



DELVE

User Guide

Contents

1.	Introduction	3
2.	Loading the snapshots to analyze	5
3.	Aggregated Results (classpath is not configured yet)	10
4.	Individual Results (classpath is not configured yet)	11
5.	Program Execution	15
6.	Individual Results (classpath is configured)	32
7.	Code Instrumentation	36
7.1	Instrumentation Overview	39
7.2	Step 1: Program State	40
7.3	Step 2: Replace GUI with non-GUI components (bypass UI behavior)	51
7.4	Step 3: Get rid of user-interaction code	66
7.5	Step 4: Replace random with deterministic behavior	74
7.6	Step 5: Save the program state	82
7.7	Step 6: Provide a correct implementation as reference	102
7.8	Step 7: Provide additional libraries (if needed)	105
8.	Instrumentation Analysis Results	107

1. Introduction

Delve is a software tool designed for programming course instructors. It can analyze programs written in Java, aggregate the results and present an overview about the class performance in any given assignment. Although Delve is intended for CS1 and CS2 courses it can be used in any programming-oriented course.

Although Delve can analyze just the final submission, its strength lies on analyzing the program as it progressed over time, from start to final submission. Therefore, it is recommended that you collect every program variation over time (referred from now on as ‘code snapshots’ or simply ‘snapshots’) for every student and put them in a folder. To avoid conflicts in the file names, each snapshot uses the following naming convention:

```
<student ID>_<program name>_<snapshot number; ascending order>.java
```

For example, if student A produced 40 snapshots for a program Foo.java, the snapshots should be named:

```
a_Foo_01.java (first snapshot)  
a_Foo_02.java  
...  
a_Foo_40.java (last snapshot/submitted program)
```

All snapshots can be dumped under the same directory. Delve will automatically sort them to subfolders (one subfolder per student who submitted a program).

Delve can perform a time analysis and determine how long it took to complete the assignment. To do so, each snapshot should be accompanied with a timestamp. The snapshot timestamps appear in file **timestamps.txt**. Each line in this file has the following format:

```
<snapshot name>#<timestamp in sec>
```

The timestamp must be expressed in seconds. Usually the timestamp is the difference, measured in seconds, between the snapshot creation time and midnight, January 1, 1970 UTC. However, it can be anything as long as it uses the first snapshot as reference.

For example, both:

```
foo_Bar_01.java#1609470315  
foo_Bar_02.java#1609470326  
...  
foo_Bar_20.java#1609471626
```

and

```
foo_Bar_01.java#0  
foo_Bar_02.java#11  
...  
foo_Bar_20.java#1311
```

are the same.

Similar to the snapshots which can be dumped in the same folder, there can b a single **timestamps.txt** with all snapshot timestamps, one per line, and in any order.

Delve will find automatically the timestamps that correspond to the snapshots of a given student and create a file **timestamps.txt** in the student's subfolder.

The following sections will guide you step-by-step how to use Delve.

2. Loading the snapshots to analyze

When Delve first launches the following window shows up:

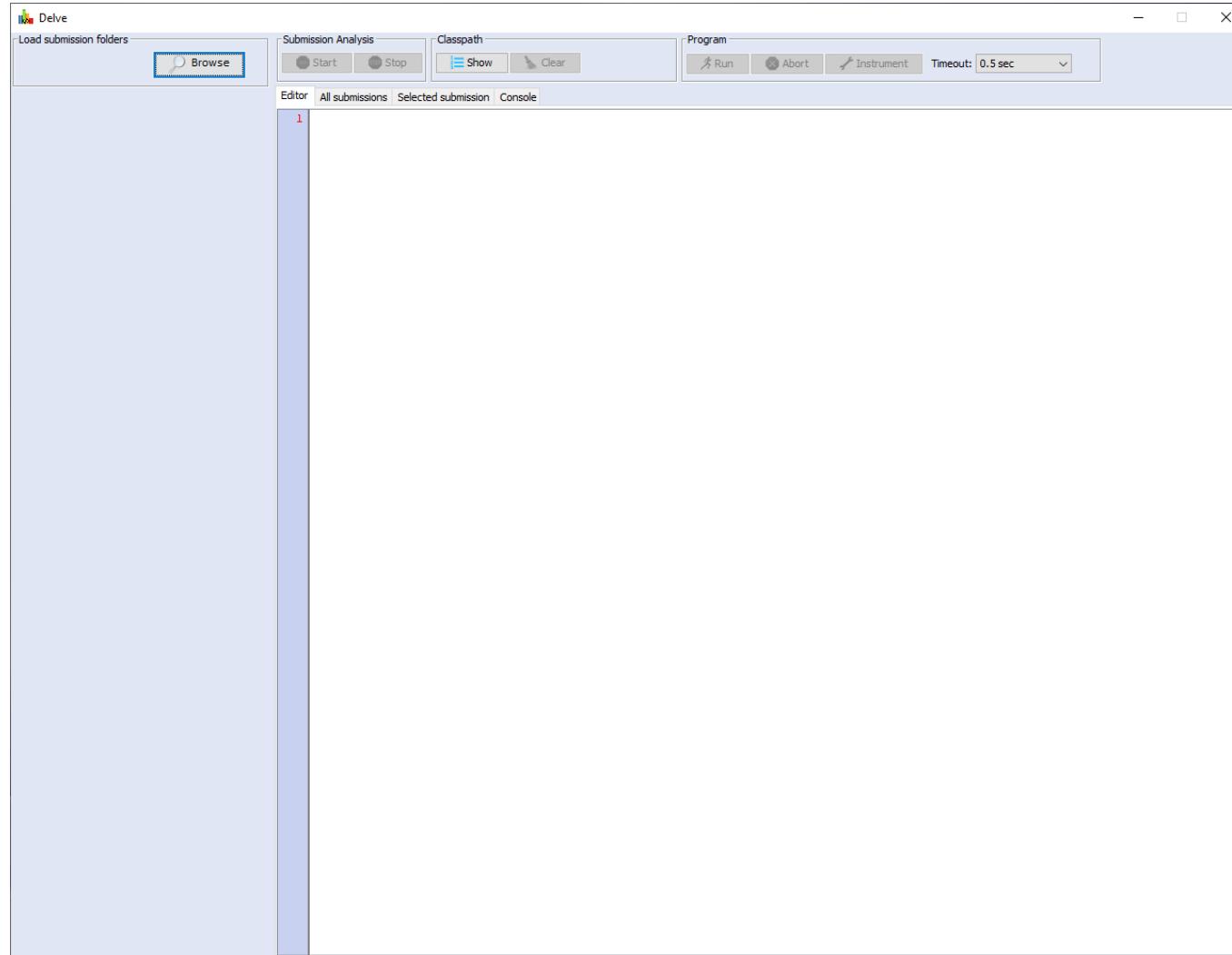


Figure 1: Delve upon launch.

Now, the user must select the root directory with the student snapshots. The root directory does not necessarily have to be organized (i.e., individual subfolders with the code snapshots and timestamps for every student). Delve will do that.

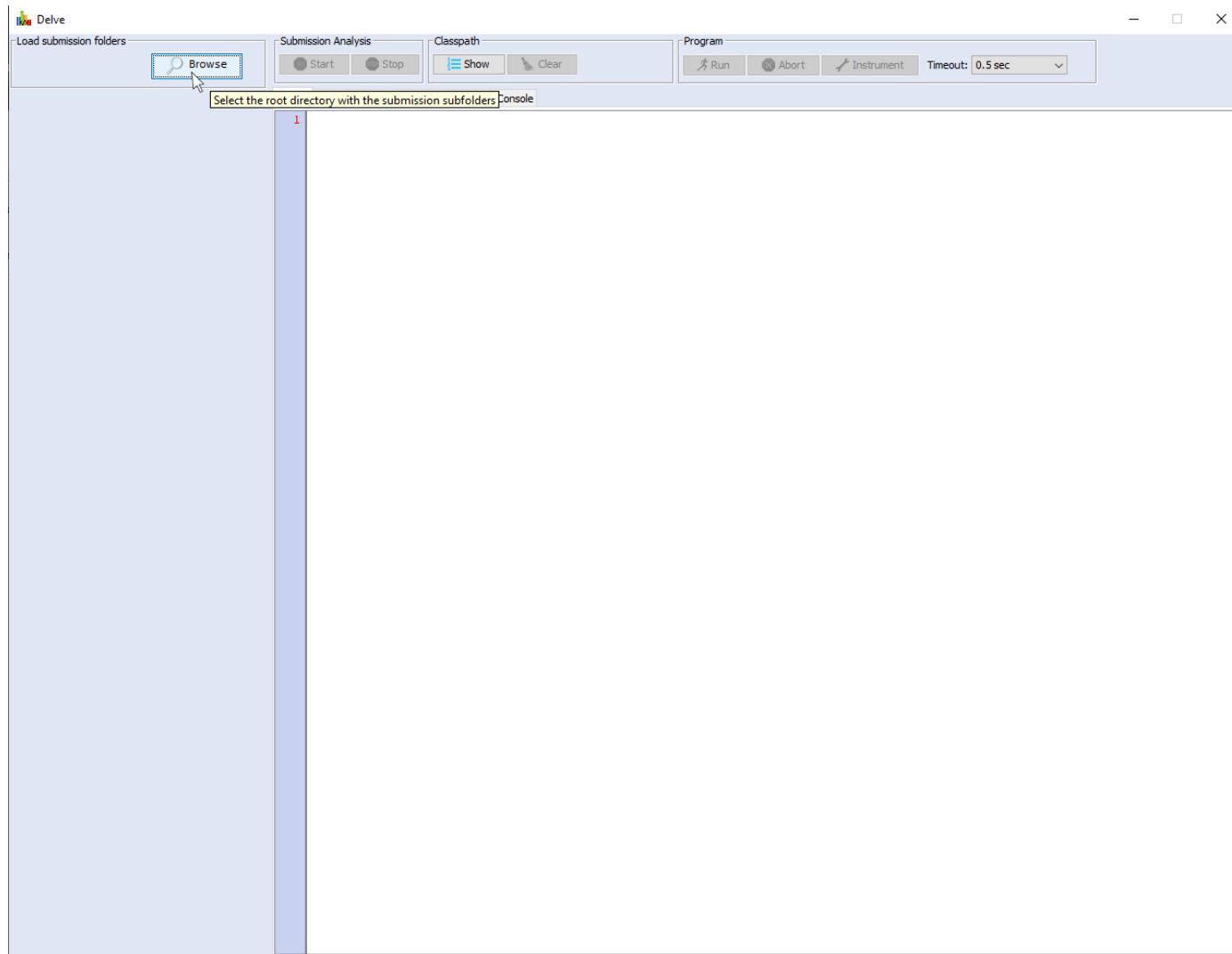


Figure 2: The user needs to click first on the *Browse* button to locate the root directory with the student code snapshots.

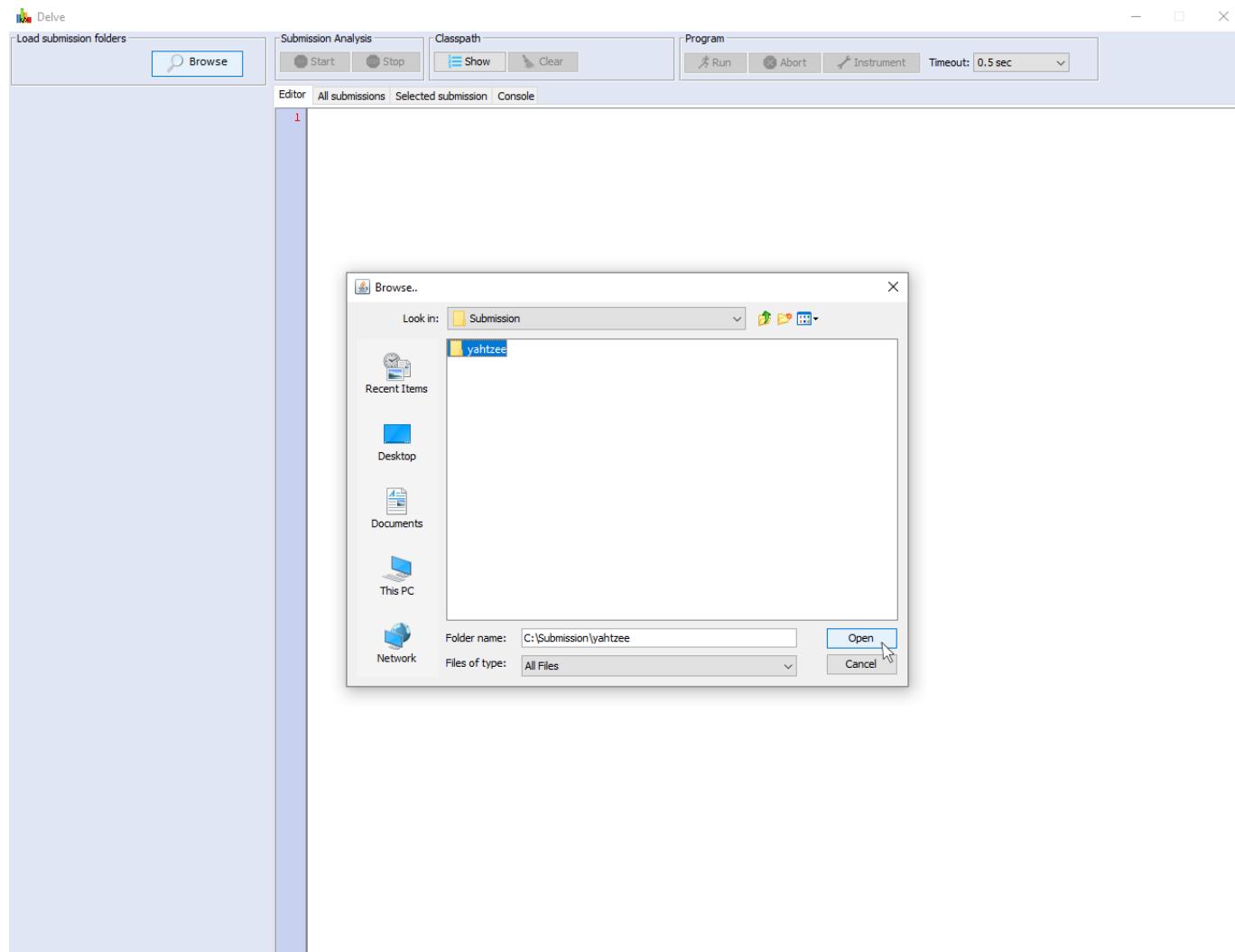


Figure 3: The user can now locate the root directory with the student code snapshots.

At this point, Delve processes the student code snapshots and their timestamps, organizes them into subfolders (one per student) and then analyzes those subfolders to find out how long it took every student to complete the assignment.

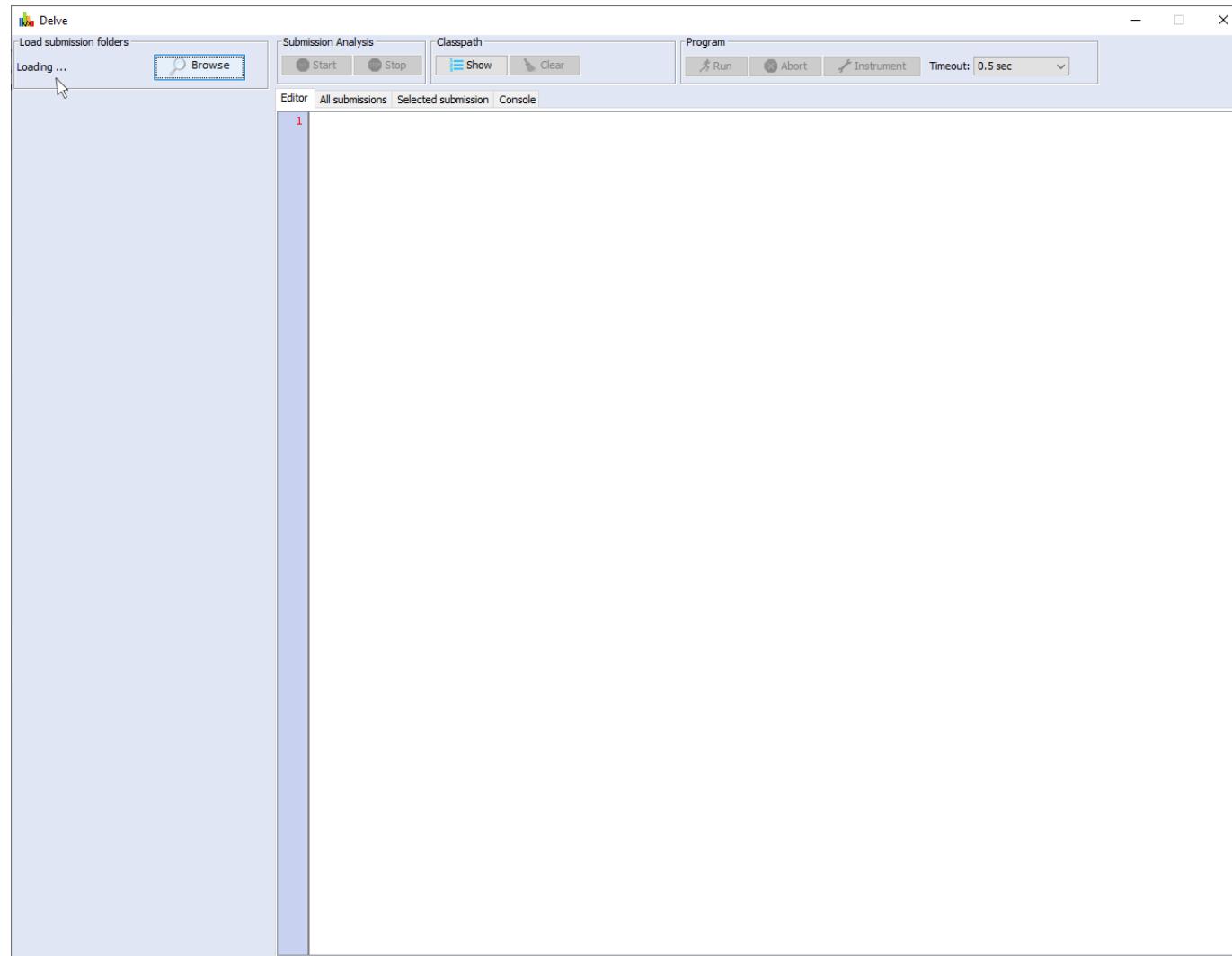


Figure 4: Delve organizes and then processes the student snapshots. This takes a few seconds, up to a minute.

Once done, we can see the student submissions on the left. Delve has created a subfolder for every student, put there the corresponding snapshots and created the file with their timestamps.

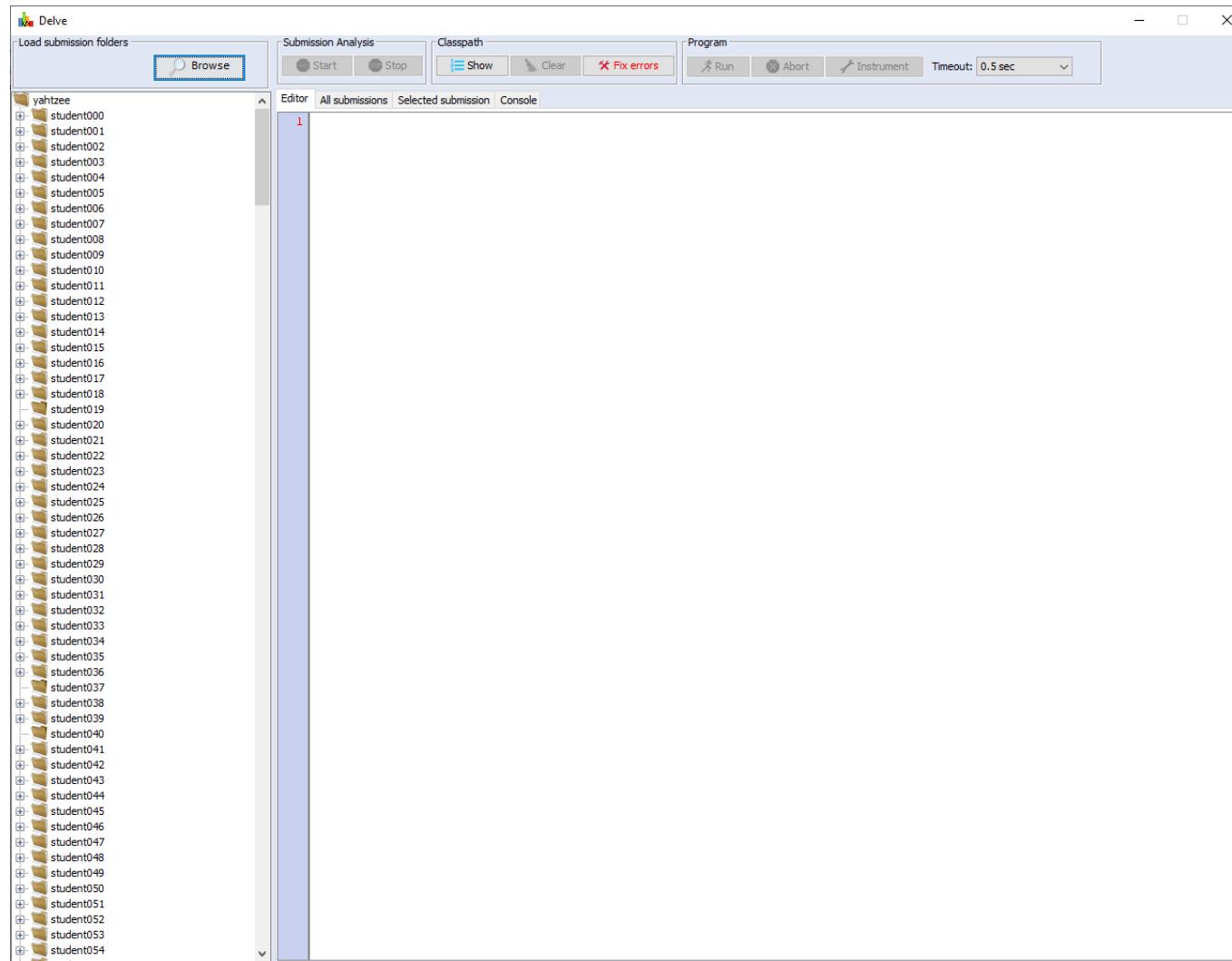


Figure 5: Once the snapshots are loaded they are organized into subfolders and presented on the left.

3. Aggregated Results (classpath is not configured yet)

Tab **All Submissions** displays the aggregated results in the form of histograms.

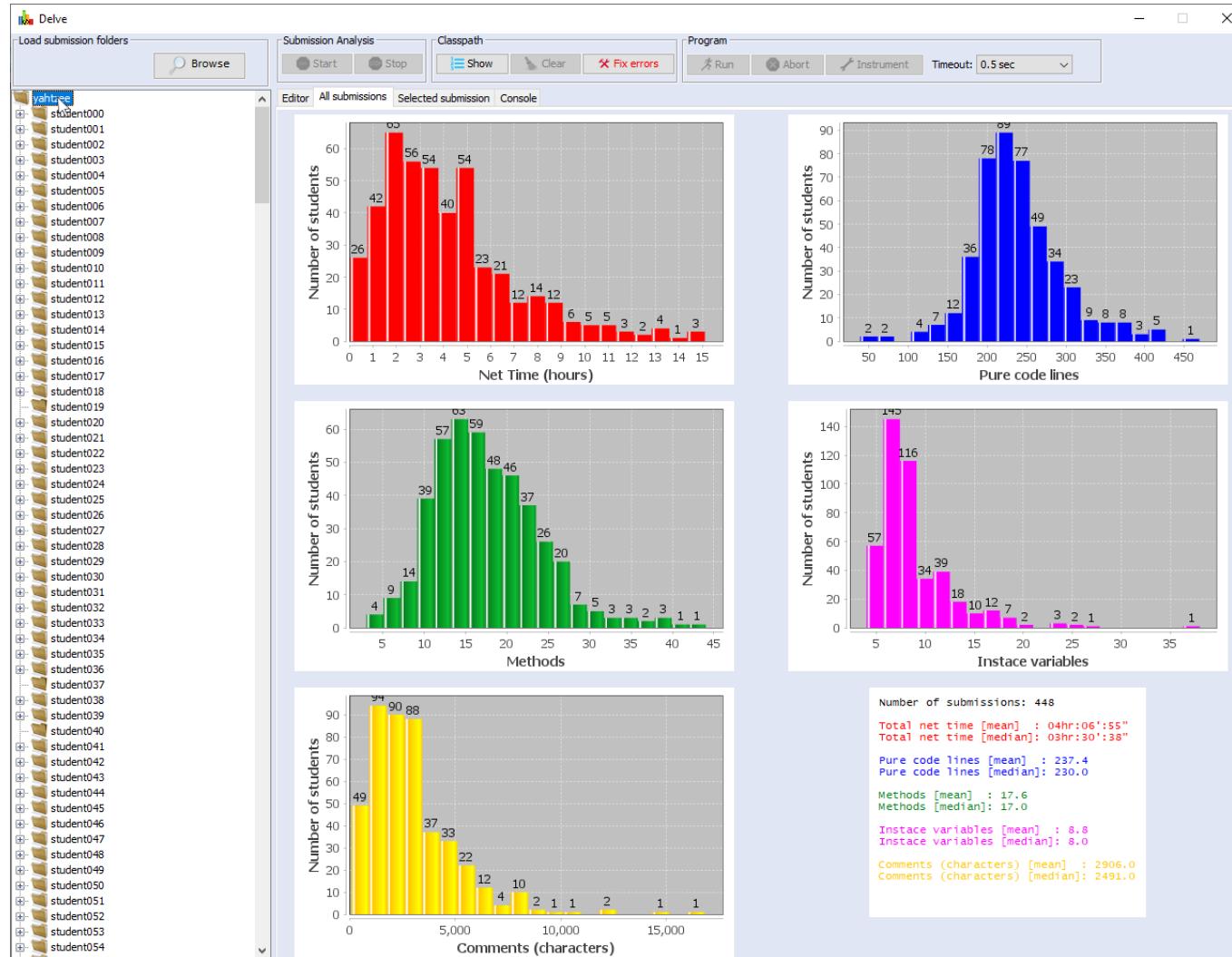


Figure 6: Aggregated histograms (time to complete the assignment, number of lines, methods, ivars and comments).

4. Individual Results (classpath is not configured yet)

The **Editor** tab shows a detailed breakdown of the time spent on the assignment by this particular student.

The screenshot shows the Delve tool interface with the 'Editor' tab selected. The left pane displays a tree view of submission folders under 'yahtzee'. The right pane shows a detailed breakdown of time spent on various Java files for student009. The output is as follows:

```
1 Number of snapshots: 150
2
3 Total time: 04hr:33':02"
4 Last snapshot: student009_Yahtzee_150.java
5 Number of lines: 286
6 Number of methods: 12
7
8 Time breakdown:
9   Filename           Time Diff      Total Time
10  student009.Yahtzee_001.java  start
11  student009.Yahtzee_002.java  7':58"  00hr:07':58"
12  student009.Yahtzee_003.java  1':07"  00hr:09':05"
13  student009.Yahtzee_004.java  b r e a k
14  student009.Yahtzee_005.java  b r e a k
15  student009.Yahtzee_006.java  b r e a k
16  student009.Yahtzee_007.java  2':29"  00hr:11':34"
17  student009.Yahtzee_008.java  4':14"  00hr:15':48"
18  student009.Yahtzee_009.java  9':24"  00hr:25':12"
19  student009.Yahtzee_010.java  1':00"  00hr:26':12"
20  student009.Yahtzee_011.java  0':57"  00hr:27':09"
21  student009.Yahtzee_012.java  2':08"  00hr:29':17"
22  student009.Yahtzee_013.java  0':57"  00hr:30':14"
23  student009.Yahtzee_014.java  1':30"  00hr:31':44"
24  student009.Yahtzee_015.java  2':33"  00hr:34':17"
25  student009.Yahtzee_016.java  b r e a k
26  student009.Yahtzee_017.java  b r e a k
27  student009.Yahtzee_018.java  b r e a k
28  student009.Yahtzee_019.java  b r e a k
29  student009.Yahtzee_020.java  b r e a k
30  student009.Yahtzee_021.java  9':11"  00hr:43':28"
31  student009.Yahtzee_022.java  4':35"  00hr:48':03"
32  student009.Yahtzee_023.java  4':18"  00hr:52':21"
33  student009.Yahtzee_024.java  7':15"  00hr:59':36"
34  student009.Yahtzee_025.java  0':27"  01hr:00':03"
35  student009.Yahtzee_026.java  1':01"  01hr:01':04"
36  student009.Yahtzee_027.java  5':39"  01hr:06':43"
37  student009.Yahtzee_028.java  2':10"  01hr:08':53"
38  student009.Yahtzee_029.java  0':22"  01hr:09':15"
39  student009.Yahtzee_030.java  0':29"  01hr:09':44"
40  student009.Yahtzee_031.java  1':53"  01hr:11':37"
41  student009.Yahtzee_032.java  1':27"  01hr:13':04"
42  student009.Yahtzee_033.java  2':13"  01hr:15':17"
43  student009.Yahtzee_034.java  3':41"  01hr:18':58"
44  student009.Yahtzee_035.java  3':14"  01hr:22':12"
45  student009.Yahtzee_036.java  1':08"  01hr:23':20"
46  student009.Yahtzee_037.java  0':37"  01hr:23':57"
47  student009.Yahtzee_038.java  1':18"  01hr:25':15"
48  student009.Yahtzee_039.java  0':50"  01hr:26':05"
49  student009.Yahtzee_040.java  2':16"  01hr:28':21"
50  student009.Yahtzee_041.java  0':27"  01hr:28':48"
51  student009.Yahtzee_042.java  0':22"  01hr:29':10"
52  student009.Yahtzee_043.java  0':27"  01hr:29':37"
53  student009.Yahtzee_044.java  2':54"  01hr:32':31"
54  student009.Yahtzee_045.java  0':33"  01hr:33':04"
55  student009.Yahtzee_046.java  0':26"  01hr:33':30"
56  student009.Yahtzee_047.java  b r e a k
57  student009.Yahtzee_048.java  1':10"  01hr:34':40"
58  student009.Yahtzee_049.java  b r e a k
59  student009.Yahtzee_050.java  4':26"  01hr:39':06"
60  student009.Yahtzee_051.java  2':10"  01hr:41':16"
61  student009.Yahtzee_052.java  1':09"  01hr:42':25"
62  student009.Yahtzee_053.java  0':22"  01hr:42':51"
```

Figure 7:Detailed breakdown of the time spent on the assignment by a particular student.

After pressing ***Start***, Delve analyzes the student snapshots and display how several features progress over time.

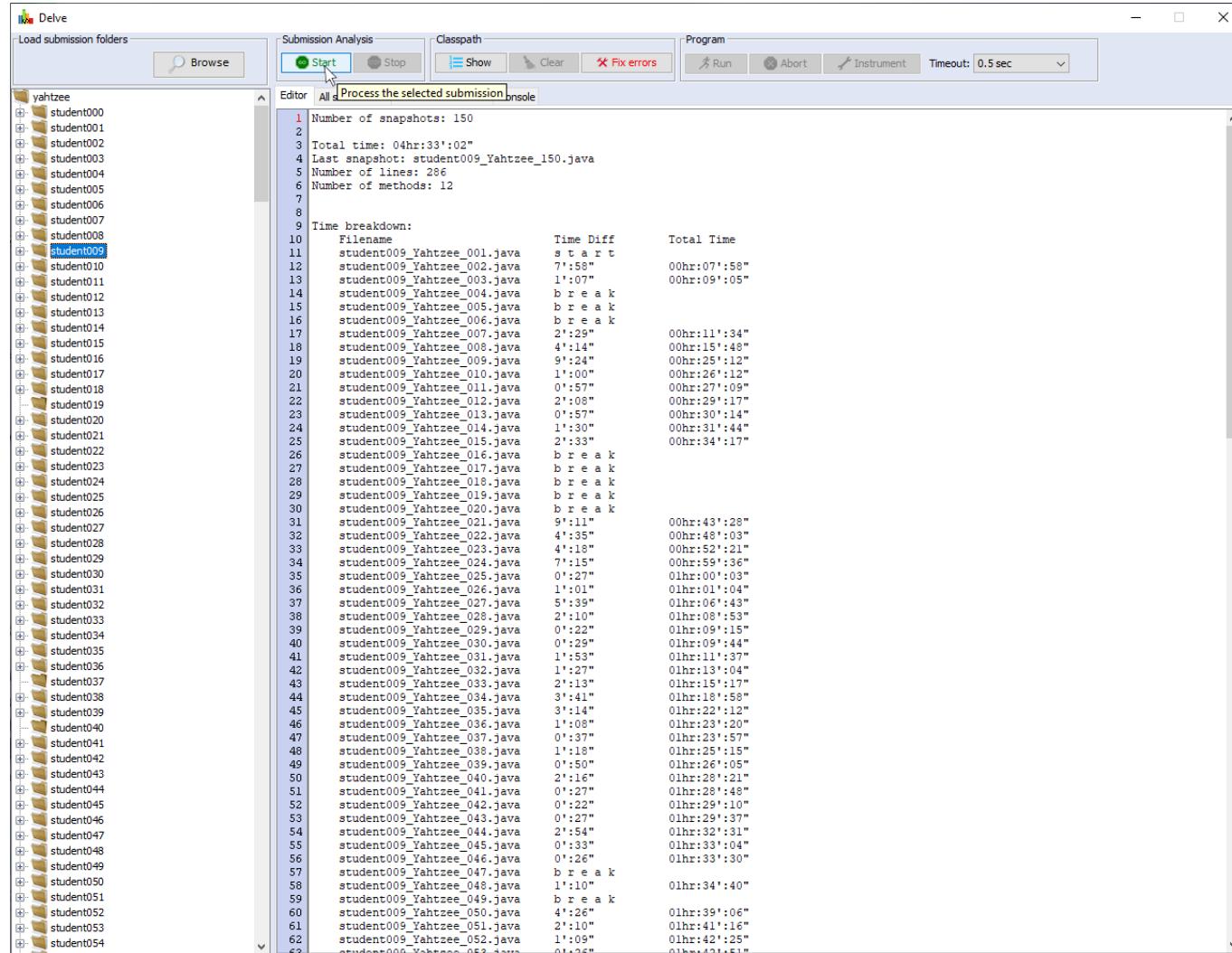


Figure 8: As soon as the user clicks on the *Start* button, Delve begins analyzing the selected submission.

The screenshot shows the Delve tool interface with the following details:

- Load submission folders:** A tree view showing submissions from students 000 to 054, with **student009** selected.
- Submission Analysis:** Tab selected, showing the following output:


```

1 Number of snapshots: 150
2
3 Total time: 04hr:33'02"
4 Last snapshot: student009_Yahtzee_150.java
5 Number of lines: 286
6 Number of methods: 12
7
8 Time breakdown:
9   Filename           Time Diff    Total Time
10  student009_Yahtzee_001.java  s t a r t
11  student009_Yahtzee_002.java  7':58"  00hr:07':58"
12  student009_Yahtzee_003.java  1':07"  00hr:09':05"
13  student009_Yahtzee_004.java  b r e a k
14  student009_Yahtzee_005.java  b r e a k
15  student009_Yahtzee_006.java  b r e a k
16  student009_Yahtzee_007.java  2':29"  00hr:11':34"
17  student009_Yahtzee_008.java  4':14"  00hr:15':48"
18  student009_Yahtzee_009.java  9':24"  00hr:25':12"
19  student009_Yahtzee_010.java  1':00"  00hr:26':12"
20  student009_Yahtzee_011.java  0':57"  00hr:27':09"
21  student009_Yahtzee_012.java  2':08"  00hr:29':17"
22  student009_Yahtzee_013.java  0':57"  00hr:30':14"
23  student009_Yahtzee_014.java  1':30"  00hr:31':44"
24  student009_Yahtzee_015.java  2':33"  00hr:34':17"
25  student009_Yahtzee_016.java  b r e a k
26  student009_Yahtzee_017.java  b r e a k
27  student009_Yahtzee_018.java  b r e a k
28  student009_Yahtzee_019.java  b r e a k
29  student009_Yahtzee_020.java  b r e a k
30  student009_Yahtzee_021.java  9':11"  00hr:43':28"
31  student009_Yahtzee_022.java  4':35"  00hr:48':03"
32  student009_Yahtzee_023.java  4':18"  00hr:52':21"
33  student009_Yahtzee_024.java  7':15"  00hr:59':36"
34  student009_Yahtzee_025.java  0':27"  01hr:00':03"
35  student009_Yahtzee_026.java  1':01"  01hr:01':04"
36  student009_Yahtzee_027.java  5':39"  01hr:06':43"
37  student009_Yahtzee_028.java  2':10"  01hr:08':53"
38  student009_Yahtzee_029.java  0':22"  01hr:09':15"
39  student009_Yahtzee_030.java  0':29"  01hr:09':44"
40  student009_Yahtzee_031.java  1':53"  01hr:11':37"
41  student009_Yahtzee_032.java  1':27"  01hr:13':04"
42  student009_Yahtzee_033.java  2':13"  01hr:15':17"
43  student009_Yahtzee_034.java  3':41"  01hr:18':58"
44  student009_Yahtzee_035.java  3':14"  01hr:22':12"
45  student009_Yahtzee_036.java  1':08"  01hr:23':20"
46  student009_Yahtzee_037.java  0':37"  01hr:23':57"
47  student009_Yahtzee_038.java  1':18"  01hr:25':15"
48  student009_Yahtzee_039.java  0':50"  01hr:26':05"
49  student009_Yahtzee_040.java  2':16"  01hr:28':21"
50  student009_Yahtzee_041.java  0':27"  01hr:28':48"
51  student009_Yahtzee_042.java  0':22"  01hr:29':10"
52  student009_Yahtzee_043.java  0':27"  01hr:29':37"
53  student009_Yahtzee_044.java  2':54"  01hr:32':31"
54  student009_Yahtzee_045.java  0':33"  01hr:33':04"
55  student009_Yahtzee_046.java  0':26"  01hr:33':30"
56  student009_Yahtzee_047.java  b r e a k
57  student009_Yahtzee_048.java  1':10"  01hr:34':40"
58  student009_Yahtzee_049.java  b r e a k
59  student009_Yahtzee_050.java  4':26"  01hr:39':06"
60  student009_Yahtzee_051.java  2':10"  01hr:41':16"
61  student009_Yahtzee_052.java  1':09"  01hr:42':25"
62  student009_Yahtzee_053.java  0':56"  01hr:43':51"
63  student009_Yahtzee_054.java
      
```
- Classpath:** Shows the current classpath configuration.
- Program:** Buttons for Start, Stop, Show, Clear, Fix errors, Run, Abort, Instrument, and Timeout (set to 0.5 sec).
- Progress:** Shows 68% completion.

Figure 9: Delve analyzing the selected submission.

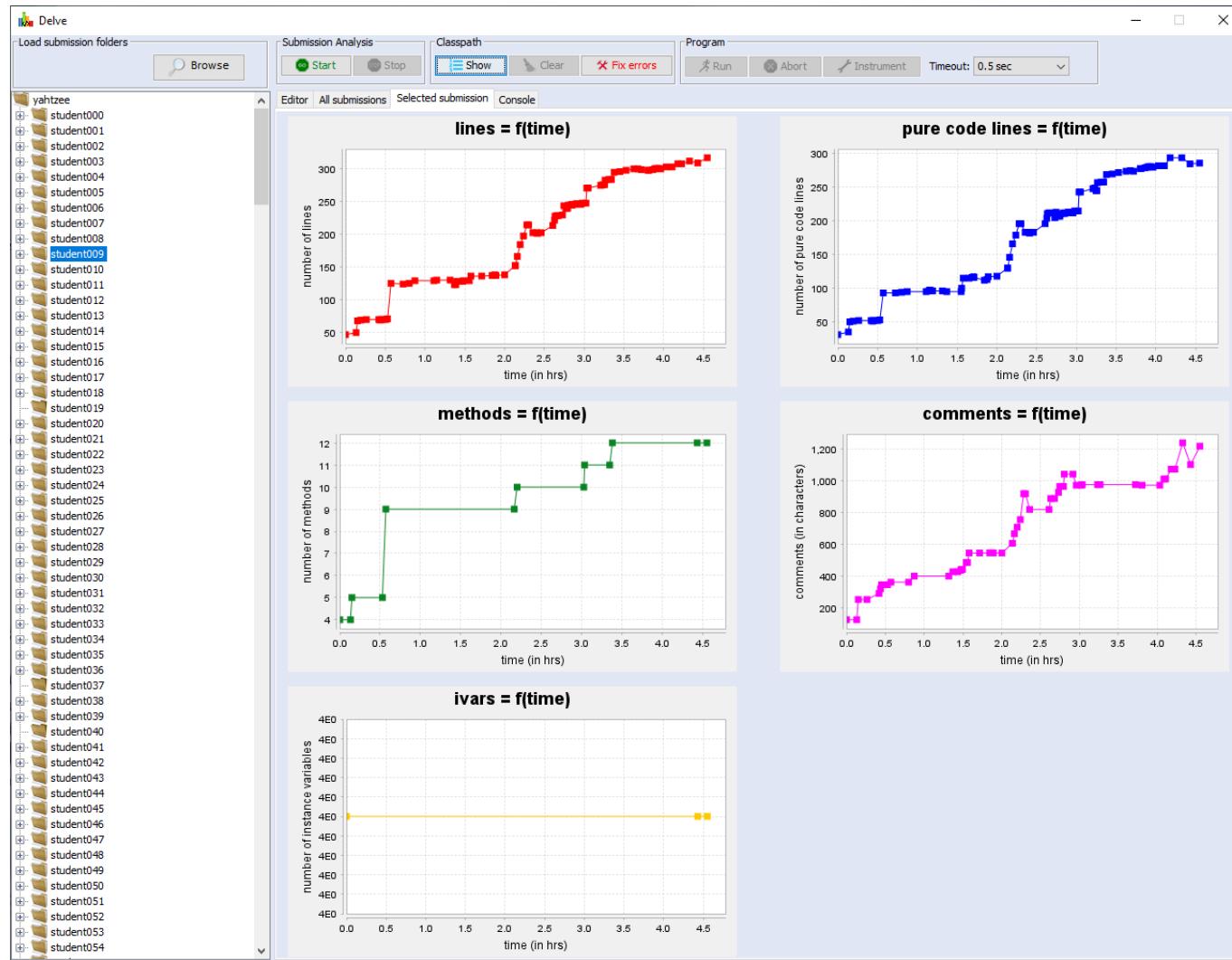
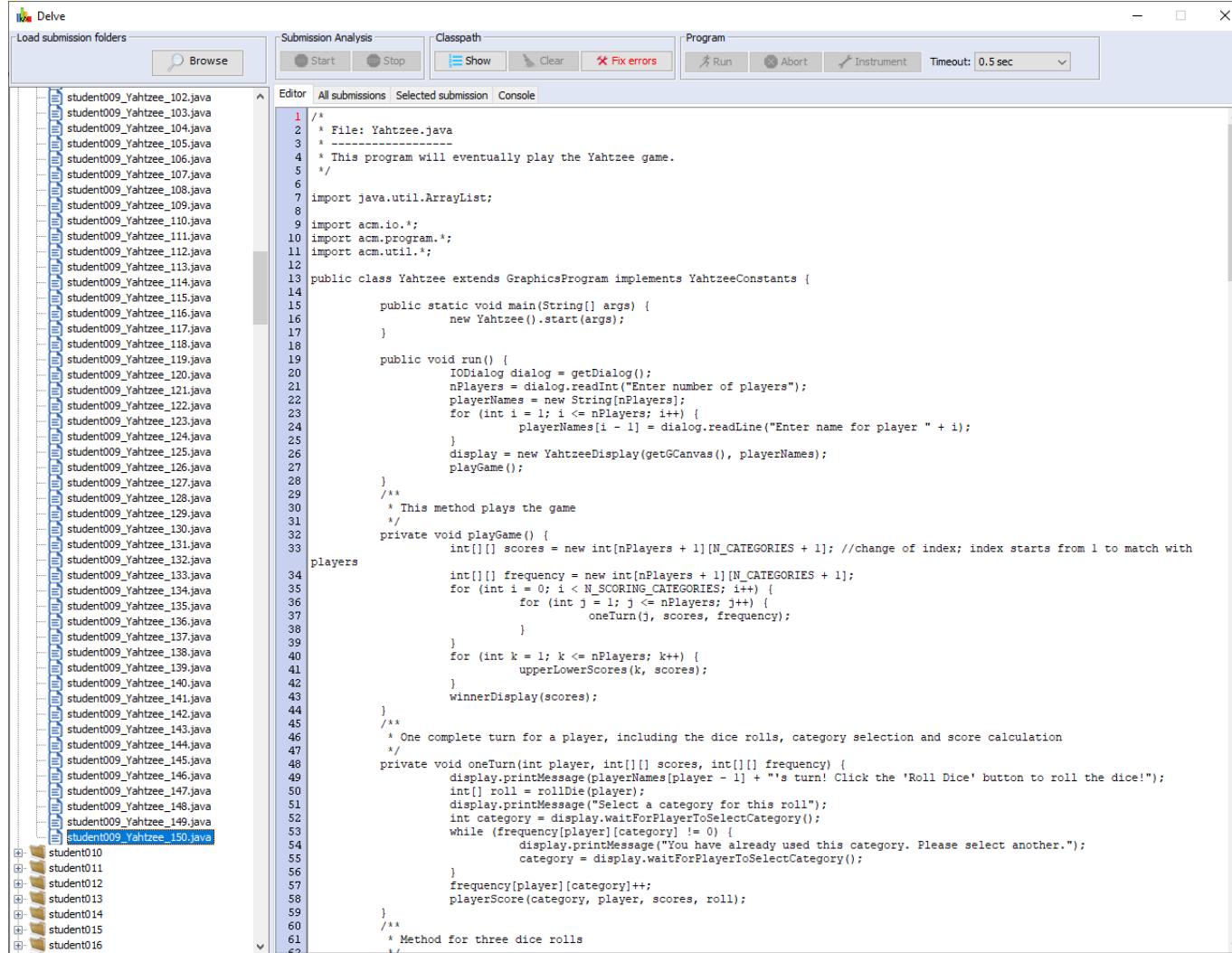


Figure 10: Time plots of various features for an individual student. Note that all plots have the same number of points (measured values for the same time), only points that changed since the last measurement are displayed to avoid overloading the plots. For example, the number of lines kept changing slightly from 0.5hrs to 2hrs, whereas the number of methods was constant for the entire period (although we took the exact same number of points).

5. Program Execution

The user can expand the folder with the student submission and select anyone of its snapshots.



The screenshot shows the Delve tool interface. On the left, there's a tree view labeled "Load submission folders" with several student submissions listed, such as "student009_Yahzee_102.java" through "student009_Yahzee_150.java". The main area is a code editor titled "Editor" with tabs for "All submissions", "Selected submission", and "Console". The code in the editor is as follows:

```
/*
 * File: Yahtzee.java
 *
 * -----
 * This program will eventually play the Yahtzee game.
 */
import java.util.ArrayList;
import acm.util.*;
import acm.program.*;
import acm.util.*;

public class Yahtzee extends GraphicsProgram implements YahtzeeConstants {
    public static void main(String[] args) {
        new Yahtzee().start(args);
    }

    public void run() {
        IDialog dialog = getDialog();
        nPlayers = dialog.readInt("Enter number of players");
        playerNames = new String[nPlayers];
        for (int i = 1; i <= nPlayers; i++) {
            playerNames[i - 1] = dialog.readLine("Enter name for player " + i);
        }
        display = new YahtzeeDisplay(getGCanvas(), playerNames);
        playGame();
    }

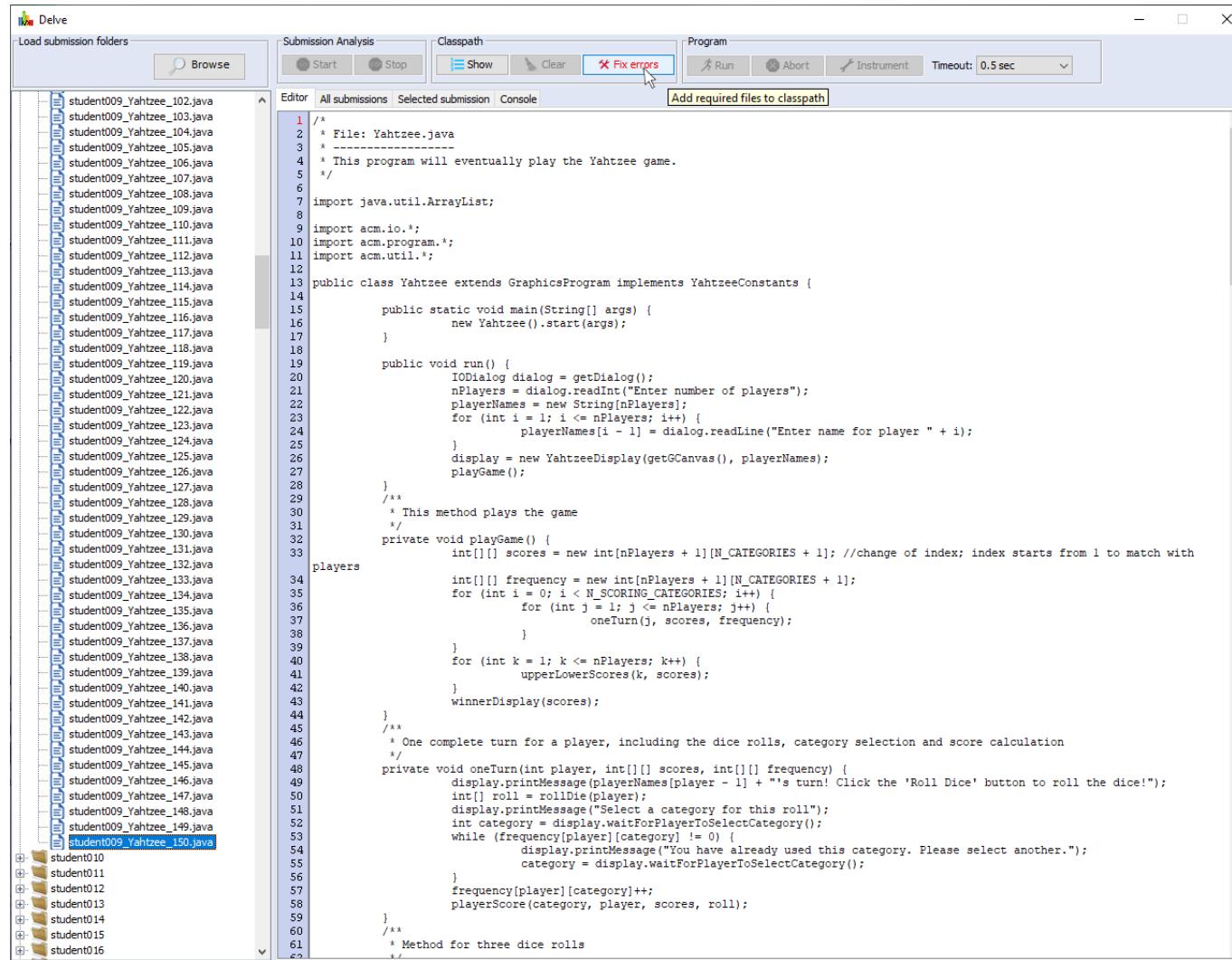
    /**
     * This method plays the game
     */
    private void playGame() {
        int[][] scores = new int[nPlayers + 1][N_CATEGORIES + 1]; //change of index; index starts from 1 to match with
        players
        int[][] frequency = new int[nPlayers + 1][N_SCORING_CATEGORIES + 1];
        for (int i = 0; i < N_SCORING_CATEGORIES; i++) {
            for (int j = 1; j <= nPlayers; j++) {
                oneTurn(j, scores, frequency);
            }
        }
        for (int k = 1; k <= nPlayers; k++) {
            upperLowerScores(k, scores);
        }
        winnerDisplay(scores);
    }

    /**
     * One complete turn for a player, including the dice rolls, category selection and score calculation
     */
    private void oneTurn(int player, int[][] scores, int[][] frequency) {
        display.printMessage(playerNames[player - 1] + "'s turn! Click the 'Roll Dice' button to roll the dice!");
        int[] roll = rollDie(player);
        display.printMessage("Select a category for this roll");
        int category = display.waitForPlayerToSelectCategory();
        while (frequency[player][category] != 0) {
            display.printMessage("You have already used this category. Please select another.");
            category = display.waitForPlayerToSelectCategory();
        }
        frequency[player][category]++;
        playerScore(category, player, scores, roll);
    }

    /**
     * Method for three dice rolls
     */
}
```

Figure 11: As soon as we selecting a snapshot, the source code appears in the editor.

We notice that there are one or more files missing from the classpath and therefore the code does not compile successfully.



The screenshot shows the Delve IDE interface. On the left, a tree view displays multiple Java files under 'student009_Yahtzee' and a folder 'student01'. The main window contains the source code for 'student009_Yahtzee_150.java'. The code is a Java program for the Yahtzee game, including imports for ArrayList, acm.io, acm.program, and acm.util, and a class Yahtzee that implements GraphicsProgram. The code includes methods for running the game, displaying player names, playing the game, calculating scores, and displaying winners. A 'Fix errors' button is highlighted in red at the top right of the editor area. Below the editor, a status bar shows 'Add required files to classpath'.

```
/*
 * File: Yahtzee.java
 *
 * -----
 * This program will eventually play the Yahtzee game.
 */
import java.util.ArrayList;
import acm.io.*;
import acm.program.*;
import acm.util.*;

public class Yahtzee extends GraphicsProgram implements YahtzeeConstants {
    public static void main(String[] args) {
        new Yahtzee().start(args);
    }

    public void run() {
        IDialog dialog = getDialog();
        nPlayers = dialog.readInt("Enter number of players");
        playerNames = new String[nPlayers];
        for (int i = 1; i <= nPlayers; i++) {
            playerNames[i - 1] = dialog.readLine("Enter name for player " + i);
        }
        display = new YahtzeeDisplay(getGCanvas(), playerNames);
        playGame();
    }
    /**
     * This method plays the game
     */
    private void playGame() {
        int[][] scores = new int[nPlayers + 1][N_CATEGORIES + 1]; //change of index; index starts from 1 to match with
        players
        int[][] frequency = new int[nPlayers + 1][N_CATEGORIES + 1];
        for (int i = 0; i < N_SCORING_CATEGORIES; i++) {
            for (int j = 1; j <= nPlayers; j++) {
                oneTurn(j, scores, frequency);
            }
        }
        for (int k = 1; k <= nPlayers; k++) {
            upperLowerScores(k, scores);
        }
        winnerDisplay(scores);
    }
    /**
     * One complete turn for a player, including the dice rolls, category selection and score calculation
     */
    private void oneTurn(int player, int[][] scores, int[][] frequency) {
        display.printMessage(playerNames[player - 1] + "'s turn! Click the 'Roll Dice' button to roll the dice!");
        int[] roll = rollDie(player);
        display.printMessage("Select a category for this roll");
        int category = display.waitForPlayerToSelectCategory();
        while (frequency[player][category] != 0) {
            display.printMessage("You have already used this category. Please select another.");
            category = display.waitForPlayerToSelectCategory();
        }
        frequency[player][category]++;
        playerScore(category, player, scores, roll);
    }
    /**
     * Method for three dice rolls
     */
}
```

Figure 12: Fixing the classpath is needed to run the program.

Typically, those are the starter files that accompany the assignment (e.g. libraries or other files). In the example, the assignment has three starter files (two jar files and a file with constants).

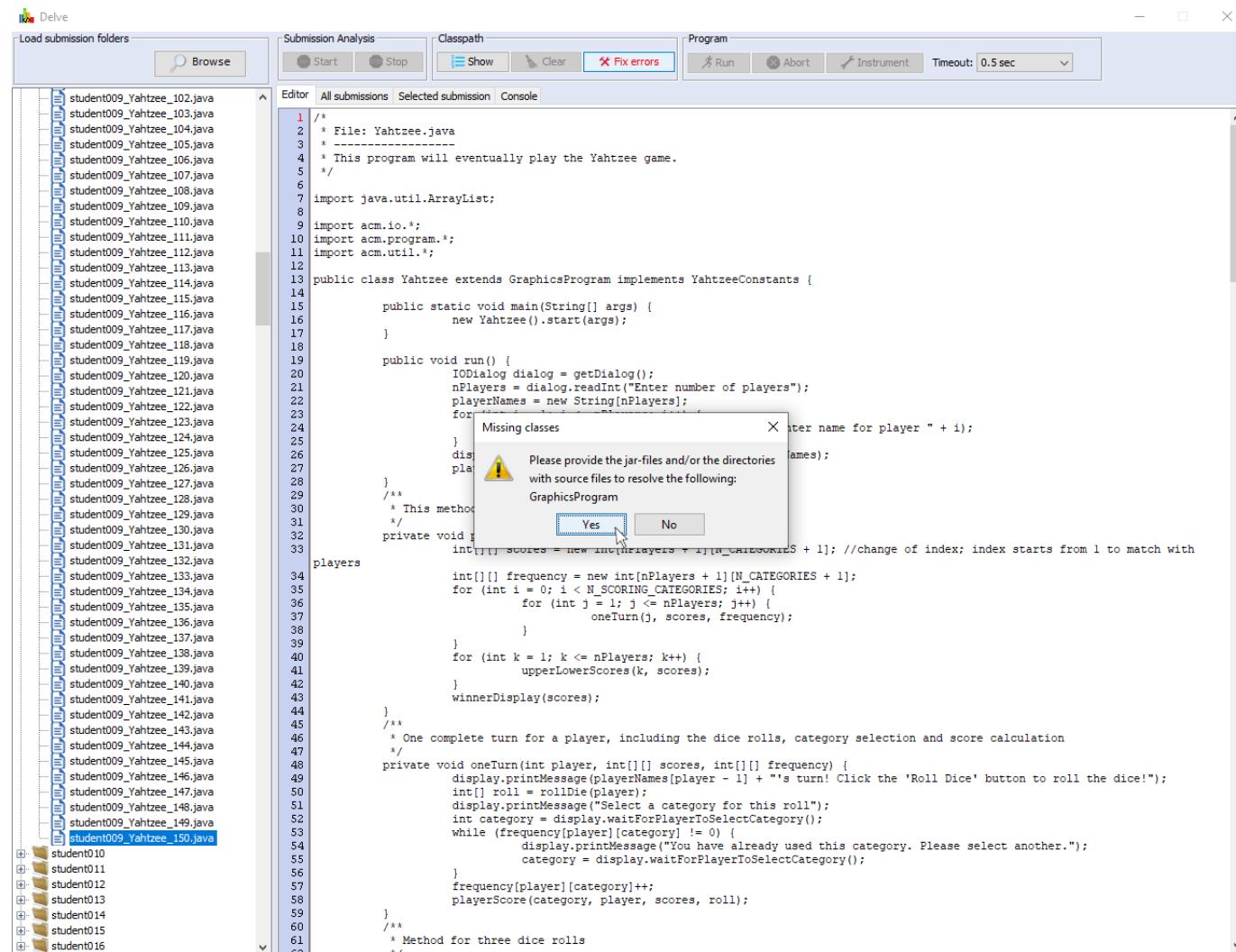


Figure 13:Preparing to add the first file that is missing from the classpath

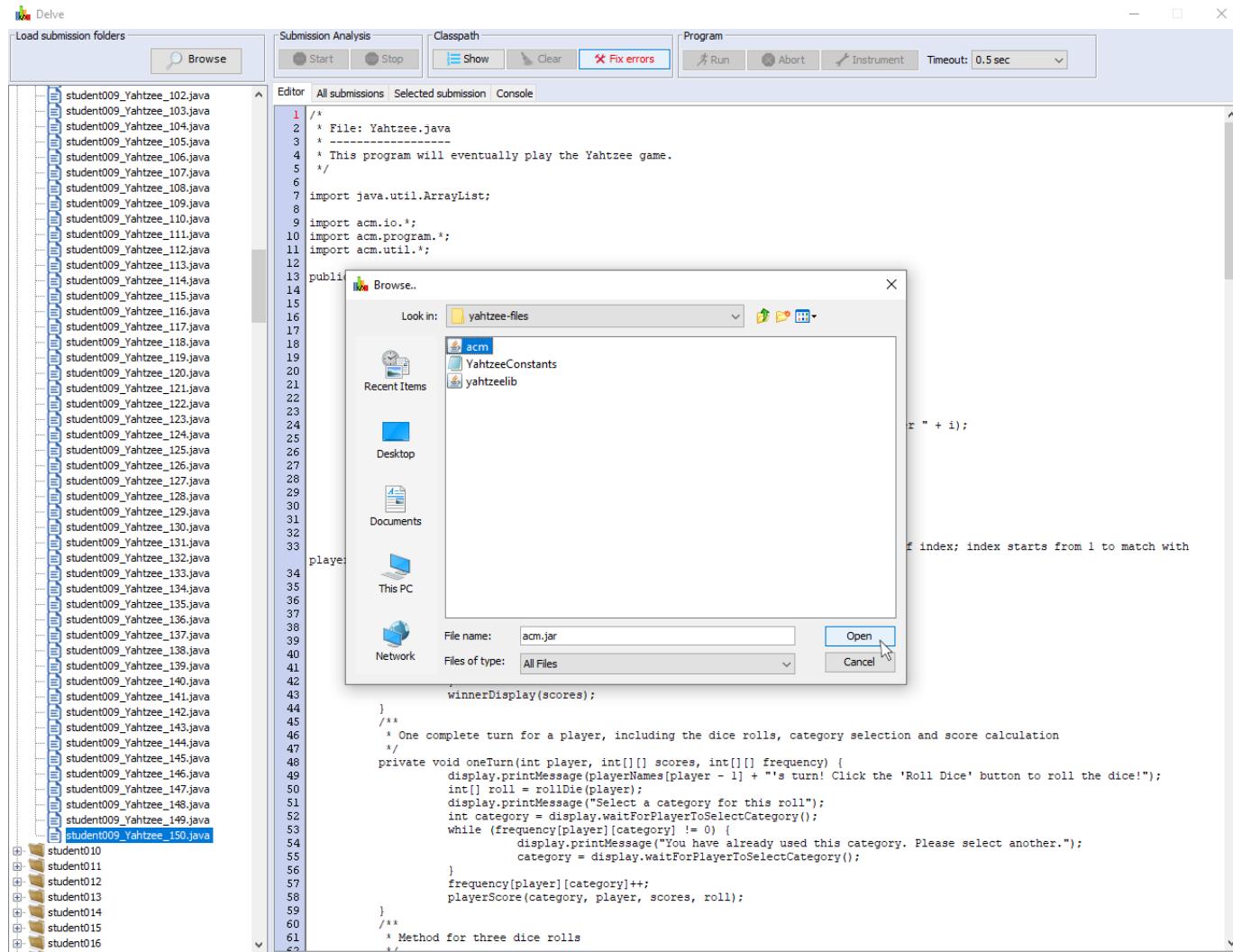


Figure 14: Selecting the first of the three files that are missing from the classpath.

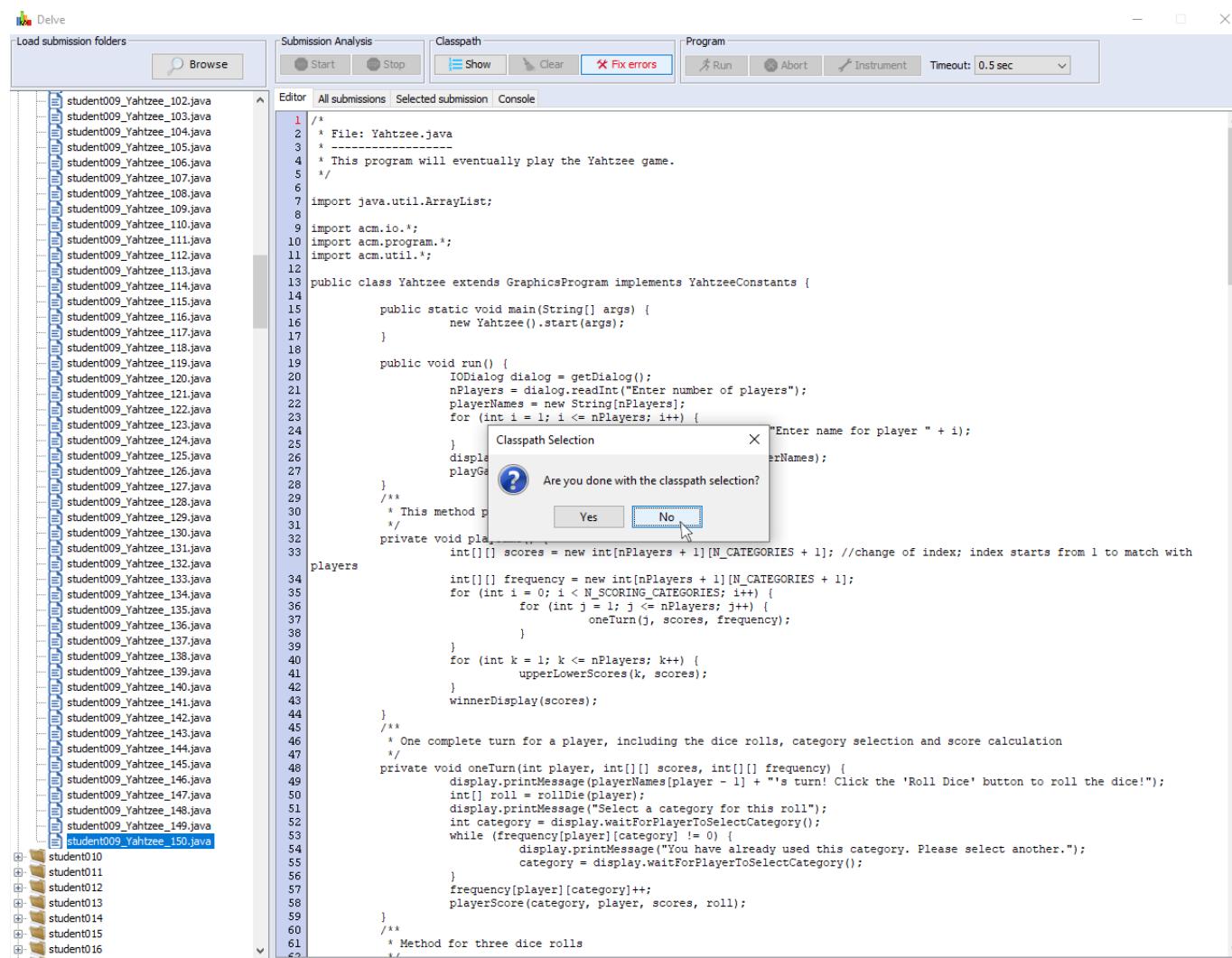


Figure 15: There are more files missing from the classpath. Preparing to add the second required file to the classpath.

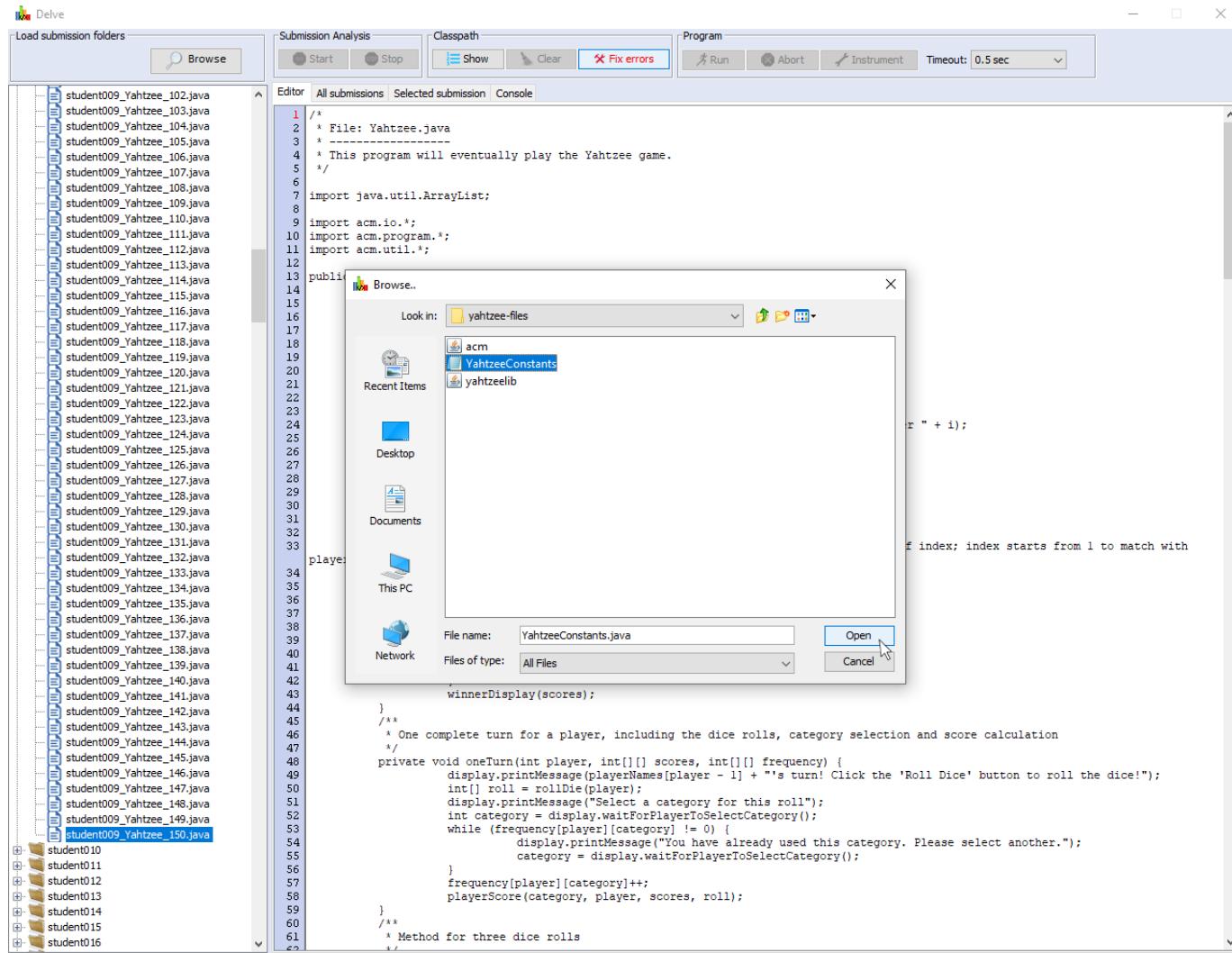


Figure 16: Selecting the second of the three files that are missing from the classpath.

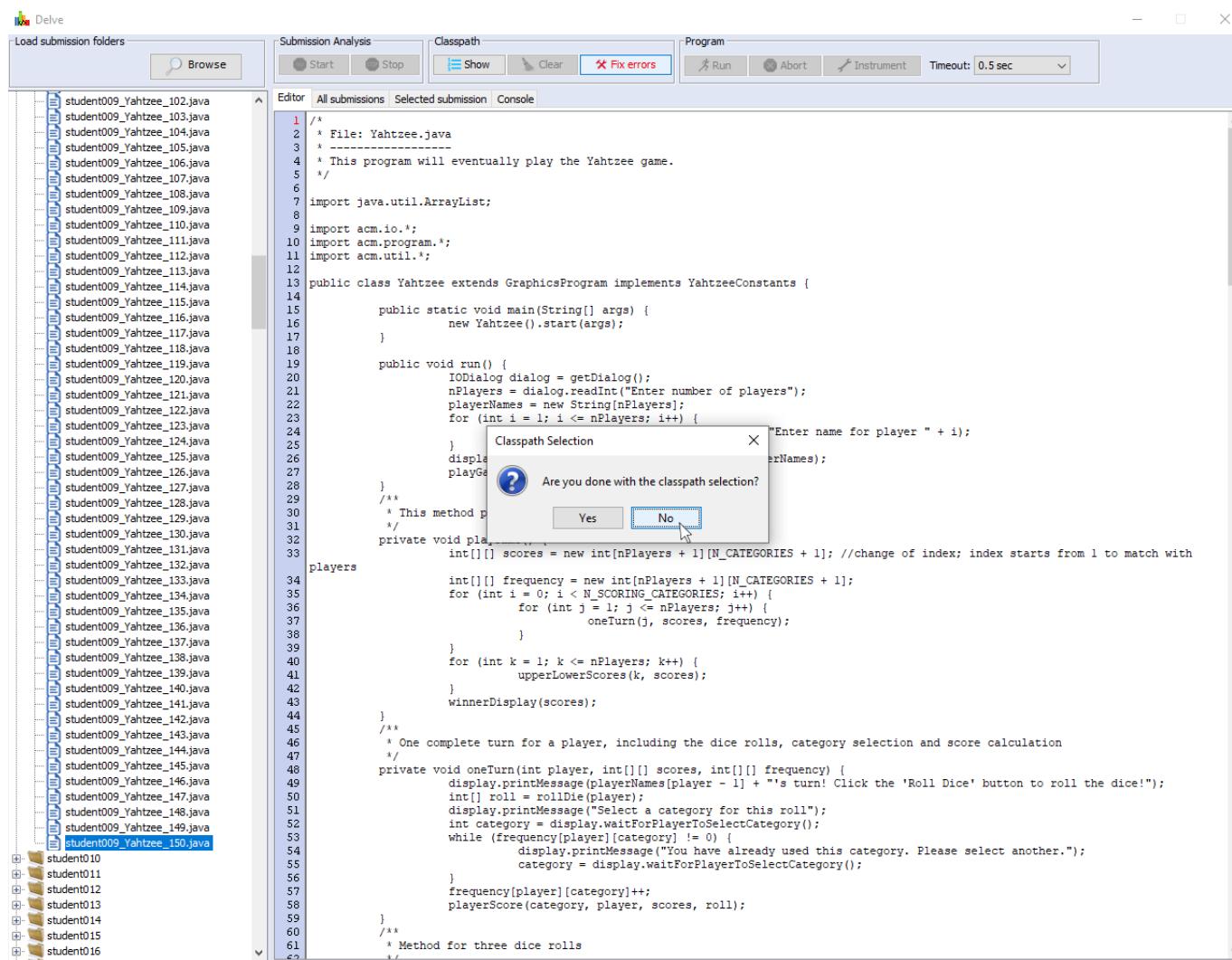


Figure 17: There is one more file missing from the classpath. Preparing to add it.

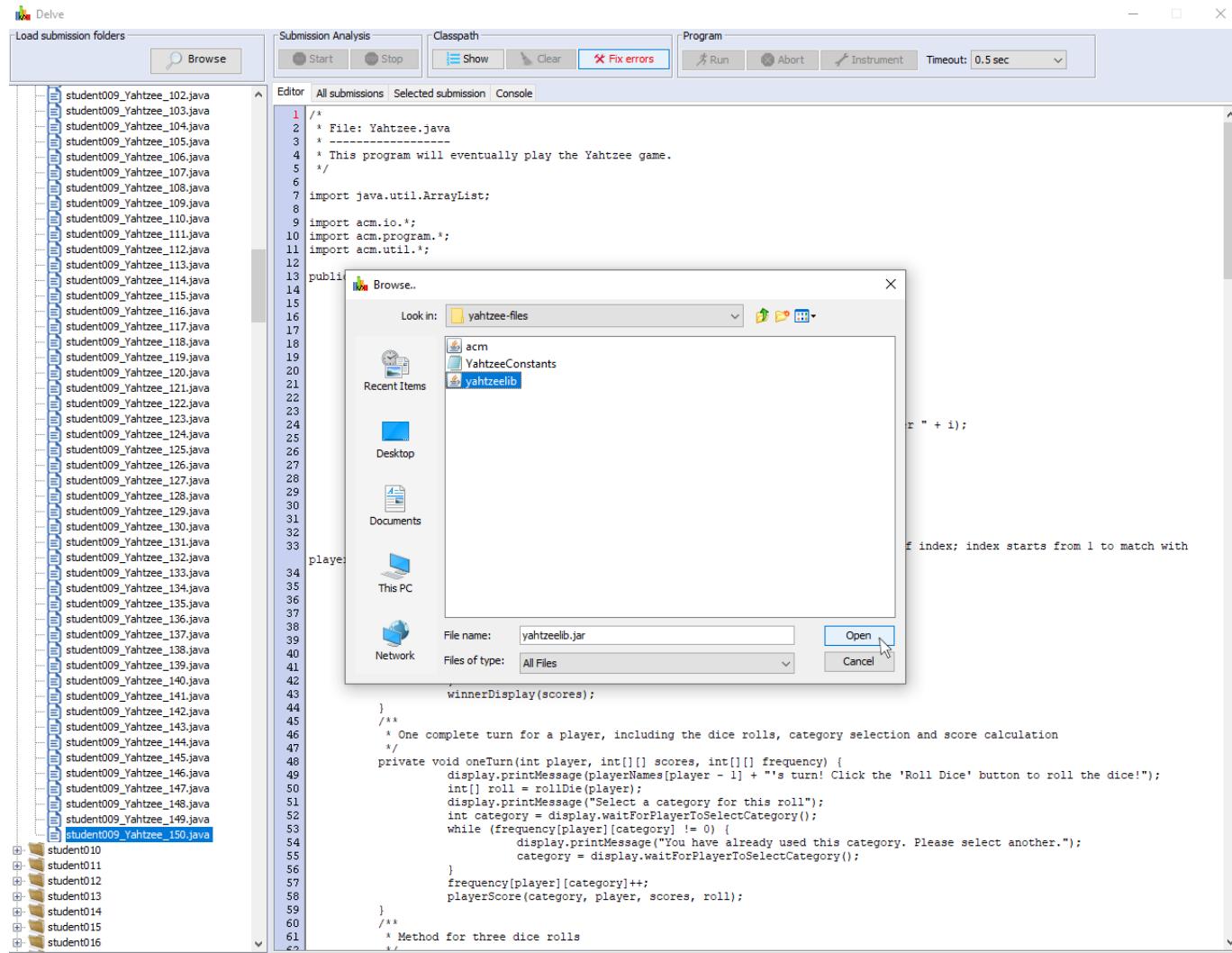


Figure 18: Selecting the last of the three files that are missing from the classpath.

```

/*
 * File: Yahtzee.java
 *
 * -----
 * This program will eventually play the Yahtzee game.
 */
import java.util.ArrayList;
import acm.io.*;
import acm.program.*;
import acm.util.*;

public class Yahtzee extends GraphicsProgram implements YahtzeeConstants {
    public static void main(String[] args) {
        new Yahtzee().start(args);
    }

    public void run() {
        IDialog dialog = getDialog();
        nPlayers = dialog.readInt("Enter number of players");
        playerNames = new String[nPlayers];
        for (int i = 1; i <= nPlayers; i++) {
            dialog.showText("Enter name for player " + i);
            playerNames[i - 1] = dialog.readLine();
        }
    }

    /**
     * This method performs one complete turn for a player.
     */
    private void playGame() {
        int[][] scores = new int[nPlayers + 1][N_CATEGORIES + 1]; //change of index; index starts from 1 to match with
        players
        int[][] frequency = new int[nPlayers + 1][N_CATEGORIES + 1];
        for (int i = 0; i < N_SCORING_CATEGORIES; i++) {
            for (int j = 1; j <= nPlayers; j++) {
                oneTurn(j, scores, frequency);
            }
        }
        for (int k = 1; k <= nPlayers; k++) {
            upperLowerScores(k, scores);
        }
        winnerDisplay(scores);
    }

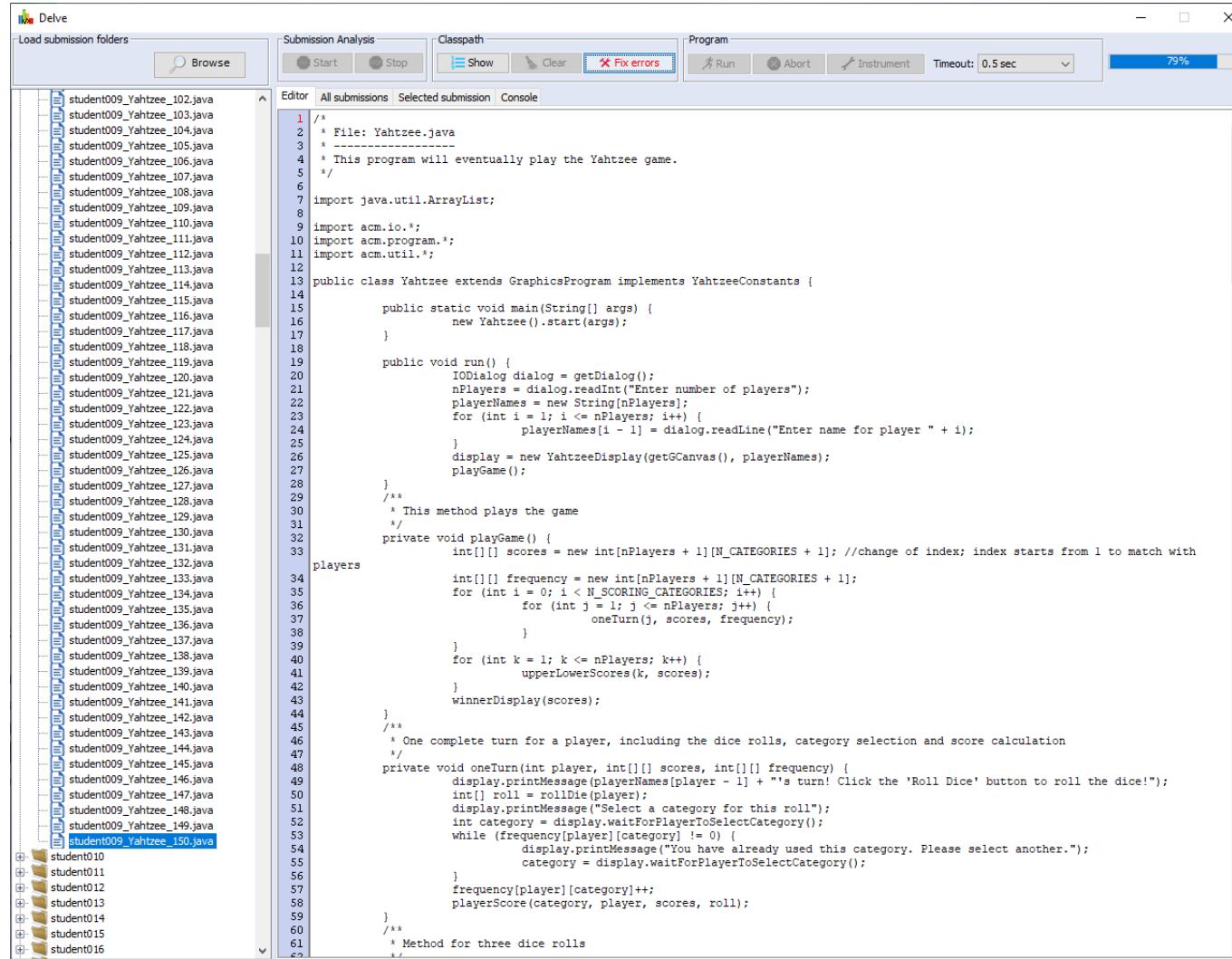
    /**
     * One complete turn for a player, including the dice rolls, category selection and score calculation
     */
    private void oneTurn(int player, int[][] scores, int[][] frequency) {
        display.printMessage(playerNames[player - 1] + "'s turn! Click the 'Roll Dice' button to roll the dice!");
        int[] roll = rollDie(player);
        display.printMessage("Select a category for this roll");
        int category = display.waitForPlayerToSelectCategory();
        while (frequency[player][category] != 0) {
            display.printMessage("You have already used this category. Please select another.");
            category = display.waitForPlayerToSelectCategory();
        }
        frequency[player][category]++;
        playerScore(category, player, scores, roll);
    }

    /**
     * Method for three dice rolls
     */
}

```

Figure 19: There are no more files missing from the classpath.

Once we have added one or more files, Delve tries to determine if the classpath is correctly configured. It does that by analyzing the final snapshots for every student as those are expected to compile. If 75%+ of them compile, the classpath is configured successfully.



The screenshot shows the Delve application window. In the top-left, there's a sidebar titled "Load submission folders" with a "Browse" button. The main area has tabs for "Submission Analysis", "Classpath" (which is selected), and "Program". Below these are buttons for "Start" and "Stop", "Show", "Clear", and "Fix errors". To the right of these are "Run", "Abort", and "Instrument" buttons, along with a "Timeout: 0.5 sec" dropdown set to "79%".

The central part of the window is the "Editor" tab, which displays the Java code for a "Yahtzee" program. The code includes imports for `java.util.ArrayList`, `acm.io.*`, `acm.program.*`, and `acm.util.*`. It defines a `Yahtzee` class that extends `GraphicsProgram` and implements `YahtzeeConstants`. The `main` method initializes a dialog and starts the game. The `run` method handles player input and starts the game. The `playGame` method is annotated with `/** * This method plays the game */` and contains logic for calculating scores and displaying winners. The `oneTurn` method is annotated with `/** * One complete turn for a player, including the dice rolls, category selection and score calculation */` and handles player input for rolling dice and selecting categories. The `upperLowerScores` method is annotated with `/** * Method for three dice rolls */` and calculates scores for upper and lower categories.

On the left side of the editor, there's a tree view showing submissions from students 009 to 16. The tree view shows a folder for each student, with individual Java files listed under them. The code in the editor corresponds to the `student009_Yahtzee_102.java` file.

Figure 20: Delve tries to determine if the classpath is correctly configured by trying to compile the last snapshots of every student.

The screenshot shows the Delve debugger interface. On the left, a tree view displays submission folders for students 09 through 16, with student09 expanded to show numerous Java files named student09_Yahhtzee_102.java, student09_Yahhtzee_103.java, etc., up to student09_Yahhtzee_150.java. The main window contains an Editor tab showing the source code for `Yahtzee.java`. The code implements a `GraphicsProgram` that reads player names and scores, then displays the game and calculates a winner. A `Classpath` tab at the top indicates the classpath is correctly configured.

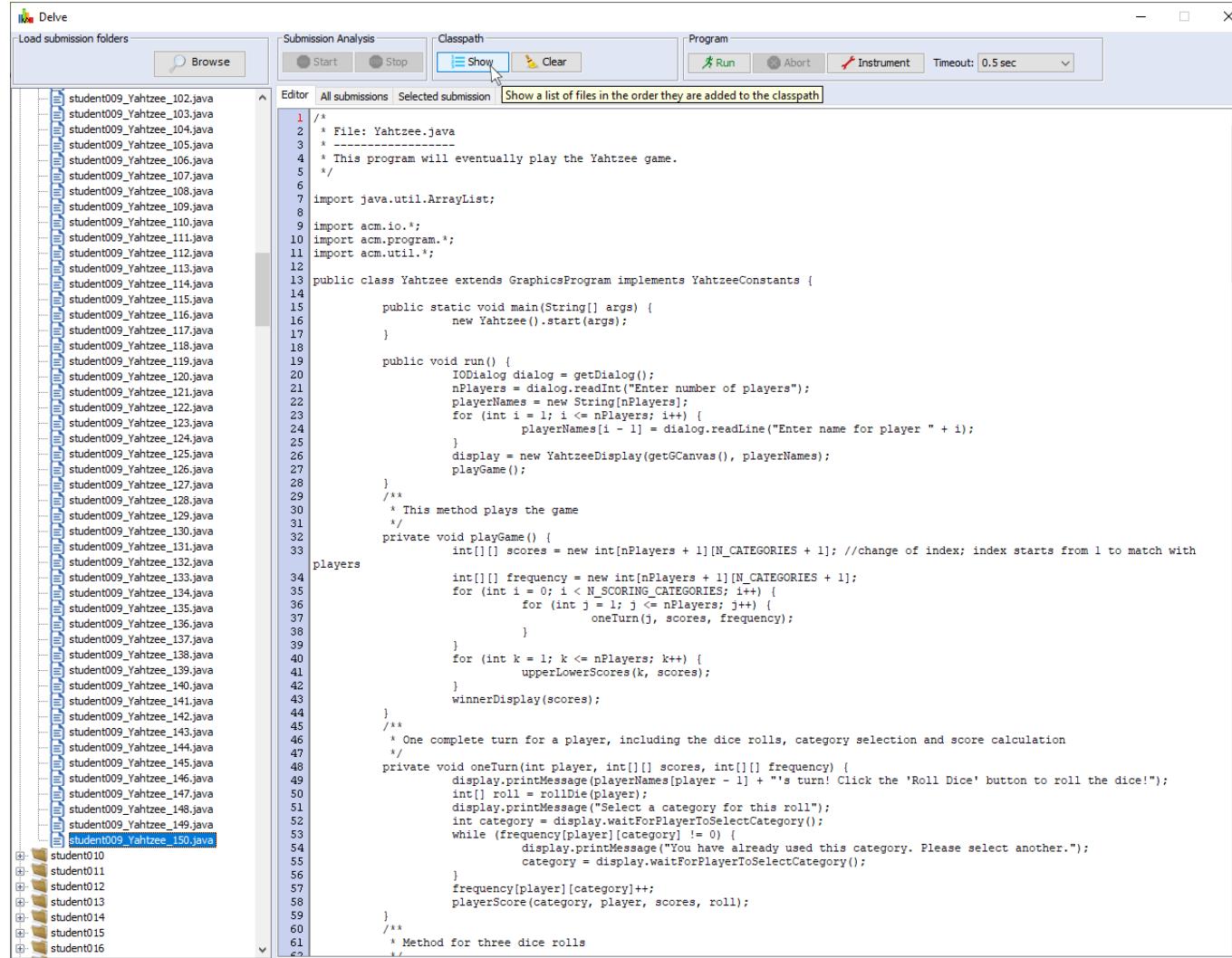
```

1  /*
2   * File: Yahtzee.java
3   * -----
4   * This program will eventually play the Yahtzee game.
5   */
6
7 import java.util.ArrayList;
8
9 import acm.io.*;
10 import acm.program.*;
11 import acm.util.*;
12
13 public class Yahtzee extends GraphicsProgram implements YahtzeeConstants {
14
15     public static void main(String[] args) {
16         new Yahtzee().start(args);
17     }
18
19     public void run() {
20         IODialog dialog = getDialog();
21         nPlayers = dialog.readInt("Enter number of players");
22         playerNames = new String[nPlayers];
23         for (int i = 1; i <= nPlayers; i++) {
24             playerNames[i - 1] = dialog.readLine("Enter name for player " + i);
25         }
26         display = new YahtzeeDisplay(getGCanvas(), playerNames);
27         playGame();
28     }
29
30     /**
31      * This method plays the game
32     */
33     private void playGame() {
34         int[][] scores = new int[nPlayers + 1][N_CATEGORIES + 1]; //change of index; index starts from 1 to match with
35         players
36         int[][] frequency = new int[nPlayers + 1][N_CATEGORIES + 1];
37         for (int i = 0; i < N_SCORING_CATEGORIES; i++) {
38             for (int j = 1; j <= nPlayers; j++) {
39                 oneTurn(j, scores, frequency);
40             }
41         }
42         for (int k = 1; k <= nPlayers; k++) {
43             upperLowerScores(k, scores);
44         }
45         winnerDisplay(scores);
46     }
47
48     /**
49      * One complete turn for a player, including the dice rolls, category selection and score calculation
50     */
51     private void oneTurn(int player, int[][] scores, int[][] frequency) {
52         display.printMessage(playerNames[player - 1] + "'s turn! Click the 'Roll Dice' button to roll the dice!");
53         int[] roll = rollDie(player);
54         display.printMessage("Select a category for this roll");
55         int category = display.waitForPlayerToSelectCategory();
56         while (frequency[player][category] != 0) {
57             display.printMessage("You have already used this category. Please select another.");
58             category = display.waitForPlayerToSelectCategory();
59         }
60         frequency[player][category]++;
61         playerScore(category, player, scores, roll);
62     }
63
64     /**
65      * Method for three dice rolls
66     */

```

Figure 21: The button *Fix errors* disappeared which means that the classpath is configured correctly.

At any point, we can check the classpath.



The screenshot shows the Delve IDE interface. On the left, there's a tree view of submission folders: student09 (containing many files from student09_Yahzee_102.java to student09_Yahzee_150.java) and student010 through student016. The main window has tabs for Submission Analysis, Classpath, and Program. The Classpath tab is active, featuring a 'Show' button with a magnifying glass icon. Below it is a text area titled 'Show a list of files in the order they are added to the classpath'. The code shown is the Yahtzee.java file, which defines a class that extends GraphicsProgram and implements YahtzeeConstants. It includes methods for running the game, playing, and calculating scores. The code is annotated with numerous comments explaining its functionality.

```
/*
 * File: Yahtzee.java
 *
 * -----
 * This program will eventually play the Yahtzee game.
 */
import java.util.ArrayList;
import acm.io.*;
import acm.program.*;
import acm.util.*;

public class Yahtzee extends GraphicsProgram implements YahtzeeConstants {
    public static void main(String[] args) {
        new Yahtzee().start(args);
    }

    public void run() {
        IDialog dialog = getDialog();
        nPlayers = dialog.readInt("Enter number of players");
        playerNames = new String[nPlayers];
        for (int i = 1; i <= nPlayers; i++) {
            playerNames[i - 1] = dialog.readLine("Enter name for player " + i);
        }
        display = new YahtzeeDisplay(getGCanvas(), playerNames);
        playGame();
    }

    /**
     * This method plays the game
     */
    private void playGame() {
        int[][] scores = new int[nPlayers + 1][N_CATEGORIES + 1]; //change of index; index starts from 1 to match with
        players
        int[][] frequency = new int[nPlayers + 1][N_CATEGORIES + 1];
        for (int i = 0; i < N_SCORING_CATEGORIES; i++) {
            for (int j = 1; j <= nPlayers; j++) {
                oneTurn(j, scores, frequency);
            }
        }
        for (int k = 1; k <= nPlayers; k++) {
            upperLowerScores(k, scores);
        }
        winnerDisplay(scores);
    }

    /**
     * One complete turn for a player, including the dice rolls, category selection and score calculation
     */
    private void oneTurn(int player, int[][] scores, int[][] frequency) {
        display.printMessage(playerNames[player - 1] + "'s turn! Click the 'Roll Dice' button to roll the dice!");
        int[] roll = rollDie(player);
        display.printMessage("Select a category for this roll");
        int category = display.waitForPlayerToSelectCategory();
        while (frequency[player][category] != 0) {
            display.printMessage("You have already used this category. Please select another.");
            category = display.waitForPlayerToSelectCategory();
        }
        frequency[player][category]++;
        playerScore(category, player, scores, roll);
    }

    /**
     * Method for three dice rolls
     */
}
```

Figure 22: We can click button 'Show' at any point in time to see the existing classpath.

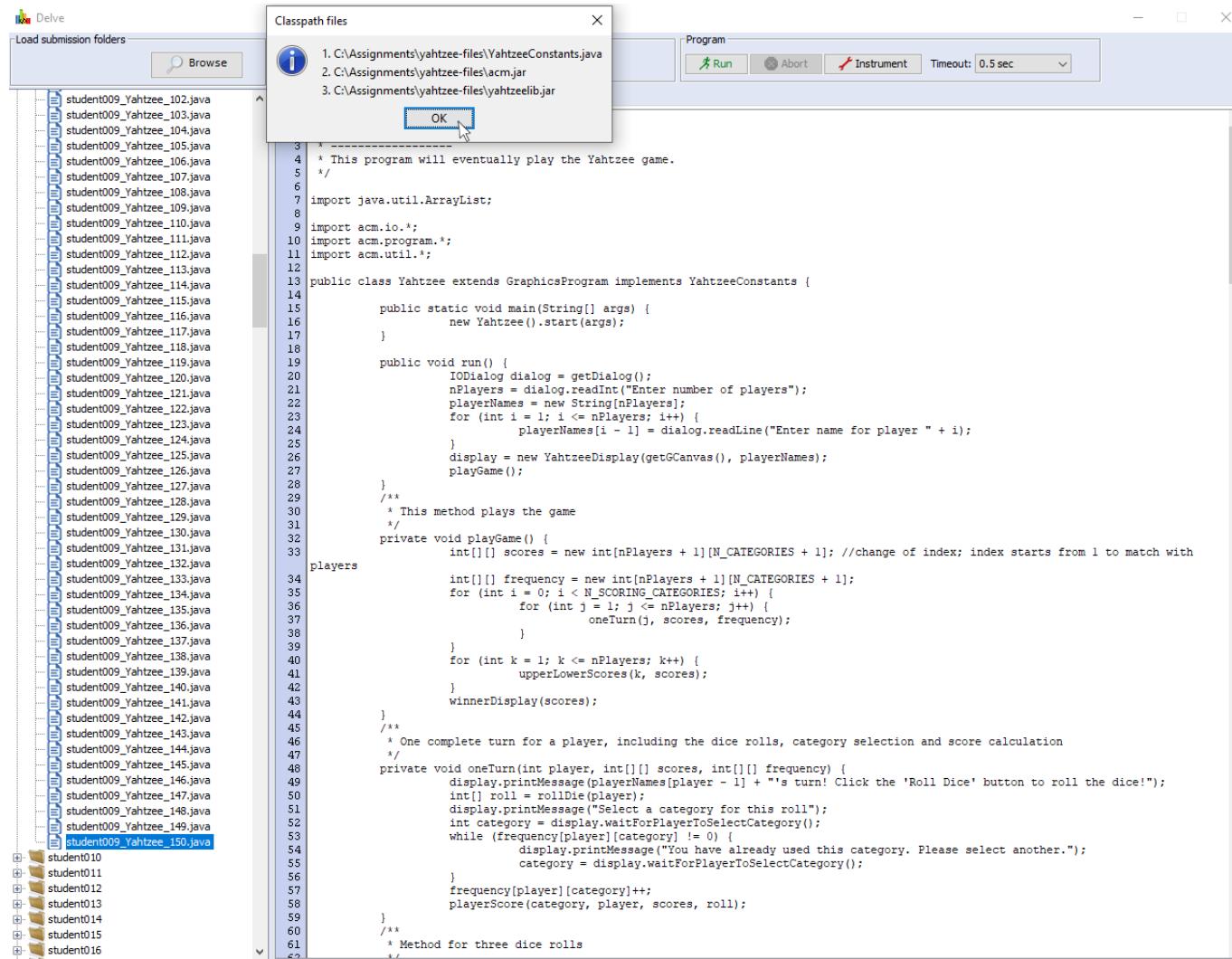
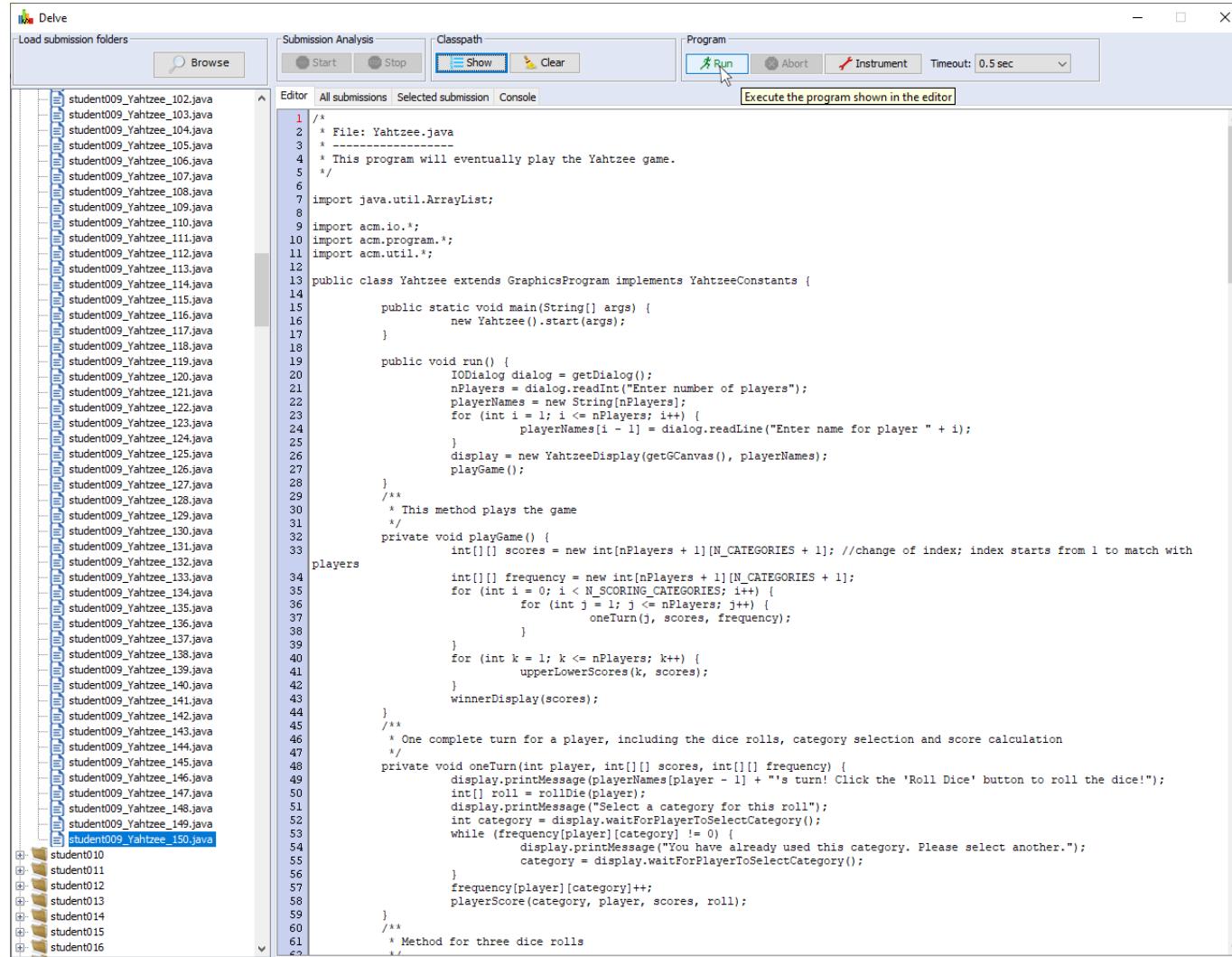


Figure 23: A small pop-up window displays the three files that we added to the classpath.

Now, every snapshot that compiles and includes method `public static void main(String[])` can be executed. When an executable snapshot is selected button **Run** is enabled.



The screenshot shows the Delve interface with the following details:

- Submission Analysis:** Shows "All submissions" selected.
- Program:** The "Run" button is highlighted in blue, indicating it is enabled.
- Editor:** Displays the Java code for `Yahtzee.java`. The code includes imports for `java.util.ArrayList`, `acm.io.*`, `acm.program.*`, and `acm.util.*`. It defines a class `Yahtzee` that extends `GraphicsProgram` and implements `YahtzeeConstants`. The `main` method creates a new `Yahtzee` object and starts it with arguments. The `run` method handles player input for the number of players and their names, initializes a display, and calls `playGame`. The `playGame` method initializes scores and frequency arrays, iterates over players, and calls `oneTurn` for each. The `oneTurn` method prints a message, rolls dice, selects a category, and updates scores and frequency. The code also handles category selection and roll frequency.
- File List:** On the left, there is a tree view of submission folders for students 009 through 16, with `student009_Yahtzee_150.java` selected.

Figure 24: Snapshot `student009_Yahtzee_150.java` compiles and includes also `public static void main(String[])`. Therefore, button **Run** is enabled, ready to execute the program.

Pressing **Run** kicks off the program execution.

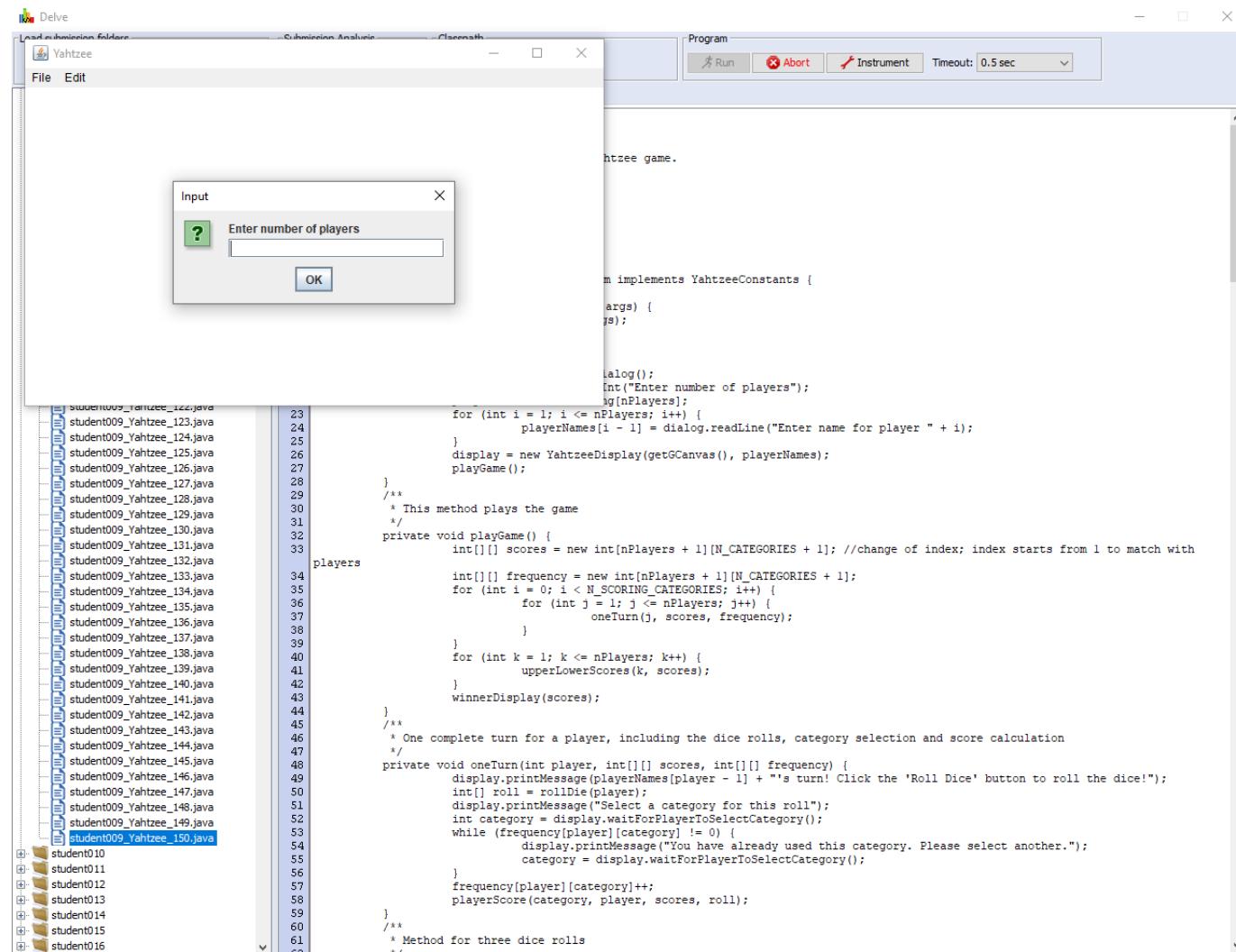


Figure 25: To execute the program we simply press *Run*.

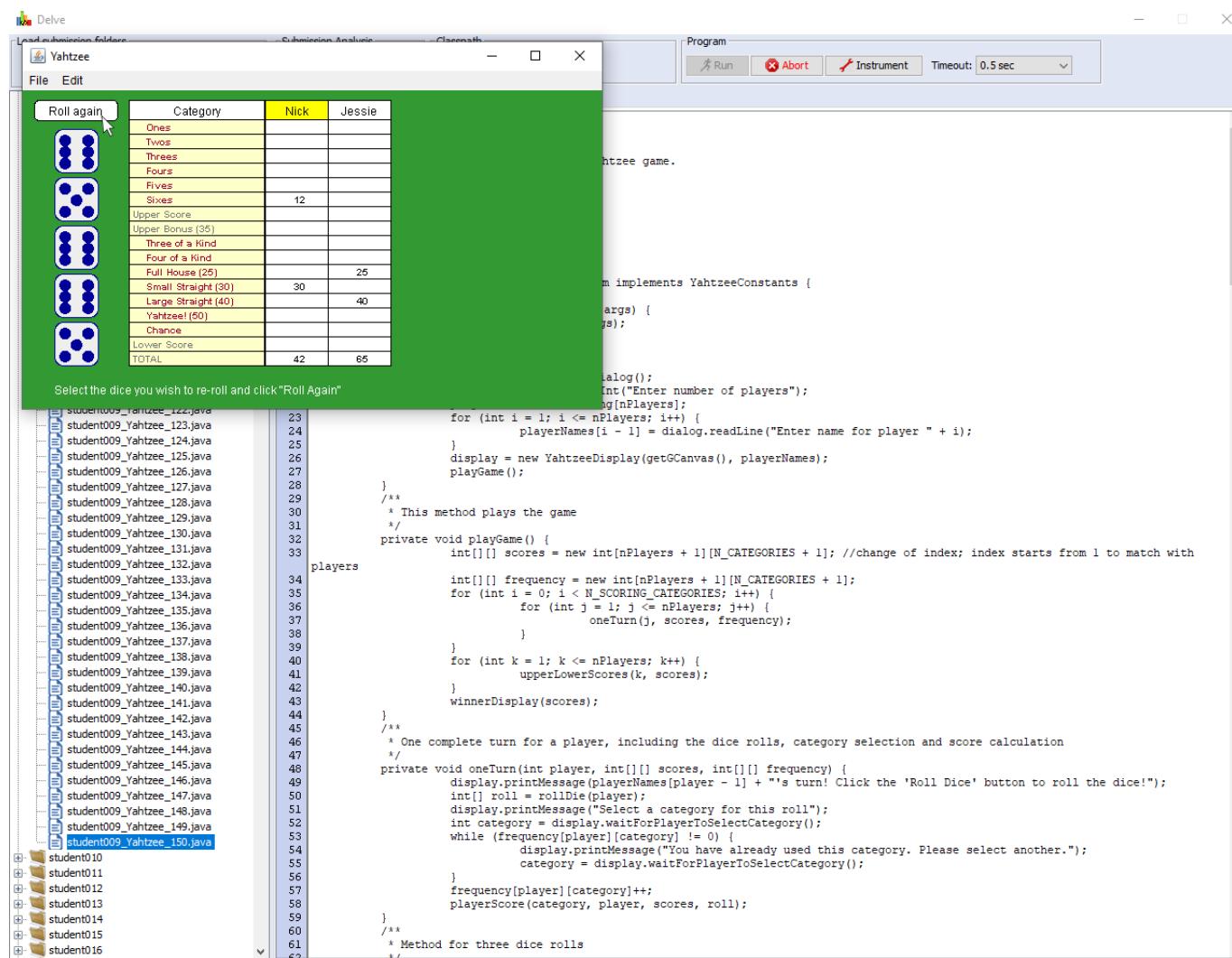


Figure 26: The program runs normally, just like in an IDE environment.

The screenshot shows the Delve IDE interface. On the left, there's a file tree with several Java files under 'student09_Yahtzee' and a folder for 'student10' through 'student16'. In the center, there's a 'Yahtzee' window showing a dice roll simulation with three dice showing faces 1, 2, and 3. To the right of the window is a table comparing scores for categories like Ones, Twos, etc., between two players, Nick and Jessie. The table also includes columns for Upper Score, Upper Bonus, Lower Score, and a TOTAL row. The rightmost pane displays the source code for 'Yahtzee.java'. The code implements a Yahtzee game with methods for rolling dice, selecting categories, and calculating scores. It includes imports for java.util.* and javax.swing.*. The code uses a dialog to get player names and a display to show the game state.

```

    public class Yahtzee {
        // ... (imports and constants)
        public void playGame() {
            int[][] scores = new int[nPlayers + 1][N_CATEGORIES + 1]; //change of index; index starts from 1 to match with players
            int[][] frequency = new int[nPlayers + 1][N_CATEGORIES + 1];
            for (int i = 0; i < N_SCORING_CATEGORIES; i++) {
                for (int j = 1; j <= nPlayers; j++) {
                    oneTurn(j, scores, frequency);
                }
            }
            for (int k = 1; k <= nPlayers; k++) {
                upperLowerScores(k, scores);
            }
            winnerDisplay(scores);
        }
        // ...
        private void oneTurn(int player, int[][] scores, int[][] frequency) {
            display.printMessage(playerNames[player - 1] + "'s turn! Click the 'Roll Dice' button to roll the dice!");
            int[] roll = rollDie(player);
            display.printMessage("Select a category for this roll");
            int category = display.waitForPlayerToSelectCategory();
            while (frequency[player][category] != 0) {
                display.printMessage("You have already used this category. Please select another.");
                category = display.waitForPlayerToSelectCategory();
            }
            frequency[player][category]++;
            playerScore(category, player, scores, roll);
        }
        // ...
    }
}

```

Figure 27: The program runs eventually to completion.

6. Individual Results (classpath is configured)

Now that classpath is configured, we can press **Start** to get additional analysis results about an individual submission.

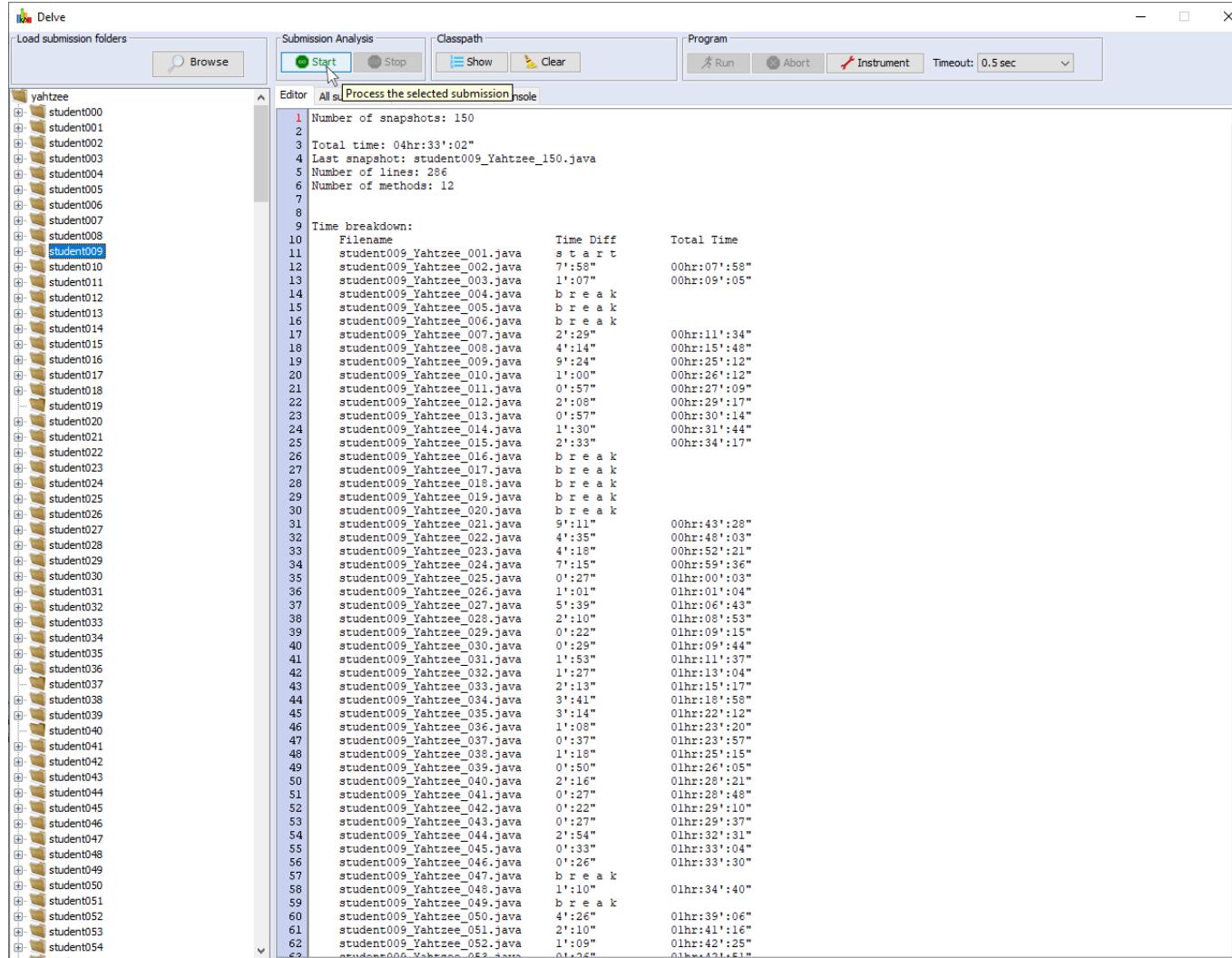


Figure 28: On *Start*, Delve begins analyzing the submission to get additional results now that classpath is configured.

The screenshot shows the Delve tool interface with the following details:

- Load submission folders:** A tree view showing submissions from students 000 to 054, with student009 selected.
- Submission Analysis:** Buttons for Start, Stop, Show, Clear, Run, Abort, Instrument, and Timeout (set to 0.5 sec).
- Program:** Progress bar at 59%.
- Editor:** Tab selection for All submissions, Selected submission, and Console.
- Console Output:**

```

1 Number of snapshots: 150
2
3 Total time: 04hr:33'02"
4 Last snapshot: student009_Yahtzee_150.java
5 Number of lines: 286
6 Number of methods: 12
7
8 Time breakdown:
9
10 Filename           Time Diff      Total Time
11 student009_Yahtzee_001.java    s t a r t   00hr:07':58"
12 student009_Yahtzee_002.java    7':58"       00hr:07':58"
13 student009_Yahtzee_003.java    1':07"       00hr:09':05"
14 student009_Yahtzee_004.java    b r e a k
15 student009_Yahtzee_005.java    b r e a k
16 student009_Yahtzee_006.java    b r e a k
17 student009_Yahtzee_007.java    2':29"       00hr:11':34"
18 student009_Yahtzee_008.java    4':14"       00hr:15':48"
19 student009_Yahtzee_009.java    9':24"       00hr:25':12"
20 student009_Yahtzee_010.java    1':00"       00hr:26':12"
21 student009_Yahtzee_011.java    0':57"       00hr:27':09"
22 student009_Yahtzee_012.java    2':08"       00hr:29':17"
23 student009_Yahtzee_013.java    0':57"       00hr:30':14"
24 student009_Yahtzee_014.java    1':30"       00hr:31':44"
25 student009_Yahtzee_015.java    2':33"       00hr:34':17"
26 student009_Yahtzee_016.java    b r e a k
27 student009_Yahtzee_017.java    b r e a k
28 student009_Yahtzee_018.java    b r e a k
29 student009_Yahtzee_019.java    b r e a k
30 student009_Yahtzee_020.java    b r e a k
31 student009_Yahtzee_021.java    9':11"       00hr:43':28"
32 student009_Yahtzee_022.java    4':35"       00hr:48':03"
33 student009_Yahtzee_023.java    4':18"       00hr:52':21"
34 student009_Yahtzee_024.java    7':15"       00hr:59':36"
35 student009_Yahtzee_025.java    0':27"       01hr:00':03"
36 student009_Yahtzee_026.java    1':01"       01hr:01':04"
37 student009_Yahtzee_027.java    5':39"       01hr:06':43"
38 student009_Yahtzee_028.java    2':10"       01hr:08':53"
39 student009_Yahtzee_029.java    0':22"       01hr:09':15"
40 student009_Yahtzee_030.java    0':29"       01hr:09':44"
41 student009_Yahtzee_031.java    1':53"       01hr:11':37"
42 student009_Yahtzee_032.java    1':27"       01hr:13':04"
43 student009_Yahtzee_033.java    2':13"       01hr:15':17"
44 student009_Yahtzee_034.java    3':41"       01hr:18':58"
45 student009_Yahtzee_035.java    3':14"       01hr:22':12"
46 student009_Yahtzee_036.java    1':08"       01hr:23':20"
47 student009_Yahtzee_037.java    0':37"       01hr:23':57"
48 student009_Yahtzee_038.java    1':18"       01hr:25':15"
49 student009_Yahtzee_039.java    0':50"       01hr:26':05"
50 student009_Yahtzee_040.java    2':16"       01hr:28':21"
51 student009_Yahtzee_041.java    0':27"       01hr:28':48"
52 student009_Yahtzee_042.java    0':22"       01hr:29':10"
53 student009_Yahtzee_043.java    0':27"       01hr:29':37"
54 student009_Yahtzee_044.java    2':54"       01hr:32':31"
55 student009_Yahtzee_045.java    0':33"       01hr:33':04"
56 student009_Yahtzee_046.java    0':28"       01hr:33':30"
57 student009_Yahtzee_047.java    b r e a k
58 student009_Yahtzee_048.java    1':10"       01hr:34':40"
59 student009_Yahtzee_049.java    b r e a k
60 student009_Yahtzee_050.java    4':26"       01hr:39':06"
61 student009_Yahtzee_051.java    2':10"       01hr:41':16"
62 student009_Yahtzee_052.java    1':09"       01hr:42':25"
63 student009_Yahtzee_053.java    0':24"       01hr:43':41"

```

Figure 29: Analyzing an individual submission takes a few seconds.

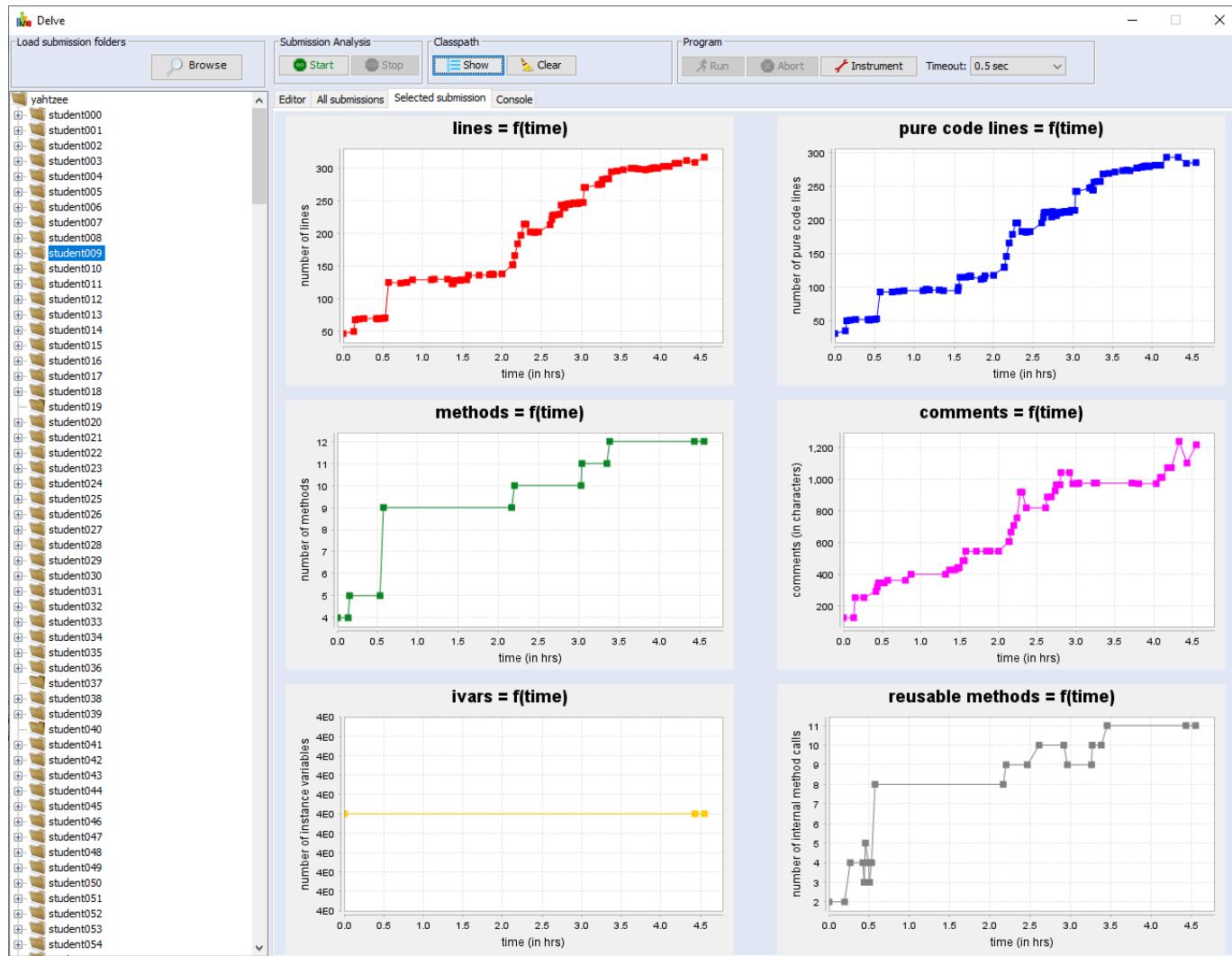


Figure 30: Compared to before setting the classpath (*Figure 10*), we notice that Delve displays the number of calls to user-defined methods which is an indicator of code reusability.

The screenshot shows the Delve tool interface with the 'Editor' tab selected. On the left is a tree view of submission folders under 'yahtzee'. The main area displays a table of snapshot details. The columns are:

- Line number
- Snapshot ID
- File name
- Time Diff
- Total Time
- Status

Sample data from the table:

Line	Snapshot	File	Time Diff	Total Time	Status
1		Number of snapshots: 150			
2					
3		Total time: 04hr:33':02"			
4		Last snapshot: student009_Yahtzee_150.java			
5		Number of lines: 286			
6		Number of methods: 12			
7					
8		Time breakdown:			
9					
10		Filename	Time Diff	Total Time	Status
11		student009_Yahtzee_001.java	s t a r t		Compilable
12		student009_Yahtzee_002.java	7':58"	00hr:07':58"	Compilable
13		student009_Yahtzee_003.java	1':07"	00hr:09':05"	Compilable
14		student009_Yahtzee_004.java	b r e a k		Compilable
15		student009_Yahtzee_005.java	b r e a k		Non-Compilable
16		student009_Yahtzee_006.java	b r e a k		Compilable
17		student009_Yahtzee_007.java	2':29"	00hr:11':34"	Compilable
18		student009_Yahtzee_008.java	4':14"	00hr:15':48"	Compilable
19		student009_Yahtzee_009.java	9':24"	00hr:25':12"	Compilable
20		student009_Yahtzee_010.java	1':00"	00hr:26':12"	Compilable
21		student009_Yahtzee_011.java	0':57"	00hr:27':09"	Compilable
22		student009_Yahtzee_012.java	2':08"	00hr:29':17"	Compilable
23		student009_Yahtzee_013.java	0':57"	00hr:30':14"	Compilable
24		student009_Yahtzee_014.java	1':30"	00hr:31':44"	Compilable
25		student009_Yahtzee_015.java	2':33"	00hr:34':17"	Compilable
26		student009_Yahtzee_016.java	b r e a k		Non-Compilable
27		student009_Yahtzee_017.java	b r e a k		Non-Compilable
28		student009_Yahtzee_018.java	b r e a k		Non-Compilable
29		student009_Yahtzee_019.java	b r e a k		Non-Compilable
30		student009_Yahtzee_020.java	b r e a k		Compilable
31		student009_Yahtzee_021.java	9':11"	00hr:43':28"	Compilable
32		student009_Yahtzee_022.java	4':35"	00hr:48':03"	Compilable
33		student009_Yahtzee_023.java	4':18"	00hr:52':21"	Compilable
34		student009_Yahtzee_024.java	7':15"	00hr:59':36"	Compilable
35		student009_Yahtzee_025.java	0':27"	01hr:00':03"	Compilable
36		student009_Yahtzee_026.java	1':01"	01hr:01':04"	Compilable
37		student009_Yahtzee_027.java	5':39"	01hr:06':43"	Compilable
38		student009_Yahtzee_028.java	2':10"	01hr:08':53"	Compilable
39		student009_Yahtzee_029.java	0':22"	01hr:09':15"	Compilable
40		student009_Yahtzee_030.java	0':29"	01hr:09':44"	Compilable
41		student009_Yahtzee_031.java	1':53"	01hr:11':37"	Compilable
42		student009_Yahtzee_032.java	1':27"	01hr:13':04"	Compilable
43		student009_Yahtzee_033.java	2':13"	01hr:15':17"	Compilable
44		student009_Yahtzee_034.java	3':41"	01hr:18':58"	Compilable
45		student009_Yahtzee_035.java	3':14"	01hr:22':12"	Compilable
46		student009_Yahtzee_036.java	1':08"	01hr:23':20"	Compilable
47		student009_Yahtzee_037.java	0':37"	01hr:23':57"	Compilable
48		student009_Yahtzee_038.java	1':18"	01hr:25':15"	Compilable
49		student009_Yahtzee_039.java	0':50"	01hr:26':05"	Compilable
50		student009_Yahtzee_040.java	2':16"	01hr:28':21"	Compilable
51		student009_Yahtzee_041.java	0':27"	01hr:28':48"	Compilable
52		student009_Yahtzee_042.java	0':22"	01hr:29':10"	Compilable
53		student009_Yahtzee_043.java	0':27"	01hr:29':37"	Compilable
54		student009_Yahtzee_044.java	2':54"	01hr:32':31"	Compilable
55		student009_Yahtzee_045.java	0':33"	01hr:33':04"	Compilable
56		student009_Yahtzee_046.java	0':28"	01hr:33':30"	Compilable
57		student009_Yahtzee_047.java	b r e a k		Compilable
58		student009_Yahtzee_048.java	1':10"	01hr:34':40"	Compilable
59		student009_Yahtzee_049.java	b r e a k		Compilable
60		student009_Yahtzee_050.java	4':26"	01hr:39':06"	Compilable
61		student009_Yahtzee_051.java	2':10"	01hr:41':16"	Compilable
62		student009_Yahtzee_052.java	1':09"	01hr:42':25"	Compilable
63		student009_Yahtzee_053.java	0':24"	01hr:43':41"	Compilable

Figure 31: The *Editor* tab displays an extra column for the status of each snapshot.

7. Code Instrumentation

Delve facilitates testing the program functionality at scale, by assisting the user instrument the code to save the program state. The idea is that if a program is **deterministic** and we can capture its state over time, we can run a program with correct implementation, collect the states it produced and use them as reference to compare with the states that any student program produces. If they match, the student program is functionally correct.

The goal is to instrument the program to collect the states it goes through during execution. It is equally important to make the program complete without delays or stalls. This means that any user interaction (e.g., wait for user input) should be avoided. Ideally, one should get rid of user interface which is adding delay.

One of the big challenges is to guarantee that the program execution is always **deterministic**. It should not matter how many times we execute the program. Every execution must produce an identical result. To achieve the repeatable behavior, Delve's approach is to assist the user replace any source of randomness in the program.

This User Guide illustrates how to instrument the code using an example that falls in the hardest category. [Yahtzee](#) is a CS1 assignment with all the factors that make instrumentation difficult: **i) user interface** (slows down the execution), **ii) user input** (stalls the execution), **iii) randomness** (non-deterministic behavior). Note that code instrumentation is easier in assignments that lack one or more of those factors.

The remaining section covers the instrumentation process step-by-step.

Before the instrumentation the user has to set the timeout value, which is the amount of time (in seconds) to wait for the program complete and terminate it if still running. This is important to avoid situations of infinite-loops etc. The user can choose the proper value (or no timeout at all) based on the nature of the program.

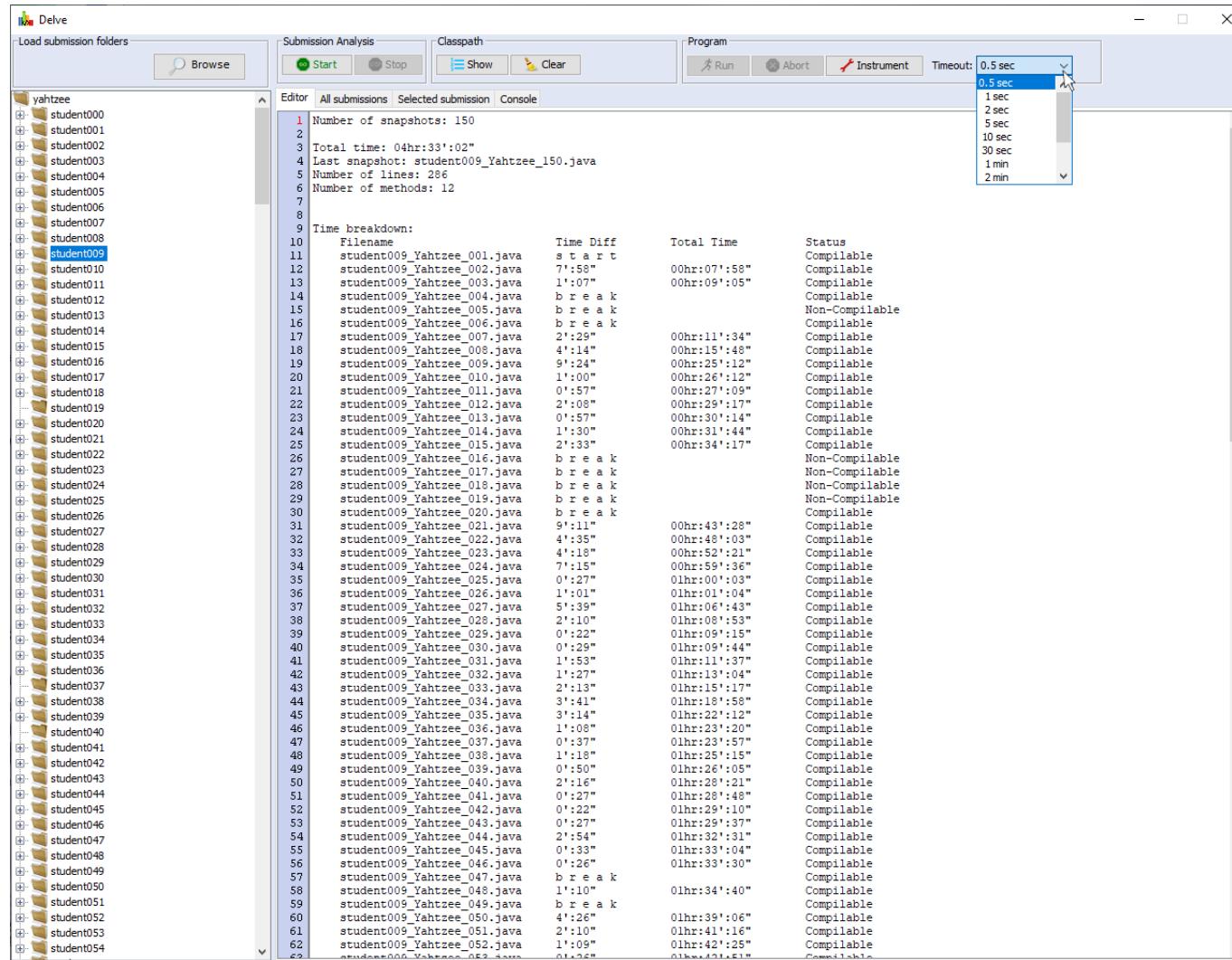


Figure 32: The user can select the amount of time to wait for the program to complete or else terminate it.

To initiate the instrumentation process, the user has to press button **Instrument**.

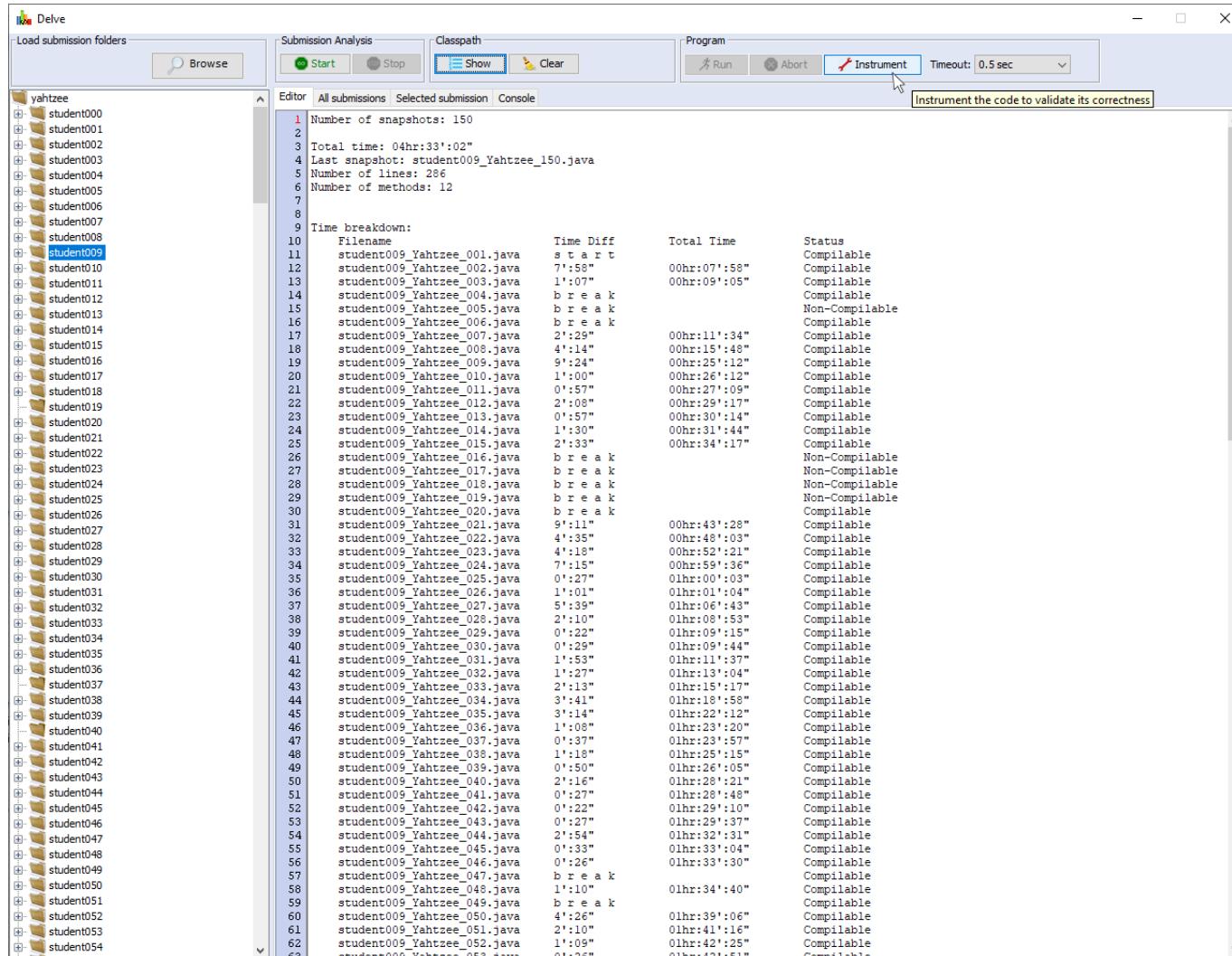


Figure 33: Pressing button **Instrument** initiates the instrumentation process.

7.1 Instrumentation Overview

At this point, a dialog window to guide the user shows up. The first screen lists the instrumentation steps.

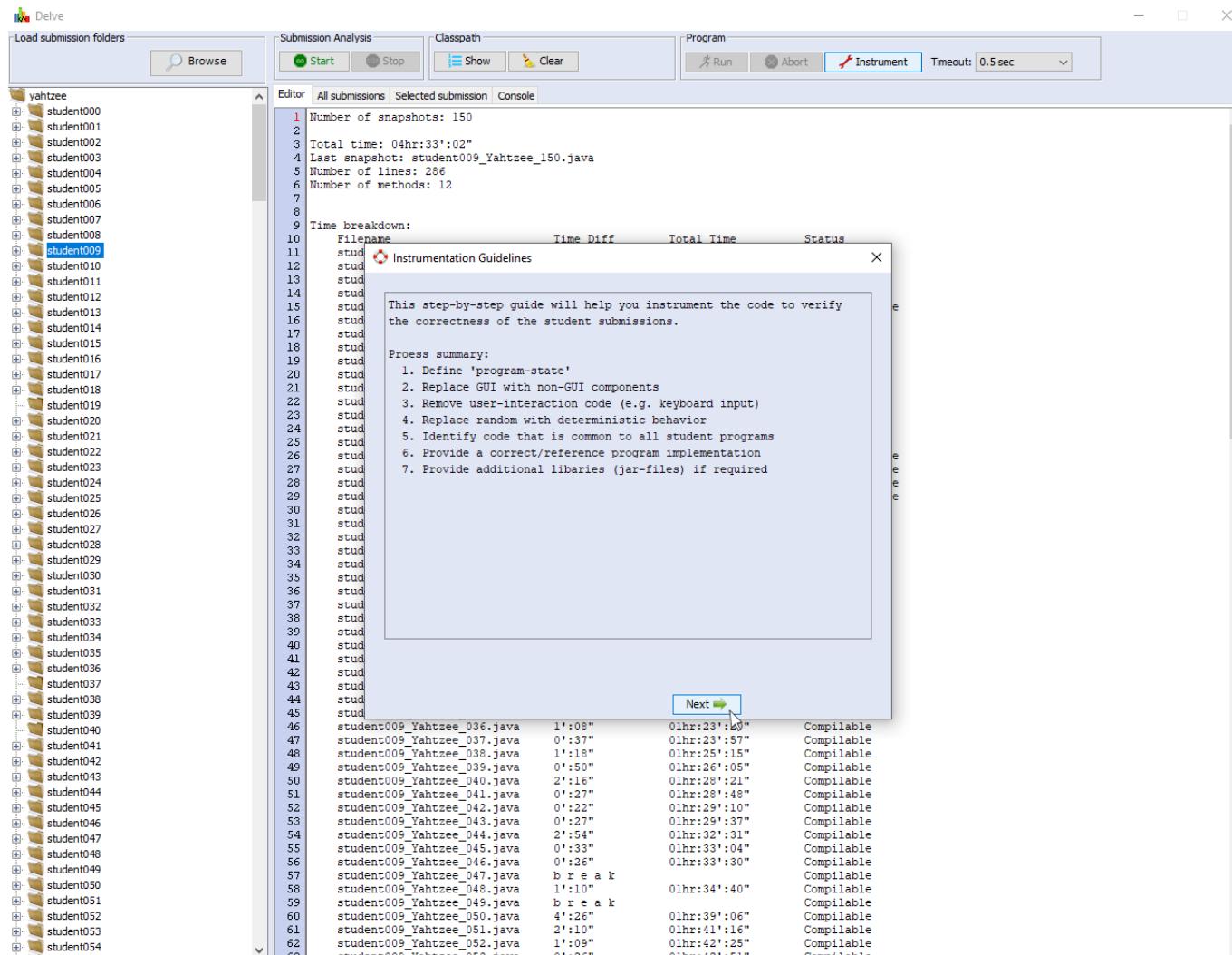


Figure 34: The first dialog screen with an overview of the instrumentation steps.

7.2 Step 1: Program State

The first step is to define what **program state** means for this assignment. This varies between different programs.

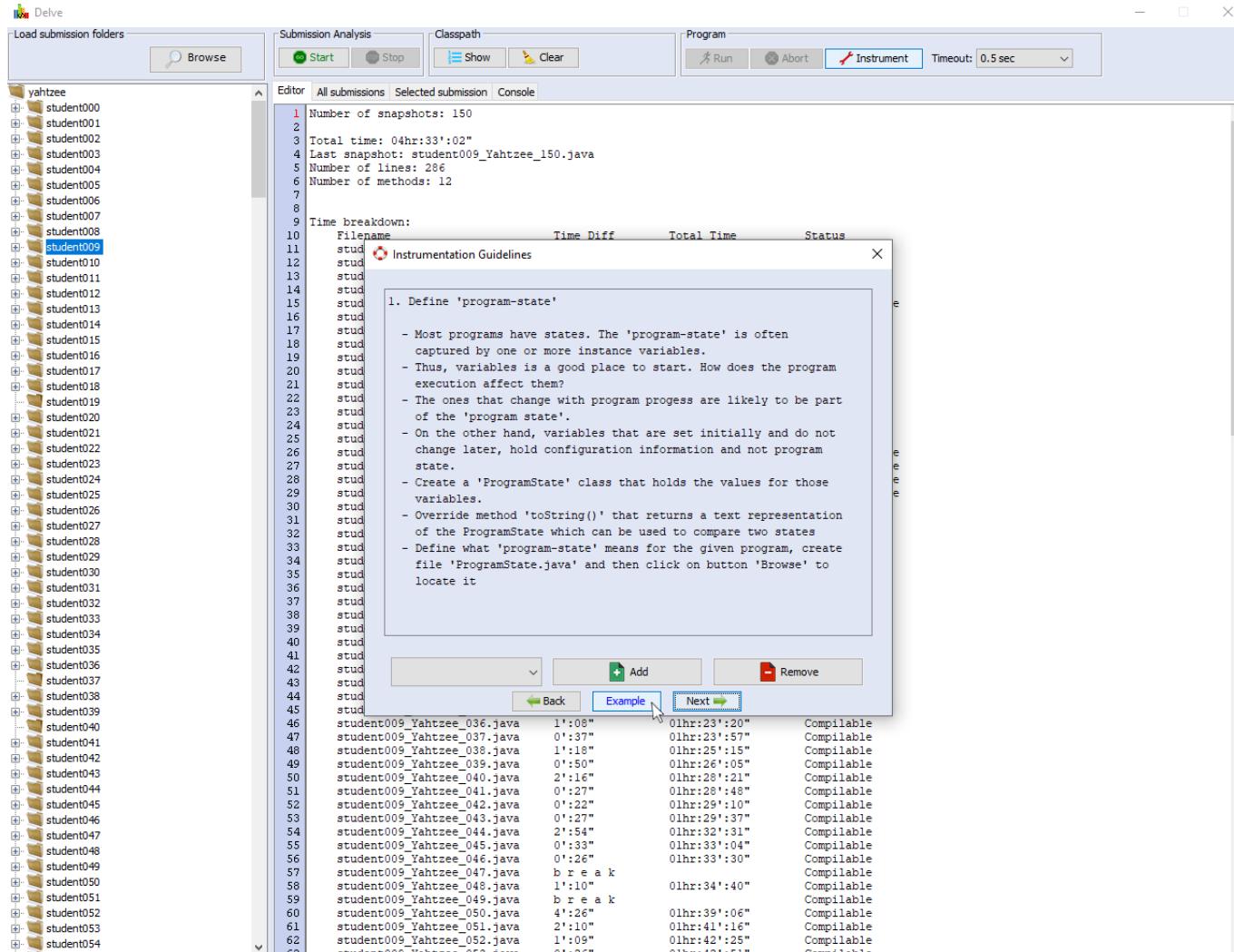


Figure 35: Delve provides a number of hints to help the user define the program state for the given assignment.

To get a better idea about the **program state**, the user can go over an example.

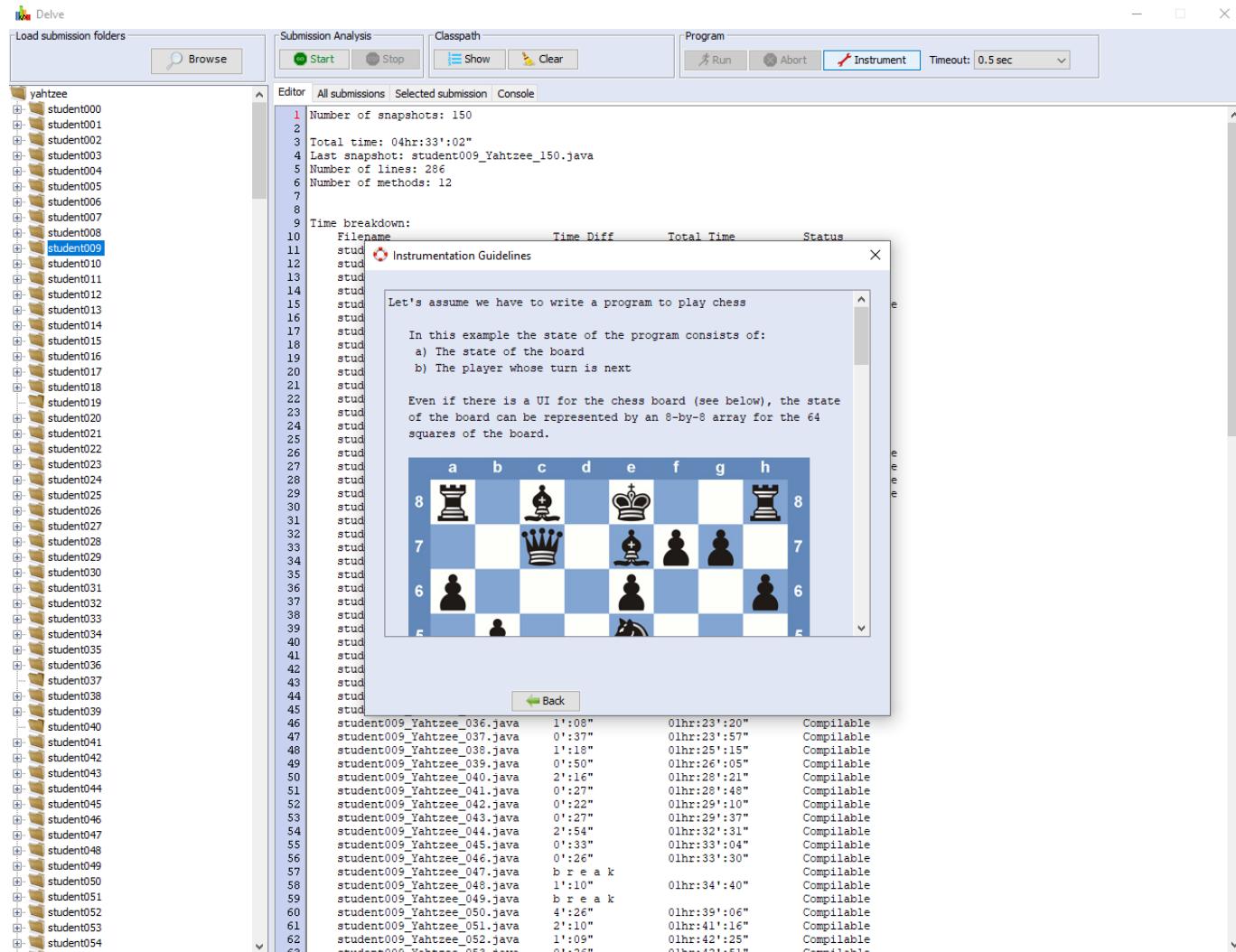


Figure 36: An example to explain what program state means including the source code.

Delve

Load submission folders

Submission Analysis

Classpath

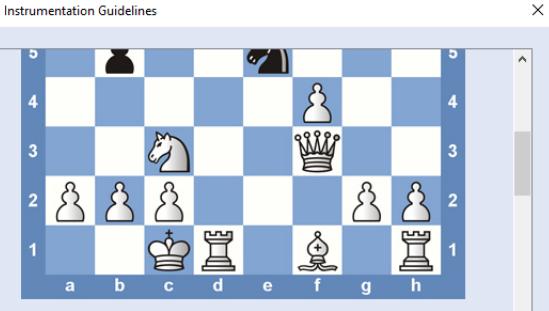
Program Timeout: 0.5 sec

Editor All submissions Selected submission Console

```

1 Number of snapshots: 150
2
3 Total time: 04hr:33':02"
4 Last snapshot: student009_Yahtzee_150.java
5 Number of lines: 286
6 Number of methods: 12
7
8
9 Time breakdown:
10   Filename           Time Diff      Total Time      Status
11   stud                Instrumentation Guidelines
12   stud
13   stud
14   stud
15   stud
16   stud
17   stud
18   stud
19   stud
20   stud
21   stud
22   stud
23   stud
24   stud
25   stud
26   stud
27   stud
28   stud
29   stud
30   stud
31   stud
32   stud
33   stud
34   stud
35   stud
36   stud
37   stud
38   stud
39   stud
40   stud
41   stud
42   stud
43   stud
44   stud
45   stud
46   student009_Yahtzee_036.java  1':08"    0lhr:23':20"  Compilable
47   student009_Yahtzee_037.java  0':37"    0lhr:23':57"  Compilable
48   student009_Yahtzee_038.java  1':18"    0lhr:25':15"  Compilable
49   student009_Yahtzee_039.java  0':50"    0lhr:26':05"  Compilable
50   student009_Yahtzee_040.java  2':16"    0lhr:28':21"  Compilable
51   student009_Yahtzee_041.java  0':27"    0lhr:28':48"  Compilable
52   student009_Yahtzee_042.java  0':22"    0lhr:29':10"  Compilable
53   student009_Yahtzee_043.java  0':27"    0lhr:29':37"  Compilable
54   student009_Yahtzee_044.java  2':54"    0lhr:32':31"  Compilable
55   student009_Yahtzee_045.java  0':33"    0lhr:33':04"  Compilable
56   student009_Yahtzee_046.java  0':26"    0lhr:33':30"  Compilable
57   student009_Yahtzee_047.java  b r e a k     0lhr:34':40"  Compilable
58   student009_Yahtzee_048.java  1':10"    0lhr:39":06"  Compilable
59   student009_Yahtzee_049.java  b r e a k     0lhr:41':16"  Compilable
60   student009_Yahtzee_050.java  4':26"    0lhr:42':25"  Compilable
61   student009_Yahtzee_051.java  2':10"    0lhr:42':25"  Compilable
62   student009_Yahtzee_052.java  1':09"    0lhr:42':25"  Compilable
63   student009_Yahtzee_053.java  0':25"    0lhr:42':25"  Compilable

```



On each square there can be a piece or nothing. For the pieces we can use classes like 'Pawn', 'Knight', 'Bishop', 'Rook', 'Queen' and 'King' and have each of those classes extend 'Piece'. Thus, the state of the chess board can be represented by a 2D array 'Piece[][] chessboard' (8-by-8 array). If there is no piece on a given square then the element in the array for the given

Figure 37: An example to explain what program state means including the source code.

The screenshot shows the Delve debugger interface. In the top navigation bar, there are tabs for "Submission Analysis" (which is selected), "Classpath", "Program", and "Instrument". The "Instrument" tab has a dropdown for "Timeout: 0.5 sec". Below the tabs, there's a search bar labeled "Browse" and a "Start" button.

The left sidebar displays a tree view of submission folders under "yahtzee", including "student000" through "student054".

The main pane shows the following output:

```

1 Number of snapshots: 150
2
3 Total time: 04hr:33':02"
4 Last snapshot: student009_Yahtzee_150.java
5 Number of lines: 286
6 Number of methods: 12
7
8
9 Time breakdown:
10   Filename           Time Diff      Total Time      Status
11   stud               0.000ms       0.000ms       Compilable
12   stud               0.000ms       0.000ms       Compilable
13   stud               0.000ms       0.000ms       Compilable
14   stud               0.000ms       0.000ms       Compilable
15   stud               0.000ms       0.000ms       Compilable
16   stud               0.000ms       0.000ms       Compilable
17   stud               0.000ms       0.000ms       Compilable
18   stud               0.000ms       0.000ms       Compilable
19   stud               0.000ms       0.000ms       Compilable
20   stud               0.000ms       0.000ms       Compilable
21   stud               0.000ms       0.000ms       Compilable
22   stud               0.000ms       0.000ms       Compilable
23   stud               0.000ms       0.000ms       Compilable
24   stud               0.000ms       0.000ms       Compilable
25   stud               0.000ms       0.000ms       Compilable
26   stud               0.000ms       0.000ms       Compilable
27   stud               0.000ms       0.000ms       Compilable
28   stud               0.000ms       0.000ms       Compilable
29   stud               0.000ms       0.000ms       Compilable
30   stud               0.000ms       0.000ms       Compilable
31   stud               0.000ms       0.000ms       Compilable
32   stud               0.000ms       0.000ms       Compilable
33   stud               0.000ms       0.000ms       Compilable
34   stud               0.000ms       0.000ms       Compilable
35   stud               0.000ms       0.000ms       Compilable
36   stud               0.000ms       0.000ms       Compilable
37   stud               0.000ms       0.000ms       Compilable
38   stud               0.000ms       0.000ms       Compilable
39   stud               0.000ms       0.000ms       Compilable
40   stud               0.000ms       0.000ms       Compilable
41   stud               0.000ms       0.000ms       Compilable
42   stud               0.000ms       0.000ms       Compilable
43   stud               0.000ms       0.000ms       Compilable
44   stud               0.000ms       0.000ms       Compilable
45   stud               0.000ms       0.000ms       Compilable
46   student009_Yahtzee_036.java 1':08"    0lhr:23':20"   Compilable
47   student009_Yahtzee_037.java 0':37"    0lhr:23':57"   Compilable
48   student009_Yahtzee_038.java 1':18"    0lhr:25':15"   Compilable
49   student009_Yahtzee_039.java 0':50"    0lhr:26':05"   Compilable
50   student009_Yahtzee_040.java 2':16"    0lhr:28':21"   Compilable
51   student009_Yahtzee_041.java 0':27"    0lhr:28':48"   Compilable
52   student009_Yahtzee_042.java 0':22"    0lhr:29':10"   Compilable
53   student009_Yahtzee_043.java 0':27"    0lhr:29':37"   Compilable
54   student009_Yahtzee_044.java 2':54"    0lhr:32':31"   Compilable
55   student009_Yahtzee_045.java 0':33"    0lhr:33':04"   Compilable
56   student009_Yahtzee_046.java 0':26"    0lhr:33':30"   Compilable
57   student009_Yahtzee_047.java b r e a k 0lhr:33":41"   Compilable
58   student009_Yahtzee_048.java 1':10"    0lhr:34':40"   Compilable
59   student009_Yahtzee_049.java b r e a k 0lhr:34":41"   Compilable
60   student009_Yahtzee_050.java 4":26"    0lhr:39":06"   Compilable
61   student009_Yahtzee_051.java 2':10"    0lhr:41':16"   Compilable
62   student009_Yahtzee_052.java 1':09"    0lhr:42":25"   Compilable
63   student009_Yahtzee_053.java 0":42"    0lhr:43":41"   Compilable

```

A tooltip window titled "Instrumentation Guidelines" is open over the code editor, containing the following text:

on a given square then the element in the array for the given (row, col) indices will be null.
In this example, file 'ProgramState.java' looks like:

```

import <homework assignment package>.Piece;

public class ProgramState
{
    private final Piece[][] chessboard;
    private final int nextPlayer;

    public ProgramState(Piece[][] chessboard, int nextPlayer) {
        this.chessboard = chessboard;
        this.nextPlayer = nextPlayer;
    }

    public Piece[][] getChessboard() {
        return chessboard;
    }
}

```

At the bottom of the tooltip window is a "Back" button.

Figure 38: An example to explain what program state means including the source code.

The screenshot shows the Delve tool interface. On the left, there's a tree view of submission folders under 'Load submission folders'. The main area has tabs for 'Submission Analysis', 'Classpath', and 'Program'. The 'Submission Analysis' tab is active, showing the following output:

```

1 Number of snapshots: 150
2
3 Total time: 04hr:33':02"
4 Last snapshot: student009_Yahtzee_150.java
5 Number of lines: 286
6 Number of methods: 12
7
8 Time breakdown:
9
10   Filename           Time Diff      Total Time      Status
11   stud
12   stud
13   stud
14   stud
15   stud
16   stud
17   stud
18   stud
19   stud
20   stud
21   stud
22   stud
23   stud
24   stud
25   stud
26   stud
27   stud
28   stud
29   stud
30   stud
31   stud
32   stud
33   stud
34   stud
35   stud
36   stud
37   stud
38   stud
39   stud
40   stud
41   stud
42   stud
43   stud
44   stud
45   stud
46 student009_Yahtzee_036.java 1':08" 0lhr:23':20" Compilable
47 student009_Yahtzee_037.java 0':37" 0lhr:23':57" Compilable
48 student009_Yahtzee_038.java 1':18" 0lhr:25':15" Compilable
49 student009_Yahtzee_039.java 0':50" 0lhr:26':05" Compilable
50 student009_Yahtzee_040.java 2':16" 0lhr:28':21" Compilable
51 student009_Yahtzee_041.java 0':27" 0lhr:28':48" Compilable
52 student009_Yahtzee_042.java 0':22" 0lhr:29':10" Compilable
53 student009_Yahtzee_043.java 0':27" 0lhr:29':37" Compilable
54 student009_Yahtzee_044.java 2':54" 0lhr:32':31" Compilable
55 student009_Yahtzee_045.java 0':33" 0lhr:33':04" Compilable
56 student009_Yahtzee_046.java 0':26" 0lhr:33':30" Compilable
57 student009_Yahtzee_047.java b r e a k 0lhr:34":00" Compilable
58 student009_Yahtzee_048.java 1':10" 0lhr:34':40" Compilable
59 student009_Yahtzee_049.java b r e a k 0lhr:34":40" Compilable
60 student009_Yahtzee_050.java 4':26" 0lhr:39':06" Compilable
61 student009_Yahtzee_051.java 2':10" 0lhr:41':16" Compilable
62 student009_Yahtzee_052.java 1':09" 0lhr:42':25" Compilable
63 student009_Yahtzee_053.java 0lhr:43":11" 0lhr:43":11" Compilable

```

A 'Instrumentation Guidelines' pop-up window is open, showing the following Java code and instructions:

```

public Piece[][] getChessboard() {
    return chessboard;
}

public int getNextPlayer() {
    return nextPlayer;
}

```

The instructions say: "The last thing that is missing from this code is to override method 'public String toString()' to provide a meaningful text representation of the ProgramState which we can use also to verify if two states are the same. Thus, the complete file 'ProgramState.java' for chess is:

```

import <homework assignment package>.Piece;

public class ProgramState
{

```

Figure 39: An example to explain what program state means including the source code.

The screenshot shows the Delve debugger interface with the following details:

- Load submission folders:** A tree view showing submissions from students 000 to 054, with "yahtzee" selected.
- Submission Analysis:** Buttons for Start, Stop, Show, Clear, Run, Abort, Instrument, and Timeout (set to 0.5 sec).
- Editor:** Displays the source code for `student009_Yahtzee_150.java`. The code defines a class `ProgramState` with methods to get the chessboard and the next player.
- Time breakdown:** A table showing execution time for each line of code. The first few lines are:

Line	Time Diff	Total Time	Status	
1				
2				
3				
4				
5				
6				
7				
8				
9				
10	Filepath	Time Diff	Total Time	Status
11	stud			
12	stud			
13	stud			
14	stud			
15	stud			
16	stud			
17	stud			
18	stud			
19	stud			
20	stud			
21	stud			
22	stud			
23	stud			
24	stud			
25	stud			
26	stud			
27	stud			
28	stud			
29	stud			
30	stud			
31	stud			
32	stud			
33	stud			
34	stud			
35	stud			
36	stud			
37	stud			
38	stud			
39	stud			
40	stud			
41	stud			
42	stud			
43	stud			
44	stud			
45	stud			
- Instrumentation Guidelines:** A modal window showing the generated Java code with instrumentation points. It includes imports, a constructor, and methods for getting the chessboard and the next player.
- Table of snapshots:** A table listing 62 snapshots with their names, times, and compilation status. Most are compilable, with some marked as "break".

Figure 40: An example to explain what program state means including the source code.

The screenshot shows the Delve debugger interface. On the left, a tree view displays submission folders under 'yahzee'. The 'student009' folder is selected. The main window has tabs for 'Submission Analysis', 'Classpath', 'Program', and 'Instrument'. The 'Instrument' tab is active, showing a timeout of 0.5 sec. The 'Program' tab displays a stack trace and source code. The stack trace shows multiple frames for 'stud' and 'stud' with instrumentation guidelines. The source code is a Java class with methods for printing chessboard pieces and calculating total time.

```

1 Number of snapshots: 150
2
3 Total time: 04hr:33':02"
4 Last snapshot: student009_Yahzee_150.java
5 Number of lines: 286
6 Number of methods: 12
7
8
9 Time breakdown:
10   Filename           Time Diff      Total Time      Status
11   stud               0ms          0ms             Compilable
12   stud               0ms          0ms             Compilable
13   stud               0ms          0ms             Compilable
14   stud               0ms          0ms             Compilable
15   stud               0ms          0ms             Compilable
16   stud               0ms          0ms             Compilable
17   stud               0ms          0ms             Compilable
18   stud               0ms          0ms             Compilable
19   stud               0ms          0ms             Compilable
20   stud               0ms          0ms             Compilable
21   stud               0ms          0ms             Compilable
22   stud               0ms          0ms             Compilable
23   stud               0ms          0ms             Compilable
24   stud               0ms          0ms             Compilable
25   stud               0ms          0ms             Compilable
26   stud               0ms          0ms             Compilable
27   stud               0ms          0ms             Compilable
28   stud               0ms          0ms             Compilable
29   stud               0ms          0ms             Compilable
30   stud               0ms          0ms             Compilable
31   stud               0ms          0ms             Compilable
32   stud               0ms          0ms             Compilable
33   stud               0ms          0ms             Compilable
34   stud               0ms          0ms             Compilable
35   stud               0ms          0ms             Compilable
36   stud               0ms          0ms             Compilable
37   stud               0ms          0ms             Compilable
38   stud               0ms          0ms             Compilable
39   stud               0ms          0ms             Compilable
40   stud               0ms          0ms             Compilable
41   stud               0ms          0ms             Compilable
42   stud               0ms          0ms             Compilable
43   stud               0ms          0ms             Compilable
44   stud               0ms          0ms             Compilable
45   stud               0ms          0ms             Compilable
46   student009_Yahzee_036.java  0*:08"    01hr:23':20"  Compilable
47   student009_Yahzee_037.java  0*:37"    01hr:23':57"  Compilable
48   student009_Yahzee_038.java  1*:18"    01hr:25':15"  Compilable
49   student009_Yahzee_039.java  0*:50"    01hr:26':05"  Compilable
50   student009_Yahzee_040.java  2*:16"    01hr:28':21"  Compilable
51   student009_Yahzee_041.java  0*:27"    01hr:28':48"  Compilable
52   student009_Yahzee_042.java  0*:22"    01hr:29':10"  Compilable
53   student009_Yahzee_043.java  0*:27"    01hr:29':37"  Compilable
54   student009_Yahzee_044.java  2*:54"    01hr:32':31"  Compilable
55   student009_Yahzee_045.java  0*:33"    01hr:33':04"  Compilable
56   student009_Yahzee_046.java  0*:26"    01hr:33':30"  Compilable
57   student009_Yahzee_047.java  b r e a k     01hr:34':40"  Compilable
58   student009_Yahzee_048.java  1*:10"    b r e a k     Compilable
59   student009_Yahzee_049.java  b r e a k     b r e a k     Compilable
60   student009_Yahzee_050.java  4*:26"    01hr:39':06"  Compilable
61   student009_Yahzee_051.java  2*:10"    01hr:41':16"  Compilable
62   student009_Yahzee_052.java  1*:09"    01hr:42':25"  Compilable
63   student009_Yahzee_053.java  0*:45"    01hr:43':41"  Compilable

```

Figure 41: An example to explain what program state means including the source code.

To add the code definition for the **program state**, the user can click the **Add** button

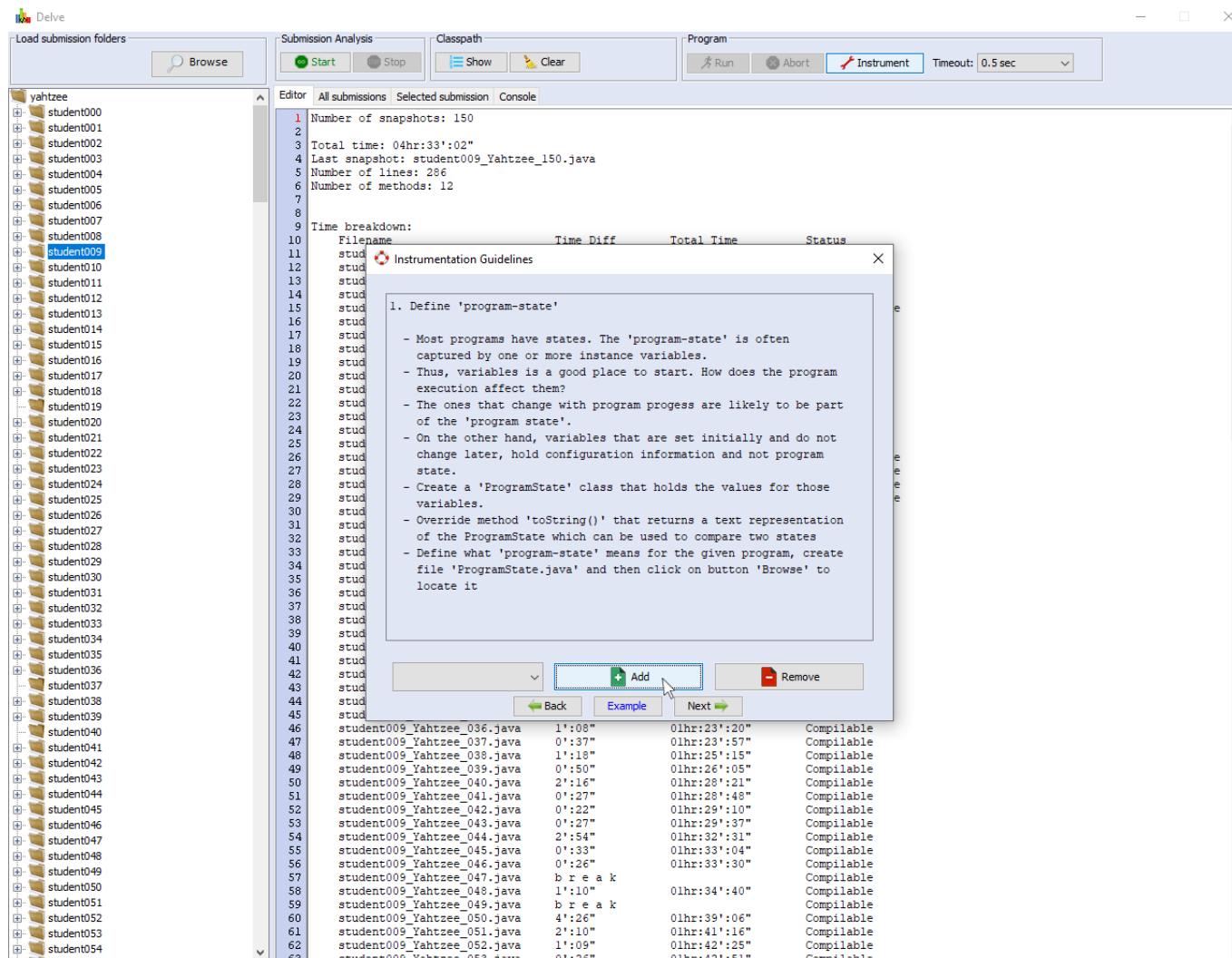


Figure 42: Click the **Add** button to locate the file with the code definition for the *program state*.

and then locate that file (i.e., **ProgramState.java**).

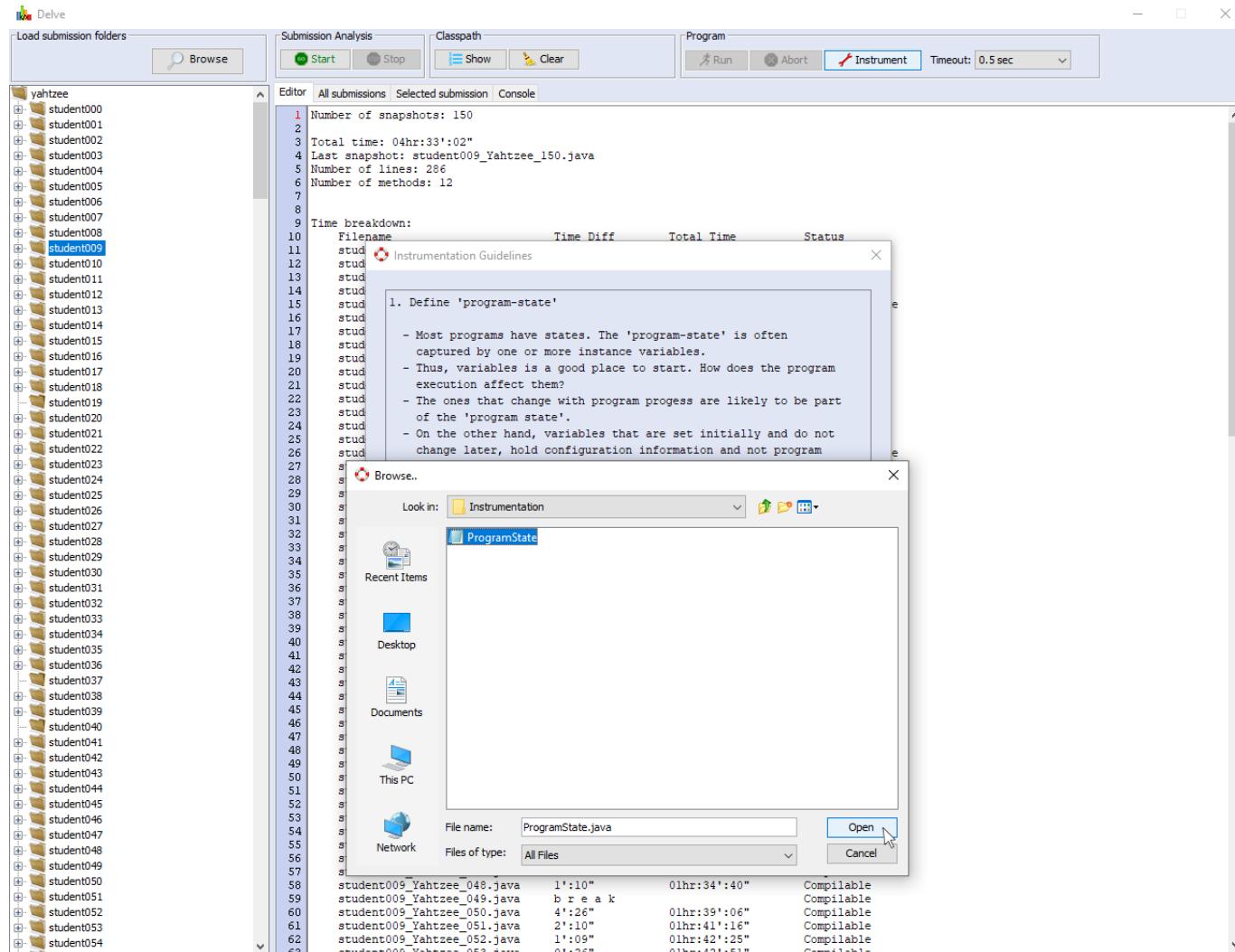
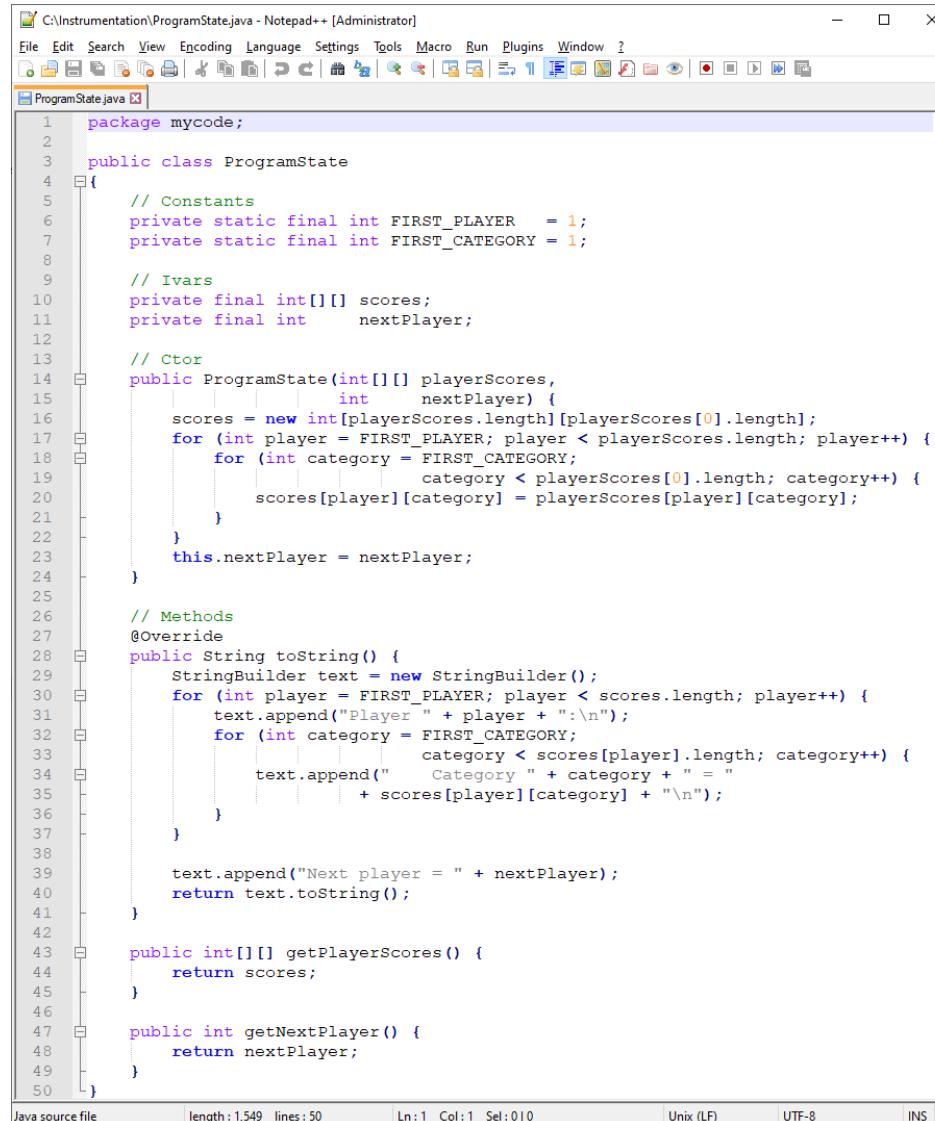


Figure 43: The user can now locate the file with the code for the program state.

This code can contain any package declaration (in this example ‘`package mycode;`’). The only required method is **public String toString()** which is used to compare a state with another.



The screenshot shows a Notepad++ window with the file `ProgramState.java` open. The code defines a class `ProgramState` with various fields, constants, and methods. The `toString()` method is highlighted, showing its implementation which builds a string representation of the game state. The Notepad++ interface includes a toolbar, menu bar, and status bar at the bottom.

```
1 package mycode;
2
3 public class ProgramState {
4
5     // Constants
6     private static final int FIRST_PLAYER = 1;
7     private static final int FIRST_CATEGORY = 1;
8
9     // Ivars
10    private final int[][] scores;
11    private final int nextPlayer;
12
13    // Ctor
14    public ProgramState(int[][] playerScores,
15                        int nextPlayer) {
16        scores = new int[playerScores.length][playerScores[0].length];
17        for (int player = FIRST_PLAYER; player < playerScores.length; player++) {
18            for (int category = FIRST_CATEGORY;
19                 category < playerScores[0].length; category++) {
20                scores[player][category] = playerScores[player][category];
21            }
22        }
23        this.nextPlayer = nextPlayer;
24    }
25
26    // Methods
27    @Override
28    public String toString() {
29        StringBuilder text = new StringBuilder();
30        for (int player = FIRST_PLAYER; player < scores.length; player++) {
31            text.append("Player " + player + ":\n");
32            for (int category = FIRST_CATEGORY;
33                 category < scores[player].length; category++) {
34                text.append("    Category " + category + " = "
35                           + scores[player][category] + "\n");
36            }
37        }
38
39        text.append("Next player = " + nextPlayer);
40        return text.toString();
41    }
42
43    public int[][] getPlayerScores() {
44        return scores;
45    }
46
47    public int getNextPlayer() {
48        return nextPlayer;
49    }
50}
```

Figure 44: The code for the *program state* in our example.

At this point we have completed the first step. Note that this step is required for instrumentation.

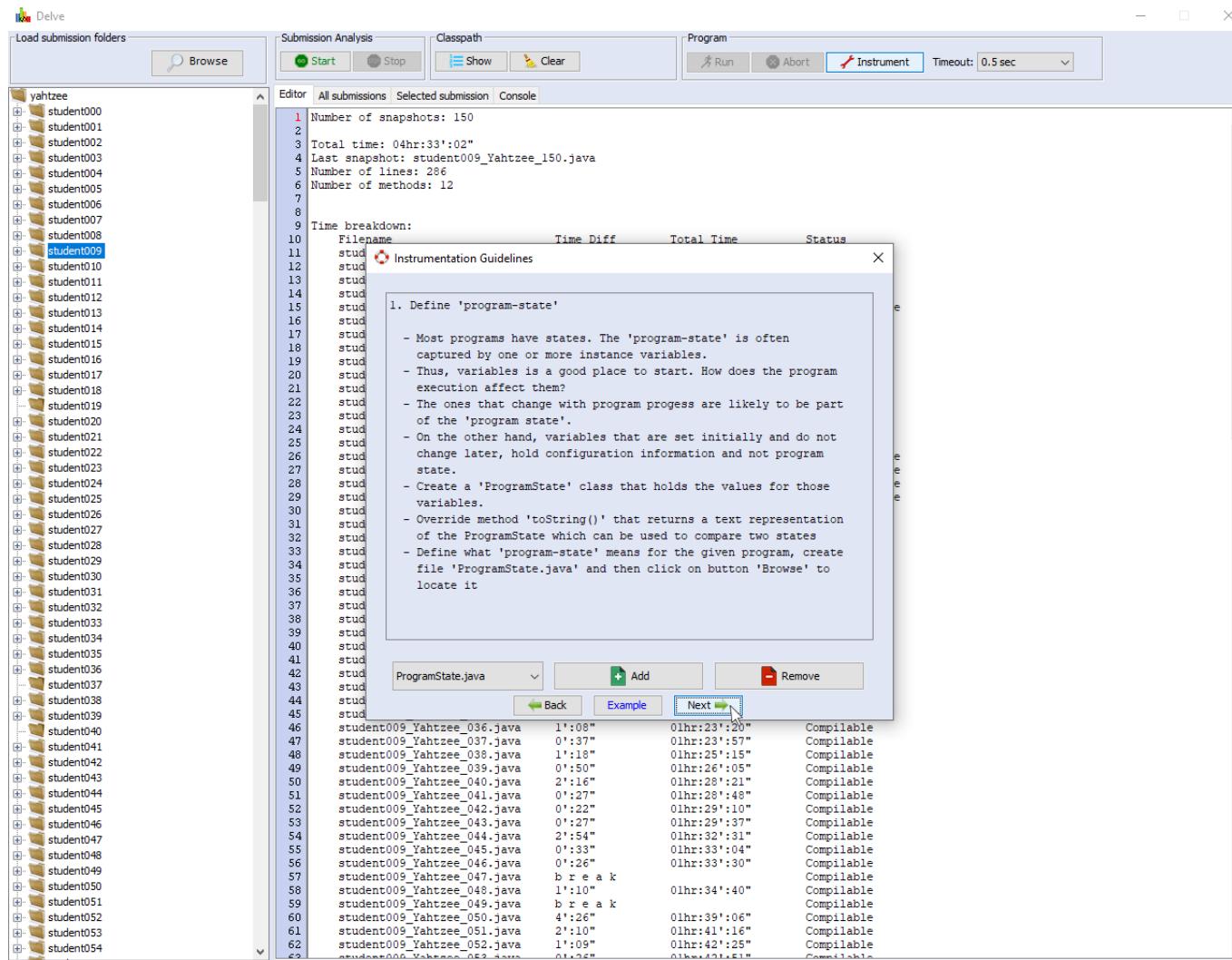


Figure 45: The first step is complete. Ready to move on to the second step.

7.3 Step 2: Replace GUI with non-GUI components (bypass UI behavior)

The second step is optional and in fact not even required for programs without a UI. However, it is highly recommended.

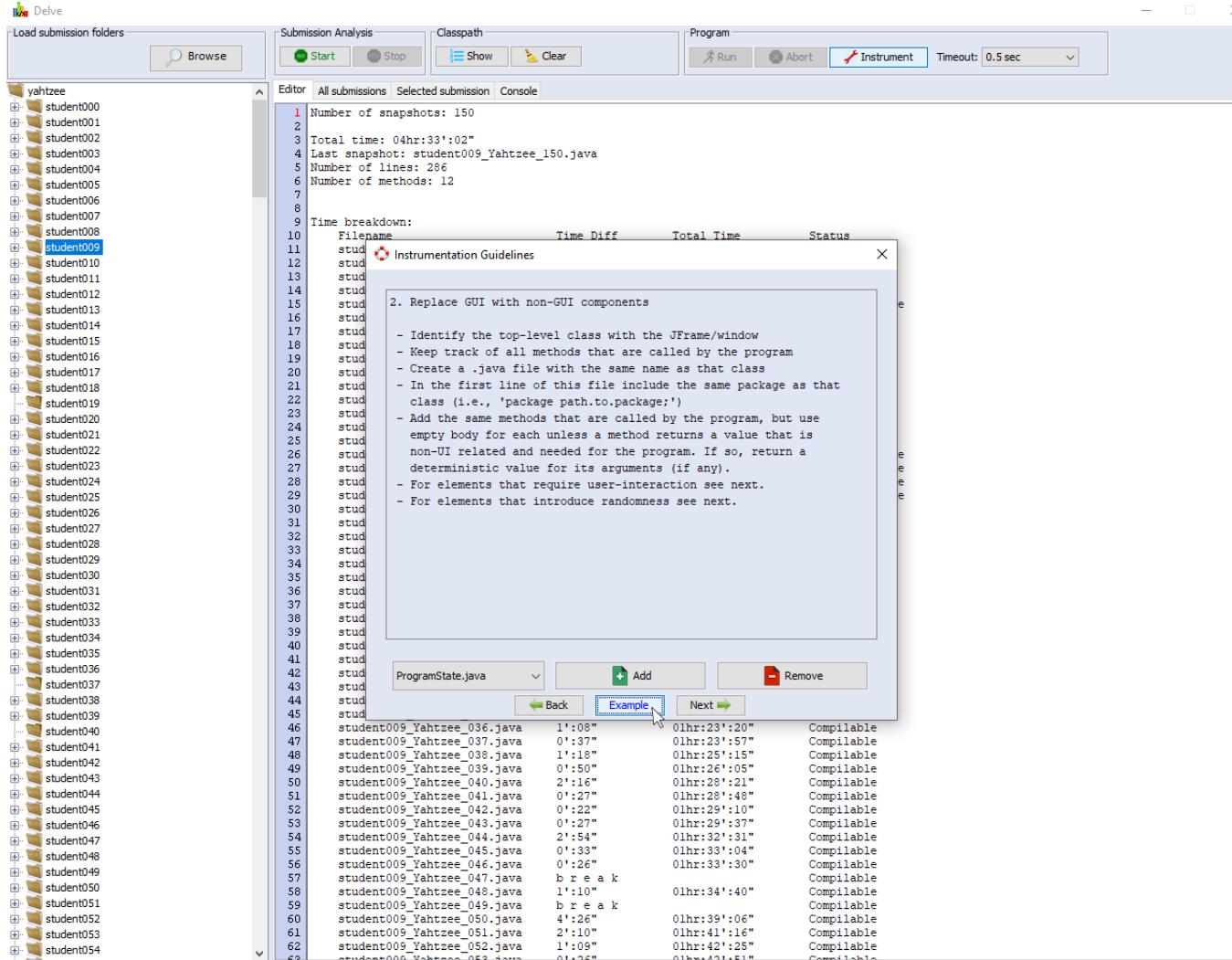


Figure 46: Second step: Replacing GUI with non-GUI components.

Again, Delve offers an example to illustrate how this works in practice.

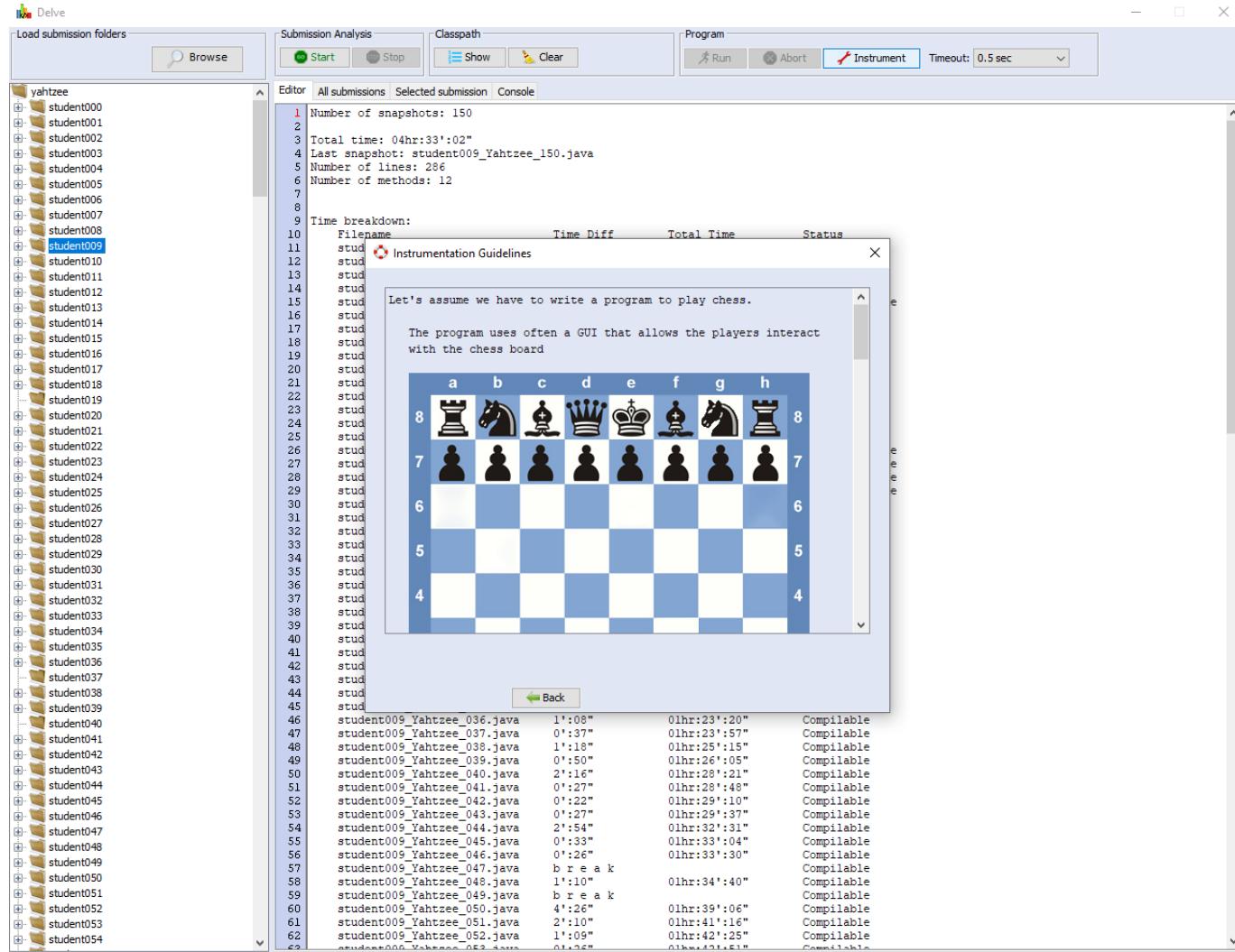


Figure 47: An example that illustrates how to replace GUI with non-GUI components.

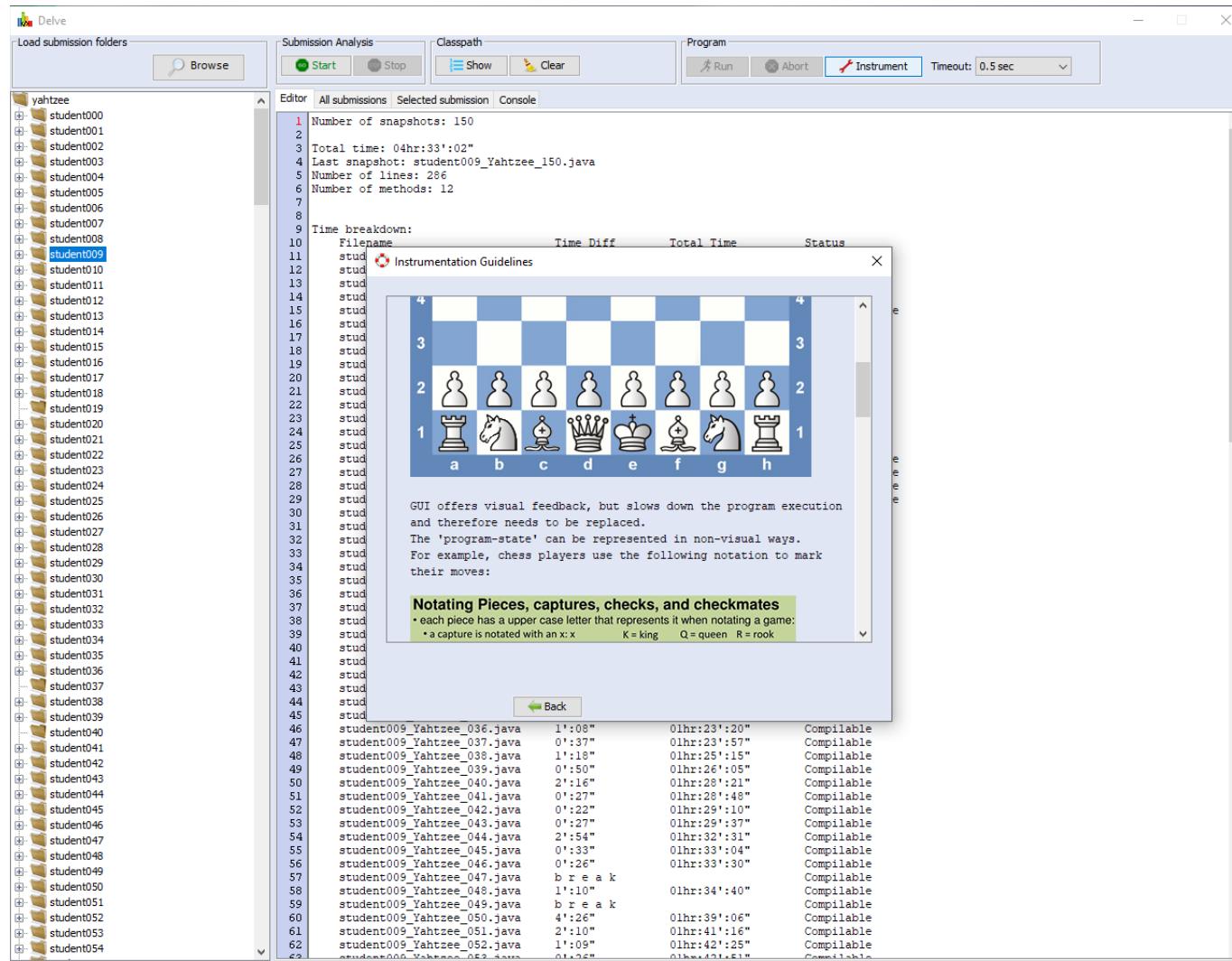


Figure 48: An example that illustrates how to replace GUI with non-GUI components.

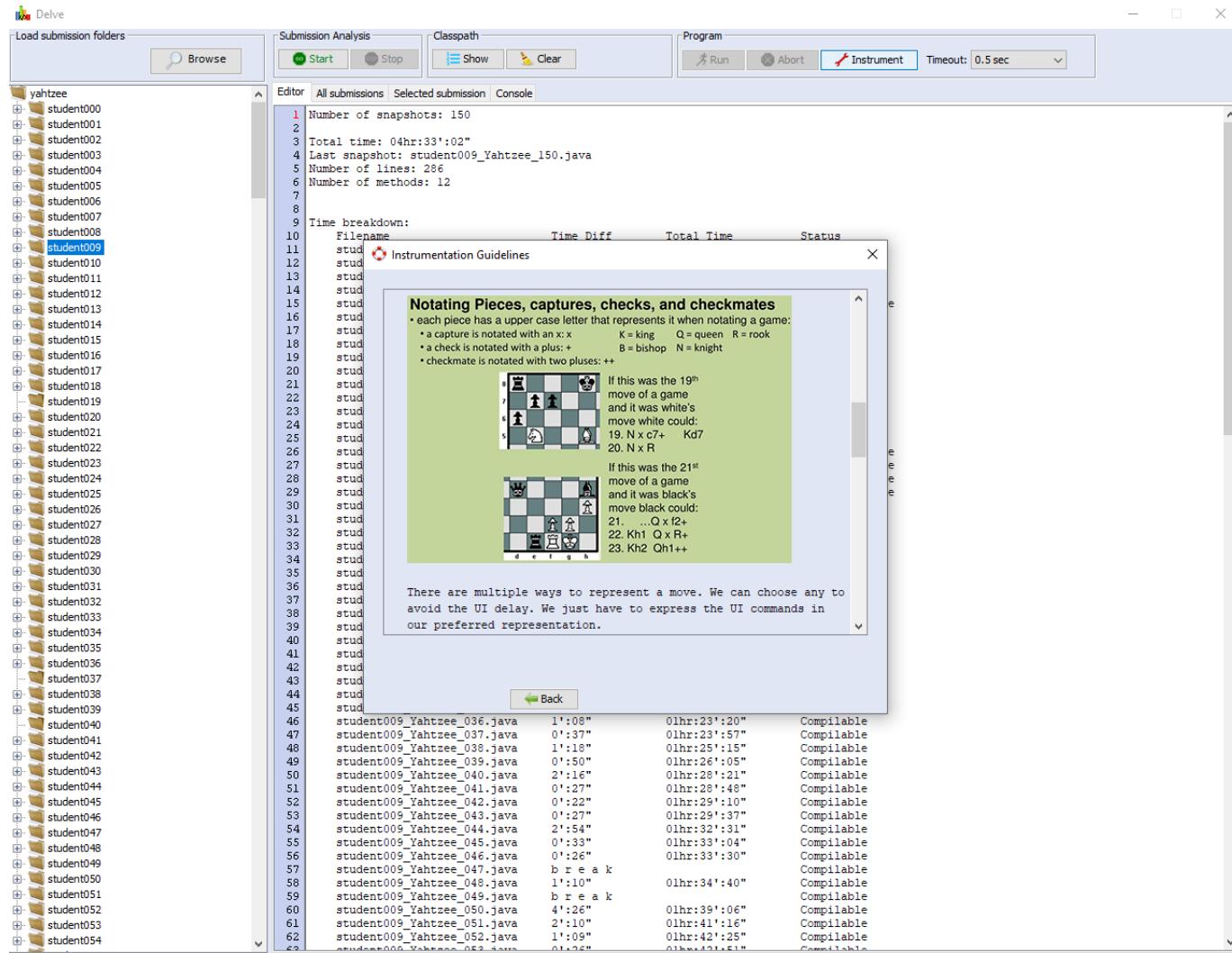


Figure 49: An example that illustrates how to replace GUI with non-GUI components.

The screenshot shows the Delve tool interface. On the left, there's a tree view of submission folders under 'Load submission folders'. A folder named 'student09' is selected. The main area is a code editor with the following content:

```

1 Number of snapshots: 150
2
3 Total time: 04hr:33':02"
4 Last snapshot: student009_Yahtzee_150.java
5 Number of lines: 286
6 Number of methods: 12
7
8
9 Time breakdown:
10   Filename           Time Diff      Total Time      Status
11   stud
12   stud
13   stud
14   stud
15   stud
16   stud
17   stud
18   stud
19   stud
20   stud
21   stud
22   stud
23   stud
24   stud
25   stud
26   stud
27   stud
28   stud
29   stud
30   stud
31   stud
32   stud
33   stud
34   stud
35   stud
36   stud
37   stud
38   stud
39   stud
40   stud
41   stud
42   stud
43   stud
44   stud
45   stud
46 student009_Yahtzee_036.java  1':08"    0lhr:23':20"  Compilable
47 student009_Yahtzee_037.java  0':37"    0lhr:23':57"  Compilable
48 student009_Yahtzee_038.java  1':18"    0lhr:25':15"  Compilable
49 student009_Yahtzee_039.java  0':50"    0lhr:26':05"  Compilable
50 student009_Yahtzee_040.java  2':16"    0lhr:28':21"  Compilable
51 student009_Yahtzee_041.java  0':27"    0lhr:28':48"  Compilable
52 student009_Yahtzee_042.java  0':22"    0lhr:29':10"  Compilable
53 student009_Yahtzee_043.java  0':27"    0lhr:29':37"  Compilable
54 student009_Yahtzee_044.java  2':54"    0lhr:32':31"  Compilable
55 student009_Yahtzee_045.java  0':33"    0lhr:33':04"  Compilable
56 student009_Yahtzee_046.java  0':26"    0lhr:33':30"  Compilable
57 student009_Yahtzee_047.java  b r e a k
58 student009_Yahtzee_048.java  1':10"    0lhr:34':40"  Compilable
59 student009_Yahtzee_049.java  b r e a k
60 student009_Yahtzee_050.java  4':26"    0lhr:39':06"  Compilable
61 student009_Yahtzee_051.java  2':10"    0lhr:41':16"  Compilable
62 student009_Yahtzee_052.java  1':09"    0lhr:42':25"  Compilable
63 student009_Yahtzee_053.java  n i k s . a t . k l "  Compilable

```

A modal dialog box titled 'Instrumentation Guidelines' is open in the center of the editor window. It contains the following text:

our preferred representation.
For example, we can use the 'ProgramState' class (see previous step) to store the state of the board before and after the move.

The UI class provides methods to move a piece. The UI knows how to display the move on screen.
For example, 'move(int row, int col, int destRow, int destCol)' can be used to move the piece from a square (row, col) to another square (destRow, destCol). The UI interprets this by drawing the pixels on screen.

An easy way to bypass the UI is to fake its behavior. We can create a class with the same name (also same package) as the UI class, add the UI methods that our program calls, but replace them with non-UI code.
Thus, our fake UI class will still have the 'move()' method, but the body of this method can simply store the new state (after the move) and draw nothing on the screen.
Thus, the code to replace the GUI class can look like is:

```

-----
package <same package as GUI_Class to replace>;

```

Figure 50: An example that illustrates how to replace GUI with non-GUI components.

The screenshot shows the Delve IDE interface. On the left, there's a tree view of submission folders under 'Load submission folders'. In the center, there's a code editor with tabs for 'Editor', 'All submissions', 'Selected submission', and 'Console'. The 'Selected submission' tab is active, displaying a Java file with code related to a game move. A modal dialog titled 'Instrumentation Guidelines' is overlaid on the code editor, containing Java code with annotations. The code is as follows:

```

package <same package as GUI_Class to replace>;
public class <GUI_Class>
{
    public void move(int row,int col,int destRow,int destCol) {
        // Make sure there is a piece at (row, col) that
        // belongs to the player who makes the move

        // Make sure this piece can move to the destination
        // square (destRow, destCol)

        // Determine if an opponent piece is captured

        // Update the program state:
        // remove the captured piece (if any), move the piece
        // to the destination and change the player's turn to
        // play next
    }
}

```

Below the code editor, there's a table showing the results of the analysis for various Java files, including execution time and compilability status.

	student009_Yahtzee_036.java	student009_Yahtzee_037.java	student009_Yahtzee_038.java	student009_Yahtzee_039.java	student009_Yahtzee_040.java	student009_Yahtzee_041.java	student009_Yahtzee_042.java	student009_Yahtzee_043.java	student009_Yahtzee_044.java	student009_Yahtzee_045.java	student009_Yahtzee_046.java	student009_Yahtzee_047.java	student009_Yahtzee_048.java	student009_Yahtzee_049.java	student009_Yahtzee_050.java	student009_Yahtzee_051.java	student009_Yahtzee_052.java	student009_Yahtzee_053.java	student009_Yahtzee_054.java
1	1':08"	0':37"	0':37"	0':50"	2':16"	0':27"	0':22"	0':27"	2':54"	0':33"	0':26"	b r e a k	1':10"	b r e a k	4':26"	2':10"	1':09"	0":42"	
2																			
3																			
4																			
5																			
6																			
7																			
8																			
9																			
10																			
11																			
12																			
13																			
14																			
15																			
16																			
17																			
18																			
19																			
20																			
21																			
22																			
23																			
24																			
25																			
26																			
27																			
28																			
29																			
30																			
31																			
32																			
33																			
34																			
35																			
36																			
37																			
38																			
39																			
40																			
41																			
42																			
43																			
44																			
45																			

Figure 51: An example that illustrates how to replace GUI with non-GUI components.

The screenshot shows the Delve tool interface with the following details:

- Left Panel:** "Load submission folders" tree view showing submissions from students 000 to 054, with "student009" selected.
- Top Bar:** "Submission Analysis" tab is active, along with "Classpath" and "Program" tabs. Buttons include "Start" (green), "Stop" (grey), "Show" (blue), "Clear" (yellow), "Run" (grey), "Abort" (grey), "Instrument" (red), and a timeout dropdown set to "0.5 sec".
- Editor Area:** Shows the content of `student009_Yahtzee_150.java`. A modal window is overlaid on the editor area, titled "Instrumentation Guidelines".
- Modal Content:**
 - Code snippet showing a method implementation:

```

1 Number of snapshots: 150
2
3 Total time: 04hr:33':02"
4 Last snapshot: student009_Yahtzee_150.java
5 Number of lines: 286
6 Number of methods: 12
7
8 Time breakdown:
9
10    Filename           Time Diff      Total Time      Status
11    stud
12    stud
13    stud
14    stud
15    stud
16    stud
17    stud
18    stud
19    stud
20    stud
21    stud
22    stud
23    stud
24    stud
25    stud
26    stud
27    stud
28    stud
29    stud
30    stud
31    stud
32    stud
33    stud
34    stud
35    stud
36    stud
37    stud
38    stud
39    stud
40    stud
41    stud
42    stud
43    stud
44    stud
45    stud
46    student009_Yahtzee_036.java  1':08"   0lhr:23':20"  Compilable
47    student009_Yahtzee_037.java  0':37"   0lhr:23':57"  Compilable
48    student009_Yahtzee_038.java  1':18"   0lhr:25':15"  Compilable
49    student009_Yahtzee_039.java  0':50"   0lhr:26':05"  Compilable
50    student009_Yahtzee_040.java  2':16"   0lhr:28':21"  Compilable
51    student009_Yahtzee_041.java  0':27"   0lhr:28':48"  Compilable
52    student009_Yahtzee_042.java  0':22"   0lhr:29':10"  Compilable
53    student009_Yahtzee_043.java  0':27"   0lhr:29':37"  Compilable
54    student009_Yahtzee_044.java  2':54"   0lhr:32':31"  Compilable
55    student009_Yahtzee_045.java  0':33"   0lhr:33':04"  Compilable
56    student009_Yahtzee_046.java  0':28"   0lhr:33':30"  Compilable
57    student009_Yahtzee_047.java  b r e a k     0lhr:34":40"  Compilable
58    student009_Yahtzee_048.java  1':10"   0lhr:34':40"  Compilable
59    student009_Yahtzee_049.java  b r e a k     0lhr:34":40"  Compilable
60    student009_Yahtzee_050.java  4':26"   0lhr:39':06"  Compilable
61    student009_Yahtzee_051.java  2':10"   0lhr:41':16"  Compilable
62    student009_Yahtzee_052.java  1':09"   0lhr:42':25"  Compilable
63    student009_Yahtzee_053.java  0':24"   0lhr:43':41"  Compilable

```

 - Text explaining the need to replace GUI methods:

Note that we need to replace every GUI method that the program may call, but we do not necessarily have to add code for all all methods. For example, when the player selects a piece the

Figure 52: An example that illustrates how to replace GUI with non-GUI components.

The screenshot shows the Delve tool interface. On the left, a tree view displays submission folders under 'Load submission folders'. A folder named 'yahtzee' is expanded, showing numerous student submissions from 'student000' to 'student054'. The main window contains an 'Editor' tab with the following code snippet:

```

1 Number of snapshots: 150
2
3 Total time: 04hr:33':02"
4 Last snapshot: student009_Yahtzee_150.java
5 Number of lines: 286
6 Number of methods: 12
7
8 Time breakdown:
9
10    Filename           Time Diff      Total Time      Status
11    stud
12    stud
13    stud
14    stud
15    stud
16    stud
17    stud
18    stud
19    stud
20    stud
21    stud
22    stud
23    stud
24    stud
25    stud
26    stud
27    stud
28    stud
29    stud
30    stud
31    stud
32    stud
33    stud
34    stud
35    stud
36    stud
37    stud
38    stud
39    stud
40    stud
41    stud
42    stud
43    stud
44    stud
45    stud
46    student009_Yahtzee_036.java   1':08"        01hr:23':20"    Compilable
47    student009_Yahtzee_037.java   0':37"        01hr:23':57"    Compilable
48    student009_Yahtzee_038.java   1':18"        01hr:25':15"    Compilable
49    student009_Yahtzee_039.java   0':50"        01hr:26':05"    Compilable
50    student009_Yahtzee_040.java   2':16"        01hr:28':21"    Compilable
51    student009_Yahtzee_041.java   0':27"        01hr:28':48"    Compilable
52    student009_Yahtzee_042.java   0':22"        01hr:29':10"    Compilable
53    student009_Yahtzee_043.java   0':27"        01hr:29':37"    Compilable
54    student009_Yahtzee_044.java   2':54"        01hr:32':31"    Compilable
55    student009_Yahtzee_045.java   0':33"        01hr:33':04"    Compilable
56    student009_Yahtzee_046.java   0':26"        01hr:33':30"    Compilable
57    student009_Yahtzee_047.java   b r e a k
58    student009_Yahtzee_048.java   1':10"        01hr:34':40"    Compilable
59    student009_Yahtzee_049.java   b r e a k
60    student009_Yahtzee_050.java   4':26"        01hr:39':06"    Compilable
61    student009_Yahtzee_051.java   2':10"        01hr:41':16"    Compilable
62    student009_Yahtzee_052.java   1':09"        01hr:42':25"    Compilable
63    student009_Yahtzee_053.java   01hr:43':41"    Compilable

```

A modal dialog titled 'Instrumentation Guidelines' is open, containing the following text:

```

// The GUI highlights the squares where the selected piece can land on. Thus, we don't need to do anything and leave the code empty since it does not affect the current program state
}
<other GUI methods>
...
-----
Note that we need to replace every GUI method that the program may call, but we do not necessarily have to add code for all methods. For example, when the player selects a piece the program calls 'touchPiece(row,col)' which takes care of any visual effects (e.g., highlight the squares where this piece can move to).
However, we do not have to do anything given that we are replacing the UI and don't have to worry about visual feedback. Thus, the method 'void touchPiece(int row, int col)' will be empty.

```

Figure 53: An example that illustrates how to replace GUI with non-GUI components.

To add the code to avoid UI delays, the user can click the **Add** button

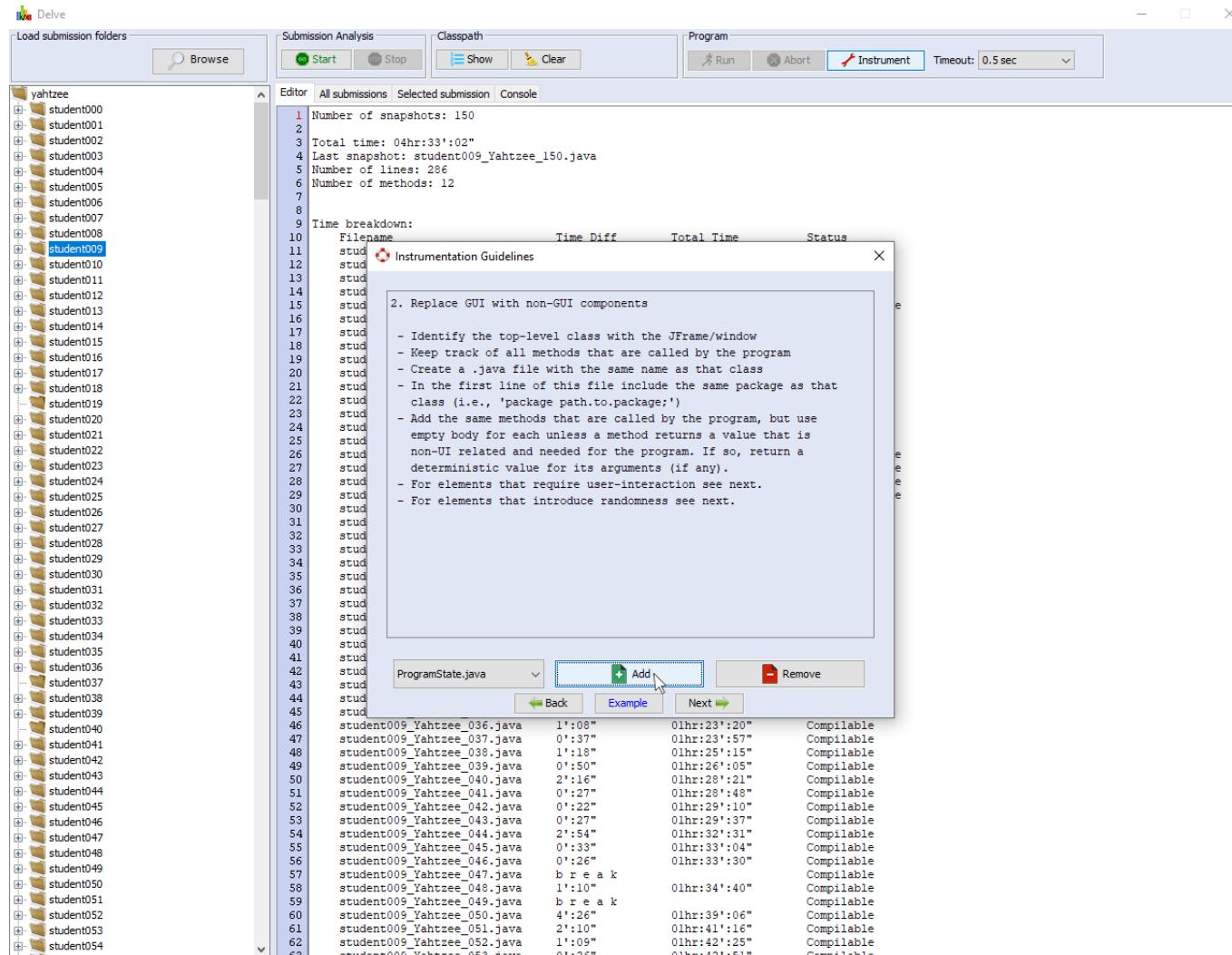


Figure 54: Click the **Add** button to locate the file with the code to avoid UI delays.

and then locate the file(s) to replace the GUI components.

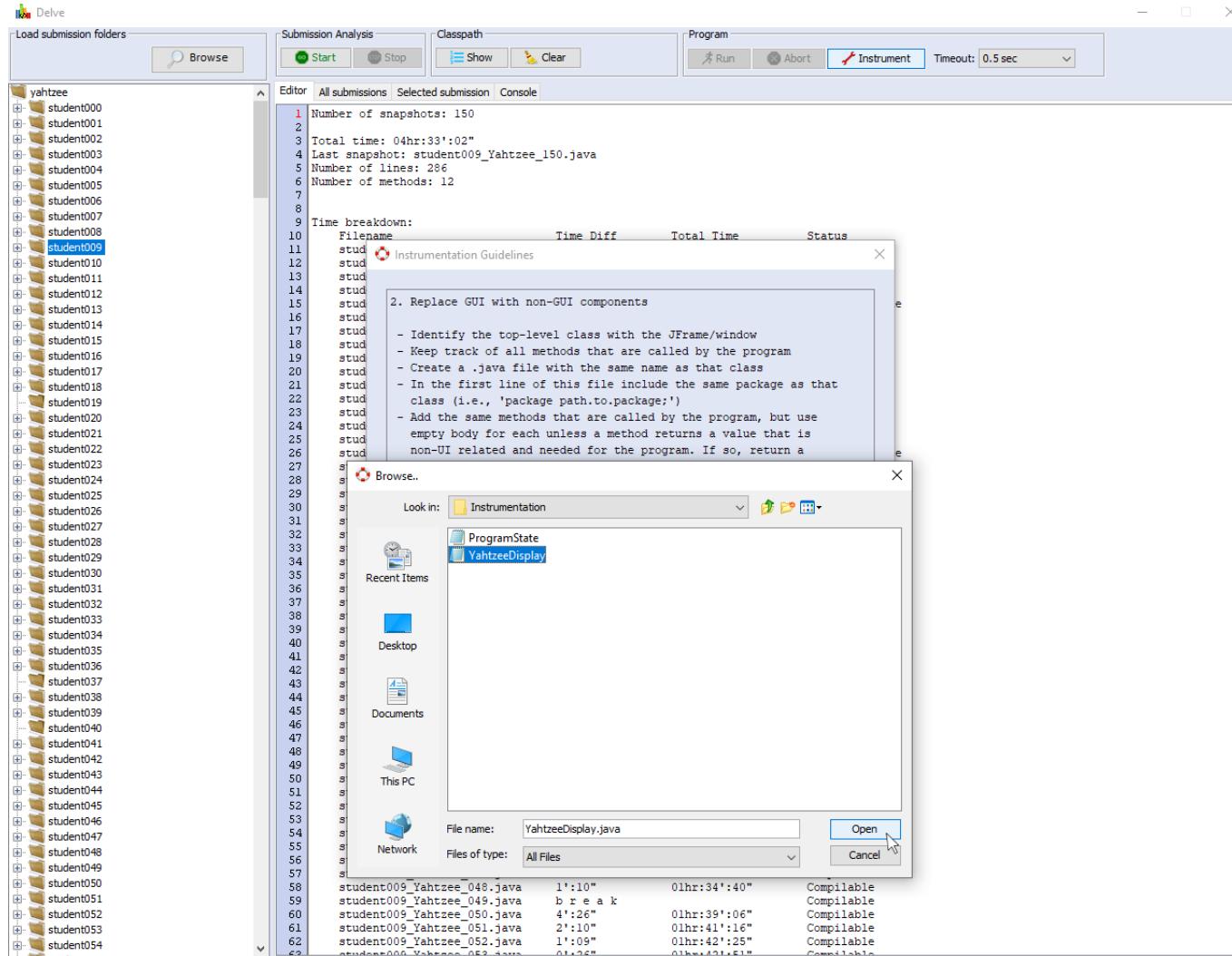


Figure 55: The user can now locate the file with the module that replaces the first UI component.

Most of the UI replacement code is actually doing nothing. We can take advantage of the fact that every program is making UI calls and use some methods to save the program state (we will revisit this in Section 7.6).

```
C:\Instrumentation\YahtzeeDisplay.java - Notepad++ [Administrator]
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
Java so| length : 2,106 lines : 54 Ln:1 Col:1 Sel:0|0 Windows (CR LF) UTF-8 INS . .
YahtzeeDisplay.java
1 import java.awt.Container;
2 import mycode.ProgramState;
3
4 public class YahtzeeDisplay implements YahtzeeConstants {
5
6     // Constants
7     private static final int NUM_OF_PLAYERS = 3;
8     private static final int FIRST_PLAYER = 1;
9     private static final int FIRST_CATEGORY = 1;
10    public static final int[] ROUNDS = new int[] {
11        ONES, TWOS, THREES, FOURS, FIVES, SIXES, THREE_OF_A_KIND, FOUR_OF_A_KIND,
12        FULL_HOUSE, SMALL_STRAIGHT, LARGE_STRAIGHT, YAHTZEE, CHANCE };
13
14     // Ivars
15    private final int[][] scores;
16    private final int[] currRound;
17    private int currPlayer;
18
19     // Ctors
20    public YahtzeeDisplay(Container container, String[] playerNames) {
21        this(playerNames);
22    }
23    public YahtzeeDisplay(String[] playerNames) {
24        currRound = new int[NUM_OF_PLAYERS + FIRST_PLAYER];
25        scores = new int[NUM_OF_PLAYERS + FIRST_PLAYER][N_CATEGORIES + FIRST_CATEGORY];
26        for (int player = FIRST_PLAYER;
27            player < NUM_OF_PLAYERS + FIRST_PLAYER; player++) {
28            currRound[player] = -1; // Not started the game yet
29            for (int category = FIRST_CATEGORY;
30                category < N_CATEGORIES + FIRST_CATEGORY; category++) {
31                scores[player][category] = 0;
32            }
33        }
34    }
35
36     // Methods
37    public void updateScorecard(int category, int player, int score) {
38        scores[player][category] = score;
39        int nextPlayer = (player < NUM_OF_PLAYERS) ? player + 1 : 1;
40        ProgramState programState = new ProgramState(scores, nextPlayer);
41        InstrumentedProgram.setProgramState(programState);
42    }
43    public void waitForPlayerToClickRoll(int player) {
44        currPlayer = player;
45        currRound[currPlayer]++;
46    }
47    public int waitForPlayerToSelectCategory() {
48        return ROUNDS[currRound[currPlayer]];
49    }
50    public boolean isDieSelected(int index) { return false; }
51    public void printMessage(String message) {}
52    public void displayDice(int[] dice) {}
53    public void waitForPlayerToSelectDice() {}
54 }
```

Figure 56: Most UI related methods can be substituted with empty or single-line body.

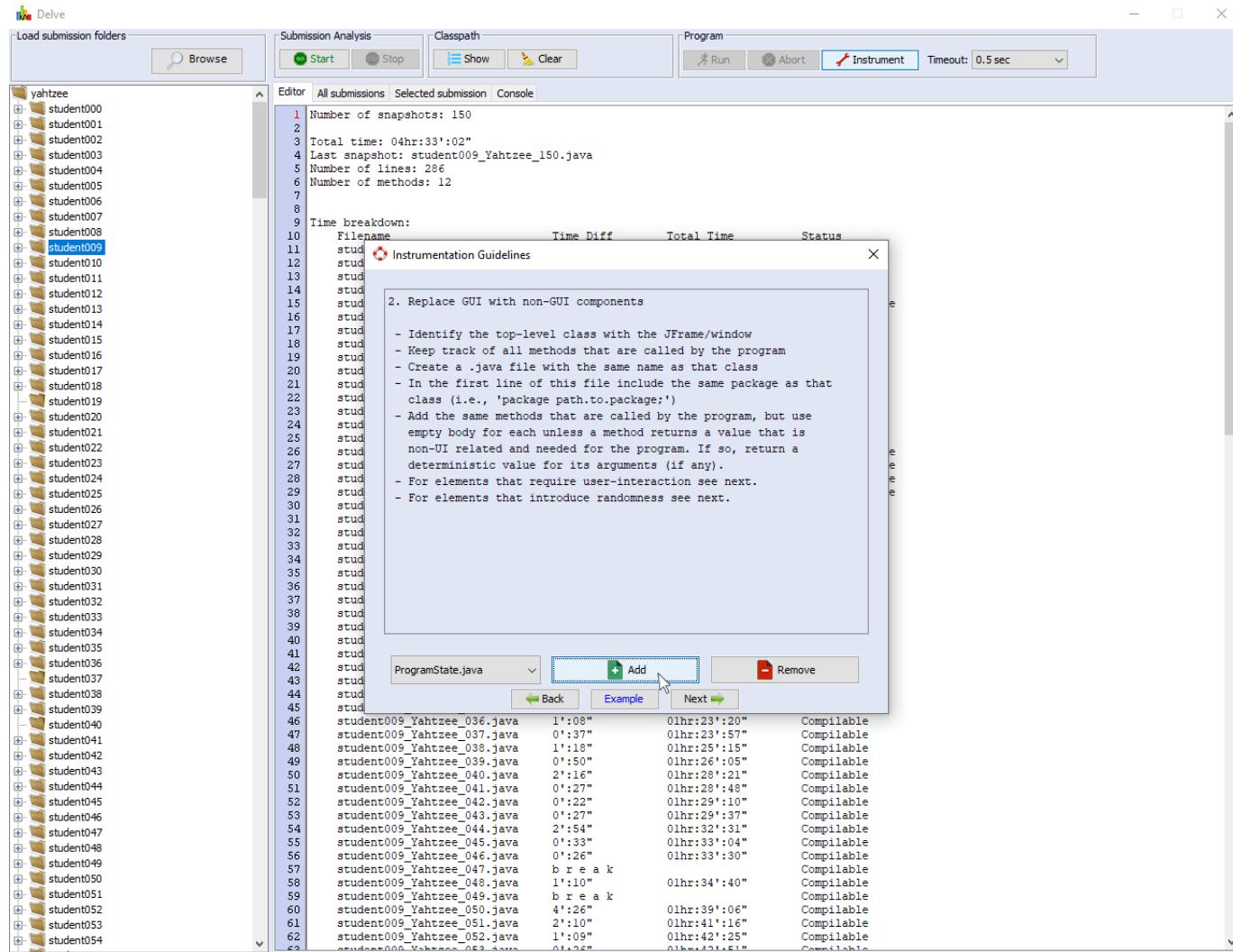


Figure 57: Click the *Add* button to locate a second file to substitute another UI module.

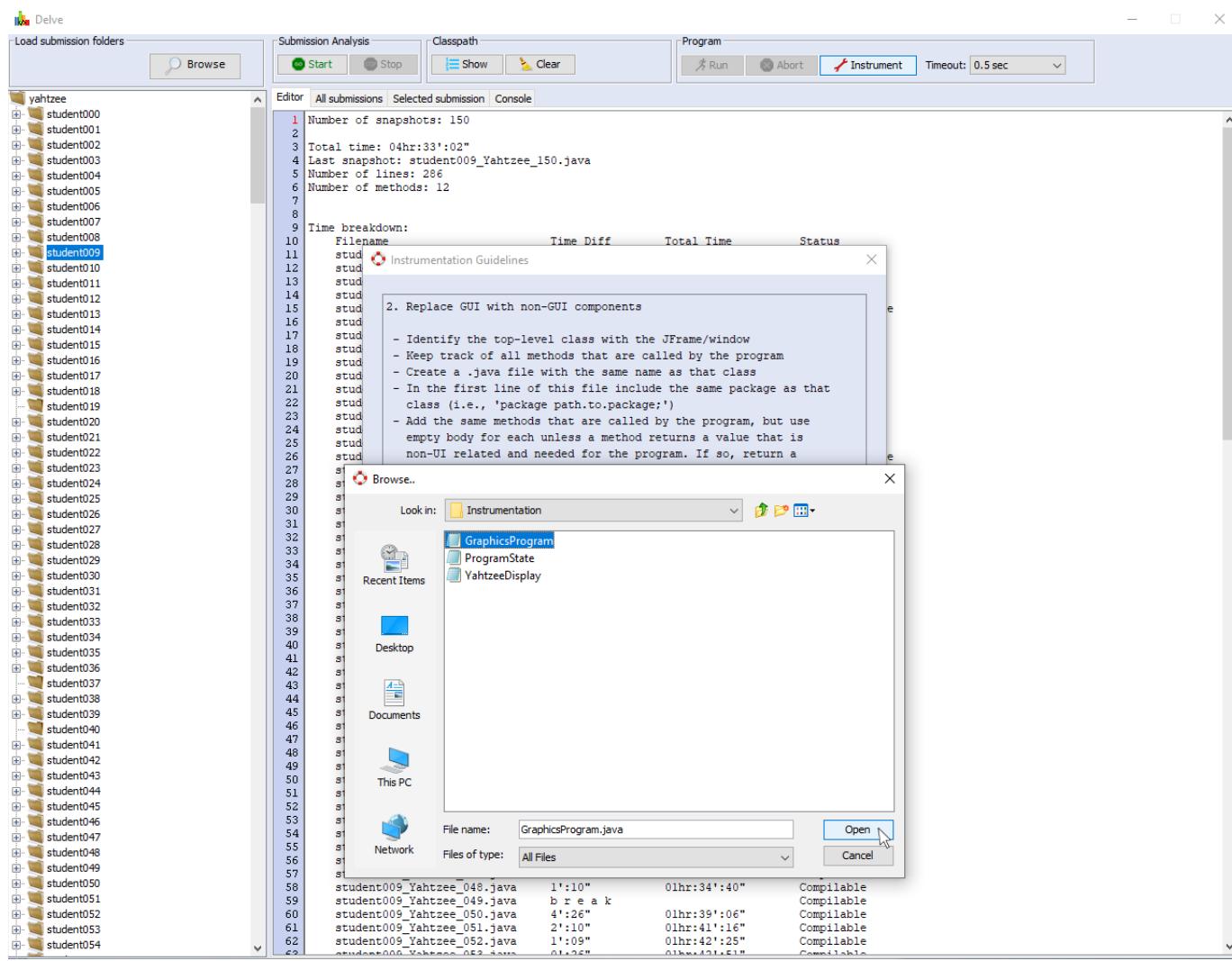
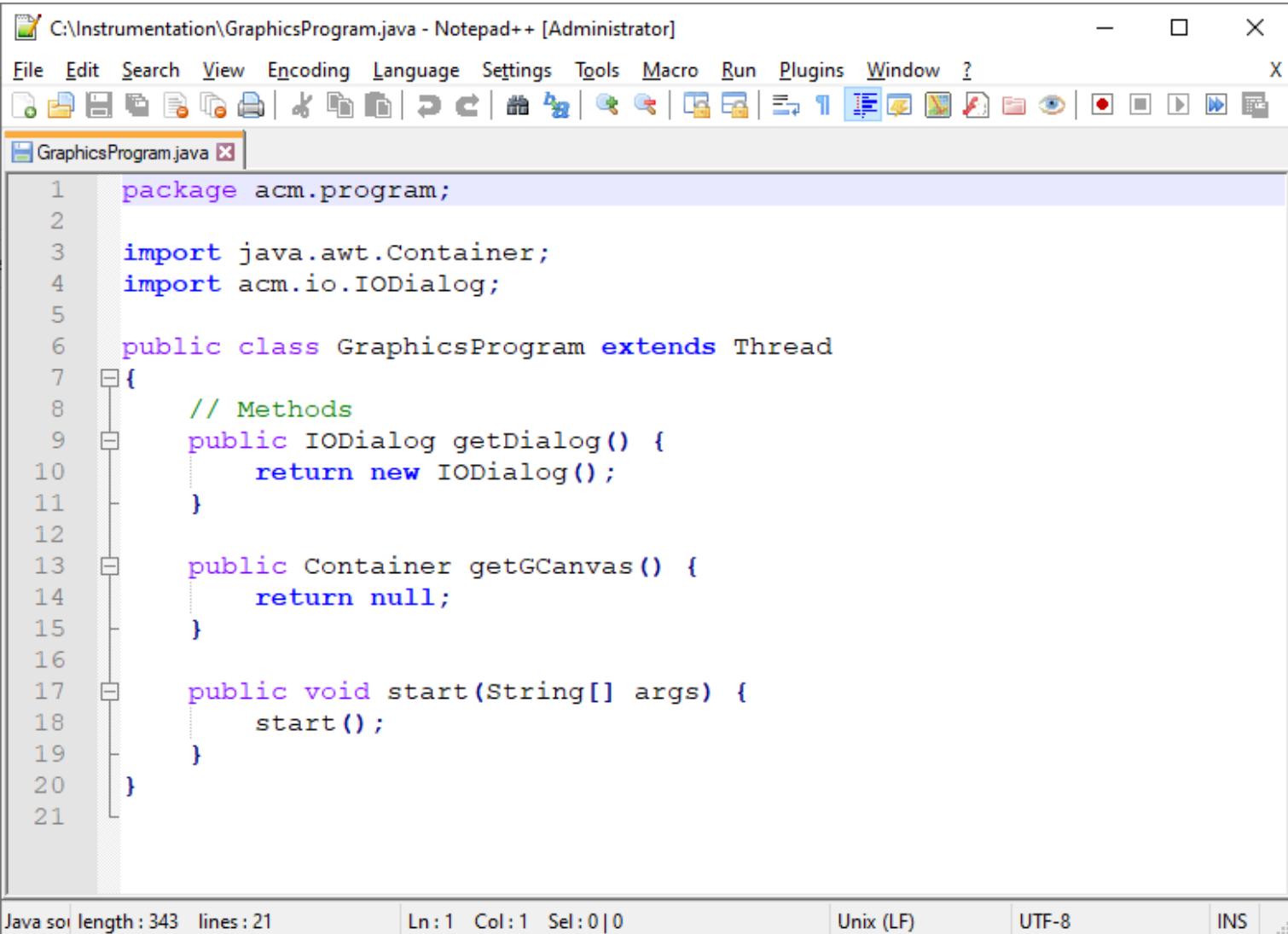


Figure 58: The user can now locate the file with the module that replaces the second UI component.

Replacing a UI module with a dummy to avoid delays is relatively easy. The goal is to bypass the UI behavior and either do nothing instead or try to capture the program state if it makes sense to do so upon a UI method call.



The screenshot shows a Notepad++ window with the file 'C:\Instrumentation\GraphicsProgram.java' open. The code defines a class 'GraphicsProgram' that extends 'Thread'. It contains three methods: 'getDialog()', 'getGCanvas()', and 'start()'. The code is as follows:

```
1 package acm.program;
2
3 import java.awt.Container;
4 import acm.io.IODialog;
5
6 public class GraphicsProgram extends Thread
7 {
8     // Methods
9     public IODialog getDialog() {
10         return new IODialog();
11     }
12
13     public Container getGCanvas() {
14         return null;
15     }
16
17     public void start(String[] args) {
18         start();
19     }
20 }
```

The status bar at the bottom of the Notepad++ window displays: 'Java source length : 343 lines : 21 Ln : 1 Col : 1 Sel : 0 | 0 Unix (LF) UTF-8 INS'.

Figure 59: The UI implementation is usually simple. The goal is to bypass the UI behavior and either do nothing or try to capture the program state.

At this point we have completed the second step and are ready to move to the next step.

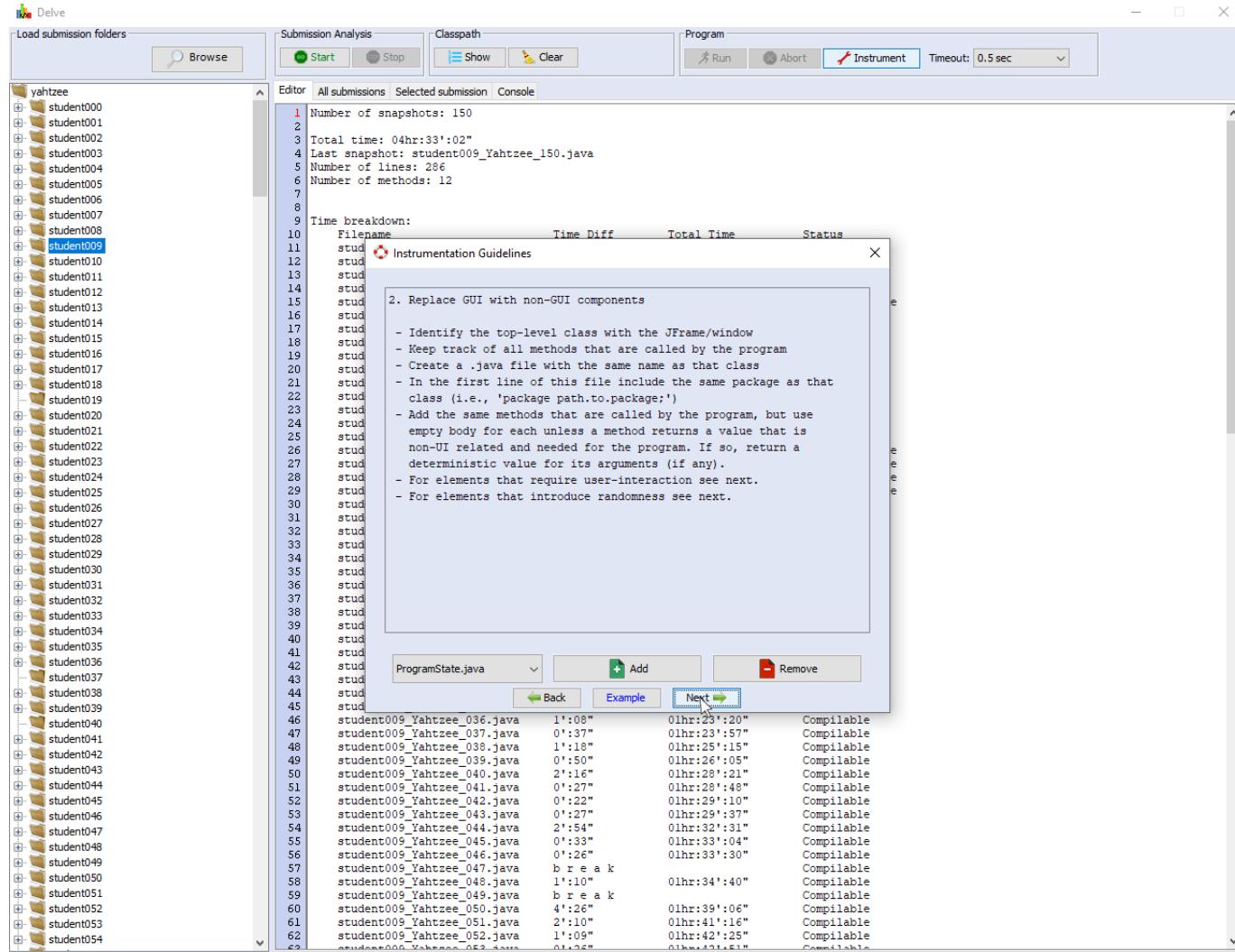


Figure 60: The second step is complete. Ready to move on to the third step.

7.4 Step 3: Get rid of user-interaction code

The third step is required if the program stalls waiting for the user to take an action (e.g., type something).

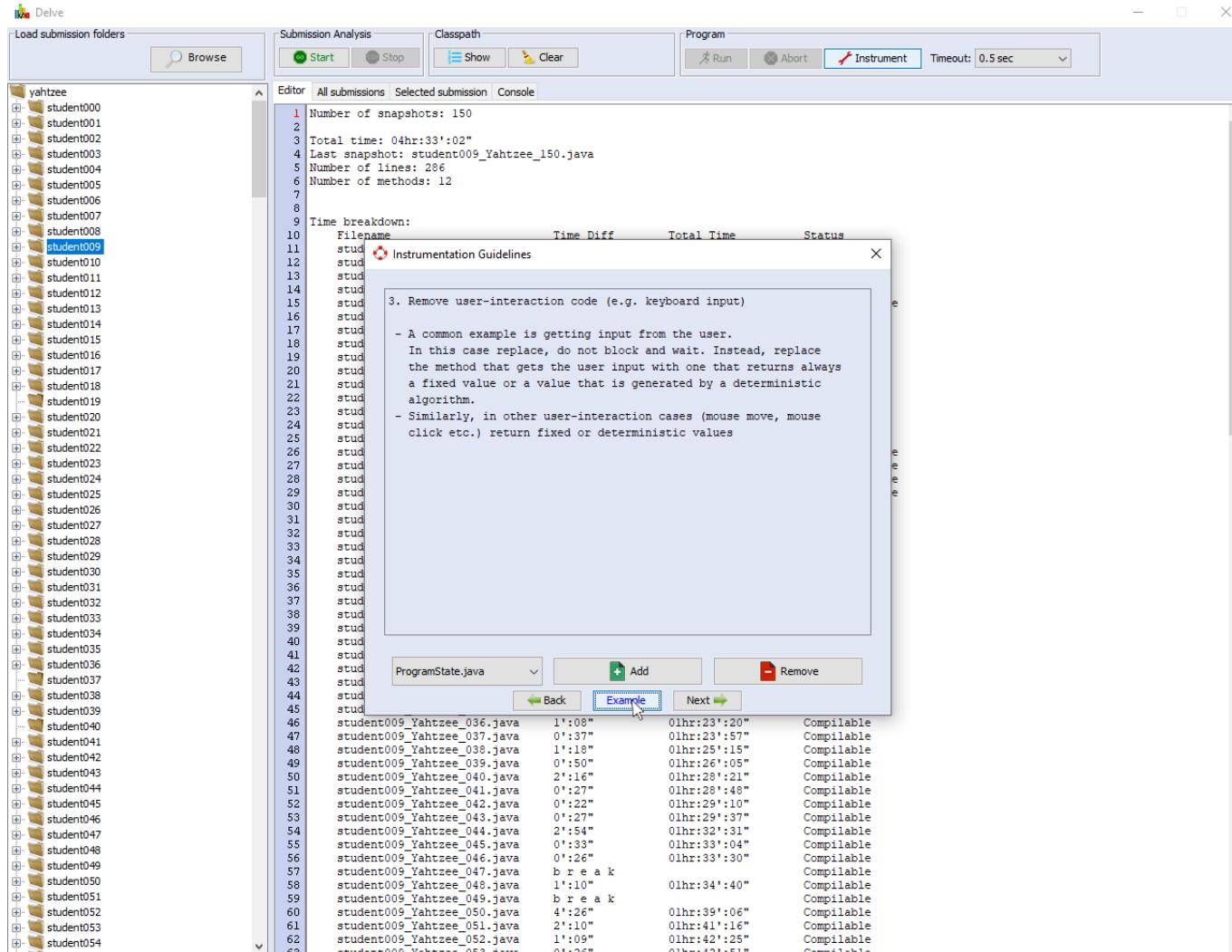


Figure 61: If there is user-interaction that stalls the execution, it should be bypassed.

Again, Delve offers an example to illustrate how this works in practice.

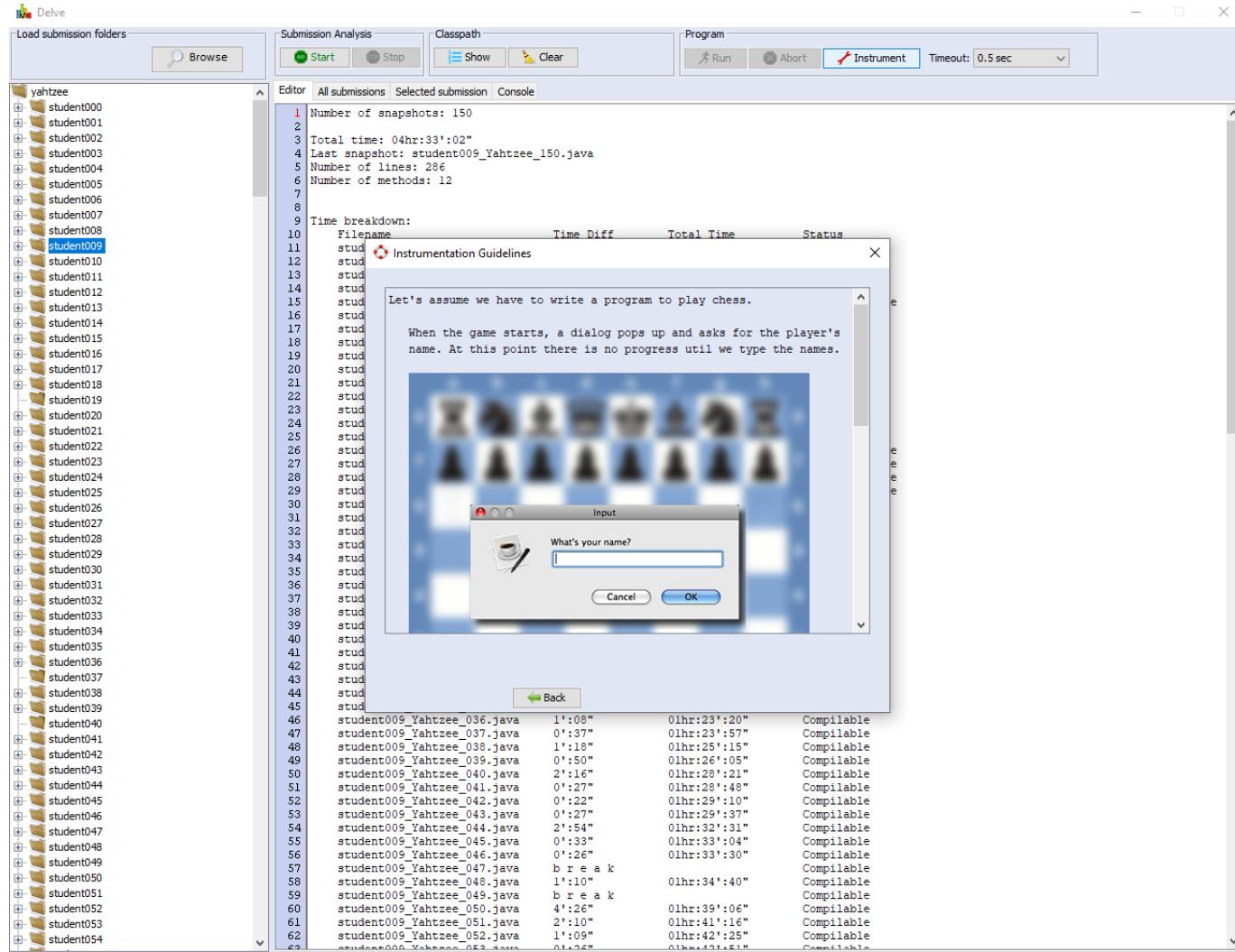


Figure 62: An example that illustrates how to bypass user-interaction that may stall the program execution.

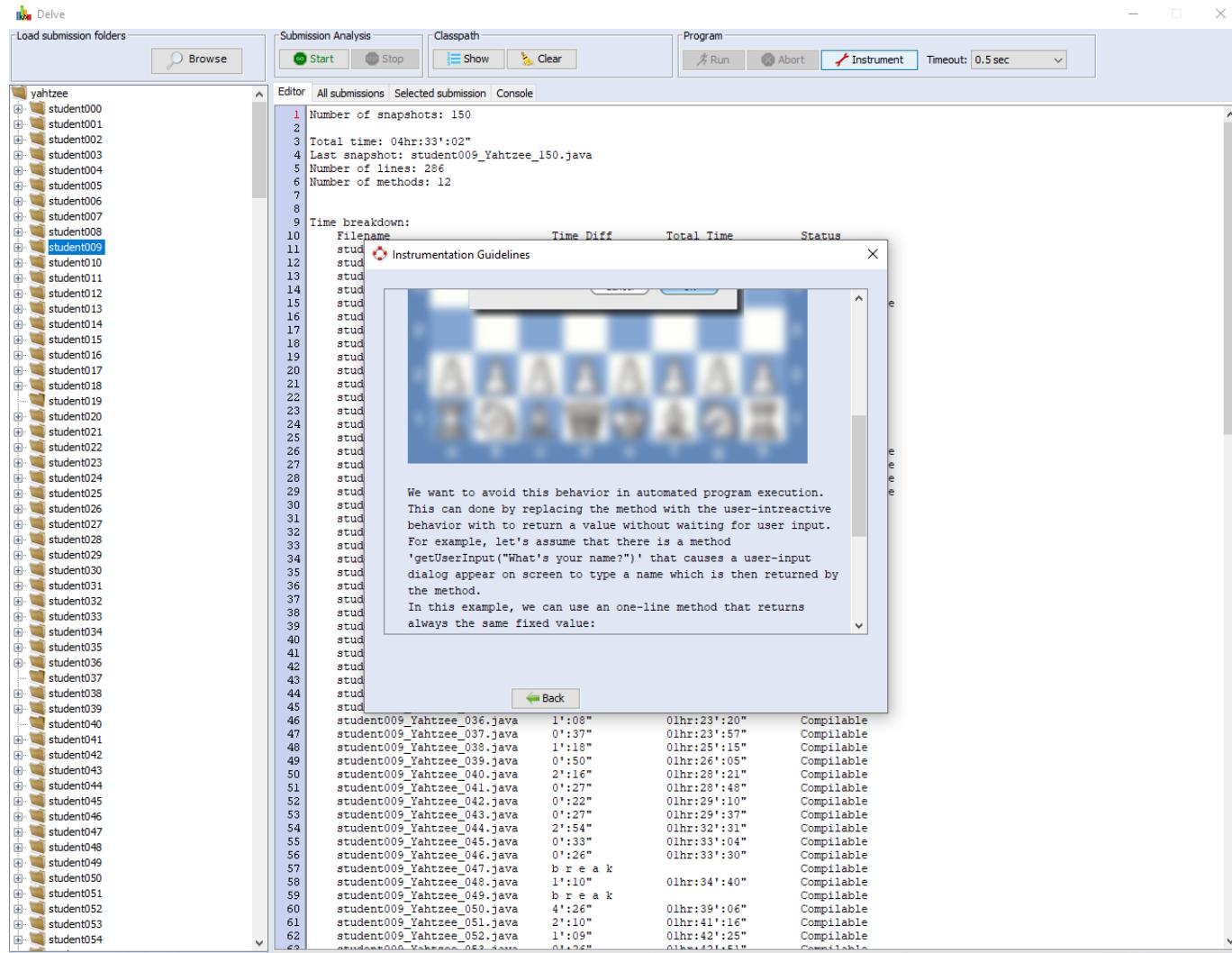


Figure 63: An example that illustrates how to bypass user-interaction that may stall the program execution.

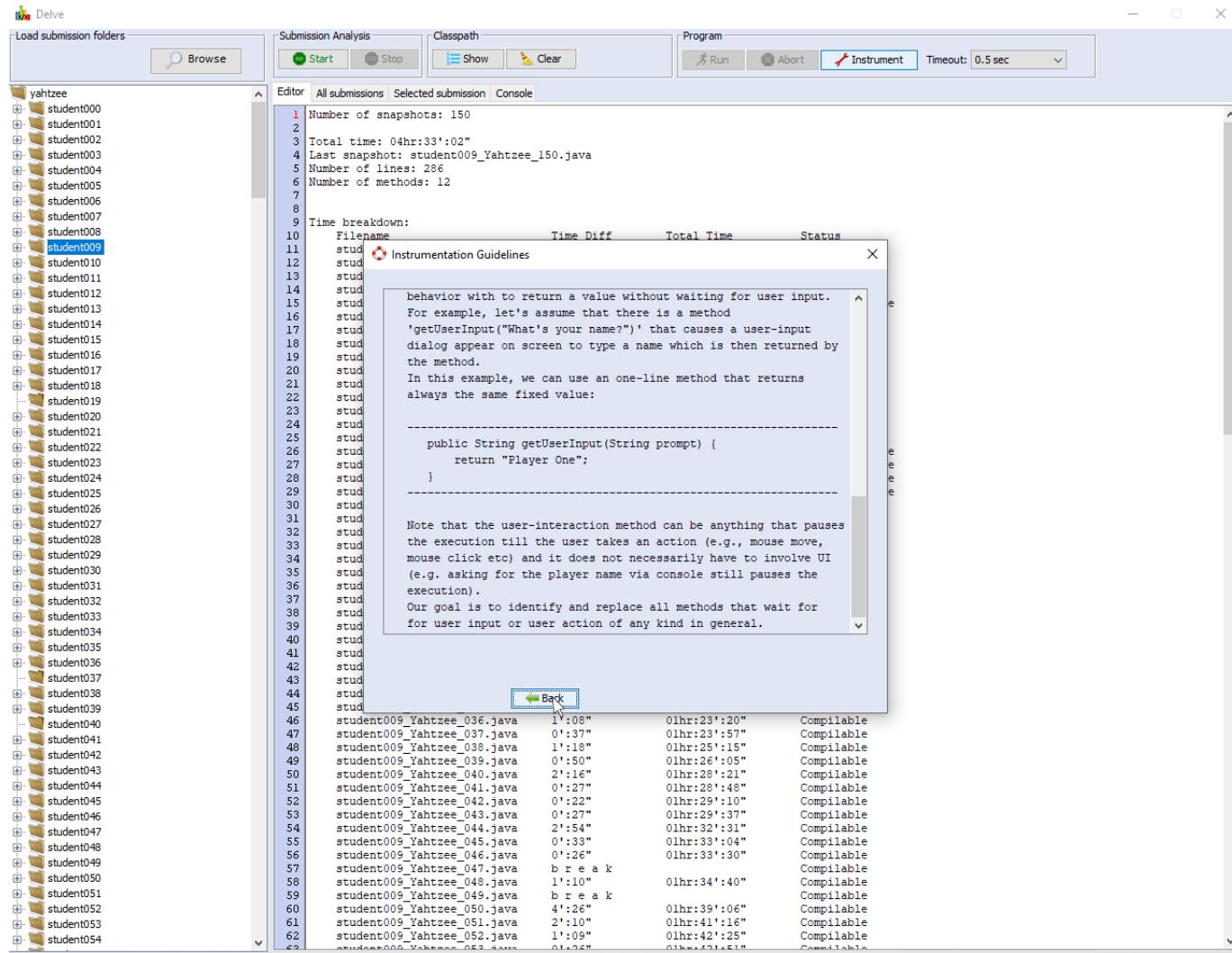


Figure 64: An example that illustrates how to bypass user-interaction that may stall the program execution.

To add the code to bypass user-interaction that stalls execution, the user can click the **Add** button

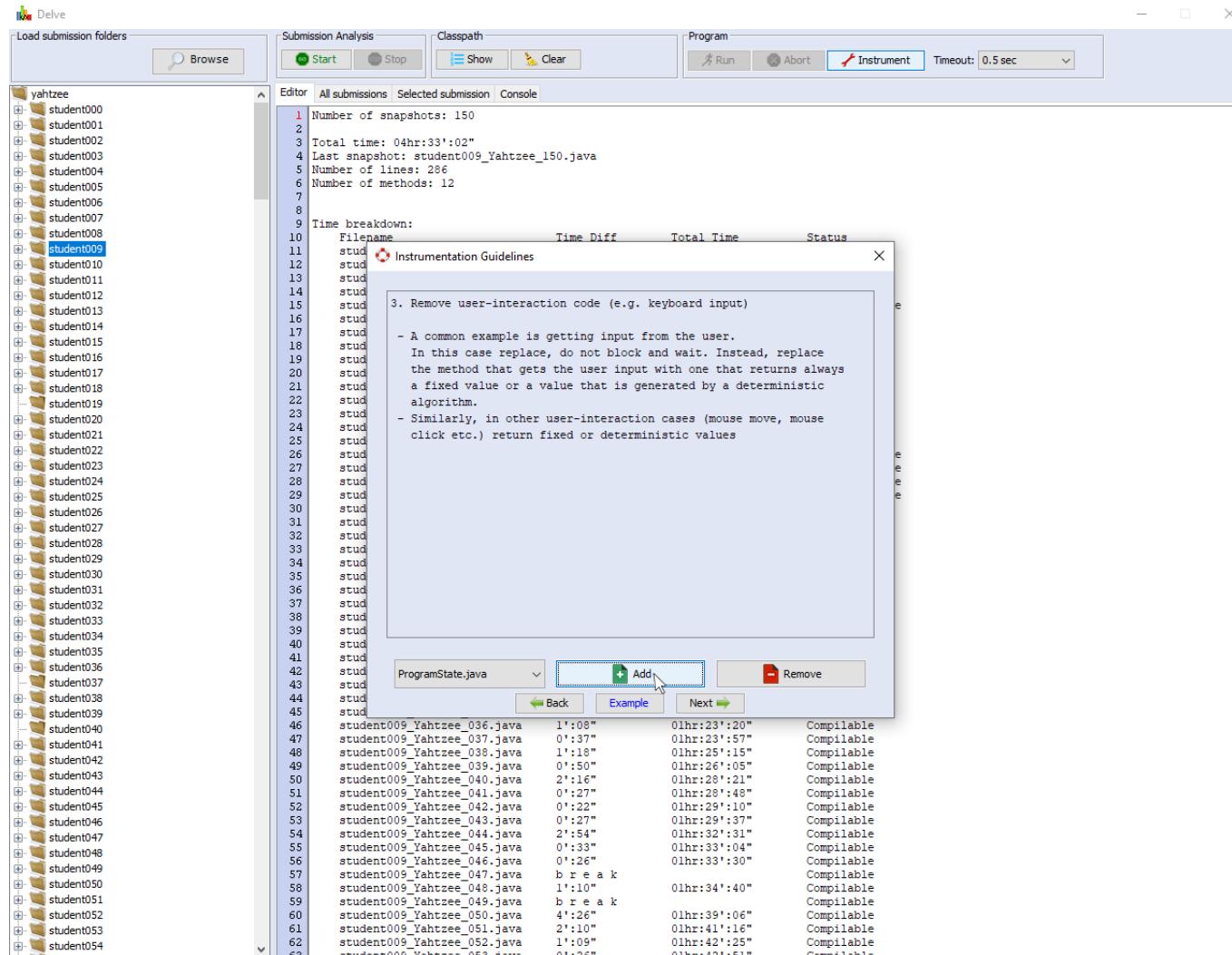


Figure 65: Click the *Add* button to locate the file with the code to bypass any user-interaction that may stall the execution.

and then locate the file(s) to bypass user-interaction.

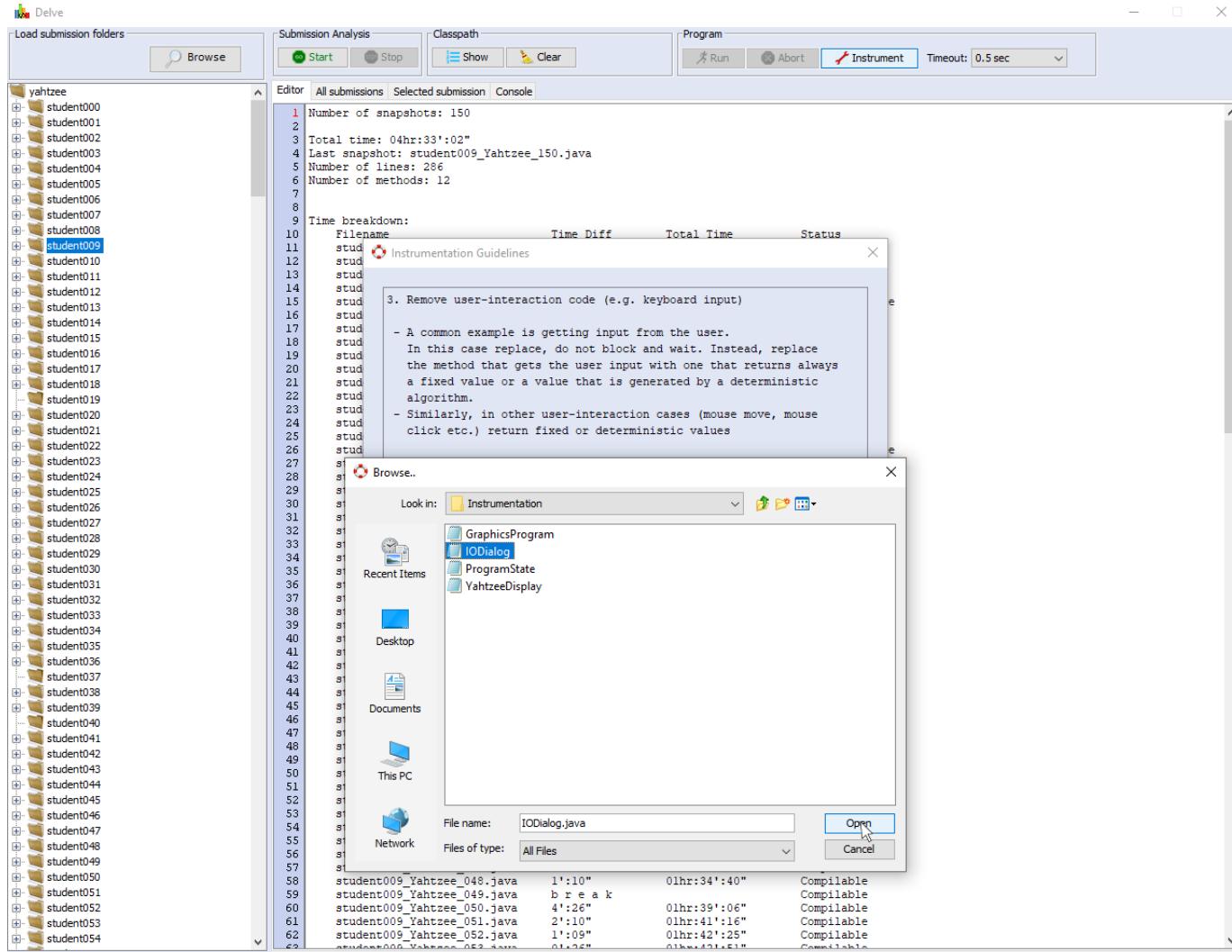
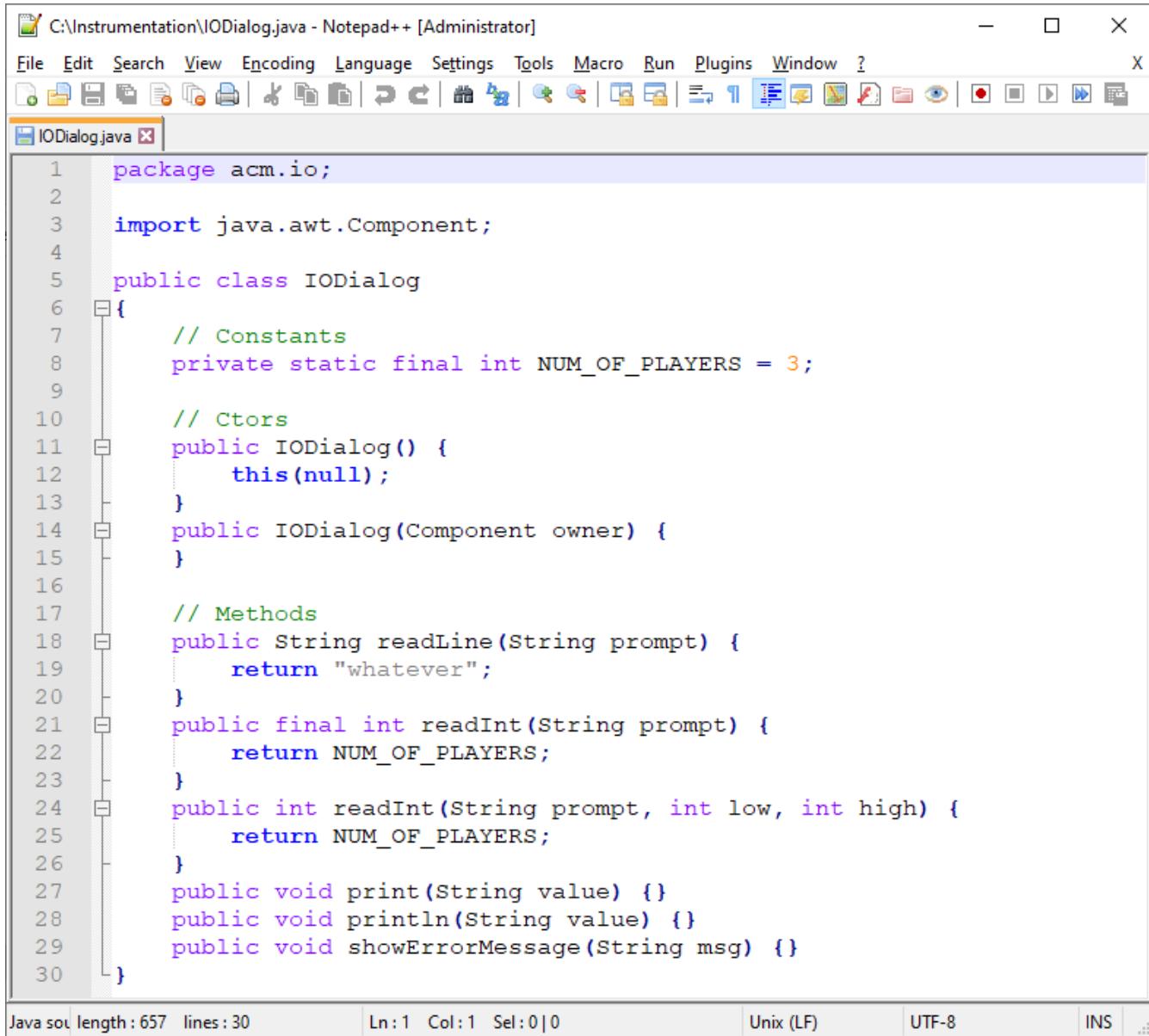


Figure 66: The user can now locate the file with the code to bypass any user-interaction.



The screenshot shows a Notepad++ window with the title bar "C:\Instrumentation\IODialog.java - Notepad++ [Administrator]". The menu bar includes File, Edit, Search, View, Encoding, Language, Settings, Tools, Macro, Run, Plugins, Window, and Help. The toolbar below the menu has various icons for file operations like Open, Save, Print, and Find. The main editor area contains Java code for the IODialog class:

```
1 package acm.io;
2
3 import java.awt.Component;
4
5 public class IODialog
6 {
7     // Constants
8     private static final int NUM_OF_PLAYERS = 3;
9
10    // Ctors
11    public IODialog() {
12        this(null);
13    }
14    public IODialog(Component owner) {
15    }
16
17    // Methods
18    public String readLine(String prompt) {
19        return "whatever";
20    }
21    public final int readInt(String prompt) {
22        return NUM_OF_PLAYERS;
23    }
24    public int readInt(String prompt, int low, int high) {
25        return NUM_OF_PLAYERS;
26    }
27    public void print(String value) {}
28    public void println(String value) {}
29    public void showErrorMessage(String msg) {}
30 }
```

The status bar at the bottom shows "Java source length: 657 lines: 30", "Ln:1 Col:1 Sel:0|0", "Unix (LF)", "UTF-8", and "INS".

Figure 67: Replacing the user-interaction methods is straightforward. Occasionally, it may require some logic to return the proper value.

At this point we have completed the third step and are ready to move to the next step.

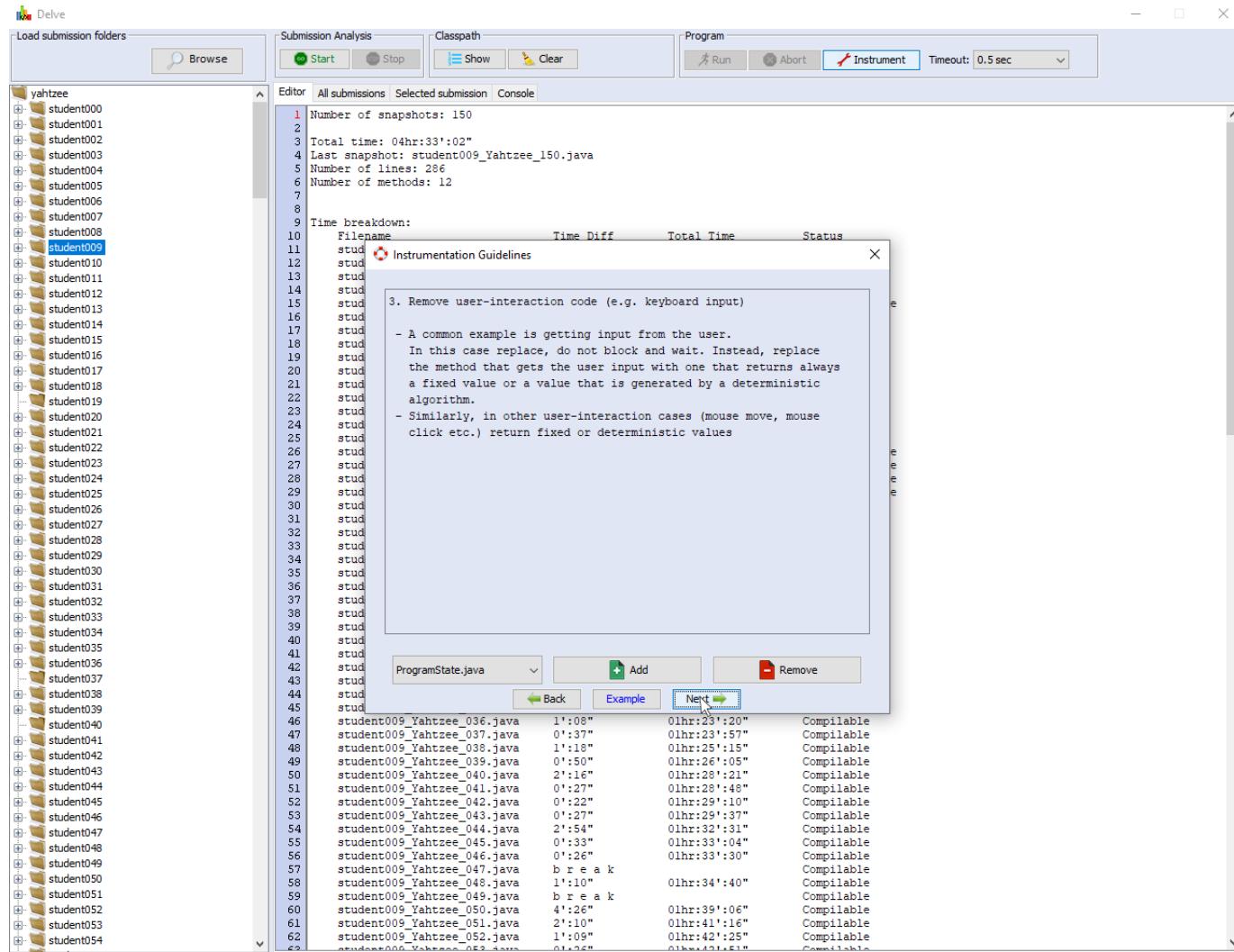


Figure 68: The third step is complete. Ready to move on to the fourth step.

7.5 Step 4: Replace random with deterministic behavior

The fourth step is required if the program has an element of randomness and does not produce always the same result.

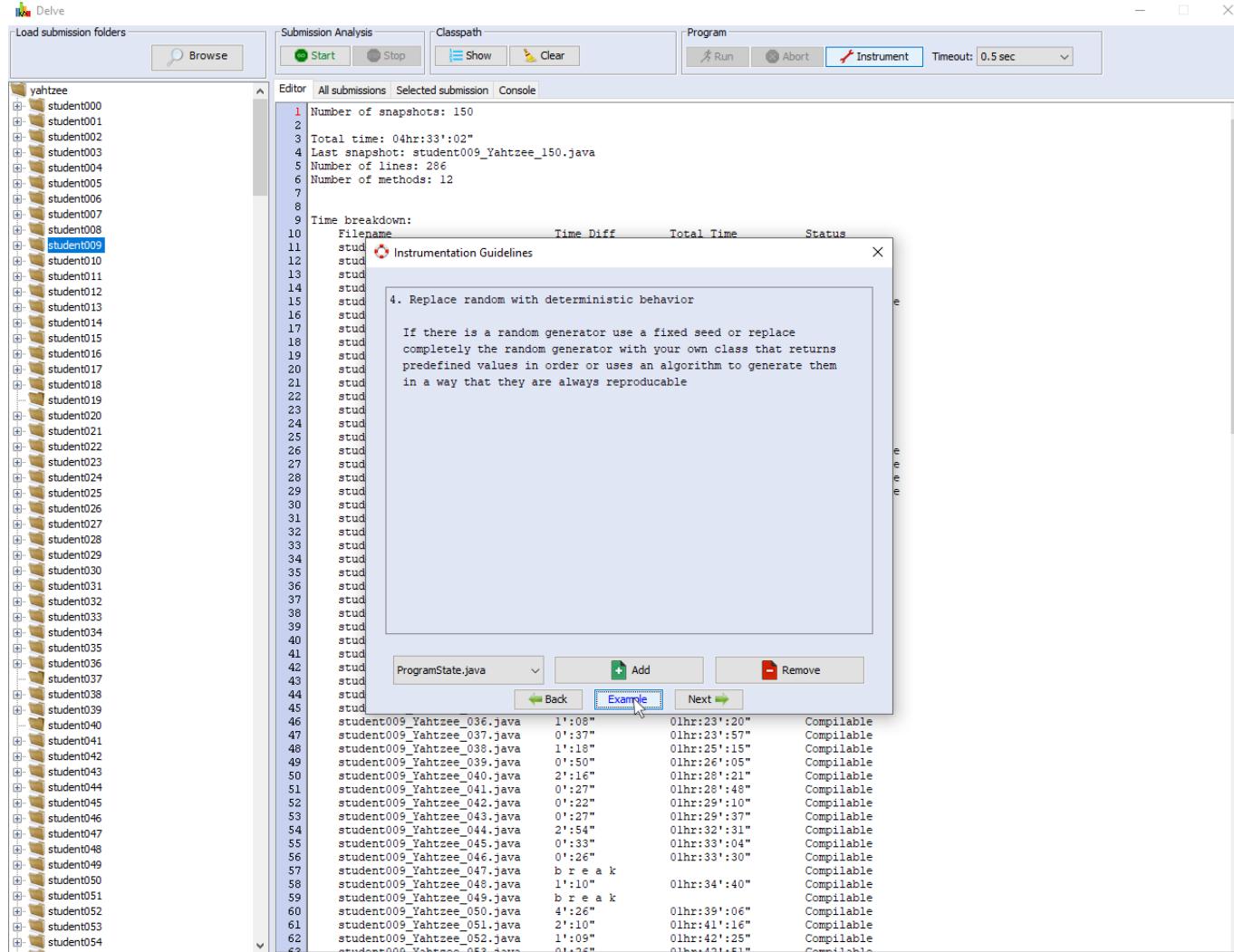


Figure 69: If there is an element of randomness, it should be replaced by deterministic behavior.

Again, Delve offers an example to illustrate how this works in practice.

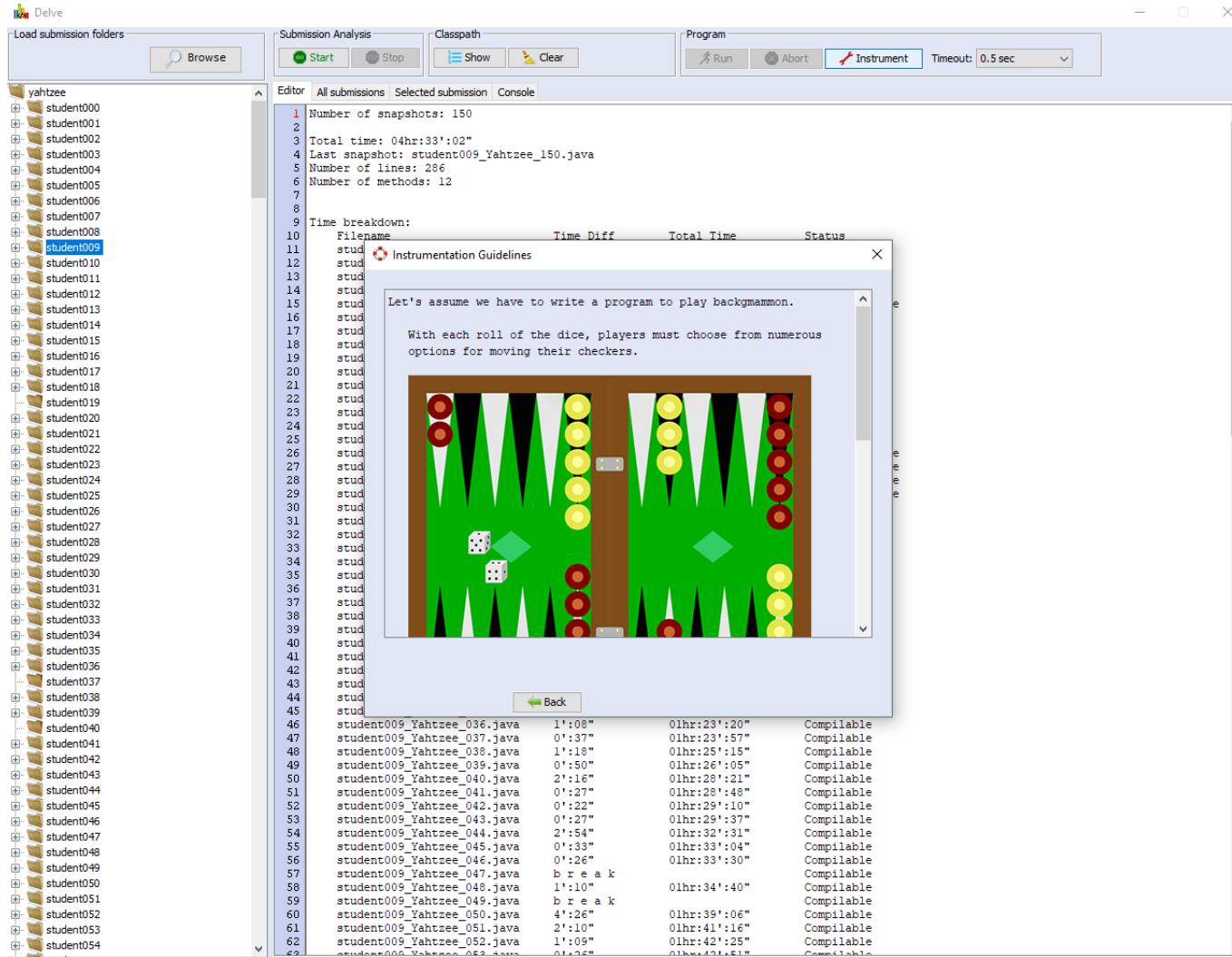


Figure 70: An example that illustrates how to remove elements of randomness from the program execution.

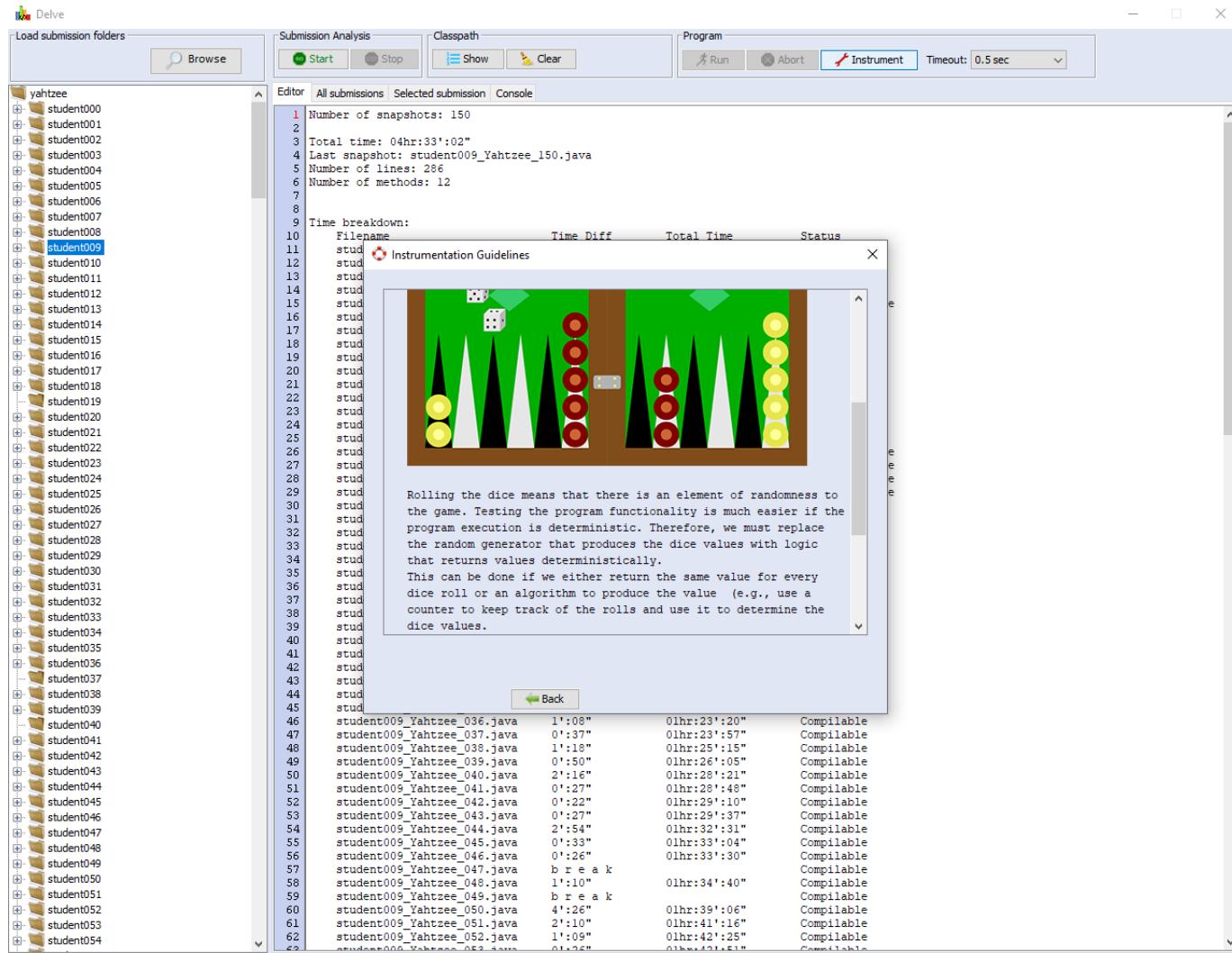


Figure 71: An example that illustrates how to remove elements of randomness from the program execution.

The screenshot shows the Delve debugger interface. On the left, a tree view lists submission folders for students 000 through 054 under a 'yahtzee' root folder. The 'student009' folder is selected. The main window contains a terminal-like interface with the following text:

```

1 Number of snapshots: 150
2
3 Total time: 04hr:33':02"
4 Last snapshot: student009_Yahtzee_150.java
5 Number of lines: 286
6 Number of methods: 12
7
8
9 Time breakdown:
10   Filename           Time Diff      Total Time      Status
11   stud                0.000ms      0.000ms      Compilable
12   stud                0.000ms      0.000ms      Compilable
13   stud                0.000ms      0.000ms      Compilable
14   stud                0.000ms      0.000ms      Compilable
15   stud                0.000ms      0.000ms      Compilable
16   stud                0.000ms      0.000ms      Compilable
17   stud                0.000ms      0.000ms      Compilable
18   stud                0.000ms      0.000ms      Compilable
19   stud                0.000ms      0.000ms      Compilable
20   stud                0.000ms      0.000ms      Compilable
21   stud                0.000ms      0.000ms      Compilable
22   stud                0.000ms      0.000ms      Compilable
23   stud                0.000ms      0.000ms      Compilable
24   stud                0.000ms      0.000ms      Compilable
25   stud                0.000ms      0.000ms      Compilable
26   stud                0.000ms      0.000ms      Compilable
27   stud                0.000ms      0.000ms      Compilable
28   stud                0.000ms      0.000ms      Compilable
29   stud                0.000ms      0.000ms      Compilable
30   stud                0.000ms      0.000ms      Compilable
31   stud                0.000ms      0.000ms      Compilable
32   stud                0.000ms      0.000ms      Compilable
33   stud                0.000ms      0.000ms      Compilable
34   stud                0.000ms      0.000ms      Compilable
35   stud                0.000ms      0.000ms      Compilable
36   stud                0.000ms      0.000ms      Compilable
37   stud                0.000ms      0.000ms      Compilable
38   stud                0.000ms      0.000ms      Compilable
39   stud                0.000ms      0.000ms      Compilable
40   stud                0.000ms      0.000ms      Compilable
41   stud                0.000ms      0.000ms      Compilable
42   stud                0.000ms      0.000ms      Compilable
43   stud                0.000ms      0.000ms      Compilable
44   stud                0.000ms      0.000ms      Compilable
45   stud                0.000ms      0.000ms      Compilable
46 student009_Yahtzee_036.java  1':08"    0lhr:23':20"  Compilable
47 student009_Yahtzee_037.java  0':37"    0lhr:23':57"  Compilable
48 student009_Yahtzee_038.java  1':18"    0lhr:25':15"  Compilable
49 student009_Yahtzee_039.java  0':50"    0lhr:26':05"  Compilable
50 student009_Yahtzee_040.java  2':16"    0lhr:28':21"  Compilable
51 student009_Yahtzee_041.java  0':27"    0lhr:28':48"  Compilable
52 student009_Yahtzee_042.java  0':22"    0lhr:29':10"  Compilable
53 student009_Yahtzee_043.java  0':27"    0lhr:29':37"  Compilable
54 student009_Yahtzee_044.java  2':54"    0lhr:32':31"  Compilable
55 student009_Yahtzee_045.java  0':33"    0lhr:33':04"  Compilable
56 student009_Yahtzee_046.java  0':26"    0lhr:33':30"  Compilable
57 student009_Yahtzee_047.java  b r e a k      0lhr:34":00"  Compilable
58 student009_Yahtzee_048.java  1':10"    0lhr:34':40"  Compilable
59 student009_Yahtzee_049.java  b r e a k      0lhr:35":10"  Compilable
60 student009_Yahtzee_050.java  4':26"    0lhr:39':06"  Compilable
61 student009_Yahtzee_051.java  2':10"    0lhr:41':16"  Compilable
62 student009_Yahtzee_052.java  1':09"    0lhr:42':25"  Compilable
63 student009_Yahtzee_053.java  n/a      n/a      n/a      Compilable

```

A modal dialog box titled 'Instrumentation Guidelines' is open, containing the following text:

the game. Testing the program functionality is much easier if the program execution is deterministic. Therefore, we must replace the random generator that produces the dice values with logic that returns values deterministically.

This can be done if we either return the same value for every dice roll or an algorithm to produce the value (e.g., use a counter to keep track of the rolls and use it to determine the dice values).

Assuming there is a RandomGenerator class with a method 'public int generateValue(int min, int max)' that returns a random value between [min, max] we can replace it with code like:

```

-----  

  public int generateValue(int min, int max) {  

    // counter is an instance variable to keep track of the  

    // number of times we call this method  

    counter++;  

    return min + counter % max;  

}
-----
```

Figure 72: An example that illustrates how to remove elements of randomness from the program execution.

To add the code to replace random with deterministic behavior, the user can click the **Add** button

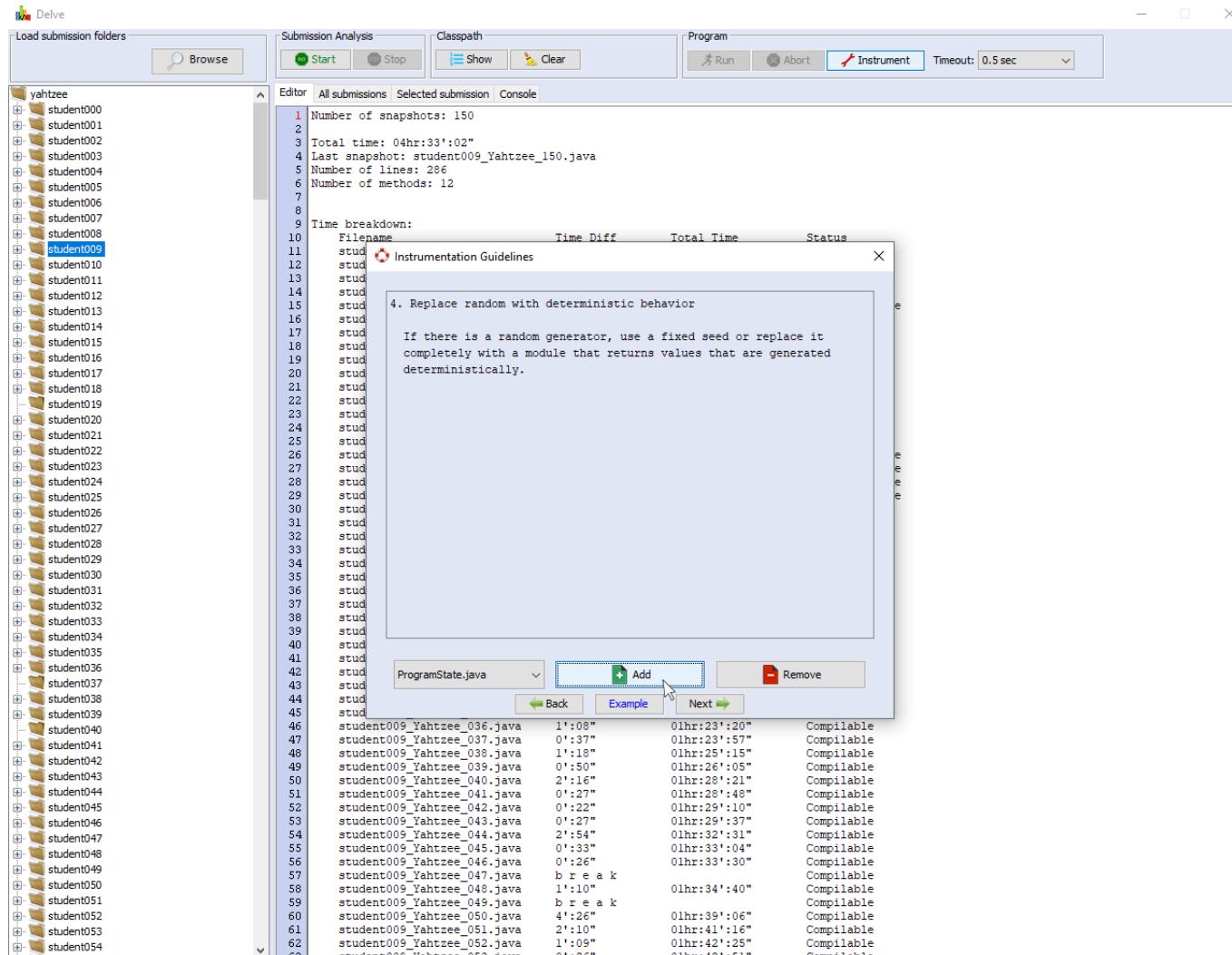


Figure 73: Click the **Add** button to locate the file with the code to replace random with deterministic behavior.

and then locate the file(s) to ensure deterministic behavior.

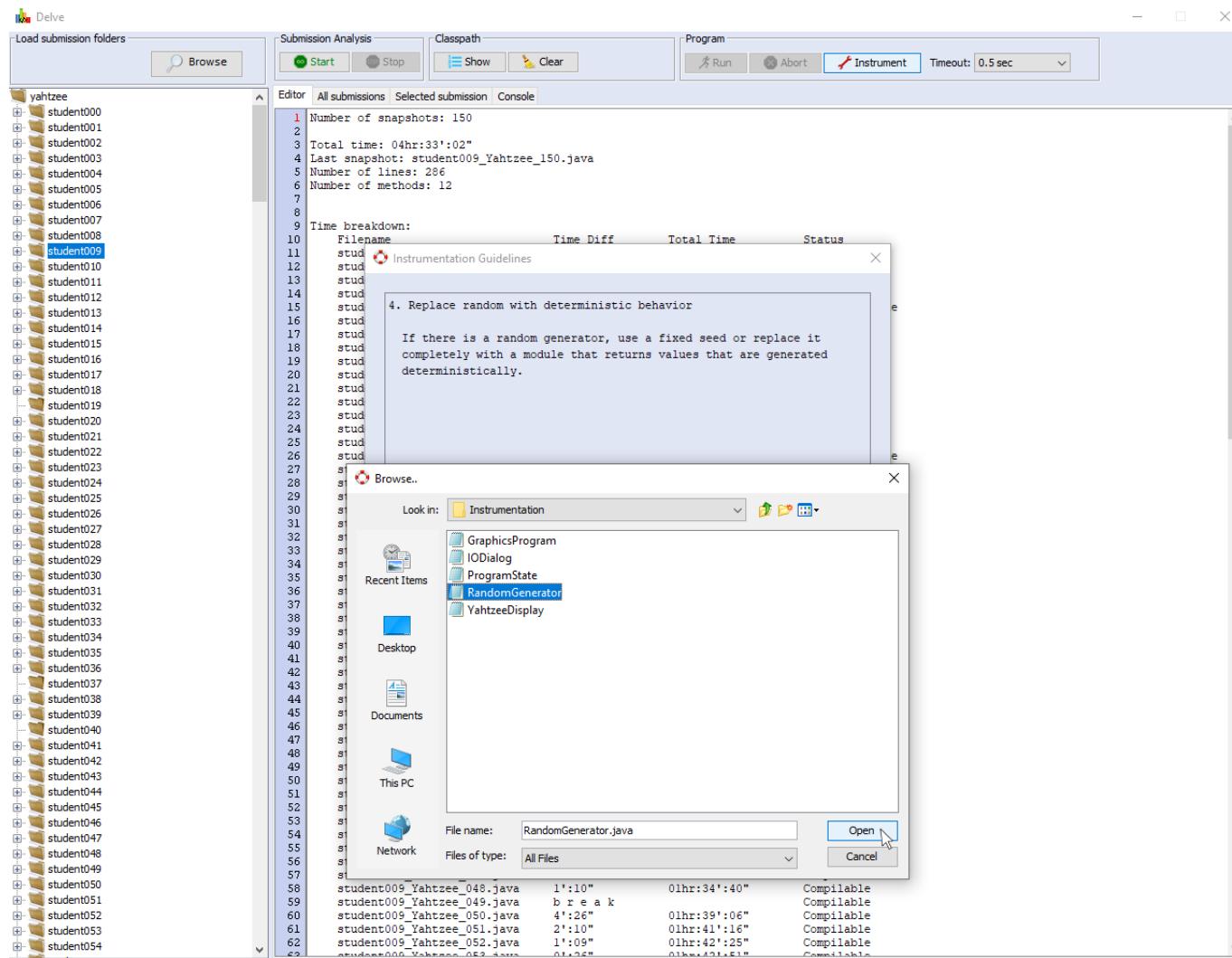


Figure 74: The user can now locate the file with the code to replace random with deterministic behavior.

```

C:\Instrumentation\RandomGenerator.java - Notepad++ [Administrator]
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
RandomGenerator.java

1 package acm.util;
2
3 public class RandomGenerator {
4
5     // Vars
6     private static RandomGenerator instance = null;
7     private int counter;
8     private int[] values;
9
10    // Ctor
11    public RandomGenerator() {
12        counter = 0;
13        values = generateValues();
14    }
15
16    // Methods
17    public static RandomGenerator getInstance() {
18        if (instance == null) { instance = new RandomGenerator(); }
19        return instance;
20    }
21
22    public int nextInt(int low, int high) {
23        return values[counter++];
24    }
25
26    private int[] generateValues() {
27        return new int[] {
28            // Expected Total Scores: Player 1: 375, Player 2: 229, Player 3: 5
29            // Round 0: ONES
30            // Player 1 score = 5 - Player 2 score = 3 - Player 3 score = 0
31            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
32            // Round 1: TWOS
33            // Player 1 score = 10 - Player 2 score = 6 - Player 3 score = 0
34            2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
35            // Round 2: THREES
36            // Player 1 score = 15 - Player 2 score = 9 - Player 3 score = 0
37            3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
38            // Round 3: FOURS
39            // Player 1 score = 20 - Player 2 score = 12 - Player 3 score = 0
40            4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
41            // Round 4: FIVES
42            // Player 1 score = 25 - Player 2 score = 15 - Player 3 score = 0
43            5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
44            // Round 5: SIXES
45            // Player 1 score = 30 - Player 2 score = 18 - Player 3 score = 0
46            6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
47            // Round 6: THREE_OF_A_KIND
48            // Player 1 score = 30 - Player 2 score = 13 - Player 3 score = 0
49            6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
50            // Round 7: FOUR_OF_A_KIND
51            // Player 1 score = 30 - Player 2 score = 0 - Player 3 score = 0
52            6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
53            // Round 8: FULL_HOUSE
54            // Player 1 score = 25 - Player 2 score = 25 - Player 3 score = 0
55            6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
56            // Round 9: SMALL_STRAIGHT
57            // Player 1 score = 30 - Player 2 score = 30 - Player 3 score = 0
58            2, 3, 4, 5, 6, 2, 3, 4, 5, 6, 2, 3, 4, 5, 6, 2, 3, 4, 5, 6,
59            // Round 10: LARGE_STRAIGHT
60            // Player 1 score = 40 - Player 2 score = 0 - Player 3 score = 0
61            2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6,
62            // Round 11: YAHTZEE
63            // Player 1 score = 50 - Player 2 score = 50 - Player 3 score = 0
64            6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
65        };
66    }
}

```

length: 3,109 lines: Ln:1 Col:1 Sel:0|0 Unix (LF) UTF-8 INS

Figure 75: In this example, we want to emulate rolling the dice. Instead of producing random values, the code keeps track of the values returned so far and uses that variable (i.e., count) as index to a fixed array of predefined values.

At this point we have completed the fourth step and are ready to move to the next step.

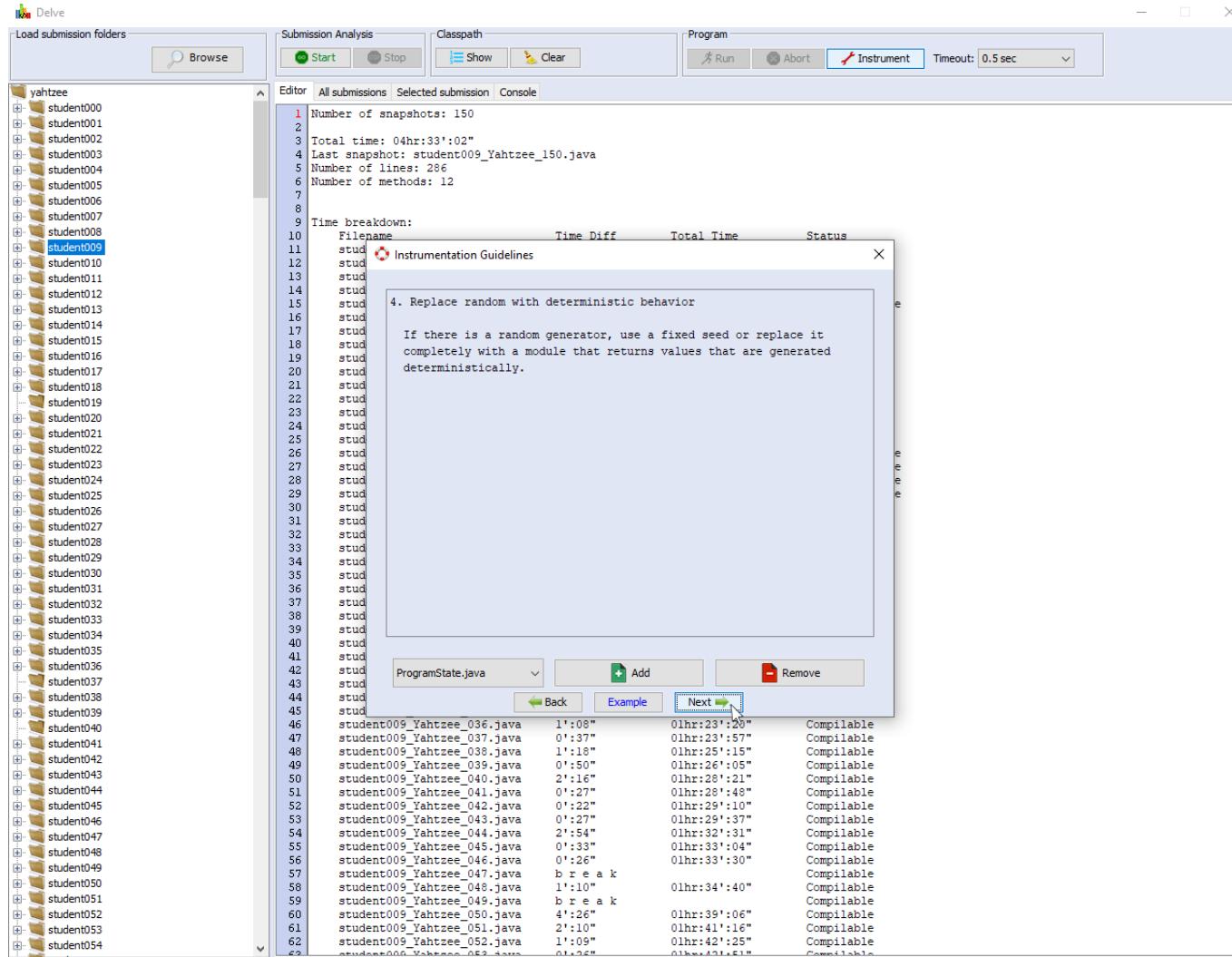


Figure 76: The fourth step is complete. Ready to move on to the fifth step.

7.6 Step 5: Save the program state

Each student program is different. If we want to instrument the code to save the state for all programs, we have to identify one or more methods that every program calls and try to capture the state at this point, if possible.

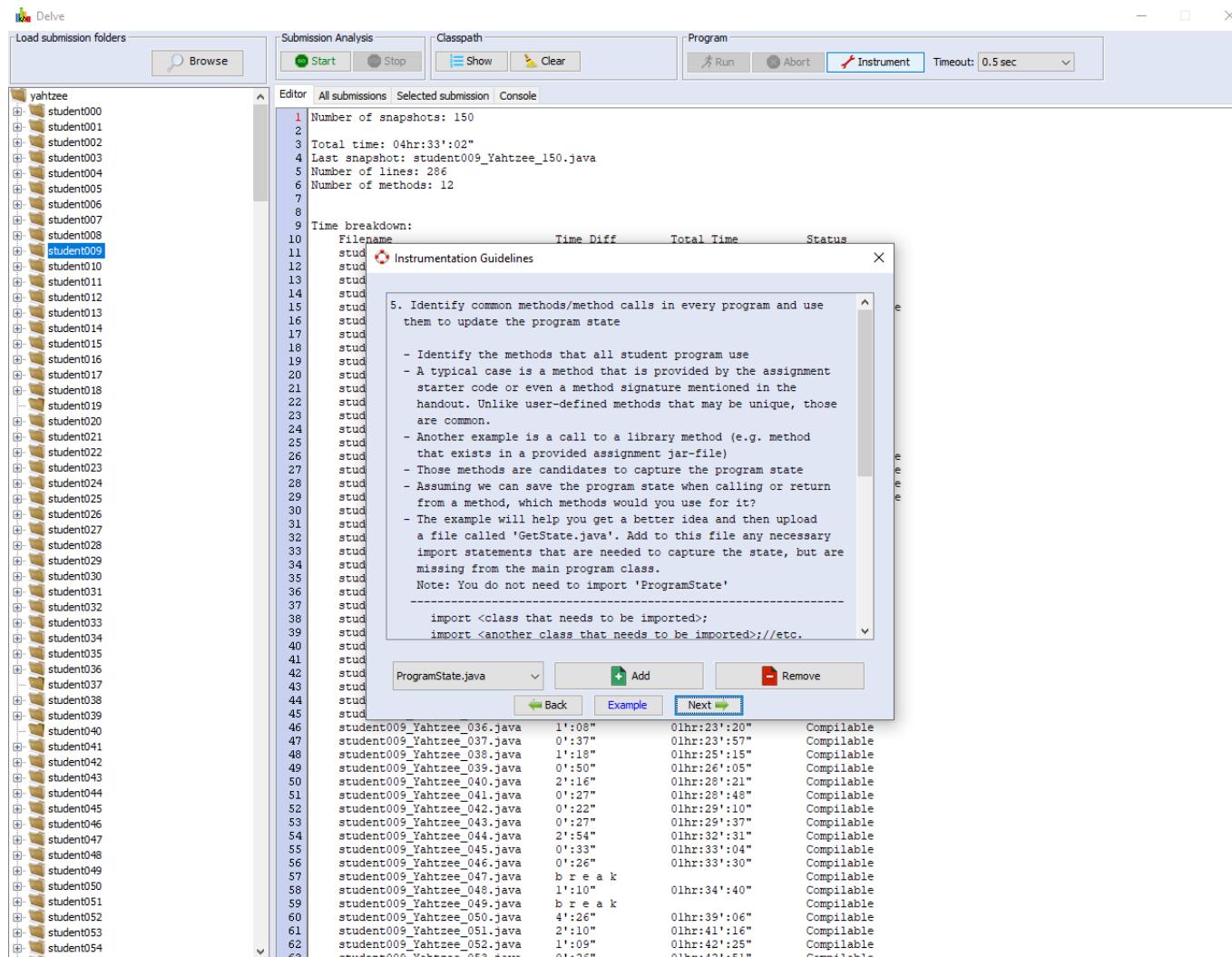


Figure 77: Delve's tips to identify common methods that can help to capture the program state.

Delve provides a few hints and lists also all the common internal program methods (i.e., methods that belong to the program that the student writes) as shown in the figure below. However, there are more places to consider. For example, each program may call a library API. Thus, candidate places to capture the state, are not just common methods in the program itself, but also external methods that all programs call.

The screenshot shows the Delve tool interface with the following details:

- Left Panel:** A tree view of submissions under the 'yahtzee' folder, listing students from student000 to student054.
- Top Bar:** Includes 'Load submission folders' (with a 'Browse' button), 'Submission Analysis' (with 'Start' and 'Stop' buttons), 'Classpath' (with 'Show' and 'Clear' buttons), 'Program' (with 'Run', 'Abort', 'Instrument' (highlighted in blue), and 'Timeout: 0.5 sec' dropdown).
- Central Area:**
 - Output Window:** Displays the following text:

```

1 Number of snapshots: 150
2
3 Total time: 04hr:33'02"
4 Last snapshot: student009_Yahtzee_150.java
5 Number of lines: 286
6 Number of methods: 12
7
8
9 Time breakdown:
10   Filename           Time Diff      Total Time      Status
11   stud                0.000ms  Instrument Guidelines
12   stud
13   stud
14   stud
15   stud
16   stud
17   stud
18   stud
19   stud
20   stud
21   stud
22   stud
23   stud
24   stud
25   stud
26   stud
27   stud
28   stud
29   stud
30   stud
31   stud
32   stud
33   stud
34   stud
35   stud
36   stud
37   stud
38   stud
39   stud
40   stud
41   stud
42   stud
43   stud
44   stud
45   stud
46 student009_Yahtzee_036.java  1':08" 0lhr:23':20" Compilable
47 student009_Yahtzee_037.java  0':37" 0lhr:23':57" Compilable
48 student009_Yahtzee_038.java  1':18" 0lhr:25':15" Compilable
49 student009_Yahtzee_039.java  0':50" 0lhr:26':05" Compilable
50 student009_Yahtzee_040.java  2':16" 0lhr:28':21" Compilable
51 student009_Yahtzee_041.java  0':27" 0lhr:28':48" Compilable
52 student009_Yahtzee_042.java  0':22" 0lhr:29':10" Compilable
53 student009_Yahtzee_043.java  0':27" 0lhr:29':37" Compilable
54 student009_Yahtzee_044.java  2':54" 0lhr:32':31" Compilable
55 student009_Yahtzee_045.java  0':33" 0lhr:33':04" Compilable
56 student009_Yahtzee_046.java  0':26" 0lhr:33':30" Compilable
57 student009_Yahtzee_047.java  b r e a k 1':10" 0lhr:34':40" Compilable
58 student009_Yahtzee_048.java  b r e a k 1':10" 0lhr:34':40" Compilable
59 student009_Yahtzee_049.java  b r e a k 1':10" 0lhr:34':40" Compilable
60 student009_Yahtzee_050.java  4':26" 0lhr:39':06" Compilable
61 student009_Yahtzee_051.java  2':10" 0lhr:41':16" Compilable
62 student009_Yahtzee_052.java  1':09" 0lhr:42':25" Compilable
63 student009_Yahtzee_053.java  0':24" 0lhr:42':51" Compilable

```
 - Code Editor:** Shows a Java code snippet for 'ProgramState.java' with annotations and a 'Common methods in student programs:' section.
 - Bottom Buttons:** Includes 'ProgramState.java' dropdown, 'Add', 'Remove', 'Back', 'Example' (highlighted in blue), and 'Next' buttons.

Figure 78: Delve lists the internal methods that are common in the student programs.

Again, Delve offers an example to illustrate how this works in practice.

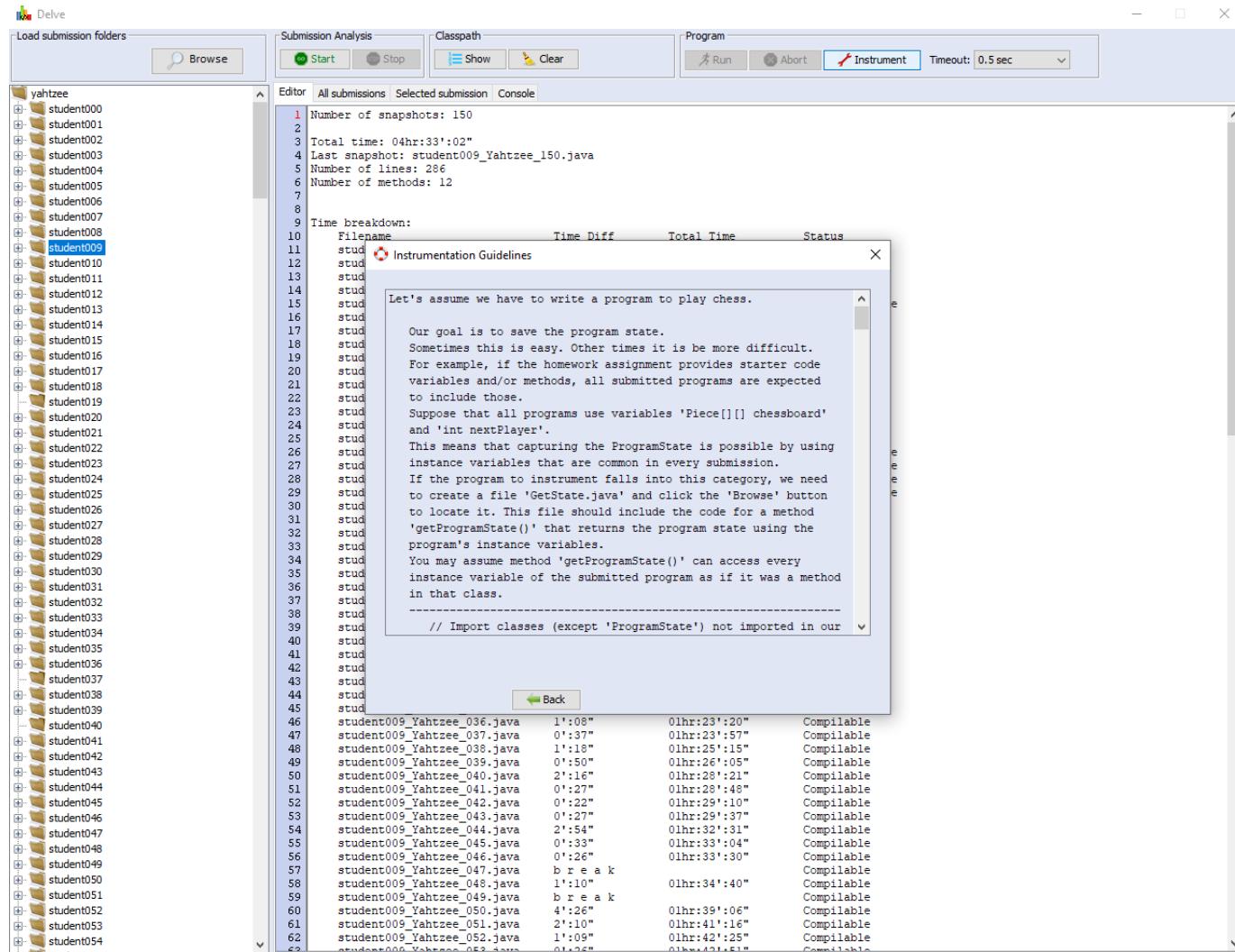


Figure 79: An example to show how to save the program state in cases when it is not so obvious.

The screenshot shows the Delve tool interface with the following details:

- Load submission folders:** A tree view showing submissions from students 000 to 054 under a folder named "yahtzee".
- Submission Analysis:** Buttons for "Start" and "Stop".
- Classpath:** Buttons for "Show" and "Clear".
- Program:** Buttons for "Run", "Abort", "Instrument" (highlighted in red), and "Timeout: 0.5 sec".
- Editor:** Shows the code for `student009_Yahtzee_150.java`. The code includes imports for `ProgramState` and a class `GetState` with a method `getProgramState()`.
- Instrumentation Guidelines:** A modal window titled "Instrumentation Guidelines" with the following content:
 - Notes about importing classes and the `getProgramState()` method.
 - A note about the absence of starter code and how students can use their own internal representation.
- Time breakdown:** A table showing execution times for various files. The table has columns: Line number, File name, Time Diff, Total Time, and Status.

Line number	File name	Time Diff	Total Time	Status
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				
15				
16				
17				
18				
19				
20				
21				
22				
23				
24				
25				
26				
27				
28				
29				
30				
31				
32				
33				
34				
35				
36				
37				
38				
39				
40				
41				
42				
43				
44				
45				
46	student009_Yahtzee_036.java	1':08"	0lhr:23':20"	Compilable
47	student009_Yahtzee_037.java	0':37"	0lhr:23':57"	Compilable
48	student009_Yahtzee_038.java	1':18"	0lhr:25':15"	Compilable
49	student009_Yahtzee_039.java	0':50"	0lhr:26':05"	Compilable
50	student009_Yahtzee_040.java	2':16"	0lhr:28':21"	Compilable
51	student009_Yahtzee_041.java	0':27"	0lhr:28':48"	Compilable
52	student009_Yahtzee_042.java	0':22"	0lhr:29':10"	Compilable
53	student009_Yahtzee_043.java	0':27"	0lhr:29':37"	Compilable
54	student009_Yahtzee_044.java	2':54"	0lhr:32':31"	Compilable
55	student009_Yahtzee_045.java	0':33"	0lhr:33':04"	Compilable
56	student009_Yahtzee_046.java	0':26"	0lhr:33':30"	Compilable
57	student009_Yahtzee_047.java	b r e a k		Compilable
58	student009_Yahtzee_048.java	1':10"	0lhr:34':40"	Compilable
59	student009_Yahtzee_049.java	b r e a k		Compilable
60	student009_Yahtzee_050.java	4':26"	0lhr:39':06"	Compilable
61	student009_Yahtzee_051.java	2':10"	0lhr:41':16"	Compilable
62	student009_Yahtzee_052.java	1':09"	0lhr:42':25"	Compilable
63	student009_Yahtzee_053.java	0lhr:43':41"		Compilable

Figure 80: An example to show how to save the program state in cases when it is not so obvious.

The screenshot shows the Delve debugger interface. On the left, a file browser displays a folder structure under 'Load submission folders' with many student submissions. The main window shows 'Submission Analysis' results for 'student009_Yahtzee_150.java'. It includes a 'Program' tab with an 'Instrument' button and a 'Time Diff' section. A central window displays a 'Time breakdown' table and a comparison of two Java code snippets. The left snippet is a simplified version of the right snippet, which contains annotations like 'stud' and 'etc.'.

Time	Time Diff	Total Time	Status	
1				
2				
3				
4				
5				
6				
7				
8				
9				
10	Time breakdown:			
11	Filename	Time Diff	Total Time	
12	stud			
13	stud			
14	stud			
15	stud			
16	stud			
17	stud			
18	stud			
19	stud			
20	stud			
21	stud			
22	stud			
23	stud			
24	stud			
25	stud			
26	stud			
27	stud			
28	stud			
29	stud			
30	stud			
31	stud			
32	stud			
33	stud			
34	stud			
35	stud			
36	stud			
37	stud			
38	stud			
39	stud			
40	stud			
41	stud			
42	stud			
43	stud			
44	stud			
45	stud			
46	student009_Yahtzee_036.java	1':08"	0lhr:23':20"	Compilable
47	student009_Yahtzee_037.java	0':37"	0lhr:23':57"	Compilable
48	student009_Yahtzee_038.java	1':18"	0lhr:25':15"	Compilable
49	student009_Yahtzee_039.java	0':50"	0lhr:26':05"	Compilable
50	student009_Yahtzee_040.java	2':16"	0lhr:28':21"	Compilable
51	student009_Yahtzee_041.java	0':27"	0lhr:28':48"	Compilable
52	student009_Yahtzee_042.java	0':22"	0lhr:29':10"	Compilable
53	student009_Yahtzee_043.java	0':27"	0lhr:29':37"	Compilable
54	student009_Yahtzee_044.java	2':54"	0lhr:32':31"	Compilable
55	student009_Yahtzee_045.java	0':33"	0lhr:33':04"	Compilable
56	student009_Yahtzee_046.java	0':26"	0lhr:33':30"	Compilable
57	student009_Yahtzee_047.java	b r e a k		Compilable
58	student009_Yahtzee_048.java	1':10"	0lhr:34':40"	Compilable
59	student009_Yahtzee_049.java	b r e a k		Compilable
60	student009_Yahtzee_050.java	4':26"	0lhr:39':06"	Compilable
61	student009_Yahtzee_051.java	2':10"	0lhr:41':16"	Compilable
62	student009_Yahtzee_052.java	1':09"	0lhr:42':25"	Compilable
63	student009_Yahtzee_053.java	0':42"	0lhr:43':41"	Compilable

Figure 81: An example to show how to save the program state in cases when it is not so obvious.

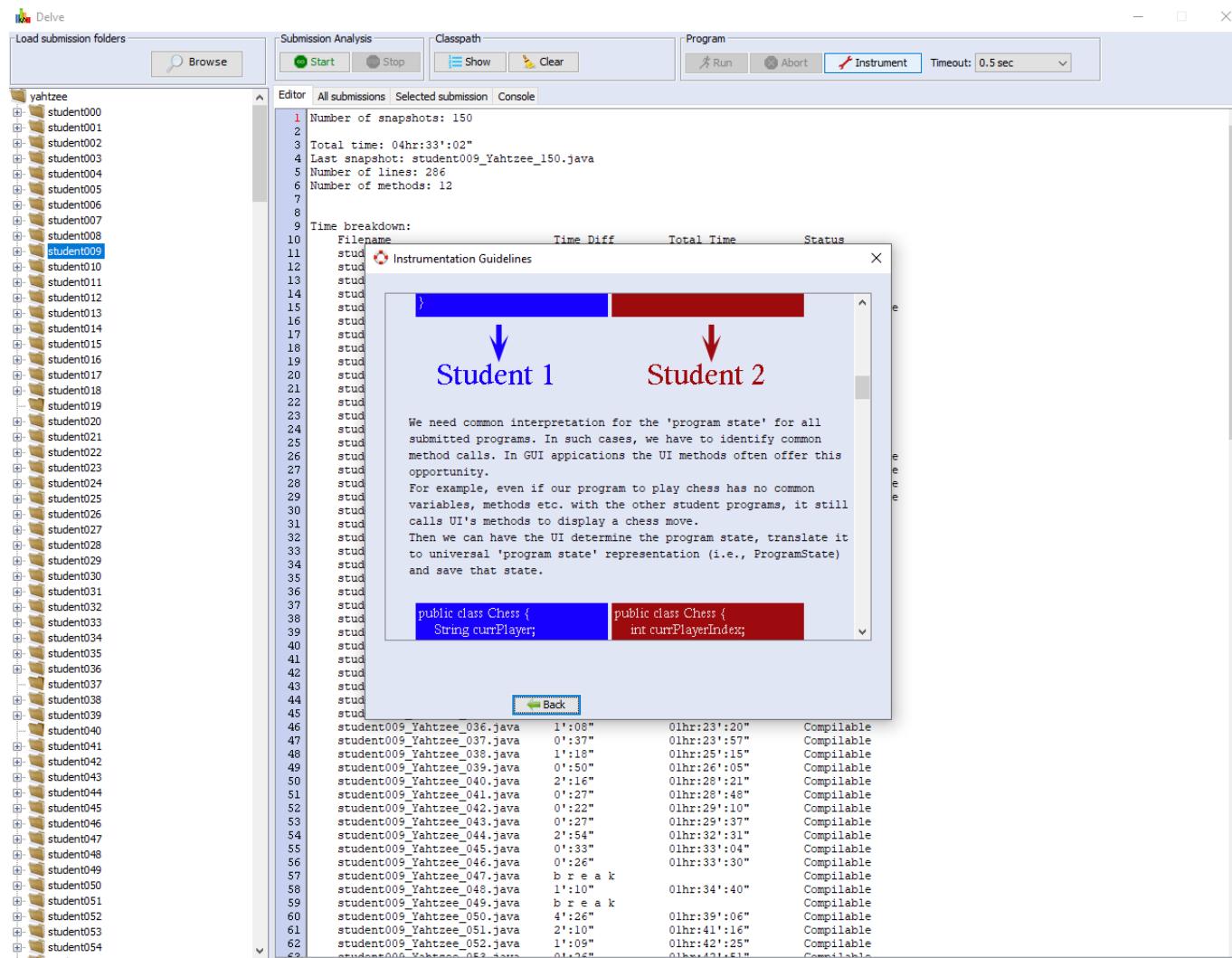


Figure 82: An example to show how to save the program state in cases when it is not so obvious.

The screenshot shows the Delve debugger interface. On the left, a tree view lists submission folders under 'Load submission folders'. The 'student009' folder is selected. The main window has tabs for 'Submission Analysis', 'Classpath', 'Program', and 'Instrument'. The 'Instrument' tab is active, showing a timeout of '0.5 sec'. The 'Program' tab displays a code editor with the following content:

```

1 Number of snapshots: 150
2
3 Total time: 04hr:33':02"
4 Last snapshot: student009_Yahtzee_150.java
5 Number of lines: 286
6 Number of methods: 12
7
8 Time breakdown:
9
10  Filename           Time Diff      Total Time      Status
11  stud               0.000ms
12  stud               0.000ms
13  stud               0.000ms
14  stud               0.000ms
15  stud               0.000ms
16  stud               0.000ms
17  stud               0.000ms
18  stud               0.000ms
19  stud               0.000ms
20  stud               0.000ms
21  stud               0.000ms
22  stud               0.000ms
23  stud               0.000ms
24  stud               0.000ms
25  stud               0.000ms
26  stud               0.000ms
27  stud               0.000ms
28  stud               0.000ms
29  stud               0.000ms
30  stud               0.000ms
31  stud               0.000ms
32  stud               0.000ms
33  stud               0.000ms
34  stud               0.000ms
35  stud               0.000ms
36  stud               0.000ms
37  stud               0.000ms
38  stud               0.000ms
39
40  stud               0.000ms
41  stud               0.000ms
42  stud               0.000ms
43  stud               0.000ms
44  stud               0.000ms
45  stud               0.000ms
46  student009_Yahtzee_036.java  1':08"
47  student009_Yahtzee_037.java  0':37"
48  student009_Yahtzee_038.java  1':18"
49  student009_Yahtzee_039.java  0':50"
50  student009_Yahtzee_040.java  2':16"
51  student009_Yahtzee_041.java  0':27"
52  student009_Yahtzee_042.java  0':22"
53  student009_Yahtzee_043.java  0':27"
54  student009_Yahtzee_044.java  2':54"
55  student009_Yahtzee_045.java  0':33"
56  student009_Yahtzee_046.java  0':26"
57  student009_Yahtzee_047.java  b r e a k
58  student009_Yahtzee_048.java  1':10"
59  student009_Yahtzee_049.java  b r e a k
60  student009_Yahtzee_050.java  4':26"
61  student009_Yahtzee_051.java  2':10"
62  student009_Yahtzee_052.java  1':09"
63  student009_Yahtzee_053.java  0':42"
64  student009_Yahtzee_054.java  n/a

```

A modal dialog titled 'Instrumentation Guidelines' is open, comparing two versions of a 'Chess' class. The left version is blue and the right version is red. Both versions have identical code except for the instrumentation logic. The instrumentation logic is highlighted with yellow boxes around the 'board.move()' calls and the 'isCheckMate()' method.

Figure 83: An example to show how to save the program state in cases when it is not so obvious.

```

1 Number of snapshots: 150
2
3 Total time: 04hr:33':02"
4 Last snapshot: student009_Yahtzee_150.java
5 Number of lines: 286
6 Number of methods: 12
7
8 Time breakdown:
9
10   Filename          Time Diff      Total Time     Status
11   stud              0:00:00       0:00:00       Compilable
12   stud              0:00:00       0:00:00       Compilable
13   stud              0:00:00       0:00:00       Compilable
14   stud              0:00:00       0:00:00       Compilable
15   stud              0:00:00       0:00:00       Compilable
16   stud              0:00:00       0:00:00       Compilable
17   stud              0:00:00       0:00:00       Compilable
18   stud              0:00:00       0:00:00       Compilable
19   stud              0:00:00       0:00:00       Compilable
20   stud              0:00:00       0:00:00       Compilable
21   stud              0:00:00       0:00:00       Compilable
22   stud              0:00:00       0:00:00       Compilable
23   stud              0:00:00       0:00:00       Compilable
24   stud              0:00:00       0:00:00       Compilable
25   stud              0:00:00       0:00:00       Compilable
26   stud              0:00:00       0:00:00       Compilable
27   stud              0:00:00       0:00:00       Compilable
28   stud              0:00:00       0:00:00       Compilable
29   stud              0:00:00       0:00:00       Compilable
30   stud              0:00:00       0:00:00       Compilable
31   stud              0:00:00       0:00:00       Compilable
32   stud              0:00:00       0:00:00       Compilable
33   stud              0:00:00       0:00:00       Compilable
34   stud              0:00:00       0:00:00       Compilable
35   stud              0:00:00       0:00:00       Compilable
36   stud              0:00:00       0:00:00       Compilable
37   stud              0:00:00       0:00:00       Compilable
38   stud              0:00:00       0:00:00       Compilable
39   stud              0:00:00       0:00:00       Compilable
40   stud              0:00:00       0:00:00       Compilable
41   stud              0:00:00       0:00:00       Compilable
42   stud              0:00:00       0:00:00       Compilable
43   stud              0:00:00       0:00:00       Compilable
44   stud              0:00:00       0:00:00       Compilable
45   stud              0:00:00       0:00:00       Compilable
46 student009_Yahtzee_036.java  1':08"   0lhr:23':20"   Compilable
47 student009_Yahtzee_037.java  0':37"   0lhr:23':57"   Compilable
48 student009_Yahtzee_038.java  1':18"   0lhr:25':15"   Compilable
49 student009_Yahtzee_039.java  0':50"   0lhr:26':05"   Compilable
50 student009_Yahtzee_040.java  2':16"   0lhr:28':21"   Compilable
51 student009_Yahtzee_041.java  0':27"   0lhr:28':48"   Compilable
52 student009_Yahtzee_042.java  0':22"   0lhr:29':10"   Compilable
53 student009_Yahtzee_043.java  0':27"   0lhr:29':37"   Compilable
54 student009_Yahtzee_044.java  2':54"   0lhr:32':31"   Compilable
55 student009_Yahtzee_045.java  0':33"   0lhr:33':04"   Compilable
56 student009_Yahtzee_046.java  0':26"   0lhr:33':30"   Compilable
57 student009_Yahtzee_047.java  b r e a k   0lhr:33":41"   Compilable
58 student009_Yahtzee_048.java  1':10"   0lhr:34':40"   Compilable
59 student009_Yahtzee_049.java  b r e a k   0lhr:34":41"   Compilable
60 student009_Yahtzee_050.java  4':26"   0lhr:39':06"   Compilable
61 student009_Yahtzee_051.java  2':10"   0lhr:41':16"   Compilable
62 student009_Yahtzee_052.java  1':09"   0lhr:42':25"   Compilable
63 student009_Yahtzee_053.java  0":42"   0lhr:43":41"   Compilable

```

Figure 84: An example to show how to save the program state in cases when it is not so obvious.

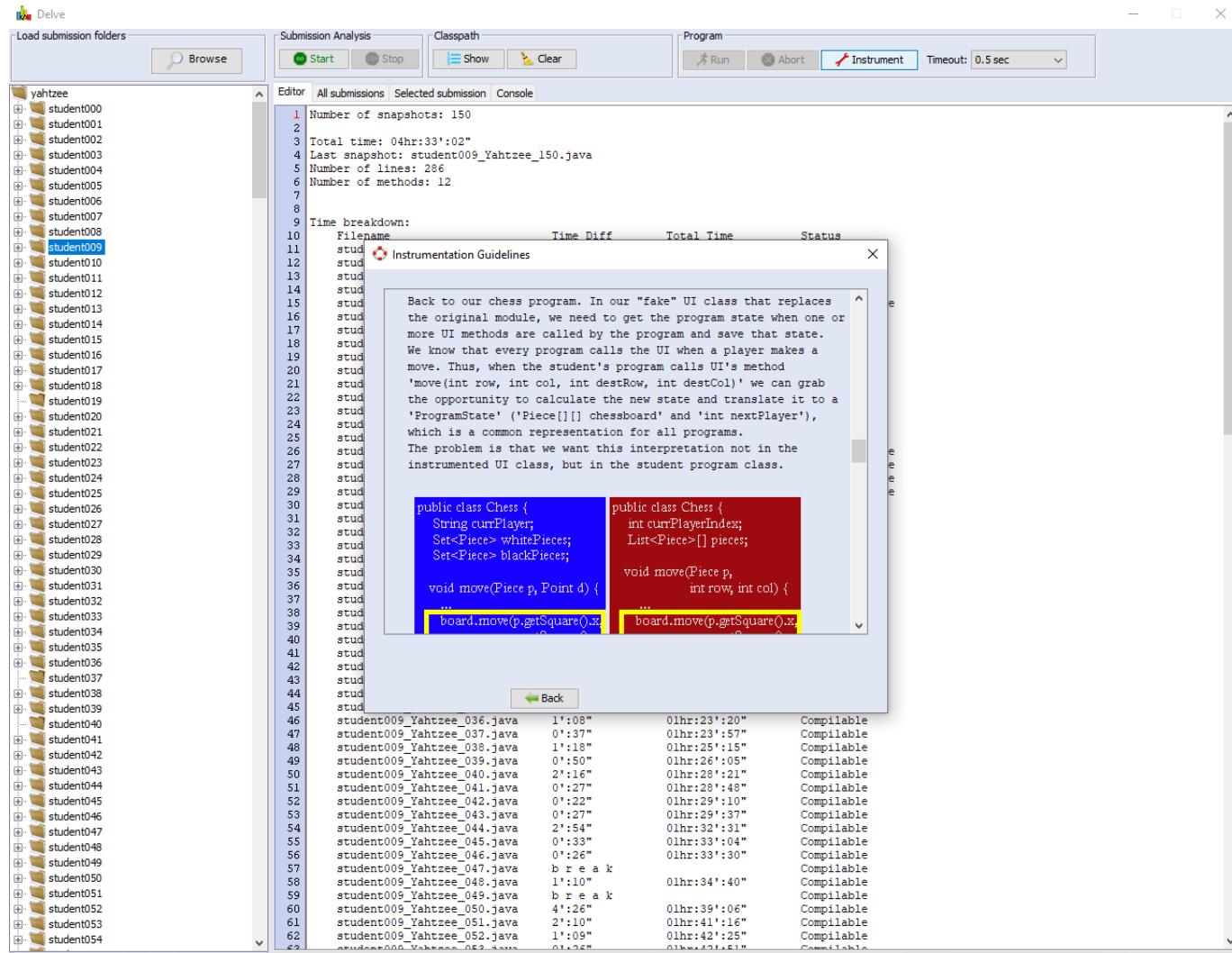


Figure 85: An example to show how to save the program state in cases when it is not so obvious.

The screenshot shows the Delve debugger interface. On the left, a tree view lists submission folders under 'Load submission folders'. The 'student009' folder is selected. The main window has tabs for 'Submission Analysis' (with 'Start' and 'Stop' buttons), 'Classpath' (with 'Show' and 'Clear' buttons), 'Program' (with 'Run', 'Abort', 'Instrument' checked, and 'Timeout: 0.5 sec' dropdown), and 'Console'.

The 'Console' tab displays the following output:

```

1 Number of snapshots: 150
2
3 Total time: 04hr:33':02"
4 Last snapshot: student009_Yahtzee_150.java
5 Number of lines: 286
6 Number of methods: 12
7
8 Time breakdown:
9
10  Filename           Time Diff      Total Time      Status
11  stud               0.000ms
12  stud               0.000ms
13  stud               0.000ms
14  stud               0.000ms
15  stud               0.000ms
16  stud               0.000ms
17  stud               0.000ms
18  stud               0.000ms
19  stud               0.000ms
20  stud               0.000ms
21  stud               0.000ms
22  stud               0.000ms
23  stud               0.000ms
24  stud               0.000ms
25  stud               0.000ms
26  stud               0.000ms
27  stud               0.000ms
28  stud               0.000ms
29  stud               0.000ms
30  stud               0.000ms
31  stud               0.000ms
32  stud               0.000ms
33  stud               0.000ms
34  stud               0.000ms
35  stud               0.000ms
36  stud               0.000ms
37  stud               0.000ms
38  stud               0.000ms
39  stud               0.000ms
40  stud               0.000ms
41  stud               0.000ms
42  stud               0.000ms
43  stud               0.000ms
44  stud               0.000ms
45  stud               0.000ms
46  student009_Yahtzee_036.java  1':08"
47  student009_Yahtzee_037.java  0':37"
48  student009_Yahtzee_038.java  1':18"
49  student009_Yahtzee_039.java  0':50"
50  student009_Yahtzee_040.java  2':16"
51  student009_Yahtzee_041.java  0':27"
52  student009_Yahtzee_042.java  0':22"
53  student009_Yahtzee_043.java  0':27"
54  student009_Yahtzee_044.java  2':54"
55  student009_Yahtzee_045.java  0':33"
56  student009_Yahtzee_046.java  0':26"
57  student009_Yahtzee_047.java  b r e a k
58  student009_Yahtzee_048.java  1':10"
59  student009_Yahtzee_049.java  b r e a k
60  student009_Yahtzee_050.java  4':26"
61  student009_Yahtzee_051.java  2':10"
62  student009_Yahtzee_052.java  1':09"
63  student009_Yahtzee_053.java  0":42"
64  student009_Yahtzee_054.java  n/a

```

A modal dialog titled 'Instrumentation Guidelines' compares two versions of the 'Chess' class. The left version is blue and the right version is red. Both versions have identical code except for the instrumentation code highlighted in yellow. The instrumentation code is as follows:

```

public class Chess {
    String currPlayer;
    Set<Piece> whitePieces;
    Set<Piece> blackPieces;

    void move(Piece p, Point d) {
        board.move(p.getSquare().x,
                   p.getSquare().y,
                   d.x, d.y);
    }

    boolean canMove(Piece p, Point d) {
        ...
    }

    etc.
}

```

Figure 86: An example to show how to save the program state in cases when it is not so obvious.

The screenshot shows the Delve tool interface. On the left, a tree view displays submission folders under 'Load submission folders'. A folder named 'yahtzee' is expanded, showing numerous student submissions from student000 to student054. In the center, the 'Program' tab is active, showing a code editor with the following Java code:

```

1 Number of snapshots: 150
2
3 Total time: 04hr:33':02"
4 Last snapshot: student009_Yahtzee_150.java
5 Number of lines: 286
6 Number of methods: 12
7
8 Time breakdown:
9
10  Filename           Time Diff   Total Time   Status
11  stud
12  stud
13  stud
14  stud
15  stud
16  stud
17  stud
18  stud
19  stud
20  stud
21  stud
22  stud
23  stud
24  stud
25  stud
26  stud
27  stud
28  stud
29  stud
30  stud
31  stud
32  stud
33  stud
34  stud
35  stud
36  stud
37  stud
38  stud
39  stud
40  stud
41  stud
42  stud
43  stud
44  stud
45  stud
46  student009_Yahtzee_036.java  1':08"
47  student009_Yahtzee_037.java  0':37"
48  student009_Yahtzee_038.java  1':18"
49  student009_Yahtzee_039.java  0':50"
50  student009_Yahtzee_040.java  2':16"
51  student009_Yahtzee_041.java  0':27"
52  student009_Yahtzee_042.java  0':22"
53  student009_Yahtzee_043.java  0':27"
54  student009_Yahtzee_044.java  2':54"
55  student009_Yahtzee_045.java  0':33"
56  student009_Yahtzee_046.java  0':26"
57  student009_Yahtzee_047.java  b r e a k
58  student009_Yahtzee_048.java  1':10"
59  student009_Yahtzee_049.java  b r e a k
60  student009_Yahtzee_050.java  4':26"
61  student009_Yahtzee_051.java  2':10"
62  student009_Yahtzee_052.java  1':09"
63  student009_Yahtzee_053.java  0':42"
64  student009_Yahtzee_054.java  n/a

```

A yellow callout box highlights a section of the code in the editor:

```

UI Class
public class Chessboard extends JFrame {
    ProgramState state;
    public void move(int row,
                     int col,
                     int destRow,
                     int destCol) {
        state = ...
    }
    public void touchPiece(int row, int col) {
        ...
    }
    etc.
}

```

Two yellow arrows point from the top of the UI Class title to the start of the code block, indicating the scope of the highlighted code.

Figure 87: An example to show how to save the program state in cases when it is not so obvious.

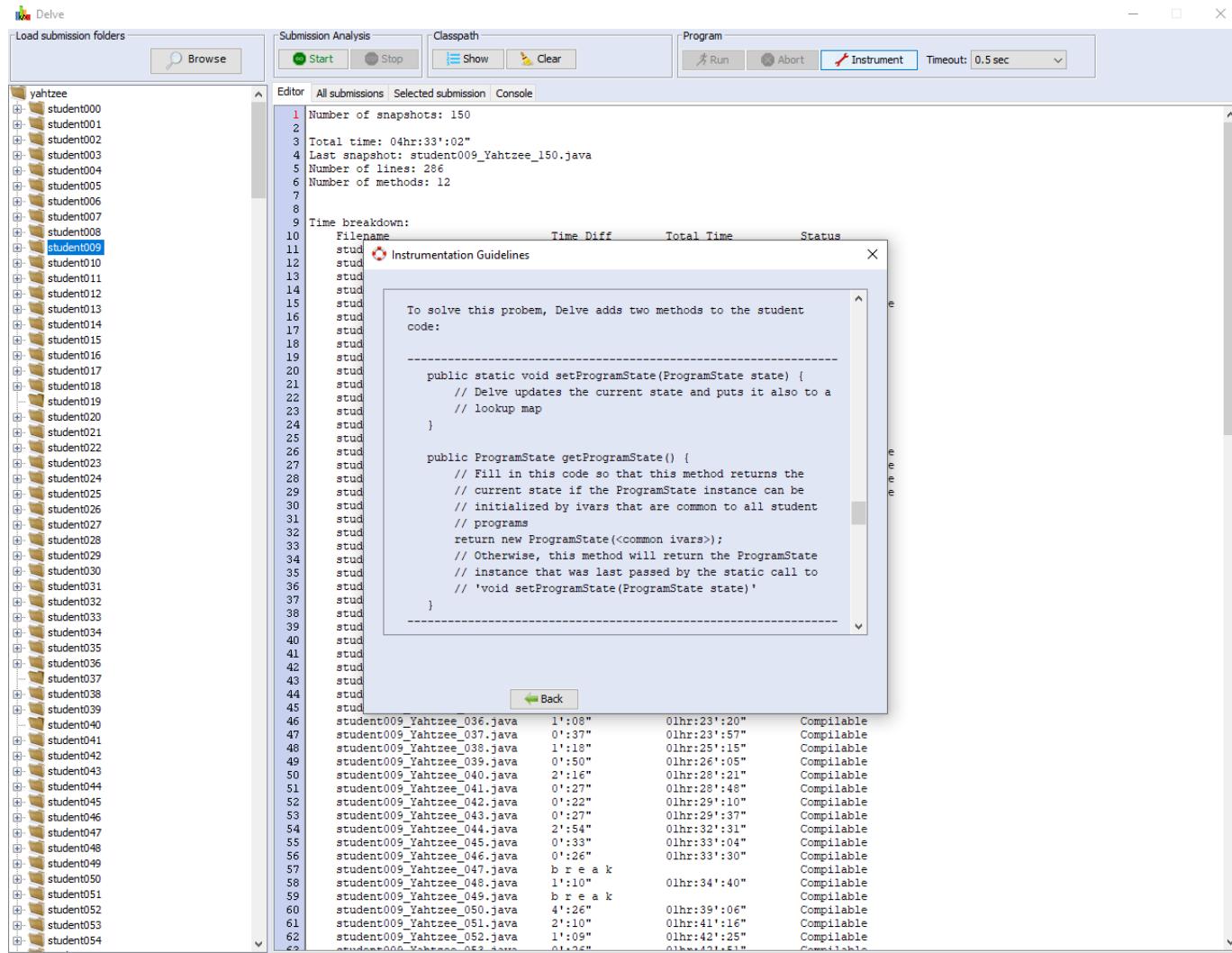


Figure 88: An example to show how to save the program state in cases when it is not so obvious.

The screenshot shows the Delve debugger interface. On the left, a tree view displays submission folders under 'Load submission folders'. The 'student009' folder is selected. The main window has tabs for 'Submission Analysis', 'Classpath', 'Program', and 'Instrument'. The 'Instrument' tab is active, showing a timeout of '0.5 sec'. The 'Program' tab displays code and a 'Time breakdown' table. A modal dialog titled 'Instrumentation Guidelines' compares two versions of the 'Chess' class. The left version is blue and contains logic for moving pieces and setting the current state. The right version is red and contains the same logic but lacks the 'currState' variable. Below the dialog, the 'Program' tab shows a table of submissions with their execution times and compilation status.

Filenumber	Filename	Time Diff	Total Time	Status
1	Number of snapshots: 150			
2				
3	Total time: 04hr:33':02"			
4	Last snapshot: student009_Yahtzee_150.java			
5	Number of lines: 286			
6	Number of methods: 12			
7				
8				
9	Time breakdown:			
10	Filename	Time Diff	Total Time	Status
11	stud			
12	stud			
13	stud			
14	stud			
15	stud			
16	stud			
17	stud			
18	stud			
19	stud			
20	stud			
21	stud			
22	stud			
23	stud			
24	stud			
25	stud			
26	stud			
27	stud			
28	stud			
29	stud			
30	stud			
31	stud			
32	stud			
33	stud			
34	stud			
35	stud			
36	stud			
37	stud			
38	stud			
39	stud			
40	stud			
41	stud			
42	stud			
43	stud			
44	stud			
45	stud			
46	student009_Yahtzee_036.java	1':08"	01hr:23':20"	Compilable
47	student009_Yahtzee_037.java	0':37"	01hr:23':57"	Compilable
48	student009_Yahtzee_038.java	1':18"	01hr:25':15"	Compilable
49	student009_Yahtzee_039.java	0':50"	01hr:26':05"	Compilable
50	student009_Yahtzee_040.java	2':16"	01hr:28':21"	Compilable
51	student009_Yahtzee_041.java	0':27"	01hr:28':48"	Compilable
52	student009_Yahtzee_042.java	0':22"	01hr:29':10"	Compilable
53	student009_Yahtzee_043.java	0':27"	01hr:29':37"	Compilable
54	student009_Yahtzee_044.java	2':54"	01hr:32':31"	Compilable
55	student009_Yahtzee_045.java	0':33"	01hr:33':04"	Compilable
56	student009_Yahtzee_046.java	0':26"	01hr:33':30"	Compilable
57	student009_Yahtzee_047.java	b r e a k		Compilable
58	student009_Yahtzee_048.java	1':10"	01hr:34':40"	Compilable
59	student009_Yahtzee_049.java	b r e a k		Compilable
60	student009_Yahtzee_050.java	4':26"	01hr:39':06"	Compilable
61	student009_Yahtzee_051.java	2':10"	01hr:41':16"	Compilable
62	student009_Yahtzee_052.java	1':09"	01hr:42':25"	Compilable
63	student009_Yahtzee_053.java	01hr:42':41"		Compilable

Figure 89: An example to show how to save the program state in cases when it is not so obvious.

The screenshot shows the Delve tool interface. On the left, there's a tree view of submission folders under 'Load submission folders'. The 'student009' folder is selected. The main area has tabs for 'Submission Analysis' (with 'Start' and 'Stop' buttons), 'Classpath' (with 'Show' and 'Clear' buttons), 'Program' (with 'Run', 'Abort', 'Instrument' checked, and 'Timeout: 0.5 sec' dropdown), and 'Console'. The 'Console' tab is active, displaying a log of snapshots taken. A modal window titled 'UI Class' is overlaid on the code editor. The code editor shows a Java class named 'Chessboard' extending 'JFrame'. A specific line of code, 'Chess.setProgramState(state);', is highlighted with a red box and circled by two red arrows, indicating it's a point of interest for saving program state.

```

1 Number of snapshots: 150
2
3 Total time: 04hr:33':02"
4 Last snapshot: student009_Yahtzee_150.java
5 Number of lines: 286
6 Number of methods: 12
7
8 Time breakdown:
9
10 Filename Time Diff Total Time Status
11 stud
12 stud
13 stud
14 stud
15 stud
16 stud
17 stud
18 stud
19 stud
20 stud
21 stud
22 stud
23 stud
24 stud
25 stud
26 stud
27 stud
28 stud
29 stud
30 stud
31 stud
32 stud
33 stud
34 stud
35 stud
36 stud
37 stud
38 stud
39 stud
40 stud
41 stud
42 stud
43 stud
44 stud
45 stud
46 student009_Yahtzee_036.java 1':08"
47 student009_Yahtzee_037.java 0':37"
48 student009_Yahtzee_038.java 1':18"
49 student009_Yahtzee_039.java 0':50"
50 student009_Yahtzee_040.java 2':16"
51 student009_Yahtzee_041.java 0':27"
52 student009_Yahtzee_042.java 0':22"
53 student009_Yahtzee_043.java 0':27"
54 student009_Yahtzee_044.java 2':54"
55 student009_Yahtzee_045.java 0':33"
56 student009_Yahtzee_046.java 0':26"
57 student009_Yahtzee_047.java b r e a k
58 student009_Yahtzee_048.java 1':10"
59 student009_Yahtzee_049.java b r e a k
60 student009_Yahtzee_050.java 4':26"
61 student009_Yahtzee_051.java 2':10"
62 student009_Yahtzee_052.java 1':09"
63 student009_Yahtzee_053.java 0':42"
64 student009_Yahtzee_054.java n/a

```

Figure 90: An example to show how to save the program state in cases when it is not so obvious.

The screenshot shows the Delve debugger interface with the following details:

- Load submission folders:** A tree view showing submissions from students 000 to 054 under a folder named "yahtzee".
- Submission Analysis:** Tab showing the following statistics:
 - Number of snapshots: 150
 - Total time: 04hr:33'02"
 - Last snapshot: student009_Yahtzee_150.java
 - Number of lines: 286
 - Number of methods: 12
- Classpath:** Buttons for Show and Clear.
- Program:** Buttons for Run, Abort, Instrument (highlighted), and Timeout: 0.5 sec.
- Editor:** Shows the Java code for student009_Yahtzee_150.java. An overlay window titled "Instrumentation Guidelines" is displayed over the code editor, containing the following text:


```

      }
      The instrumented UI class can pass back the new state by calling
      the static method 'void setProgramState(ProgramState state)'.
      Note: The static method is added automatically.
      -----
      package <same package as GUI_Class to replace>;
      public class <GUI_Class>
      {
          public void move(int row,int col,int destRow,int destCol) {
              // Make sure there is a piece at (row, col) that
              // belongs to the player who makes the move
              // Make sure this piece can move to the destination
              // square (destRow, destCol)
              // Determine if an opponent piece is captured
      }
      
```
- File List:** A list of all Java files in the submission, showing their names, execution times, and compilation status.

Figure 91: An example to show how to save the program state in cases when it is not so obvious.

The screenshot shows the Delve debugger interface with the following details:

- Load submission folders:** A tree view showing submissions from students 000 to 054, with "student009" selected.
- Submission Analysis:** Shows 150 snapshots, a total time of 04hr:33'02", and the last snapshot is student009_Yahtzee_150.java.
- Classpath:** Buttons for "Show" and "Clear".
- Program:** Buttons for "Run", "Abort", "Instrument" (which is selected), and a "Timeout: 0.5 sec" dropdown.
- Editor:** Displays the Java code for student009_Yahtzee_150.java. A modal window titled "Instrumentation Guidelines" is open over the code editor, containing the following text:

```

// Update the program state:
// remove the captured piece (if any), move the piece
// to the destination and change the player's turn to
// play next

// Pass back the new ProgramState to the class which
// called this method (for convenience static method)
<ProgramClass>.setProgramState(state);
}

public void touchPiece(int row, int col) {
    // The GUI highlights the squares where the selected
    // piece can land on. Thus, we don't need to do
    // anything and leave the code empty since it does
    // not affect the current program state
}

<other GUI methods>
...
}

```

- Table:** A table showing the execution history of the program, listing each snapshot with its duration and status.

Snapshot	Time	Total Time	Status
student009_Yahtzee_036.java	1':08"	01hr:23':20"	Compilable
student009_Yahtzee_037.java	0':37"	01hr:23':57"	Compilable
student009_Yahtzee_038.java	1':18"	01hr:25':15"	Compilable
student009_Yahtzee_039.java	0':50"	01hr:26':05"	Compilable
student009_Yahtzee_040.java	2':16"	01hr:28':21"	Compilable
student009_Yahtzee_041.java	0':27"	01hr:28':48"	Compilable
student009_Yahtzee_042.java	0':22"	01hr:29':10"	Compilable
student009_Yahtzee_043.java	0':27"	01hr:29':37"	Compilable
student009_Yahtzee_044.java	2':54"	01hr:32':31"	Compilable
student009_Yahtzee_045.java	0':33"	01hr:33':04"	Compilable
student009_Yahtzee_046.java	0':26"	01hr:33':30"	Compilable
student009_Yahtzee_047.java	b r e a k		Compilable
student009_Yahtzee_048.java	1':10"	01hr:34':40"	Compilable
student009_Yahtzee_049.java	b r e a k		Compilable
student009_Yahtzee_050.java	4':26"	01hr:39':06"	Compilable
student009_Yahtzee_051.java	2':10"	01hr:41':16"	Compilable
student009_Yahtzee_052.java	1':09"	01hr:42':25"	Compilable
student009_Yahtzee_053.java	0':42"	01hr:43':11"	Compilable

Figure 92: An example to show how to save the program state in cases when it is not so obvious.

In our example, we save the program state in one of the UI methods that are called in every program (see **Figure 56**).

To add the code for the **getProgramState()** in the **GetState** class, the user can click the **Add** button.

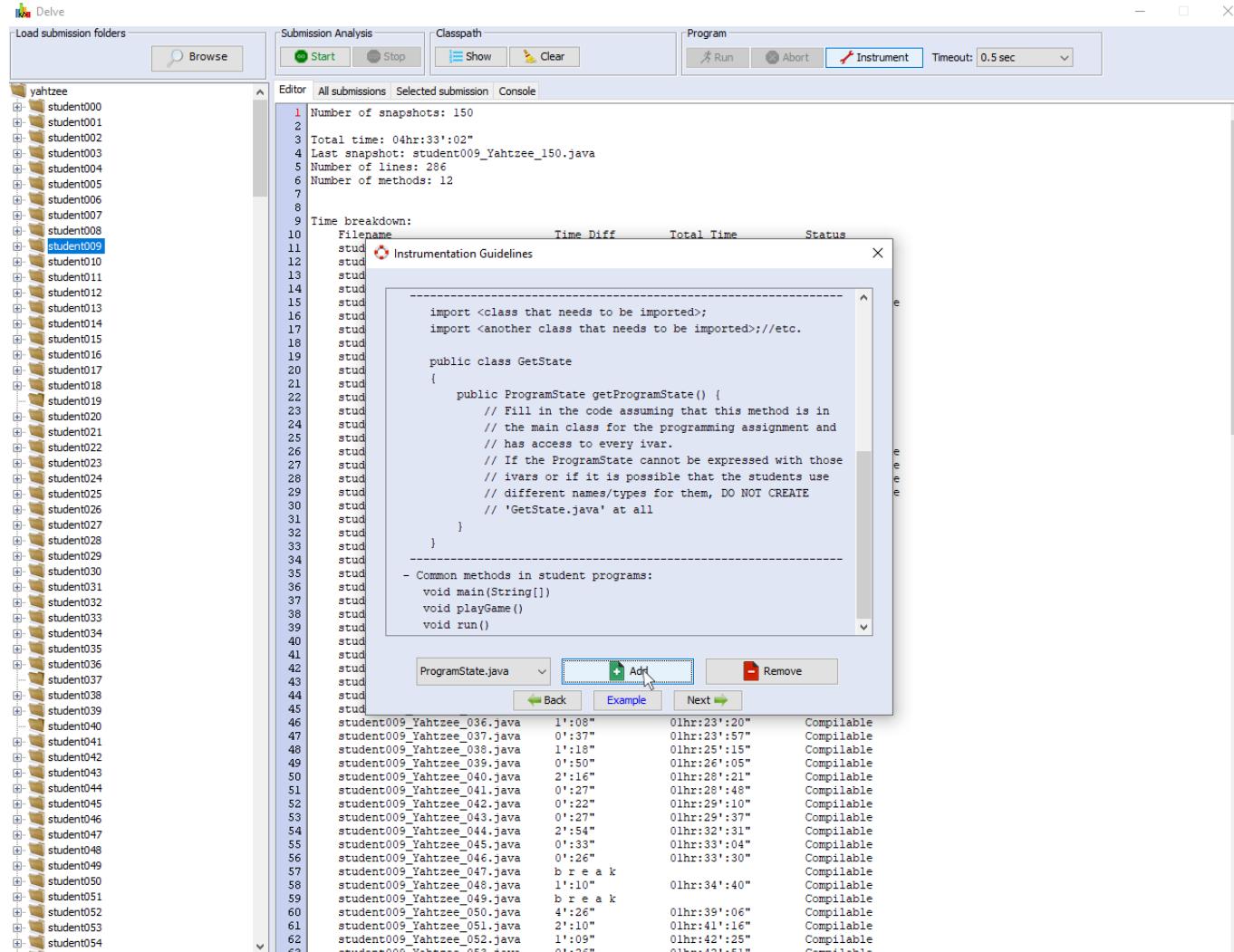


Figure 93: Click the *Add* button to locate file *GetState.java* which includes method *public ProgramState getProgramState()*.

and then locate file **GetState.java**.

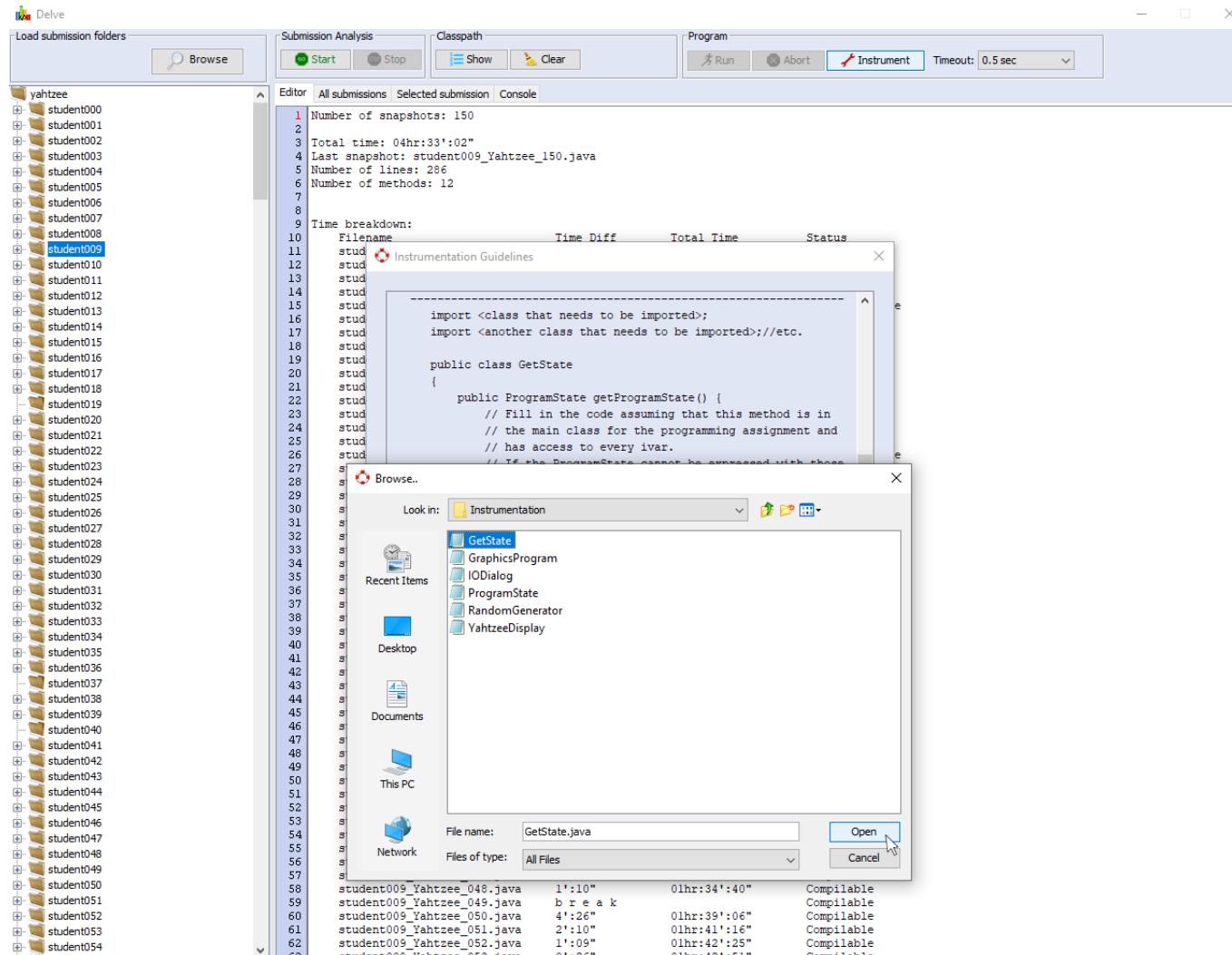
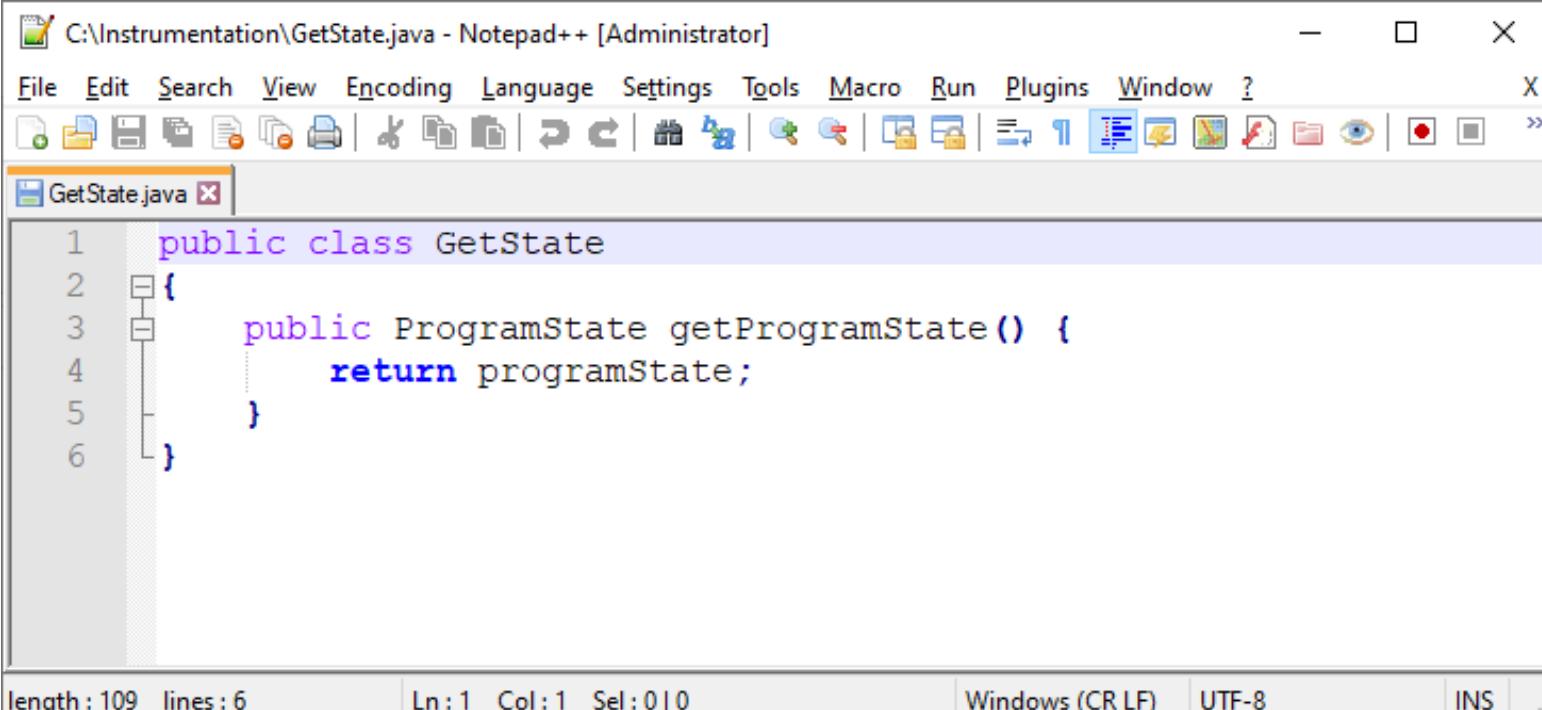


Figure 94: The user can now locate *GetState.java* to specify method *public ProgramState getProgramState()*.

In our example, the code for **GetState.java** is the default behavior which means that we could have omitted this step. This is useful for assignments where one can express the program state via common instance variables for all student submissions.

In most cases, **GetState.java** is unlikely to provide a different code to return the program state other than the default in **Figure 95**.



The screenshot shows a Notepad++ window with the title bar "C:\Instrumentation\GetState.java - Notepad++ [Administrator]". The menu bar includes File, Edit, Search, View, Encoding, Language, Settings, Tools, Macro, Run, Plugins, Window, and Help. The toolbar below has various icons for file operations like Open, Save, Print, and Find. The main editor area contains the following Java code:

```
1 public class GetState
2 {
3     public ProgramState getProgramState() {
4         return programState;
5     }
6 }
```

The status bar at the bottom shows "length : 109 lines : 6" and "Ln : 1 Col : 1 Sel : 0 | 0". It also indicates the encoding is Windows (CR LF), the file is in UTF-8, and the current mode is INS.

Figure 95: Default behavior for *GetState* (i.e., if *GetState.java* is not specified).

The goal is to set the program state in a method that all student programs call. This method can be in any class (e.g., UI class), which may not have direct access to the instance of the main module (in our example the main module is class Chess). To deal with that, Delve conveniently adds to the main module a public static method that we can access from any class: **public static void setProgramState(ProgramState state)** which is shown in **Figure 89** for our example. Then, the UI makes a direct call to it to set the program state (**Figure 90**).

At this point we have completed the fifth step and are ready to move to the next step.

The screenshot shows the Delve tool interface with the following details:

- Load submission folders:** A tree view showing a folder named "yahtzee" containing numerous sub-folders labeled "student000" through "student054".
- Submission Analysis:** Buttons for "Start" and "Stop".
- Classpath:** Buttons for "Show" and "Clear".
- Program:** Buttons for "Run", "Abort", "Instrument" (which is selected), and a "Timeout: 0.5 sec" dropdown.
- Editor:** Displays the following code snippet from "ProgramState.java":

```
import <class that needs to be imported>;
import <another class that needs to be imported>/etc.

public class GetState
{
    public ProgramState getProgramState() {
        // Fill in the code assuming that this method is in
        // the main class for the programming assignment and
        // has access to every ivar.
        // If the ProgramState cannot be expressed with those
        // ivars or if it is possible that the students use
        // different names/types for them, DO NOT CREATE
        // 'GetState.java' at all
    }
}

- Common methods in student programs:
void main(String[])
void playGame()
void run()
```
- Time Diff:** A table showing execution times for various Java files:

File	Time Diff	Total Time	Status
student009_Yahtzee_036.java	1':08"	0lhr:23':20"	Compilable
student009_Yahtzee_037.java	0':37"	0lhr:23':57"	Compilable
student009_Yahtzee_038.java	1':18"	0lhr:25':15"	Compilable
student009_Yahtzee_039.java	0':50"	0lhr:26':05"	Compilable
student009_Yahtzee_040.java	2':16"	0lhr:28':21"	Compilable
student009_Yahtzee_041.java	0':27"	0lhr:28':48"	Compilable
student009_Yahtzee_042.java	0':22"	0lhr:29':10"	Compilable
student009_Yahtzee_043.java	0':27"	0lhr:29':37"	Compilable
student009_Yahtzee_044.java	2':54"	0lhr:32':31"	Compilable
student009_Yahtzee_045.java	0':33"	0lhr:33':04"	Compilable
student009_Yahtzee_046.java	0':26"	0lhr:33':30"	Compilable
student009_Yahtzee_047.java	b r e a k		Compilable
student009_Yahtzee_048.java	1':10"	0lhr:34':40"	Compilable
student009_Yahtzee_049.java	b r e a k		Compilable
student009_Yahtzee_050.java	4':26"	0lhr:39':06"	Compilable
student009_Yahtzee_051.java	2':10"	0lhr:41':16"	Compilable
student009_Yahtzee_052.java	1':09"	0lhr:42':25"	Compilable
student009_Yahtzee_053.java	0':24"	0lhr:42':51"	Compilable

Figure 96: The fifth step is complete. Ready to move on to the sixth step.

7.7 Step 6: Provide a correct implementation as reference

Code instrumentation allows us to capture the states when the program executes. However, we do not know if a program state that gets produced is correct or not. To determine that we need a correct assignment implementation.

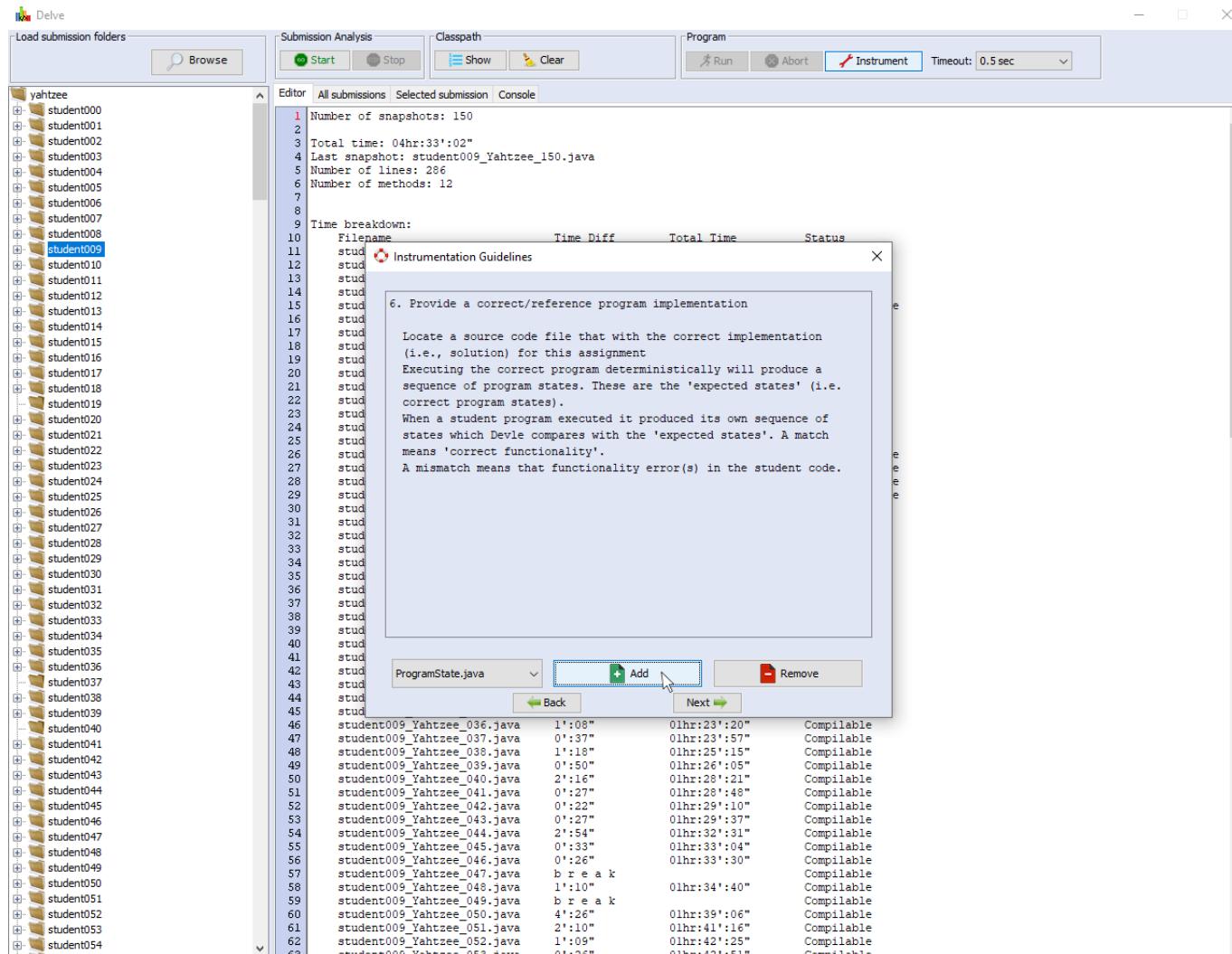


Figure 97: Click the **Add** button to locate the file with the correct implementation.

Delve instruments the code and executes it to produce a set of states. Deterministic nature means that every program runs with the same preconditions and therefore all programs that are correct should produce the same set of states.

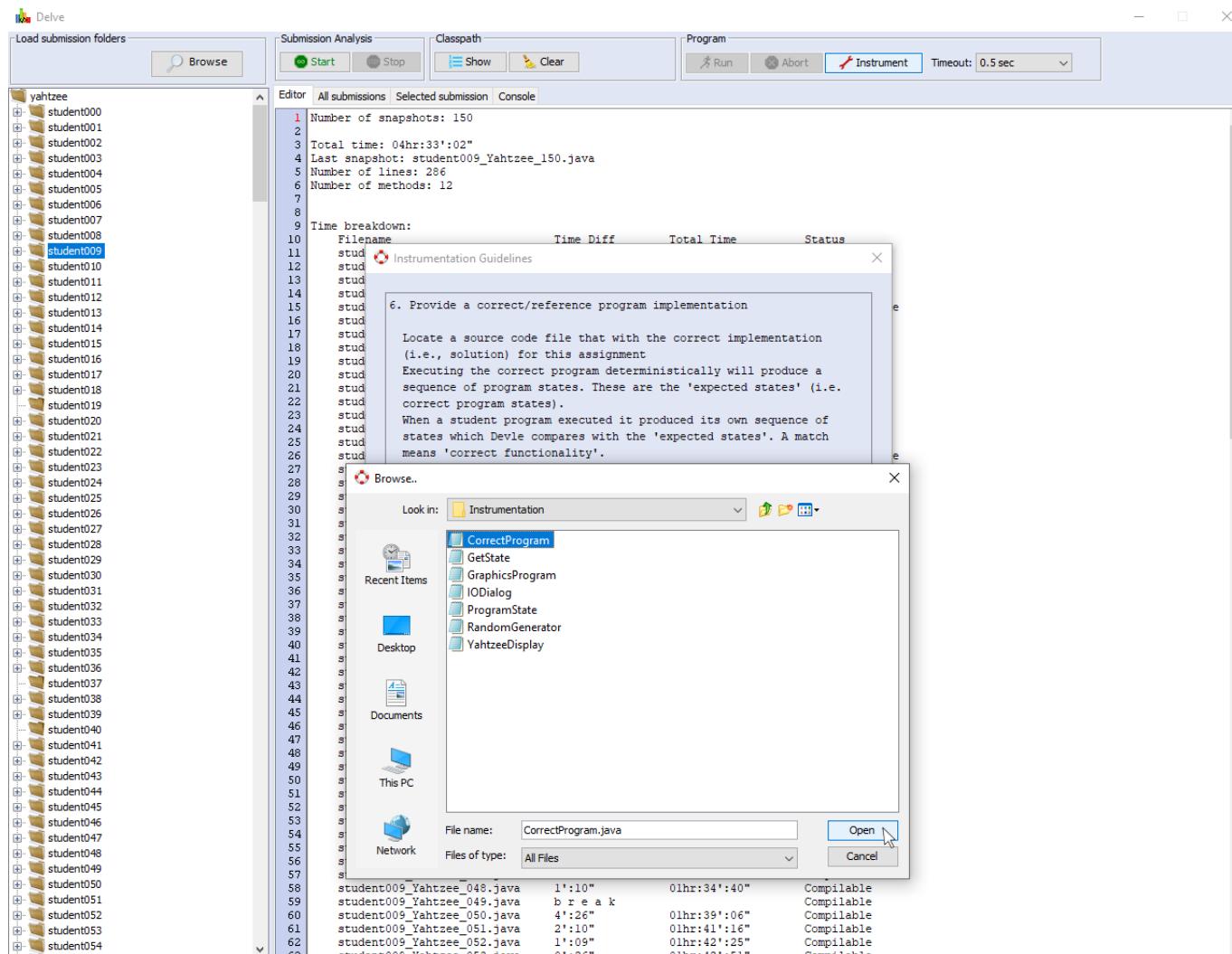


Figure 98: The user can now locate the program with the correct implementation.

At this point we have completed the sixth step and are ready to move to the next and final step.

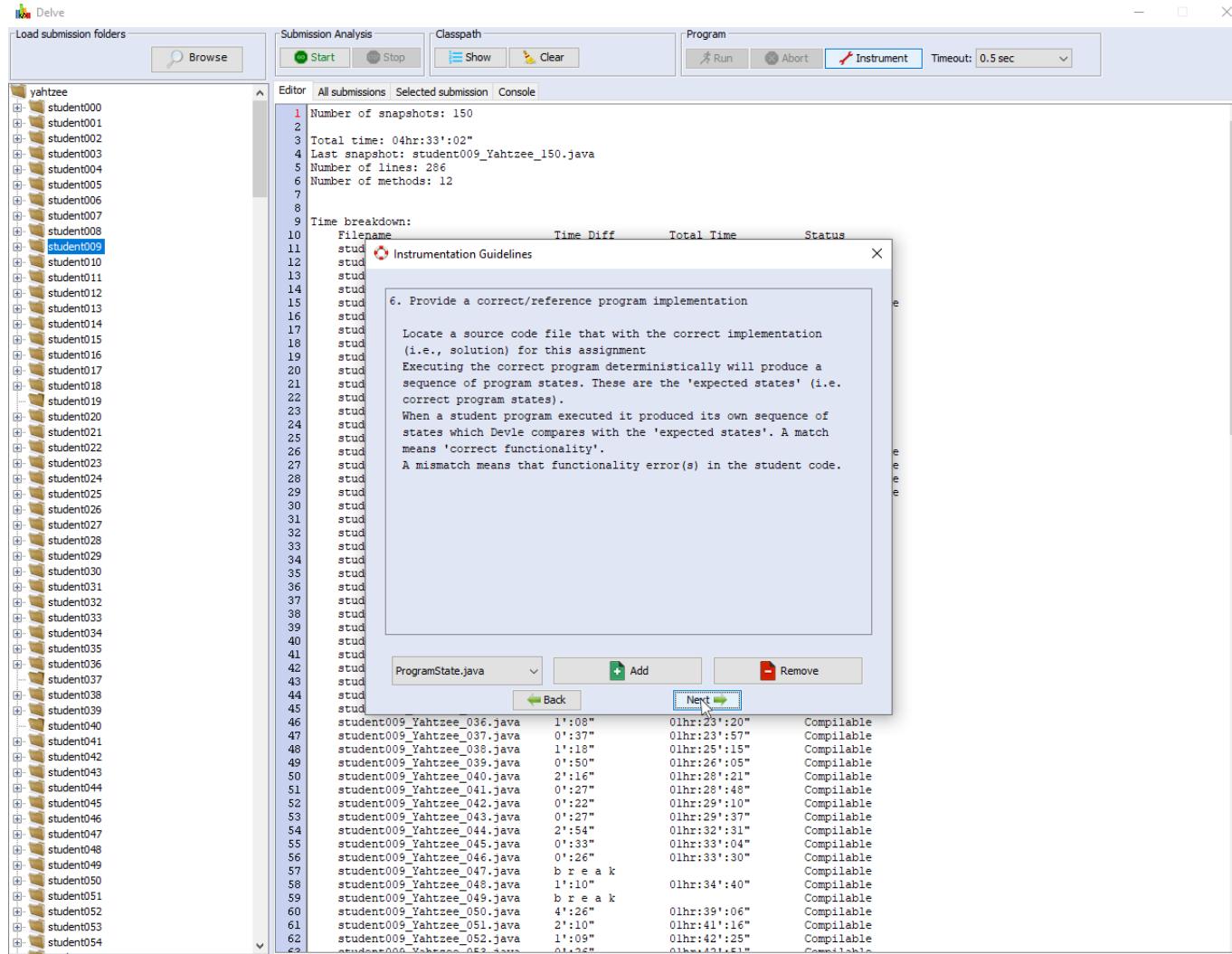


Figure 99: The sixth step is complete. Ready to move on to the last step.

7.8 Step 7: Provide additional libraries (if needed)

Rarely the extra code that is added for instrumentation uses a library (jar-file) that was not required before. If so, we must specify in this step the file(s) for those missing libraries.

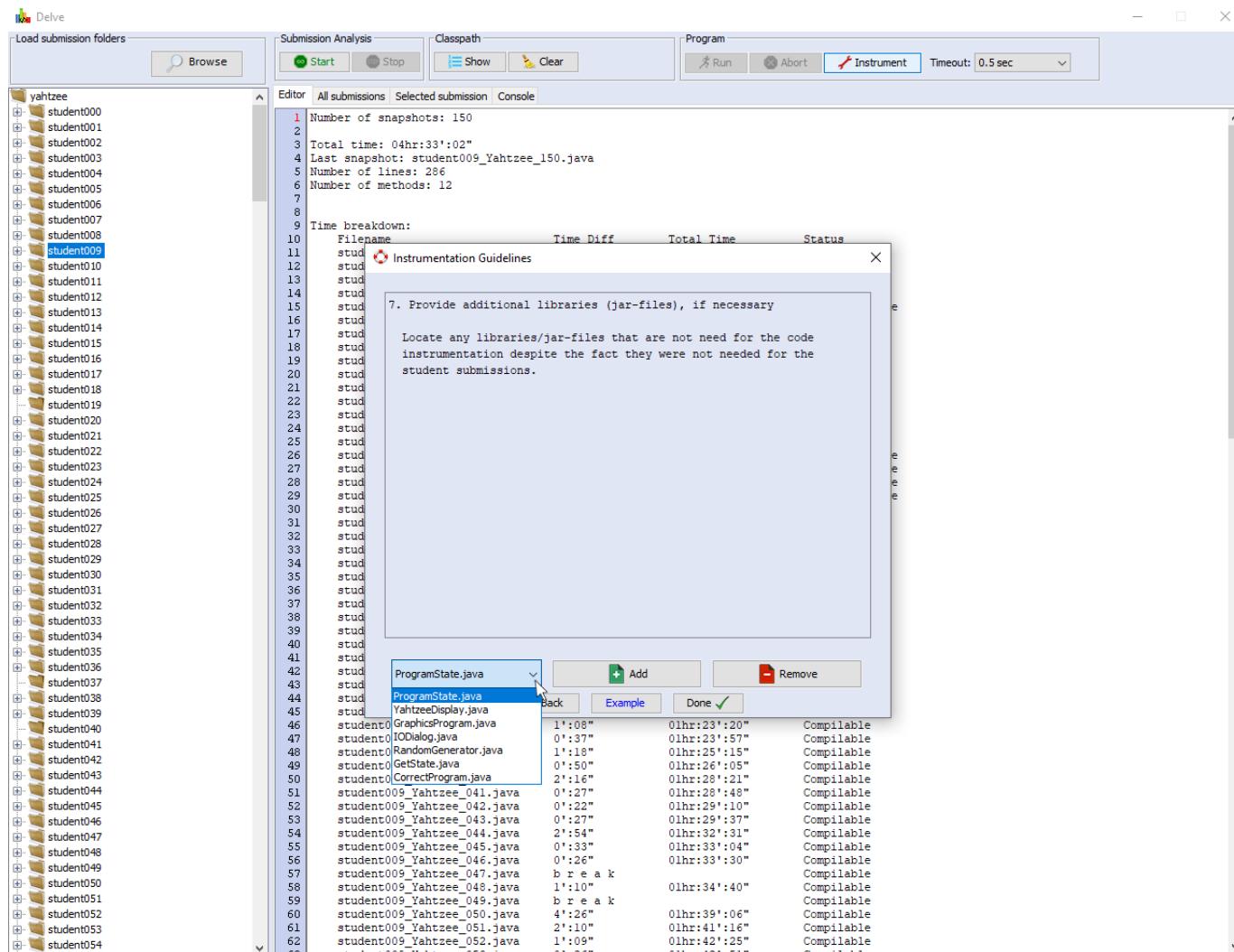


Figure 100: In our example there is no action in this step. The drop-down menu shows all files for the instrumentation.

At this point we are done with the instrumentation.

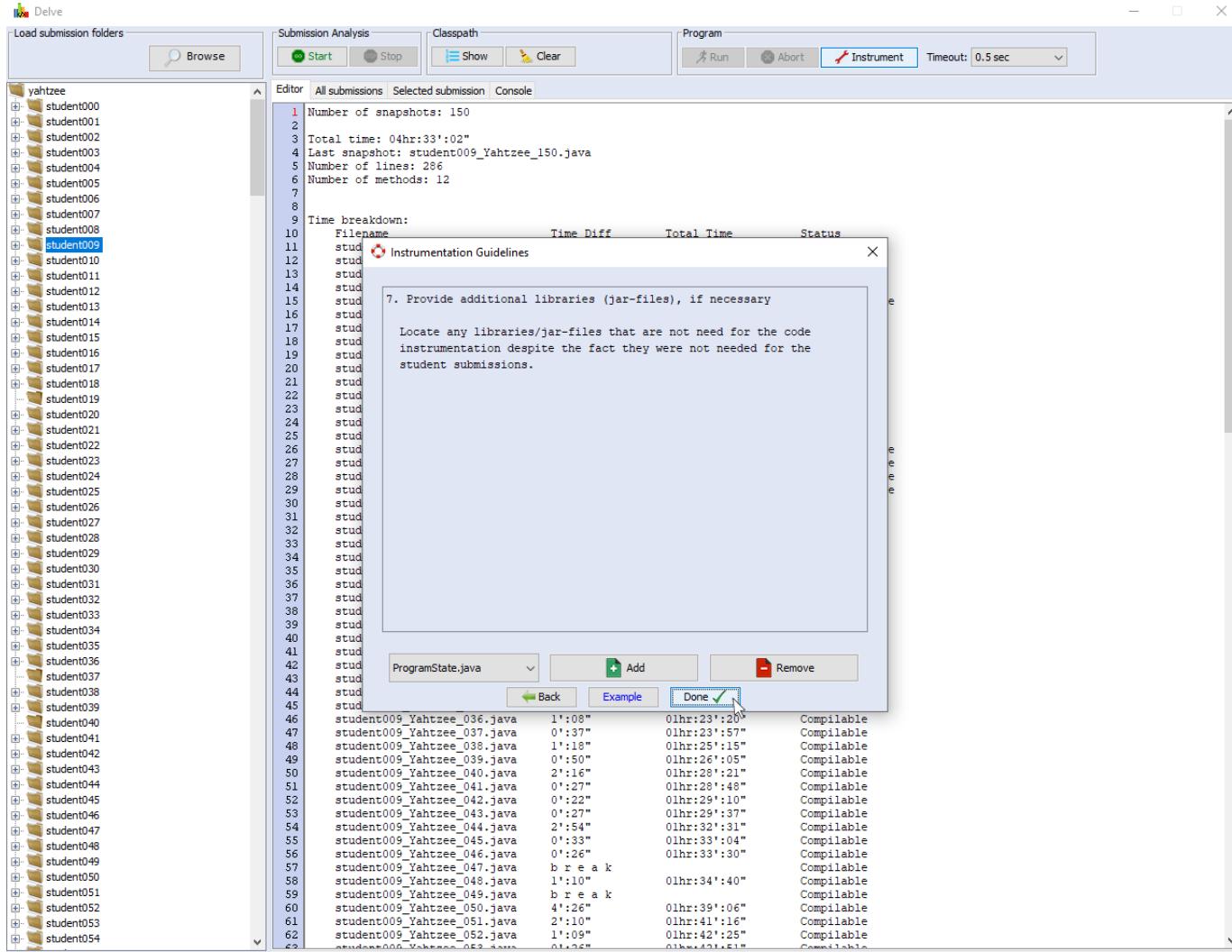
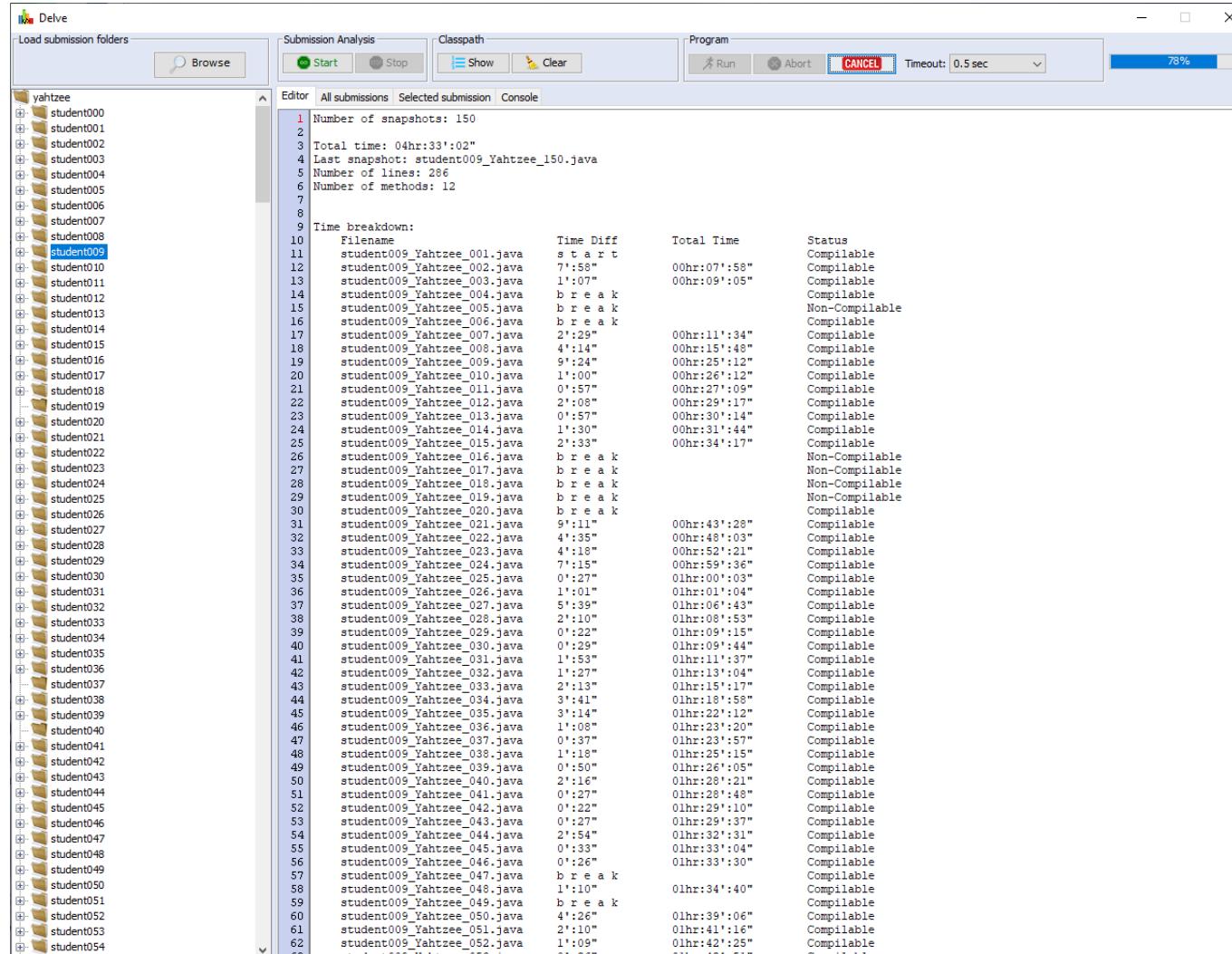


Figure 101: Click the *Done* button to begin the instrumentation analysis.

8. Instrumentation Analysis Results

Once done with the instrumentation setup and click Done (**Figure 101**), Delve starts the analysis which takes 2-3 minutes.



The screenshot shows the Delve software interface with the 'Submission Analysis' tab selected. The left sidebar displays a tree view of submission folders under 'yahtzee'. The main pane shows a list of 150 Java files, each with its filename, total time, time difference, total time, and status. The analysis results are as follows:

File	Total Time	Time Diff	Status
student009_Yahtzee_001.java	00hr:07':58"	start	Compilable
student009_Yahtzee_002.java	00hr:07':58"	7':58"	Compilable
student009_Yahtzee_003.java	00hr:09':05"	1':07"	Compilable
student009_Yahtzee_004.java	00hr:26':12"	b r e a k	Compilable
student009_Yahtzee_005.java	00hr:26':12"	b r e a k	Non-Compilable
student009_Yahtzee_006.java	00hr:26':12"	b r e a k	Compilable
student009_Yahtzee_007.java	00hr:11':34"	21:29"	Compilable
student009_Yahtzee_008.java	00hr:15':48"	4':14"	Compilable
student009_Yahtzee_009.java	00hr:25':12"	9':24"	Compilable
student009_Yahtzee_010.java	00hr:26':12"	1':00"	Compilable
student009_Yahtzee_011.java	00hr:27':09"	0':57"	Compilable
student009_Yahtzee_012.java	00hr:29':17"	2':08"	Compilable
student009_Yahtzee_013.java	00hr:30':14"	0':57"	Compilable
student009_Yahtzee_014.java	00hr:31':44"	1':30"	Compilable
student009_Yahtzee_015.java	00hr:34':17"	21:33"	Compilable
student009_Yahtzee_016.java	00hr:43':28"	b r e a k	Non-Compilable
student009_Yahtzee_017.java	00hr:52':21"	b r e a k	Non-Compilable
student009_Yahtzee_018.java	00hr:59':36"	b r e a k	Non-Compilable
student009_Yahtzee_019.java	01:27"	b r e a k	Non-Compilable
student009_Yahtzee_020.java	01:31"	b r e a k	Compilable
student009_Yahtzee_021.java	01:39"	9':11"	Compilable
student009_Yahtzee_022.java	01:48"	4':35"	Compilable
student009_Yahtzee_023.java	01:52"	4':18"	Compilable
student009_Yahtzee_024.java	01:59"	7':15"	Compilable
student009_Yahtzee_025.java	02:27"	0':27"	Compilable
student009_Yahtzee_026.java	02:31"	1':01"	Compilable
student009_Yahtzee_027.java	02:39"	5':39"	Compilable
student009_Yahtzee_028.java	02:43"	2':10"	Compilable
student009_Yahtzee_029.java	02:47"	0':22"	Compilable
student009_Yahtzee_030.java	02:51"	0':29"	Compilable
student009_Yahtzee_031.java	02:55"	1':53"	Compilable
student009_Yahtzee_032.java	02:59"	1':27"	Compilable
student009_Yahtzee_033.java	03:03"	2':13"	Compilable
student009_Yahtzee_034.java	03:07"	3':41"	Compilable
student009_Yahtzee_035.java	03:11"	3':14"	Compilable
student009_Yahtzee_036.java	03:15"	1':08"	Compilable
student009_Yahtzee_037.java	03:19"	0':37"	Compilable
student009_Yahtzee_038.java	03:23"	1':18"	Compilable
student009_Yahtzee_039.java	03:27"	0':50"	Compilable
student009_Yahtzee_040.java	03:31"	2':16"	Compilable
student009_Yahtzee_041.java	03:35"	0':27"	Compilable
student009_Yahtzee_042.java	03:39"	0':22"	Compilable
student009_Yahtzee_043.java	03:43"	0':27"	Compilable
student009_Yahtzee_044.java	03:47"	2':54"	Compilable
student009_Yahtzee_045.java	03:51"	0':33"	Compilable
student009_Yahtzee_046.java	03:55"	0':26"	Compilable
student009_Yahtzee_047.java	03:59"	b r e a k	Compilable
student009_Yahtzee_048.java	04:03"	1':10"	Compilable
student009_Yahtzee_049.java	04:07"	b r e a k	Compilable
student009_Yahtzee_050.java	04:11"	4':26"	Compilable
student009_Yahtzee_051.java	04:15"	2':10"	Compilable
student009_Yahtzee_052.java	04:19"	1':09"	Compilable
student009_Yahtzee_053.java	04:23"	0':52"	Compilable
student009_Yahtzee_054.java	04:27"	0':52"	Compilable

Figure 102: Once we are done with the instrumentation setup, Delve performs the analysis which takes a while.

Note that only programs that include `public static void main(String[])` can be executed. Delve, shows after the analysis an aggregated error message that lists the files that did not have this method.

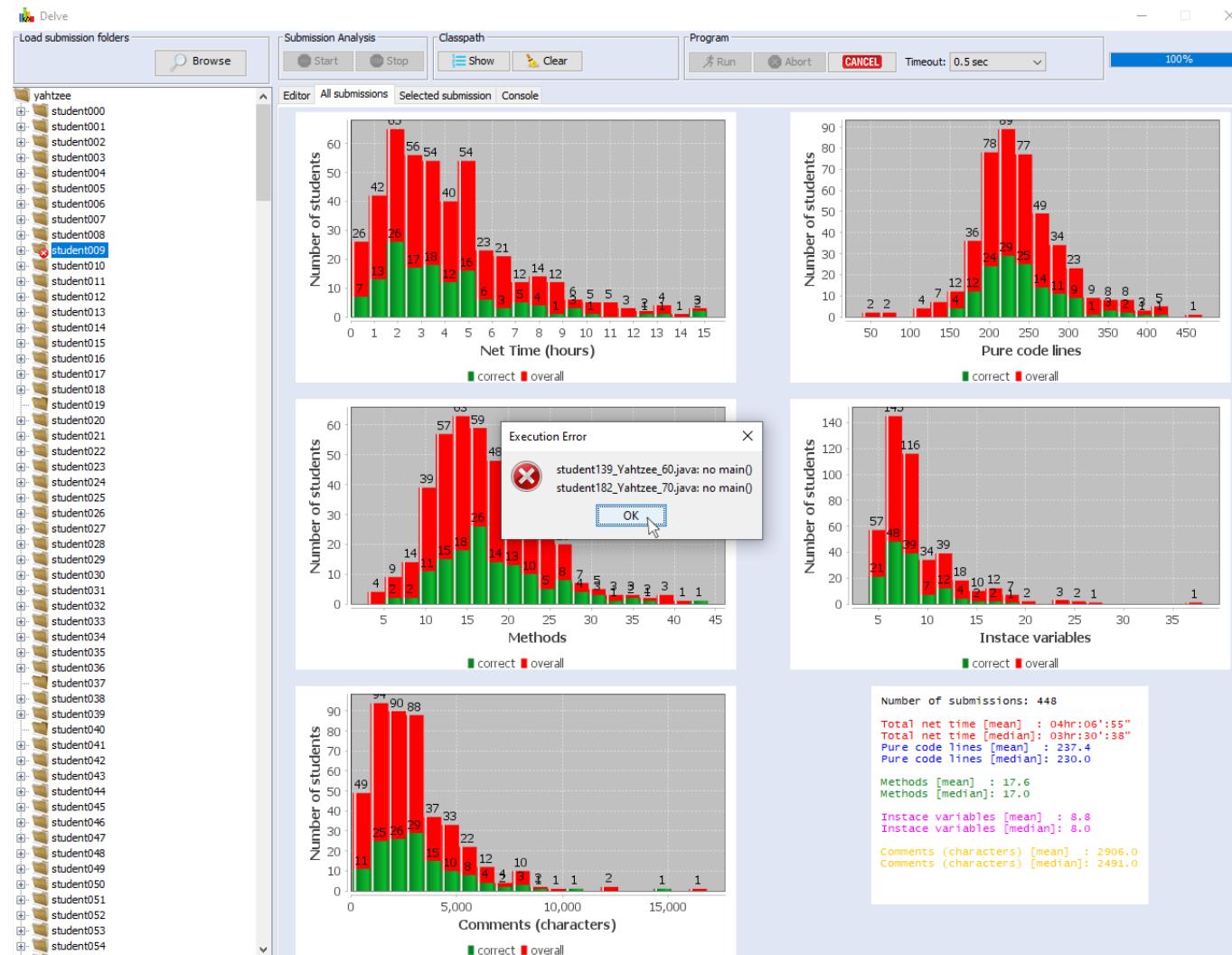


Figure 103: Only programs with `public static void main(String[])` can be analyzed. Delve lists the files that do not have `main()`.

In the main panel, Delve plots the same histograms (in red) as before (**Figure 6**), but now it plots also (in green) the number of correct submissions for each bin. These plots provide a useful insight on things that work well and not so well.

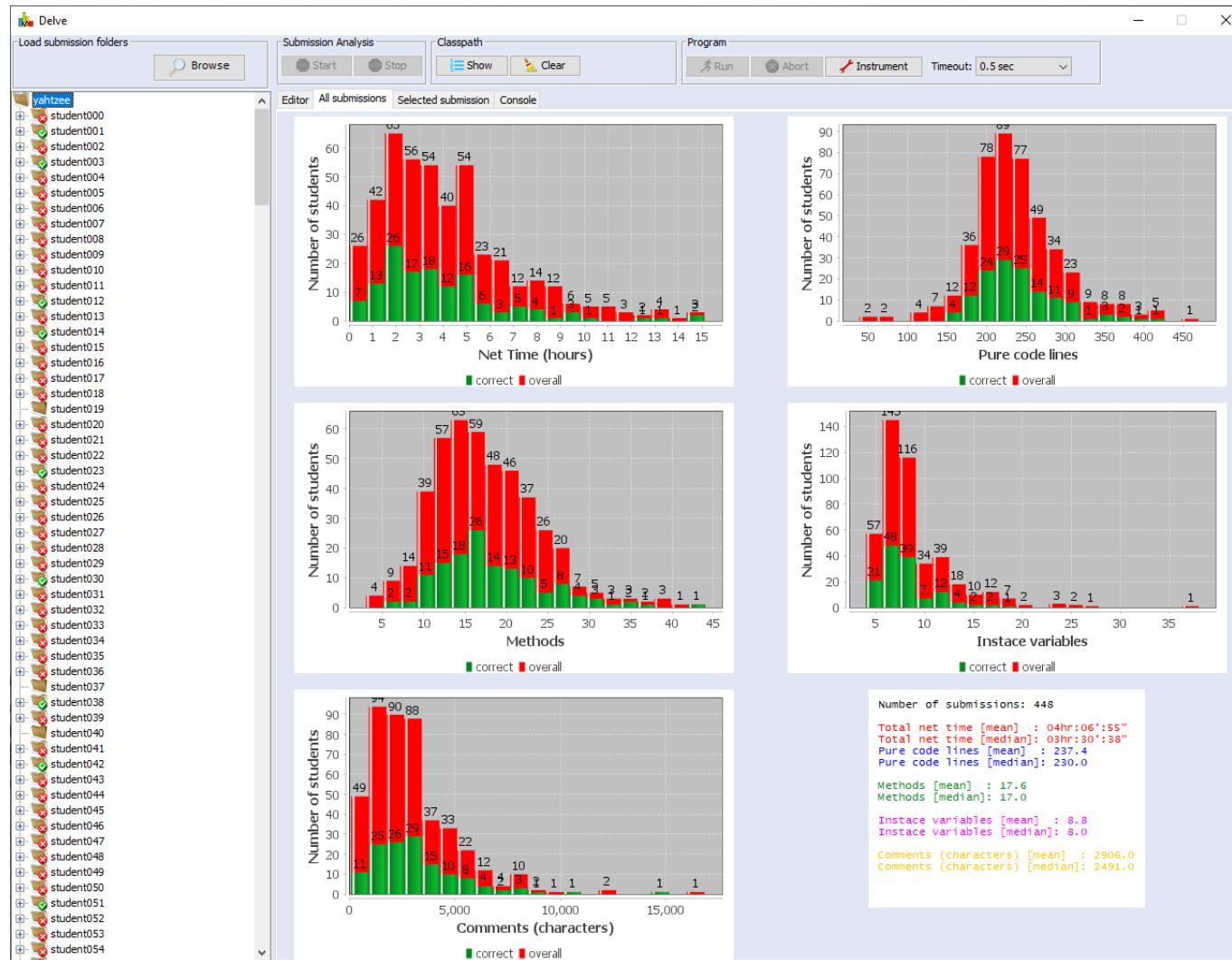


Figure 104: Histograms for correct submissions in contrast to the histograms for all submissions provide useful insight.

At this point Delve has graded every submission and the individual results are shown on the left, in the file tree. The red mark on the submission folder means that the submission was incorrect as opposed to the green mark. Note that there is no degree of correctness. Green means “100% correct” and red means “not 100% correct”. The snapshot that is graded is also marked in green or red once we expand any given folder.

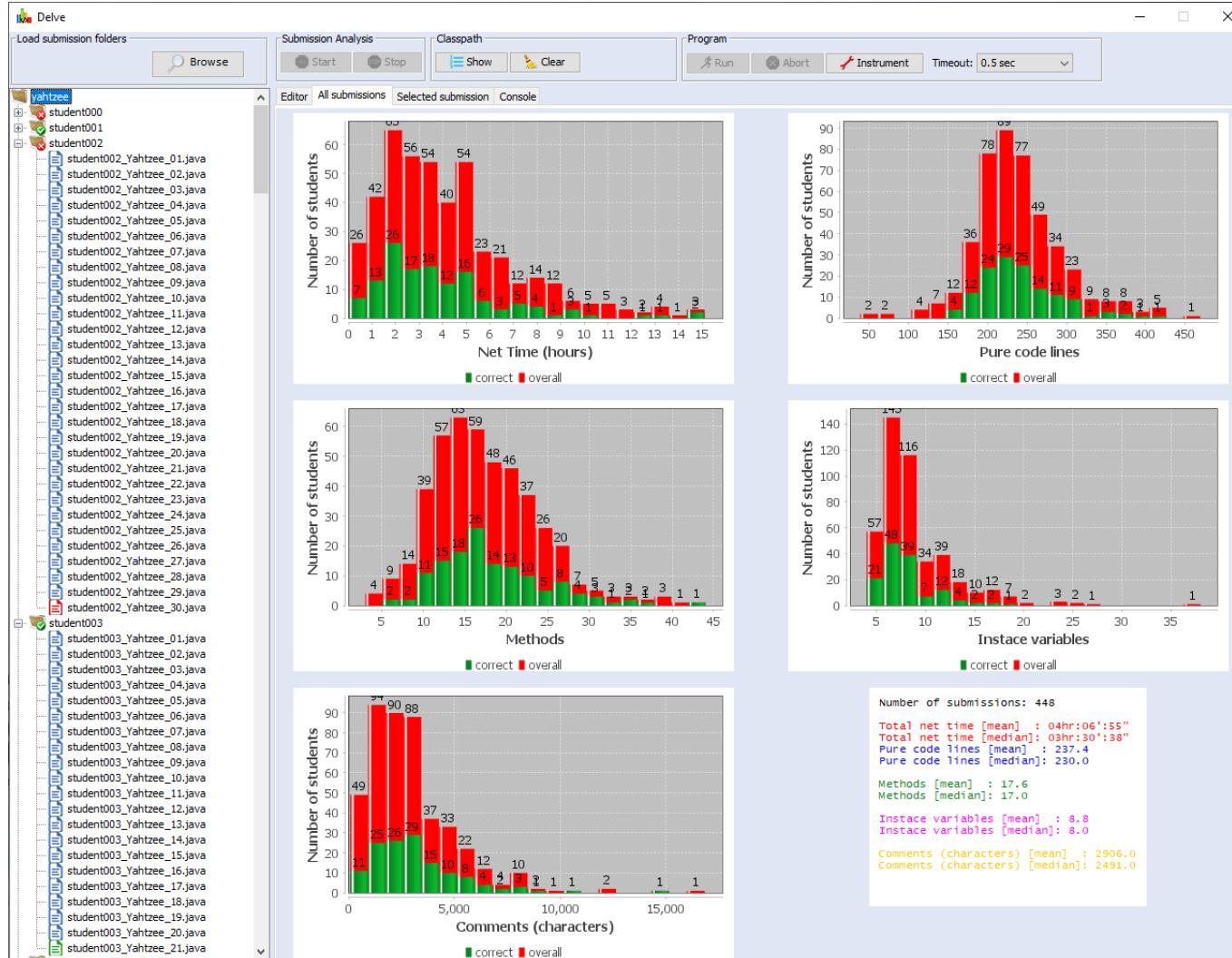


Figure 105: Delve's auto-grading notation.