Java Game Suite Project Design

Revision 10

CMSC 495 6382

September 11, 2021

Group Charlie

Sherry Funches, Oyewole Sanusi, Janee’ Jones, Wayne Mack, Jeffrey McGlinn

## Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Revision Number** | **Date** | **Description** | **Name** |
| 1 | 9/7 | Creation of Google Doc, outline | Oyewole |
| 2 | 9/11 | Creation of Sequence Diagrams | Jeff |
| 3 | 9/11 | Create Snakes UML diagram and pseudocode | Oyewole |
| 4 | 9/11 | Create Maze and Main menu pseudocode | Wayne |
| 5 | 9/12 | Create Slider UML diagram and pseudocode | Jeff |
| 6 | 9/12 | Create Word search UML diagram and pseudocode | Sherry |
| 7 | 9/12 | Create Sudoku Class Diagram and Code | Janee’ |
| 8 | 9/12 | Create Maze and Main Menu UML | Wayne |
| 9 | 9/12 | Combine all documents into one. Format unified document. Add table of contents. | Sherry |
| 10 | 9/13 | Fixed typos. Fixed formating errors for word search. Adjusted images. Placed pseudcode for programs on separate pages | Sherry |

## Table of Contents

## I. Class Diagram…………………………………………………………………………………...4

Main Menu………………………………………………………………………………...4

Maze…………………………………………………………………………………….....5

Snakes……………………………………………………………………………………..6

Sudoku…………………………………………………………………………………….7

Word search……………………………………………………………………………….8

Slider puzzle……………………………………………………………………………….9

II. Sequence to Sequence Diagrams……………………………………………………………...10

Start up Scenario………………………………………………………………………....10

Shutdown Scenario……………………………………………………………………....11

Normal Operation Scenario………………………………………………………….......12

Error Handling Scenario 1………………………………………………………….........13

Error Handling Scenario 2………………………………………………………….........14

Error Handling Scenario 3………………………………………………………….........15

III. Pseudocode and Code………………………………………………………………………..17

Main Menu……………………………………………………………………………….17

Maze……………………………………………………………………………………...17

Snakes……………………………………………………………………………………20

Sudoku…………………………………………………………………………………...22

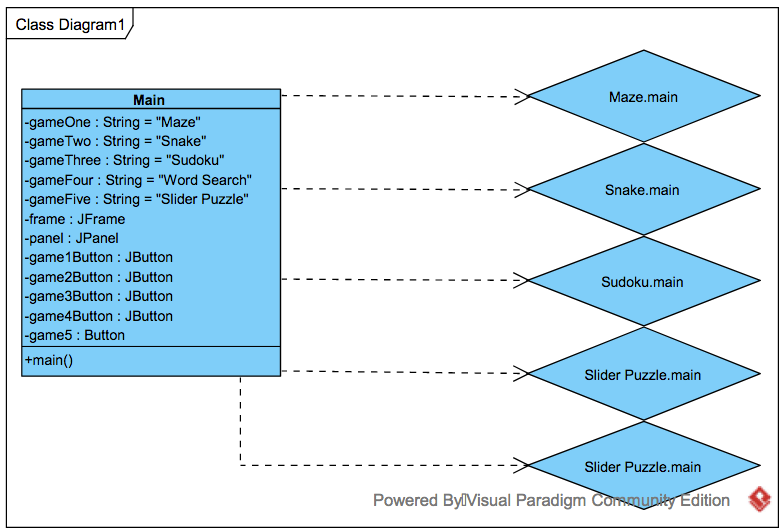
Word search……………………………………………………………………………...34

Slider puzzle……………………………………………………………………………...40

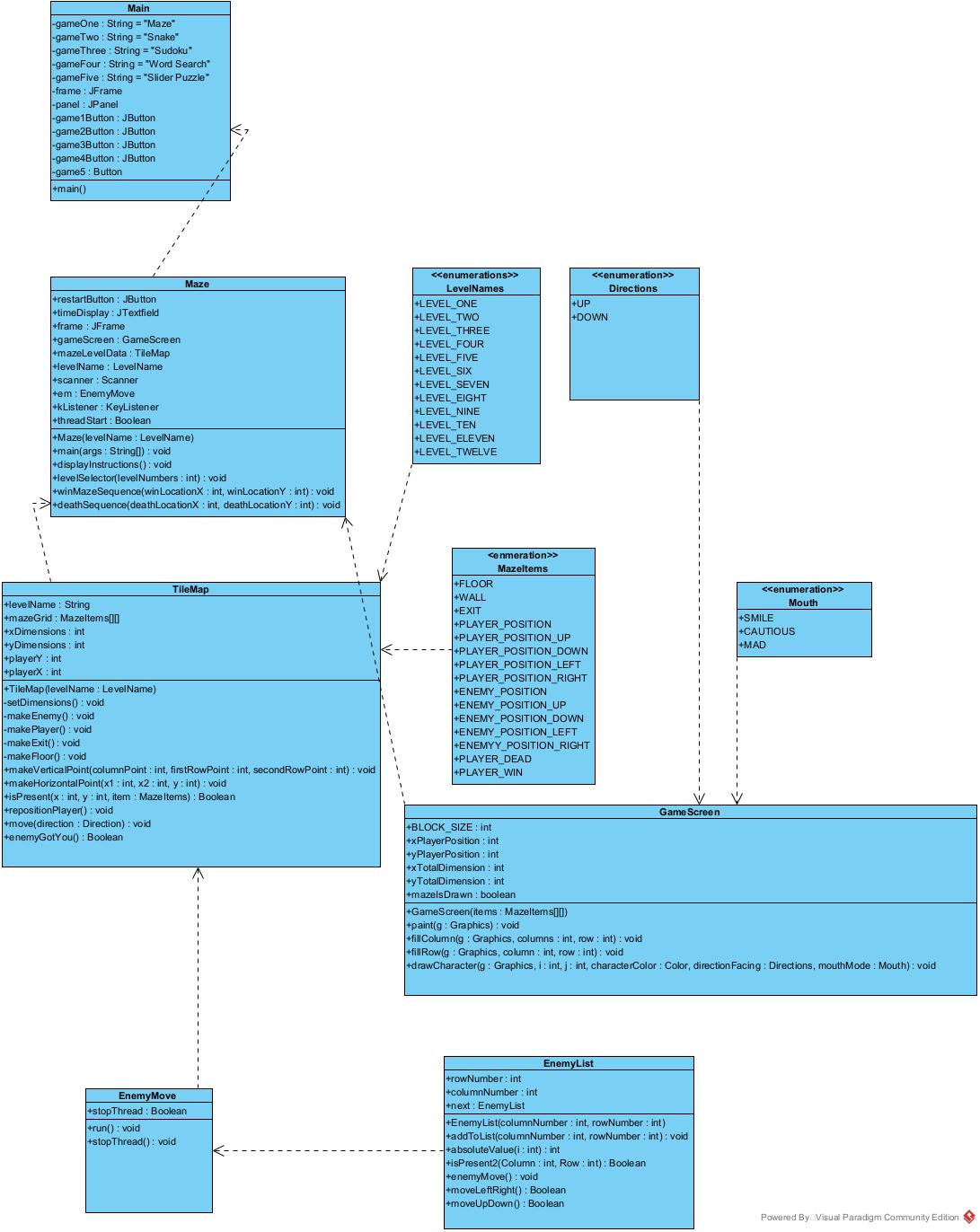
IV. References …………………………………………………………………………………...45

## I. Class Diagram

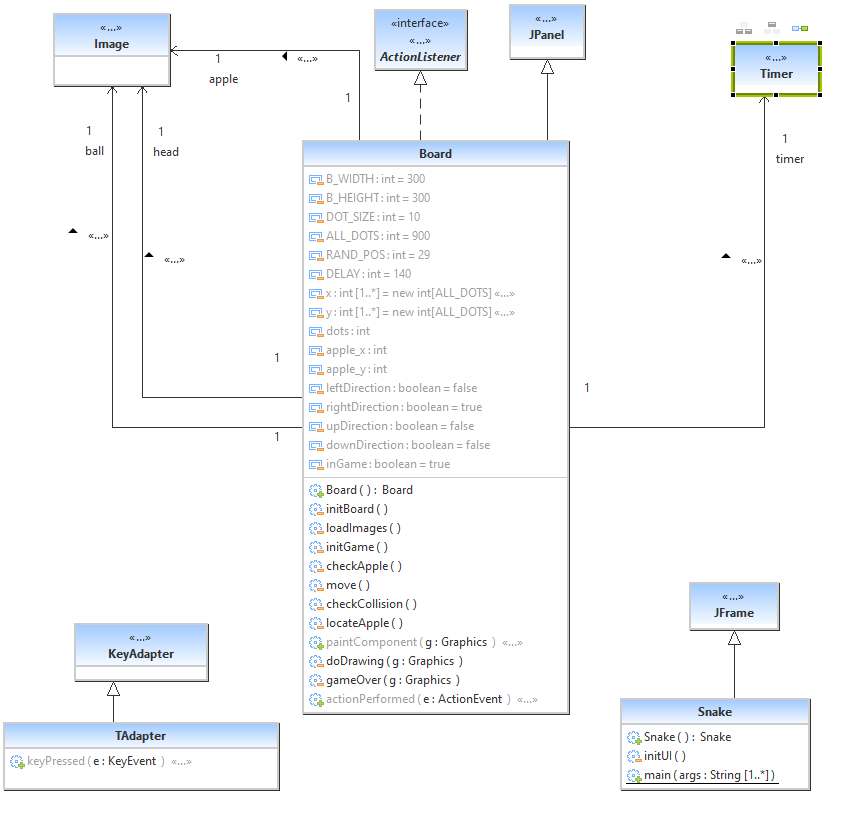
Main Menu



Maze



Snakes

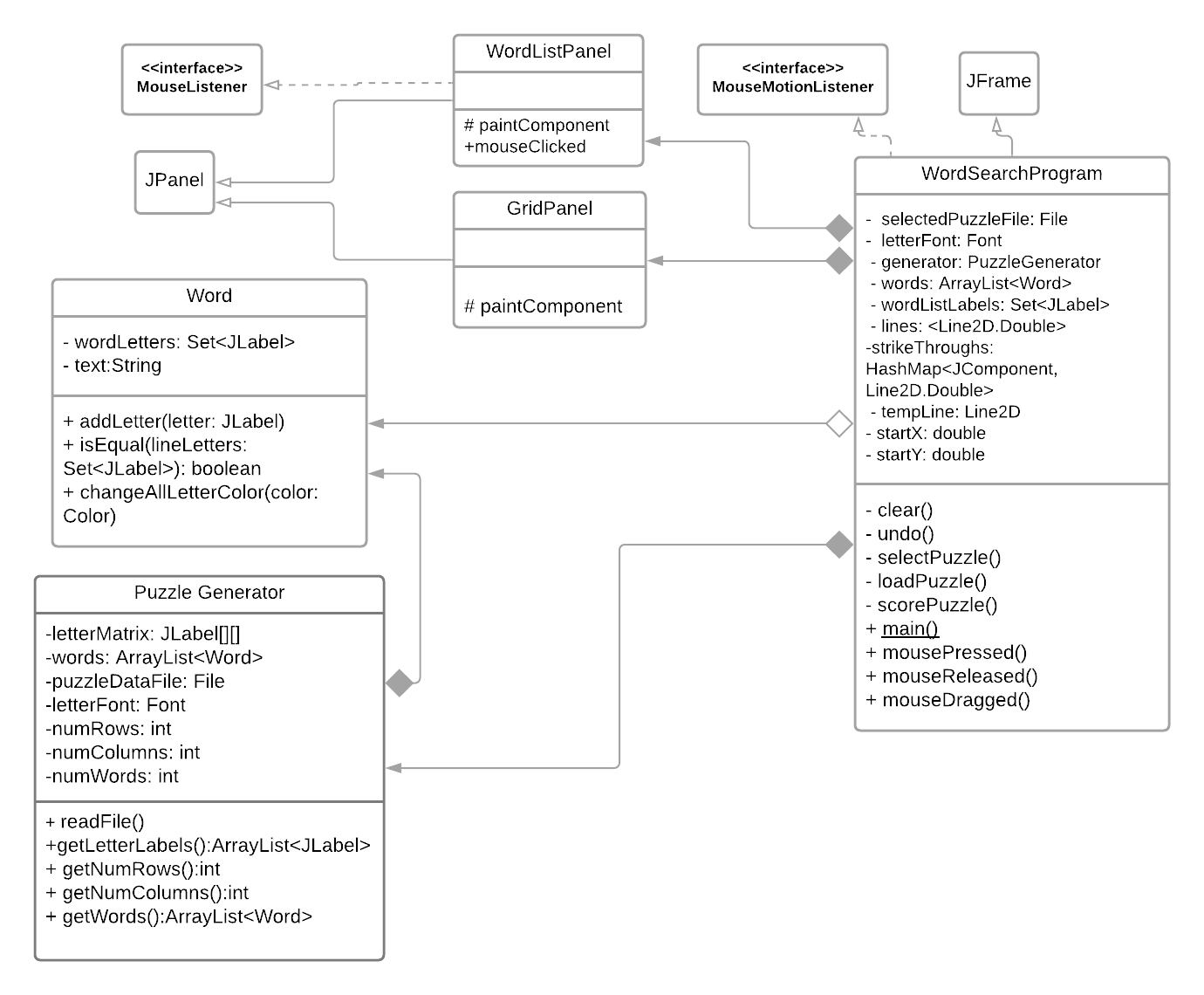


Sudoku

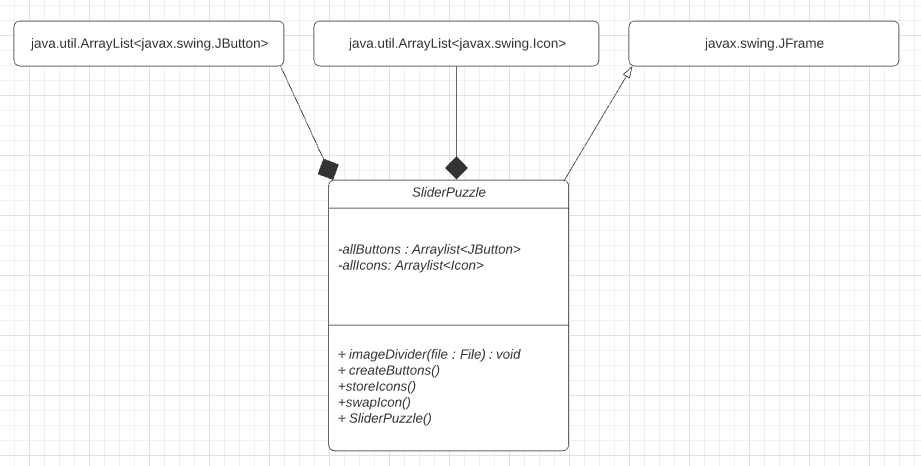
Diagram

Description automatically generated

Word Search



Slider



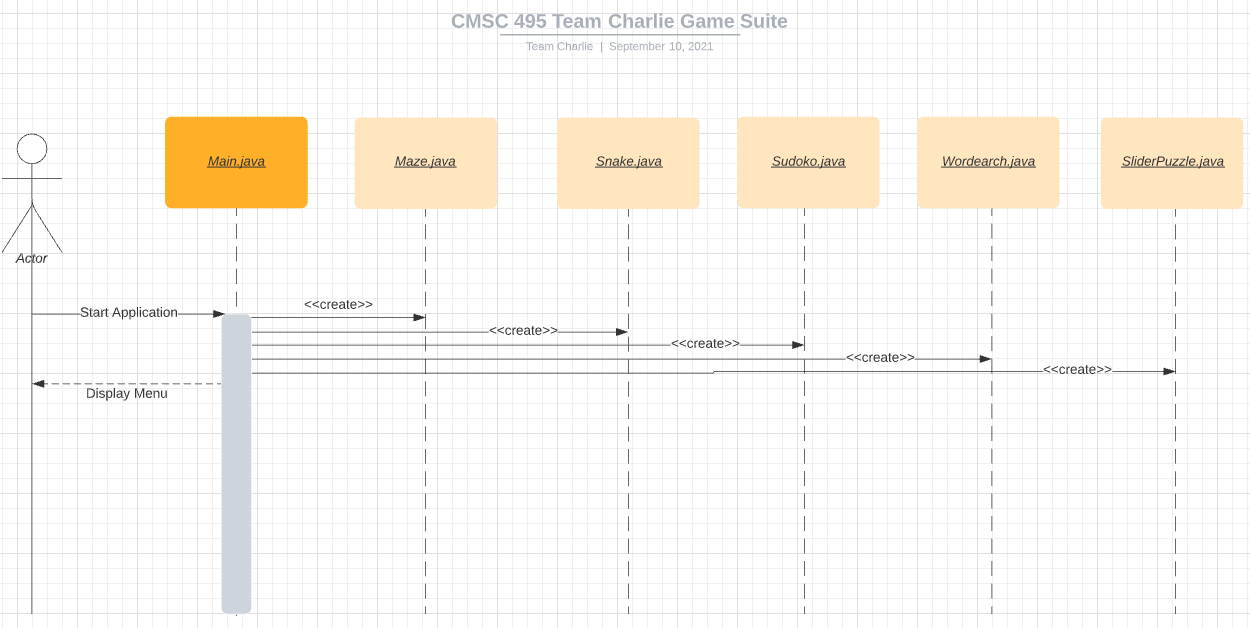
## II. Sequence Diagrams

**Start-up Scenario:**

**Description:** User starts up the application.

**Pre-Condition:** No hardware or system failure that would hinder the application occurs.

**Post-Condition**: Application starts without hardware or system failure.

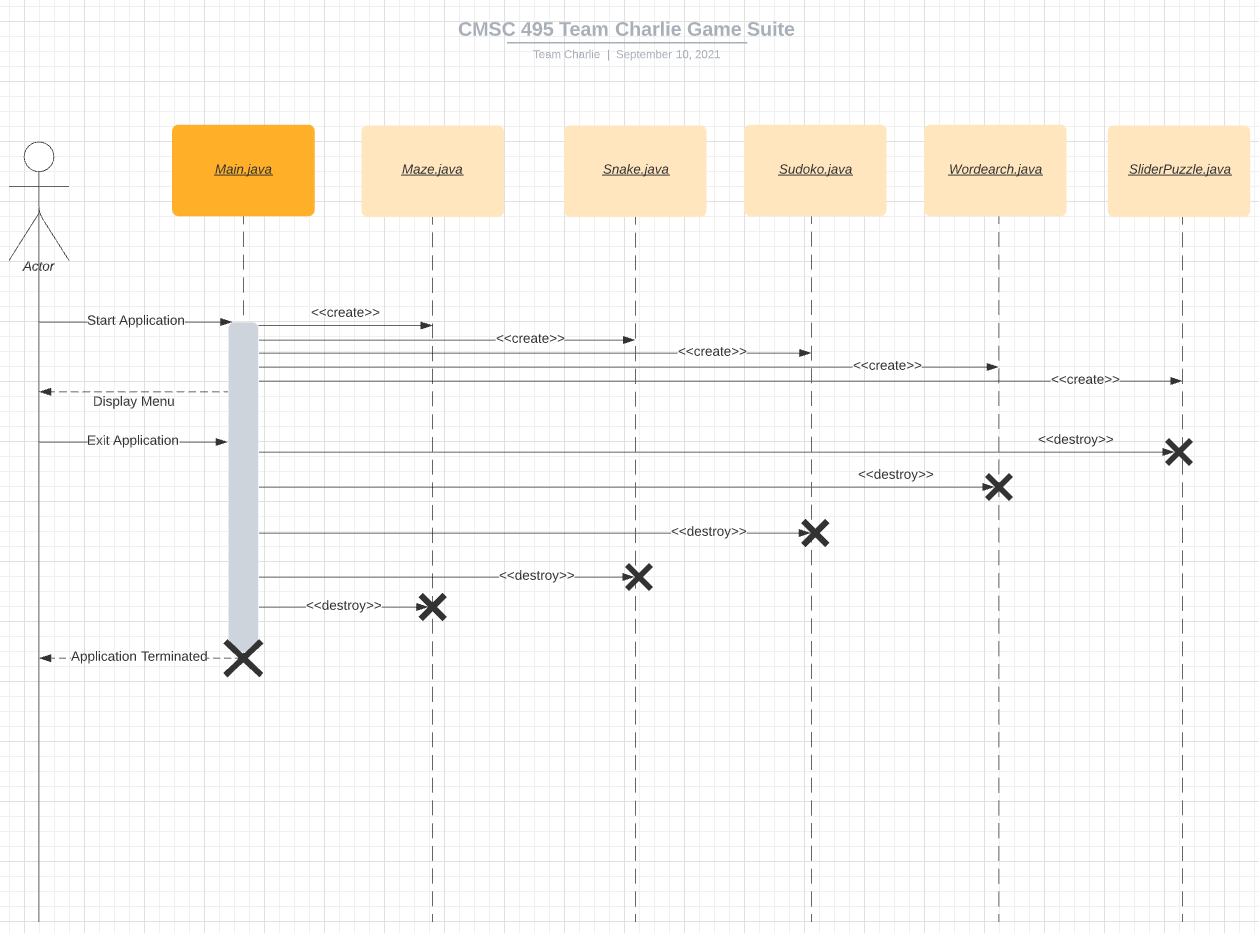


**Shut-down Scenario:**

**Description:** User exits the program.

**Pre-Condition:** Program has been running successfully without crashing.

**Post-Condition**: Program is terminated and all objects are deconstructed.

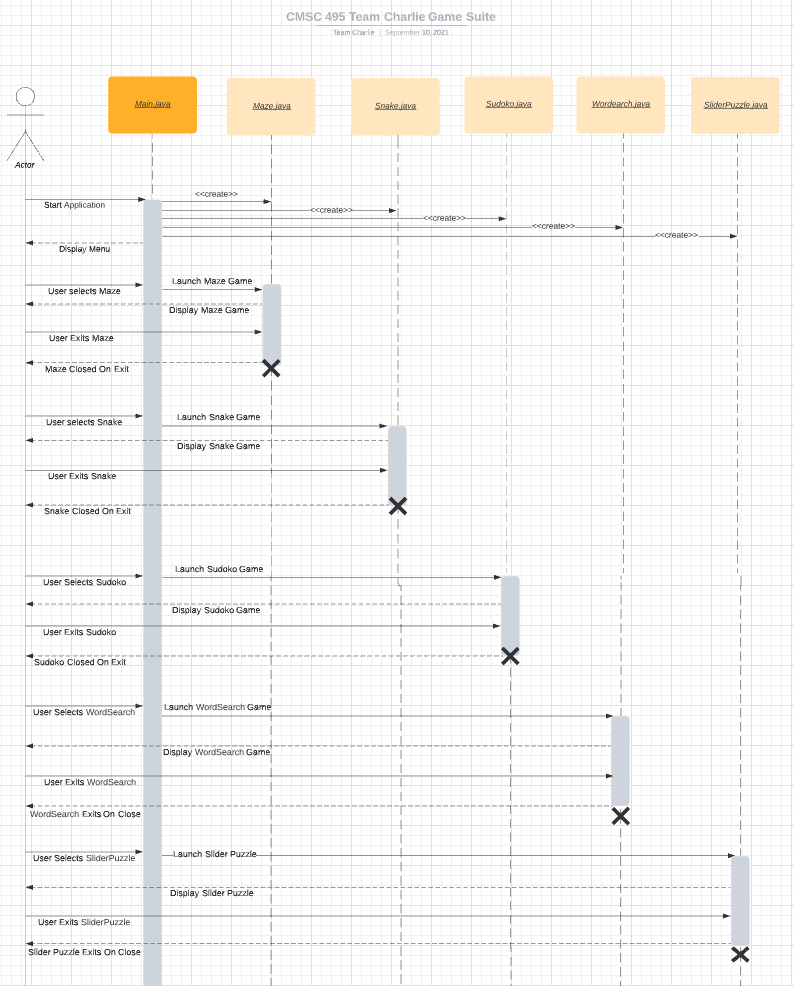


**Normal Operation Scenario:**

**Description:** User starts application and selects games from the menu.

**Pre-Condition:** Program has been running successfully.

**Post-Condition**: The program displays and exits each game that has been selected without hardware or system failure.

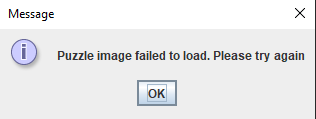
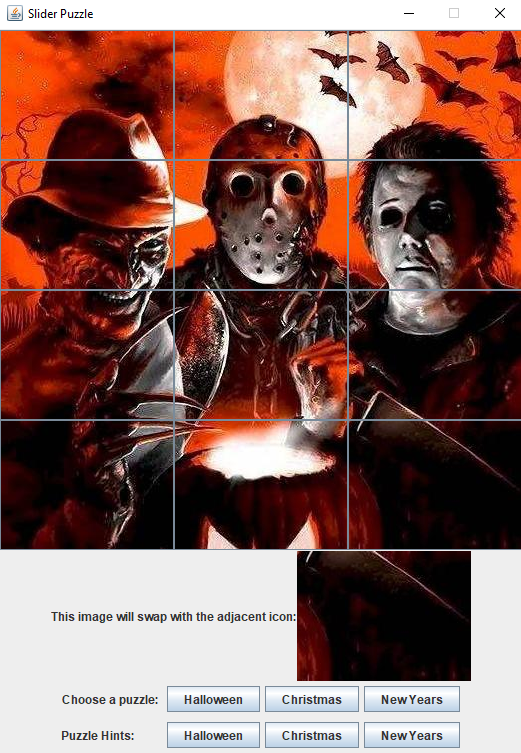


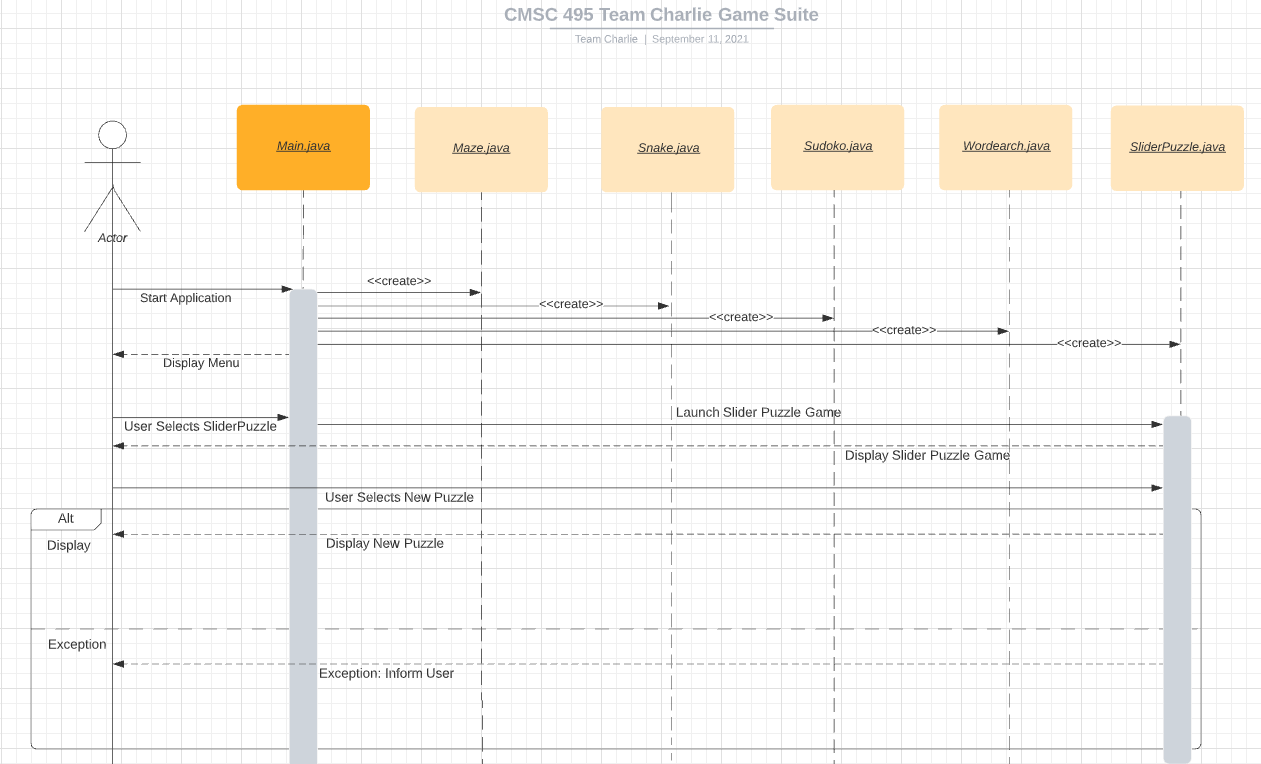
**Error-Handling Scenario 1:**

**Description:** User starts application, launches the slider puzzle game, and selects a new puzzle.

**Pre-Condition:** Program has been running successfully.

**Post-Condition**: If a puzzle image file is not found, handle exception, and inform the user with a message:



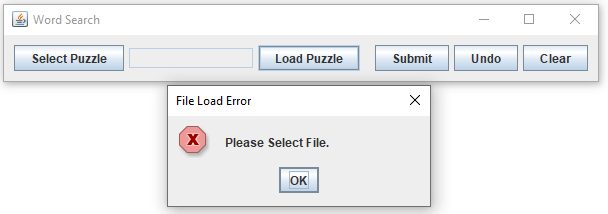


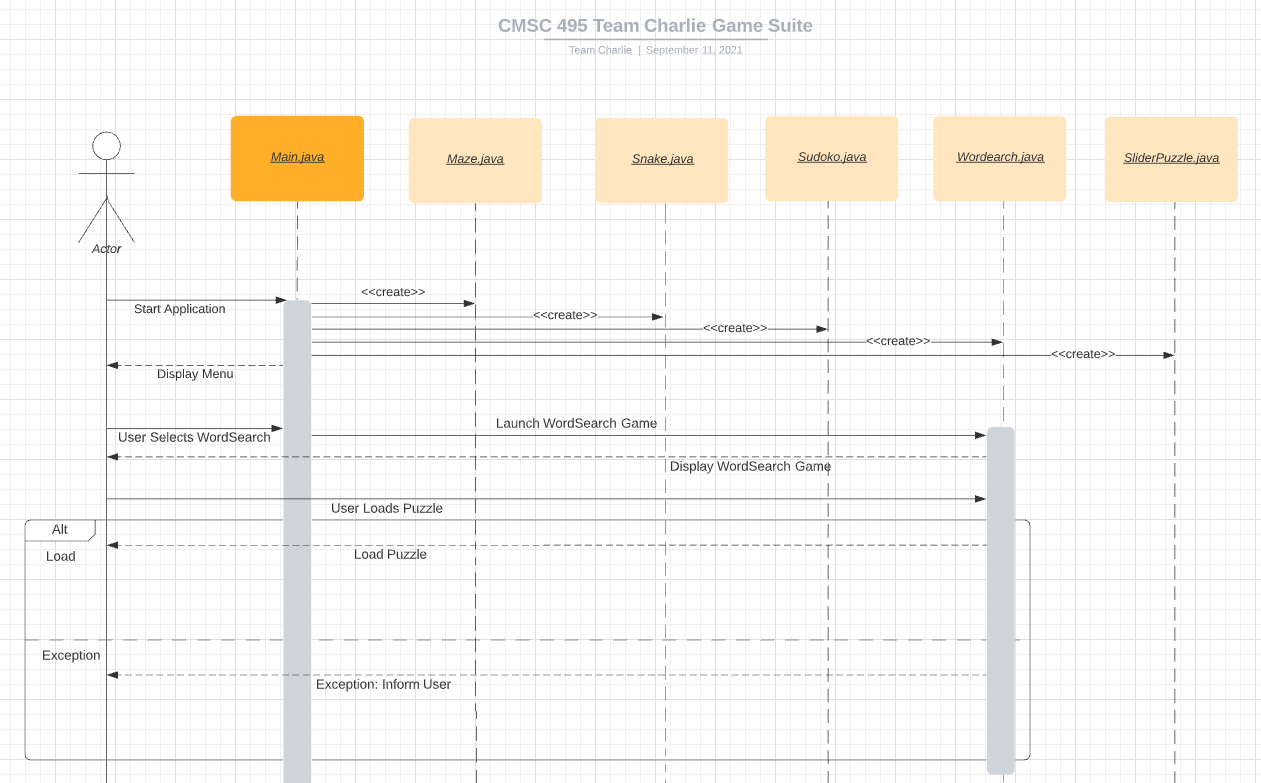
**Error-Handling Scenario 2:**

**Description:** User starts application, launches the word search game, and attempts to load a new puzzle file.

**Pre-Condition:** Program has been running successfully.

**Post-Condition**: If a puzzle file fails to load, handle exception, and inform user:



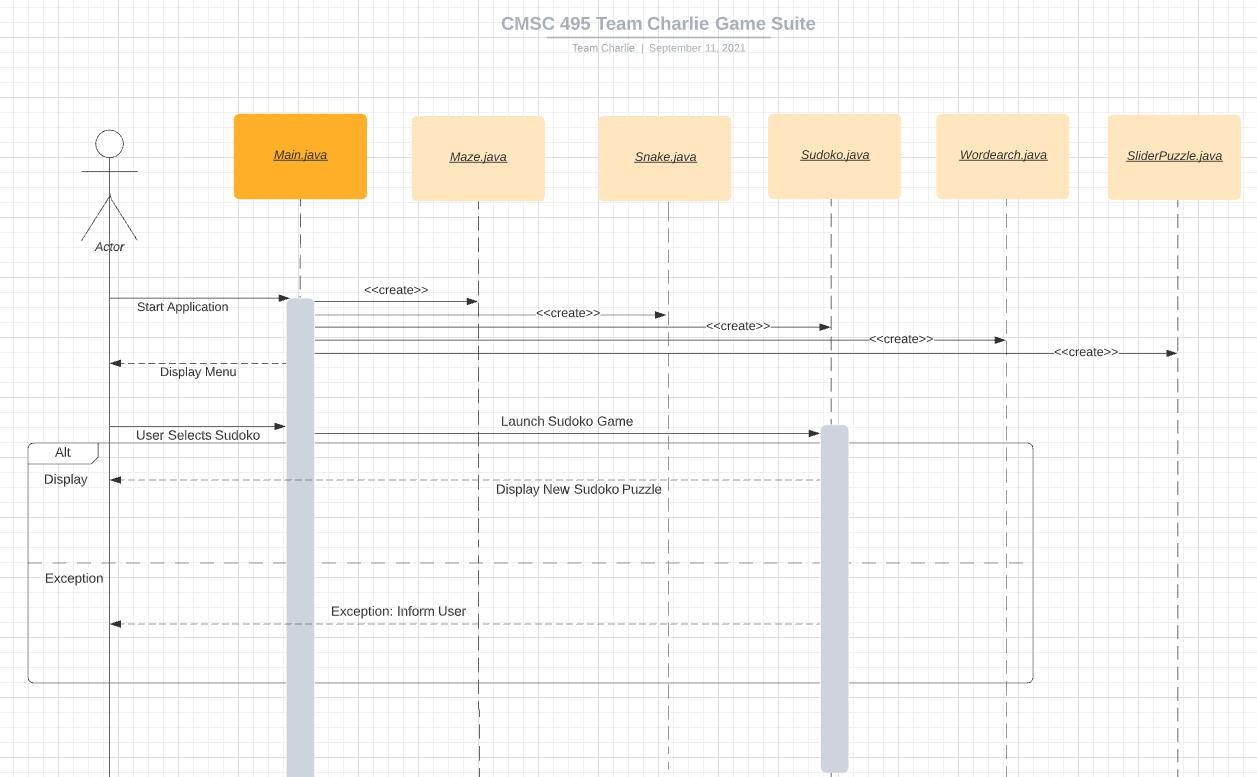


**Error-Handling Scenario 3:**

**Description:** User starts application and attempts to load a Sudoku puzzle.

**Pre-Condition:** Program has been running successfully.

**Post-Condition**: If Sudoku fails to load, handle exception, and display a message containing the line number and class name where the exception occurred using .printStackTrace():



## III. Pseudo Code / Code

**Main Menu**:

Build main menu

Ask to select a game

Show buttons for every game

IF Maze is selected

Start Maze Game

IF Snake is selected

Start Snake Game

IF Sudoku is selected

Start Sudoku Game

IF Word Search is selected

Start Word Search Game

If Slider Puzzle is Selected

Start Slider Puzzle

**Maze**:

Constructor (LevelNames levelName)

Call LevelName with levelName;

CALL JFrame with Frame("Maze Game");

Make Frame not resizable

CALL TIleMap with mazeLevelData

CALL GameScreen with gameScreen

ADD gameScreen to Frame

SET Frame as Visible

CALL EnemyMove with em

CALL KeyListener with KeyListener

Key is Pressed

If UP key or W is pressed

CALL move in mazeLevelData with argument UP enumeration

If DOWN key of S is Pressed

CALL move in mazeLevelData with argument DOWN enumeration

If LEFT key or A is Pressed

CALL Move in mazeLevelData with argument LEFT enumeration

IF RIGHT key or D is Pressed

CALL Move in mazeLevelData with argument RIGHT enumeration

CALL repaint in gameScreen

START em Thread

main Method

Initialize Menu Screen

IF start button is pressed

Start Maze

IF Instruction button is pressed

Show instructions

If High Score button pressed

Show High scores

win maze Method

Show congratulations screen

Go to next maze

death screen

Encourage player to try again

Restart current maze

Direction enumerations

UP

DOWN

LEFT

RIGHT

Level Name Enumerations

Level 1

Level 2 . . .

Level 12

Maze Items Enumeration

WALL

FLOOR

PLAYER

ENEMIES

EXIT

TIle Map

Constructor

Build Maze Grid according to level number

Set Dimensions of Maze according to grid width

Calculate width

Make Enemy (arguments X and y Coordinates)

Add enemy to grid using X and Y coordinates

Make Player

Add player start point

Make Exit

Add Exit to grid

Make Floor

Add Floor to grid

Make Vertical Path

FOR each grid place in path

CALL make Floor for given point

Make Horizontal Path

For each grid place in path

CALL Make Floor for given Point

Boolean isPresent with Arguments Maze Items and X and Y point

return True if Maze item exists for point in grid

Move with arguments Direction

IF UP is Direction and Grid point above is not WALL

Move to grid point

IF DOWN is Direction and Grid point below is not WALL

Move to grid point

IF LEFT is Direction and Grid point to left is not WALL

Move to grid point

IF RIGHT is Direction and Grid point to right is not WALL

Move to Grid Point

Boolean enemyGotYou with arguments x and y coordinate

Return true if enemy coordinate is x and y

GameScreen

Constructor with Argument Maze Level Data for Level

Fill Grid with maze level data

Draw Characters with arguments facing direction, Color and facial expression

Draw characters with all arguments

TitleScreen

Constructor

Make JFrame

Make Start button

Make instruction Button

Make High score button

Enemy Move

Run Thread

While STOP is not called

FOR each Enemy

CALL enemyMove for enemy

enemyMove

IF player is further away Vertically than Horizontally

CALL moveUpDown

IF could not move UP or DOWN

CALL moveLeftRight

ELSE

CALL moveLeftRight

IF cannot move left Right

CALL moveUpDown

moveUPDOWN

IF player is above you

IF no wall is above you

Move up

ELSE if player is below you

IF no wall is below you

Move down

ELSE

Return FALSE

CALL repaint in gameScreen

Return true (if ELSE is not called)

moveLeftRight

IF player is to left of you

IF no wall is to leftyou

Move left

ELSE if player is to right of you

IF no wall is to right ofyou

Move Right

ELSE

Return FALSE

CALL repaint in gameScreen

Return true (if ELSE is not called)

**Snakes**:

a. Board Subsystem:

class Board extends JPanel implements ActionListener {

//Define constants used in snake game

//The B\_WIDTH and B\_HEIGHT constants determine the size of the board.

private final int B\_WIDTH = 300;

private final int B\_HEIGHT = 300;

//DOT\_SIZE is the size of the apple and the dot of the snake.

private final int DOT\_SIZE = 10;

//ALL\_DOTS constant defines maximum number of possible dots on board (900 = (300\*300)/(10\*10)).

private final int ALL\_DOTS = 900;

//RAND\_POS constant is used to calculate a random position for an apple.

private final int RAND\_POS = 29;

//DELAY constant determines the speed of the game.

private final int DELAY = 140;

// Both arrays stores x and y coordinates of all joints of snake.

private final int x[] = new int[ALL\_DOTS];

private final int y[] = new int[ALL\_DOTS];

}

private void loadImages() {

//The ImageIcon class is used for displaying PNG images.

ImageIcon iid = new ImageIcon("src/resources/dot.png");

ball = iid.getImage();

ImageIcon iia = new ImageIcon("src/resources/apple.png");

apple = iia.getImage();

ImageIcon iih = new ImageIcon("src/resources/head.png");

head = iih.getImage();

}

private void initGame() {

//The initGame() method create the snake, randomly locate an apple on the board, and start the timer.

dots = 3;

for (int z = 0; z < dots; z++) {

x[z] = 50 - z \* 10;

y[z] = 50;

}

locateApple();

timer = new Timer(DELAY, this);

timer.start();

}

private void checkApple() {

//If the apple collides with the head, we increase the number of joints of the snake.

if ((x[0] == apple\_x) && (y[0] == apple\_y)) {

dots++;

locateApple();

}

}

//Moving snake joints up the chain.

for (int z = dots; z > 0; z--) {

x[z] = x[(z - 1)];

y[z] = y[(z - 1)];

}

//Moving snake head to the left.

if (leftDirection) {

x[0] -= DOT\_SIZE;

}

//If snake hits one of its joints with head the game is over.

for (int z = dots; z > 0; z--) {

if ((z > 4) && (x[0] == x[z]) && (y[0] == y[z])) {

inGame = false;

}

}

//Game is over if snake hits bottom of the board.

if (y[0] >= B\_HEIGHT) {

inGame = false;

}

b. Snake Subsystem:

//This is the main class along with the setResizable() method that affects the insets of JFrame

setResizable(false);

pack();

**Sudoku**

**Button Controller Class:**

**//Class controls the user actions from the ButtonPanel**

public class ButtonController implements ActionListener {

private Game game;

**// Constructor, sets game. @param game Game to be set.**

public ButtonController(Game game) {

this.game = game;

}

**// Performs action after user pressed button. @param e ActionEvent.**

public void actionPerformed(ActionEvent e) {

if (e.getActionCommand().equals("New"))

game.newGame();

else if (e.getActionCommand().equals("Check"))

game.checkGame();

else if (e.getActionCommand().equals("Exit"))

System.exit(0);

else if (e.getActionCommand().equals("Help on"))

game.setHelp(((JCheckBox)e.getSource()).isSelected());

else

game.setSelectedNumber(Integer.parseInt(e.getActionCommand()));

}

}

**Sudoku Controller Class:**

**//Class controls the user actions from SudokuPanel.**

public class SudokuController implements MouseListener {

private SudokuPanel sudokuPanel; **// Panel to control.**

private Game game; **// Current Sudoku game.**

**//Constructor, sets game. @param game Game to be set.**

public SudokuController(SudokuPanel sudokuPanel, Game game) {

this.sudokuPanel = sudokuPanel;

this.game = game;

}

**//Recovers if user clicked field in game. If so it sets the selected number at clicked position in game and updates clicked field. If user clicked a field and used left mouse button, number at clicked position will be cleared in game and clicked field will be updated. @param e MouseEvent.**

public void mousePressed(MouseEvent e) {

JPanel panel = (JPanel)e.getSource();

Component component = panel.getComponentAt(e.getPoint());

if (component instanceof Field) {

Field field = (Field)component;

int x = field.getFieldX();

int y = field.getFieldY();

if (e.getButton() == MouseEvent.BUTTON1 && (game.getNumber(x, y) == 0 || field.getForeground().equals(Color.BLUE))) {

int number = game.getSelectedNumber();

if (number == -1)

return;

game.setNumber(x, y, number);

field.setNumber(number, true);

} else if (e.getButton() == MouseEvent.BUTTON3 && !field.getForeground().equals(Color.BLACK)) {

game.setNumber(x, y, 0);

field.setNumber(0, false);

}

sudokuPanel.update(game, UpdateAction.CANDIDATES);

}

}

public void mouseClicked(MouseEvent e) { }

public void mouseEntered(MouseEvent e) { }

public void mouseExited(MouseEvent e) { }

public void mouseReleased(MouseEvent e) { }

}

**Sudoku Game Class:**

**//Class represents a Sudoku game. It contains the solution, the user input, the selected number and methods to check the validation of the user input.**

public class Game extends Observable {

private int[][] solution; **// Generated solution.**

private int[][] game; **// Generated game with user input.**

private boolean[][] check; **// Holder for checking validity of game.**

private int selectedNumber; **// Selected number by user.**

private boolean help; **// Help turned on or off.**

**// Constructor**

public Game() {

newGame();

check = new boolean[9][9];

help = true;

}

**//Generates a new Sudoku game. All observers will be notified, update action: new game.**

public void newGame() {

solution = generateSolution(new int[9][9], 0);

game = generateGame(copy(solution));

setChanged();

notifyObservers(UpdateAction.NEW\_GAME);

}

**//Checks user input again, the solution and puts it into a check matrix. All observers will be notified, update action: check.**

public void checkGame() {

selectedNumber = 0;

for (int y = 0; y < 9; y++) {

for (int x = 0; x < 9; x++)

check[y][x] = game[y][x] == solution[y][x];

}

setChanged();

notifyObservers(UpdateAction.CHECK);

}

**//Sets help turned on or off. All observers will be notified, update action: help. @param help True for help on, false for help off.**

public void setHelp(boolean help) {

this.help = help;

setChanged();

notifyObservers(UpdateAction.HELP);

}

**// Sets selected number to user input. All observers will be notified, update action: selected number. @param selectedNumber Number selected by user.**

public void setSelectedNumber(int selectedNumber) {

this.selectedNumber = selectedNumber;

setChanged();

notifyObservers(UpdateAction.SELECTED\_NUMBER);

}

**//Returns number selected user. @return Number selected by user.**

public int getSelectedNumber() {

return selectedNumber;

}

**//Returns whether help is turned on or off. @return True if help is turned on, false if help is turned off.**

public boolean isHelp() {

return help;

}

**//Returns whether selected number is candidate at given position. @param x X position in game. @param y Y position in game. @return True if selected number on given position is candidate, false otherwise.**

public boolean isSelectedNumberCandidate(int x, int y) {

return game[y][x] == 0 && isPossibleX(game, y, selectedNumber)

&& isPossibleY(game, x, selectedNumber) && isPossibleBlock(game, x, y, selectedNumber);

}

**//Sets given number on given position in the game. @param x, The x position in the game. @param y, The y position in the game. @param number, The number to be set.**

public void setNumber(int x, int y, int number) {

game[y][x] = number;

}

**//Returns number of given position. @param x, X position in game. @param y, Y position in game. @return, Number of given position.**

public int getNumber(int x, int y) {

return game[y][x];

}

**//Returns whether user input is valid of given position. @param x, X position in game. @param y, Y position in game. @return, True if user input of given position is valid, false otherwise.**

public boolean isCheckValid(int x, int y) {

return check[y][x];

}

**//Returns whether given number is candidate on x axis for given game. @param game, Game to check. @param y, Position of x axis to check. @param number, Number to check. @return, True if number is candidate on x axis, false otherwise.**

private boolean isPossibleX(int[][] game, int y, int number) {

for (int x = 0; x < 9; x++) {

if (game[y][x] == number)

return false;

}

return true;

}

**//Returns whether given number is candidate on y axis for given game. @param game, Game to check. @param x, Position of y axis to check. @param number, Number to check. @return, True if number is candidate on y axis, false otherwise.**

private boolean isPossibleY(int[][] game, int x, int number) {

for (int y = 0; y < 9; y++) {

if (game[y][x] == number)

return false;

}

return true;

}

**//Returns whether given number is candidate in block for given game. @param game, Game to check. @param x, Position of number on x axis in game to check. @param y, Position of number on y axis in game to check. @param number, Number to check. @return, True if number is candidate in block, false otherwise.**

private boolean isPossibleBlock(int[][] game, int x, int y, int number) {

int x1 = x < 3 ? 0 : x < 6 ? 3 : 6;

int y1 = y < 3 ? 0 : y < 6 ? 3 : 6;

for (int yy = y1; yy < y1 + 3; yy++) {

for (int xx = x1; xx < x1 + 3; xx++) {

if (game[yy][xx] == number)

return false;

}

}

return true;

}

**//Returns next posible number from list for given position or -1 when list is empty. @param game, Game to check. @param x, X position in game. @param y, Y position in game. @param numbers, List of remaining numbers. @return, Next possible number for position in game or -1 when list is empty.**

private int getNextPossibleNumber(int[][] game, int x, int y, List<Integer> numbers) {

while (numbers.size() > 0) {

int number = numbers.remove(0);

if (isPossibleX(game, y, number) && isPossibleY(game, x, number) && isPossibleBlock(game, x, y, number))

return number;

}

return -1;

}

**// Generates Sudoku game solution. @param game, Game to fill, user should pass 'new int[9][9]'. @param index, Current index, user should pass 0. @return, Sudoku game solution.**

private int[][] generateSolution(int[][] game, int index) {

if (index > 80)

return game;

int x = index % 9;

int y = index / 9;

List<Integer> numbers = new ArrayList<Integer>();

for (int i = 1; i <= 9; i++) numbers.add(i);

Collections.shuffle(numbers);

while (numbers.size() > 0) {

int number = getNextPossibleNumber(game, x, y, numbers);

if (number == -1)

return null;

game[y][x] = number;

int[][] tmpGame = generateSolution(game, index + 1);

if (tmpGame != null)

return tmpGame;

game[y][x] = 0;

}

return null;

}

**//Generates Sudoku game from solution. @param game, Game to be generated, user should pass a solution. @return, Generated Sudoku game.**

private int[][] generateGame(int[][] game) {

List<Integer> positions = new ArrayList<Integer>();

for (int i = 0; i < 81; i++)

positions.add(i);

Collections.shuffle(positions);

return generateGame(game, positions);

}

**//Generates Sudoku game from solution, user should use the other generateGame method. This method simple removes a number at a position. If the game isn't valid after this action, the game will be brought back to previous state. @param game,Game to be generated. @param positions, List of remaining positions to clear. @return, Generated Sudoku game.**

private int[][] generateGame(int[][] game, List<Integer> positions) {

while (positions.size() > 0) {

int position = positions.remove(0);

int x = position % 9;

int y = position / 9;

int temp = game[y][x];

game[y][x] = 0;

if (!isValid(game))

game[y][x] = temp;

}

return game;

}

**// Checks whether given game is valid. @param game, Game to check. @return, True if game is valid, false otherwise.**

private boolean isValid(int[][] game) {

return isValid(game, 0, new int[] { 0 });

}

**//Checks whether given game is valid, user should use the other isValid method. There may only be one solution @param game, Game to check. @param index, Current index to check. @param numberOfSolutions, Number of found solutions. Int[] instead of, int because of pass by reference @return, True if game is valid, false otherwise.**

private boolean isValid(int[][] game, int index, int[] numberOfSolutions) {

if (index > 80)

return ++numberOfSolutions[0] == 1;

int x = index % 9;

int y = index / 9;

if (game[y][x] == 0) {

List<Integer> numbers = new ArrayList<Integer>();

for (int i = 1; i <= 9; i++)

numbers.add(i);

while (numbers.size() > 0) {

int number = getNextPossibleNumber(game, x, y, numbers);

if (number == -1)

break;

game[y][x] = number;

if (!isValid(game, index + 1, numberOfSolutions)) {

game[y][x] = 0;

return false;

}

game[y][x] = 0;

}

} else if (!isValid(game, index + 1, numberOfSolutions))

return false;

return true;

}

**//Copies a game. @param game, Game to be copied. @return, Copy of given game.**

private int[][] copy(int[][] game) {

int[][] copy = new int[9][9];

for (int y = 0; y < 9; y++) {

for (int x = 0; x < 9; x++)

copy[y][x] = game[y][x];

}

return copy;

}

**/\***

**\* Prints given game to console. Used for debug.**

**\***

**\* @param game Game to be printed.**

**\***

**private void print(int[][] game) {**

**System.out.println();**

**for (int y = 0; y < 9; y++) {**

**for (int x = 0; x < 9; x++)**

**System.out.print(" " + game[y][x]);**

**System.out.println();**

**}**

**}\*/**

}

**Update Action:**

**// Enumeration used to inform observers what to update.**

public enum UpdateAction {

NEW\_GAME,

CHECK,

SELECTED\_NUMBER,

CANDIDATES,

HELP

}

**Button Panel Class:**

**//Class draws the button panel and reacts to updates from the model.**

public class ButtonPanel extends JPanel implements Observer {

JButton btnNew, btnCheck, btnExit; **// Used buttons.**

JCheckBox cbHelp; **// Used check box.**

ButtonGroup bgNumbers; **// Group for grouping the toggle buttons.**

JToggleButton[] btnNumbers; **// Used toggle buttons.**

**//Constructs the panel and arranges all components.**

public ButtonPanel() {

super(new BorderLayout());

JPanel pnlAlign = new JPanel();

pnlAlign.setLayout(new BoxLayout(pnlAlign, BoxLayout.PAGE\_AXIS));

add(pnlAlign, BorderLayout.NORTH);

JPanel pnlOptions = new JPanel(new FlowLayout(FlowLayout.LEADING));

pnlOptions.setBorder(BorderFactory.createTitledBorder(" Options "));

pnlAlign.add(pnlOptions);

btnNew = new JButton("New");

btnNew.setFocusable(false);

pnlOptions.add(btnNew);

btnCheck = new JButton("Check");

btnCheck.setFocusable(false);

pnlOptions.add(btnCheck);

btnExit = new JButton("Exit");

btnExit.setFocusable(false);

pnlOptions.add(btnExit);

JPanel pnlNumbers = new JPanel();

pnlNumbers.setLayout(new BoxLayout(pnlNumbers, BoxLayout.PAGE\_AXIS));

pnlNumbers.setBorder(BorderFactory.createTitledBorder(" Numbers "));

pnlAlign.add(pnlNumbers);

JPanel pnlNumbersHelp = new JPanel(new FlowLayout(FlowLayout.LEADING));

pnlNumbers.add(pnlNumbersHelp);

cbHelp = new JCheckBox("Help on", true);

cbHelp.setFocusable(false);

pnlNumbersHelp.add(cbHelp);

JPanel pnlNumbersNumbers = new JPanel(new FlowLayout(FlowLayout.LEADING));

pnlNumbers.add(pnlNumbersNumbers);

bgNumbers = new ButtonGroup();

btnNumbers = new JToggleButton[9];

for (int i = 0; i < 9; i++) {

btnNumbers[i] = new JToggleButton("" + (i + 1));

btnNumbers[i].setPreferredSize(new Dimension(40, 40));

btnNumbers[i].setFocusable(false);

bgNumbers.add(btnNumbers[i]);

pnlNumbersNumbers.add(btnNumbers[i]);

}

}

**// Method called when model sends update notification. @param o, The model. @param arg, The UpdateAction.**

public void update(Observable o, Object arg) {

switch ((UpdateAction)arg) {

case NEW\_GAME:

case CHECK:

bgNumbers.clearSelection();

break;

}

}

**//Adds controller to all components. @param buttonController, Controller which controls all user actions.**

public void setController(ButtonController buttonController) {

btnNew.addActionListener(buttonController);

btnCheck.addActionListener(buttonController);

btnExit.addActionListener(buttonController);

cbHelp.addActionListener(buttonController);

for (int i = 0; i < 9; i++)

btnNumbers[i].addActionListener(buttonController);

}

}

**Field Class:**

**//Class represents a field on the SudokuPanel.**

public class Field extends JLabel {

private int x; // X position in game.

private int y; // Y position in game.

**//Constructs the label and sets x and y positions in game. @param x, X position in game. @param y, Y position in game.**

public Field(int x, int y) {

super("", CENTER);

this.x = x;

this.y = y;

setPreferredSize(new Dimension(40, 40));

setBorder(BorderFactory.createLineBorder(Color.GRAY));

setFont(new Font(Font.DIALOG, Font.PLAIN, 20));

setOpaque(true);

}

**//Sets number and foreground color according to userInput. @param number, Number to be set. @param userInput, Boolean indicating number is user input or not.**

public void setNumber(int number, boolean userInput) {

setForeground(userInput ? Color.BLUE : Color.BLACK);

setText(number > 0 ? number + "" : "");

}

**// Returns x position in game. @return X position in game.**

public int getFieldX() {

return x;

}

**// Return y position in game. @return Y position in game.**

public int getFieldY() {

return y;

}

}

**Sudoku Class:**

**// Main class of program.**

public class Sudoku extends JFrame {

public Sudoku() {

super("Sudoku");

setDefaultCloseOperation(EXIT\_ON\_CLOSE);

getContentPane().setLayout(new BorderLayout());

Game game = new Game();

ButtonController buttonController = new ButtonController(game);

ButtonPanel buttonPanel = new ButtonPanel();

buttonPanel.setController(buttonController);

add(buttonPanel, BorderLayout.EAST);

SudokuPanel sudokuPanel = new SudokuPanel();

SudokuController sudokuController = new SudokuController(sudokuPanel, game);

sudokuPanel.setGame(game);

sudokuPanel.setController(sudokuController);

add(sudokuPanel, BorderLayout.CENTER);

game.addObserver(buttonPanel);

game.addObserver(sudokuPanel);

pack();

setLocationRelativeTo(null);

setVisible(true);

}

**// Main entry point of program. @param args Command line arguments.**

public static void main(String[] args) {

// Use System Look and Feel

try { UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName()); }

catch (Exception ex) { ex.printStackTrace(); }

new Sudoku();

}

}

**Sudoku Panel Class:**

**//Class draws the sudoku panel and reacts to updates from the model.**

public class SudokuPanel extends JPanel implements Observer {

**// Color constant for candidates.**

private static final Color COLOR\_CANDIDATE = new Color(102, 153, 255);

private Field[][] fields; **// Array of fields.**

private JPanel[][] panels; **// Panels holding the fields.**

**//Constructs the panel, adds sub panels and adds fields to these sub panels.**

public SudokuPanel() {

super(new GridLayout(3, 3));

panels = new JPanel[3][3];

for (int y = 0; y < 3; y++) {

for (int x = 0; x < 3; x++) {

panels[y][x] = new JPanel(new GridLayout(3, 3));

panels[y][x].setBorder(BorderFactory.createLineBorder(Color.DARK\_GRAY));

add(panels[y][x]);

}

}

fields = new Field[9][9];

for (int y = 0; y < 9; y++) {

for (int x = 0; x < 9; x++) {

fields[y][x] = new Field(x, y);

panels[y / 3][x / 3].add(fields[y][x]);

}

}

}

**//Method called when model sends update notification. @param o, The model. @param arg, The UpdateAction.**

public void update(Observable o, Object arg) {

switch ((UpdateAction)arg) {

case NEW\_GAME:

setGame((Game)o);

break;

case CHECK:

setGameCheck((Game)o);

break;

case SELECTED\_NUMBER:

case CANDIDATES:

case HELP:

setCandidates((Game)o);

break;

}

}

**//Sets the fields corresponding to given game. @param game, Game to be set.**

public void setGame(Game game) {

for (int y = 0; y < 9; y++) {

for (int x = 0; x < 9; x++) {

fields[y][x].setBackground(Color.WHITE);

fields[y][x].setNumber(game.getNumber(x, y), false);

}

}

}

**// Sets fields validity according to given game. @param game, Current game.**

private void setGameCheck(Game game) {

for (int y = 0; y < 9; y++) {

for (int x = 0; x < 9; x++) {

fields[y][x].setBackground(Color.WHITE);

if (fields[y][x].getForeground().equals(Color.BLUE))

fields[y][x].setBackground(game.isCheckValid(x, y) ? Color.GREEN : Color.RED);

}

}

}

**//Shows the candidates according to given game. @param game, Current game.**

private void setCandidates(Game game) {

for (int y = 0; y < 9; y++) {

for (int x = 0; x < 9; x++) {

fields[y][x].setBackground(Color.WHITE);

if (game.isHelp() && game.isSelectedNumberCandidate(x, y))

fields[y][x].setBackground(COLOR\_CANDIDATE);

}

}

}

**//Adds controller to all sub panels. @param sudokuController, Controller which controls all user actions.**

public void setController(SudokuController sudokuController) {

for (int y = 0; y < 3; y++) {

for (int x = 0; x < 3; x++)

panels[y][x].addMouseListener(sudokuController);

}

}

}

**Word Search**

//Note that “Set” is a data structure like an array, while “SET” is an assignment instruction

//The data type for “letter” is undefined as this is pseudocode and higher-level. Most likely it will be a Java JComponent.

//This class will create a word object

//The object holds a set of letters and a string of text for the word

CLASS word () {

Set letters

String text

FUNCTION constructor (String wordText)

SET text to wordText

FUNCTION addLetter(letter)

ADD letter to letters

FUNCTION toString

RETURN text

FUNTION isEqual (Set otherLetters)

IF Set difference of letters and otherLetters is zero

return true

ELSE

return false

ENDIF

//changes all the colors of a word’s letters for GUI display

FUNCTION changeAllLetterColor (newColor)

FOR letter in letters

SET letter’s color to newColor

END FOR

}

// Class reads in a data file and generates and formats puzzle data for the GUI program

CLASS PuzzleGenerator (puzzleDataFile, letterFont){

Array[][] letterMatrix

Array Word words

int numRows

int numColumns

int numWords

//creates and returns an ordered array of letters to place on the GUI

FUNCTION getLetterLabels()

Array letterList

FOR each row in letterMatrix

FOR each column in letterMatrix

APPEND letterMatrix(row, column) to end of letterList

END loop

END loop

RETURN letterList;

//returns the number of rows in the word search grid

FUNCTION **g**etNumRows()

RETURN numRows

//returns the number of columns in the word search grid

FUNCTION getNumColumns()

RETURN numColumns

//returns the words hidden in the puzzle

FUNCTION getWords()

RETURN words

//Reads in a Puzzle File and Sets Global Variables

FUNCTION readFile()

String fileLine

int lineCounter

SET lineCounter to 0

FOR each line in puzzleFile

fileLine = current line in the File

//Split line by tabs

CALL Split on fileLine RETURNING Array data

//Read in first row of puzzle file, which has puzzle specifications

IFlineCounter==0

numRows = data[0]

numColumns = data[1]

numWords = data[2]

//create matrix

INIT letterMatrix to Array[numRows][numColumns];

INCREMENT lineCounter

//then add letters to the matrix

ELSE IF lineCounter>0 AND lineCounter<=numRows

FOR each letter in data

SET letterMatrix[lineCounter-1][letter’s index] to letter END LOOP

//Read in the hidden words

//Create word object and add text

ELSE IF (lineCounter>numRows AND

lineCounter<=numRows + numWords)

words.add(**new** Word(data[0]))

INCREMENT lineCounter

//Read in the matrix location of the hidden words' letters

// and add those letters to the word

ELSEIF lineCounter>=numRows + numWords AND

lineCounter<=numRows + numWords+numWords

FORevery two numbers in data

SET row to the first number

SET column to the second number

SET letter to letterMatrix (row, column);

//get the corresponding word from Array

SET word to words[lineCounter-numWords -numRows-1]

CALL word.addLetter with letter

END LOOP

INCREMENT lineCounter

END IF

END LOOP

}

//Class displays the GUI

CLASS WordSearchProgram{

Array lines

Map strikeThroughs

int startX

int startY

puzzleFile

FUNCTION constructor{

Create loadButton

Create select puzzle button

Create clear button

Create undo button

Create field for puzzleFileTitle

Add buttons and fields to GUI

FUNCTION main

Displays GUI on start up

//adds the users’ highlights of letters onto grid and strikethroughs through words on the wordlist

FUNCTION paintComponent

FOR each line in lines

PAINT line

FOR strikethrough in strikeThroughs

PAINT strikethrough

FUNCTION mousePressed()

SET startX to cursor’s x coordinate

SET startY to cursor’s y coordinate

FUNCTION mouseReleased()

double x

double y

SET x to cursor’s x coordinate

SET y to cursor’s y coordinate

Create line from startX, startY, x, y

Add line to lines

SET startX, startY to 0

FUNCTION actionPerformed()

IF undoButton clicked

CALL undo

IF selectButton clicked

CALL selectPuzzle

IF clearButton clicked

CALL clear

IF submitButton clicked

CALL submit

END IF

FUNCTION undo()

Remove last line from lines

Repaint GUI

FUNCTION clear()

Remove all lines from lines

Remove all strikethroughs from strikethroughs

Repaint GUI

FUNCTION selectPuzzle():

CALL file chooser program

SET puzzleFile to file selected from file chooser program

SET puzzleTitleField to puzzle file’s name

FUNCTION loadPuzzle():

INIT generator with puzzle file

CALL generator.readFile()

SET letters to generator.getLetterLabels

SET words to generator.getWords

FOR each word in words

Add word text to wordlist display on GUI

END LOOP

FOR each letter in letters

Add letter to letter display on GUI

END LOOP

REPAINT GUI

//adds or removes a strikethrough from a word on the word list

FUNCTION mouseClicked()

IF mouse clicked on word in wordlist display on GUI

IF word in strikeThroughs

Remove word from strikeThroughs

ELSE

Create line through word coordinates

ADD word and line through word to strikeThroughs

END IF

ENDIF

FUNCTION scorePuzzle();

//First get the total number of words in the puzzle

SET numWords to length of words

FOR each line in lines

Set lineLetters

FOR each letter in letters

IF line intersects letter

add letter to lineLetters

ENDIF

END LOOP

FOR each word in words

IF words.isEquals(lineLetters) RETURNS True

//words intersecting line match actual word

Remove word from words

ENDIF

END LOOP

SET numFoundWords to numWords – length of words

DISPLAY numFoundWords on GUI

//Change color of all unfound words on GUI

FOR word in words:

CALL word.changeAllLetterColor(Red)

Repaint GUI

}

**Slider Puzzle**

public class SliderPuzzle extends JFrame {

Initialize "allicons" as new arraylist to store icons

Initialize "allButtons" as new arraylist to store buttons

// Constructor for SliderPuzzle class

public SliderPuzzle(){

// call method that splits the main image into 12 jpg files

imageDivider(new File("PuzzleName.jpg"));

// call method that stores icons

storeButtons();

//call method that creates buttons

createButtons();

//Initialize 11 image icons that will be used as tiles

Icon image1 through image12 = new ImageIcon("TileName.jpg");

// Initialize final image icons that will be used for hints

Icon finalPictureIcon0 = new ImageIcon("Puzzle1.jpg");

Icon finalPictureIcon1 = new ImageIcon("Puzzle2.jpg");

Icon finalPictureIcon2 = new ImageIcon("Puzzle3.jpg");

// Create main and sub panels that will be added to frame

JPanel backGroundPanel = new JPanel();

JPanel subPanel = new JPanel();

JPanel mainPanel = new JPanel();

JPanel puzzleChooserPanel = new JPanel();

JPanel finalImageSubPanel = new JPanel();

// Create JButtons that will display image icons to form puzzle

JButton jButton1 through jButton12 = new JButton(image1 through 12);

// JLabel used for choosing puzzle

JLabel newPuzzleJLabel = new JLabel("Choose a Puzzle");

// JButtons for choosing new puzzle

JButton newPuzzleButton1 = new JButton("Puzzle 1");

JButton newPuzzleButton2 = new JButton("Puzzle 2");

JButton newPuzzleButton3 = new JButton("Puzzle 3");

// JLabel and JButtons that displays puzzle hints when selected

JLabel finalPictureLabel = new JLabel("Puzzle Hints:");

JButton finalImageJButton = new JButton("Final Image 1");

JButton finalImageJButton1 = new JButton("Final Image 2");

JButton finalImageJButton2 = new JButton("Final Image 3");

// JLabels used to show which icon has the power to swap with adjacent icon

JLabel jLabelPowerIcon = new JLabel(image12);

JLabel powerIconDefinitionLabel = new JLabel("This image will swap with the adjacent icon:");

// Set layouts for panels

subPanel.setLayout(new BoxLayout(subPanel, BoxLayout.X\_AXIS));

backGroundPanel.setLayout(new BoxLayout(backGroundPanel, BoxLayout.Y\_AXIS));

mainPanel.setPreferredSize(new Dimension(524, 522));

mainPanel.setLayout(new GridLayout(4,3,0,0));

setSize(new Dimension(524,522));

// Add image tiles to main panel

mainpanel.add(jButton1 through jButton12);

// Add power icon definition labels

subPanel.add(powerIconDefinitionLabel);

subPanel.add(jLabelPowerIcon);

// Add JLabel and JButtons to puzzle chooser panel

puzzleChooserPanel.add(newPuzzleJLabel);

puzzleChooserPanel.add(newPuzzleButton1);

puzzleChooserPanel.add(newPuzzleButton2);

puzzleChooserPanel.add(newPuzzleButton3);

// Add JLabel and JButtons to final image sub panel

finalImageSubPanel.add(finalPictureLabel);

finalImageSubPanel.add(finalImageJButton);

finalImageSubPanel.add(finalImageJButton1);

finalImageSubPanel.add(finalImageJButton2);

// Add panels

backGroundPanel.add(mainPanel);

backGroundPanel.add(subPanel);

backGroundPanel.add(puzzleChooserPanel);

backGroundPanel.add(finalImageSubPanel);

add(backGroundPanel);

}

// Add action listeners to puzzle hint JButtons

finalImageJButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

JOptionPane.showMessageDialog(null,"", "Hint",

JOptionPane.INFORMATION\_MESSAGE, finalPictureIcon0);

}

}) ;

finalImageJButton1.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

JOptionPane.showMessageDialog(null,"","Hint",

JOptionPane.INFORMATION\_MESSAGE, finalPictureIcon1);

}

});

finalImageJButton2.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

JOptionPane.showMessageDialog(null,"","Hint",

JOptionPane.INFORMATION\_MESSAGE, finalPictureIcon2);

}

});

pack();

setResizable(false);

setLocationRelativeTo(null);

setVisible(true);

/\*\*

\* Stores all icons for puzzle in an arraylist

\*/

public void storeIcons(){

for(int i = 0; i < 12; i++){

Icon icon = new ImageIcon("Name" + i + ".jpg");

allIcons.add(icon);

}

}

/\*\*

\* Generates 12 JButtons

\*/

public void createButtons(){

for (int i = 0; i < 12; i++){

new JButton(allIcons.get(i));

}

Collections.shuffle(allIcons);

for(int j = 0; j < 12; j++){

mainPanel.add(allButtons.get(j));

}

}

/\*\*

\* Splits the complete image into separate tiles

\* @param file to split

\*/

public void imageDivider(File file) {

try {

FileInputStream fileInputStream = new FileInputStream(file);

BufferedImage image = ImageIO.read(fileInputStream);

int rows = 4;

int cols = 3;

int tiles = rows \* cols;

// Calculate tile width and height

int tileWidth = image.getWidth() / cols;

int tileHeight = image.getHeight() / rows;

int count = 0;

// This array stores the image tiles

BufferedImage images[] = new BufferedImage[tiles];

for (int x = 0; x < rows; x++) {

for (int y = 0; y < cols; y++) {

images[count] = new BufferedImage(tileWidth, tileHeight, image.getType());

Graphics2D gr = images[count++].createGraphics();

gr.drawImage(image, 0, 0, tileWidth, tileHeight, tileWidth \* y, tileHeight \* x,

tileWidth \* y + tileWidth, tileHeight \* x + tileHeight, null);

gr.dispose();

}

}

// Create new files for each puzzle piece 1 through 12

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

ImageIO.write(images[i], "jpg", new File("Puzzle Piece Names" + i + ".jpg"));

}

// Display message if file is not found in the working directory

} catch (IOException e) {

JOptionPane.showMessageDialog(null, e.getMessage());

}

}

public static void main(String[] args) {

ImagePuzzle imagePuzzle = new ImagePuzzle();

imagePuzzle.setDefaultCloseOperation(EXIT\_ON\_CLOSE);

}

}

## References:

“Bi-directional and uni-directional associations UML.” (2017, Sept). Stack Overflow. Retrieved September 12, 2021 from

https://stackoverflow.com/questions/15487372/bi-directional-and-uni-directional-associations-uml

Creately. (2021). Sequence diagram tutorial – complete guide with examples. Retrieved from

https://creately.com/blog/diagrams/sequence-diagram-tutorial/

Dalby, J. (2003, February 21). Pseudocode Standard. Cal Poly Computer Science & Software Engineering. Retrieved September 10, 2021 from “https://users.csc.calpoly.edu/~jdalbey/SWE/pdl\_std.html

IBM. (n.d.). “Aggregation Relationships.” Rational Software Architect 9.5. Retrieved from <https://www.ibm.com/docs/en/rational-soft-arch/9.5?topic=diagrams-aggregation-relationships>

University of Washington Computer Science and Engineering. (2011). *Design and UML Class Diagrams*. The University of Washington. Retrieved September 9, 2021 from <https://courses.cs.washington.edu/courses/cse403/11sp/lectures/lecture08-uml1.pdf>