CS 342: PROJECT 3
Rush Hour

Bryan Spahr
George Saldaña

TABLE OF CONTENTS

INTRODUCTION

1.1. Purpose

This design document describes the architecture and system design of this project, which is Rush Hour

1.2. Scope

This project is a GUI interface game, written in Java, that contains five classes

1.3. Reference Material

Various Java SWING web sources

2. SYSTEM OVERVIEW

The goal of this game is to slide the blocks in their respective directions one at a time until the red block reaches the right side of the board. Ten levels are included, after which the game will be finished.

3. SYSTEM ARCHITECTURE

3.1. Architectural Design / Overview of Classes

Button: The Button class is the basic unit of this game. It contains necessary info like its coordinate on the grid, its position in a larger piece, whether or not it is selected, and other attributes such as its tag and color.

Solutions: This class will take in a snapshot of the initial state of the current level and find a solution if there exists a path to it. This is done by creating a tree structure of all the possible unique moves from each state of the game. This class used the Node class to create the structure. Each node of the tree is an instance of the Node class. The use of a hash map to store all unique states (snapshots) was used prevent getting to the same state more than once thus preventing the loops when performing the breadth first search to find the solution of the map. Once the tree structure is set the use of a breadth first search was used to find a shortest path to a solution.

Grid: The Grid class is the heart of the program that contains most of the functionality of the program, ranging from displaying the formated game board, and performing all necessary Rush Hour game logic, like selecting pieces and performing moves. Variables include JSWING elements (JFrame, JLabel, JPanel), an array of Button objects, size of board, lists of type String to keep track of snapshots and piece locations, and various integers to keep track of game state (such as counters). Included are methods to initialize the board, show hints, show the complete solution, and reset the board.

Main: The main class being necessary to run the program simply creates a game object and sets frame to say “Rush Hour”.

Node: This class was used in conjunction with the solutions class to create the tree structure used to find the solution to current level.

3.2. Design Rationale

The goal for the source code was to separate it into as many classes as possible, to make it more concise and easier to understand. The Main class was created to isolate the Main method that starts the program. The Button class was created to implement JButton, but contain additional functions and global variables to interact with the board. The Solutions class contains the algorithm for solving the levels of there exists a solution. This class uses the Node class to find the solution. The Grid class contains everything else in the game, including GUI elements and various Listeners.

4. DATA DESIGN

4.1. Data Description

Most collections of objects in this codebase are stored in ArrayLists, while a few are stored in HashMaps. Also using queues for the breadth first search algorithm.

4.2. Data Dictionary

Button:

```
public Button(int i, int j, int h, int w)

public void setAttributes(String tag, int h, int w, int pos, Color c,
boolean blank)

public int getI()

public void setI(int i)

public int getJ()

public void setJ(int j)

public boolean isSelected()

public void setSelected(boolean selected)

public Color getC()

public void setC(Color c)

public boolean isBlank()

public void setBlank(boolean blank)

public int getPosition()

public void setPosition(int position)
```

```
public String getTag()

public void setTag(String tag)

public int getH()

public void setH(int h)

public int getW()

public void setW(int w)

public void setAllMoves(int [] Upmoves, int [] Dnmoves, int [] Ltmoves,
int [] Rtmoves, int arraySize)

public void setUMoves(int[] Upmoves, int arraySize)

public void setDMoves(int[] Dnmoves, int arraySize)

public void setLMoves(int[] Ltmoves, int arraySize)

public void setRMoves(int[] Rtmoves, int arraySize)

public int [] getUMoves()

public int [] getDMoves()

public int [] getLMoves()

public int [] getRMoves()

public void printMoves()
```

Main:

```
public static void main(String[] args)
```

Solutions:

```
public Solutions(String InComingSnapshot, Button [][] incomingButtonGrid,
int GridSize)

public ArrayList<String> getSolution()

public ArrayList<String> printSolutionBackwards(Node N)
```

```
public void printFromString(String SnapShot)

public void setChildrenList(Node parentNode, Button[][] Grid)

private boolean gameWonSnap(String SnapShot)

public Button[][] copyGrid(Button [][] Grid)

public String getSnapshot(Button [][] Grid)

public void setAllMovesLists(Button [][] Grid, HashMap BMap)

public HashMap setGridHashMap(String CurrentSnapshot, Button [][] Grid)

public void printGrid(Button [][] Grid)

public void printMap(HashMap HMap)
```

Grid:

```
public Grid(String windowLabel)

public void create_menu()

public void init()

public void countFiles()

public void readFile()

public void createRequiredButtons()

public void showHint()

public void showSoln()

public List<String> parseSnapshot(String s)

public void createColors()

public void performMove(int destI, int destJ)

public void selectButton(Button b, boolean val)

public void endLevel(String message, boolean nextLvl)

public void mouseClicked(MouseEvent e)
```

```
        private boolean gameWonSnap(String SnapShot)

        public String getSnapshot()

        public List<String> getSolutionPath()

        public void printGrid()

        public class snapChar

        public class TagColor

        public class repaintTask extends TimerTask

Node:
        Node(String inputSnapshot)

        public void setParentNode( Node parent)
```

5. COMPONENT DESIGN

Full API in Javadoc formatted html attached to web page.

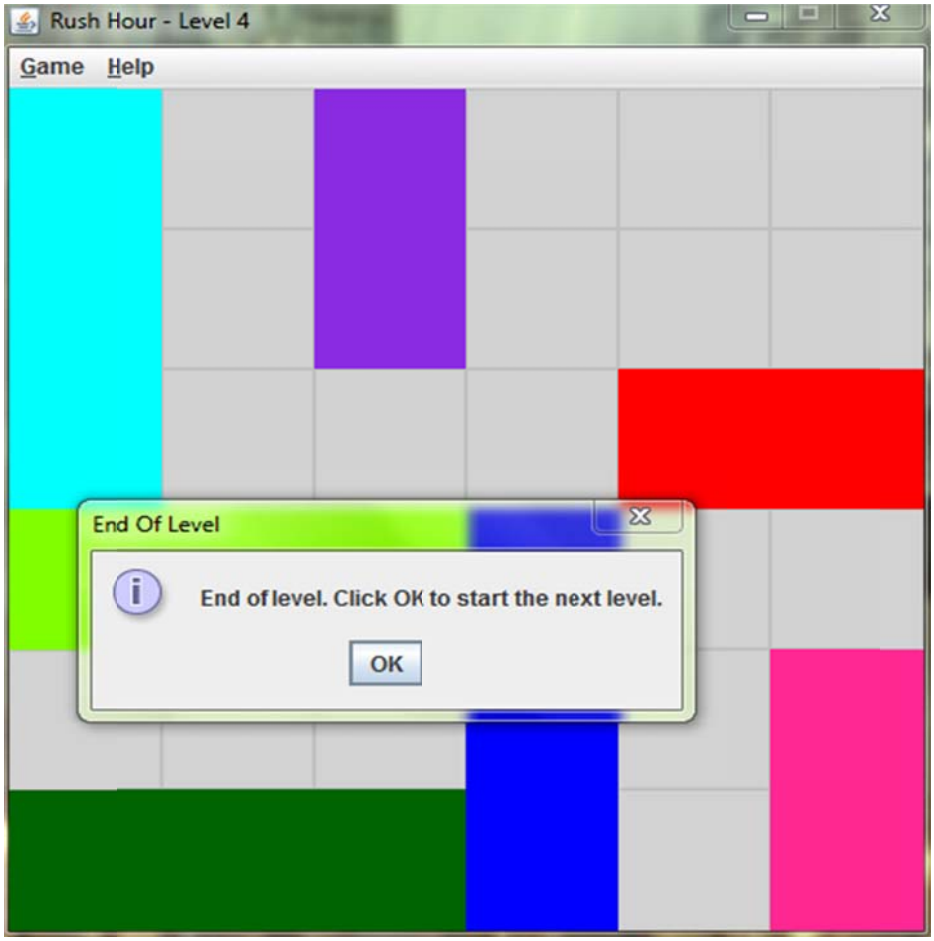
6. HUMAN INTERFACE DESIGN

6.1. Overview of User Interface

Gameplay is as described in Section 2. The program displays a grid of blank tiles with slidable blocks places in predetermined locations on said grid. The window contains various dropdown menus, which include: Game and Help.
Game includes:
eXit - this will end the game and exit.

Help includes:
Help - gives the player instructions on how to play the game.
About - provides names of the game developers.
Show Hint - makes the next move in the shortest path to a solution.
Show Solution - displays an awesome animation of the step by step process to the solution of the level.
Reset board - resets the board pieces to their original positions.

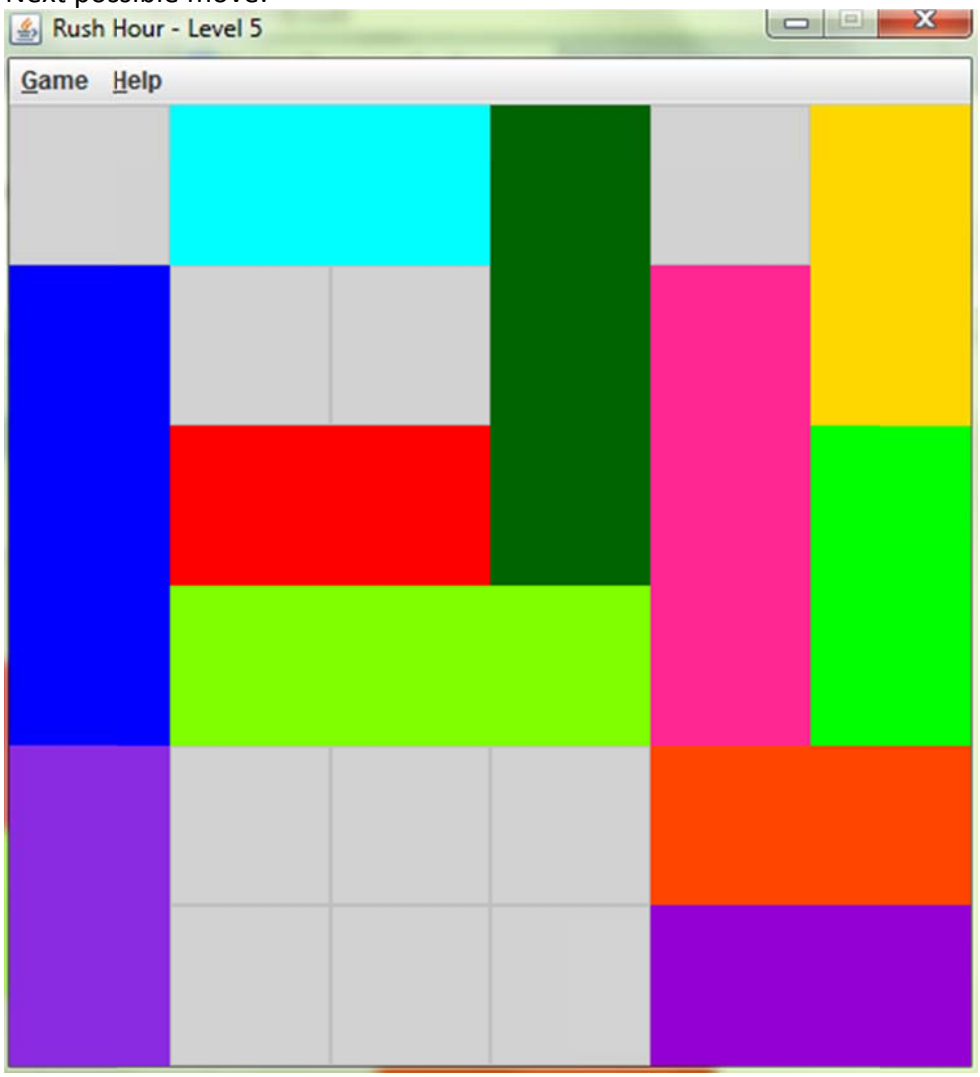
6.2. Screenshots

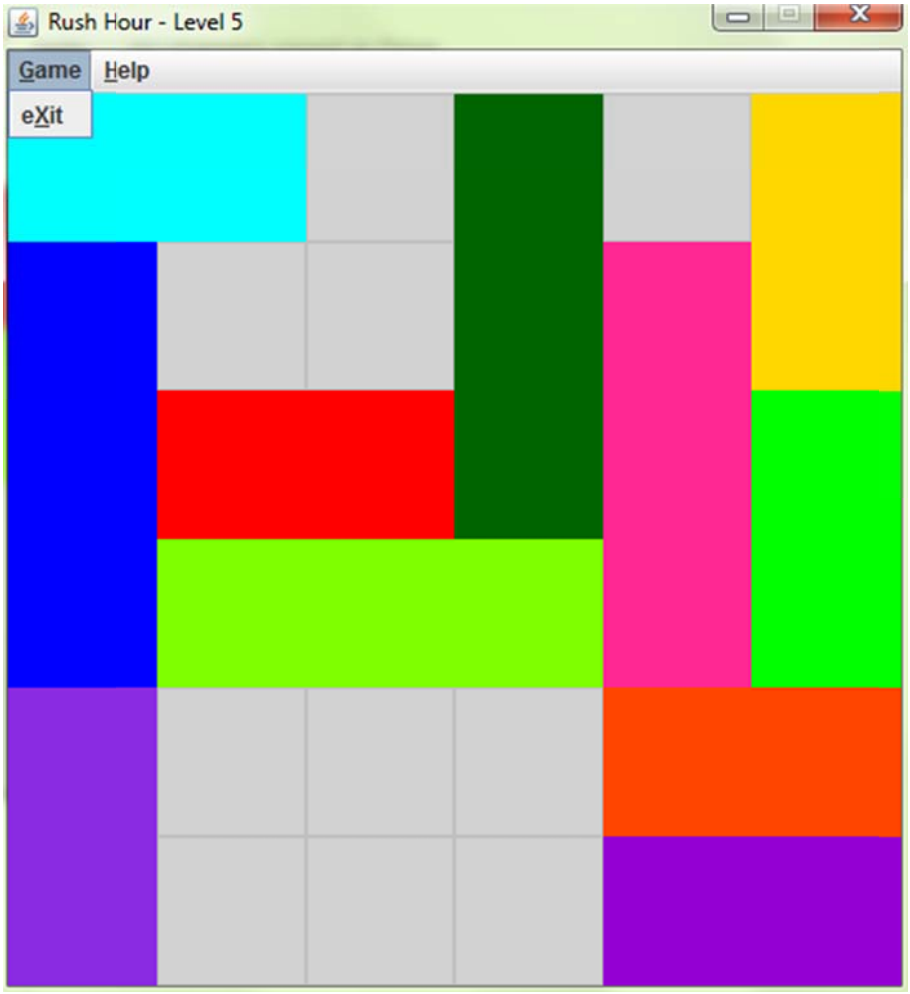


When the user has the goal piece all the way to the right most column she/he is informed the puzzle is solved and prompts the user to move to the next level.

Shows the user the next possible move.

Next possible move:





Exit the game.

7. **Requirements Matrix:**

Requirement	**Satisfied By**
Java Swing library	Grid
Algorithm to solve level in least amount of moves	Solutions
Multiple source code files	Grid, Buttons, Nodes, Solutions
Inform the user when the puzzle is solved or unsolvable	Grid
2x1,3x1, 1x2, 1x3 pieces in each puzzle level	.data level files
Create al least 10 puzzle levels	.data level files
Game menu and Help menu	Grid
Exit button	Grid
Reset button	Grid
Help button	Grid
About button	Grid
Show a hint	Grid
Create animation solving the level in least amount of moves	Grid