1. Class Diagram

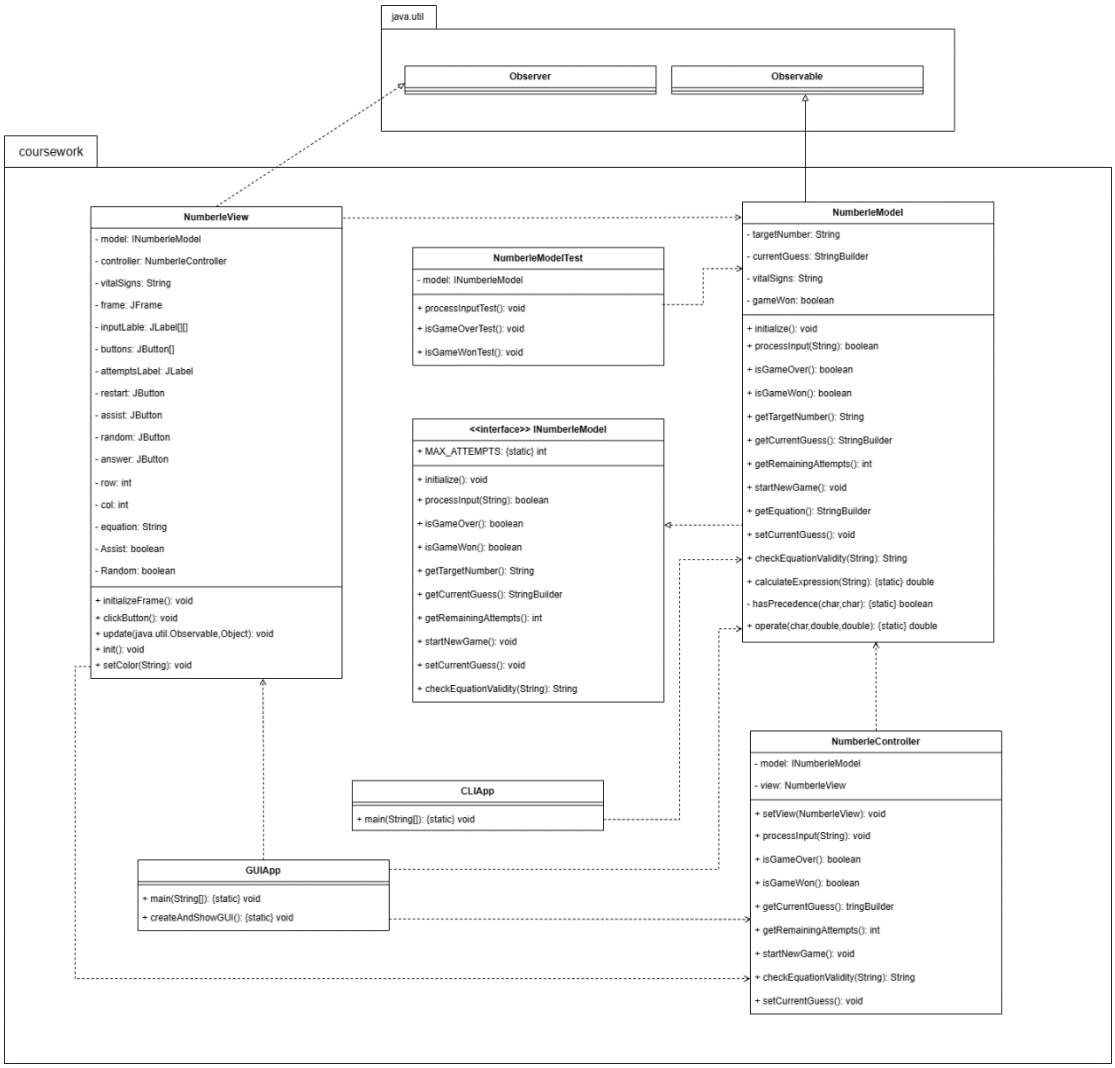

Figure 1. Class Diagram

2. Testing

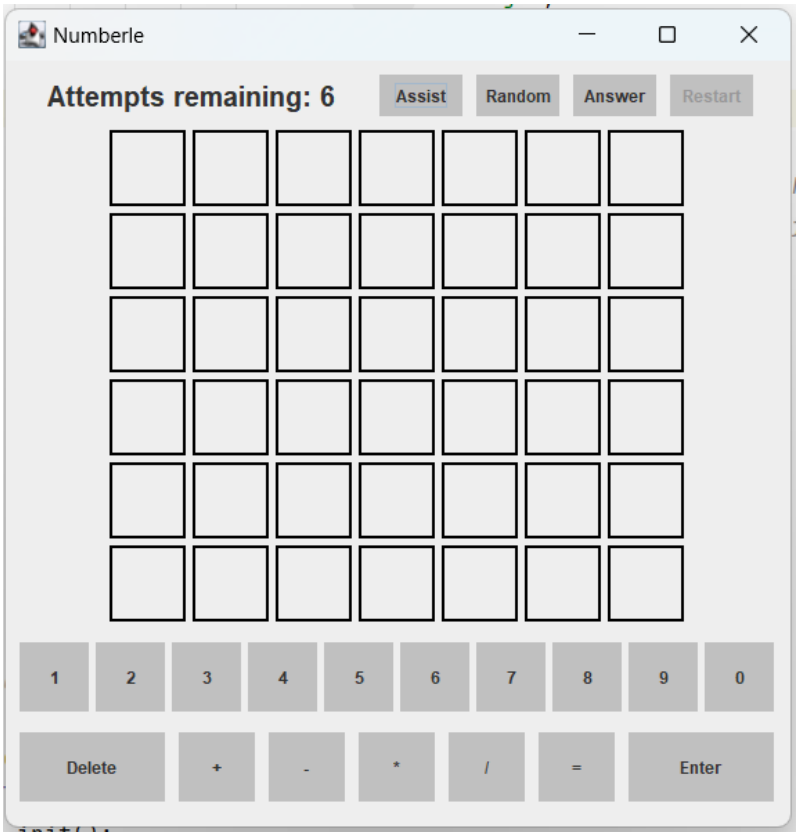

Figure 2. GUI Testing Diagram: Start

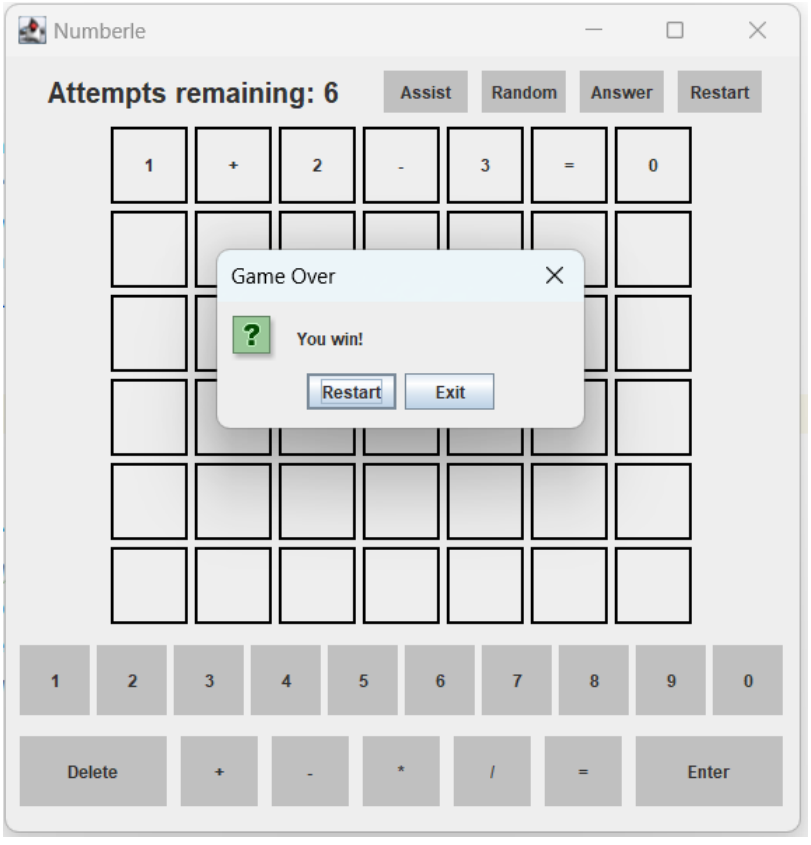

Figure 3. GUI Testing Diagram: Win The Game

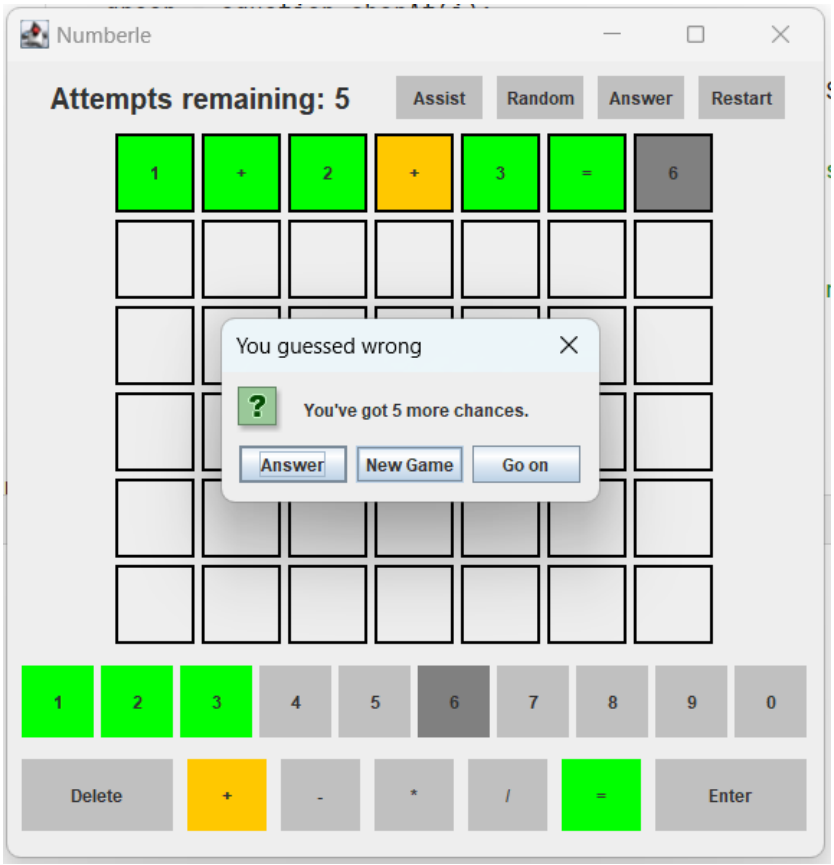

Figure 4. GUI Testing Diagram: Incorrect Answer

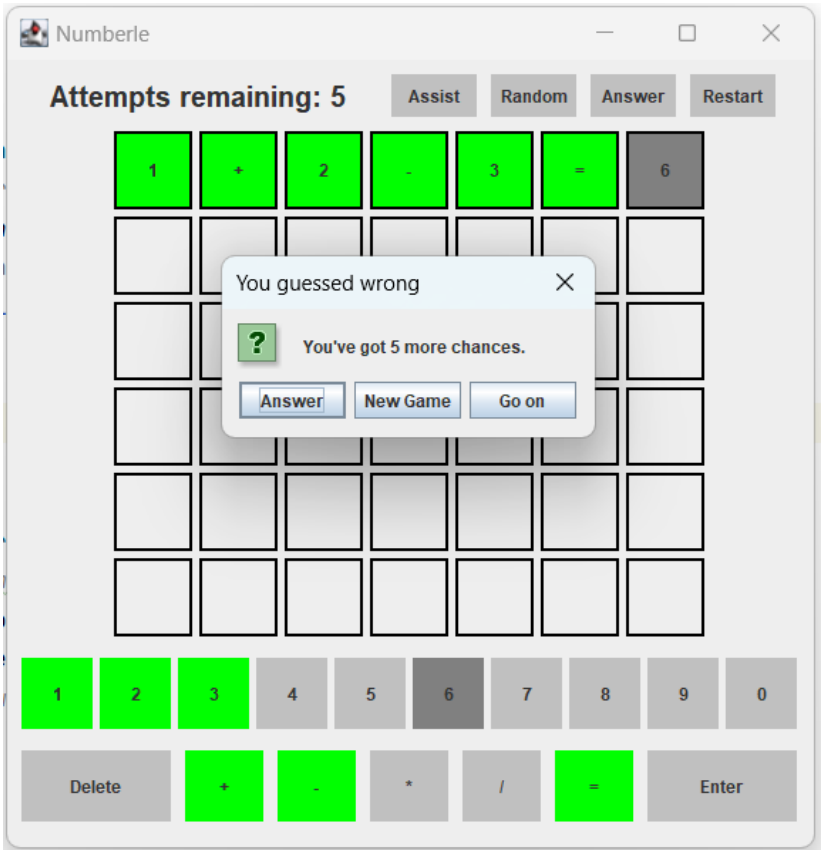

Figure 5. GUI Testing Diagram: No Equal and No Assist

Figure 6. GUI Testing Diagram: No “=” and No Assist

Figure 7. GUI Testing Diagram: No Enough Length and No Assist

Figure 8. GUI Testing Diagram: More Math Symbol and No Assist

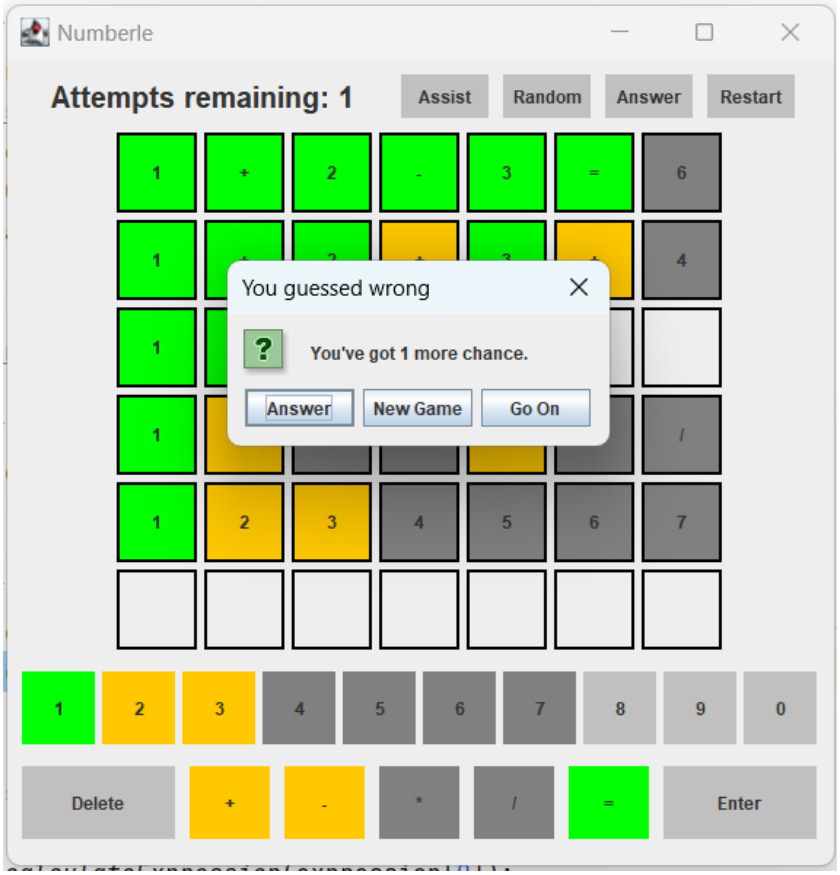

Figure 9. GUI Testing Diagram: No Math Symbol and No Assist

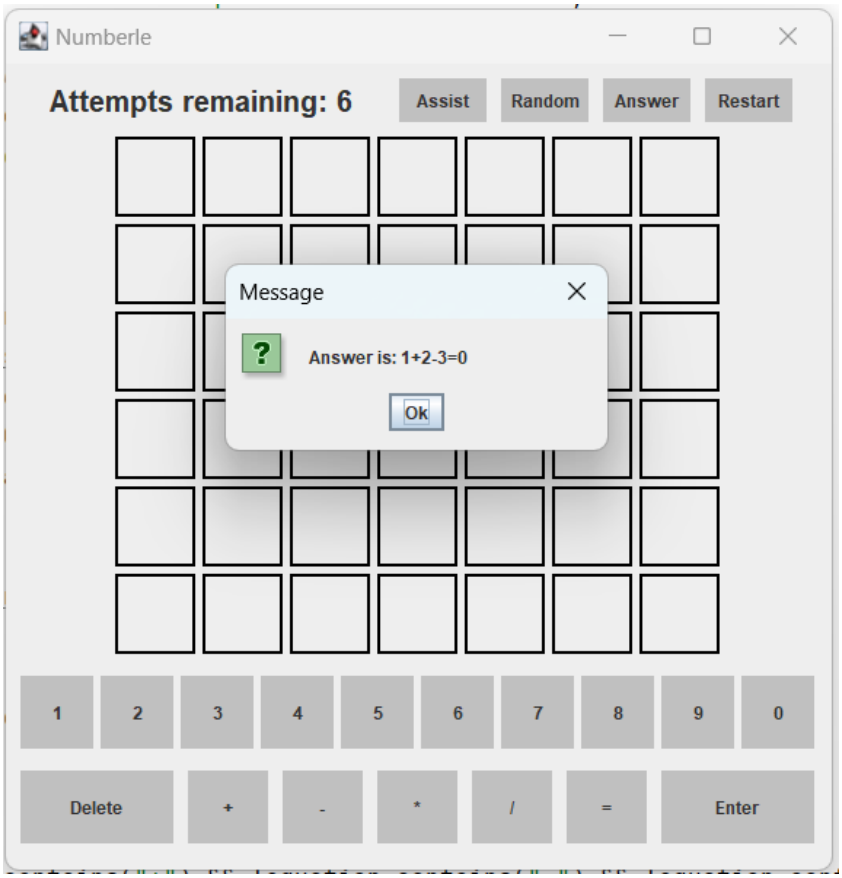

Figure 10. GUI Testing Diagram: Restart and No Random

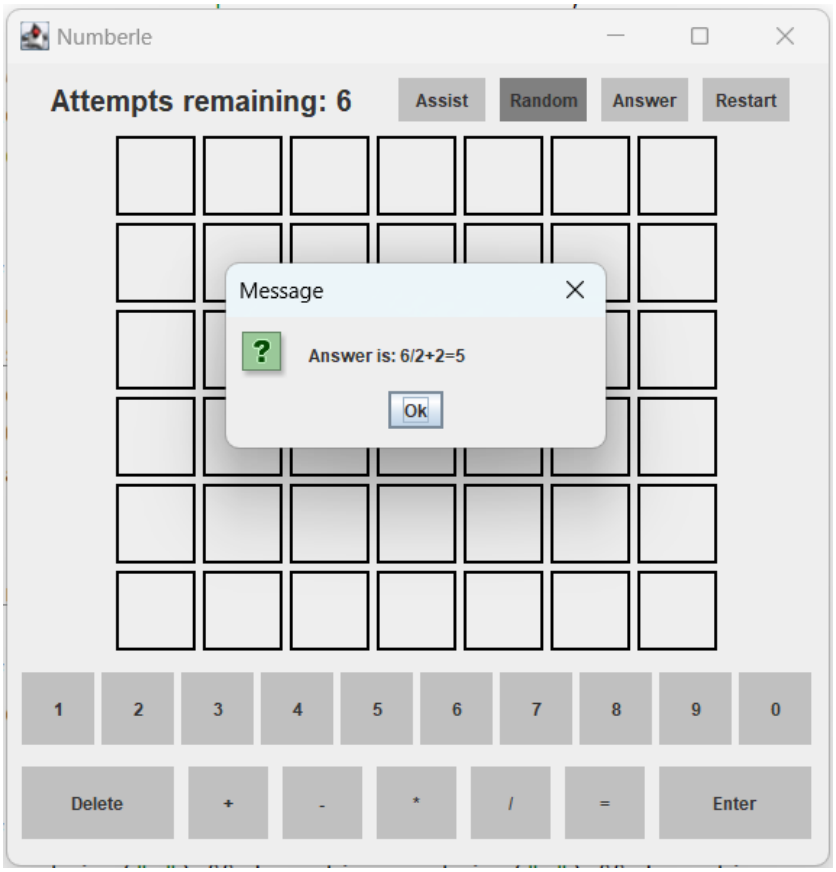

Figure 11. GUI Testing Diagram: Restart and Use Random

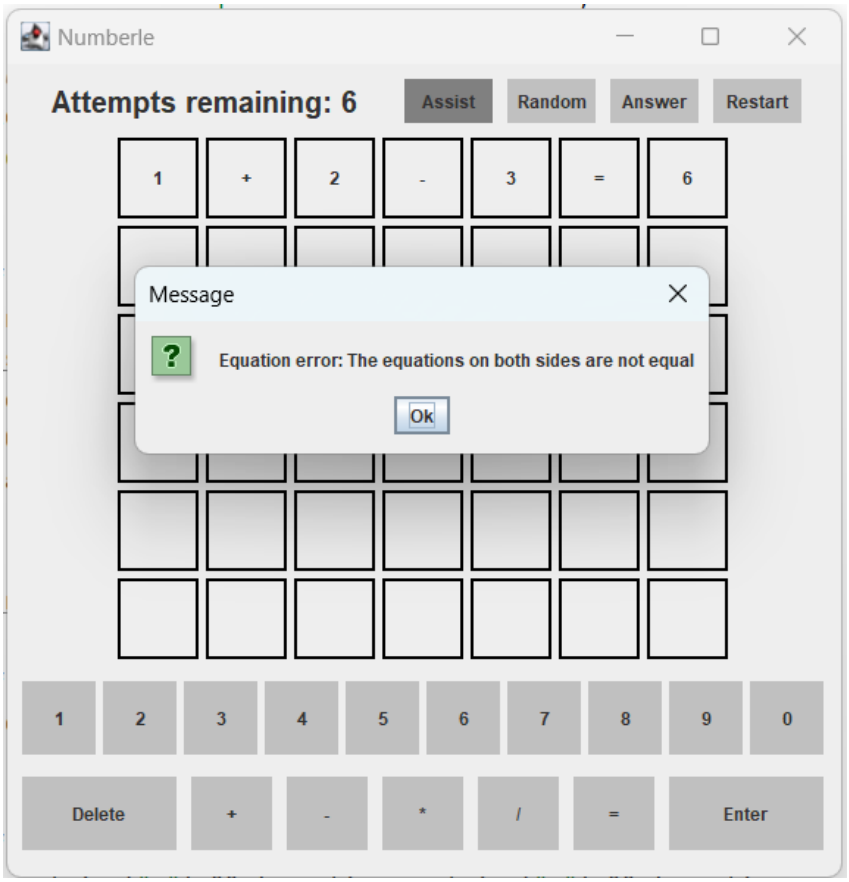

Figure 12. GUI Testing Diagram: No Equal and Use Assist

Figure 13. GUI Testing Diagram: No “=” and Use Assist

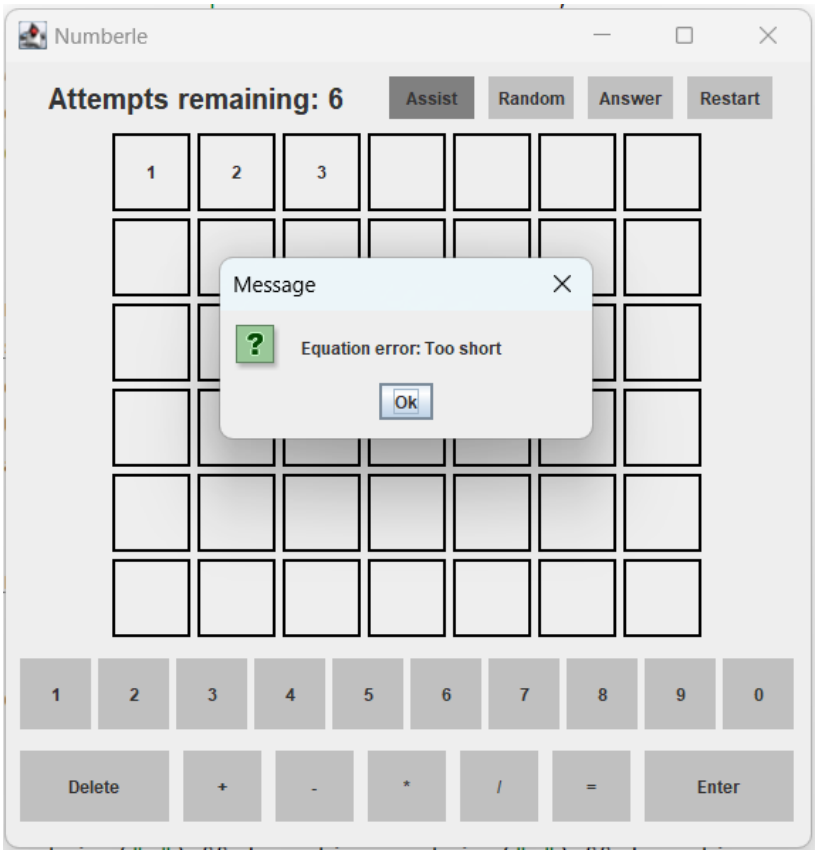

Figure 14. GUI Testing Diagram: No Enough Length and Use Assist

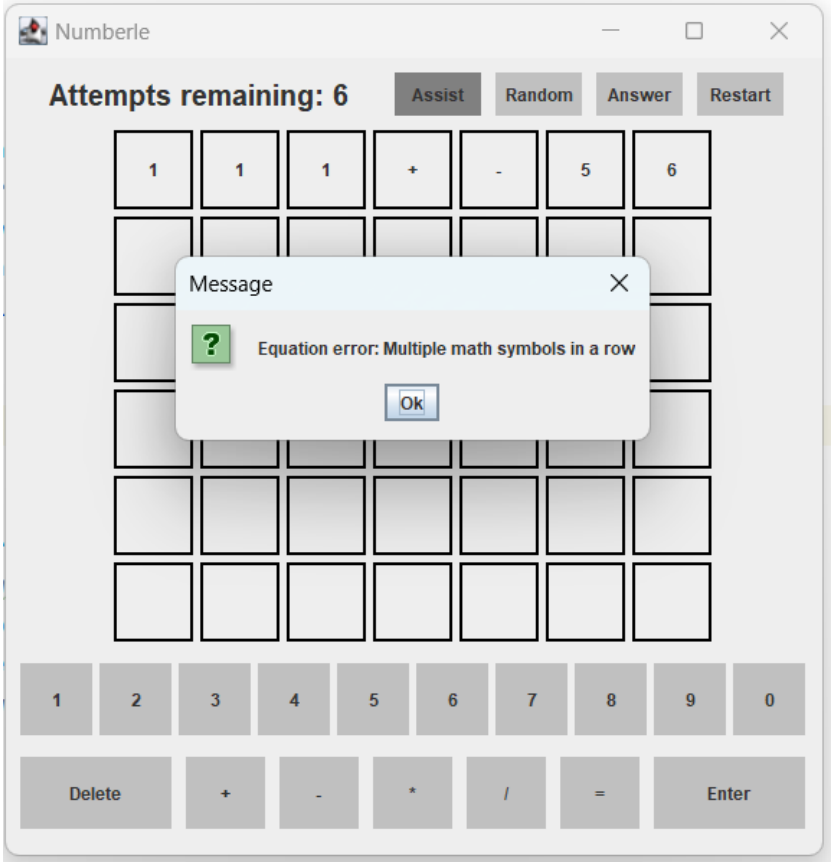

Figure 15. GUI Testing Diagram: More Math Symbol and Use Assist

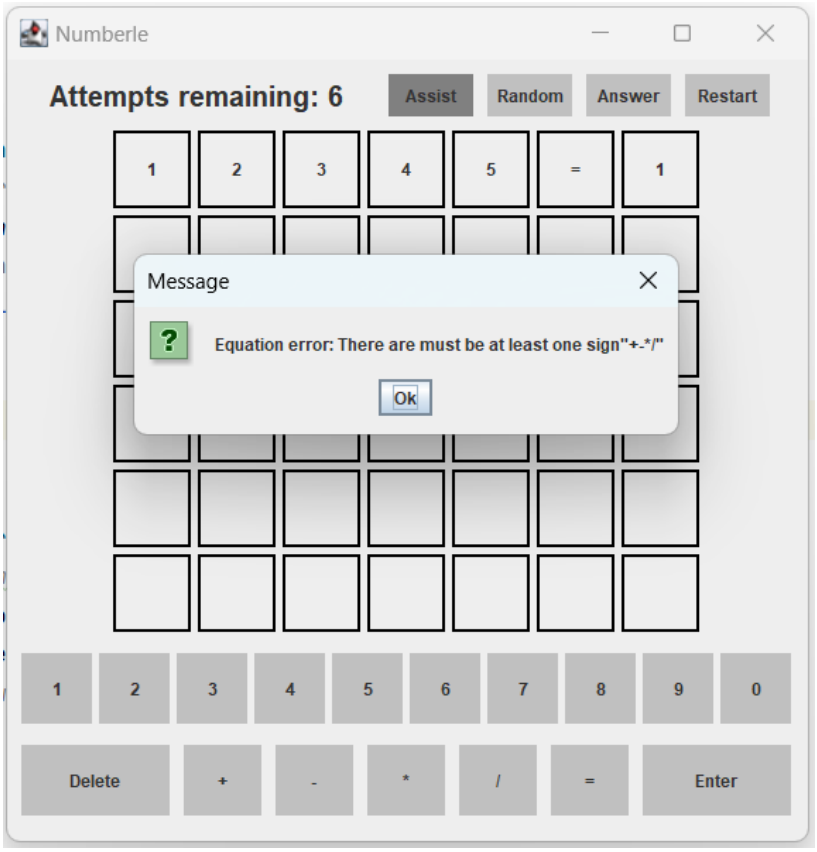

Figure 16. GUI Testing Diagram: No Math Symbol and Use Assist

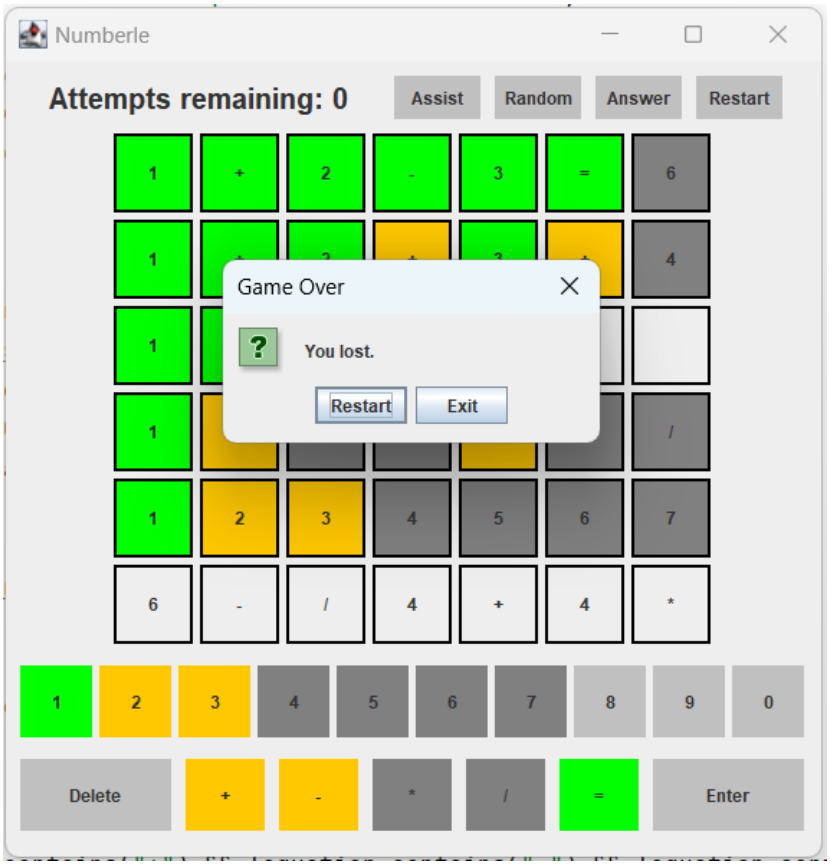

Figure 17. GUI Testing Diagram: Lost The Game

```
C:\Users\mi\.jdks\corretto-1.8.0_412\bin\java.exe ...

Process finished with exit code 0
```

Figure 18. GUI Testing Diagram: Exit (By Flick Window)

```
C:\Users\mi\.jdks\corretto-1.8.0_412\bin\java.exe ...
Welcome to the Numberle gaming system
----Menu----
Attempts remaining: 6
1.Display Equation
2.Input Equation
Please choose: 1
1+2-3=0
```

Figure 19. CLI Testing Diagram: Print Answer (Choose 1)

```
----Menu----
Attempts remaining: 6
1.Display Equation
2.Input Equation
Please choose: 2
Please input Equation: 1+2-3=0
Game over. You win!

New Game
----Menu----
Attempts remaining: 6
1.Display Equation
2.Input Equation
Please choose:
```

Figure 20. CLI Testing Diagram: Win The Game (Choose 2)

```
----Menu----
Attempts remaining: 6
1.Display Equation
2.Input Equation
Please choose: 2
Please input Equation: 1+2-3=6
The equations on both sides are not equal
```

Figure 21. CLI Testing Diagram: No Equal (Choose 2)

```
----Menu----
Attempts remaining: 6
1.Display Equation
2.Input Equation
Please choose: 2
Please input Equation: 1234567
No equal '=' sign
```

Figure 22. CLI Testing Diagram: No “=” (Choose 2)

```
----Menu----
Attempts remaining: 6
1.Display Equation
2.Input Equation
Please choose: 2
Please input Equation: 123
Too short
```

Figure 23. CLI Testing Diagram: No Enough Length (Choose 2)

```
Attempts remaining: 6
1.Display Equation
2.Input Equation
Please choose: 2
Please input Equation: 12345-+
Multiple math symbols in a row
```

Figure 24. CLI Testing Diagram: More Math Symbol (Choose 2)

```
----Menu----
Attempts remaining: 6
1.Display Equation
2.Input Equation
Please choose: 2
Please input Equation: 1234=46
There are must be at least one sign"+-*/"
```

Figure 25. CLI Testing Diagram: No Math Symbol (Choose 2)

```
----Menu----
Attempts remaining: 6
1.Display Equation
2.Input Equation
Please choose: 2
Please input Equation: 1+2+3=6
You guessed wrong
1--------The character is not being used.
+--------The character is not being used.
2--------The character is being used but in the wrong place.
+--------The character is not being used.
3--------The character is not being used.
=--------The character is correct.
6--------The character is correct.
```

Figure 26. CLI Testing Diagram: Incorrect Answer (Choose 2)

```
----Menu----
Attempts remaining: 5
1.Display Equation
2.Input Equation
3.Restart Game
Please choose: 2
Please input Equation: 1+2+4=7
You guessed wrong
1--------The character is correct.
+--------The character is correct.
2--------The character is correct.
+--------The character is being used but in the wrong place.
4--------The character is not being used.
=--------The character is correct.
7--------The character is not being used.
```

Figure 27. CLI Testing Diagram: Incorrect Answer and Display Choose 3 (Choose 2)

```
----Menu----
Attempts remaining: 1
1.Display Equation
2.Input Equation
3.Restart Game
4.Exit Game
Please choose: 2
Please input Equation: 1+1+1=3
Game over. You lost.

New Game
----Menu----
Attempts remaining: 6
1.Display Equation
2.Input Equation
Please choose:
```

Figure 28. CLI Testing Diagram: Lost The Game (Choose 2)

```
----Menu----
Attempts remaining: 5
1.Display Equation
2.Input Equation
3.Restart Game
4.Exit Game
Please choose: 3

New Game
----Menu----
Attempts remaining: 6
1.Display Equation
2.Input Equation
Please choose:
```

Figure 29. CLI Testing Diagram: Start New Game (Choose 3)

```
----Menu----
Attempts remaining: 5
1.Display Equation
2.Input Equation
3.Restart Game
4.Exit Game
Please choose: 4

Process finished with exit code 0
```

Figure 30. CLI Testing Diagram: Exit Game (Choose 4)

```
----Menu----
Attempts remaining: 6
1.Display Equation
2.Input Equation
Please choose: 5
Input error, please re-enter
```

Figure 31. CLI Testing Diagram: Choose Error

NumberleModelTest.java | UML.puml | NumberleModel.java | NumberleView.java

```java
public class NumberleModelTest {
    9 usages
    INumberleModel model=new NumberleModel();

    @Test
    public void processInputTest() {
        //Input correct answer, the new game will start, remaining attempts was updated to 6
        model.startNewGame();
        model.processInput("1+2-3=0");
        assertEquals( expected: 6, model.getRemainingAttempts());
    }

    @Test
    public void isGameOverTest() {
        //Input 6 incorrect equations in the game, it should be the end of the game
        model.initialize();
        for(int i=0;i<6;i++) {
            model.processInput("1+1+1=3");
        }
        assertTrue( message: "Game Over",model.isGameOver());
    }

    @Test
    public void isGameWonTest() {
        model.initialize();
        //Enter the correct equation in the game to win the game
        model.processInput("1+2-3=0");
        assertTrue( message: "Game Won",model.isGameWon());
    }
}
```

Figure 32. Junit Testing Diagram

3. Implementation

File - D:\programs\Java\coursework\src\coursework\CLIApp.java

```
1  package coursework;
2
3  /**
4   *
5   * @author Fay
6   */
7  import java.util.Scanner;
8
9  public class CLIApp {
10    public static void main(String[] args) {
11       // Initialize the Numberle model
12       INumberleModel model=new NumberleModel();
13       model.initialize();
14       // Create a scanner object for user input
15       Scanner sc=new Scanner(System.in);
16       System.out.println("Welcome to the Numberle gaming system");
17       //Loop when the current RemainingAttempts value is greater than 0
18       while(model.getRemainingAttempts()>0) {
19          System.out.println("----Menu----");
20          System.out.println("Attempts remaining: "+model.
    getRemainingAttempts());
21          System.out.println("1.Display Equation");
22          System.out.println("2.Input Equation");
23          //When entering a valid equation once, the game can be restarted, allow
     game restart once after a valid equation has been entered
24          if(model.getRemainingAttempts()<6) {
25             System.out.println("3.Restart Game");
26             System.out.println("4.Exit Game");
27          }
28          System.out.print("Please choose: ");
29          int choose=sc.nextInt();
30          //Display Equation
31          if(choose==1) {
32             System.out.println(model.getCurrentGuess());
33          }else if(choose==2) {//Input Equation
34             System.out.print("Please input Equation: ");
35             String equation=sc.next();
36             String check=model.checkEquationValidity(equation);
37             //Equation check passed and meets formatting requirements
38             if(check.equals("")) {
39                //Verify if two equations are consistent
40                model.processInput(equation);
41                // Check game status
42                if(model.isGameWon()) {
43                   System.out.println("Game over. You win!");
44                   model.startNewGame();
45                   model.setCurrentGuess();
46                   System.out.println();
47                   System.out.println("New Game");
```

```
48                  }else if(model.isGameOver()) {//Lost the game and initialized it
49                      System.out.println("Game over. You lost.");
50                      model.startNewGame();
51                      model.setCurrentGuess();
52                      System.out.println();
53                      System.out.println("New Game");
54                  }else {//Guessing incorrectly but the game is not over
55                      System.out.println("You guessed wrong");
56                      for(int i=0;i<equation.length();i++) {
57                          char green, orange, gray;
58                          if(equation.charAt(i)==model.getCurrentGuess().charAt(i)) {
59                              green = equation.charAt(i);
60                              System.out.println(green+"--------The character is correct."
    );
61                          } else if(model.getCurrentGuess().toString().contains(String.
    valueOf(equation.charAt(i)))) {
62                              orange = equation.charAt(i);
63                              System.out.println(orange+"--------The character is being
    used but in the wrong place.");
64                          }else {
65                              gray = equation.charAt(i);
66                              System.out.println(gray+"--------The character is not being
    used.");
67                          }
68                      }
69                  }
70              }else {
71                  System.out.println(check);
72              }
73          }else if(choose==3) {//When entering a valid equation once, the game
    can be restarted
74                  model.startNewGame();
75                  model.setCurrentGuess();
76                  System.out.println();
77                  System.out.println("New Game");
78          }else if(choose==4) {
79                  System.exit(0);
80
81          }else {
82              System.out.println("Input error, please re-enter");
83          }
84      }
85  }
86  }
87
```

```java
package coursework;

/**
 *
 * @author Fay
 */
import javax.swing.*;

public class GUIApp {
    public static void main(String[] args) {
        // Run the GUI creation on the event dispatch thread
        SwingUtilities.invokeLater(
                new Runnable() {
                    public void run() {
                        createAndShowGUI();
                    }
                }
        );
    }

    public static void createAndShowGUI() {
        // Create the Numberle model, controller, and view
        INumberleModel model = new NumberleModel();
        NumberleController controller = new NumberleController(model);
        NumberleView view = new NumberleView(model, controller);
    }
}
```

```java
package coursework;

/**
 *
 * @author Fay
 */
import javax.swing.*;
import javax.swing.border.LineBorder;

import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Observer;

public class NumberleView implements Observer {
    private final INumberleModel model;
    private final NumberleController controller;
    private final JFrame frame = new JFrame("Numberle");
    private final JLabel[][] inputLable = new JLabel[6][7];
    private final JButton[] buttons = new JButton[17];
    private final JLabel attemptsLabel = new JLabel("Attempts remaining: ");
//  private final JButton start = new JButton();
    private final JButton restart = new JButton("Restart");
    private final JButton assist = new JButton("Assist");
    private final JButton random = new JButton("Random");
    private final JButton answer = new JButton("Answer");
    private int row=0;//Current row
    private int col=0;//Current column
    String equation="";
    private boolean Assist = false;
    private boolean Random = false;

    public NumberleView(INumberleModel model, NumberleController
controller) {
        this.controller = controller;
        this.model = model;
        // Start a new game when the view is initialized
        this.controller.startNewGame();
        // Register this view as an observer of the model
        ((NumberleModel)this.model).addObserver(this);
        // Initialize the GUI frame
        initializeFrame();
        // Set this view as the view of the controller
        this.controller.setView(this);
        // Update the view with the initial state of the model
        update((NumberleModel)this.model, null);
    }

    public void initializeFrame() {
```

```
49          // Set the window close operation
50          frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
51          // Set the window size
52          frame.setSize(585, 600);
53          // Set the layout to null layout
54          frame.setLayout(null);
55          // Set the position and font style of the attempts remaining label
56          attemptsLabel.setBounds(30, 10, 400, 30);
57          attemptsLabel.setFont(new Font("Roman",Font.BOLD,20));
58          // Add the attempts remaining label to the window
59          frame.add(attemptsLabel);
60          // Add input labels array to the window
61          for(int i=0;i<inputLable.length;i++) {
62             for(int j=0;j<inputLable[i].length;j++) {
63                // Set the border, horizontal alignment, and position of the input label
64                inputLable[i][j]=new JLabel();
65                inputLable[i][j].setBorder(new LineBorder(Color.black,2));
66                inputLable[i][j].setHorizontalAlignment(SwingConstants.CENTER);
67                inputLable[i][j].setBounds(75+j*60, 50+i*60, 55, 55);
68                inputLable[i][j].setOpaque(true);
69                // Add the input label to the window
70                frame.add(inputLable[i][j]);
71             }
72          }
73          // Set the text array for buttons
74          String key[]= {"1","2","3","4","5","6","7","8","9","0","Delete","+","-","*",
    "/","=","Enter"};
75          // Add number buttons to the window
76          for(int i=0;i<10;i++) {
77             buttons[i]=new JButton(key[i]);
78             buttons[i].setBackground(Color.LIGHT_GRAY);
79             buttons[i].setBorder(null);
80             buttons[i].setBounds(10+i*55, 420, 50, 50);
81             frame.add(buttons[i]);
82          }
83          // Add the Delete button to the window
84          buttons[10]=new JButton(key[10]);
85          buttons[10].setBackground(Color.LIGHT_GRAY);
86          buttons[10].setBorder(null);
87          buttons[10].setBounds(10, 485, 105, 50);
88          frame.add(buttons[10]);
89          // Add operator buttons to the window
90          for(int i=11;i<16;i++) {
91             buttons[i]=new JButton(key[i]);
92             buttons[i].setBackground(Color.LIGHT_GRAY);
93             buttons[i].setBorder(null);
94             buttons[i].setBounds(125+(i-11)*65, 485, 55, 50);
95             frame.add(buttons[i]);
96          }
```

```
97          // Add the Enter button to the window
98          buttons[16]=new JButton(key[16]);
99          buttons[16].setBackground(Color.LIGHT_GRAY);
100         buttons[16].setBorder(null);
101         buttons[16].setBounds(450, 485, 105, 50);
102         // Add button click event handling
103         clickButton();
104         // Add the Enter button to the window
105         frame.add(buttons[16]);
106         frame.add(restart);
107         assist.setBounds(270, 10, 60, 30);
108         assist.setBackground(Color.LIGHT_GRAY);
109         assist.setBorder(null);
110         frame.add(assist);
111         random.setBounds(340, 10, 60, 30);
112         random.setBackground(Color.LIGHT_GRAY);
113         random.setBorder(null);
114         frame.add(random);
115         answer.setBounds(410, 10, 60, 30);
116         answer.setBackground(Color.LIGHT_GRAY);
117         answer.setBorder(null);
118         frame.add(answer);
119         restart.setBounds(480, 10, 60, 30);
120         restart.setBackground(Color.LIGHT_GRAY);
121         restart.setBorder(null);
122         restart.setEnabled(false);
123         frame.add(restart);
124         // Set the window visible
125         frame.setVisible(true);
126     }
127     //Button click event
128     public void clickButton() {
129         for(int i=0;i<17;i++) {
130             int x=i;
131             //Click the keyboard button
132             buttons[x].addActionListener(new ActionListener() {
133                 @Override
134                 public void actionPerformed(ActionEvent e) {
135                     String title=buttons[x].getText();
136                     //Click Delete
137                     if(title.equals("Delete")) {
138                         if(col>0) {//Delete last character
139                             col--;
140                             inputLable[row][col].setText("");
141                         }
142                     }else if(title.equals("Enter")) {//Click Enter
143                         equation="";
144                         //Read input characters
145                         for (int i = 0; i < 7; i++) {
```

```
146                     equation += inputLable[row][i].getText();
147                 }
148                 String check = controller.checkEquationValidity(equation);
149                 //Equation check passed and meets formatting requirements
150                 if (check.equals("")) {
151                     restart.setEnabled(true);//Can restart the game
152                     controller.processInput(equation);//Verify if two equations
    are consistent
153                 } else {
154                     if (Assist == true) {
155                         // If validation fails, display error message
156                         Object[] option = {"Ok"};
157                         JOptionPane.showOptionDialog(null,
158                                 "Equation error: " + check,
159                                 "Message",
160                                 JOptionPane.YES_NO_OPTION,
161                                 JOptionPane.QUESTION_MESSAGE,
162                                 null, option, option[0]);
163                     }else{
164                         restart.setEnabled(true);//Can restart the game
165                         controller.processInput(equation);//Verify if two equations
    are consistent
166                     }
167                 }
168             }else {//Click on numbers and arithmetic symbols
169                 if(col<7) {
170                     inputLable[row][col].setText(title);
171                     col++;
172                 }
173             }
174         }
175     });
176     }
177     //When clicking the restart once, the game can be restarted
178     restart.addActionListener(new ActionListener() {
179         @Override
180         public void actionPerformed(ActionEvent e) {
181             init();
182         }
183     });
184     //When clicking the Assist once, the player will get help
185     assist.addActionListener(new ActionListener() {
186         @Override
187         public void actionPerformed(ActionEvent e) {
188             if (Assist == false) {
189                 assist.setBackground(Color.GRAY);
190                 Assist = true;
191             }else{
192                 assist.setBackground(Color.lightGray);
```

```
193                 Assist = false;
194             }
195         }
196     });
197     //When clicking the Random once, the game can be restarted and have
    deferent answer
198     random.addActionListener(new ActionListener() {
199         @Override
200         public void actionPerformed(ActionEvent e) {
201             if (Random == false) {
202                 random.setBackground(Color.GRAY);
203                 Random = true;
204             }else{
205                 random.setBackground(Color.lightGray);
206                 Random = false;
207             }
208         }
209     });
210     //Display answer
211     answer.addActionListener(new ActionListener() {
212         @Override
213         public void actionPerformed(ActionEvent e) {
214             Object[] option = {"Ok"};
215             JOptionPane.showOptionDialog(null,
216                     "Answer is: " + controller.getCurrentGuess(),
217                     "Message",
218                     JOptionPane.YES_NO_OPTION,
219                     JOptionPane.QUESTION_MESSAGE,
220                     null, option, option[0]);
221         }
222     });
223 }
224
225 @Override
226 public void update(java.util.Observable o, Object arg) {
227     // Update the attempts remaining label with the current remaining
    attempts
228     attemptsLabel.setText("Attempts remaining: " + controller.
getRemainingAttempts());
229     // Check if the game is won
230     if (controller.isGameWon()) {
231         // Display win message and options to restart or exit
232         Object[] options = {"Restart", "Exit"};
233         int result = JOptionPane.showOptionDialog(
234                 null,
235                 "You win!",
236                 "Game Over",
237                 JOptionPane.YES_NO_OPTION,
238                 JOptionPane.QUESTION_MESSAGE,
```

```
239                 null, options, options[0]
240             );
241             // Handle user choice
242             if (result == 1) {
243                 System.exit(0);
244             }
245             init(); // Restart the game
246         } else if (controller.isGameOver()) {
247             // Check if the game is lost
248             // Display loss message and options to restart or exit
249             Object[] options = {"Restart", "Exit"};
250             int result = JOptionPane.showOptionDialog(
251                     null,
252                     "You lost.",
253                     "Game Over",
254                     JOptionPane.YES_NO_OPTION,
255                     JOptionPane.QUESTION_MESSAGE,
256                     null, options, options[0]
257             );
258             // Handle user choice
259             if (result == 1) {
260                 System.exit(0);
261             }
262             init(); // Restart the game
263         } else if (controller.getRemainingAttempts() < 6) { // Guessing
    incorrectly but the game is not over
264             // Set color for the equation
265             setColor(equation);
266             if (controller.getRemainingAttempts() > 1) {
267                 // If more than one attempt left, display options to show answer,
    start a new game, or continue
268                 Object[] options = {"Answer", "New Game", "Go on"};
269                 int n = JOptionPane.showOptionDialog(
270                         null,
271                         "You've got " + controller.getRemainingAttempts() + " more
    chances.",
272                         "You guessed wrong",
273                         JOptionPane.YES_NO_OPTION,
274                         JOptionPane.QUESTION_MESSAGE,
275                         null, options, options[0]);
276                 // Handle user choice
277                 if (n == 0) {
278                     Object[] option = {"Ok"};
279                     JOptionPane.showOptionDialog(null,
280                             "Answer is: " + controller.getCurrentGuess(),
281                             "Message",
282                             JOptionPane.YES_NO_OPTION,
283                             JOptionPane.QUESTION_MESSAGE,
284                             null, option, option[0]);
```

```
285                 col = 0;
286                 row++;
287             } else if (n == 1) {
288                 init(); // Restart the game
289             }else{
290                 col = 0;
291                 row++;
292             }
293         } else {
294             // If only one attempt left, display options to show answer, start a
    new game, or go on
295             Object[] options = {"Answer", "New Game", "Go On"};
296             int n = JOptionPane.showOptionDialog(null,
297                     "You've got " + controller.getRemainingAttempts() + " more
    chance.",
298                     "You guessed wrong",
299                     JOptionPane.YES_NO_OPTION,
300                     JOptionPane.QUESTION_MESSAGE,
301                     null, options, options[0]);
302             // Handle user choice
303             if (n == 0) {
304                 Object[] option = {"Ok"};
305                 JOptionPane.showOptionDialog(null,
306                         "Answer is: " + controller.getCurrentGuess(),
307                         "Message",
308                         JOptionPane.YES_NO_OPTION,
309                         JOptionPane.QUESTION_MESSAGE,
310                         null, option, option[0]);
311                 col = 0;
312                 row++;
313             } else if (n == 1) {
314                 init(); // Restart the game
315             }else{
316                 col = 0;
317                 row++;
318             }
319         }
320     }
321
322 }
323
324 //Start a new game
325 public void init() {
326     row=0;
327     col=0;
328     // Reset input labels
329     for(int i=0;i<inputLable.length;i++) {
330         for(int j=0;j<inputLable[i].length;j++) {
331             inputLable[i][j].setBackground(null);
```

```
332                 inputLable[i][j].setText("");
333             }
334         }
335         // Reset button colors
336         for(int i=0;i<buttons.length;i++) {
337             buttons[i].setBackground(Color.LIGHT_GRAY);
338         }
339         controller.getRemainingAttempts();
340         // Start a new game and set the current guess
341         if (Random == true){
342             controller.startNewGame();
343             controller.setCurrentGuess();
344         }else{
345             model.initialize();
346         }
347         // Update the attempts remaining label with the current remaining
    attempts
348         attemptsLabel.setText("Attempts remaining: " + controller.
    getRemainingAttempts());
349     }
350
351     //Set button color
352     public void setColor(String equation) {
353         for(int i=0;i<equation.length();i++) {
354             //Set the button with the correct position to green
355             if(equation.charAt(i)==controller.getCurrentGuess().charAt(i)) {
356                 inputLable[row][i].setBackground(Color.GREEN);
357                 for(int j=0;j<17;j++) {
358                     if(buttons[j].getText().equals(String.valueOf(equation.charAt(i
    )))) {
359                         buttons[j].setBackground(Color.GREEN);
360                     }
361                 }
362             }
363             //The button with incorrect position is set to orange
364             else if(controller.getCurrentGuess().toString().contains(String.valueOf
    (equation.charAt(i)))) {
365                 inputLable[row][i].setBackground(Color.ORANGE);
366                 for(int j=0;j<17;j++) {
367                     if(buttons[j].getText().equals(String.valueOf(equation.charAt(i
    )))) {
368                         buttons[j].setBackground(Color.ORANGE);
369                     }
370                 }
371             }
372             //Set non-existent buttons to gray
373             else {
374                 inputLable[row][i].setBackground(Color.GRAY);
375                 for(int j=0;j<17;j++) {
```

```
376                     if(buttons[j].getText().equals(String.valueOf(equation.charAt(i
    )))) {
377                         buttons[j].setBackground(Color.GRAY);
378                     }
379                 }
380             }
381         }
382     }
383 }
```

```java
package coursework;

/*
 *
 * @author Fay
 */
import java.util.Random;
import java.util.Stack;
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Observable;

public class NumberleModel extends Observable implements INumberleModel
 {
    private String targetNumber;
    private StringBuilder currentGuess;
    private int remainingAttempts;
    private boolean gameWon;

    @Override
    /**
     * Initializes the game by generating a random target number, setting the
default current guess,
     * and resetting the remaining attempts and game status.
     *
     @ensures The game is initialized with a random target number, default
current guess,
             maximum remaining attempts, and game status indicating not won.
     */
    public void initialize() {
        Random rand = new Random();
        targetNumber = Integer.toString(rand.nextInt(10000000));
        // Set the default current guess
        currentGuess = new StringBuilder("1+2-3=0");
        // Set the remaining attempts to the maximum allowed attempts
        remainingAttempts = MAX_ATTEMPTS;
        // Set the game status to indicate that the game has not been won yet
        gameWon = false;

//        setChanged();
//        notifyObservers();
    }

    @Override
    /**
     * Processes the user input by comparing it with the current guessing
```

```
46  equation.
47      * If the input equation matches the current guessing equation:
48      *   - Sets gameWon to true.
49      *   - Notifies observers.
50      *   - Returns true.
51      * If the input equation does not match the current guessing equation:
52      *   - Decrements remainingAttempts.
53      *   - Notifies observers.
54      *   - Returns false.
55      *
56      @param input The equation entered by the user.
57      @return true if the input equation matches the current guessing equation,
    false otherwise.
58      @requires input != null
59      @ensures
60          - If the input equation matches the current guessing equation:
61              - gameWon is set to true.
62              - Observers are notified.
63              - Returns true.
64          - If the input equation does not match the current guessing equation:
65              - remainingAttempts is decremented.
66              - Observers are notified.
67              - Returns false.
68      */
69      public boolean processInput(String input) {
70          // Ensure that the input is not null
71          assert input != null : "Input must not be null";
72
73          // Check if the input equation matches the current guessing equation
74          boolean result;
75          if(input.equals(currentGuess.toString())) {
76              // If matches, set gameWon to true, notify observers, and return true
77              gameWon = true;
78              setChanged();
79              notifyObservers();
80              result = true;
81          }else {
82              // If doesn't match, decrement remainingAttempts, notify observers, and
    return false
83              remainingAttempts--;
84              setChanged();
85              notifyObservers();
86              result = false;
87          }
88
89          // Ensure that remaining attempts is non-negative
90          assert remainingAttempts >= 0 : "Remaining attempts should be non-
    negative";
91
```

```
92          return result;
93      }
94
95      @Override
96      /**
97       * Checks if the game is over.
98       @return true if the game is over (either remainingAttempts is less than or
    equal to 0 or gameWon is true), false otherwise.
99       */
100     public boolean isGameOver() {
101
102         // Ensure that remaining attempts is non-negative
103         assert remainingAttempts >= 0 : "Remaining attempts should be non-
    negative";
104
105         // Check if the game is over (either remainingAttempts is less than or
    equal to 0 or gameWon is true)
106         boolean result = remainingAttempts <= 0 || gameWon;
107
108         //Ensure result is a boolean
109         assert result == true || result == false: "result is not a boolean";
110
111         return result;
112     }
113
114     @Override
115     public boolean isGameWon() {
116         //Checks if the game has been won.
117         //return true if the game has been won, false otherwise.
118         return gameWon;
119     }
120
121     @Override
122     public String getTargetNumber() {
123         //Returns the target number of the game.
124         return targetNumber;
125     }
126
127     @Override
128     public StringBuilder getCurrentGuess() {
129         //Returns the current guess of the player.
130         return currentGuess;
131     }
132
133     @Override
134     public int getRemainingAttempts() {
135         //Returns the number of remaining attempts in the game.
136         return remainingAttempts;
137     }
```

```
138
139     @Override
140     /**
141      * Starts a new game by initializing game parameters.
142      @ensure The game parameters are initialized.
143      @invariant The game parameters are valid after initialization.
144      */
145     public void startNewGame() {
146         initialize();
147     }
148
149     /**
150      * Retrieves an equation from a file and returns it as a StringBuilder object.
151      *
152      @return a StringBuilder object representing an equation retrieved from a
    file.
153      @requires The file "src/equations.txt" exists and contains valid equations.
154      @ensures The returned StringBuilder object contains a valid equation
    from the file.
155      */
156     public StringBuilder getEquation() {
157         // Create a list to store equations
158         ArrayList<String> list=new ArrayList<String>();
159         BufferedReader br = null;
160         try {
161             // Read equations from the file and add them to the list
162             br = new BufferedReader(new FileReader("src/equations.txt"));
163             String line = null;
164             while ((line = br.readLine()) != null) {
165                 list.add(line);
166             }
167         } catch (FileNotFoundException e) {
168             // Handle file not found exception
169             e.printStackTrace();
170         } catch (IOException e) {
171             // Handle IO exception
172             e.printStackTrace();
173         }
174
175         // Ensure that the list of equations is not empty
176         assert !list.isEmpty() : "Equation list must not be empty";
177
178         // Select a random equation from the list and return it as a StringBuilder
    object
179         return new StringBuilder(list.get(new Random().nextInt(list.size())));
180     }
181
182     @Override
183     /**
```

```
184     * Sets the current guess by retrieving an equation from a file and assigning it to the current guess.
185     * The new current guess replaces the previous one.
186     *
187     @requires The file "src/equations.txt" exists and contains valid equations.
188     @ensures The current guess is set to a valid equation retrieved from the file.
189     */
190    public void setCurrentGuess() {
191        //Sets the current guess by retrieving an equation from a file and assigning it to the current guess.
192        this.currentGuess=getEquation();
193    }
194    @Override
195    /**
196     * Checks the validity of an equation.
197     *
198     @param equation The equation to be checked.
199     @return a String indicating the validity of the equation:
200       - "Too short" if the equation length is not equal to 7.
201       - "Multiple math symbols in a row" if there are consecutive arithmetic operators.
202       - "No equal sign" if the equation does not contain an equal sign.
203       - "There must be at least one sign" if the equation does not contain any arithmetic symbols.
204       - "The equations on both sides are not equal" if the equations on both sides of the equal sign are not equal.
205       - Empty string ("") if the equation is valid.
206     @requires equation != null
207     @ensures The returned string indicates the validity of the equation.
208     */
209    public String checkEquationValidity(String equation) {
210
211        // Ensure that the equation is not null
212        assert equation != null : "Equation must not be null";
213
214        // Check the length of the equation
215        if (equation.length() != 7) {
216            return "Too short";
217        }
218
219        //Determine if there are consecutive arithmetic operators present
220        String operators = "+-*/=";
221        boolean lastWasOperator = false;
222        for (char c : equation.toCharArray()) {
223            boolean isOperator = operators.indexOf(c) != -1;
224            if (isOperator && lastWasOperator) {
225                return "Multiple math symbols in a row";
226            }
```

```
227             lastWasOperator = isOperator;
228         }
229         //Determine if it contains an equal sign
230         if (!equation.contains("=")) {
231             return "No equal '=' sign";
232         }
233         //Determine if it contains arithmetic symbols
234         if (!equation.contains("+") && !equation.contains("-") && !equation.
    contains("*") && !equation.contains("/")) {
235             return "There are must be at least one sign\"+-*/\"";
236         }
237 
238         String[] expression = equation.split("=");
239 
240         double left = calculateExpression(expression[0]);
241         double right = calculateExpression(expression[1]);
242 
243         //The equations on both sides are not equal
244         if (left != right) {
245             return "The equations on both sides are not equal";
246         }
247         //All checks passed, return ""
248         return "";
249     }
250 
251     /**
252      * Calculates the result of a mathematical expression.
253      *
254      @param expression The mathematical expression to be evaluated.
255      @return the result of the mathematical expression.
256      @requires expression != null
257      @ensures The returned value is the result of evaluating the expression.
258      */
259     public static double calculateExpression(String expression) {
260 
261         // Ensure that the expression is not null
262         assert expression != null : "Expression must not be null";
263 
264         // Initialize stacks to store numbers and operators
265         Stack<Double> numbers = new Stack<>();
266         Stack<Character> operators = new Stack<>();
267         StringBuilder number = new StringBuilder();
268 
269         // Iterate through each character in the expression
270         for (char ch : expression.toCharArray()) {
271             if (Character.isDigit(ch)) {
272                 //If the character is a number, append it to the current number
    string
273                 number.append(ch);
```

```
274             } else {
275                 //If the character is an operator or the end of an expression,
    perform the calculation
276                 numbers.push(Double.parseDouble(number.toString()));
277                 number.setLength(0);
278                 //If the operator stack is not empty and the top operator priority is
    higher than or equal to the current operator (or if the stack is empty)
279                 //Then perform the operation until a higher priority operator is
    found or the stack is empty
280                 while (!operators.isEmpty() && hasPrecedence(ch, operators.peek
    ())) {
281
282                     assert numbers.size() >= 2 : "Insufficient operators to perform
    operation";
283
284                     numbers.push(operate(operators.pop(), numbers.pop(), numbers.
    pop()));
285                 }
286                 // Push the current operator onto the stack
287                 operators.push(ch);
288             }
289         }
290         //Handle the last number and remaining operators
291         numbers.push(Double.parseDouble(number.toString()));
292         while (!operators.isEmpty()) {
293
294             assert numbers.size() >= 2 : "Insufficient operators to perform
    operation";
295
296             numbers.push(operate(operators.pop(), numbers.pop(), numbers.pop
    ()));
297         }
298         return numbers.pop();
299     }
300
301     /**
302      * Determines if one operator has higher precedence than another.
303      *
304      @param op1 The first operator.
305      @param op2 The second operator.
306      @return true if op1 has higher precedence than op2, false otherwise.
307      @requires op1 and op2 are valid arithmetic operators ('+', '-', '*', '/').
308      @ensures The returned value indicates whether op1 has higher precedence
     than op2.
309      */
310     private static boolean hasPrecedence(char op1, char op2) {
311         // Check if op1 has higher precedence than op2 based on operator
    priority
312         if ((op1 == '*' || op1 == '/') && (op2 == '+' || op2 == '-')) return false;
```

```
313         return true;
314     }
315
316     /**
317      * Performs arithmetic operation based on the specified operator and
    operands.
318      *
319      @param op The operator specifying the arithmetic operation.
320      @param b The second operand.
321      @param a The first operand.
322      @return the result of the arithmetic operation.
323      @requires op is a valid arithmetic operator ('+', '-', '*', '/').
324      @ensures The returned value is the result of performing the arithmetic
    operation.
325      */
326     private static double operate(char op, double b, double a) {
327         // Perform arithmetic operation based on the specified operator
328         switch (op) {
329             case '+': return a + b;
330             case '-': return a - b;
331             case '*': return a * b;
332             case '/': return a / b;
333         }
334         // Default case: return 0 if operator is not recognized
335         return 0;
336     }
337 }
338
```

File - D:\programs\Java\coursework\src\coursework\INumberleModel.java

```java
package coursework;

/**
 *
 * @author Fay
 */
import java.util.List;

public interface INumberleModel {
    int MAX_ATTEMPTS = 6;

    // Initializes the game
    void initialize();
    // Processes the input equation provided by the user
    boolean processInput(String input);
    // Checks if the game is over
    boolean isGameOver();
    // Checks if the game is won
    boolean isGameWon();
    // Gets the target number for the game
    String getTargetNumber();
    // Gets the current guess made by the player
    StringBuilder getCurrentGuess();
    // Gets the remaining attempts allowed in the game
    int getRemainingAttempts();
    // Starts a new game
    void startNewGame();
    // Sets the current guess based on generated equation
    void setCurrentGuess();
    // Checks the validity of the equation provided by the user
    String checkEquationValidity(String equation);
}
```

File - D:\programs\Java\coursework\src\coursework\NumberleModelTest.java

```java
package coursework;

/**
 *
 * @author Fay
 */
import org.junit.Test;
import static org.junit.Assert.*;


public class NumberleModelTest {
    INumberleModel model=new NumberleModel();

    @Test
    public void processInputTest() {
        //Input correct answer, the new game will start, remaining attempts was updated to 6
        model.startNewGame();
        model.processInput("1+2-3=0");
        assertEquals(6, model.getRemainingAttempts());
    }

    @Test
    public void isGameOverTest() {
        //Input 6 incorrect equations in the game, it should be the end of the game
        model.startNewGame();
        for(int i=0;i<6;i++) {
            model.processInput("1+1+1=3");
        }
        assertTrue("Game Over",model.isGameOver());
    }

    @Test
    public void isGameWonTest() {
        model.initialize();
        //Enter the correct equation in the game to win the game
        model.processInput("1+2-3=0");
        assertTrue("Game Won",model.isGameWon());
    }
}

```

File - D:\programs\Java\coursework\src\coursework\NumberleController.java

```java
package coursework;

/**
 *
 * @author Fay
 */
public class NumberleController {
    private INumberleModel model;
    private NumberleView view;

    // Constructor to initialize with a model
    public NumberleController(INumberleModel model) {
        this.model = model;
    }

    // Method to set the associated view
    public void setView(NumberleView view) {
        this.view = view;
    }

    // Method to process user input
    public void processInput(String input) {
        model.processInput(input);
    }

    // Method to check if the game is over
    public boolean isGameOver() {
        return model.isGameOver();
    }

    // Method to check if the game is won
    public boolean isGameWon() {
        return model.isGameWon();
    }

    // Method to get the current guess made by the player
    public StringBuilder getCurrentGuess() {
        return model.getCurrentGuess();
    }

    // Method to get the remaining attempts allowed in the game
    public int getRemainingAttempts() {
        return model.getRemainingAttempts();
    }

    // Method to start a new game
    public void startNewGame() {
        model.startNewGame();
    }
```

```
50
51      // Method to check the validity of the equation provided by the user
52      public String checkEquationValidity(String equation) {
53          return model.checkEquationValidity(equation);
54      }
55
56      // Method to set the current guess based on generated equation
57      public void setCurrentGuess() {
58          model.setCurrentGuess();
59      }
60  }
```