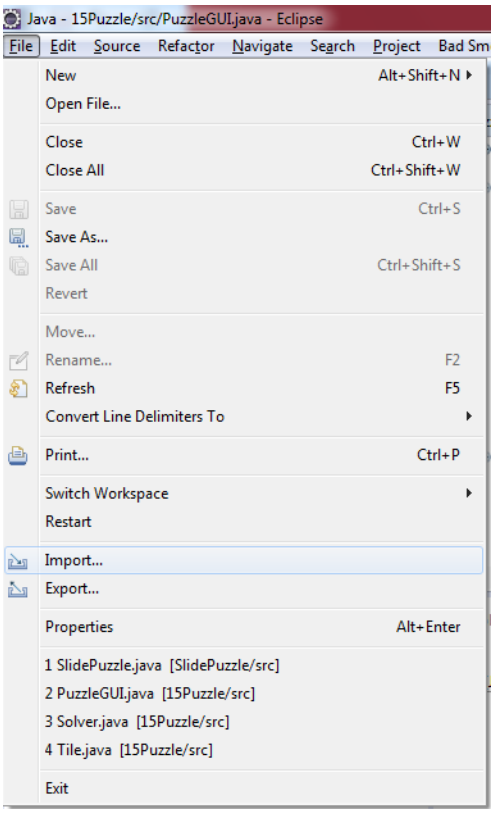
15 Puzzle

1. Scope and purpose:

The scope of this document is defined within the associated implementation of the 15 Puzzle game. The purpose of this document is to provide the overview and details of the implementation. The document is generated to provide better understanding of the implementation and used for evaluation of the Course 7336 Advanced Software Engineering.

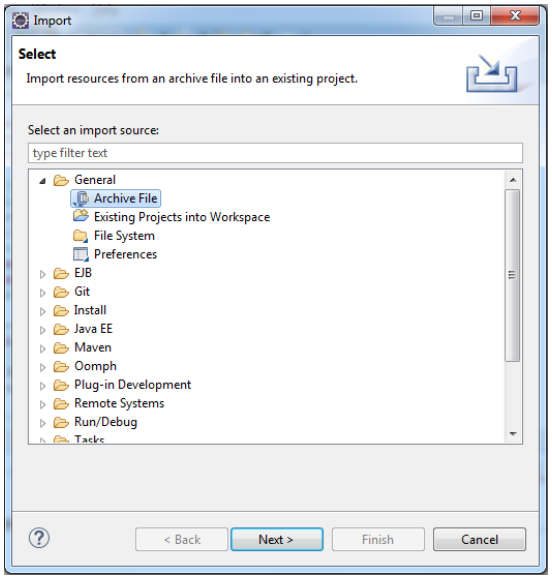
2. Introduction:

15 Puzzle game is developed with one of the famous programming language Java. The 15 puzzle project shows a grid of tiles which must be structured in ascending order with last tile as empty tile. This project requires Eclipse IDE to run. The Project can be run using the eclipse IDE, after successful import using ***File -> Import….***



There are multiple options to import eclipse project depending on the project.

1. ***General -> Archieve File***: This option can be used to import eclipse project from an archive file. Click ***Next*** to select the eclipse project to import.

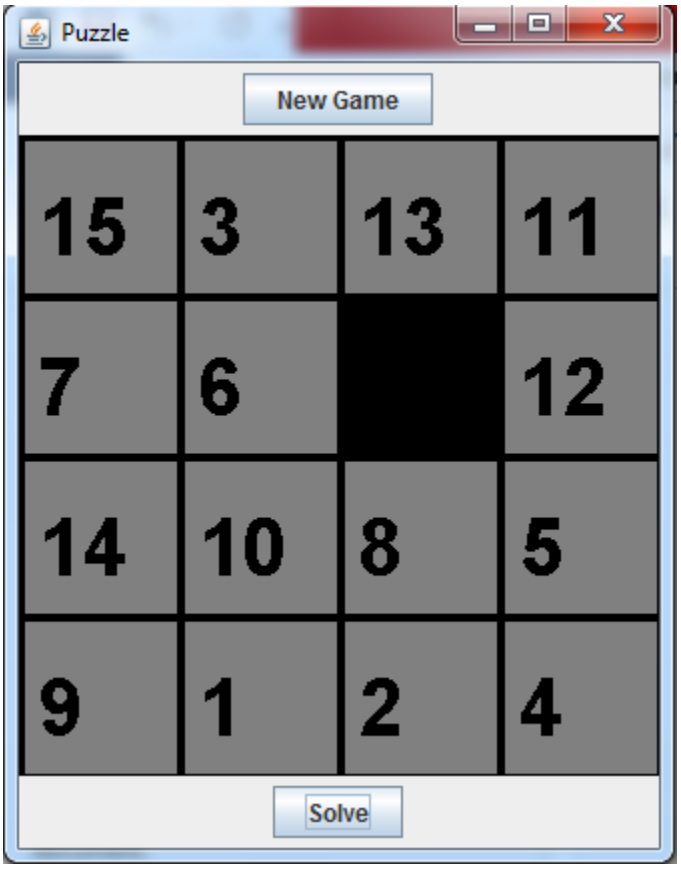


2. ***General -> Existing Projects into Workspace***: This option can be used to choose existing project to workspace. Click ***Next*** to select root directory. Use ***Select All*** option and click ***Next*** to import eclipse project.

3. ***General -> File System***: This option can be used to directly import project from file system of the computer. Select ***Browse*** to select the ***From directory:*** and click ***Next*** to import eclipse project.

   Once the project is imported successfully, select the project and Right click to ***Run as -> 2. Java application***.

After successfully running the project, Project displays the following GUI shown below:



The user can click on "New game" button to generate new game which is the result of calling shuffeTiles() in PuzzleModel.

The user can click on "Solve" button to generate the possible moves to solve the puzzle grid.

The project is organized with 3 different parts:
a. Standard main (Puzzle) which creates the instance of GUI and calls method to show the GUI,
b. A GUI (PuzzleGUI) made by extending JPanel. This creates an object of the logic of the program (PuzzleModel) and calls its methods.
c. The logic of the 15 puzzle is defined inside the PuzzleModel. This class creates multiple instances of Tiles in order to create a grid of tiles to complete the puzzle. The Empty tile is also declared as tile with null value associated to it.

3. Architecture:

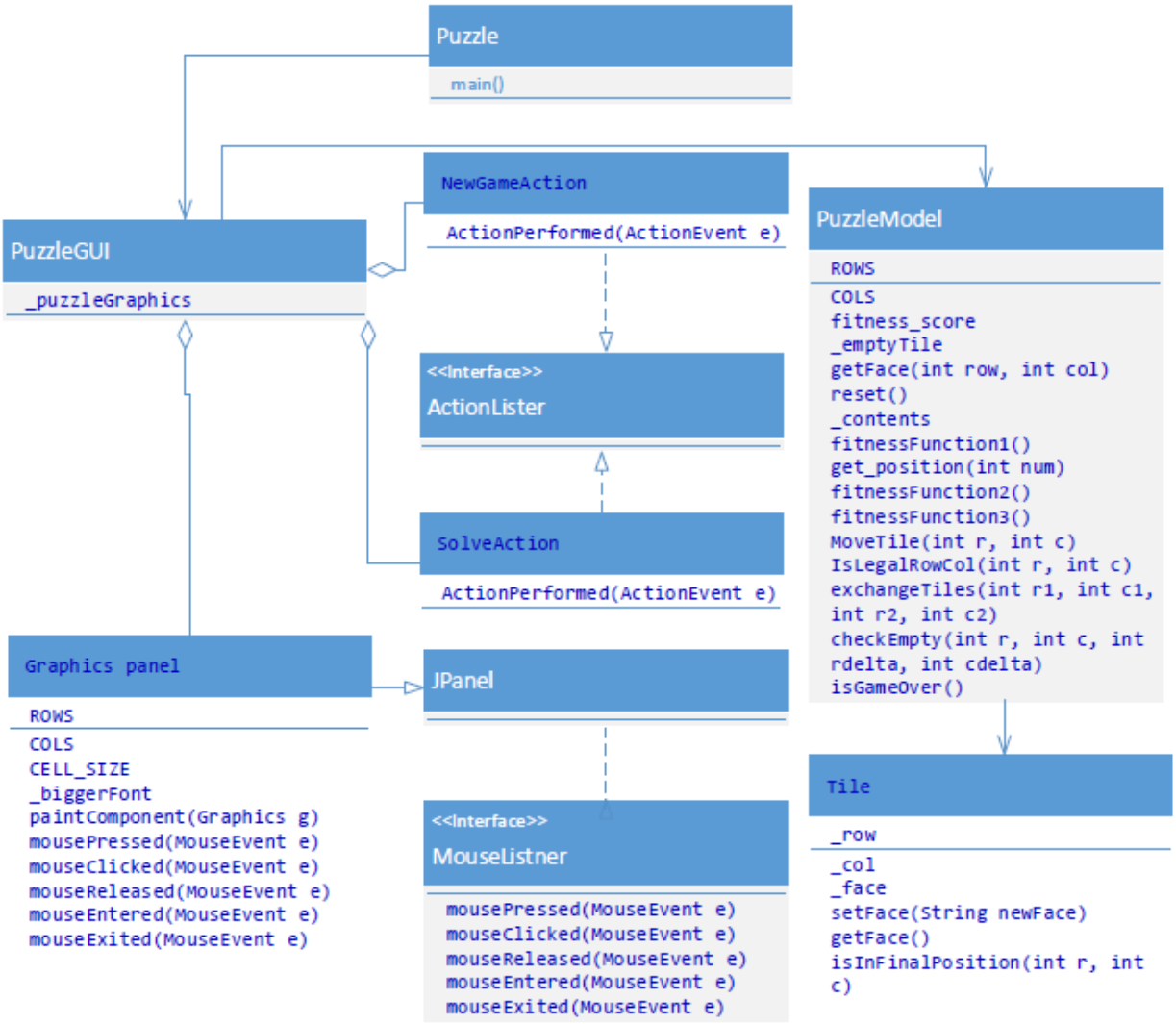
The 15 puzzle project is combination of different outer classes, inner classes and interfaces as follows:

a. Class Puzzle, //class for main method to instantiate PuzzleGUI
b. Class PuzzleGUI, //GUI class for Puzzle
c. Class Graphics Panel, //inner class of PuzzleGUI
d. Class JPanel, //super class for Graphics Panel
e. Interface MouseListner, // to trigger the action for mouse
f. Class SolveAction, //inner class of PuzzleGUI
g. Class NewGameAction, //inner class of PuzzleGUI
h. PuzzleModel, // class to implement logic of the puzzle
i. Tile, // class to create multiple instances to make puzzle grid

a. Class Puzzle: This class contains the main method which is the starting point of the project and creates an instance of JFrame which adds supports for the Swing component architecture.
b. **PuzzleGUI:** The 15 puzzle project is run using Puzzle main method and creates an instance of PuzzleGUI which extends JPanel and includes three inner class: Graphics Panel, NewGameAction, SolveAction.

c. **Class Graphics Panel:** This class includes the rows, columns variables for the GUI of tile, font variable, paintComponent, mousePressed, mouseClicked, mouseReleased, mouseEntered, mouseExited methods.

d. **Class JPanel:** JPanel contains all the parts of GUI like setLayout(<JButton object>), add (inherited from Java.awt.component), setLayout(<BorderLayout object>), setPreferredSize(<java.awt.Dimension.Dimension(int width, int height>), addMouseListner(), setBackground(<MouseListner instance>), setBackground(<color instance>), paintComponents(<Graphics instance>). This class also uses the method of graphics like setColor(<color object>), setFont(<Font object>), drawstring(<String object>,< x- coordinate>,<y-coordinate>), FillRect(<x-coordiante>, <y-coordiante>, <height>, <width>). This class also uses the Java.awt.Components class to use method like repaint to call to paint or update method. This class also includes Java.awt.Toolkit.getDefaultToolkit to use beep() method for invalid move.



e. **Interface MouseListner:** This interface contains all the methods related to mouseEvents like mousePressed, mouseClicked, mouseReleased, mouseEntered, mouseExited with parameters mouseEvents. After implementing this interface, class Graphics panel implements all these methods but only mousePressed is implemented to use for this project not others.

f. **Class NewGameAction:** This class is used to call the actionPerformed method with mousePressed event on New Game button. This class is inner class of the PuzzleGUI and uses the instances of PuzzleGUI. It includes the method actionPerformed which calls two other methods:
reset();
repaint();

g. **Class SolveAction:** This class is used to call the actionPerformed method with mousePressed event on Solve button. This class is inner class of the PuzzleGUI and uses the instances of PuzzleGUI. It includes the method actionPerformed which executes the solve algorithms depicted in fitnessFunction1, fitnessFunction2 and fitnessFunction3. It provides the possible moves on console to solve the puzzle.

h. **Class PuzzleModel:** This class is used to provide the program logic associated with the project. It includes various methods and variables:
ROWS, COLS, fitness_score, _contents, _emptyTile, getFace(), reset(), get_position(), fitnessFunction1(), fitnessFunctin2(), fitnessFunction3(), moveTile(), isLegalMove(), exchangeTiles(), checkEmpty(), isGameOver()

ROWS & **COLS:** These are the int return variables for rows and columns of the tiles and assigned with final value =4.
fitness_score: This is the float return type fitness_score variable which gets computed with fitnessFunction1, fitnessFunction2 and fitnessFunction3 methods.
_contents: This is the Tile return type 2D array based on rows and columns and used to get different new instances to generate grid of Tiles.
_emptyTile: This is the Tile return type 2D array based on rows and columns and used to get new instance for empty tile only and associated with last array element index.
getFace(): This is the String return type method which returns the corresponding value of the tile according to the 2D array index provides in parameter.
reset(): This is the method with void return type which sets the face of each tile and shuffles the tiles with random variable.
get_position(): This is the int[] return type method which returns integer array to hold value of row and column. It takes int variable number to check its position in the grid.
fitnessFunction1(), fitnessFunction2(), fitnessFunction3: These methods with float return type and puzzleModel parameter calculates the value of fitness_score to find the best possible move. The maximum fitness_score can be 1 which is corresponding to all misplaced tiles and minimum fitness_score can be 0 which is corresponding to all correctly placed tiles. fitnessFunction1 checks the face of the tile in shuffled grid against solved grid based on each tile. fitnessFunction2 finds the distance of the tile from its original position. fitnessFunction3 is the average of both fitnessFunction1 and fitnessFunction2.
moveTile(): This method returns boolean true if the move is valid, so it checks the emptytile if it finds near to the number associated with row and column otherwise it returns false
checkEmpty(): This method returns boolean true if its neighbor is empty tile, otherwise it returns false.
isLegalMove(): This method returns Boolean true if the move associated is legal and can be applied.
exchangeTiles(): This method returns void and shuffles the position of tiles.
isGameOver(): This method checks all tiles' positions whether the game is over.

i. **Class Tile:** This class contains three variables, constructor and methods as follows:
_row, _col, _face: this variables define the tile's position and face with _row, _col and _face.
Tile(): this constructor is used to instantiate new tiles with above three parameters.
getFace() & setFace(): these getter and setter method are used to set the value of face and get the value of face.

4. **Testing:** The Junit class TestPuzzle tests the methods of the PuzzleModel class whereas the testing of the GUI class is performed manually. Manual test cases includes the color, buttons availability of the puzzle.
Manual test cases:
Test case 1: Verify that there are two buttons available on the puzzle GUI. (PASS/FAIL)?
Test case 2: Verify that one button is located on North of the GUI and other button is located on South of GUI (PASS/FAIL)?
Test scenario: If Test case 2 passes then verify Test case 3 onwards otherwise leave the test case.
Test case 3: Verify that the button located on North direction of the GUI has text "New Game". (PASS/FAIL)?
Test case 4: Verify that the button located on South direction of the GUI has text "Solve". (PASS/FAIL)?
Test case 5: Verify that the numbers on tiles are shown correctly. (PASS/FAIL)?
Test case 6: Verify that the numbers are shuffled. (PASS/FAIL)?
Test case 7: Verify that the numbers from 1 to 15 can be seen. (PASS/FAIL)?
Test case 8: Verify that the numbers on Empty tile is not seen. (PASS/FAIL)?
Test case 9: Verify that the numbers are shuffled after mousePressed on New Game button. (PASS/FAIL)?