Sym4 Game Term Project

*By Jeremy Liu*

*Table of Contents:*

1. Product Requirement………………………………………..2
2. High-Level Design……………………………………….....3
3. Low-Level Design………………………………………….6
4. Phase 4: Coding………………………………………….....9
5. Testing…………………………………………………….25
6. Product Release with User Manual……………………….25
7. Bibliography………………………………………………26

Source Files:

*Sym4Game.java*

*Sym4Game.class*

*Sym4Game$1.class*

*Sym4Game$HandlerClass.class*

*Sym4Game$Action.class*

*Sym4Game.jar*

**Phase 1: Product Requirement**

The Sym4Game is a fun interactive miniature version of Sudoku. Sym4 features a 4x4 grid that requires players to input values (1 – 4) to fill up all 16 squares.

There are four conditions that must hold true in order to complete a valid solution.

1. Every row must not contain any repeats
2. Every column must not contain any repeats
3. Every 2x2 square must not contain any repeats
4. Both Diagonals must not contain any repeats

Score is based off of the formula:

(if a player uses k moves and k <= 20) Points Gained = 20 – k

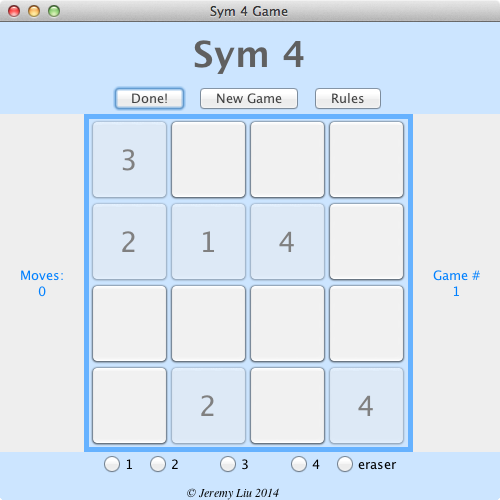
(if a player uses k moves and k > 20) Points Lost = k - 20

A board is randomly generated with 6 values already filled in and un-editable. That leaves 10 spaces available for user input.

The board must switch between different symbols to keep the game exciting

{1,2,3,4}, {A,B,C,D}, {P,Q,R,S}, {D,C,H,S}

The total score and the total number of games played are recorded.



**Phase 2: High-Level Design**

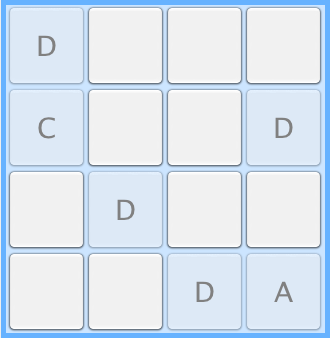
GUI is fun and simple. Very interactive board and intuitive design makes Sym4 desirable and highly addictive!

Uses 5 different panels on a JFrame, all composed together with a BorderLayout.

Panel on top displays the game name and possible game options:



Board Panel is in GridLayout and consists of clickable buttons to play the game:



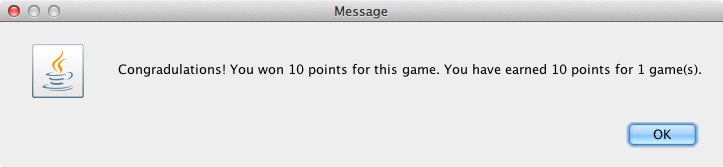
Two panels on each side: one shows the number of games, the other shows the number of moves.

Panel on the bottom displays the available choices to place on the board, including an ERASER!!

Macintosh HD:Users:Jeremy:Desktop:Screen Shot 2014-04-23 at 5.47.26 PM.png

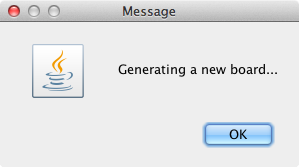
Clicking the “Done!” button checks the board and displays whether you board is correct:



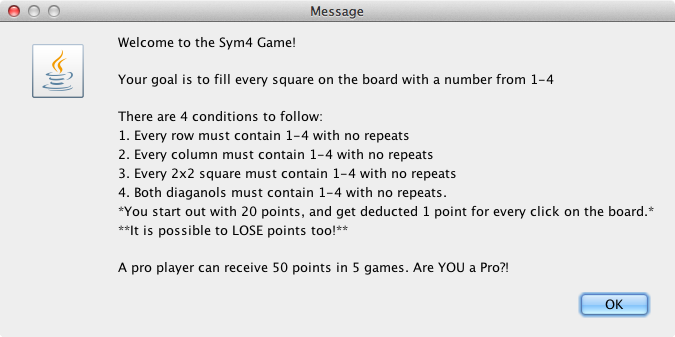
Or incorrect:



Clicking on “New Game” displays a generating new board message



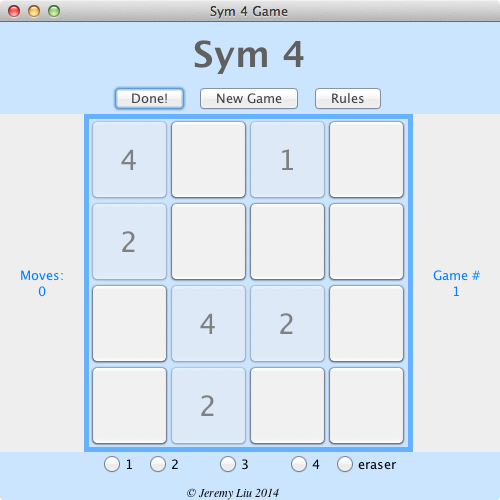
Clicking on the “Rules” displays the rules for Sym4



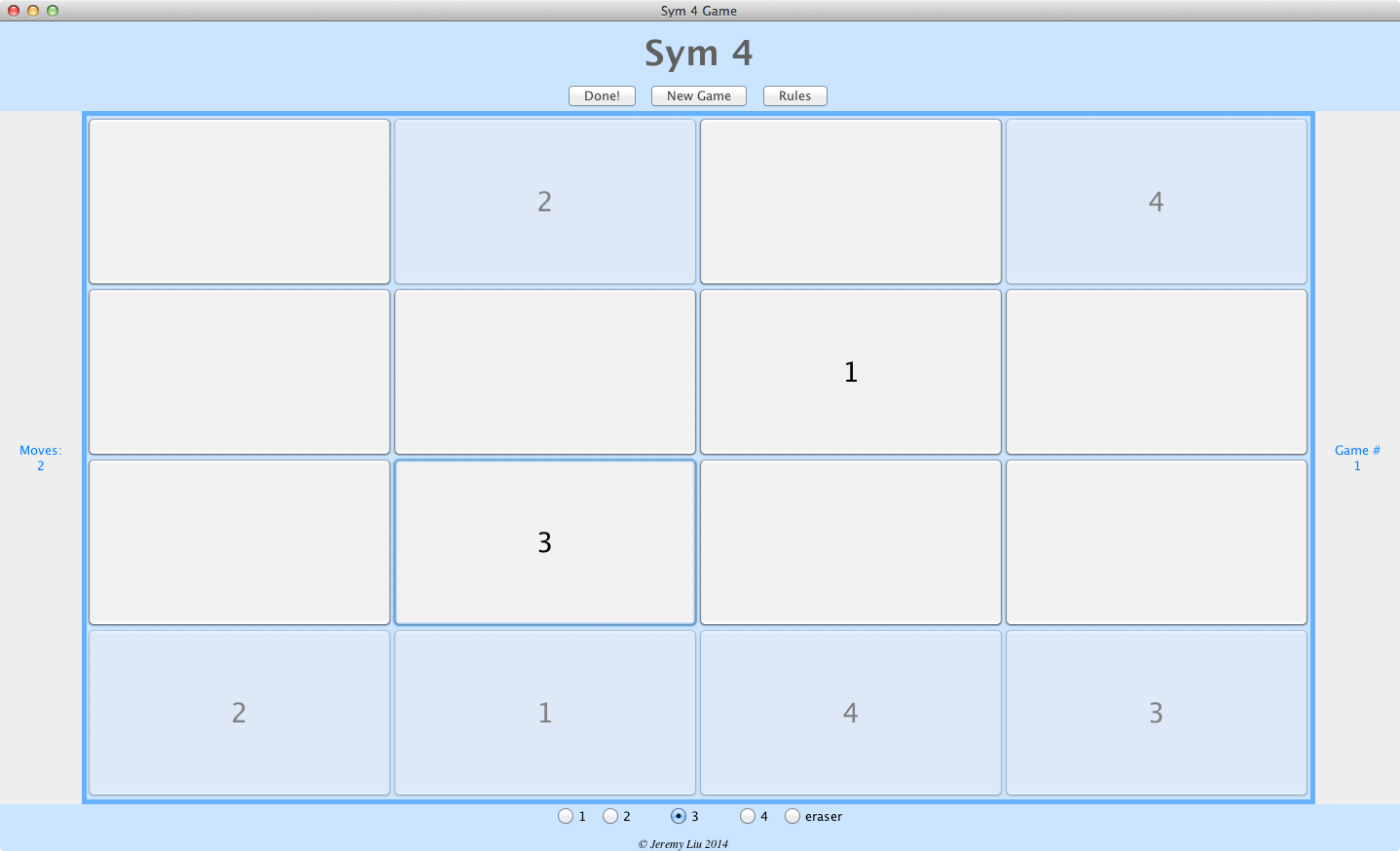
Full game, with all components together:

The 6 random positions that are already filled cannot be clicked.

(and are “grayed” out because of it)

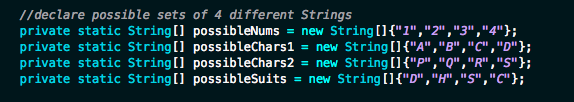


Fully resizable, and buttons change size to fit desired size. Here is example of full screen:



**Phase 3: Low-Level Design**

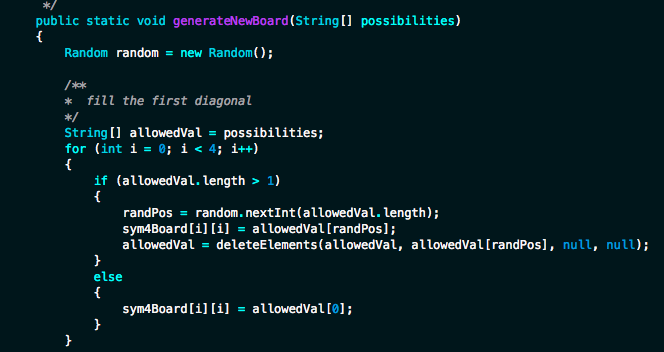
All arrays are Strings, so very flexible to create a different set of characters, as seen in the generateNewBoard method



**generateNewBoard(String[] possibilities)** method:

Takes a String array of the desired character set and creates a board based off of them

First fill the diagonal, then the square, column, and row, then fill leftover spaces with the remaining available characters.



**deleteElements(String[], String, String, String)**

This method is crucial in creating the board. When a random number is generated, it randomly picks the position of the String array (0,1,2, or 3) After that string has been used, it calls on deleteElements() to pass the array being used, and the value(s) to delete from the String array.

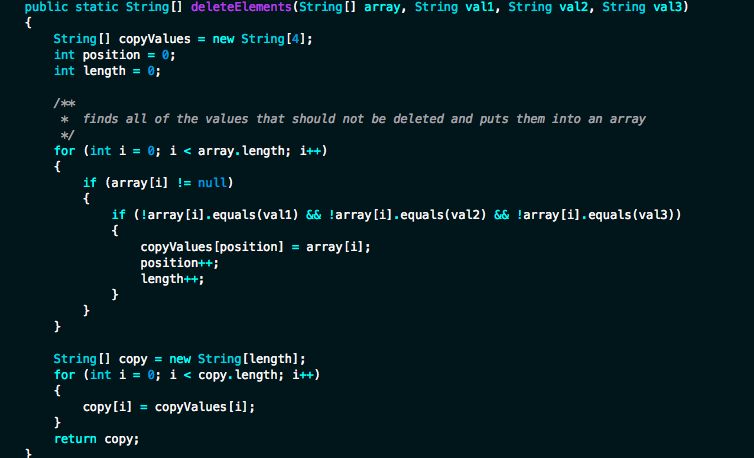
Next random number generated can only be from (0, 1, or 2) to pick the next position.

Then (0 or 1) for the third position

Finally places the last remaining String into the last available position

Good because it doesn’t waste time generating numbers or characters that are not allowed to go into the board.

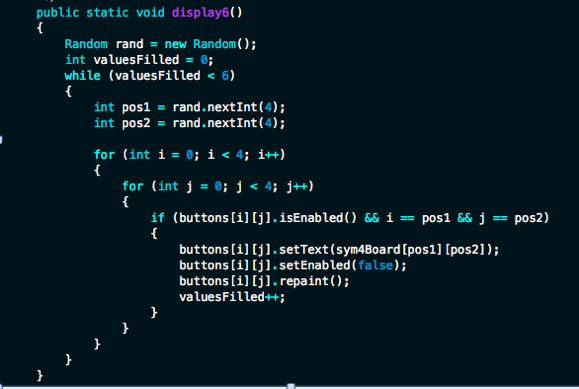
Also is flexible because it generates a random **position** not a **value.** So String arrays can be used, or any other array you want. (You don’t need multiple methods for different data types)



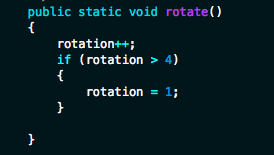
**display6()** method generates 6 random positions from the sym4Board (solution generated to play the game).

Also sets the button to be disabled so a user cannot click on it

If a position is already taken, it will reset the while loop and generate a new position until 6 are filled.



**rotate()** method rotates between different game types (or character sets), so every game is different



**Phase 4: Coding**

/\*\*

\* Sym4Game.java

\* Liu, Jeremy

\* April 21, 2014

\* Sym4 is a 4x4 game that is very similar to sudoku. Every column, row, square, and diagnol

\* must have only one sequence of 1, 2, 3, and 4 Sym4Game provides a GUI to play the game, dones

\* the validity, and rewards points determined by how many moves it took to complete the game

\* 4 different game modes: 1,2,3,4 or A,B,C,D or P,Q,R,S or D,S,C,H (Diamond, Spade, Club, Heart)

\*/

import java.awt.\*;

import javax.swing.\*;

import javax.swing.border.\*;

import java.awt.event.\*;

import java.util.Random;

public class Sym4Game extends JFrame

{

//Create GUI components

private JFrame gameBoard;

private JPanel board, p1, p2, p3, p4;

private JLabel sym4,moves,games,copyright;

private static JButton[][] buttons = new JButton[4][4];

private static JButton done, newGame, rules, reset;

private static JRadioButton choice1,choice2,choice3,choice4,erase;

private static ButtonGroup group;

private static JLabel moveNum,gameNum;

private static int moveCount = 0;

private static int totalGames = 1;

private static int score = 0;

private static int totalScore = 0;

private Border border;

private static String setValue = null;

private static int rotation = 1;//rotation tracker for different game modes

private static String[][] sym4Board = new String[4][4]; //create Sym4 game board

//declare possible sets of 4 different Strings

private static String[] possibleNums = new String[]{"1","2","3","4"};

private static String[] possibleChars1 = new String[]{"A","B","C","D"};

private static String[] possibleChars2 = new String[]{"P","Q","R","S"};

private static String[] possibleSuits = new String[]{"D","H","S","C"};

private static int randPos = 0; //position that randomly accesses an element from the possible choices

public Sym4Game()

{

/\*\* Initialize the boards and panels. \*/

gameBoard = new JFrame("Sym 4 Game");

gameBoard.setLayout(new BorderLayout());

gameBoard.setBackground(new Color(204,229,255));

gameBoard.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

Font s4f = new Font("3", Font.PLAIN, 28);

Font sym4F = new Font("sansserif", Font.BOLD, 36);

Font cr = new Font("serif", Font.ITALIC, 12);

border = BorderFactory.createLineBorder(new Color(102,178,255),5);

sym4 = new JLabel("Sym 4");

sym4.setFont(sym4F);

sym4.setForeground(new Color(96,96,96));

moves = new JLabel("Moves:");

moves.setForeground(new Color(0,128,255));

moveNum = new JLabel("0");

moveNum.setForeground(new Color(0,128,255));

games = new JLabel("Game #");

games.setForeground(new Color(0,128,255));

gameNum = new JLabel("1");

gameNum.setForeground(new Color(0,128,255));

copyright = new JLabel("© Jeremy Liu 2014");

copyright.setFont(cr);

board = new JPanel(new GridLayout(4,4));

p1 = new JPanel(new GridBagLayout());

p2 = new JPanel(new GridBagLayout());

p3 = new JPanel(new GridBagLayout());

p4 = new JPanel(new GridBagLayout());

/\*\* Initialize the radio buttons and add item listeners \*/

choice1 = new JRadioButton();

choice2 = new JRadioButton();

choice3 = new JRadioButton();

choice4 = new JRadioButton();

erase = new JRadioButton("eraser",false);

group = new ButtonGroup();

group.add(choice1);

group.add(choice2);

group.add(choice3);

group.add(choice4);

group.add(erase);

HandlerClass hc = new HandlerClass();

choice1.addItemListener(hc);

choice2.addItemListener(hc);

choice3.addItemListener(hc);

choice4.addItemListener(hc);

erase.addItemListener(hc);

/\*\* Initialize the 16 Jbuttons in a grid \*/

for (int i = 0; i < 4; i++)

{

for (int j = 0; j < 4; j++)

{

buttons[i][j] = new JButton();

buttons[i][j].setFont(s4f);

buttons[i][j].addActionListener(new Action());

}

}

/\*\* Initialize the other buttons \*/

done = new JButton("Done!");

done.setForeground(new Color(64,64,64));

done.addActionListener(new Action());

newGame = new JButton("New Game");

newGame.setForeground(new Color(64,64,64));

newGame.addActionListener(new Action());

rules = new JButton("Rules");

rules.setForeground(new Color(64,64,64));

rules.addActionListener(new Action());

reset = new JButton("reset");

reset.setForeground(new Color(64,64,64));

reset.setVisible(false);

reset.addActionListener(new Action());

/\*\* Add all the components to the board \*/

for (int i = 0; i < 4; i++)

{

for (int j = 0; j < 4; j++)

{

board.add(buttons[i][j]);

}

}

board.setBackground(new Color(204,229,255));

GridBagConstraints c = new GridBagConstraints();

c.insets = new Insets(0,3,3,3);

c.gridx = 0;

c.gridy = 0;

p1.add(choice1,c);

c.gridx = 1;

c.gridy = 0;

p1.add(choice2,c);

c.gridx = 2;

c.gridy = 0;

p1.add(choice3,c);

c.gridx = 3;

c.gridy = 0;

p1.add(choice4,c);

c.gridx = 4;

c.gridy = 0;

p1.add(erase,c);

c.insets = new Insets(10,0,0,3);

c.gridx = 2;

c.gridy = 3;

p1.add(copyright,c);

p1.setBackground(new Color(204,229,255));

c.gridx = 1;

c.gridy = 0;

p2.add(sym4,c);

c.gridx = 1;

c.gridy = 1;

p2.add(reset,c);

c.gridx = 0;

c.gridy = 2;

p2.add(done,c);

c.gridx = 1;

c.gridy = 2;

p2.add(newGame,c);

c.gridx = 2;

c.gridy = 2;

p2.add(rules,c);

p2.setBackground(new Color(204,229,255));

c.insets = new Insets(0,20,0,20);

c.gridx = 0;

c.gridy = 0;

p3.add(moves,c);

c.gridx = 0;

c.gridy = 1;

p3.add(moveNum,c);

c.gridx = 0;

c.gridy = 0;

p4.add(games,c);

c.gridx = 0;

c.gridy = 1;

p4.add(gameNum,c);

board.setBorder(border);

gameBoard.add(board, BorderLayout.CENTER);

gameBoard.add(p1, BorderLayout.SOUTH);

gameBoard.add(p2, BorderLayout.NORTH);

gameBoard.add(p3, BorderLayout.WEST);

gameBoard.add(p4, BorderLayout.EAST);

gameBoard.setSize(500,500);

gameBoard.setVisible(true);

gameBoard.setLocationRelativeTo(null);

}

/\*\* Action Class

\*

\* Action listener for all the buttons that are pressed (excluding radio buttons)

\*

\* @param ActionEvent: button clicked

\*/

static class Action implements ActionListener

{

public void actionPerformed (ActionEvent ae)

{

int correct = 0;

if (ae.getSource() == done)

{

for (int i = 0; i < 4; i++)

{

for (int j = 0; j < 4; j++)

{

if (buttons[i][j].getText() == sym4Board[i][j])

{

correct++;

}

}

}

if (correct == 16) //all 16 squares are correct

{

score = getScore(moveCount);

totalScore += score;

gameNum.setText(Integer.toString(totalGames));

if (score < 0)

{

score \*= -1;

JOptionPane.showMessageDialog(null, "Sorry! You lost " + score

+ " points for this game. You have earned " + totalScore

+ " points for " + totalGames + " game(s).");

}

else

{

JOptionPane.showMessageDialog(null, "Congradulations! You won " + score

+ " points for this game. You have earned " + totalScore

+ " points for " + totalGames + " game(s).");

}

clearBoard();

score = 0;//reset score for next game

moveCount = 0;//reset move counter

moveNum.setText(Integer.toString(moveCount));

moveNum.revalidate();

totalGames++;

gameNum.setText(Integer.toString(totalGames));

rotate();

if (rotation == 1)

{

generateNewBoard(possibleNums);

setChoices(possibleNums);

display6();

reset.doClick();

}

if (rotation == 2)

{

generateNewBoard(possibleChars1);

setChoices(possibleChars1);

display6();

reset.doClick();

}

if (rotation == 3)

{

generateNewBoard(possibleChars2);

setChoices(possibleChars2);

display6();

reset.doClick();

}

if (rotation == 4)

{

generateNewBoard(possibleSuits);

setChoices(possibleSuits);

display6();

reset.doClick();

}

refresh();

}

else

{

JOptionPane.showMessageDialog(null, "Sorry, the board you entered is incorrect.");

}

}

else if (ae.getSource() == newGame)

{

clearBoard();

rotate();

moveCount = 0; //reset the moveCount

moveNum.setText(Integer.toString(0));

moveNum.repaint();

if (rotation == 1)

{

generateNewBoard(possibleNums);

setChoices(possibleNums);

display6();

JOptionPane.showMessageDialog(null, "Generating a new board...");

reset.doClick();

}

if (rotation == 2)

{

generateNewBoard(possibleChars1);

setChoices(possibleChars1);

display6();

JOptionPane.showMessageDialog(null, "Generating a new board...");

reset.doClick();

}

if (rotation == 3)

{

generateNewBoard(possibleChars2);

setChoices(possibleChars2);

display6();

JOptionPane.showMessageDialog(null, "Generating a new board...");

reset.doClick();

}

if (rotation == 4)

{

generateNewBoard(possibleSuits);

setChoices(possibleSuits);

display6();

JOptionPane.showMessageDialog(null, "Generating a new board...");

reset.doClick();

}

}

else if (ae.getSource() == rules)

{

JOptionPane.showMessageDialog(null, "Welcome to the Sym4 Game!\n"

+ "\nYour goal is to fill every square on the board with a number from 1-4\n"

+ "\nThere are 4 conditions to follow:\n"

+ "1. Every row must contain 1-4 with no repeats\n"

+ "2. Every column must contain 1-4 with no repeats\n"

+ "3. Every 2x2 square must contain 1-4 with no repeats\n"

+ "4. Both diaganols must contain 1-4 with no repeats.\n"

+ "\*You start out with 20 points, and get deducted 1 point for every click on the board.\*\n"

+ "\*\*It is possible to LOSE points too!\*\*\n"

+ "\nA pro player can receive 50 points in 5 games. Are YOU a Pro?!");

}

/\*\*

\* What to do when a user clicks on any square on the board

\*/

else if (ae.getSource() == buttons[0][0])

{

buttons[0][0].setText(setValue);

moveCount++;

moveNum.setText(Integer.toString(moveCount));

}

else if (ae.getSource() == buttons[0][1])

{

buttons[0][1].setText(setValue);

moveCount++;

moveNum.setText(Integer.toString(moveCount));

}

else if (ae.getSource() == buttons[0][2])

{

buttons[0][2].setText(setValue);

moveCount++;

moveNum.setText(Integer.toString(moveCount));

}

else if (ae.getSource() == buttons[0][3])

{

buttons[0][3].setText(setValue);

moveCount++;

moveNum.setText(Integer.toString(moveCount));

}

else if (ae.getSource() == buttons[1][0])

{

buttons[1][0].setText(setValue);

moveCount++;

moveNum.setText(Integer.toString(moveCount));

}

else if (ae.getSource() == buttons[1][1])

{

buttons[1][1].setText(setValue);

moveCount++;

moveNum.setText(Integer.toString(moveCount));

}

else if (ae.getSource() == buttons[1][2])

{

buttons[1][2].setText(setValue);

moveCount++;

moveNum.setText(Integer.toString(moveCount));

}

else if (ae.getSource() == buttons[1][3])

{

buttons[1][3].setText(setValue);

moveCount++;

moveNum.setText(Integer.toString(moveCount));

}

else if (ae.getSource() == buttons[2][0])

{

buttons[2][0].setText(setValue);

moveCount++;

moveNum.setText(Integer.toString(moveCount));

}

else if (ae.getSource() == buttons[2][1])

{

buttons[2][1].setText(setValue);

moveCount++;

moveNum.setText(Integer.toString(moveCount));

}

else if (ae.getSource() == buttons[2][2])

{

buttons[2][2].setText(setValue);

moveCount++;

moveNum.setText(Integer.toString(moveCount));

}

else if (ae.getSource() == buttons[2][3])

{

buttons[2][3].setText(setValue);

moveCount++;

moveNum.setText(Integer.toString(moveCount));

}

else if (ae.getSource() == buttons[3][0])

{

buttons[3][0].setText(setValue);

moveCount++;

moveNum.setText(Integer.toString(moveCount));

}

else if (ae.getSource() == buttons[3][1])

{

buttons[3][1].setText(setValue);

moveCount++;

moveNum.setText(Integer.toString(moveCount));

}

else if (ae.getSource() == buttons[3][2])

{

buttons[3][2].setText(setValue);

moveCount++;

moveNum.setText(Integer.toString(moveCount));

}

else if (ae.getSource() == buttons[3][3])

{

buttons[3][3].setText(setValue);

moveCount++;

moveNum.setText(Integer.toString(moveCount));

}

else if (ae.getSource() == reset)

{

refresh();

}

}

}

/\*\* HandlerClass

\*

\* class to handle what to do when a radio button is selected

\*/

private class HandlerClass implements ItemListener

{

public void itemStateChanged(ItemEvent e)

{

if (e.getStateChange() == ItemEvent.SELECTED)

{

if (e.getSource() == choice1)

{

setValue = choice1.getText(); //set the set value to the chosen radio button

}

else if (e.getSource() == choice2)

{

setValue = choice2.getText(); //set the set value to the chosen radio button

}

else if (e.getSource() == choice3)

{

setValue = choice3.getText(); //set the set value to the chosen radio button

}

else if (e.getSource() == choice4)

{

setValue = choice4.getText(); //set the set value to the chosen radio button

}

else if (e.getSource() == erase)

{

setValue = null;

}

}

}

}

public static void main(String[] args)

{

new Sym4Game();

generateNewBoard(possibleNums);

setChoices(possibleNums);

display6();

}

/\*\* generateNewBoard

\*

\* Randomly generate a new board to display

\*

\* @param possibilities: String[] of what character set to use.

\* @return void

\*/

public static void generateNewBoard(String[] possibilities)

{

Random random = new Random();

/\*\*

\* fill the first diagonal

\*/

String[] allowedVal = possibilities;

for (int i = 0; i < 4; i++)

{

if (allowedVal.length > 1)

{

randPos = random.nextInt(allowedVal.length);

sym4Board[i][i] = allowedVal[randPos];

allowedVal = deleteElements(allowedVal, allowedVal[randPos], null, null);

}

else

{

sym4Board[i][i] = allowedVal[0];

}

}

/\*\*

\* fill first square

\*/

allowedVal = deleteElements(possibilities, sym4Board[0][0], sym4Board[1][1], null);

randPos = random.nextInt(allowedVal.length);

sym4Board[0][1] = allowedVal[randPos];

allowedVal = deleteElements(allowedVal, allowedVal[randPos], null, null);

sym4Board[1][0] = allowedVal[0];

/\*\*

\* fill the first row

\*/

allowedVal = deleteElements(possibilities, sym4Board[0][0], sym4Board[0][1], null);

sym4Board[0][2] = allowedVal[0];

sym4Board[0][3] = allowedVal[1];

if (!checkCols())

{

sym4Board[0][2] = allowedVal[1];

sym4Board[0][3] = allowedVal[0];

}

/\*\*

\* fill the first col

\*/

allowedVal = deleteElements(possibilities, sym4Board[0][0], sym4Board[1][0], null);

sym4Board[2][0] = allowedVal[0];

sym4Board[3][0] = allowedVal[1];

if (!checkRows())

{

sym4Board[2][0] = allowedVal[1];

sym4Board[3][0] = allowedVal[0];

}

/\*\*

\* fill the second diagonal, paying attention to rows and columns

\*/

allowedVal = deleteElements(possibilities, sym4Board[0][3], sym4Board[3][0], null);

sym4Board[1][2] = allowedVal[0];

sym4Board[2][1] = allowedVal[1];

if (!checkCols() || !checkRows())

{

sym4Board[1][2] = allowedVal[1];

sym4Board[2][1] = allowedVal[0];

}

/\*\*

\* fill the last values

\*/

allowedVal = deleteElements(possibilities, sym4Board[1][0], sym4Board[1][1], sym4Board[1][2]);

sym4Board[1][3] = allowedVal[0];

allowedVal = deleteElements(possibilities, sym4Board[2][0], sym4Board[2][1], sym4Board[2][2]);

sym4Board[2][3] = allowedVal[0];

allowedVal = deleteElements(possibilities, sym4Board[0][1], sym4Board[1][1], sym4Board[2][1]);

sym4Board[3][1] = allowedVal[0];

allowedVal = deleteElements(possibilities, sym4Board[0][2], sym4Board[1][2], sym4Board[2][2]);

sym4Board[3][2] = allowedVal[0];

}

/\*\* deleteElements

\*

\* create a new String[] and returns it without the specified values

\*

\* @param array: the array that needs a value deleted

\* @param val1: the 1st value to delete

\* @param val2: the 2nd value to delete (optional)

\* @param val3: the 3rd value to delete (optional)

\* @return copy: String[] with specified values deleted

\*/

public static String[] deleteElements(String[] array, String val1, String val2, String val3)

{

String[] copyValues = new String[4];

int position = 0;

int length = 0;

/\*\*

\* finds all of the values that should not be deleted and puts them into an array

\*/

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

{

if (array[i] != null)

{

if (!array[i].equals(val1) && !array[i].equals(val2) && !array[i].equals(val3))

{

copyValues[position] = array[i];

position++;

length++;

}

}

}

String[] copy = new String[length];

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

{

copy[i] = copyValues[i];

}

return copy;

}

/\*\* checkRows

\*

\* Check every row in the board for repeats. Each row should only have

\* numbers 1-4, and none of the numbers should be repeated

\*

\* @param none

\* @return boolean: true if there are no repeats, false if there are

\*/

public static boolean checkRows()

{

for (int currRow = 0; currRow < 4; currRow++) //this for loop checks all 4 rows

{

/\*\*

\* use a nested for loop to check every number in the specified row with all of the

\* following numbers in that same row to check for duplicates.

\*/

for (int i = 0; i < 4; i++)

{

for (int j = i + 1; j < 4; j++)

{

if (sym4Board[currRow][i] != null && sym4Board[currRow][i].equals(sym4Board[currRow][j]))

{

return false; //if there are any repeats, return false

}

}

}

}

return true;

}

/\*\* checkCols

\*

\* Check every column in the board for repeats. Each column should only have

\* numbers 1-4, and none of the numbers should be repeated

\*

\* @param none

\* @return boolean: true if there are no repeats, false if there are

\*/

public static boolean checkCols()

{

for (int currCol = 0; currCol < 4; currCol++) //keep track of the current column

{

/\*\*

\* use a nested for loop to test every number in the specified column with all

\* of the following numbers in that same column for duplicates.

\*/

for (int i = 0; i < 4; i++)

{

for (int j = i + 1; j < 4; j++)

{

if (sym4Board[i][currCol] != null && sym4Board[i][currCol].equals(sym4Board[j][currCol]))

{

return false; //if there are any repeats, return false

}

}

}

}

return true;

}

/\*\* checkSqrs

\*

\* The following diagram explains which square is which:

\* ---------

\* | 1 | 2 |

\* ---------

\* | 3 | 4 |

\* ---------

\* Checks 4 different squares in the board for repeats. Each square should only have

\* numbers 1-4 and none of the numbers should be repeated

\*

\* @param none

\* @return boolean: true if there are no repeats, false if there are.

\*/

public static boolean checkSqrs()

{

/\*\*

\* create a 2D array with 4 rows that contain 4 elements each

\* copy[x][] simulates a row in sudoku

\* each square in board is equivalent to one row in the copy array.

\*/

String[][] copy = new String[4][4];

int counter = 0; //tracker to allow the row in copy to reach up to position 3

//copies square 1 into row 0 of the copy array.

for (int row = 0; row < 2; row++)

{

for (int col = 0; col < 2; col++)

{

/\*\*

\* accesses every element in the 2x2 square and maps it to the appropriate

\* element in a single row of the copy array

\*/

copy[0][counter] = sym4Board[row][col];

counter++; //counter will go up to 3, so all elements can be reached

}

}

//copies square 2 into row 1 of the copy array.

counter = 0; //reset the counter.

for (int row = 0; row < 2; row++)

{

for (int col = 2; col < 4; col++)

{

copy[1][counter] = sym4Board[row][col];

counter++;

}

}

//copies square 3 into row 2 of the copy array

counter = 0; //reset the counter.

for (int row = 2; row < 4; row++)

{

for (int col = 0; col < 2; col++)

{

copy[2][counter] = sym4Board[row][col];

counter++;

}

}

//copies square 4 into row 3 of the copy array

counter = 0; //reset the counter.

for (int row = 2; row < 4; row++)

{

for (int col = 2; col < 4; col++)

{

copy[3][counter] = sym4Board[row][col];

counter++;

}

}

// triple nested for loop to check all squares at once.

// the current square in the board array is equivalent to the current row in the copy array

for (int currBox = 0; currBox < 4; currBox++) //keeps track of the current row in the copy array

{

/\*\*

\* use a nested for loop to test every number in the specified 2x2 square with all

\* of the following numbers in that same square for duplicates or zeros.

\*/

for (int i = 0; i < 4; i++)

{

for (int j = i + 1; j < 4; j++)

{

if (copy[currBox][i] != null && copy[currBox][i].equals(copy[currBox][j]))

{

return false; //if there are any repeats, return false

}

}

}

}

return true;

}

/\*\* clearBoard

\*

\* clear the game board for a new solution plate

\* sets all buttons to editable again

\*

\* @param none

\* @return void

\*/

public static void clearBoard()

{

for (int i = 0; i < 4; i++)

{

for (int j = 0; j < 4; j++)

{

sym4Board[i][j] = null;

buttons[i][j].setEnabled(true);

buttons[i][j].setText(null);

}

}

refresh();

}

/\*\* rotate

\*

\* switch between the different sets of characters

\*

\* @param none

\* @return void

\*/

public static void rotate()

{

rotation++;

if (rotation > 4)

{

rotation = 1;

}

}

/\*\* display6

\*

\* display 6 values on the board and leave the rest blank.

\*

\* @param none

\* @return void

\*/

public static void display6()

{

Random rand = new Random();

int valuesFilled = 0;

while (valuesFilled < 6)

{

int pos1 = rand.nextInt(4);

int pos2 = rand.nextInt(4);

for (int i = 0; i < 4; i++)

{

for (int j = 0; j < 4; j++)

{

if (buttons[i][j].isEnabled() && i == pos1 && j == pos2)

{

buttons[i][j].setText(sym4Board[pos1][pos2]);

buttons[i][j].setEnabled(false);

buttons[i][j].repaint();

valuesFilled++;

}

}

}

}

}

/\*\* setChoices

\*

\* Change the available choices to a different character set and clear the previous selection

\*

\* @param possibilities: String[] of which character set to use

\* @return void

\*/

public static void setChoices(String[] possibilities)

{

choice1.setText(possibilities[0]);

choice2.setText(possibilities[1]);

choice3.setText(possibilities[2]);

choice4.setText(possibilities[3]);

setValue = null;

choice1.repaint();

choice2.repaint();

choice3.repaint();

choice4.repaint();

group.clearSelection();

}

/\*\* getScore

\*

\* translate the moves taken into the total score

\*

\* @param moves: how many moves the user took

\* @return points: how many points the user was awarded

\*/

public static int getScore(int moves)

{

int points = 20 - moves;

return points;

}

/\*\* refresh

\*

\* refresh the buttons on the board

\*

\* @param none

\* @return void

\*/

public static void refresh()

{

for (int i = 0; i < 4; i++)

{

for (int j = 0; j < 4; j++)

{

buttons[i][j].repaint();

}

}

}

}

**Phase 5: Testing**

Testing requires too many pictures to add to the word document. (20+ pictures per test, 7 tests)

So included in the .zip file is a folder titled “Test Cases” to look through all 7 test cases.

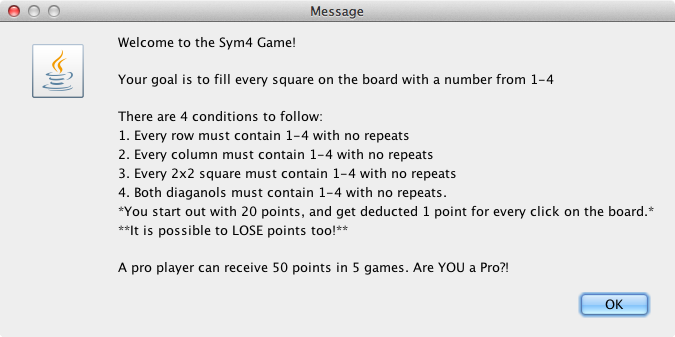
**Phase 6: Product Release With User Manual**

Manual:

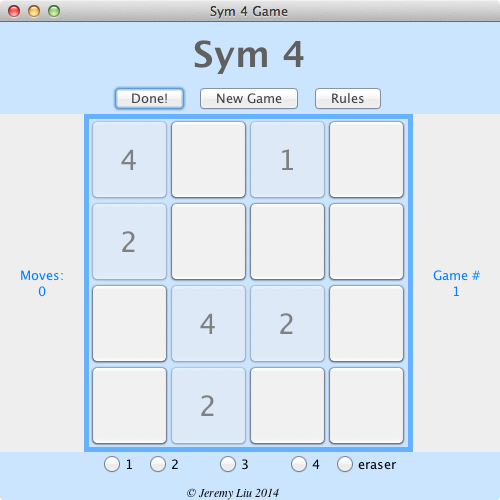
1. Double click on Sym4Game.jar



1. Click on “rules” to find out the rules of Sym4



1. HAVE FUN!



**Bibliography:**

Prichard, Janet J., Carrano, Frank M. “Data Abstraction & Problem Solving with Java” 3rd edition. Pearson Education Inc., publishing as Addison-Wesley. 2004-2011.

Various Authors. “Java Platform, Standard Edition 7 API Specification”. <http://docs.oracle.com/javase/7/docs/api/overview-summary.html>. Oracle, 1993 – 2014.

Accessed on 4/21/14

Various Authors. “A Visual Guide To Layout Managers”. http://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html . Oracle, 1993 - 2014.

Accessed on 4/21/14