FAKULTETI I INXHINIERISË ELEKTRIKE DHE KOMPJUTERIKE

DEPARTAMENTI I INXHINIERISË KOMPJUTERIKE



**Projekti**

Sliding Puzzle Algorithm

**Data:** 22. 05. 2021

**Punoi:** Fitore Morina, Lirim Islami, Uran Lajçi

**Veglat e përdorura:** Microsoft Word, Google Docs, vscode

**Sistemi Operativ:** Windows 10

**ID:** , 18071410065

**Gjuha Programuese:**

**Email:** fitore.morina11@student.uni-pr.edu, lirim.islami@student.uni-pr.edu, uran.lajci@student.uni-pr.edu

**Lënda:** Dizajni dhe Analiza e Algoritmeve

**Mentorë:** Avni Rexhepi, Dardan Shabani

Permbajtja

[**1.** **Hyrje** 3](#_Toc72176067)

[**2.** **Përshkrimi dhe spjegimi i Sliding Puzzle Algoritmit** 4](#_Toc72176068)

[**3.** **Perdorimet e Sliding Puzzle Algoritmit** 5](#_Toc72176069)

[**4.** **Implementimi dhe shpjegimi i kodit bazë** 6](#_Toc72176070)

[**4.1. Kodi Burimor** 7](#_Toc72176071)

[**4.2. Testimet** 25](#_Toc72176072)

[**4.2.1. Shpjegimi se si punon aplikacioni** 25](#_Toc72176073)

[**4.2.2. Krahasimi i algoritmeve** 28](#_Toc72176074)

[**4.2.3. Levizja e pozitave** 31](#_Toc72176075)

[**4.2.4. Validimet** 32](#_Toc72176076)

[**5.** **Referencat** 34](#_Toc72176077)

# **Hyrje**

Puzzles rrëshqitëse filluan në 1880. Në pak muaj pas prezantimit të tij, njerëzit në të gjithë botën ishin të zhytur në përpjekje për të zgjidhur atë që u bë e njohur si Puzzle 15. Ai përbëhej nga një rrjet 4 me 4, me 15 katrorë të numëruar dhe një hapësirë të lënë bosh. Ideja ishte që të përziheshin numrat, dhe pastaj t'i rirregulloni ato në renditje numerike duke i rrëshqitur në mënyrë të njëpasnjëshme në hapësirën boshe. Ky enigmë e parë rrëshqitëse u shpik nga Noyes Chapman nga New York, i cili doli me këtë ide gjatë viteve 1870. Fabrikat nuk mund të vazhdonin me kërkesën për 15 Puzzle. Aq shumë njerëz u kapën nga keto enigma, saqë punëdhënësit ishin të shqetësuar për punëtorët duke injoruar detyrat e tyre për të punuar në enigmë. Ligjvënësit në Gjermani u panë duke u përpjekur t'i zgjidhnin ato. Mania rrëshqitëse e enigmave u përmend në këngët dhe shfaqjet popullore. Një dentist nga Worcester, Massachusetts i ofroi një sërë dhëmbësh dhe 100 dollarë çdo personi që mund të zgjidhte enigmën nga një rregullim i caktuar fillestar. Ai më vonë e rriti çmimin në 1.000 dollarë. Njerëzit në të gjithë vendin u përpoqën ta fitonin këtë shumë të madhe, por askush nuk ia doli. Zhvillimi tjetër në puzzles rrëshqitëse erdhi në 1909 kur Lewis W. Hardy shpiku enigmën e parë rrëshqitëse duke përdorur pjesë që ishin drejtkëndëshe. Ai e quajti atë Puzzle Pennant, duke i dhënë asaj një temë bejsbolli. Ai ishte i njohur gjithashtu si Puzzle i Babait .Ma's Puzzle, e cila pasoi në 1927, prezantoi dy pjesë në formë L, duke e bërë edhe më të vështirë zgjidhjen. Që kur kompjuteri filloi të luante një rol në hartimin dhe zgjidhjen e enigmave, shumëllojshmëria e enigmave është rritur. Qindra fotografi janë shndërruar në rrëshqitës dhe rregullimet gjithnjë e më të vështira të formave gjeometrike kanë sfiduar zgjidhësit.

# **Përshkrimi dhe spjegimi i Sliding Puzzle Algoritmit**

# **Perdorimet e Sliding Puzzle Algoritmit**

# **Implementimi dhe shpjegimi i kodit bazë**

Per te zgjedhur Sliding Puzzle me ane te nje algoritmi duhet qe te prezantohen gjendjet e enigmës si nje graf, dhe pastaj te perdoren algoritmet per gjetje te shtegut si DFS (Depth-first search), BFS (Breadth-first search), Dijkstra, A\* etj. Algoritmet dallojne nga menyra se si arrijn deri te zgjidhje e enigmës, disa algoritme jane me te shpejta se te tjerat siq do te shohim te pjesa e testimeve.

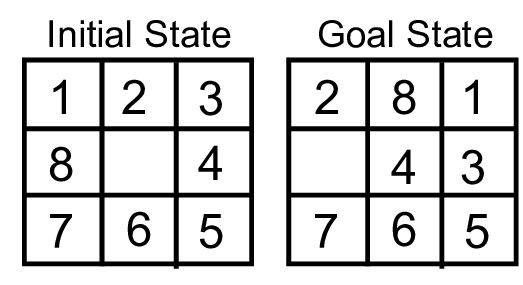


Figura 1. Gjendjet e enigmes

Enigma i ka dy gjendje kryesore, ato jane gjendja fillestare (Initial State) dhe gjendja te cilen deshirojm ta arrijme (Goal State). Midis ketyre dy gjendjeve mund te kemi zero (kur enigma eshte e gjetur) ose shume gjendje tjera.

Gjuha te cilen e kemi perdorur per te implementuar zgjidhjen e enigmes me ane te algoritmeve eshte java, dhe per paraqitjen vizuele te aplikacionit eshte perdorur javafx. Ne programin qe e kemi krijuar gjinden levizjet per te zgjedhur cdo enigme rreshqitse (slidig puzzle) qe mund te zgjidhet te rendit 3 ose 4 te matrices. Algoritmet me te cilat mundesohet zgjidhja e enigmes rreshqitse jane A\* (Astar), BFS, DFS, ID.

Ne IDE (ne e kemi perdorur Eclipse, vscode) duhet te ekzekutohet klasa App.java.

## **4.1. Kodi Burimor**

**Klasa App.java**

import **javafx**.**application**.**Application**;

import **javafx**.**geometry**.**Insets**;

import **javafx**.**geometry**.**Pos**;

import **javafx**.**scene**.**Scene**;

import **javafx**.**scene**.**control**.**Button**;

import **javafx**.**scene**.**control**.**TextField**;

import **javafx**.**scene**.**layout**.**HBox**;

import **javafx**.**scene**.**layout**.**VBox**;

import **javafx**.**scene**.**text**.**Font**;

import **javafx**.**scene**.**text**.**FontWeight**;

import **javafx**.**scene**.**text**.**Text**;

import **javafx**.**stage**.**Stage**;

import **java**.**util**.**HashMap**;

import **java**.**util**.**Stack**;

import **java**.**util**.**PriorityQueue**;

import **java**.**util**.**Queue**;

import **java**.**util**.**LinkedList**;

import **java**.**math**.**BigInteger**;

public class **App** extends **Application** {

  static public **Stack**<**GTNode**> solPath = new **Stack**<**GTNode**>();

  static public **HashMap**<**BigInteger**, **Integer**> visited = new **HashMap**<**BigInteger**, **Integer**>();

  static double startTime;

  static double endTime;

  public static void **main**(**String**[] args) {

**launch**(args);

  }

  @**Override**

  public void **start**(**Stage** primaryStage) {

    primaryStage.**setTitle**("Sliding Puzzle");

**Text** shenoDimensionin = new **Text**("Shenoni dimenzionin: ");

**TextField** dimensioni = new **TextField**();

    dimensioni.**setMaxWidth**(150);

    dimensioni.**setMaxHeight**(50);

**Text** shenoAlgoritmin = new **Text**("\nShenoni algoritmin: ");

**TextField** algoritmi = new **TextField**();

    algoritmi.**setMaxWidth**(150);

    algoritmi.**setMaxHeight**(50);

**Button** bRandom = new **Button**("Random");

    bRandom.**setMaxWidth**(150);

**Button** bShenoj = new **Button**("Shenoj");

    bShenoj.**setMaxWidth**(150);

**HBox** rootNode1 = new **HBox**();

**TextField** Value1 = new **TextField**("");

    rootNode1.**getChildren**().**add**(Value1);

    Value1.**setMaxSize**(30, 30);

**TextField** Value2 = new **TextField**("");

    rootNode1.**getChildren**().**add**(Value2);

    Value2.**setMaxSize**(30, 30);

**TextField** Value3 = new **TextField**("");

    rootNode1.**getChildren**().**add**(Value3);

    Value3.**setMaxSize**(30, 30);

**HBox**.**setMargin**(Value1, new **Insets**(0,2,0,140));

**HBox**.**setMargin**(Value2, new **Insets**(0,2,0,0));

**HBox**.**setMargin**(Value3, new **Insets**(0,2,0,0));

    rootNode1.**setVisible**(false);

**HBox** rootNode2 = new **HBox**();

**TextField** Value4 = new **TextField**("");

    rootNode2.**getChildren**().**add**(Value4);

    Value4.**setMaxSize**(30, 30);

**TextField** Value5 = new **TextField**("");

    rootNode2.**getChildren**().**add**(Value5);

    Value5.**setMaxSize**(30, 30);

**TextField** Value6 = new **TextField**("");

    rootNode2.**getChildren**().**add**(Value6);

    Value6.**setMaxSize**(30, 30);

**HBox**.**setMargin**(Value4, new **Insets**(0,2,0,140));

**HBox**.**setMargin**(Value5, new **Insets**(0,2,0,0));

**HBox**.**setMargin**(Value6, new **Insets**(0,2,0,0));

    rootNode2.**setVisible**(false);

**HBox** rootNode3 = new **HBox**();

**TextField** Value7 = new **TextField**("");

    rootNode3.**getChildren**().**add**(Value7);

    Value7.**setMaxSize**(30, 30);

**TextField** Value8 = new **TextField**("");

    rootNode3.**getChildren**().**add**(Value8);

    Value8.**setMaxSize**(30, 30);

**TextField** Value9 = new **TextField**("");

    rootNode3.**getChildren**().**add**(Value9);

    Value9.**setMaxSize**(30, 30);

**HBox**.**setMargin**(Value7, new **Insets**(0,2,0,140));

**HBox**.**setMargin**(Value8, new **Insets**(0,2,0,0));

**HBox**.**setMargin**(Value9, new **Insets**(0,2,0,0));

    rootNode3.**setVisible**(false);

**Button** gjejEnigmen = new **Button**("Gjej enigmen");

    gjejEnigmen.**setMaxWidth**(150);

    gjejEnigmen.**setVisible**(false);

    bRandom.**setOnAction**( e ->  {

      if(dimensioni.**getText**().**isEmpty**() && algoritmi.**getText**().**isEmpty**()) {

**AlertBox**("Duhet te plotesohen fushat Sheno dimenzionin" +

                 "\n\t\t   dhe Sheno algoritmin.");

      }

      else if (!dimensioni.**getText**().**isEmpty**() && algoritmi.**getText**().**isEmpty**()) {

**AlertBox**("Duhet te plotesohet fusha Sheno algoritmin.");

      }

      else if (dimensioni.**getText**().**isEmpty**() && !algoritmi.**getText**().**isEmpty**()) {

**AlertBox**("Duhet te plotesohet fusha Sheno dimenzionin.");

      }

      else {

        if(

          (dimensioni.**getText**().**equals**("3") || dimensioni.**getText**().**equals**("4")) &&

          (algoritmi.**getText**().**equals**("Astar") || algoritmi.**getText**().**equals**("BFS") ||

          algoritmi.**getText**().**equals**("DFS") || algoritmi.**getText**().**equals**("ID")))  {

            int Dimensioni = **Integer**.**parseInt**(dimensioni.**getText**());

**randomInitialState**(Dimensioni, algoritmi.**getText**());

        }

        else {

**AlertBoxMeDimenzione**(

              "Te fusha e dimenzionit duhet te shnoni 3 ose 4 per rendin\n" +

              " e matrices. Kurse te fusha e alforitmit duhet te shenoni\n" +

              " njerin nga algorimet Astar, BFS, DFS, ID.", 600, 200);

        }

      }

    });

    bShenoj.**setOnAction**(e ->  {

      if(dimensioni.**getText**().**isEmpty**() && algoritmi.**getText**().**isEmpty**()) {

**AlertBox**("Duhet te plotesohen fushat Sheno dimenzionin \n\t\t " +

                "  dhe Sheno algoritmin.");

      }

      else if (!dimensioni.**getText**().**isEmpty**() && algoritmi.**getText**().**isEmpty**()) {

**AlertBox**("Duhet te plotesohet fusha Sheno algoritmin.");

      }

      else if (dimensioni.**getText**().**isEmpty**() && !algoritmi.**getText**().**isEmpty**()) {

**AlertBox**("Duhet te plotesohet fusha Sheno dimenzionin.");

      }

      else {

        if(

          (dimensioni.**getText**().**equals**("3") || dimensioni.**getText**().**equals**("4")) &&

          (algoritmi.**getText**().**equals**("Astar") || algoritmi.**getText**().**equals**("BFS") ||

          algoritmi.**getText**().**equals**("DFS") || algoritmi.**getText**().**equals**("ID")))  {

              rootNode1.**setVisible**(true);

              rootNode2.**setVisible**(true);

              rootNode3.**setVisible**(true);

              gjejEnigmen.**setVisible**(true);

        }

        else {

**AlertBoxMeDimenzione**("Te fusha e dimenzionit duhet te shnoni " +

                "3 ose 4 per rendin\n e matrices. Kurse te fusha e alforitmit" +

                "  duhet te shenoni\n njerin nga algoritmet Astar, BFS, DFS, ID."

                , 600, 200);

        }

      }

    });

    gjejEnigmen.**setOnAction**(e -> {

      if(

        Value1.**getText**().**isEmpty**() || Value2.**getText**().**isEmpty**() ||

        Value3.**getText**().**isEmpty**() || Value4.**getText**().**isEmpty**() ||

        Value5.**getText**().**isEmpty**() || Value6.**getText**().**isEmpty**() ||

        Value7.**getText**().**isEmpty**() || Value8.**getText**().**isEmpty**() ||

        Value9.**getText**().**isEmpty**()) {

**AlertBox**("Duhet te plotesohen te gjitha fushat e matrices.");

      }

      else if(

       !Value1.**getText**().**matches**("[0-9]") || !Value2.**getText**().**matches**("[0-9]") ||

       !Value3.**getText**().**matches**("[0-9]") || !Value4.**getText**().**matches**("[0-9]") ||

       !Value5.**getText**().**matches**("[0-9]") || !Value6.**getText**().**matches**("[0-9]") ||

       !Value7.**getText**().**matches**("[0-9]") || !Value8.**getText**().**matches**("[0-9]") ||

       !Value9.**getText**().**matches**("[0-9]")) {

**AlertBox**("Duhet te shenohen vetem numra nga 0 deri ne 8.");

      }

      else {

        int Dimensioni = **Integer**.**parseInt**(dimensioni.**getText**());

**Integer** value1 = **Integer**.**valueOf**(Value1.**getText**());

**Integer** value2 = **Integer**.**valueOf**(Value2.**getText**());

**Integer** value3 = **Integer**.**valueOf**(Value3.**getText**());

**Integer** value4 = **Integer**.**valueOf**(Value4.**getText**());

**Integer** value5 = **Integer**.**valueOf**(Value5.**getText**());

**Integer** value6 = **Integer**.**valueOf**(Value6.**getText**());

**Integer** value7 = **Integer**.**valueOf**(Value7.**getText**());

**Integer** value8 = **Integer**.**valueOf**(Value8.**getText**());

**Integer** value9 = **Integer**.**valueOf**(Value9.**getText**());

        int matrix[][] = { { value1, value2, value3 },

                           { value4, value5, value6 },

                           { value7, value8, value9 } };

**preWrittenInitialState**(Dimensioni, algoritmi.**getText**(), matrix);

      }

    });

**VBox** root = new **VBox**();

**VBox**.**setMargin**(rootNode1, new **Insets**(2, 2, 2, 2));

**VBox**.**setMargin**(rootNode2, new **Insets**(2, 2, 2, 2));

**VBox**.**setMargin**(rootNode3, new **Insets**(2, 2, 2, 2));

**VBox**.**setMargin**(gjejEnigmen, new **Insets**(10, 0, 0, 0));

**VBox**.**setMargin**(bRandom, new **Insets**(10, 0, 0, 0));

**VBox**.**setMargin**(bShenoj, new **Insets**(10, 0, 10, 0));

    root.**setPadding**(new **Insets**(10, 10, 10, 10));

    root.**getChildren**().**addAll**(shenoDimensionin, dimensioni, shenoAlgoritmin,

                                algoritmi, bRandom, bShenoj, rootNode1, rootNode2,

                                rootNode3, gjejEnigmen);

    root.**setAlignment**(**Pos**.CENTER);

    primaryStage.**setScene**(new **Scene**(root, 400, 450));

    primaryStage.**show**();

}

public static void **AlertBoxMeDimenzione**(**String** mesazhi, int gjersia, int gjatesia) {

**Text** Shpjegimi = new **Text**(" ");

  Shpjegimi.**setFont**(**Font**.**font**("Arial", **FontWeight**.BOLD, 20));

  Shpjegimi.**setText**(mesazhi);

**VBox** boxShpjegimi = new **VBox**();

  boxShpjegimi.**setAlignment**(**Pos**.CENTER);

  boxShpjegimi.**getChildren**().**addAll**(Shpjegimi);

**Scene** skenaShpjegimi = new **Scene**(boxShpjegimi, gjersia, gjatesia);

**Stage** pamjaShpjegimi = new **Stage**();

  pamjaShpjegimi.**setTitle**("Shpjegimi");

  pamjaShpjegimi.**setScene**(skenaShpjegimi);

  pamjaShpjegimi.**show**();

}

public static void **AlertBox**(**String** mesazhi) {

**Text** Shpjegimi = new **Text**(" ");

  Shpjegimi.**setFont**(**Font**.**font**("Arial", **FontWeight**.BOLD, 20));

  Shpjegimi.**setText**(mesazhi);

**VBox** boxShpjegimi = new **VBox**();

  boxShpjegimi.**setAlignment**(**Pos**.CENTER);

  boxShpjegimi.**getChildren**().**addAll**(Shpjegimi);

**Scene** skenaShpjegimi = new **Scene**(boxShpjegimi, 500, 200);

**Stage** pamjaShpjegimi = new **Stage**();

  pamjaShpjegimi.**setTitle**("Shpjegimi");

  pamjaShpjegimi.**setScene**(skenaShpjegimi);

  pamjaShpjegimi.**show**();

}

public static void **randomInitialState**(int dimension, **String** choseAlgorithm) {

**GTNode** initial = new **GTNode**(dimension);

  initial.**fill**();

  initial.**shuffle**();

  if(choseAlgorithm.**equals**("Astar")) {

**Astar**(initial);

  }

  else if(choseAlgorithm.**equals**("BFS")) {

**BFS**(initial);

  }

  else if(choseAlgorithm.**equals**("DFS")) {

**DFS**(initial);

  }

  else if(choseAlgorithm.**equals**("ID")) {

**ID**(initial);

  }

  else {

**System**.out.**print**("Keni shenuar gabim");

  }

}

public static void **preWrittenInitialState**(int dimension,

**String** choseAlgorithm, int matrix[][]) {

**GTNode** initial = new **GTNode**(dimension);

  initial.matrix = matrix;

  if(!initial.**isSolvable**()) {

**AlertBox**("Engima qe e keni shenuar nuk eshte e zgjidhshme\n" +

              "\t\tju lutem ndrroni inputin.");

  }

  else {

    if(choseAlgorithm.**equals**("Astar")) {

**Astar**(initial);

    }

    else if(choseAlgorithm.**equals**("BFS")) {

**BFS**(initial);

    }

    else if(choseAlgorithm.**equals**("DFS")) {

**DFS**(initial);

    }

    else if(choseAlgorithm.**equals**("ID")) {

**ID**(initial);

    }

    else {

**System**.out.**print**("Keni shenuar gabim");

    }

  }

}

public static void **Astar**(**GTNode** initial) {

  startTime = **System**.**currentTimeMillis**();

**PriorityQueue**<**GTNode**> openList = new **PriorityQueue**<**GTNode**>();

  openList.**offer**(initial);

**GTNode** state;

  while (!openList.**isEmpty**()) {

    state = openList.**poll**();

    visited.**put**(state.**hash**(state.matrix), state.astar);

    if (state.**isComplete**()) {

**path**(state);

      endTime = **System**.**currentTimeMillis**();

**printInfo**((endTime - startTime) / 1000.00);

      return;

    }

  state.**exploreAstar**(visited, openList);

  }

}

public static void **BFS**(**GTNode** initial) {

  startTime = **System**.**currentTimeMillis**();

**Queue**<**GTNode**> openList = new **LinkedList**<**GTNode**>();

  openList.**offer**(initial);

**GTNode** state;

  while (!openList.**isEmpty**()) {

    state = openList.**poll**();

    visited.**put**(state.**hash**(state.matrix), 0);

    if (state.**isComplete**()) {

**path**(state);

      endTime = **System**.**currentTimeMillis**();

**printInfo**((endTime - startTime) / 1000.00);

      return;

    }

  state.**explore**(visited, openList);

  }

}

public static void **DFS**(**GTNode** initial) {

  startTime = **System**.**currentTimeMillis**();

**Stack**<**GTNode**> openList = new **Stack**<**GTNode**>();

  openList.**push**(initial);

**GTNode** state;

  while (!openList.**isEmpty**()) {

    state = openList.**pop**();

    visited.**put**(state.**hash**(state.matrix), 0);

    if (state.**isComplete**()) {

**path**(state);

      endTime = **System**.**currentTimeMillis**();

**printInfo**((endTime - startTime) / 1000.00);

      return;

    }

  state.**explore**(visited, openList, -1);

  }

}

public static void **ID**(**GTNode** initial) {

  startTime = **System**.**currentTimeMillis**();

**Stack**<**GTNode**> openList = new **Stack**<**GTNode**>();

  int limit = 0;

  while (true) {

    openList.**clear**();

    visited.**clear**();

    openList.**push**(initial);

**GTNode** state;

    while (!openList.**isEmpty**()) {

      state = openList.**pop**();

      visited.**put**(state.**hash**(state.matrix), 0);

      if (state.**isComplete**()) {

**path**(state);

        endTime = **System**.**currentTimeMillis**();

**printInfo**((endTime - startTime) / 1000.00);

        return;

      }

      state.**explore**(visited, openList, limit);

    }

    limit++;

  }

}

public static void **path**(**GTNode** c) {

  while (c.parent != null) {

    solPath.**push**(c);

    c = c.parent;

  }

  solPath.**push**(c);

}

public static void **printInfo**(double time) {

  int moves = solPath.**size**() - 1;

**Text** antari00 = new **Text**(" "), antari01 = new **Text**(" "),

       antari02 = new **Text**(" "), antari10 = new **Text**(" "),

       antari11 = new **Text**(" "), antari12 = new **Text**(" "),

       antari20 = new **Text**(" "), antari21 = new **Text**(" "),

       antari22 = new **Text**(" ");

  antari00.**setFont**(**Font**.**font**("Arial", **FontWeight**.BOLD, 20));

  antari01.**setFont**(**Font**.**font**("Arial", **FontWeight**.BOLD, 20));

  antari02.**setFont**(**Font**.**font**("Arial", **FontWeight**.BOLD, 20));

  antari10.**setFont**(**Font**.**font**("Arial", **FontWeight**.BOLD, 20));

  antari11.**setFont**(**Font**.**font**("Arial", **FontWeight**.BOLD, 20));

  antari12.**setFont**(**Font**.**font**("Arial", **FontWeight**.BOLD, 20));

  antari20.**setFont**(**Font**.**font**("Arial", **FontWeight**.BOLD, 20));

  antari21.**setFont**(**Font**.**font**("Arial", **FontWeight**.BOLD, 20));

  antari22.**setFont**(**Font**.**font**("Arial", **FontWeight**.BOLD, 20));

**Button** kontrollo = new **Button**("Kontrollo");

**Text** hapsire = new **Text**("\n");

  kontrollo.**setOnAction**(e -> {

    if (!solPath.**empty**()) {

**GTNode** p = (**GTNode**) solPath.**pop**();

      int matrix[][] = {

          { p.**print**().**get**(0), p.**print**().**get**(1), p.**print**().**get**(2) },

          { p.**print**().**get**(3), p.**print**().**get**(4), p.**print**().**get**(5) },

          { p.**print**().**get**(6), p.**print**().**get**(7), p.**print**().**get**(8) } };

      antari00.**setText**(**String**.**valueOf**(matrix[0][0]));

      antari01.**setText**(**String**.**valueOf**(matrix[0][1]));

      antari02.**setText**(**String**.**valueOf**(matrix[0][2]));

      antari10.**setText**(**String**.**valueOf**(matrix[1][0]));

      antari11.**setText**(**String**.**valueOf**(matrix[1][1]));

      antari12.**setText**(**String**.**valueOf**(matrix[1][2]));

      antari20.**setText**(**String**.**valueOf**(matrix[2][0]));

      antari21.**setText**(**String**.**valueOf**(matrix[2][1]));

      antari22.**setText**(**String**.**valueOf**(matrix[2][2]));

    }

  });

**HBox** rreshti1 = new **HBox**();

  rreshti1.**getChildren**().**addAll**(antari00, antari01, antari02);

  rreshti1.**setAlignment**(**Pos**.CENTER);

  rreshti1.**setSpacing**(15);

**HBox** rreshti2 = new **HBox**();

  rreshti2.**getChildren**().**addAll**(antari10, antari11, antari12);

  rreshti2.**setAlignment**(**Pos**.CENTER);

  rreshti2.**setSpacing**(15);

**HBox** rreshti3 = new **HBox**();

  rreshti3.**getChildren**().**addAll**(antari20, antari21, antari22);

  rreshti3.**setAlignment**(**Pos**.CENTER);

  rreshti3.**setSpacing**(15);

**VBox** box = new **VBox**();

  box.**getChildren**().**addAll**(kontrollo, hapsire, rreshti1, rreshti2, rreshti3);

  box.**setAlignment**(**Pos**.CENTER);

**Scene** skena = new **Scene**(box, 400, 200);

**Stage** pamja = new **Stage**();

  pamja.**setTitle**("Shfaqja e levizjeve");

  pamja.**setScene**(skena);

  pamja.**show**();

**PrintimiIKohesDheLevizjeve**(moves, time);

}

public static void **PrintimiIKohesDheLevizjeve**(int levizjet, double koha) {

**Text** shumaELevizjeve = new **Text**(" ");

**Text** kohaTotale = new **Text**(" ");

  shumaELevizjeve.**setFont**(**Font**.**font**("Arial", **FontWeight**.BOLD, 20));

  kohaTotale.**setFont**(**Font**.**font**("Arial", **FontWeight**.BOLD, 20));

  shumaELevizjeve.**setText**("Enigma eshte zgjedhur ne " + levizjet + " levizje.");

  kohaTotale.**setText**("Koha per zgjidhje eshte " + koha + " sekonda.");

**VBox** box = new **VBox**();

  box.**setAlignment**(**Pos**.CENTER);

  box.**getChildren**().**addAll**(shumaELevizjeve, kohaTotale);

**Scene** skena = new **Scene**(box, 400, 200);

**Stage** pamja = new **Stage**();

  pamja.**setTitle**("Koha dhe levizjet");

  pamja.**setScene**(skena);

  pamja.**show**();

  }

}

**Klasa GTNode.java**

import **java**.**util**.**ArrayList**;

import **java**.**util**.**HashMap**;

import **java**.**util**.**Stack**;

import **java**.**util**.**PriorityQueue**;

import **java**.**util**.**Queue**;

import **java**.**math**.**BigInteger**;

public class **GTNode** implements **Comparable**<**GTNode**> {

**String** dir[] = { "up", "down", "right", "left" };

  public int dimension;

  public **GTNode** parent;

  public int[][] matrix;

  public int level;

  public int astar;

  public **GTNode**(int size) {

    this.dimension = size;

    matrix = new int[size][size];

    parent = null;

**GTNode** p = null;

    level = 0;

    astar = **heuristic**() + level;

  }

  public **GTNode**(int[][] m, **GTNode** p) {

    matrix = m;

    parent = p;

    level = p.level + 1;

    astar = **heuristic**() + level;

  }

  public void **explore**(**HashMap**<**BigInteger**, **Integer**> visited,

**Queue**<**GTNode**> openList) {

    for (int i = 0; i < 4; i++) {

      int[][] temp = **copy**();

      boolean already = false;

      if (**move**(temp, dir[i])) {

        if (visited.**containsKey**(**hash**(temp))) {

            already = true;

            continue;

        }

        if (!already) {

            openList.**offer**(new **GTNode**(temp, this));

            visited.**put**(**hash**(temp), 0);

        }

      }

    }

  }

  public void **explore**(**HashMap**<**BigInteger**, **Integer**> visited,

**Stack**<**GTNode**> openList, int limit) {

    if (limit != -1)

      if (this.level >= limit)

        return;

    for (int i = 0; i < 4; i++) {

      int[][] temp = **copy**();

      boolean already = false;

      if (**move**(temp, dir[i])) {

        if (visited.**containsKey**(**hash**(temp))) {

          already = true;

          continue;

        }

        if (!already) {

          openList.**push**(new **GTNode**(temp, this));

          visited.**put**(**hash**(temp), 0);

        }

      }

    }

  }

  public void **exploreAstar**(**HashMap**<**BigInteger**, **Integer**> visited,

**PriorityQueue**<**GTNode**> openList) {

    for (int i = 0; i < 4; i++) {

**GTNode** temp = new **GTNode**(this.**copy**(), this);

      boolean worse = false;

      if (**move**(temp.matrix, dir[i])) {

        if (visited.**containsKey**(**hash**(temp.matrix)))

          if (visited.**get**(**hash**(temp.matrix)) < temp.astar) {

            worse = true;

            continue;

          }

          if (!worse) {

            visited.**put**(**hash**(temp.matrix), temp.astar);

            openList.**offer**(temp);

          }

      }

    }

  }

  public int **heuristic**() {

    int c = 1;

    int h = 0;

    for (int i = 0; i < matrix.length; i++) {

      for (int j = 0; j < matrix.length; j++) {

        if (c == matrix.length \* matrix.length)

          continue;

          int x = **find**(c)[0];

          int y = **find**(c)[1];

          h += **Math**.**abs**(x - i) + **Math**.**abs**(y - j);

          c++;

      }

    }

    return h;

  }

  public  **ArrayList**<**Integer**> **print**() {

**ArrayList**<**Integer**> cars = new **ArrayList**<**Integer**>();

    for (int i = 0; i < matrix.length; i++) {

      for (int j = 0; j < matrix.length; j++) {

        cars.**add**(matrix[i][j]);

      }

    }

    return cars;

  }

  public int[][] **fill**() {

    int c = 1;

    for (int i = 0; i < dimension; i++) {

      for (int j = 0; j < dimension; j++) {

        if (i == dimension - 1 && j == dimension - 1)

          break;

          matrix[i][j] = c++;

      }

    }

    return matrix;

  }

  public void **shuffle**() {

    int limit = matrix.length - (matrix.length/2 + 1);

    for (int i = 0; i < limit; i++) {

**move**(matrix, "up");

**move**(matrix, "left");

    }

    for (int i = 0; i < 33 \* dimension; i++) {

**move**(matrix, dir[(int) (**Math**.**random**() \* 4)]);

    }

  }

  public int[] **find**(int n) {

    int[] coord = new int[2];

    for (int i = 0; i < matrix.length; i++) {

      for (int j = 0; j < matrix.length; j++) {

        if (matrix[i][j] == n) {

          coord[0] = i;

          coord[1] = j;

        }

      }

    }

    return coord;

  }

  public boolean **move**(int[][] a, **String** dir) {

    int zCoord[] = **find**(0);

    int x = zCoord[0];

    int y = zCoord[1];

    switch (dir) {

    case "up":

        if (x - 1 >= 0) {

            a[x][y] = a[x - 1][y];

            a[x - 1][y] = 0;

            return true;

        }

        break;

    case "down":

        if (x + 1 <= a.length - 1) {

            a[x][y] = a[x + 1][y];

            a[x + 1][y] = 0;

            return true;

        }

        break;

    case "right":

        if (y + 1 <= a.length - 1) {

            a[x][y] = a[x][y + 1];

            a[x][y + 1] = 0;

            return true;

        }

        break;

    case "left":

        if (y - 1 >= 0) {

            a[x][y] = a[x][y - 1];

            a[x][y - 1] = 0;

            return true;

        }

        break;

    default:

        return false;

    }

    return false;

  }

  public boolean **isComplete**() {

    int c = 1;

    for (int i = 0; i < matrix.length; i++) {

        for (int j = 0; j < matrix.length; j++) {

            if (i == matrix.length - 1 && j == matrix.length - 1)

                break;

            if (matrix[i][j] != c)

                return false;

            c++;

        }

    }

    return true;

  }

  public int[][] **copy**() {

    int[][] copy = new int[matrix.length][matrix.length];

    for (int i = 0; i < matrix.length; i++)

        for (int j = 0; j < matrix.length; j++) {

            copy[i][j] = matrix[i][j];

        }

    return copy;

  }

  public **BigInteger** **hash**(int[][] matrix) {

**String** text = "";

    for (int i = 0; i < matrix.length; i++) {

        for (int j = 0; j < matrix.length; j++) {

            text += **Integer**.**toString**(matrix[i][j]);

        }

    }

**BigInteger** hash = new **BigInteger**(text);

    return hash;

  }

  public boolean **isSolvable**() {

    int linMatrix[] = new int[matrix.length \* matrix.length];

    int count = 0;

    for (int i = 0; i < matrix.length; i++)

        for (int j = 0; j < matrix.length; j++) {

            linMatrix[count] = matrix[i][j];

            count++;

        }

    int parity = 0;

    int gridWidth = (int) **Math**.**sqrt**(linMatrix.length);

    int row = 0;

    int blankRow = 0;

    for (int i = 0; i < linMatrix.length; i++) {

        if (i % gridWidth == 0) {

            row++;

        }

        if (linMatrix[i] == 0) {

            blankRow = row;

            continue;

        }

        for (int j = i + 1; j < linMatrix.length; j++) {

            if (linMatrix[i] > linMatrix[j] && linMatrix[j] != 0) {

                parity++;

            }

        }

    }

    if (gridWidth % 2 == 0) {

        if (blankRow % 2 == 0) {

            return parity % 2 == 0;

        } else {

            return parity % 2 != 0;

        }

    } else {

        return parity % 2 == 0;

    }

  }

  @**Override**

  public int **compareTo**(**GTNode** a) {

    if (this.astar > a.astar)

        return 1;

    else if (a.astar == this.astar)

        return 0;

    else

        return -1;

  }

}

## **4.2. Testimet**

### **4.2.1. Shpjegimi se si punon aplikacioni**

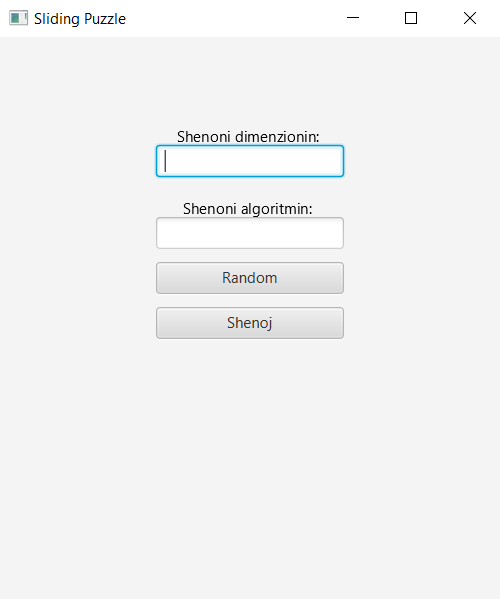


Figura 2. Pamja e pare qe shfaqet pasi te ekzekutohet programi.

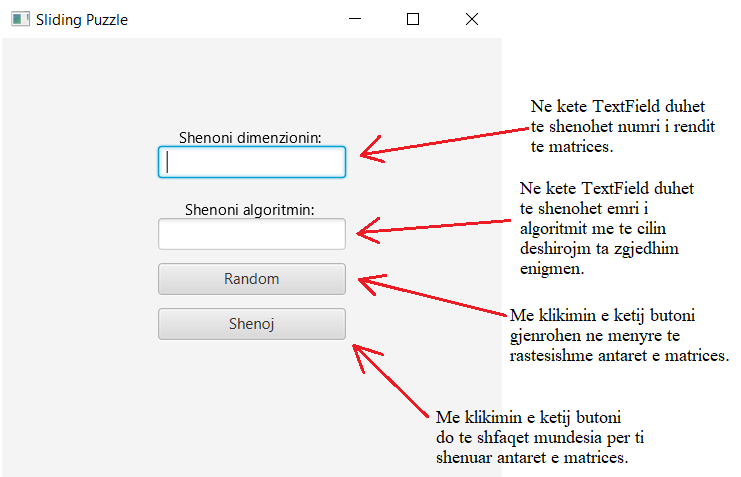


Figura 3. Shpjegimi se cka sherbejne fushat e paraqitura ne figure.

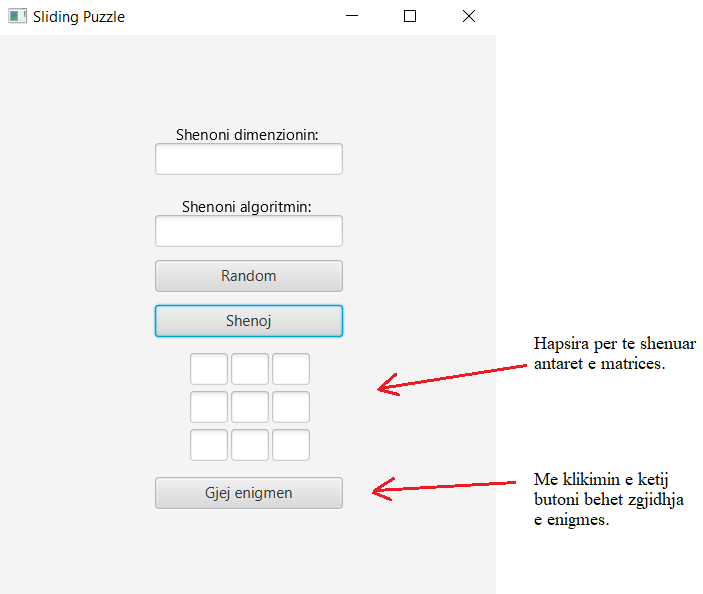


Figura 4. Pamja qe shfaqet pasi te klikohet butoni Shenoj.

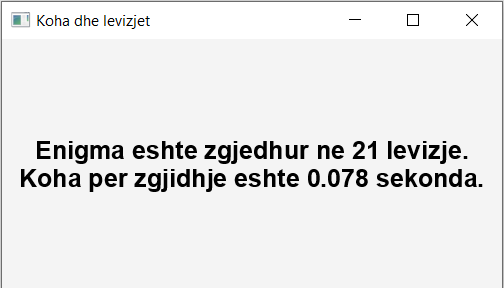


Figura 5. Pamja e pare qe shfaqet pasi te zgjidhet enigma.

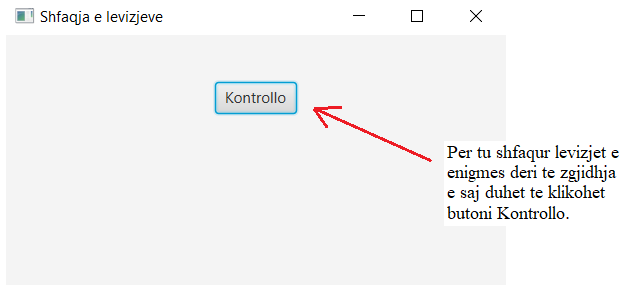


Figura 6. Pamja e dyte qe na shfaqet pasi te zgjedhet enigma.

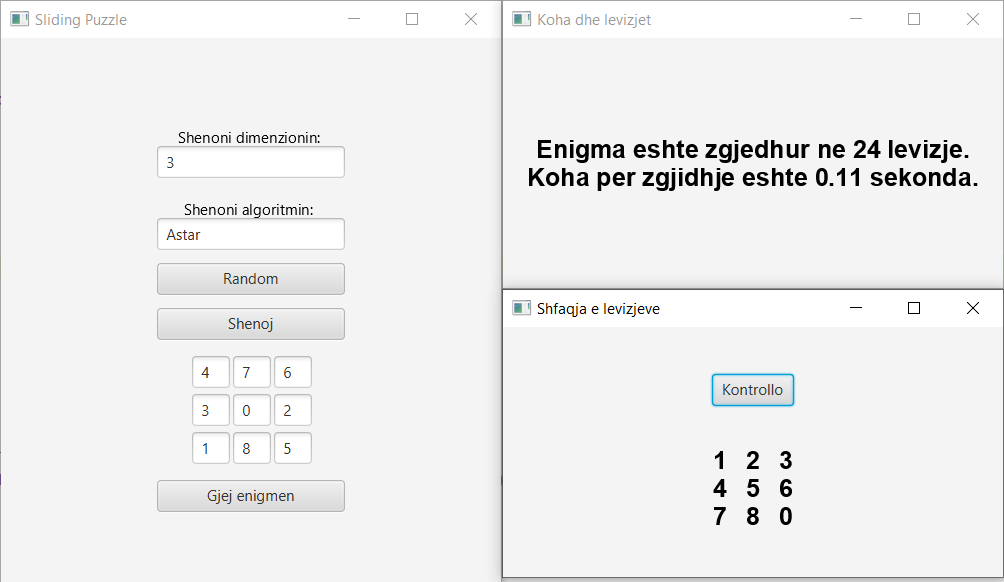


Figura 7. Pamja pas nje ekzekutimi te plote te programit.

### **4.2.2. Krahasimi i algoritmeve**

Figura 8. Koha ne sekoda qe ju duhen algoritmeve te ndryshme per te gjetur zgjidhjen e enigmes

Ne figuren 8 ne anen e majt te figures numrat paraqesin kohen e zgjidhjes se enigmes. Per tu fituar ky grafik jane bere 4 testime me 4 enigma te ndryshme, me te cilat eshte testuar koha e zgjidhjes se enigmes prej secilit algoritem. Nga kjo shohim se algoritmi A\*(Astar) e ka bere zgjidhjen me te shpejt te enigmave. Algoritmet BFS dhe DFS kane performuar perafersisht njejte, kurse Algoritmi ID ne 2 raste koha me te cilen e ka bere zgjidhjen e enigmes ka qene me e gjate.

Figura 9. Levizjet qe ju duhen algoritmeve per te gjetur zgjidhjen e enigmes

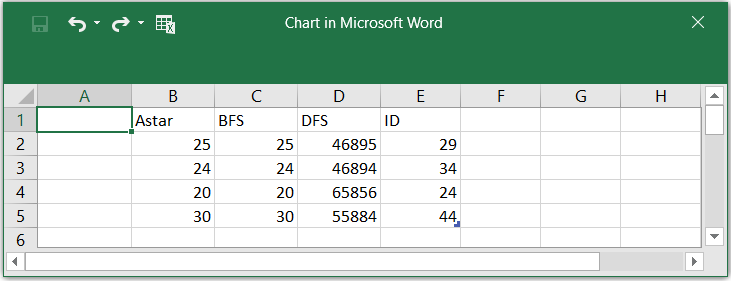


Figura 10. Numri i levizjeve qe eshte dhene si input per te fituar grafin ne figuren 9.

Nga figura 9 dhe figura 10 e shohim qe algoritmet Astar dhe BFS ne 4 raste kane zgjidhur enigmen me numer te njejte te levizjeve, kurse algoritmi ID ne te 4 rastet ka pasur numer me te madh te levizjeve per te zgjidhur algoritmin. Ndersa ajo qka mund te konkludojm eshte se algoritmi DFS ka patur nevoje per shume me shume levizje per te bere zgjidhjen e enigmes.

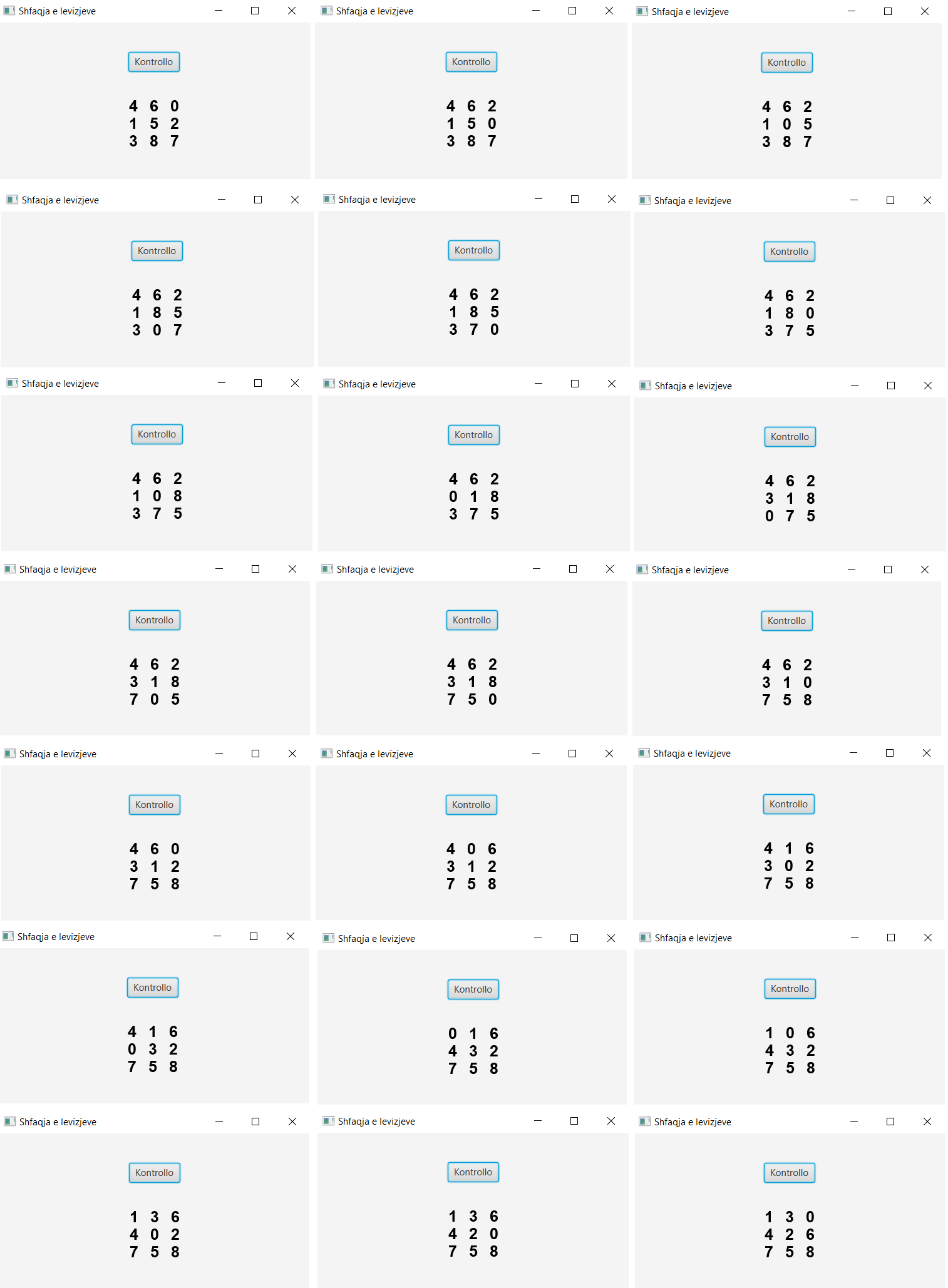
Figura 11. Numri i levizjeve i dhe numri i testimeve kane rritje lineare.

Ne anen e majt te figures 11 jane te paraqitur numri i levizjeve kurse ne anen e poshtme jane te paraqitur numri i testimeve.

Figura 12. Grafi i ndryshimi i kohes.

Nga grafiku ne figuren 12 e shohim se pas 23 testimeve koha per zgjidhjen e enigmes se njejt nga algoritmi i njejte ka tendence qe te ulet. Karakteristika kryesore qe verehet eshte se nga testimi i pare dhe testimi i dyte kemi reninen me te madhe te kohes per zgjidhjen e enigmes.

### **4.2.3. Levizja e pozitave**



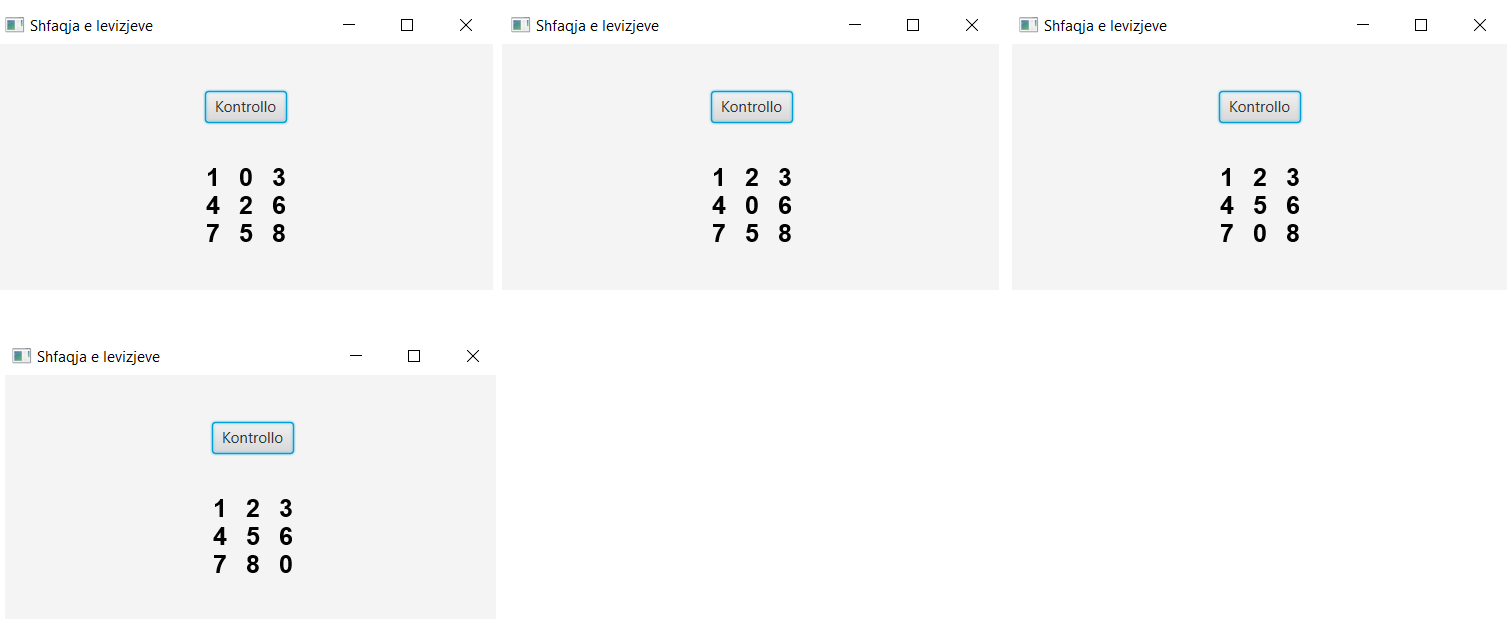


Figura 13. Duke klikuar ne butonin Kontrollo shohim hapat e zgjidhjes se enigmes.

### **4.2.4. Validimet**

Programi nuk mund te ekzekutohet si duhet nese:

1. nuk plotesohen te gjitha fushat ose ndonje prej tyre,
2. nese fushat plotesohen me te dhenat te gabuara.

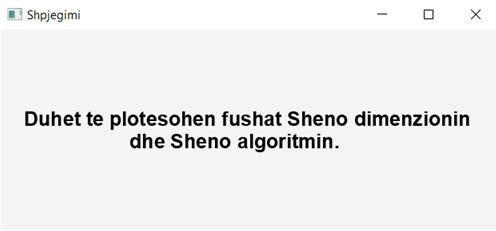


Figura 14. Pamja qe na shfaqet nese nuk plotesohen fusha e dimenzionit ose e algoritmit.

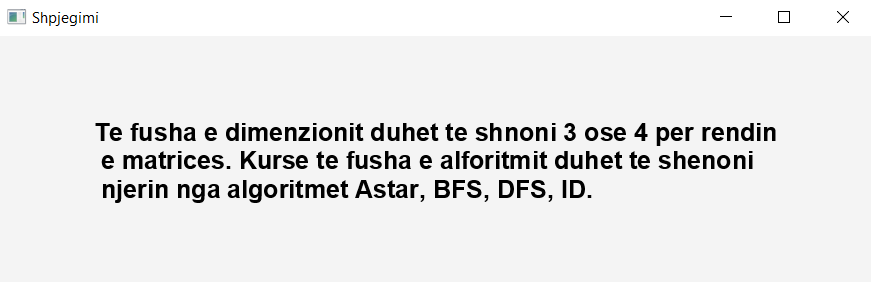


Figura 15. Pamja qe na shfaqet nese plotesohen fusha e dimenzionit dhe algoritmit me te dhena te gabuara.



Figura 16. Pamja qe na shfaqet nese nuk plotesohen te gjitha fushat e matrices.

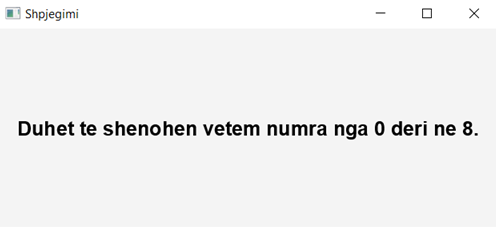


Figura 17. Pamja qe na shfaqet nese fushat e matrices plotesohen me te dhena te gabuara.

# **Referencat**

1. Kodi burimor(pjesa e algoritmit) per zgjedhjen e enigmes eshte marre nga ky burim:

https://github.com/abalyani/Sliding-Puzzle-Solver