addMouseListener(**new** java.awt.event.MouseAdapter() {  @SuppressWarnings("deprecation")  @Override  **public** **void** mouseClicked(java.awt.event.MouseEvent evt) {  th.stop();  alertDialog("Arama işlemi başarıyla durduruldu!");  clean();  dialog.dispose();  }  });  dialog.add(a);  dialog.setSize(350, 70);  dialog.setVisible(**true**);  dialog.setLocation(**this**.getLocation().x - 100 + **this**.getWidth() / 2, **this**.getLocation().y + 100);  dialog.setResizable(**false**);  dialog.setDefaultCloseOperation(WindowConstants.DO\_NOTHING\_ON\_CLOSE);  setEnabled(**false**);  th.start();  }  **catch** (Exception e) {}  }  @SuppressWarnings({ "unchecked", "rawtypes" })  **private** **void** initComponents() {  drawingArea = **new** java.awt.Panel() {  **private** **static** **final** **long** serialVersionUID = -656469038083634829L;  @Override  **public** **void** paint(Graphics g) {  **if** (drawingHelper != **null**) {  drawingHelper.intType = 1;  drawingHelper.interrupt();  drawingHelper.intType = 0;  }  }  };  prev = **new** javax.swing.JButton();  next = **new** javax.swing.JButton();  statusBar = **new** javax.swing.JLabel();  tileSelectionText = **new** javax.swing.JLabel();  whichTile = **new** javax.swing.JComboBox(**new** Vector() {  **private** **static** **final** **long** serialVersionUID = 9004038530765628146L;  {  addElement(**new** JComboboxItem(Knight.**class**.getName(), "At"));  }  });  isHeuristicActive = **new** javax.swing.JComboBox(**new** Vector() {  **private** **static** **final** **long** serialVersionUID = 9004038530765628146L;  {  addElement(**new** JComboboxItem("true", "Bilgili Arama Açık"));  addElement(**new** JComboboxItem("false", "Bilgili Arama Kapalı"));  }  });  menuBar = **new** javax.swing.JMenuBar();  file = **new** javax.swing.JMenu();  exit = **new** javax.swing.JMenuItem();  functions = **new** javax.swing.JMenu();  clear = **new** javax.swing.JMenuItem();  seperator = **new** javax.swing.JPopupMenu.Separator();  solveProblem = **new** javax.swing.JMenuItem();  aboutMenu = **new** javax.swing.JMenu();  about = **new** javax.swing.JMenuItem();  setDefaultCloseOperation(javax.swing.WindowConstants.EXIT\_ON\_CLOSE);  setTitle("Satranç: Bir Fetih Meselesi");  setLocationByPlatform(**true**);  setModalExclusionType(java.awt.Dialog.ModalExclusionType.APPLICATION\_EXCLUDE);  drawingArea.setBackground(**new** java.awt.Color(255, 254, 254));  drawingArea.setCursor(**new** java.awt.Cursor(java.awt.Cursor.DEFAULT\_CURSOR));  drawingArea.addMouseListener(**new** java.awt.event.MouseAdapter() {  @Override  **public** **void** mousePressed(java.awt.event.MouseEvent evt) {  drawShape(evt);  }  });  drawingArea.addMouseMotionListener(**new** java.awt.event.MouseMotionAdapter() {  @Override  **public** **void** mouseDragged(java.awt.event.MouseEvent evt) {  drawShape(evt);  }  });  javax.swing.GroupLayout CizimSahasiLayout = **new** javax.swing.GroupLayout(drawingArea);  drawingArea.setLayout(CizimSahasiLayout); CizimSahasiLayout.setHorizontalGroup(CizimSahasiLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING).addGap(0, 0, Short.MAX\_VALUE)); CizimSahasiLayout.setVerticalGroup(CizimSahasiLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING).addGap(0, 572, Short.MAX\_VALUE));  prev.setFont(**new** java.awt.Font("Consolas", 1, 24)); // NOI18N  prev.setText("<");  prev.setToolTipText("Önceki");  prev.addActionListener(**new** java.awt.event.ActionListener() {  @Override  **public** **void** actionPerformed(java.awt.event.ActionEvent evt) {  prevActionPerformed(evt);  }  });  next.setFont(**new** java.awt.Font("Consolas", 1, 24)); // NOI18N  next.setText(">");  next.setToolTipText("Sonraki");  next.addActionListener(**new** java.awt.event.ActionListener() {  @Override  **public** **void** actionPerformed(java.awt.event.ActionEvent evt) {  nextActionPerformed(evt);  }  });  statusBar.setBackground(**new** java.awt.Color(222, 221, 221));  statusBar.setFont(**new** java.awt.Font("Consolas", 1, 11)); // NOI18N  statusBar.setToolTipText("Durum çubuğu");  tileSelectionText.setFont(**new** java.awt.Font("Times New Roman", 1, 12)); // NOI18N  tileSelectionText.setText("Hangi taş ile yerleştirme yapılacak?");  whichTile.setFont(**new** java.awt.Font("Times New Roman", 1, 12)); // NOI18N  whichTile.addActionListener(**new** ActionListener() {  @Override  **public** **void** actionPerformed(ActionEvent arg0) {  **try** {  chessDiscoverer.getBoard().setTile((Tile) ((Class) Class.forName(((JComboboxItem) whichTile.getSelectedItem()).getId())).getConstructor().newInstance());  }  **catch** (Exception e) {  e.printStackTrace();  }  }  });  isHeuristicActive.setFont(**new** java.awt.Font("Times New Roman", 1, 12)); // NOI18N  isHeuristicActive.addActionListener(**new** ActionListener() {  @Override  **public** **void** actionPerformed(ActionEvent arg0) {  Tile tile = chessDiscoverer.getBoard().getTile();  **if** (tile != **null**) {  tile.isHeuristicActive = Boolean.parseBoolean(((JComboboxItem) isHeuristicActive.getSelectedItem()).getId());  }  }  });  file.setText("Dosya");  exit.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.awt.event.KeyEvent.VK\_F4, java.awt.event.InputEvent.ALT\_MASK));  exit.setText("Çıkış");  exit.addActionListener(**new** java.awt.event.ActionListener() {  @Override  **public** **void** actionPerformed(java.awt.event.ActionEvent evt) {  exitActionPerformed(evt);  }  });  file.add(exit);  menuBar.add(file);  file.getAccessibleContext().setAccessibleDescription("Dosya İşlemleri");  functions.setText("İşlevler");  clear.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.awt.event.KeyEvent.VK\_N, java.awt.event.InputEvent.CTRL\_MASK));  clear.setText("Temizle");  clear.addActionListener(**new** java.awt.event.ActionListener() {  @Override  **public** **void** actionPerformed(java.awt.event.ActionEvent evt) {  clean(evt);  }  });  functions.add(clear);  functions.add(seperator);  solveProblem.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.awt.event.KeyEvent.VK\_F, java.awt.event.InputEvent.CTRL\_MASK));  solveProblem.setText("Çözüm Bul");  solveProblem.setToolTipText("Çözüm Bul");  solveProblem.addActionListener(**new** java.awt.event.ActionListener() {  @Override  **public** **void** actionPerformed(java.awt.event.ActionEvent evt) {  findSolution(evt);  }  });  functions.add(solveProblem);  menuBar.add(functions);  aboutMenu.setText(" ? ");  about.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.awt.event.KeyEvent.VK\_F1, 0));  about.setText("Hakkında");  about.addActionListener(**new** java.awt.event.ActionListener() {  @Override  **public** **void** actionPerformed(java.awt.event.ActionEvent evt) {  about(evt);  }  });  aboutMenu.add(about);  menuBar.add(aboutMenu);  setJMenuBar(menuBar);  javax.swing.GroupLayout layout = **new** javax.swing.GroupLayout(getContentPane());  getContentPane().setLayout(layout);  layout.setHorizontalGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING).addGroup(  layout.createSequentialGroup()  .addContainerGap()  .addGroup(  layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  .addGroup(  layout.createSequentialGroup().addComponent(drawingArea, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)  .addContainerGap())  .addGroup(  javax.swing.GroupLayout.Alignment.TRAILING,  layout.createSequentialGroup().addGap(0, 364, Short.MAX\_VALUE)  .addComponent(statusBar, javax.swing.GroupLayout.PREFERRED\_SIZE, 233, javax.swing.GroupLayout.PREFERRED\_SIZE).addContainerGap())  .addGroup(  javax.swing.GroupLayout.Alignment.TRAILING,  layout.createSequentialGroup().addGap(8, 8, 8).addComponent(tileSelectionText).addGap(18, 18, 18)  .addComponent(whichTile, javax.swing.GroupLayout.PREFERRED\_SIZE, 127, javax.swing.GroupLayout.PREFERRED\_SIZE).addGap(18, 18, 18)  .addComponent(isHeuristicActive, javax.swing.GroupLayout.PREFERRED\_SIZE, 150, javax.swing.GroupLayout.PREFERRED\_SIZE)  .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)  .addComponent(prev, javax.swing.GroupLayout.PREFERRED\_SIZE, 52, javax.swing.GroupLayout.PREFERRED\_SIZE)  .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)  .addComponent(next, javax.swing.GroupLayout.PREFERRED\_SIZE, 53, javax.swing.GroupLayout.PREFERRED\_SIZE).addGap(25, 25, 25)))));  layout.setVerticalGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING).addGroup(  layout.createSequentialGroup()  .addContainerGap()  .addGroup(  layout.createParallelGroup(javax.swing.GroupLayout.Alignment.CENTER)  .addComponent(whichTile, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)  .addComponent(isHeuristicActive, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)  .addComponent(next, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)  .addComponent(tileSelectionText, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)  .addComponent(prev, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE))  .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)  .addComponent(drawingArea, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE).addGap(2, 2, 2)  .addComponent(statusBar, javax.swing.GroupLayout.PREFERRED\_SIZE, 22, javax.swing.GroupLayout.PREFERRED\_SIZE)));  drawingArea.getAccessibleContext().setAccessibleName("CizimSahasi");  drawingArea.getAccessibleContext().setAccessibleDescription("");  pack();  }  **private** **void** nextActionPerformed(java.awt.event.ActionEvent evt) {  chessDiscoverer.setSolutionStep(chessDiscoverer.getSolutionStep() + 1);  showStep();  }  **private** **void** prevActionPerformed(java.awt.event.ActionEvent evt) {  chessDiscoverer.setSolutionStep(chessDiscoverer.getSolutionStep() - 1);  showStep();  }  **private** **void** showStep() {  **int**[] solution = chessDiscoverer.getSolution();  **if** (solution[solution.length - 1] == -1) {  deactiveButtons(0);  statusBar.setText("");  }  **else** {  **if** (chessDiscoverer.getSolutionStep() == 0) {  deactiveButtons(2);  }  **else** **if** (chessDiscoverer.getSolutionStep() == solution.length - 1) {  deactiveButtons(3);  }  **else** {  deactiveButtons(1);  }  chessDiscoverer.getBoard().setTilesStates(Board.getPositionalBoardState(chessDiscoverer.getBoard().getTile(), chessDiscoverer.getSolution(), chessDiscoverer.getSolutionStep()));  statusBar.setText((chessDiscoverer.getSolutionStep() + 1) + ". Adım");  }  }  **class** DrawingHelper **extends** Thread {  **public** **int** con = 0;  **public** **int** intType = 0;  @Override  **public** **void** interrupt() {  **if** (intType == 0)  con++;  }  @Override  **public** **void** run() {  **while** (con == 0) {  BufferedImage img = **new** BufferedImage(drawingArea.getWidth(), drawingArea.getHeight(), BufferedImage.TYPE\_INT\_ARGB);  Graphics2D threadDrawerTMP = (Graphics2D) img.getGraphics();  threadDrawerTMP.setColor(Color.white);  threadDrawerTMP.fillRect(0, 0, drawingArea.getSize().width, drawingArea.getSize().height);  **try** {  /\*+++ Dikey ve Yatay Çizgiler \*/  **int** sliceSize = chessDiscoverer.getBoard().getTilesStates().length;  **double** YWidth = drawingArea.getSize().height / (**double**) sliceSize;  **double** XWidth = drawingArea.getSize().width / (**double**) sliceSize;  threadDrawerTMP.setColor(Color.black);  **for** (**int** i = 0; i <= sliceSize; i++) {  **if** (i == 0) {  threadDrawerTMP.drawLine(0, (**int**) (YWidth \* i), drawingArea.getSize().width, (**int**) (YWidth \* i));  }  **else** {  threadDrawerTMP.drawLine(0, (**int**) (YWidth \* i - 1), drawingArea.getSize().width, (**int**) (YWidth \* i - 1));  }  }  **for** (**int** i = 0; i <= sliceSize; i++) {  **if** (i == 0) {  threadDrawerTMP.drawLine((**int**) (XWidth \* i), 0, (**int**) (XWidth \* i), drawingArea.getSize().height);  }  **else** {  threadDrawerTMP.drawLine((**int**) (XWidth \* i - 1), 0, (**int**) (XWidth \* i - 1), drawingArea.getSize().height);  }  }  **for** (**int** x = 0; x <= sliceSize; x++) {  **for** (**int** y = 0; y <= sliceSize; y++) {  **if** (((x + y) & 1) == 1) {  threadDrawerTMP.setColor(Color.BLACK);  threadDrawerTMP.fillRect((**int**) (XWidth \* x), (**int**) (YWidth \* y), (**int**) XWidth, (**int**) YWidth);  }  }  }  /\*--- Dikey ve Yatay Çizgiler \*/  **for** (**int** j = 0; j < sliceSize; j++) { // Xs  **for** (**int** i = 0; i < sliceSize; i++) { // Ys  **switch** (chessDiscoverer.getBoard().getTilesStates()[j][i]) {  **case** AVAILABLE:  threadDrawerTMP.setColor(Color.GREEN);  threadDrawerTMP.fillRect((**int**) (XWidth \* i + XWidth \* 3 / 16), (**int**) (YWidth \* j + YWidth \* 3 / 16), (**int**) (XWidth \* 5 / 8), (**int**) (YWidth \* 5 / 8));  threadDrawerTMP.setColor(Color.BLACK);  threadDrawerTMP.drawRect((**int**) (XWidth \* i + XWidth \* 3 / 16), (**int**) (YWidth \* j + YWidth \* 3 / 16), (**int**) (XWidth \* 5 / 8), (**int**) (YWidth \* 5 / 8));  **break**;  **case** \_CURRENT\_:  **if** (chessDiscoverer.getBoard().getTile() != **null**) {  Image image = chessDiscoverer.getBoard().getTile().getIcon();  **if** (image != **null**)  threadDrawerTMP.drawImage(image, (**int**) XWidth \* i + (((**int**) XWidth - image.getWidth(**null**)) / 2), (**int**) YWidth \* j  + (((**int**) YWidth - image.getHeight(**null**)) / 2), **null**);  }  **break**;  **case** NOT\_EMPTY:  threadDrawerTMP.setColor(Color.RED);  threadDrawerTMP.fillRect((**int**) (XWidth \* i + XWidth \* 3 / 16), (**int**) (YWidth \* j + YWidth \* 3 / 16), (**int**) (XWidth \* 5 / 8), (**int**) (YWidth \* 5 / 8));  threadDrawerTMP.setColor(Color.BLACK);  threadDrawerTMP.drawRect((**int**) (XWidth \* i + XWidth \* 3 / 16), (**int**) (YWidth \* j + YWidth \* 3 / 16), (**int**) (XWidth \* 5 / 8), (**int**) (YWidth \* 5 / 8));  **break**;  **case** \_\_EMPTY\_\_:  **break**;  **default**:  }  }  }  UI.**this**.threadDrawerGraphics.drawImage(img, 0, 0, **null**);  Thread.sleep(200);  }  **catch** (InterruptedException ex) {}  }  }  }  **class** JComboboxItem {  **private** String description;  **private** String id;  **public** JComboboxItem(String id, String description) {  **this**.id = id;  **this**.description = description;  }  **public** String getDescription() {  **return** description;  }  **public** String getId() {  **return** id;  }  @Override  **public** String toString() {  **return** description;  }  }  } |

## Start.java

|  |
| --- |
| **package** com.veysi.satranc;  **import** com.veysi.satranc.ui.UI;  **public** **class** Start {  UI ui;  Start(**int** size) {  **if** (size == 0)  ui = **new** UI(8);  **else**  ui = **new** UI(size);  ui.setVisible(**true**);  }  **public** **static** **void** main(String[] args) **throws** CloneNotSupportedException, InterruptedException {  **int** size = 0;  **for** (**int** i = 0; i < args.length; i++) {  String string = args[i];  **if** (string.equals("-size"))  size = Integer.parseInt(args[i + 1]);  }  **new** Start(size);  }  } |

# Kaynakça

1. <http://en.wikipedia.org/wiki/Depth-first_search>
2. <http://www.cs.washington.edu/education/courses/cse326/03su/homework/hw3/dfs.html>
3. <http://stackoverflow.com/questions/3194545/how-to-stop-a-java-thread-gracefully>
4. <http://www.cs.cmu.edu/~sganzfri/REUPaper.pdf>
5. <http://mirran.web.surftown.se/knight/bWarnsd.htm>