Quiz-Tac-Toe

A sample dossier for IB Computer Science, Standard Level

Written by Dave Mulkey, Germany, July 2008

Comments

The intention of this sample dossier is not to present a "perfect" product, but to provide a basic model, showing some good ideas and techniques that students could use to achieve a good mark on their Internal Assessment. I have not marked this dossier (I find it difficult to mark my own work objectively), but I'm pretty sure this work would receive a grade 6 or 7. As far as I know, there is nothing "missing" in this sample, and I will encourage my students to use this as a guide to ensure that their work is complete. The dossier took 40-50 hours to complete (didn't time it exactly). Despite my best efforts, it didn't quite fit into 100 pages.

The author is an IB Computer Science teacher at Frankfurt International School. I have been teaching IB Comp Sci for over 20 years, as well as moderating IB Dossiers for about half that time. The techniques shown here are similar to many good dossiers that I have moderated in the past, as well as dossiers that my students have submitted. I believe it follows the requirements and intentions of the IA Criteria. The only "non-standard" part is the use of "user-stories" as the primary vehicle for systems analysis. This is a concept borrowed from Extreme Programming. I find this an easy methodology for my students to understand. Certainly UML, questionnaires and other techniques are equally valid.

For further guidance, I recommend studying the graded examples in the Teacher Support Materials, as well as carefully reading and following the Assessment Criteria in the Subject Guide.

Students are discouraged from copying any of the Java code in the program listing, and are reminded that any copied code from any source must be properly attributed and may not be used to satisfy mastery factors.

My students have considerable success writing GUI applications using EasyApp. Anyone interested in using EasyApp to build a GUI application can download a copy at: http://ibcomp.fis.edu/Java/EasyApp.html

I welcome questions and suggestions, especially if you find mistakes (likely). Please address questions and comments to: Dave_Mulkey@fis.edu

Table Of Contents

Criterion A1 – Analyzing the Problem

Describing the Problem

Our school has lots of computers, as well as a SmartBoard in every classroom. The teachers keep trying to find clever ways to use the computers to improve classroom instruction. They look for "educational software", like games and quizzes and videos and web-sites.

Ms Fizz is a math teacher at our school. She asked our Computer Science teacher whether the IB Computer Science students could create software for her to use on her SmartBoard. So I went to talk to Ms Fizz about some ideas. She said she wasn't interested in videos or that sort of thing, and that most web-sites were pretty useless. But her students need lots of drill and practice, and maybe she would be interested in programs that let the students do more drill and practice at home. She had seen lots of educational software, but mostly it was either for social studies – like multiple choice quizzes – or it was for **doing** math, like drawing graphs. She wanted a math **quiz** program that the students could use on their own, or maybe she could run it on the SmartBoard during class.

After a bit of discussion, and some web-surfing, Ms Fizz mentioned an old TV show called "Hollywood Squares" that was sort of a combination of Tic-Tac-Toe and Trivial Pursuit. She described it like this:

> There is a Tic-Tac-Toe board. When the X-player chooses a square, they have to answer a question correctly in order to get their X on that square. Then it's O's turn. If the player answers incorrectly, they don't get the square. So you could end up with a board looking like this if O missed a question and X answered correctly 3 times:

	O	
X	X	X

Ms Fizz had the idea that the questions could be math problems. Then 2 students could play against each other, or she could split the class into teams that play against each other using the SmartBoard.

User Stories

These user stories summarize some of the discussion, especially where it leads to ideas for input, processing, and output.

Homework is Boring and Ineffective My students say homework is boring, so I'd like a game that is fast and fun and will motivate my students to practice. It needs to have quick and easy questions, and let the students answer them quickly and with minimal effort. **Input:** problems and student answers **Processing:** check whether student's answer matches the correct answer **Output** : an X or O in the Tic-Tac-Toe board	***Clever Students Should be Rewarded*** Often the homework assignments are too easy for the more capable students. They finish quickly and get all correct answers. I'd like them to have a reason to do more problems. Also the challenge of playing against another student should provide extra motivation. **Input** : problems and answers from 2 students **Processing:** the game determines a winner **Output :** game board and notification of winner
Students Should Choose their Topic Students should be able to choose more practice on topics that they haven't mastered - especially when preparing for a test. So there needs to be a way that the students can choose the topic. **Input :** name of topic **Processing :** game loads a set of questions from the chosen topic area **Output** : game presents questions from chosen topic area and tells whether answers are correct	***Students Need a Clear Challenge*** Using a known game, like Tic-Tac-Toe, provides a clear goal and an appropriate challenge. Students can measure their own achievement. **Input :** students choose their topic and answer questions **Processing :** computer correctly enforces the standard Tic-Tac-Toe rules and decides whether answers were correct and who won **Output :** game board, responding to answers, checking who wins
Various Types of Problems I give small 10 minute quizzes about twice a week. I'd like to use the same type of questions as I have on these quizzes. Those are short problems with either multiple choice answers, or quite simple numerical answers, or single word answers (like fill-in-the-blank). **Input** : copy existing quiz problems into the computer to be used in the game **Processing** : problems are organized according to topic **Output :** problems are saved in data files on the hard-disk	***Typing Special Symbols*** Math problems often contain special symbols like square-root signs, squared and cubed exponents, and fractions. I'd like to be able to type the questions in with a normal keyboard, but the problems should appear with the correct math symbols **Input :** typing on a normal keyboard, using short-cuts for special symbols **Processing** : translate shortcut abbreviations to correct math symbols **Output :** store problems in data files with the correct math symbols

Collecting Information and Sample Data

Since the game is for a math class, the questions need to be math problems (or questions about math vocabulary). We looked at some multiple-choice quizzes, some written tests, and some of the homework problems in a math text-book. Here are some sample questions:

Quiz	***Text***	***Homework***
(1) If the discriminant of a parabola is -4, how many roots does it have? (a) 0 (b) 1 (c) 2 (d) 3 (2) What is the sum of the roots of $x^2 - 4x + 2 = 0$? (a) 2 (b) 4 (c) 0 (d) 1 (3) Where is the vertex of the parabola $y = x^2 - 4x + 2$?	(1) Draw the graph of $y = \frac{x^2 - 4x + 3}{x - 3}$ (2) Find the intersection of $y = 2x + 3$ and $y = x^2 - 4$. Show your work. (3) Explain how the **discriminant** is used.	(1) Graph each parabola: (a) $y = x^2 - 4x + 3$ (b) $y = x^2 - 4x + 4$ (c) $y = x^2 - 4x + 5$ (2) For each graph in #1, state the number of roots. (3) Write the formula of a parabola through (-1,4) , (0,1) and (1,0).

Ms Fizz was especially interested that the game be “fast” and “easy” and “fun”. She felt that her students would be more likely to use the game if the questions were quick and easy to answer, if typing the answers was easy, and if the game ended quickly so the students could play again.

It was apparent that not all the sample questions would be suitable for use in a computerized quiz game. For example, asking the players to draw a graph is probably out of the question.

- The questions should be short and have quick answers
- The students cannot be required to draw graphs or other pictures
- The answers should not require the students to write complex formulas
- Some special symbols, like powers, should be used in the problems, as computer symbols like x^2 are not understandable to many students
- “Show your work” is not sensible

Looking at the sample questions, it seems the quiz questions were most suitable. Multiple-choice is particularly good, but simple numerical answers would also be acceptable.

Systematic Analysis of Input, Output and Processing , Possible Difficulties and Advantages

The appearance of the Tic-Tac-Toe board is pretty obvious, but the input and output of math formulas and symbols might cause some difficulties. And creating problems might be tricky.

Outputting Questions	**Inputting Answers**	**Processing**
Math formulas often contain strange symbols that cannot be typed on a keyboard. These can be created using ASCII or Unicode, but probably won't display "nicely". Fractions present a large problem – might require graphics mode for printing. Surds are also difficult to print. Simple exponents like squared and cubed can be printed using simple characters, but other exponents need to be "raised" above the x, perhaps requiring graphics mode.	This has the same problems as outputting the questions, but is even more difficult because the users want to type their answers on the keyboard. They won't want to look up ASCII codes for special symbols. This might limit the types of question and answers that are possible. And typing cannot be done in graphics mode, at least not easily. What if the correct answer is a surd or a fraction like 1/3? A decimal approximation won't be good enough.	The questions need to come from somewhere. They cannot be hard-coded inside the program. They can either be stored in data-files or created by clever methods. For example, to produce a random parabola problem, the program can choose random numbers for the coefficients A, B, and C. But Ms Fizz says A, B, and C need to "fit together" - they can't just be chosen at random. Comparing users answers to correct answers can be difficult, as the user might type 0.3333 instead of 1/3.

Ms Fizz agreed to produce a list of sample problems to discuss. I warned her that some problems might not be suitable, as the printing and typing might be too difficult.

After some discussion, we agreed that multiple-choice questions were a good strategy, because they are quicker to answer and require very little typing. But I couldn't imagine how to program the computer to randomly generate 3 incorrect answers along with the right answer. So we agreed that the teacher would need to write the questions ahead of time rather than the computer "generating" the problems. True/false questions are possible, as well as simple numbers like whole numbers.

Possible Advantages of a Computer System

Since some math formulas and problems might be excluded from the game, it's important to consider the advantages of using a computerized game.

- Students can practice as much as they wish, without needing the teacher around
- The computer gives immediate feedback about whether answers are correct
- Working with another student and playing a game should be fun and motivating, and encourage students to spend more time practicing than they would otherwise
- The teacher can use the game on the SmartBoard during class as a motivational tool

Criterion A2 (preliminary) – Criteria for Success

Considering the analysis, the following goals seem sensible. The reasons in the chart were mentioned in the analysis above.

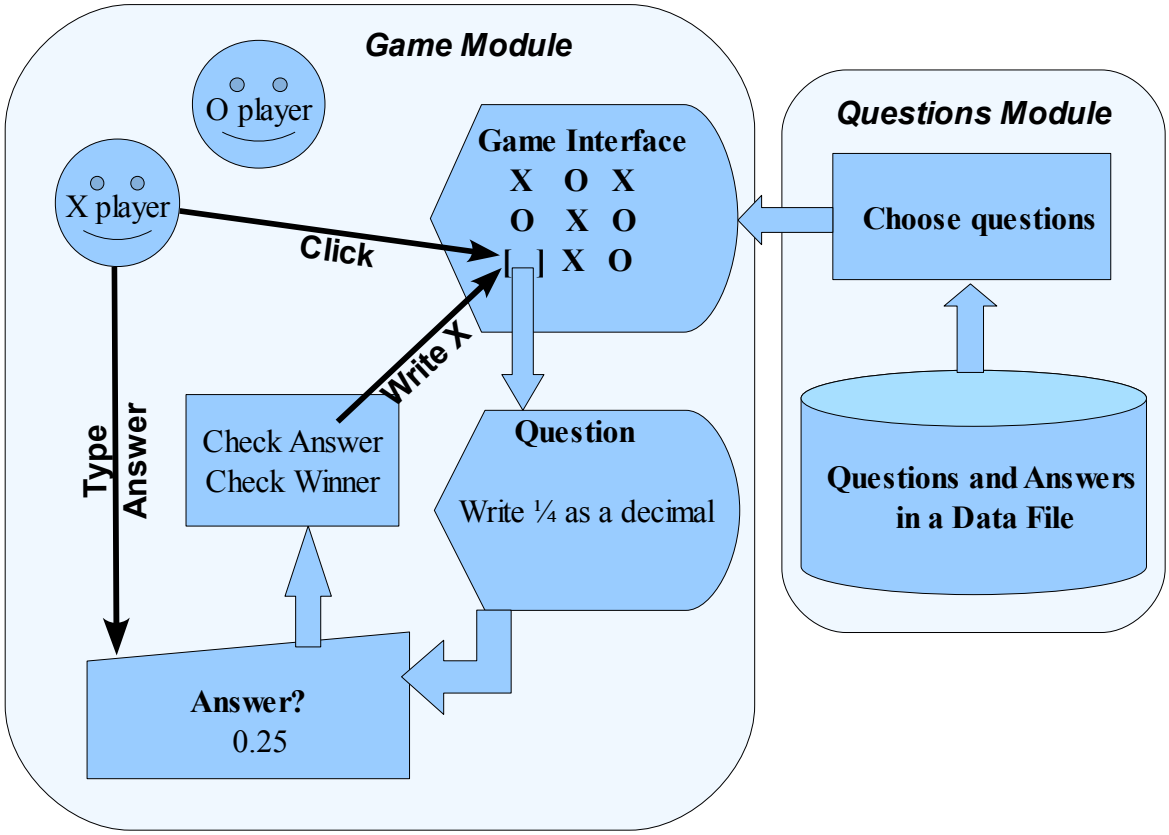
Goal	***Reason(s)***	***Limitation(s)***
Game plays Tic-Tac-Toe , using correct Tic-Tac-Toe rules	It's fun	3x3 board only
Each Tic-Tac-Toe square asks a question	That's the rules of the game	Questions will be selected from a list – not auto-generated
Questions should contain appropriate math content	Students should be learning and practicing their math	Some text-book problems are not suitable – e.g. complex formulas or graphs
Game should be quick, easy and satisfying, including a simple and clear user-interface	Motivate students to practice , more fun = more practice	Don't require complex input (fractions, complex formulas)
Teacher can create and save problems	Auto-generating problems is too difficult to program	No complex formulas Pictures?
Some special symbols can be used in the questions – squares, cubes, simple square-roots	Math without special symbols is difficult to read and understand	Many special symbols will not be implemented, especially fractions and complex surds like the discriminant in the quadratic formula

These were the preliminary goals, used to create the initial design and prototype. After discussing the prototype with the user, the goals were revised. The complete goals are presented below, following the prototype.

Criterion A3 – Prototype Solution

Initial Design

The initial design includes a data-file containing questions and the game module for playing the game.



The finished program will need a large set of questions, but the prototype will have hard-coded questions for test purposes. So the Questions Module will not appear in the prototype – it is only simulated.

Prototype

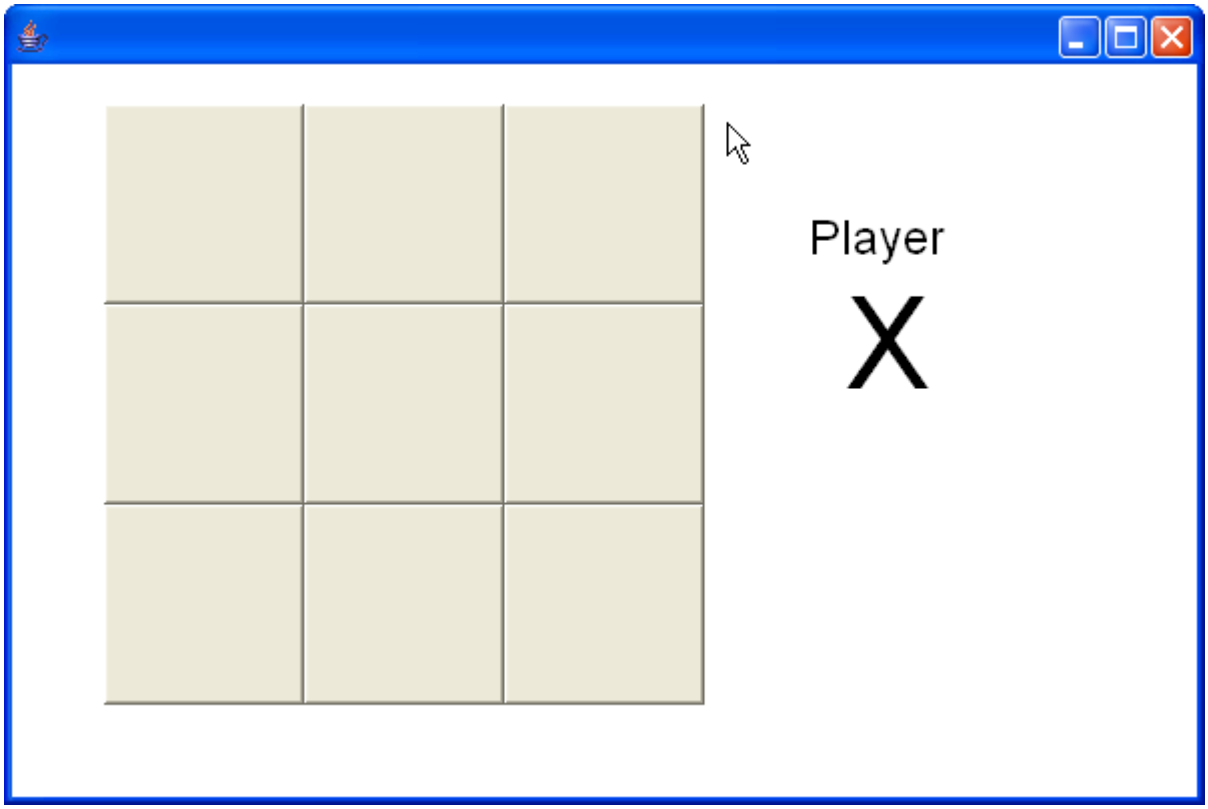
We sketched out some ideas on paper, but Ms Fizz really wanted to see "how it's gonna look." We decided a functional prototype - as a short Java program - would be best. So I wrote a short Java program that let Ms Fizz click on the Tic-Tac-Toe board, answer questions, and win the game. Although it had the same 9 questions all the time, it convinced her that the students would be able to use the game easily.

The following pages show some sample screen-shots of the running prototype. We ran the prototype more often than shown in the screen-shots - enough to convince Ms Fizz that it would be worthwhile to proceed with the project.

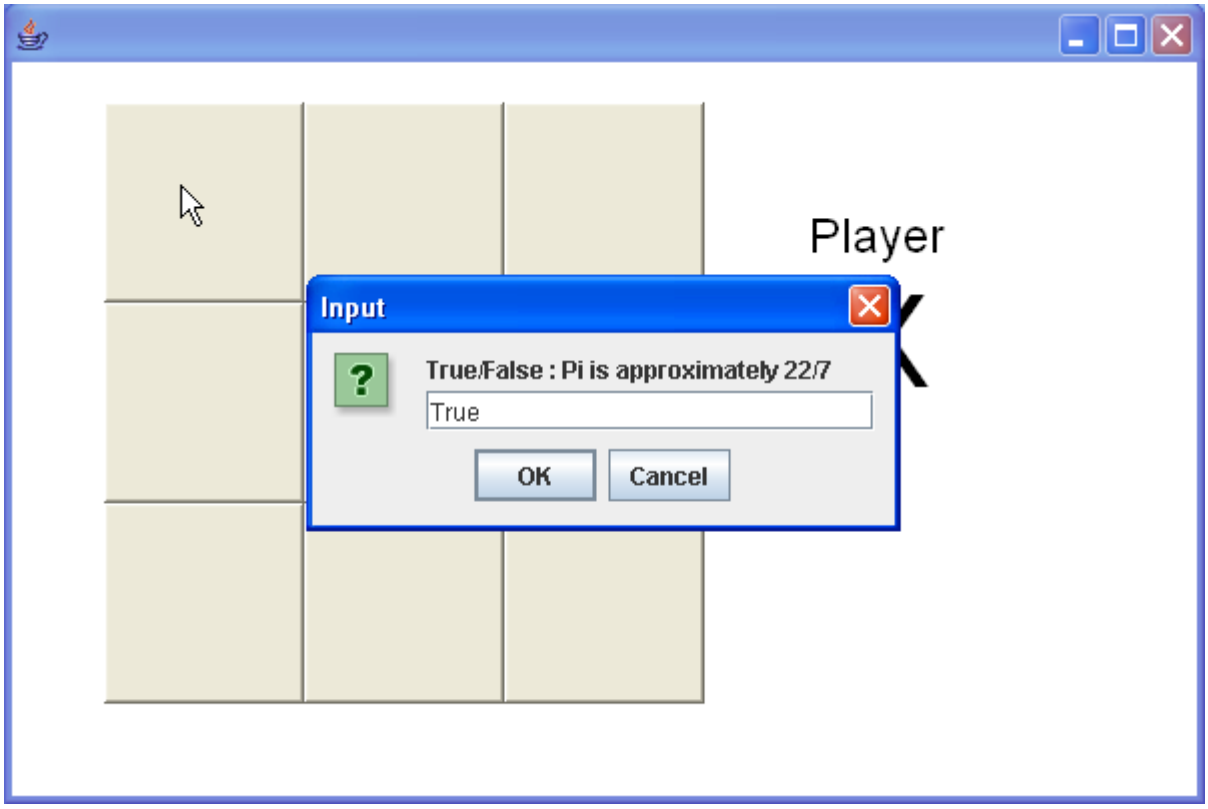
After trying out the prototype, Ms Fizz had suggestions for improvement. Her ideas are presented at the end of the screen-shots.

Prototype – Sample Screens ** the code listing is at the end of Stage A **

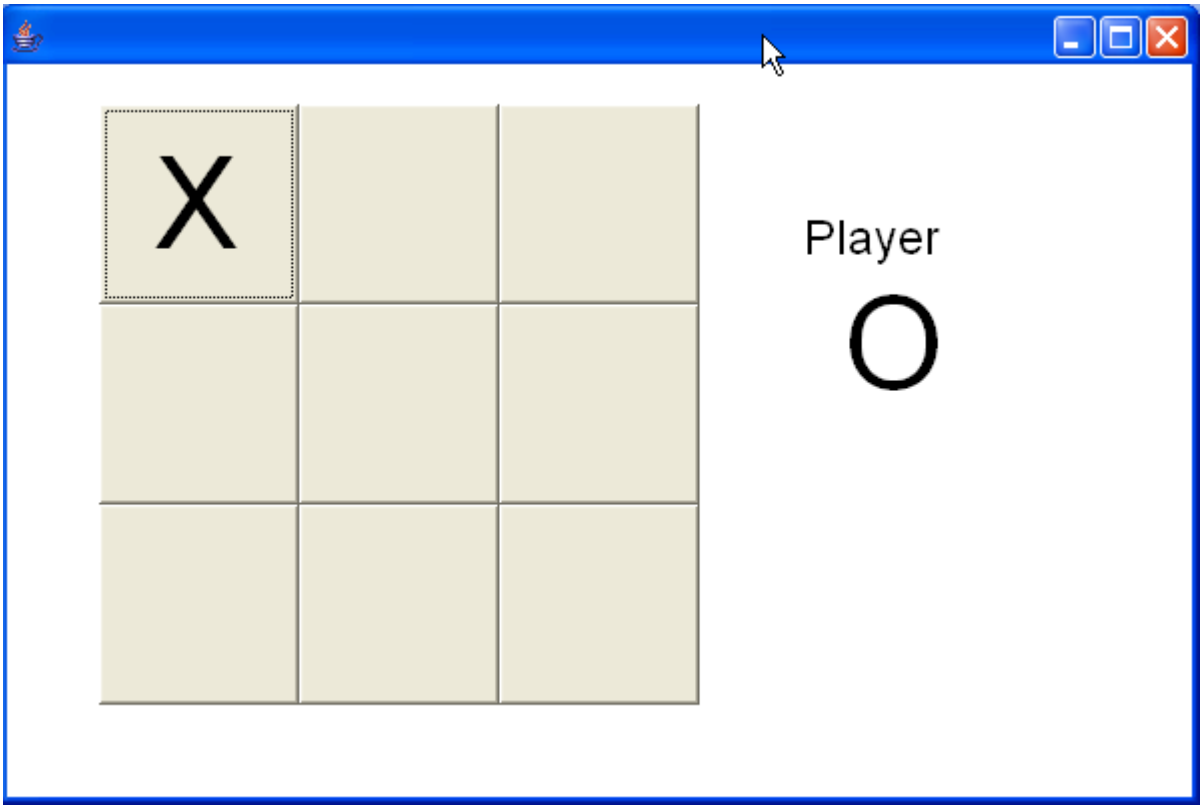
The quiz starts, X plays first.



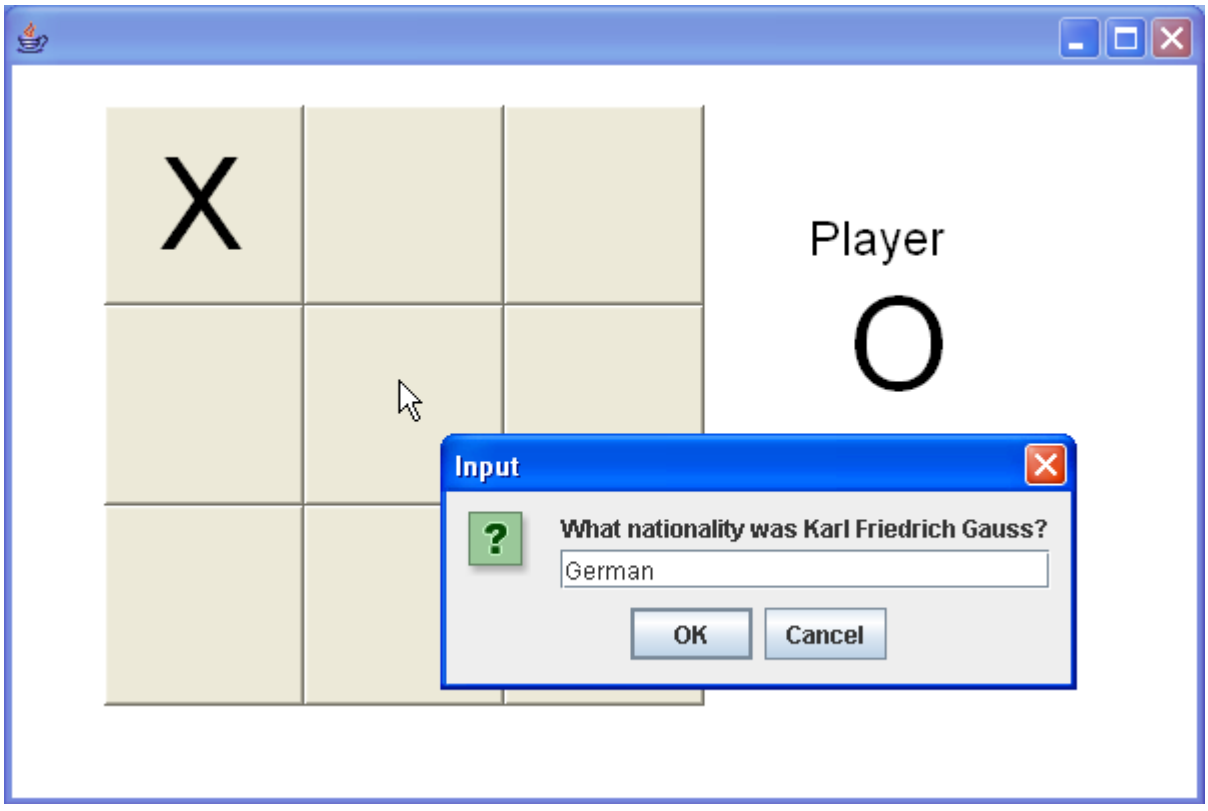
X clicks on the top-left corner and answers the question.



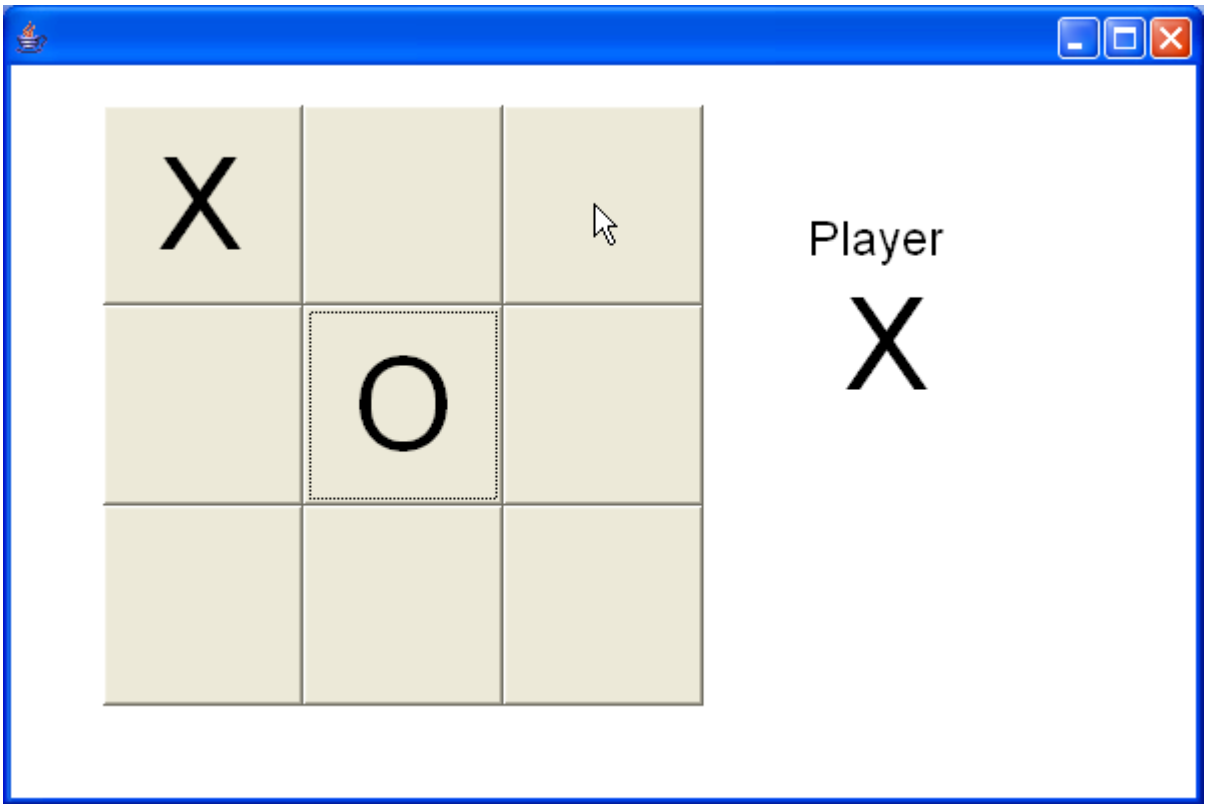
X had the correct answer, so gets the square. Now it is O's turn.



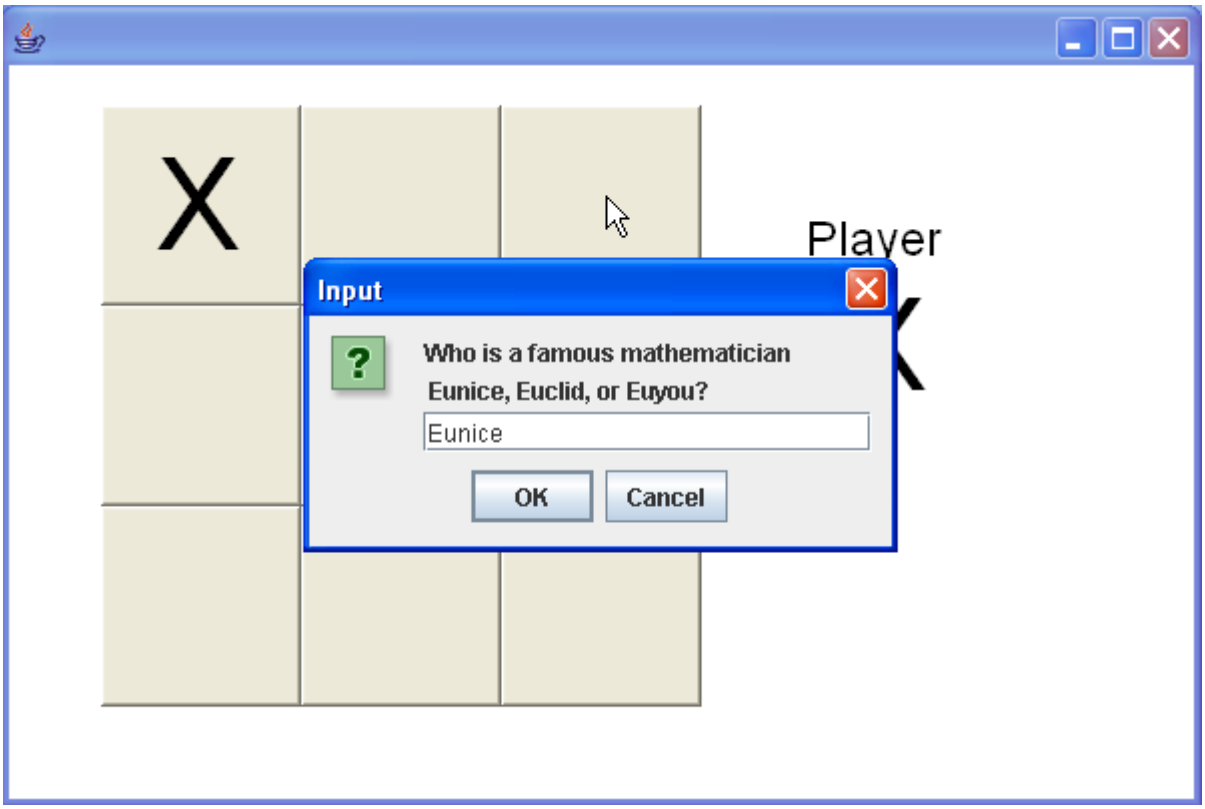
O chooses the middle square and answers the question.



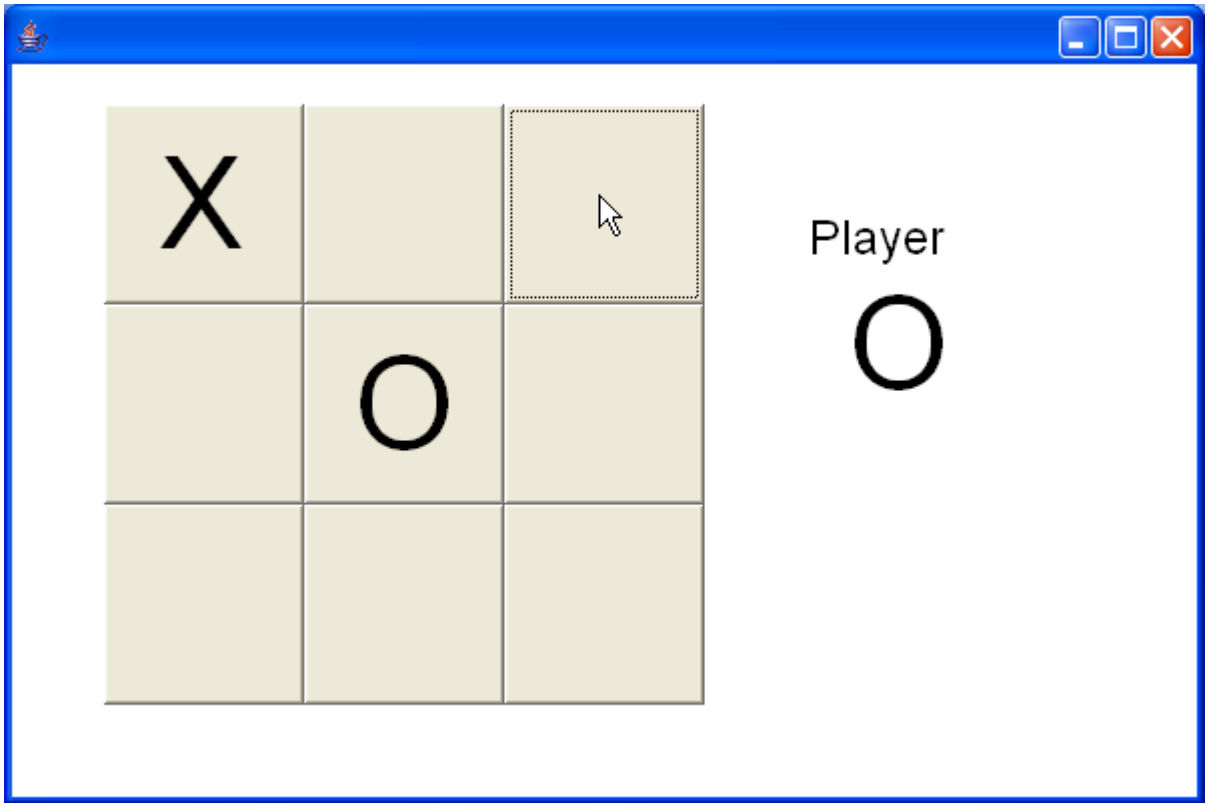
O answers correctly and gets the square.



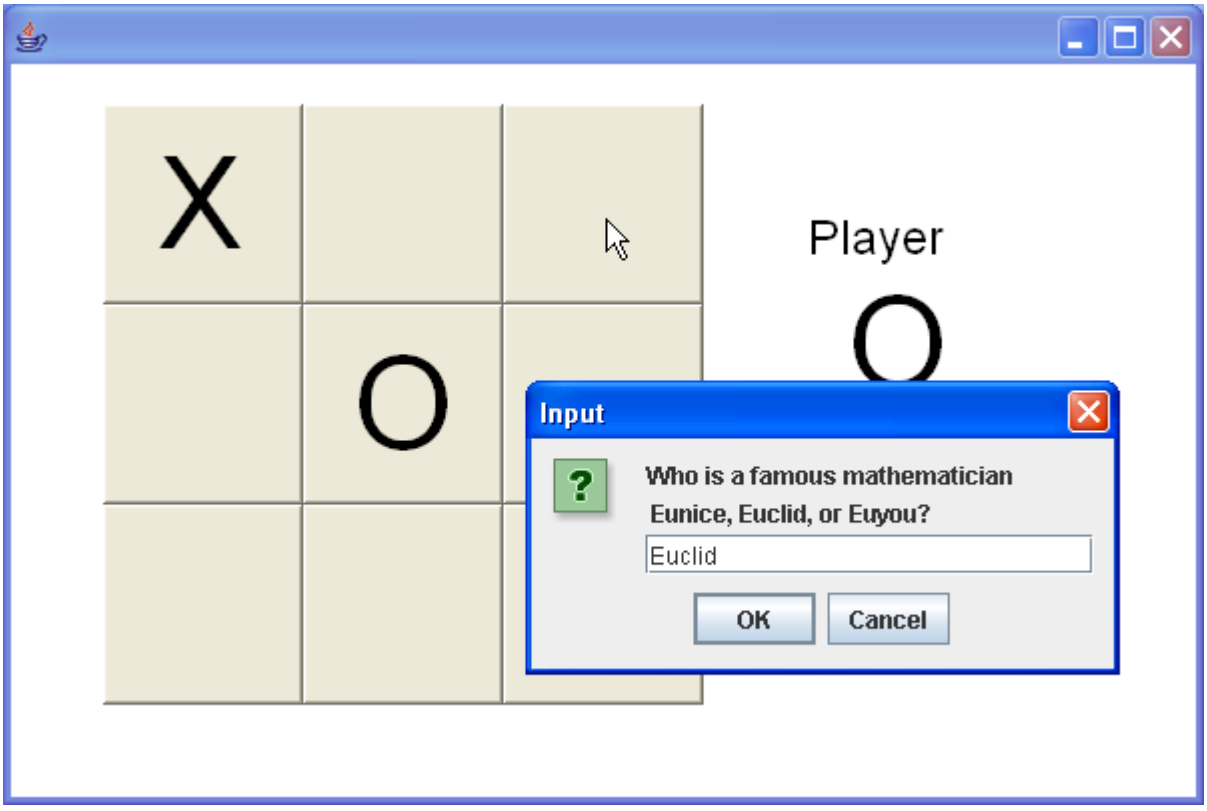
X tries for the top-right corner, but answers the question incorrectly.



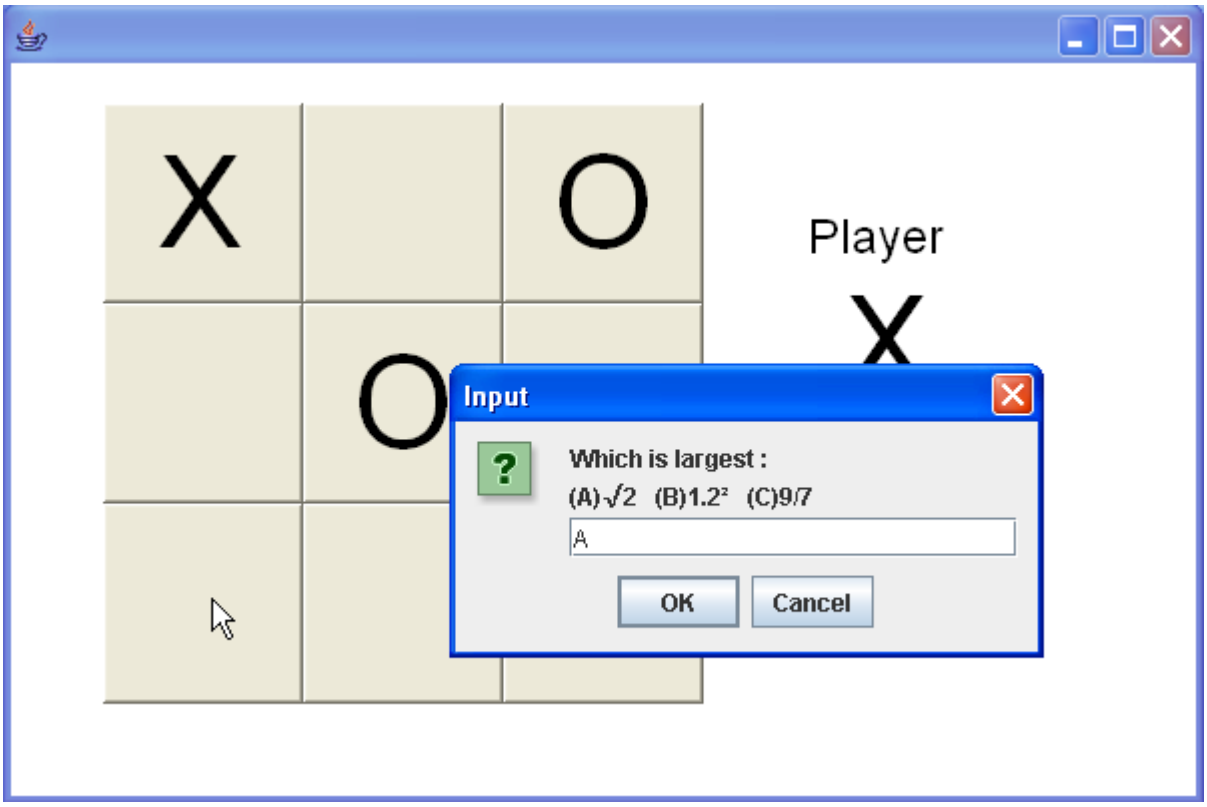
X answered incorrectly and thus didn't get the square. Now it's O's turn.



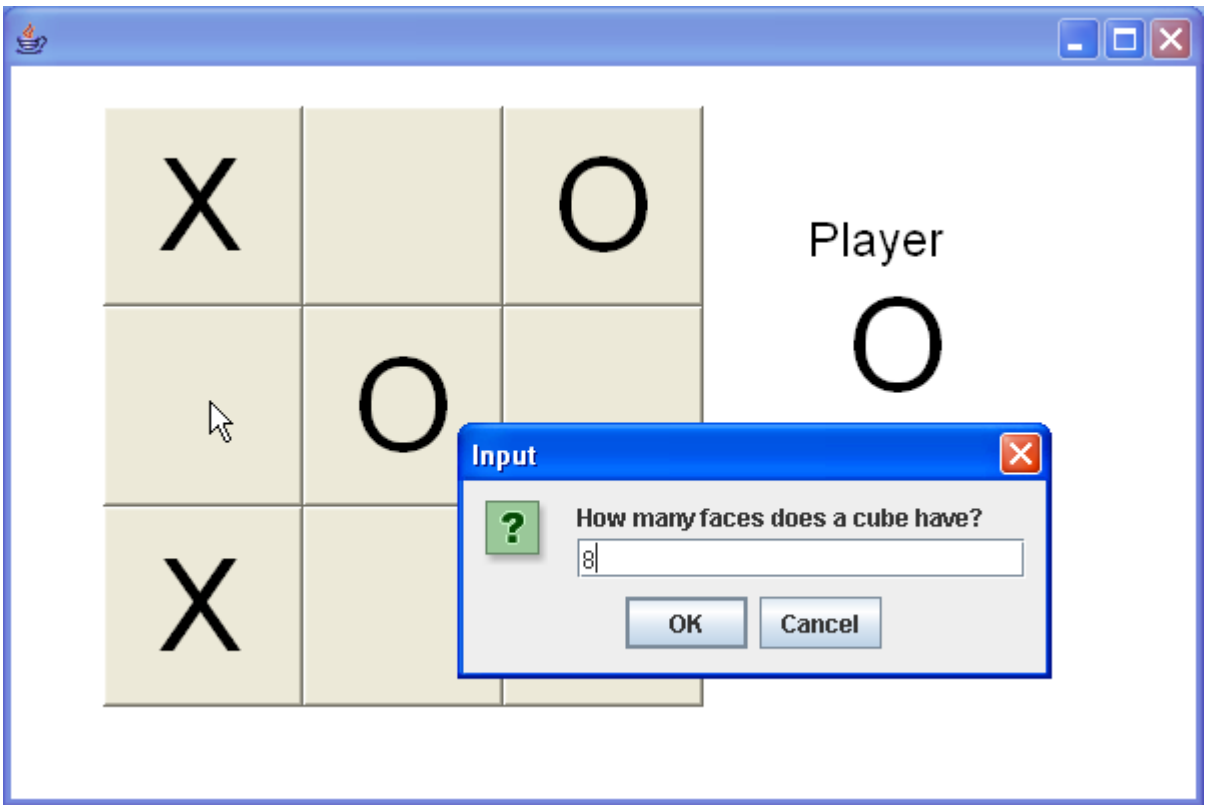
O tries the top-right square again, because the question was easy.



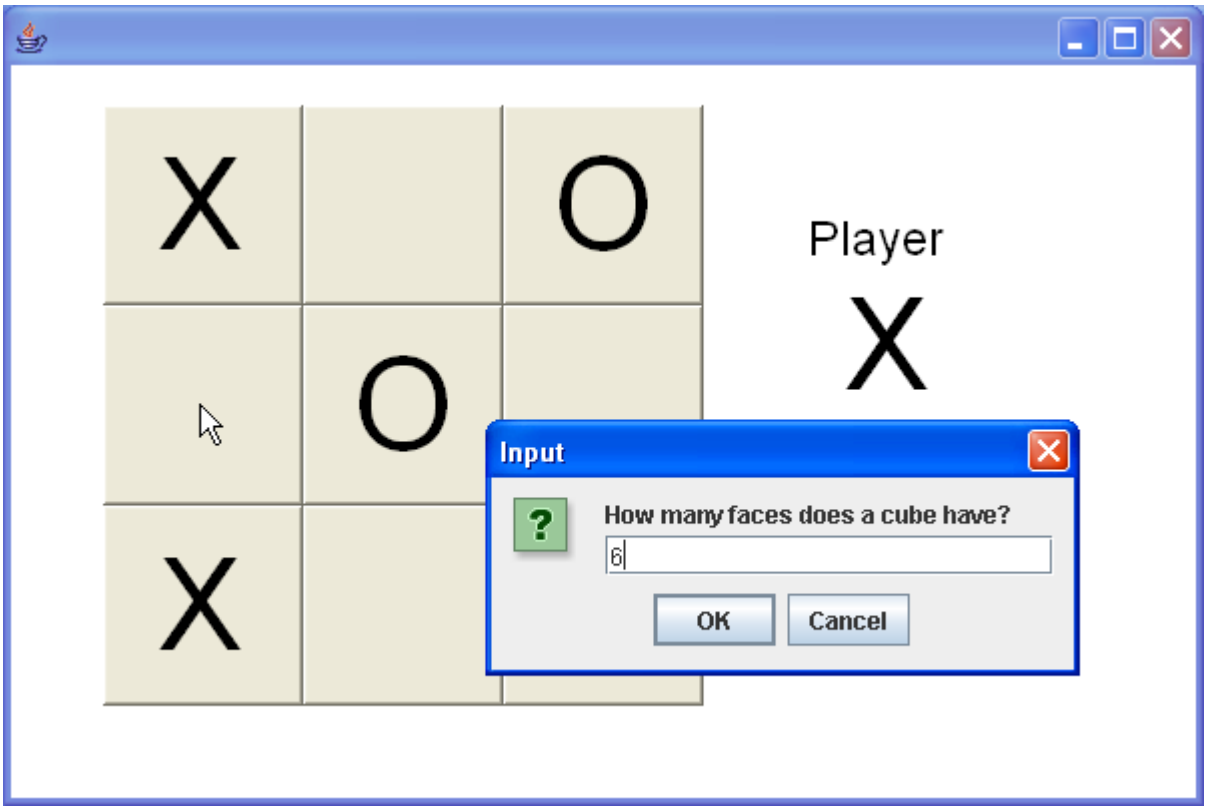
O answered correctly and got the square. Now it's X's turn. X goes for the block in the bottom left.



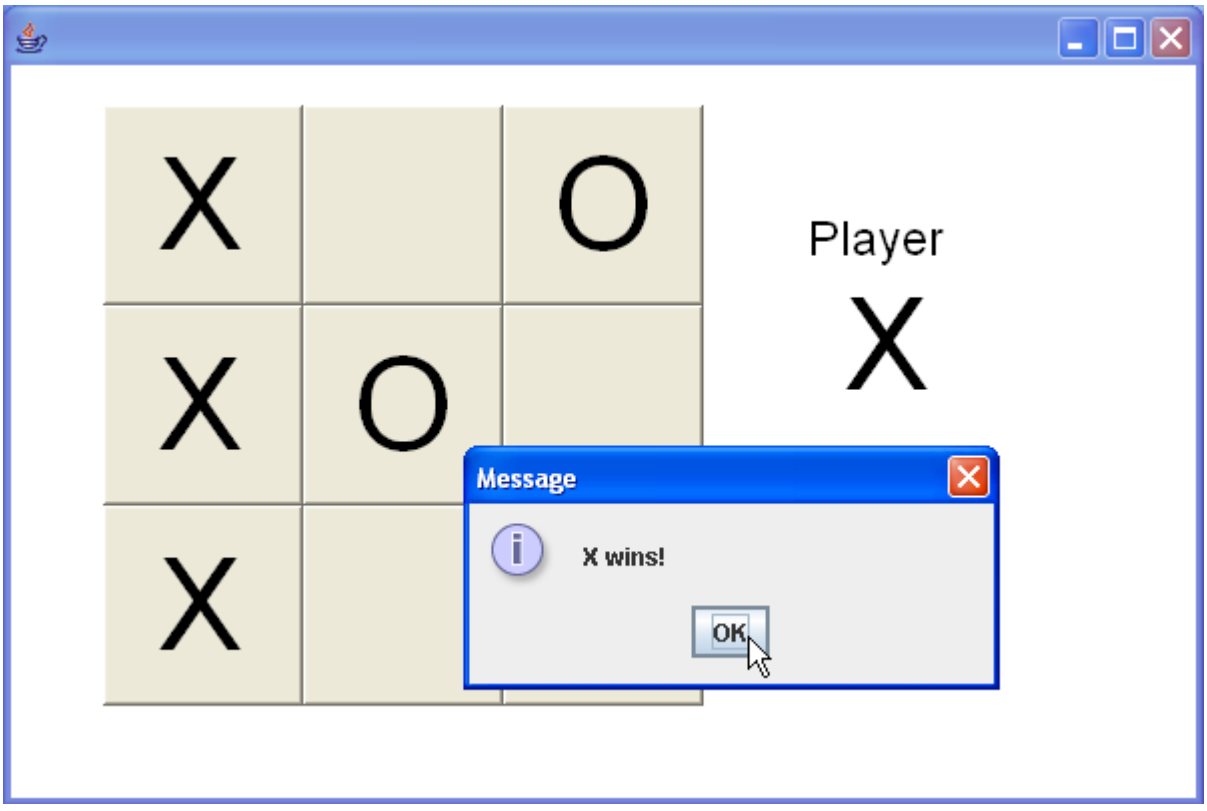
X had the right answer and gets the square. Now O goes for the block in the middle-left.



O answered incorrectly. Now X goes for the win.



X answered correctly and wins the game!



User Feedback about the Prototype

We ran the prototype several times. Ms Fizz wanted to check whether the prototype worked for wins along the diagonals – it worked okay. Ms Fizz had many questions – below is an excerpt. The discussion was longer (several sessions actually), but these outline the issues affecting goals.

User Questions, Ideas and Wishes	***Answers***
Does it correctly recognize wins in all directions (including diagonals)?	Yes (we tested all the directions)
What happens if there is a tie – the board is full but no winner?	The prototype doesn't recognize a tie, but the final program will stop and say "It's a tie". Is that okay?
If there is a tie, I guess the player with more squares should win.	Okay, we can do that.
This always has the same 9 questions. I think it should have different questions (selected randomly) each time you run it.	Okay, we can do that in the final program. Do you want the computer to invent the questions?
Can the computer invent the questions?	Not really, unless they are simple math calculations.
No, I want more flexibility in the questions. Some have words or formulas as answers, others are numbers.	Okay, then we need to make a module that lets you type in questions, then save them in a file, and the program loads the questions from the file.
I don't understand about saving in files. Is that hard?	No, we'll make a real simple interface and the saving and choosing random questions is all automatic.
But I want the questions scrambled up, so the players don't always know which question is hiding in each square.	If the questions are selected randomly, they'll also be arranged randomly in the squares.
I think in the old Hollywood Squares show, a wrong answer automatically gave the square to the other player. Is that possible?	Yes, it's possible – is that what you want?
I don't know – I'll think about it. Can I type real math formulas, with exponents and fractions?	No, I don't know how to program that. I guess you can type x-squared if you know the ASCII code for the squared sign, but no fractions.
Okay, I guess I can just type 3/4 , like that. How about pictures – can I put in pictures?	I'll think about that. I don't know how to do it, but I'll ask my teacher if it's possible.
How many questions can I have altogether?	There is no limit. But it would be easier to write the program if we set some kind of limit – like 1000 questions maximum. Is that okay?
Is that a 1000 for EVERYTHING? I have lots of different classes, different each year.	We can make separate files for different subjects. What subjects do you need?
Let's see... simple algebra, advanced algebra, geometry, statistics, a few more. Maybe a 100 for each. How many can I have?	If you make a list of subjects and/or topics, we can make a set of questions for each. You can have any limit you like, but it's easier if the limit is fixed.
Okay, I'll think about it and make a list.	Can you make a list of topics and 10 sample questions for each? You can add more later.
Will this run on the school's web-site?	No, but it can run from a server on our LAN.

Revised Criteria for Success after User Feedback

As a result of the user feedback, the original goals (Criteria for Success) were revised. Changes and additions are marked with ** asterisks.

Criterion A2 (Final) – Criteria for Success

Goal	***Reason(s)***	***Limitation(s)***
Game plays Tic-Tac-Toe , using correct Tic-Tac-Toe rules	It's fun	3x3 board only
Each Tic-Tac-Toe square asks a question	That's the rules of the game	Questions will be selected from a list – not auto-generated
** Questions will be selected randomly and scrambled	Students should not know which questions are hiding in each box	
Questions should contain appropriate math content	Students should be learning and practicing their math	Some text-book problems are not suitable – e.g. complex formulas or graphs
Game should be quick, easy and satisfying, including a simple and clear user-interface	Motivate students to practice , more fun = more practice	Don't require complex input (fractions, complex formulas)
Teacher can create and save problems	Auto-generating problems is too difficult to program	No complex formulas Pictures?
Some special symbols can be used in the questions – squares, cubes, simple square-roots	Math without special symbols is difficult to read and understand	Many special symbols will not be implemented, especially fractions and complex surds like the discriminant in the quadratic formula
** Teacher module for typing and saving questions and answers	Teacher wants to create specific questions	Only a few special symbols Text only - no pictures
** Questions are saved into various files according to topic. Teacher should be able to add more topics (files) later	Teacher has various classes and topics	There will be a limit of 1000 questions per file
** Questions in data-files can be added, changed and deleted later	Teacher may need to make changes and corrections	This will not be drag-and-drop, but will function in text-mode after a simple search
** It should be easy to copy a problem, change a few numbers and then save as a new problem	Teacher wants to make several similar questions with slightly different numbers	It will be done in text-mode, not drag and drop

Prototype Listing

```
import java.awt.*;

public class ProtoSquares extends EasyApp {
   public static void main(String[] args)
   {  new ProtoSquares(); }

   Button[][] squares = new Button[3][3];
   Label lTurn = addLabel("Player",400,100,100,35,this);
   Label turn = addLabel("X",420,120,100,100,this);

   String[] questions =
   { "True/False : Pi is approximately 22/7",
    "What is the square root of 0.25?",
    "Who is a famous mathematician \n Eunice, Euclid, or Euyou?",
    "How many faces does a cube have?",
    "What nationality was Karl Friedrich Gauss?",
    "How many feet are there in one mile?",
    "Which is largest : \n(A)\u221a2   (B)1.2²   (C)9/7  ",
    "What is the root of  x² - 8x + 16 ?",
    "What does 0! equal?"
   };

   String[] answers =
   {     "True","0.5","Euclid","6","German","5280","A","4","1"    };

   int player = 1;

   public ProtoSquares()
   { Font thefont = new Font("Arial",0,64);
     turn.setFont(thefont);
     lTurn.setFont(new Font("Arial",0,24));
     for (int row = 0; row < 3; row = row+1)
     {
        for (int col = 0; col < 3; col = col + 1)
        {
           int x = 50 + 100*col;
           int y = 50 + 100*row;
           squares[row][col] = addButton("",x,y,100,100,this);
           squares[row][col].setFont(thefont);
        }
     }
   }

   public void actions(Object source, String command)
   {  int qnum = -1;
      int rnum = -1;
      int cnum = -1;

      for (int row = 0; row < 3; row = row + 1)
      {  for (int col = 0; col < 3; col = col + 1)
         {
           if (source == squares[row][col])
           {qnum = row*3 + col;
             rnum = row;
             cnum = col;
           }
         }
      }
```

```
      if (qnum >= 0)
      {  if (squares[rnum][cnum].getLabel().equals(""))
         { String guess = input(questions[qnum]);
           if (guess.equalsIgnoreCase(answers[qnum]))
           {
            if (player == 1)
            {  squares[rnum][cnum].setLabel("X"); }
            else
            {  squares[rnum][cnum].setLabel("O"); }
           }
           checkWinner();
           player = -1*player;
           if (player==1)
           {  turn.setText("X"); }
           else
           {  turn.setText("O"); }
         }
         else
         {  output("Choose an empty square");  }
      }
   }

   public void checkWinner()
   {
      for (int row = 0; row < 3; row = row + 1)
      {
         String a = squares[row][0].getLabel();
         String b = squares[row][1].getLabel();
         String c = squares[row][2].getLabel();
         if (!a.equals("") && a.equals(b) && b.equals(c))
         { output(a + " wins!");
           System.exit(0);
         }
      }
      for (int col = 0; col < 3; col = col + 1)
      {
         String a = squares[0][col].getLabel();
         String b = squares[1][col].getLabel();
         String c = squares[2][col].getLabel();
         if (!a.equals("") && a.equals(b) && b.equals(c))
         {  output(a + " wins!");
            System.exit(0);
         }
      }
      String a = squares[0][0].getLabel();
      String b = squares[1][1].getLabel();
      String c = squares[2][2].getLabel();
      if (!a.equals("") && a.equals(b) && b.equals(c))
      {  output(a + " wins!");
         System.exit(0);
      }
      String d = squares[0][2].getLabel();
      String e = squares[1][1].getLabel();
      String f = squares[2][0].getLabel();
       if (!d.equals("") && d.equals(e) && e.equals(f))
       { output(d + " wins!");
         System.exit(0);
       }
   }
}
```

Stage B1 - Data-Structures

The program will contain 3 sections (modules) :

Students' Game Interface , Problem Storage , Teachers' Problem Interface

Problem Storage in Files

All the problems must be stored in **data-files** on a disk drive. The teacher wants separate problem lists for various topics. Each file must contain a record for each problem. Each record contains three fields : Question, Choices and Answer. If the question is longer, the Choices can contain part of the question instead of multiple answers, but then the user must type the exact answer.

A **multiple-choice** problem looks like this:

Question : What do you call a polygon with 8 sides?
Choices : (A) Eightogon (B) Stop Sign (C) Octagon (D) Octogon
Answer : C

A **type-the-exact-answer** question looks like this:

Question : What is the sum of the roots of this quadratic equation?
Choices : x^2 - 4x + 2 = 0
Answer : 4

Text-files have the advantage that it is possible to make small corrections using a text-editor. However, they allow only sequential access. Since a small set of 9 questions must be selected at random from the file, a RandomAcessFile is more efficient as single problems can be accessed directly by their record number. Although RandomAccessFiles are a bit trickier than sequential files, the coding will probably be shorter in the long run with RandomAccessFiles.

The files will each contain 1000 records (fixed file size). Blank records will contain zero-length Strings (blank). Each record will be 200 bytes, so 200 KiloBytes per file.

Ms Fizz agreed to these shortcuts: ^2 for squared, ^3 for cubed, and ^r for a square-root sign √ .
Like this: **Roots of x^2 - 6 = 0 are ^r6 , -^r6 ==>** *Roots of* $x^2-6=0$ *are* $\sqrt{6}, -\sqrt{(6)}$

Each data-file will look something like this file for **Algebra2:**

Record #	Fields and Sizes	Sample Data
0	Question: 90 bytes Choices : 90 bytes Answer : 20 bytes	Which is a root of 2x^2 - 6 = 0 ? (A) ^r2 (B) ^r3 (C) ^r6 (D) ^r12 B
1	Question: 90 bytes Choices : 90 bytes Answer : 20 bytes	Where is the vertex of this parabola? y = x^2 + 4x + 4 (-2,0)
....	....	....
999	Question: 90 bytes Choices : 90 bytes Answer : 20 bytes	“” “” “”

Ms Fizz has asked for the following topic files:

- Numbers (fractions, decimals, percents)
- Shapes (Circles, Rectangles, Triangles, etc)
- Statistics
- Equations (solving linear equations)
- Quadratics
- Graphs
- ... more to be added later ...

We agreed there should also be a feature to create a new topic. That will create an empty file with 1000 records. That way she can add more topic files whenever she wants. Here are 2 sample files showing 5 questions each:

Numbers
Which decimal equals 3/8? (A) 0.38 (B) 0.375 (C) 0.388... B
Which fraction is largest? (A) 3/4 (B) 4/5 (C) 7/8 C
What is 2/3 + 1/12 ? (A) 3/4 (B) 3/15 (C) 9/12 A
Which number is the largest? (A) Billion (B) Million (C) Googol C
Calculate 30% of 30. 9
....

Quadratics
Which is a root of 2x^2 - 6 = 0 ? (A) ^r2 (B) ^r3 (C) ^r6 (D) ^r12 B
Where is the vertex of this parabola? y = x^2 + 4x + 4 (-2,0)
Fill in the blank to “complete the square” x^2 - 6x + ___ 9
Solve : 2x^2 - 8x+ 6 = 0 (A) 1 , 3 (B) -1 , - 3 (C) 2 , 6 A
True or false: The Vertex of x^2 - 15 is located directly on the x-axis. True
....

Teacher Interface - Problems Stored in a Problem Class

The teacher must add new problems to the problem files. The program needs to control the input to prevent the teacher typing text that's too long for the 90 byte fields in the file. It must also convert shortcut codes to proper ASCII characters. The simplest way to do this is to create a data-storage Class called Problem. This class can store the 3 fields for a single problem (Question, Choices, Answer) and also ensure that these fields contain valid data by using **accessor** methods (**get** and **set** methods). The class will look something like this:

```
public class Problem
{
   private String question = "";
   private String choices  = "";
   private String answer   = "";

   public boolean setQuestion(String q)
   {
      //  replace shortcuts ^r , ^2 , ^3 with ASCII characters
      //  check that q.length is under 88 bytes
      //  if not, return false as a rejection message
      question = q;
      return true;
   }

   public boolean setChoices(String c)
   {
      // similar to setQuestion
   }

   public boolean setAnswer(String a)
   {
      // similar to setQuestion, but length must be under 18
   }

   public String getProblem() { return problem; }

   public String getChoices() { return choices; }

   public String getAnswer()  { return answer; }

   public boolean saveProblem(String fileName, int record)
   { // saves this problem into fileName
     //  in position specified by record

     // writes question, choices, and answer into the file

     // returns false if the operation failed
   }

   public boolean loadProblem(String fileName, int record)
   { // loads the problem from fileName
     //  at position specified by record

     // reads question, choices, and answer into the file

     // returns false if the operation failed
   }

}
```

Game Interface - Arrays of Problems and Buttons

Set of 9 Problems in 1-D Array

The Game Interface module can use the same **Problem** class for storing problems. Each game must select 9 random problems from the Files discussed above. So it will use those same files. After selecting 9 random problems, the problems can be stored in an array - an array of Problem objects:

```
Problem[] problems = new Problem[9] ;   // 1-dimensional array
```

The data will be the same as the data stored in the files (see above for sample data).

This array must be scrambled up randomly before starting the game - this can be done by swapping two random locations and repeating that hundreds of times. It's like shuffling cards.

GUI Buttons in a 1-D Array

Although the Tic-Tac-Toe board is in the shape of a 2-dimensional array, the coding for this is unnecessarily long. It's simpler to make a 1-dimensional array of Buttons.

```
Button[] squares = new Button[9] ;   // 1-dimensional array
```

square[0] at 50,50 » problem[0]	**square[1]** at 150,50 » problem[1]	**square[2]** at 250,50 » problem[2]
square[3] at 50,150 » problem[3]	**square[4]** at 150,150 » problem[4]	**square[5]** at 250,150 » problem[5]
square[6] at 50,250 » problem[6]	**square[7]** at 150,250 » problem[7]	**square[8]** at 250,250 » problem[9]

This approach is different than the prototype. It requires a bit of care in calculating the coordinates for the positions of the buttons - coordinates are shown in the diagram above.

The checkWinner( ) method is also a bit different than the prototype - it's actually shorter and easier to code, but requires a bit more thought than a 2-D array. Checking the columns for a winner will look something like this (it's actually less code than the prototype):

```
for (int col = 0; row < 3; row = row + 1)
{  String a = squares[col].getLabel();
   String b = squares[col+3].getLabel();
   String c = squares[col+6].getLabel();

   if (!a.equals("") && a.equals(b) && b.equals(c))
   {  output("Player " + a + " wins" ); }
}
```

This also turns the **problems[ ]** and **squares[ ]** arrays into **parallel arrays**, making lots of the coding simpler. For example, if **squares[3]** is clicked, then **problems[3]** is displayed.

Stage B3 - Modular Organization

Tasks Outline

I started organizing the solution by outlining the **tasks** that the users will perform and the **processes** that the program will perform automatically. This **outline** breaks down the **tasks** and connects them to the ~ automated processes ~ that must run in response. This is an overview - details are missing.

USER TASKS ~ relevant automated computer processes

TEACHER

> Choose a topic
~ store topic name and open data file
~ create and display an empty Tic-Tac-Toe board

> Add a new problem
~ format problem and replace keyboard shortcuts
~ save problem in data file

> Search for problem
~ input text
~ search for problems containing matching text
~ display each matching problem, until user accepts problem or says to quit

> **Edit a problem**
~ load the old version of the problem
~ allow user to make changes
~ format problem and replace keyboard shortcuts
~ save problem in data file back at the same record number

> Delete Problem
~ search for the problem, either by text or record number
~ ask user whether it's the correct problem
~ if okay, then erase the problem by writing blanks into the file

> **View Entire File**
~ input all problems from the file
~ display all problems in a scrolling text-area

STUDENTS

> Choose a topic
~ store topic name and open data file
~ create and display an empty Tic-Tac-Toe board

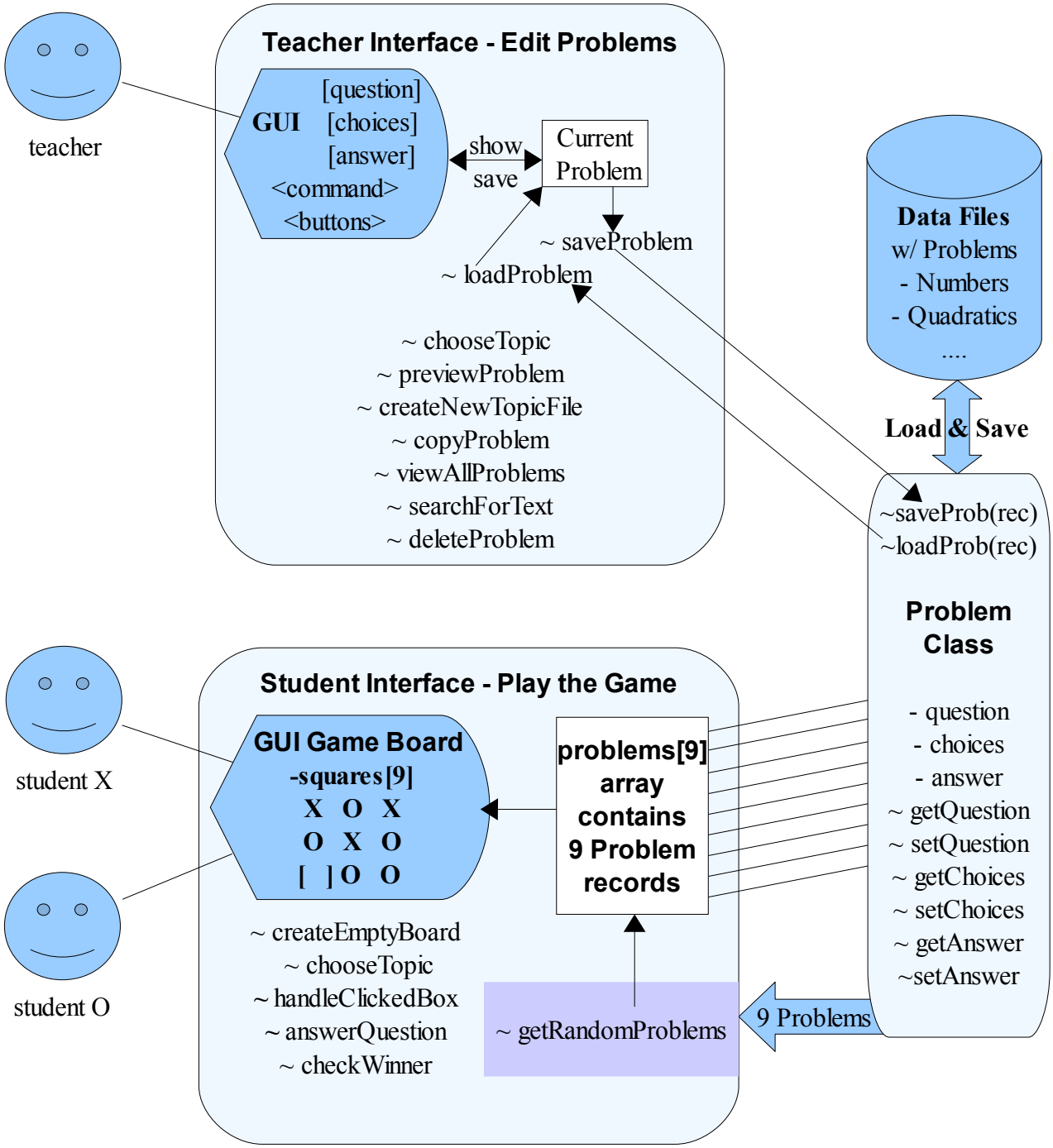
> Click on a square
~ check that square is still empty
~ if okay, then
display the question and input the student's answer
check the answer and mark square if correct
check if there is a winner, then change to other player's turn

Modular Organization Chart

This chart outlines the main modules for the Quiz-Tac-Toe system. This concentrates on the **computer program's** structure - it does not attempt to present the user tasks or actions.
The main purpose is to organize the **computer processes** from the outline (above) into modules.
The processes have been identified as **methods** and organized into modules (classes).

All access to Data Files should happen through the Problem Class, to ensure proper formatting and to avoid recoding duplicate file access methods in the teacher and student modules

The ~ symbol denotes methods. The arrows show data flow (problems) between various modules.



Stage B2 - Algorithms

Now that the methods have been named and organized, this section presents the methods as **algorithms** with detailed **pseudocode**, **parameters, return values, and pre- and post-conditions.** Simple methods are not shown in detail - only the non-obvious and non-standard algorithms are presented in detail. Pre- and post-conditions are only shown where they are significant.

== Problem Class ==

```
    private String question
    private String choices
    private String answer

** get and set accessor methods **

    getQuestion() returns String
         return question
    ... getChoices and getAnswer are similar ....

    setQuestion(String q) returns boolean
          // pre-condition: q contains text
         //  post-condition: either false is returned
         //                  or true is returned and question = q

          replace shortcuts ^r , ^2 , ^3 with ASCII characters
          if q.length > 88 then
              return false
          else
              question = q
              return true
    .... similarly for setChoices and setAnswer ....

** data file access methods **

    saveProblem(String fileName, int record) returns boolean
          // pre-condition: fileName must be a valid name
         //  post-condition: return false if method fails
         //                  else question, choices and answer
         //                  have been saved in filename at #record

         try
         {  file = open(fileName)
            file.seek(200*record)
            write question
            file.seek(200*record + 90)
            write choices
            file.seek(200*record + 180)
            write answer
            close file
            return true
         }
         catch (Exception)
         { return false }
    ..............
```

```
loadProblem(String fileName, int record) returns boolean
      // pre-condition: fileName must be a valid name and the
      //                the file should exist on the disk drive
      //
      //  post-condition: return false if method fails or if
      //                  #record contains blank Strings
      //                  else question, choices and answer
      //                  contain values from the data file

      try
      {  file = open(fileName) for reading only
         file.seek(200*record)
         read question
         file.seek(200*record + 90)
         read choices
         file.seek(200*record + 180)
         read answer
         close file
         return true
      }
      catch (Exception)
      { return false }
..............
```

== Student Game Class ==

```
Problem[] problems = new Problem[9]
Button[] squares = new Button[9]

createEmptyBoard() void

     // pre-condition  : none
     // post-condition : Buttons have been created and placed
     //                  in correct locations and blank labels

     x = 50 , y = 50
     for b = 0 to 8
          squares[b] = addButton("",x,y,100,100,this)
          x = x + 100
          if x > 250 then
               x = 50
               y = y + 100
.........

chooseTopic() returns String
     // pre-condition : none
     // post-condition : returns blank if topic file doesn't exist
     //                  else returns topic name
     String topic = input("Name of topic")
     try  open file named topic
          close file
          return topic
     catch(Exception) { return "" }
...........

handleClickedBox(Object source) void
     // pre-condition : click event has occurred
     //                 problems[ ] array contains 9 problems
     // post-condition: program has respond to the click event

     where = -1
     for b = 0 to 8
          if source == squares[b] then where = b

     if where >= 0 then
        if (squares[where].label is blank)  then
          output "Pick a different square"
          return
        output(problems[where].question + problems[where].choices)
        input guess
        if (guess matches problems[where].answer) then
          put current player's mark on squares[where]
        change to other player's turn
.........
```

```
checkWinner() void
     // pre-condition : board has been created
     // post-condition: program ends if there is a winner

     check(0,1,2)   // top row
     check(3,4,5)   // middle row
     check(6,7,8)   // bottom row
     check(0,3,6)   // left column
     check(1,4,7)   // middle column
     check(2,5,8)   // right column
     check(0,4,8)   // one diagonal
     check(6,4,2)   // other diagonal
..........

check(int a, int b , int c) void
     //  pre-condition : a, b, and c are between 0 and 8
     //  post-condition : program ends if all 3 labels match

     if ( !squares[a].label.equals("")
          && squares[a].label.equals(squares[b].label)
          && squares[b].label.equals(squares[c].label)
        )
        then
            output("Player " + squares[a].label + " wins")
            end program
..........

getRandomProblems(String fileName) boolean
     // pre-condition : fileName should be an existing file and
     //                 must contain at least 9 non-blank records
     // post-condition: problems[] array contains 9 Problem records

     String used = ""
     try
          for c = 0 to 8
               problems[c] = new Problem()
               repeat
                    repeat
                         rec = random between 0 and 999
                    until rec+"/" is not found in used
                    problems[c].loadProblem(fileName,rec)
               until problems[c].question is not blank
               used = used + rec + "/"
          next c
          return true

     catch (Exception)
      { return false }
.........
```

== Teacher Problem Editor Class ==

```
    String topic
    Problem problem     // a temporary variable for saving and loading

** GUI Interface **
    TextField questionBox   // teacher uses these boxes to type input
    TextField choicesBox    // also existing problems can be displayed
    TextField answerBox     // here for editing and then saved


    chooseTopic() returns String
         // pre-condition : none
         // post-condition : returns blank if topic file doesn't exist
         //                  else returns topic name
         String topic = input("Name of topic")
         try  open file named topic
              close file
              return topic
         catch(Exception) { return "" }
    ...........

    saveProblem() returns boolean
         // pre-condition : TextFields should contain text
         // post-condition : problem has been saved in topic file
         //                   if not possible return false
         problem.question = questionBox.getText()
         problem.choices  = choicesBox.getText()
         problem.answer   = answerBox.getText()

         if (problem.question is blank and problem.choices is blank
               or problem.answer is blank) then
            return false
         rec = inputInt("Which record number should this be - if you
                         don't care, type 0 and it will be stored
                          in the first blank record")
         if rec > 0 then
              return problem.saveProblem(topic , rec)
         else
              // need to find a blank record
              rec = 0
              do
                 Problem temp = new Problem()
                 temp.loadProblem(topic,rec)
                 if temp.answer is blank then
                    problem.saveProblem(topic , rec)
                    return true
                 end if
                 rec = rec + 1
              while rec < 1000
              return false   // if it gets here, the file is full
         end if
         return false    // if it gets here, something bad happened
    ........
```

```
loadProblem(String topic, int rec) returns boolean
     // purpose : loads and displays a problem
     // pre-condition : file for topic should exist
     // post-condition: Problem is displayed on screen

     success = problem.loadProblem(topic,rec)
     if success == false then
          return false
     else
          questionBox.setText(problem.question)
          choicesBox.setText(problem.choices)
          answerBox.setText(problem.answer)
          return true
     end if
........

createNewTopicFile(String topic) returns boolean
     // pre-condition : none
     // post-condition: file has been created if possible
     //                  otherwise return false

     try
          open file(topic) for reading only
          return false   // if it gets here, the file exists
     catch(Exception)
          // only gets here if the file does NOT exist
          // so now create it
          problem.question = ""
          problem.choices = ""
          problem.answer = ""
          problem.saveProblem(topic,999)  // creates last record
          return true
.........

tryProblem(String topic, int rec) void
     // purpose : displays problem as students will see it
     //           allows the teacher to type an answer
     //           and checks right or wrong
     //           code can be copied from Student Game class
.........

copyProblem(String topic, int rec) void
     // same as loadProblem
.........

previewAllProblems(String topic) void
     // purpose : display all text from all problems in a TextArea
     // pre-condition : toic file should exist
     // post-condition: textArea contains all problems
     clear textArea
     for rec = 0 to 999
          problem.loadProblem(topic,rec)
          textArea.append(rec+"|"+question+"|"+choices+"|"+answer)
     next rec
.........
```

```
     searchForText(String text) void
          // similar to viewAllProblems, but only displays problems
          //  containing the desired text
          // pre-condition : toic file should exist
          // post-condition: textArea contains all matching problems
          clear textArea
          for rec = 0 to 999
               problem.loadProblem(topic,rec)
               all = rec+"|"+question+"|"+choices+"|"+answer
               if all.indexOf(text) >= 0 then
                    textArea.append(all)
          next rec
     .........

     deleteProblem(String topic, int rec) void
          // pre-condition :  none
          // post-condition:  problem #rec has been erased in topic file

          put blanks into current problem
          saveProblem(topic, rec)
     .........
```

Preliminary Mastery Check

Judging from the Stage B Detailed Design, the following SL mastery factors should be satisfied:

Arrays	Button[] squares, Problem[] problems in Game class
User-defined objects	Problem class
Objects as data records	Problem class
Simple if..then..	many places
Complex if..then..	check method in Game class
Loops	many places
Nested Loops	getRandomProblems in Game class
User-defined methods	many
User-defined methods with parameters	many
User-defined methods with return values	many
Sorting	------
Searching	searchForText and deleteProblem
File i/o	RandomAccessFiles in Problem class
Additional libraries	AWT GUI interfaces in Game and Teacher modules
Sentinels or flags	boolean return values for many methods

It appears that 14/15 mastery items will be demonstrated, so this should be sufficiently challenging.

C1 - Program Listing

Problem.java

```
1  /**
2   * @author Dave Mulkey
3   * @date July 2008
4   *
5   * Quiz-Tac-Toe
6   * Game module
7   * IDE - Eclipse
8   * Java - Ver 1.5
9   * Platform - PC
10  */
11
12 import java.awt.Font;
13 import java.io.*;
14
15 import javax.swing.JOptionPane;
16
17
18 public class Problem
19 {
20    private String question = "";
21    private String choices = "";
22    private String answer = "";
23
24    public String getQuestion()
25    {
26       return question;
27    }
28
29    public String getChoices()
30    {
31       return choices;
32    }
33
34    public String getAnswer()
35    {
36       return answer;
37    }
38
39    public boolean setQuestion(String q)
40     //  precondition: q contains text
41     //  postcondition: either false is returned
42     //                 or true is returned and question = q
43    {
44        q = replace(q,"^r","\u221a");
45        q = replace(q,"^2","\u00b2");
46        q = replace(q,"^3","\u00b3");
47        if (q.length() > 88)
48        {
49           question = q.substring(0,88);
50           return false;
```

```
51          }
52          else
53          {
54             question = q;
55             return true;
56          }
57       }
58
59       public boolean setChoices(String q)
60        // precondition: q contains text
61        //  postcondition: either false is returned
62        //                 or true is returned and choices = q
63       {
64          q = replace(q,"^r","\u221a");    // square root
65          q = replace(q,"^2","\u00b2");    // squared
66          q = replace(q,"^3","\u00b3");    // cubed
67          if (q.length() > 88)
68          {
69             choices = q.substring(0,88);
70             return false;
71          }
72          else
73          {
74             choices = q;
75             return true;
76          }
77       }
78
79       public boolean setAnswer(String q)
80        //  precondition: q contains text
81        //  postcondition: either false is returned
82        //                 or true is returned and answer = q
83       {
84          q = replace(q,"^r","\u221a");
85          q = replace(q,"^2","\u00b2");
86          q = replace(q,"^3","\u00b3");
87          if (q.length() > 18)
88          {
89             answer = q.substring(0,18);
90             return false;
91          }
92          else
93          {
94             answer = q;
95             return true;
96          }
97       }
98
99       public String replace(String s, String find, String change)
100      // purpose : replace all occurences of *find* with *change*
101      // pre-condition : parameters set up correctly
102      // post-condition: s has been changed and is returned
103      {
104         int p = s.indexOf(find);
105         while (p >= 0)
106         {
```

```
107           s = s.substring(0,p) + change +
s.substring(p+find.length());
108           p = s.indexOf(find);
109        }
110        return s;
111     }
112
113     public boolean saveProblem(String fileName, int record)
114     //  pre-condition: fileName must be a valid name
115     //  post-condition: return false if method fails
116     //                else question, choices and answer
117     //                   have been saved in filename at #record
118     {
119
120         try
121        {
122          RandomAccessFile file = new RandomAccessFile(fileName,"rw");
123           file.seek(200*record);
124           file.writeUTF(question);
125           file.seek(200*record + 90);
126           file.writeUTF( choices);
127           file.seek(200*record + 180);
128           file.writeUTF( answer);
129           file.close();
130           return true;
131        }
132        catch(IOException ex)
133        { return false; }
134
135     }
136
137     public boolean loadProblem(String fileName, int record)
138      // pre-condition: fileName must be a valid name
139     //  post-condition: return false if method fails
140     //                else question, choices and answer
141     //                   have been saved in filename at #record
142     {
143
144         try
145        {
146          RandomAccessFile file = new RandomAccessFile(fileName,"r");
147           file.seek(200*record);
148           question = file.readUTF();
149           file.seek(200*record + 90);
150           choices = file.readUTF();
151           file.seek(200*record + 180);
152           answer = file.readUTF();
153           file.close();
154           return true;
155        }
156        catch(IOException ex)
157        { return false; }
158
   }
160
161     public boolean isBlank()
```

```
162     // purpose : returns true if all three fields are blank
163     // pre-condition : none
164     // post-condition: returns true if blank, false if not blank
165     {
166        if (question.length()==0
167            && answer.length()==0
168            && choices.length()==0
169           )
170        { return true; }
171        else
172        { return false; }
173     }
174 }
```

Game.java

```
/**
 * @author Dave Mulkey
 * @date July 2008
 *
 * Quiz-Tac-Toe
 * Game module
 * IDE - Eclipse
 * Java - Ver 1.5
 * Platform - PC
 */

import java.awt.*;
import javax.swing.JOptionPane;

public class Game extends EasyApp // EasyApp for easy GUI components
{
   public static void main(String[] args)
   {  new Game(); }

   Button[] squares = new Button[9];        // 3x3 game board

   Button newGameBtn = addButton("New Game",340,40,150,40,this);
   Button newTopicBtn = addButton("New Topic",340,80,150,40,this);
   Button helpBtn = addButton("Instructions",340,300,150,40,this);

   Label lTurn = addLabel("Player",380,150,100,35,this);
   Label turn = addLabel("X",390,170,100,100,this);
                                    // shows turn X or O
   String topic = "";               // name of current topic

   Problem[] problems = new Problem[9];

   String used = "";


   int player = 1; // switches between players:1 = X and -1 = O
   public Game()
   {
     setTitle("Quiz-Tac-Toe");
     setBounds(50,40,600,400);
     Font thefont = new Font("Arial",0,64);
     turn.setFont(thefont);
     lTurn.setFont(new Font("Arial",0,24));
     createEmptyBoard();
     chooseTopic();
     getRandomProblems(topic);
   }

   public void actions(Object source, String command)
   // purpose : Find out which Button was clicked
   // pre-condition : a Button was clicked
   // post-condition: click has been handled
```

```
53    {
54       int qnum = -1;
55        for (int n = 0; n < 9; n = n+1)
56        {
57          if (source == squares[n])
58          {  qnum = n; }              // remember button number
59        }
60
61       if (qnum >= 0)
62       // Handle the clicked Button
63       {  if (squares[qnum].getLabel().equals(""))
64          { String guess = input(problems[qnum].getQuestion() ,
problems[qnum].getChoices(),24);
65            if (guess.equalsIgnoreCase(problems[qnum].getAnswer()))
66            {                                    // answer was correct
67             if (player == 1)
68             {  squares[qnum].setLabel("X"); }
69             else
70             {  squares[qnum].setLabel("O"); }
71            }
72            checkWinner();
73            player = -1*player;
74            if (player==1)                       //other player's turn
75            {  turn.setText("X"); }
76            else
77            {  turn.setText("O"); }
78          }
79          else
80          {
81            output("Choose an empty square");
82          }
83       }
84       else if (source == newGameBtn)
85       {
86          for (int b = 0; b < 9; b = b+1)    // clear the board
87          {
88             squares[b].setLabel("");
89          }
90          getRandomProblems(topic);
91          player = 1;
92          turn.setText("X");
93       }
94       else if (source == newTopicBtn)
95       {
96            chooseTopic();
97          for (int b = 0; b < 9; b = b+1)    // clear the board
98          {
99             squares[b].setLabel("");
100         }
101         getRandomProblems(topic);
102         player = 1;
103         turn.setText("X");
104      }
105      else if (source == helpBtn)
106      {
107         showInstructions();
```

```
108       }
109    }
110
111    public void chooseTopic()
112    {
113       topic = "";
114       do
115       {  Problem problem = new Problem();
116           topic = input("Topic name (or quit)?");
117           if (problem.loadProblem(topic,0) == false)
118                                  // if file does not exist
119           {  topic = ""; }       //  then set topic back to blank
120       }  while (topic.equals("")) ;  // until file exists
121    }
122
123    public void getRandomProblems(String topic)
124    // purpose : choose 9 random problems from topic file
125    // pre-condition : topic file exists and contains
126    //                 at least 9 problems
127    // post-condition: problems[] array contains 9 problems
128    {
129      // find last problem in file
130       int last = 999;
131       Problem problem = new Problem();
132       problem.loadProblem(topic, last);
133       while(last > 0 && problem.isBlank())  // searching backwards
to
134       {                                     // find last non-blank
record
135         last = last - 1;
136         problem.loadProblem(topic, last);
137       }
138       if (last < 8)               // need at least 9 problems in file
139       {  output("Not enough problems in this file -\n choose a
different topic");
140          return;
141       }
142      used = "";
143      for (int p = 0; p < 9; p = p+1)   // choose 9 random problems
144      {
145         int r;
146         do
147         {
148           r = (int)Math.floor(Math.random()*(last+1)); // random
problem
149         } while (used.indexOf(r+"")>=0);   // don't pick same
problem twice
150         used = used + r + "|";
151         problems[p] = new Problem();
152         problems[p].loadProblem(topic,r);
153      }
154    }
155
156    public void createEmptyBoard()
157    // purpose : creates Buttons and displays 3x3 board
158    // pre-condition  : none
```

```
159     // post-condition : Buttons have been created and placed
160     //                  in correct locations and blank labels
161     {
162       Font thefont = new Font("Arial",0,64);
163       int x = 10 ;
164       int y = 40;
165       for (int b = 0; b < 9; b = b+1)     // create Buttons and
166       {                                   // place them in 3x3 grid
167          squares[b] = addButton("",x,y,100,100,this);
168          squares[b].setFont(thefont);
169          x = x + 100;                     // calculating coordinates
170          if (x > 210)
171          {
172           x = 10;
173           y = y + 100;
174          }
175       }
176     }
177
178
179
180     public void checkWinner()
181     // purpose : Check for a tic-tac-toe winner (3 in a row)
182     // pre-condition : tic-tac-toe board exists (squares[])
183     // post-condition: if a winner is found, game ends
184     {
185        check(0,1,2);   // top row
186        check(3,4,5);   // middle row
187        check(6,7,8);   // bottom row
188        check(0,3,6);   // left column
189        check(1,4,7);   // middle column
190        check(2,5,8);   // right column
191        check(0,4,8);   // one diagonal
192        check(6,4,2);   // other diagonal
193          checkBoardFull();
194     }
195
196     public void checkBoardFull()
197     // purpose : check whether the board is full
198     //           if so, the player with more squares wins
199     // pre-condition : have already checked for wins
200     // post-condition: if full, game ends
201     {
202        int countX = 0;
203        int countO = 0;
204        for (int s = 0; s < 9; s = s+1)
205        {
206           if (squares[s].getLabel().equals("X"))
207           {  countX++; }
208           else if (squares[s].getLabel().equals("O"))
209           {  countO++; }
210        }
211        if (countX + countO == 9)
212        {  if (countX > countO)
213           { output("X wins");
214             System.exit(0);
```

```
215           }
216           else
217           { output("O wins");
218             System.exit(0);
219           }
220        }
221     }
222
223     public void check(int a, int b , int c)
224     //  purpose : check whether buttons a,b, and c match
225     //  pre-condition : a, b, and c are between 0 and 8
226     //  post-condition : program ends if all 3 labels match
227     {
228      if ( !squares[a].getLabel().equals("")
229          && squares[a].getLabel().equals(squares[b].getLabel())
230          && squares[b].getLabel().equals(squares[c].getLabel())
231         )              // checking that 3 squares match and aren't
blank
232       {
233         output("Player " + squares[a].getLabel() + " wins");
234         System.exit(0);
235       }
236     }
237
238     public String input(String msg1,String msg2, int size)
239     // Purpose  :  Display a problem , input and return guess
240     // pre-condition : question, choices, and font-size passed
241     // post-condition: returns user guess
242     {
243        // Swing Button accepts HTML for formatting the text
244        // For example, can use <sup> for exponents
245        // This code just writes Question and Choices on 2 lines
246        javax.swing.JButton message = new javax.swing.JButton(
247           "<html><body><pre><font face='Verdana' size=5>"
248            + msg1 + "<br>" + msg2 + "</font></pre></body></html>");
249
250        message.setFont(new Font("Arial",0,size));
251
252         return JOptionPane.showInputDialog(null,message);
253     }
254
255     public void showInstructions()
256     {
257        runProgram("explorer.exe instructions.htm");
258     }
259  }
260
```

ProblemEditor.java

```
/**
 * @author Dave Mulkey
 * @date July 2008
 *
 * Quiz-Tac-Toe
 * Problem Editor module
 * IDE - Eclipse
 * Java - Ver 1.5
 * Platform - PC
 */

  import java.awt.*;
  import java.io.*;

  import javax.swing.JOptionPane;

   public class ProblemEditor extends EasyApp
   {
       public static void main(String[] args)
      {  new ProblemEditor(); }

      Problem problem = new Problem();
      Button topicBtn = addButton("Topic",10,40,90,30,this);
      TextField topicBox = addTextField("",100,40,100,30,this);
      Button createBtn = addButton("New Topic",200,40,100,30,this);
      Button showAllBtn = addButton("Show all",310,40,60,30,this);
      Button searchBtn = addButton("Search",370,40,60,30,this);

      Button clearBtn = addButton("Clear",500,40,50,30,this);

      Button saveBtn = addButton("Save",310,160,60,30,this);
      Button tryItBtn = addButton("Try It",370,160,60,30,this);

      Button eraseBtn = addButton("Delete",610,40,60,30,this);

      Label questionLbl = addLabel("Question",10,80,60,30,this);
      TextField questionBox = addTextField("",70,80,600,30,this);
      Label choicesLbl = addLabel("Choices",10,120,60,30,this);
      TextField choicesBox = addTextField("",70,120,600,30,this);
      Label answerLbl = addLabel("Answer",10,160,60,30,this);
      TextField answerBox = addTextField("",70,160,200,30,this);
      Label viewerLbl = addLabel("Viewer",10,250,50,30,this);
      List viewerBox = addList("",70,200,600,200,this);

      String topic = "";

       public ProblemEditor()
      {
         setTitle("Quiz-Tic-Tac Problems Editor");
         setBounds(100,100,680,410);
      }

       public void actions(Object source, String command)
```

```
54        {
55           if (source == topicBtn)
56           {  boolean success = chooseTopic();
57              if (success == false)
58              {  output("Topic choice did not succeed");
59                 topic = "";
60              }
61              else
62              {  topicBox.setText(topic);
63                 previewAllProblems(topic);
64              }
65           }
66           else if (source == createBtn)
67           {   topic = input("New Topic Name");
68              boolean success = createNewTopicFile(topic);
69              if (success == false)
70              {  output("Topic choice did not succeed");
71                 topic = "";
72              }
73              else
74              {  topicBox.setText(topic); }
75           }
76           else if (source == tryItBtn)
77           {
78              tryProblem();
79           }
80           else if (source == saveBtn)
81           {
82              boolean success = saveProblem(topic);
83              if (success == false)
84              {  output("Save failed"); }
85              previewAllProblems(topic);
86           }
87           else if (source == showAllBtn)
88           {  previewAllProblems(topic); }
89           else if (source == viewerBox)
90           {
91              copyProblem(viewerBox.getSelectedItem());
92           }
93           else if (source == eraseBtn)
94           {
95              deleteProblem(topic);
96              previewAllProblems(topic);
97           }
98           else if (source == clearBtn)
99           {
100             questionBox.setText("");
101             choicesBox.setText("");
102             answerBox.setText("");
103          }
104          else if (source == searchBtn)
105          {
106             search(topic);
107          }
108
109       }
```

```
110
111         public boolean chooseTopic()
112        {
113           topic = input("Topic name?");
114           if (problem.loadProblem(topic,0) == true)
115           {
116              return true; }
117           else
118           {
119              return false; }
120        }
121
122         public boolean createNewTopicFile(String topic)
123        // pre-condition : none
124        // post-condition: file has been created if possible
125        //                   otherwise return false
126        {
127           try
128           {
129              RandomAccessFile file = new RandomAccessFile(topic,"r");
130              file.seek(0);
131              String test = file.readUTF();   // try to read from file
132              return false;       // if it gets here, the file exists
133           }
134               catch(Exception ex)
135              // only gets here if the file does NOT exist
136              // so now create it
137              {   problem.setQuestion("");
138                 problem.setChoices("");
139                 problem.setAnswer("");
140                 problem.saveProblem(topic,999); //create last record
141                                         // forcing file to 1000 records
142                 return true;
143              }
144        }
145
146         public boolean saveProblem(String topic)
147        // pre-condition : TextFields should contain text
148        // post-condition : problem has been saved in topic file
149        //             if not possible return false
150        {
151           problem.setQuestion(questionBox.getText());
152           problem.setChoices(choicesBox.getText());
153           problem.setAnswer( answerBox.getText());
154
155           if ( problem.getQuestion().equals("")
                 && problem.getChoices().equals("")
156              || problem.getAnswer().equals("")
157             )
158           {  output("Your problem is incomplete, but will be saved
anyway"); }
159           int rec = inputInt("Which record number should this be?\n"
160                       +"If you don't care, type 0 to add it\n"
161                       +"in the first blank record, or\n"
162                       +"type -1 to cancel saving");
163           if (rec < 0)
```

```
164          {
165             return false; }
166          else if (rec > 0)
167          {
168             return problem.saveProblem(topic , rec);}
169          else
170          // need to find a blank record
171          {
172             rec = 0;
173             do
174             {   Problem temp = new Problem();
175                temp.loadProblem(topic,rec);
176                if (temp.getAnswer().equals("")
177                && temp.getChoices().equals("")
178                && temp.getQuestion().equals("")
179                )
180                { problem.saveProblem(topic , rec);
181                   return true;
182                }
183                else
184                {  rec = rec + 1; }
185             } while (rec < 1000);
186             return false;   // if it gets here, the file is full
187                             // so the problem cannot be saved
188          }
189       }
190
191        public void previewAllProblems(String topic)
192       // purpose : display all text from all problems in a TextArea
193       // pre-condition : topic file should exist
194       // post-condition: textArea contains all problems
195       {
196          viewerBox.removeAll();
197          for(int rec = 0; rec < 1000; rec = rec + 1)
198          {  problem.loadProblem(topic,rec);
199             if (   problem.getQuestion().length()>0
200             || problem.getChoices().length()>0
201             || problem.getAnswer().length()>0
202             )
203                viewerBox.add( rec+" | " + problem.getQuestion()+" | "
204                 + problem.getChoices()+" | "+problem.getAnswer() );
205          }
206       }
207
208        public void search(String topic)
209       // purpose : display all text from all problems in a TextArea
210       // pre-condition : toic file should exist
211       // post-condition: textArea contains all problems
212       {
213          String text = input("Text to find");
214          viewerBox.removeAll();
215          for(int rec = 0; rec < 1000; rec = rec + 1)
216          {  problem.loadProblem(topic,rec);
217             if (    problem.getQuestion().indexOf(text)>=0
218             || problem.getChoices().indexOf(text)>=0
219             || problem.getAnswer().indexOf(text)>=0
```

```
220             )
221             { viewerBox.add( rec+" | " + problem.getQuestion()+" | "
222                   + problem.getChoices()+" | "+problem.getAnswer() );
223             }
224          }
225       }
226
227        public void tryProblem()
228       // purpose : display current problem (in editor boxes)
229       //           as students will see it, and try answering it
230       // pre-condition : TextFields contain question, choices,answer
231       // post-condition: problem is displayed
232       {
233          problem.setQuestion(questionBox.getText());
234          problem.setChoices(choicesBox.getText());
235          problem.setAnswer(answerBox.getText());
236
237          String guess = input(problem.getQuestion(),
problem.getChoices(),24);
238          if (guess.equalsIgnoreCase(problem.getAnswer()))
239          { output("Right");}
240          else
241          { output("Wrong");}
242       }
243
244        public void copyProblem(String s)
245       // purpose : copy a problem from viewerBox into TextFields
246       //           then this can be edited and resaved or added
247       //           as a new problem
248       // pre-condition : problems have been displayed in ViewerBox
249       // post-condition: problem is copied into TextFields
250       {
251          int p = s.indexOf("|");
252          int rec = Integer.parseInt(s.substring(0,p-1));
253          problem.loadProblem(topic,rec);
254
255          questionBox.setText(problem.getQuestion());
256          choicesBox.setText(problem.getChoices());
257          answerBox.setText(problem.getAnswer());
258
259       }
260
261        public void deleteProblem(String topic)
262       // pre-condition :  none
263       // post-condition:  problem #rec has been erased in topic file
264       {
265          int suggest = firstNumber(viewerBox.getSelectedItem());
266          if (suggest >= 0)
267          {
268             int rec = inputInt("Record number to delete",suggest);
269             problem.setQuestion("");
270             problem.setChoices("");
271             problem.setAnswer("");
272             problem.saveProblem(topic,rec);
273          }
274          else
```

```
275          {  output("First select a problem in the viewer window"); }
276       }
277
278        public int inputInt(String prompt,int suggest)
279        // a special version of input, that displays a default value
280        // in the input box - to be used when saving, to suggest
281        // saving in the same record that was hilighted
282       {
283          try
284          {
285             return
Integer.parseInt(JOptionPane.showInputDialog(null,prompt,suggest));  }
286              catch(Exception ex)
287             {
288                return 0; }
289       }
290
291      public static String input(String msg1,String msg2, int size)
292      // a special version of output to print bigger text
293      // the Swing JButton accepts HTML formatting commands
294      {
295         javax.swing.JButton message = new javax.swing.JButton(
296            "<html><body><pre><font face='Verdana' size=5>"
297            + msg1 + "<br>" + msg2 + "</font></pre></body></html>");
298
299         message.setFont(new Font("Arial",0,size));
300
301         return JOptionPane.showInputDialog(null,message);
302      }
303
304      public int firstNumber(String s)
305       // purpose : parses the int number at the beginning of s
306       // pre-condition: s should have an int followed by a space
307       // post-condition: returns the converted int value
308       //                 or -1 if the conversion failed
309      {
310         try
311         {
312            int p = s.indexOf(" ");
313            if (p<0)
314            {
315               return -1; }
316            else
317            {
318               return Integer.parseInt(s.substring(0,p)); }
319         }
320             catch(Exception ex)
321            {
322               return -1;
323            }
324      }
325   }
```

QuizTacToe.java

```
1  /**
2   * @author Dave Mulkey
3   * @date July 2008
4   *
5   * Quiz-Tac-Toe
6   * Game module
7   * IDE - Eclipse
8   * Java - Ver 1.5
9   * Platform - PC
10  */
11
12 // This is the MAIN start-up screen.
13 // Users can click on
14 // - TEACHERS to start the ProblemEditor
15 // - STUDENTS to start the Game interface
16
17 import java.awt.*;
18 public class QuizTacToe extends EasyApp
19 {
20    public static void main(String[] args)
21    {
22       new QuizTacToe();
23    }
24
25    Button bGame = addButton("Students",40,40,100,40,this);
26    Button bEdit = addButton("Teachers",150,40,100,40,this);
27
28    public QuizTacToe()
29    {
30       setBounds(50,50,290,100);
31       setTitle("QuizTacToe");
32    }
33
34    public void actions(Object source, String command)
35    {
36       if (source == bGame)
37       { new Game();
38         this.dispose();    // close main program,
39                            // so students don't start
40                            // another Game window
41       }
42       if (source == bEdit)
43       { new ProblemEditor();  // leave main program open,
44                               // so teachers can start a Game
45                               // to test their problems
46       }
47    }
48 }
```

C2 – Usability

Usability features are shown in many of the hard-copy output screens. The following highlights the most significant usability features. Some specific screen shots are referenced, but others repeat the same features.

GUI Interface

The major usability feature is the GUI interface.

In the Game module

- users can simply click on a square to get a question *(screens G1-G10)*
- X and O markers are large and clear *(screens G1-G10)*

In the teacher's Editor module

- the problem preview looks and behaves just like the game module *(screens E25 and G4)*
- entire problem list is displayed in a scrollable list box *(screen E15)*
- problems can be selected easily by clicking on the list *(screen E3)*

Math Symbols

A few special math symbols can be typed using simple shortcuts, and are then displayed correctly by replacing the shortcuts with ASCII code characters. The result is easily readable math symbols that are easy to create. *(screen E11)*

Storage

Sets of problems for various topics are stored in data files, with simple commands for adding, deleting, searching and editing. New topic files can be added easily. *(screens E1 - E25)*

Easy to Understand

Standard Tic-Tac-Toe rules are implemented correctly, so the game is easy to understand and easy to use. *(T9 – T22)*.

C3 - Handling Errors

File Access

All file-access commands are wrapped in **try..catch..** blocks, in order to trap all IOExceptions.
See **saveProblem** and **loadProblem** in the Problem.java class.

Cheating In the Game

If a user clicks on a box that is not empty, it is rejected *(screen G17).*
See Game.java, lines 63-82.

Inputing an Incorrect Topic Name

If the students type a topic name that does not exist, it is rejected *(screen T9).*
See Game.java, lines 111-120.

If the teacher types a topic name that does not exist, it is rejected *(screen T1).*
See ProblemEditor.java, lines 55-65.

Saving Problems

If the teacher forgets to type an answer to a problem, she is warned *(screen T5).*
See ProblemEditor.java, lines 155-158.

Deleting Incorrectly

If the teacher clicks the [Delete] button before choosing a problem, an error message is displayed *(screen T7)*. See ProblemEditor.java, lines 261-272

Answers are Not Case Sensitive

Players can type capital or small letters, as they wish.
See Game.java, line 65.

Returning Boolean Flags

Many methods return a **"success"** flag as a boolean value. *(T1-T12)*. For example, the **saveProblem** and **loadProblem** methods in the **Problem.java** class.

C4 - Success of the Program - Testing Criteria for Success

The following table shows the Criteria For Success from A2, together with references to screen-shots showing that the Criteria were achieved.

Criteria for Success	***Success?***
Game plays Tic-Tac-Toe , using correct Tic-Tac-Toe rules	Yes - (T15) to (T22)
Each Tic-Tac-Toe square asks a question	Yes - various screen-shots
Questions will be selected randomly and scrambled	Yes - see various games in (G) screen-shots, especially (G2) and (G11)
Questions should contain appropriate math content	Yes - various screen-shots
Game should be quick, easy and satisfying, including a simple and clear user-interface	Yes - various screen-shots
Teacher can create and save problems	Yes - all the (E) screen-shots
Some special symbols can be used in the questions – squares, cubes, simple square-roots	Yes - (E11)
Teacher module for typing and saving questions and answers	Yes - all the (E) screen-shots
Questions are saved into various files according to topic.	Yes - various screen-shots, especially (T5) and (T6)
Teacher should be able to add more topics (files) later	Yes - (E12) to (E14)
Questions in data-files can be added, changed and deleted later	Yes - various (E) screen-shots
It should be easy to copy a problem, change a few numbers and then save as a new problem	Yes - (E17) to (E20)

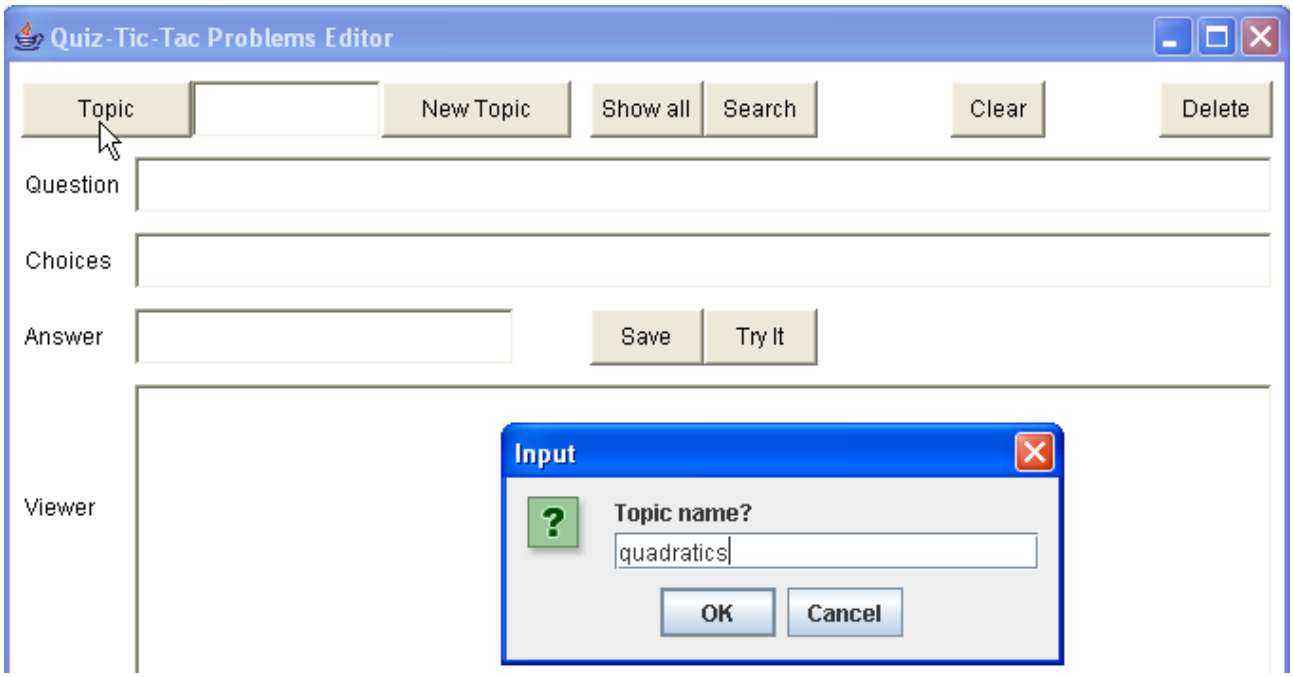
Stage D1 - Annotated Hard Copy of Test Output

The test output is organized into 4 sections

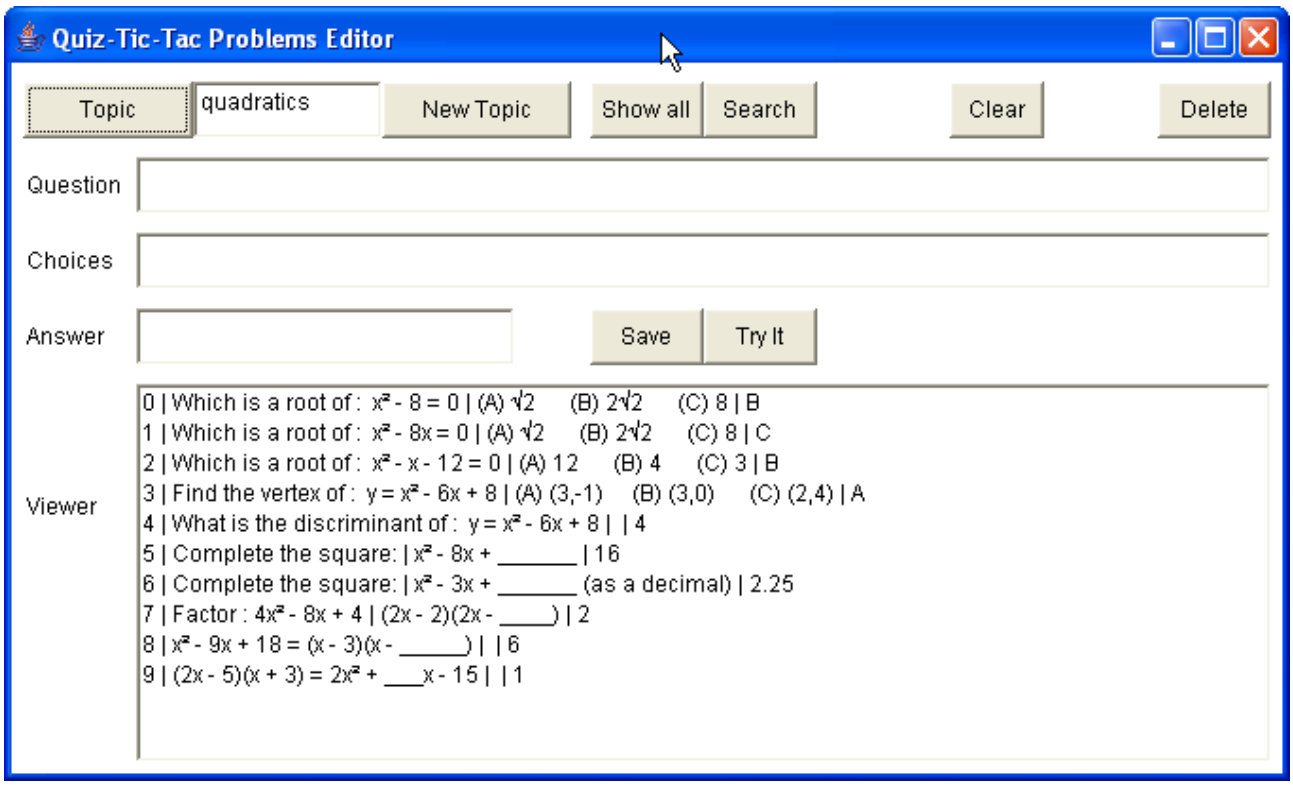
1. Typical run of the Teacher's Problem Editor
2. Typical run of the Students' Game
3. Testing reliability and error-handling
4. Testing Criteria for Success

Editor - Typical Run for Teacher's Problem Editor

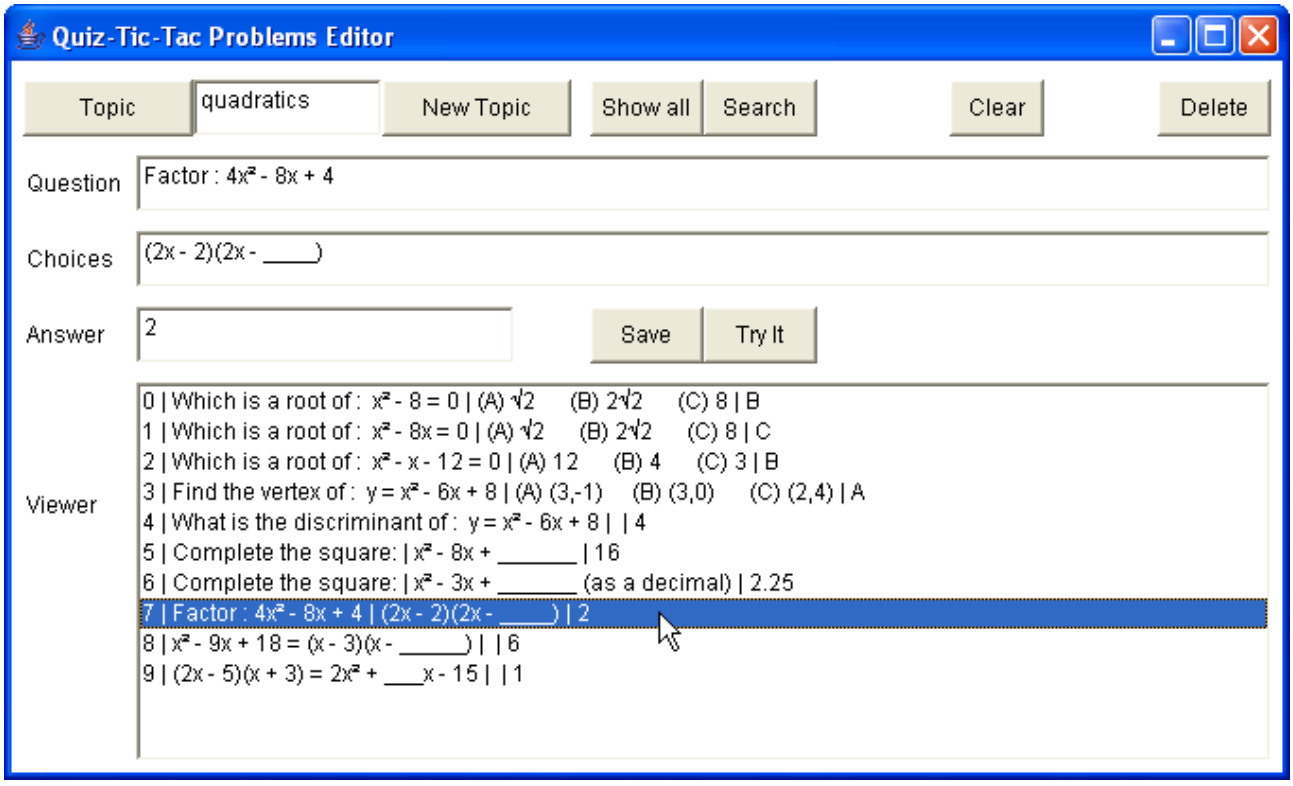
(E1) The teacher opens an existing file - quadratics.



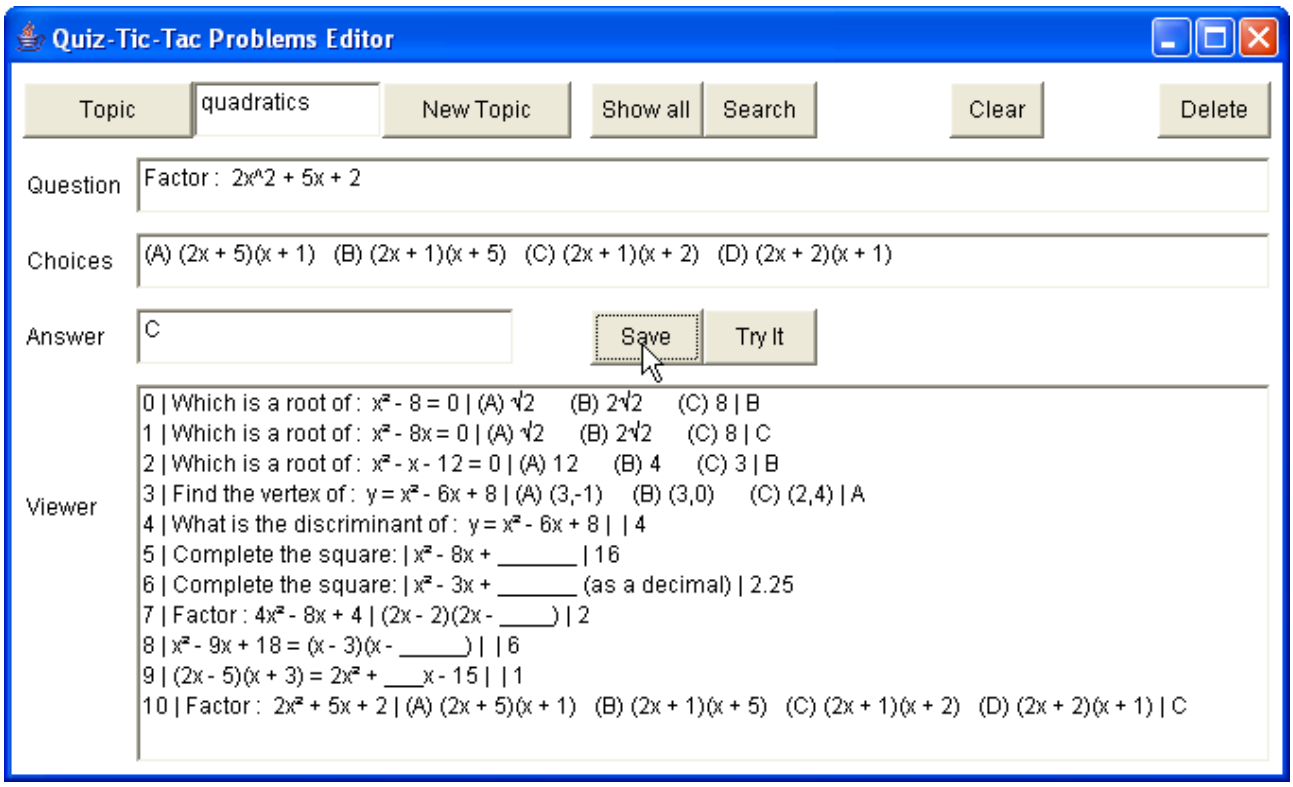
(E2) The editor automatically displays all the problems from the file.



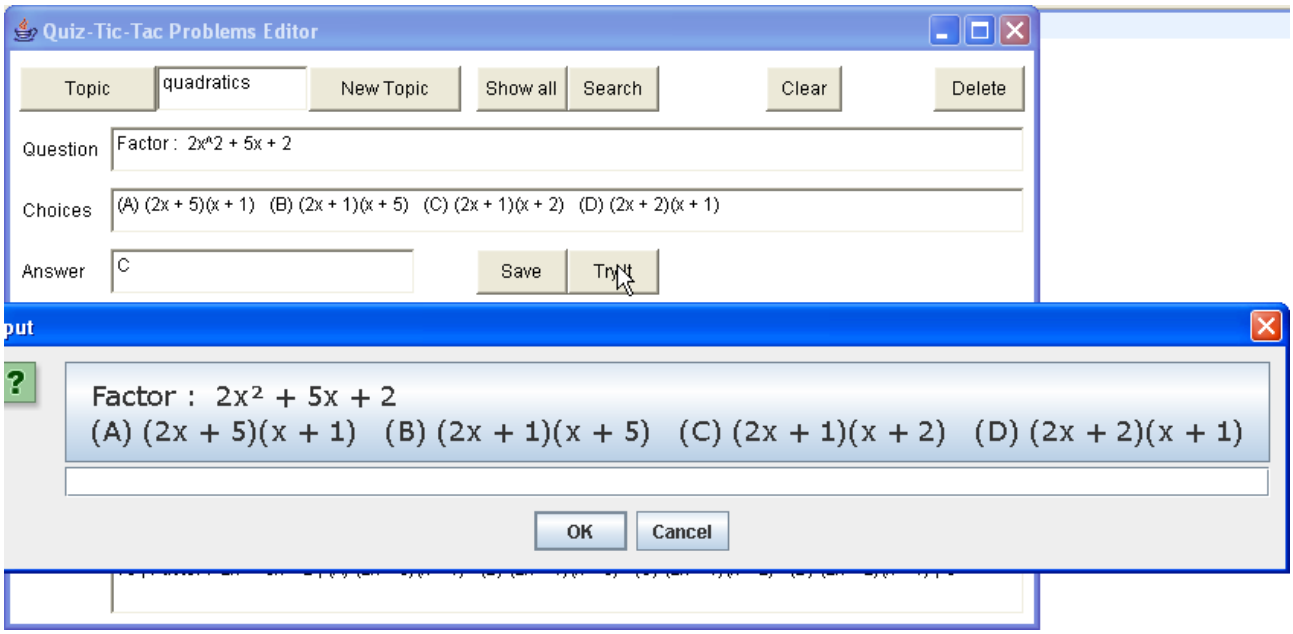
(E3) The teacher wants to make more factoring problems like #7. So she clicks on #7 and the program loads it into the editing boxes.



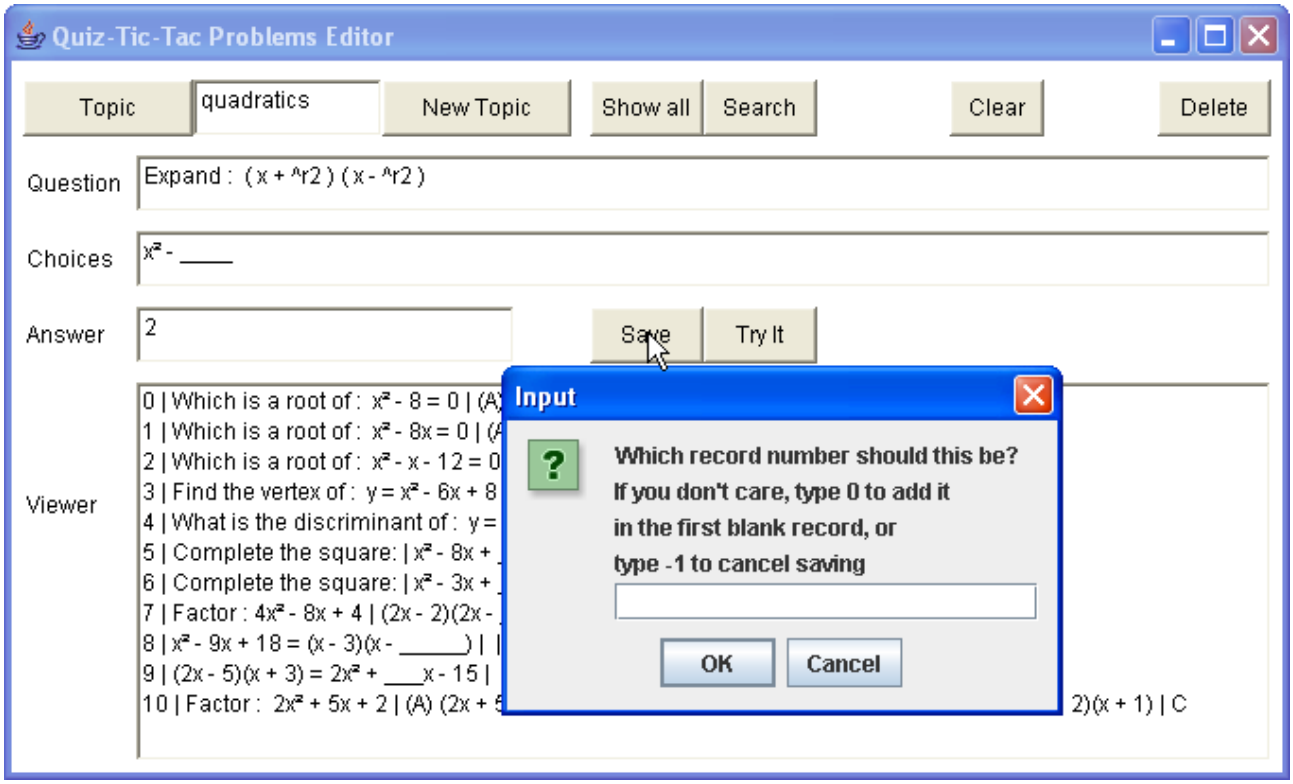
(E4) The teacher changes the question, choices, and answer boxes, then clicks [Save]. The program automatically updates the list of problems in the Viewer box.



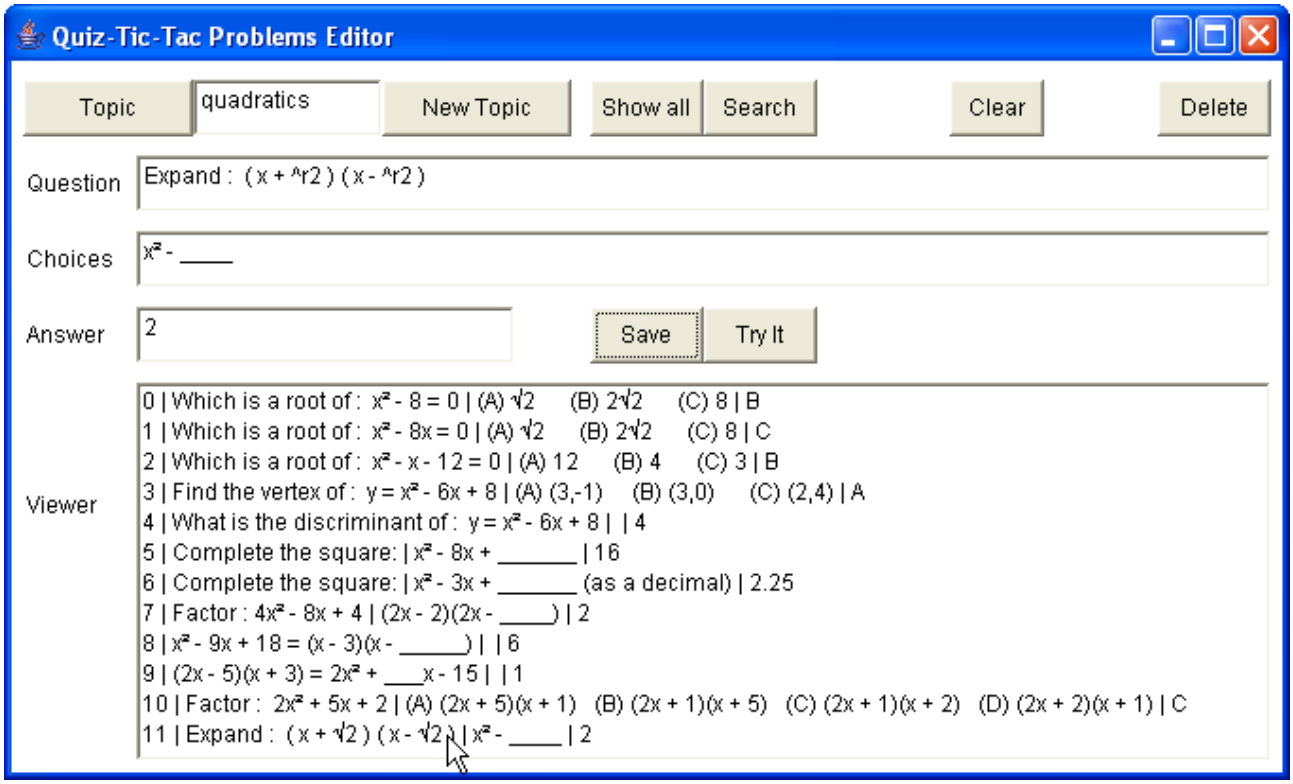
(E5) The teacher wants to see how the problem looks when students play the game, so she clicks [Try it]



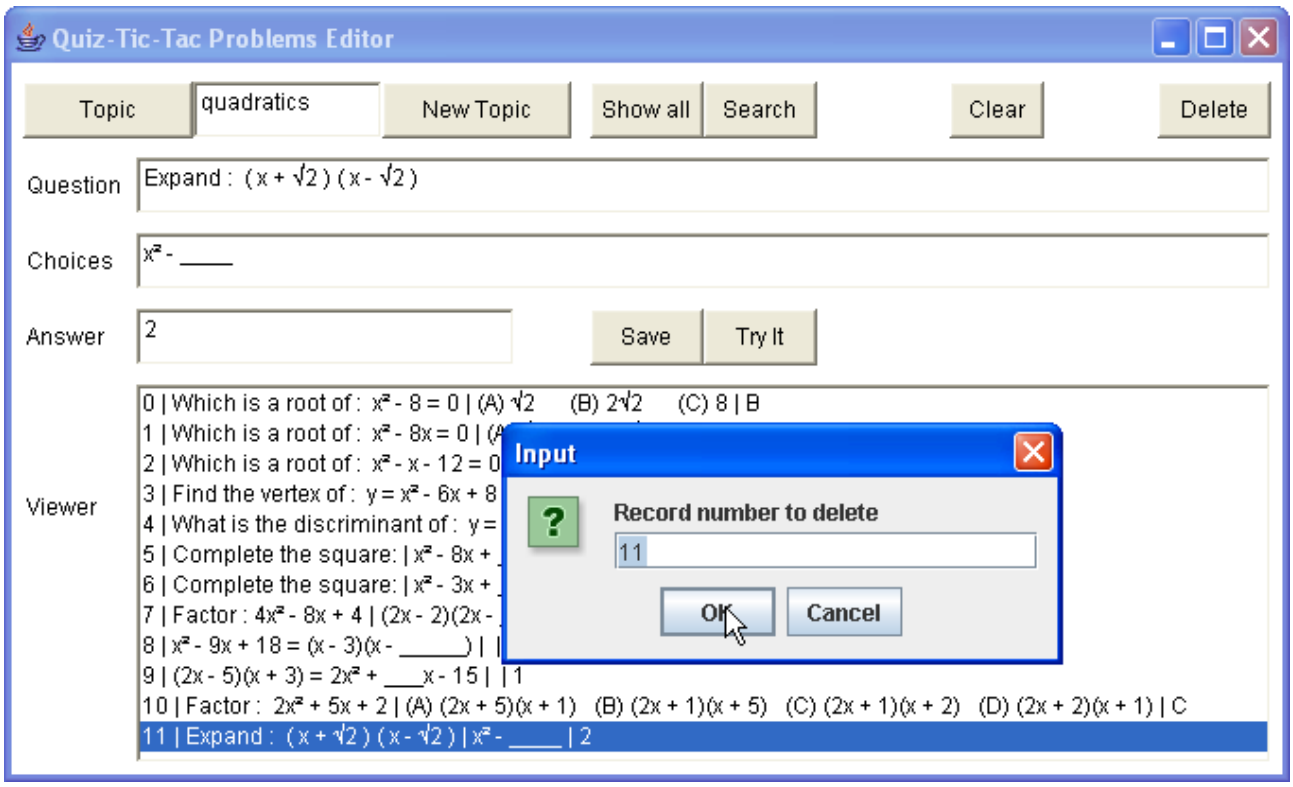
(E6) The teacher makes a more complicated problem, using ^r for square-root symbols (surds).



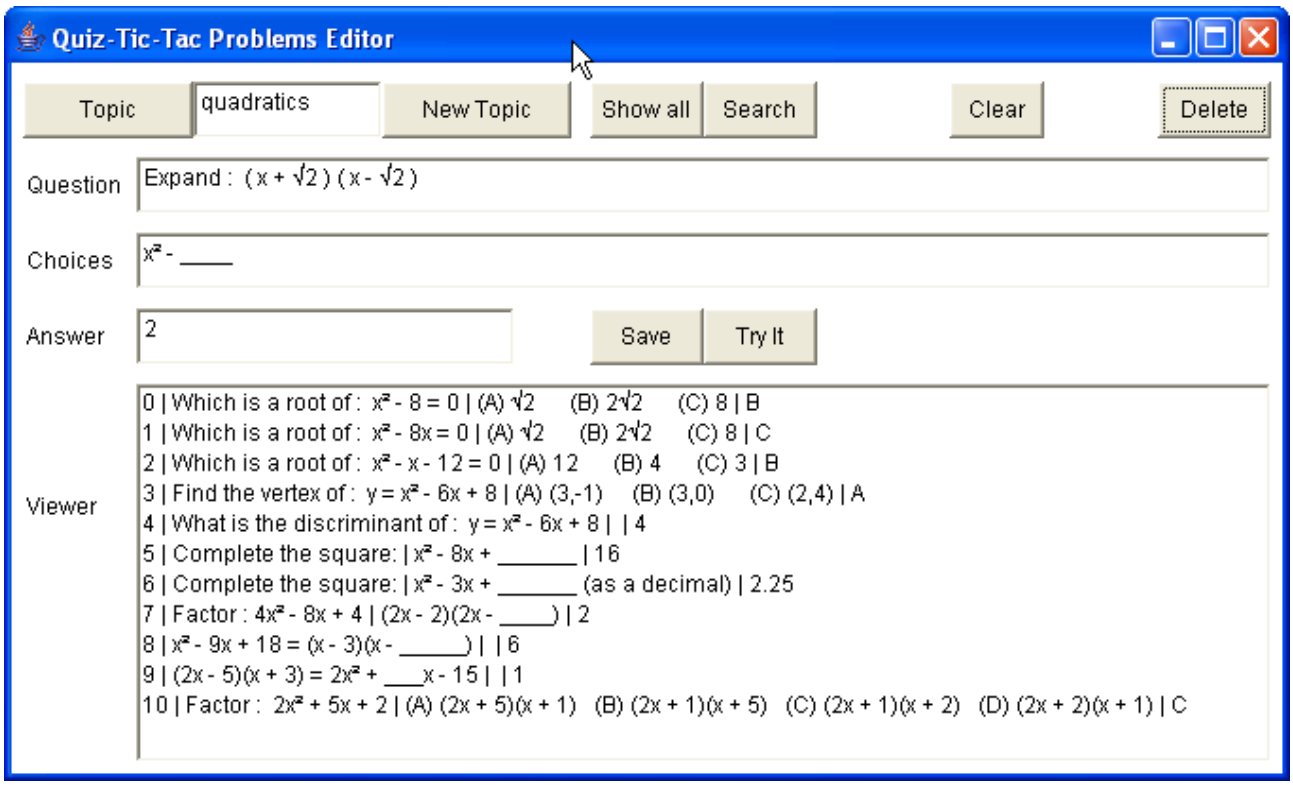
(E7) Notice that the Problem class has changed ^r2 into a proper square-root sign √2 .



(E8) The teacher changes her mind and decides to delete the problem. So she clicks on problem number 11, then clicks the [Delete] button.

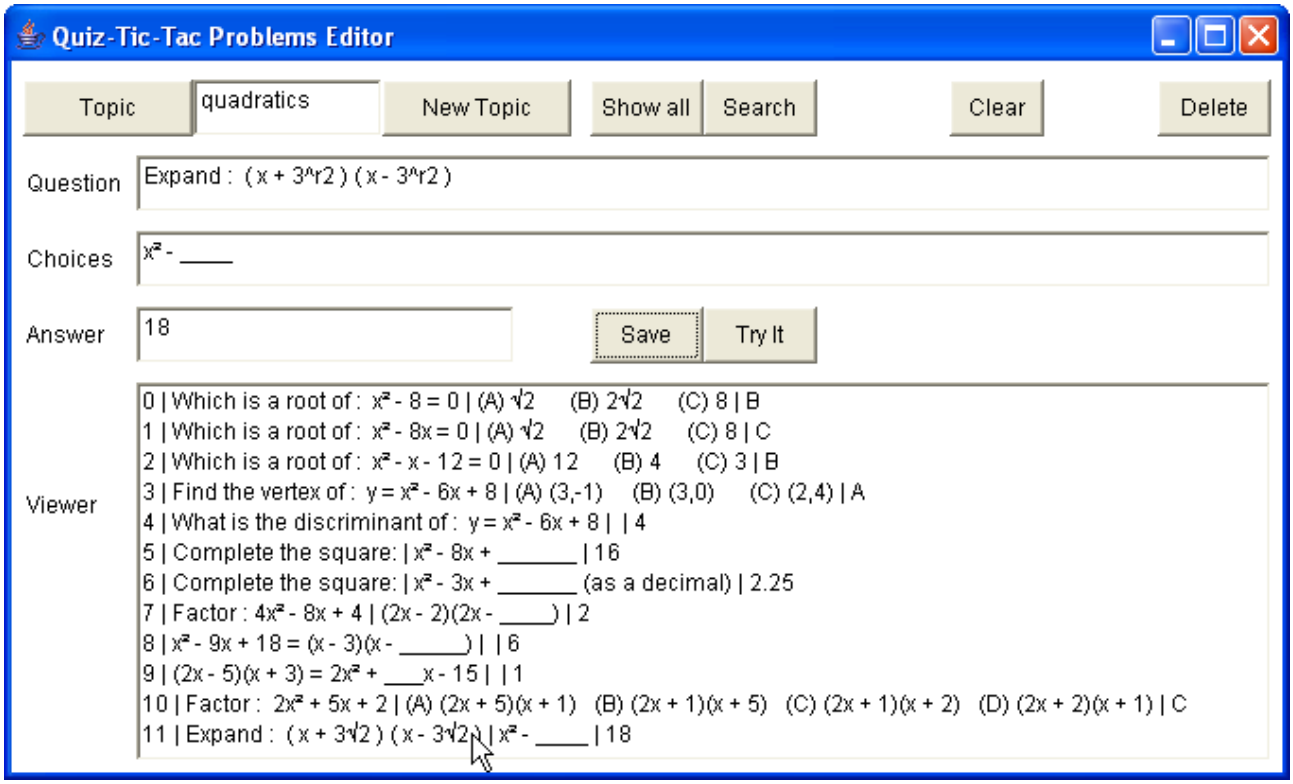


(E9) Now the problem has been erased from the file, but a copy remains in the editing boxes.

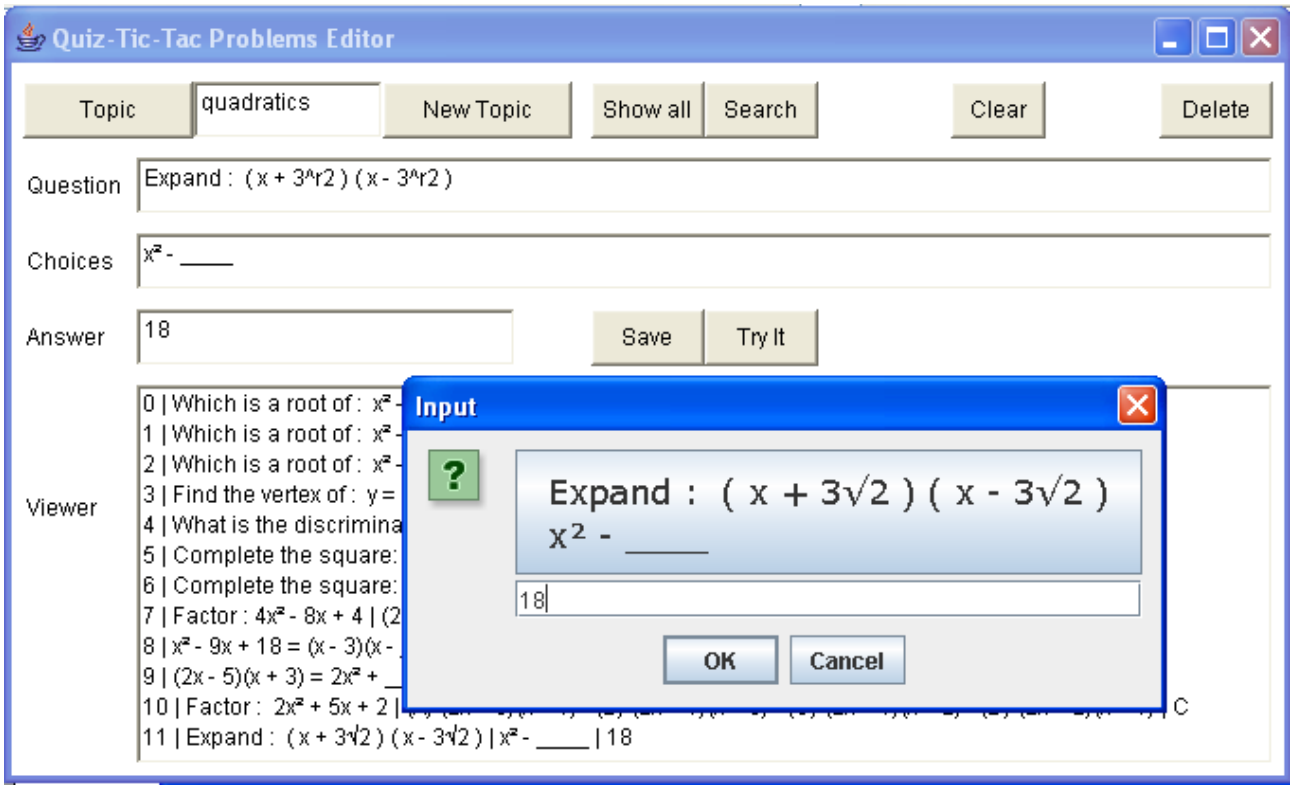


Pressing [Clear] would erase the contents of the editing boxes. Pressing [Save] would resave the problem into the file.

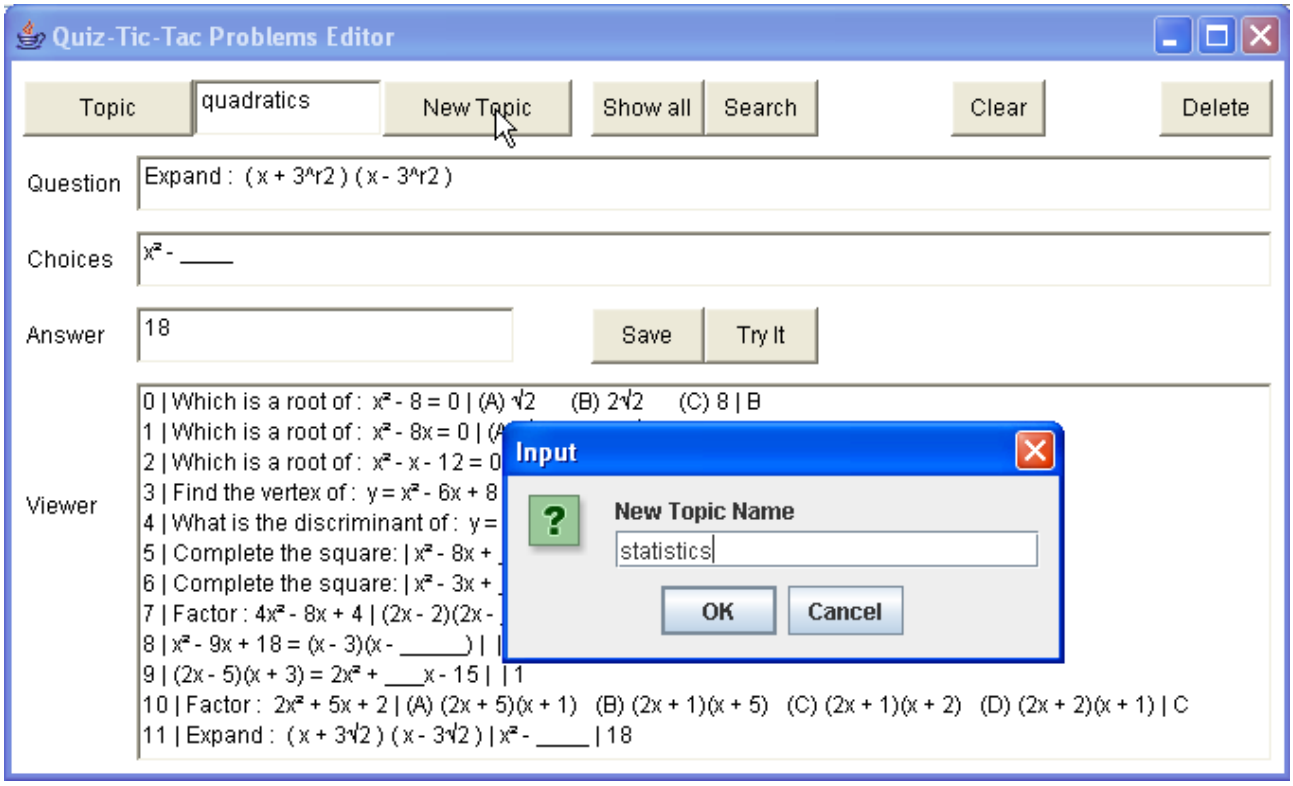
(E10) Now the teacher decides to change the problem and resave it.



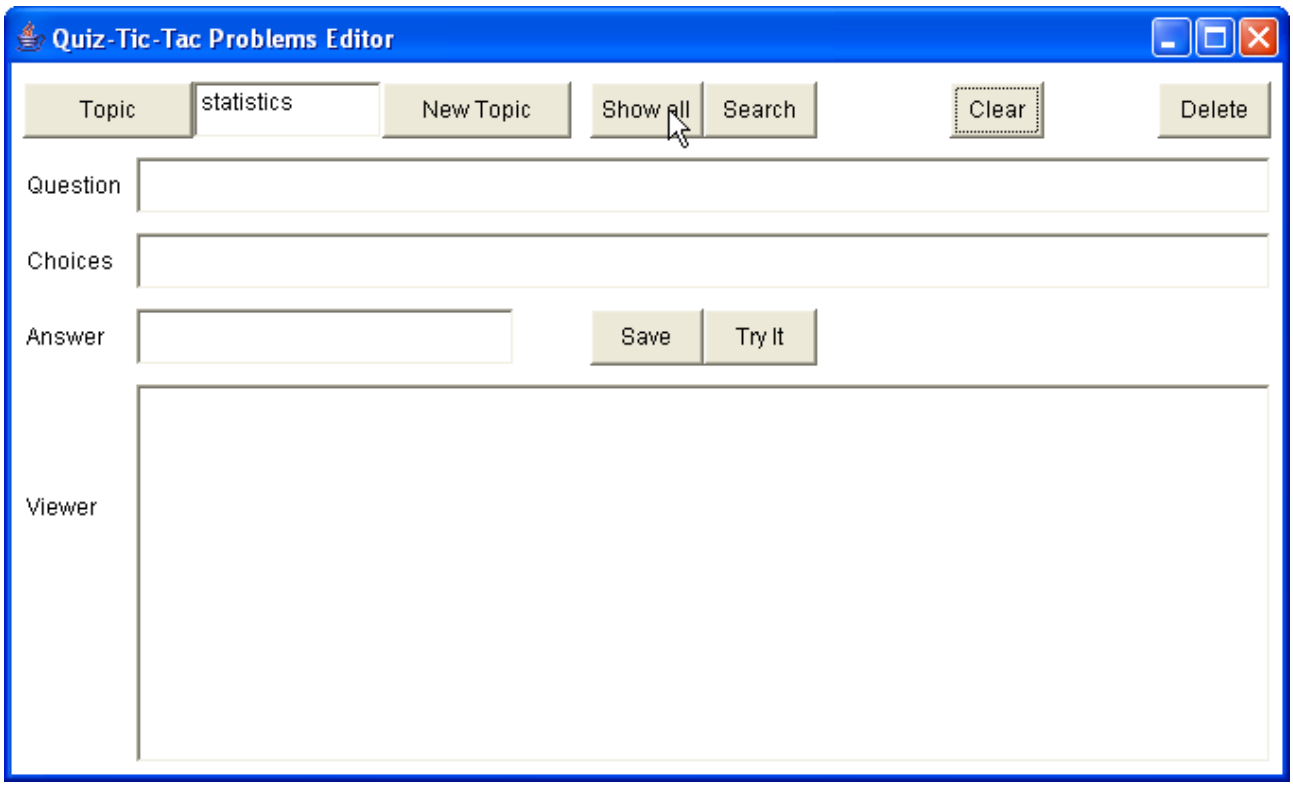
(E11) The teacher clicks [Try It] to see how the problem looks :



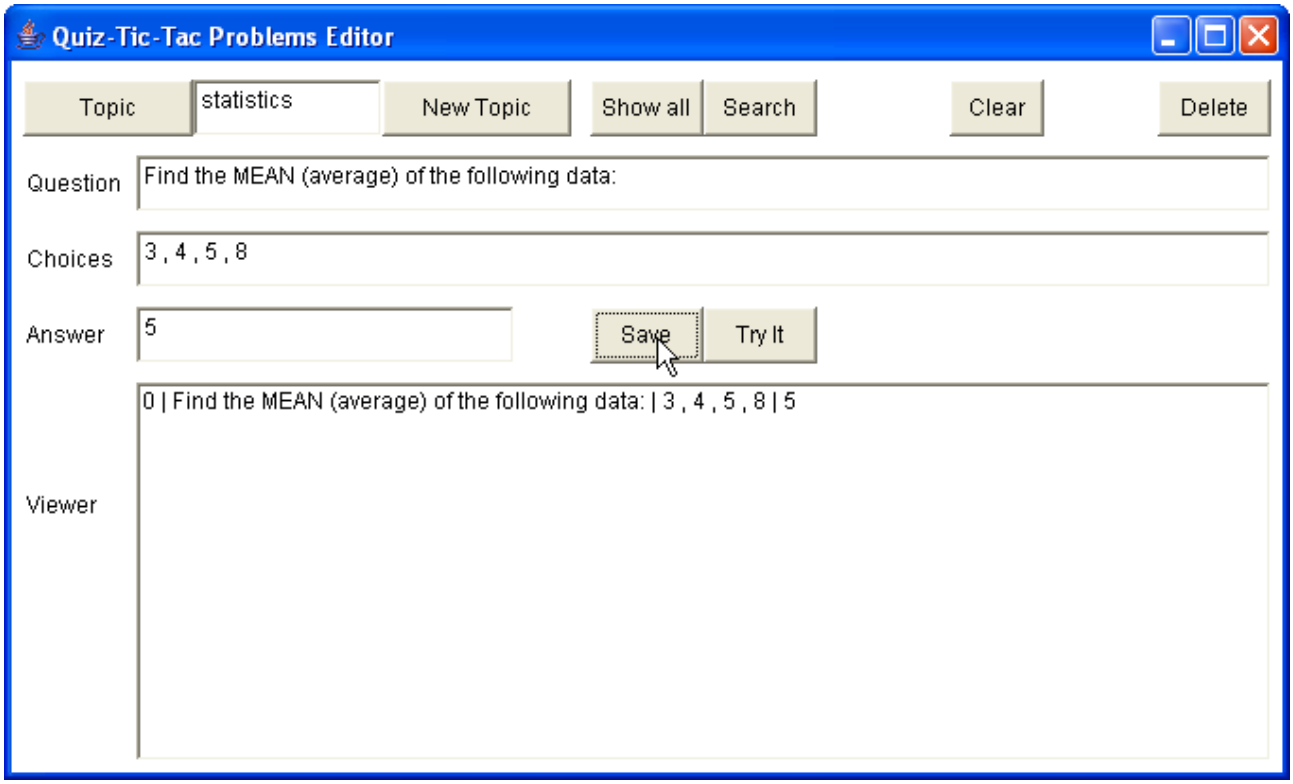
(E12) The teacher decides to create a new file for statistics.



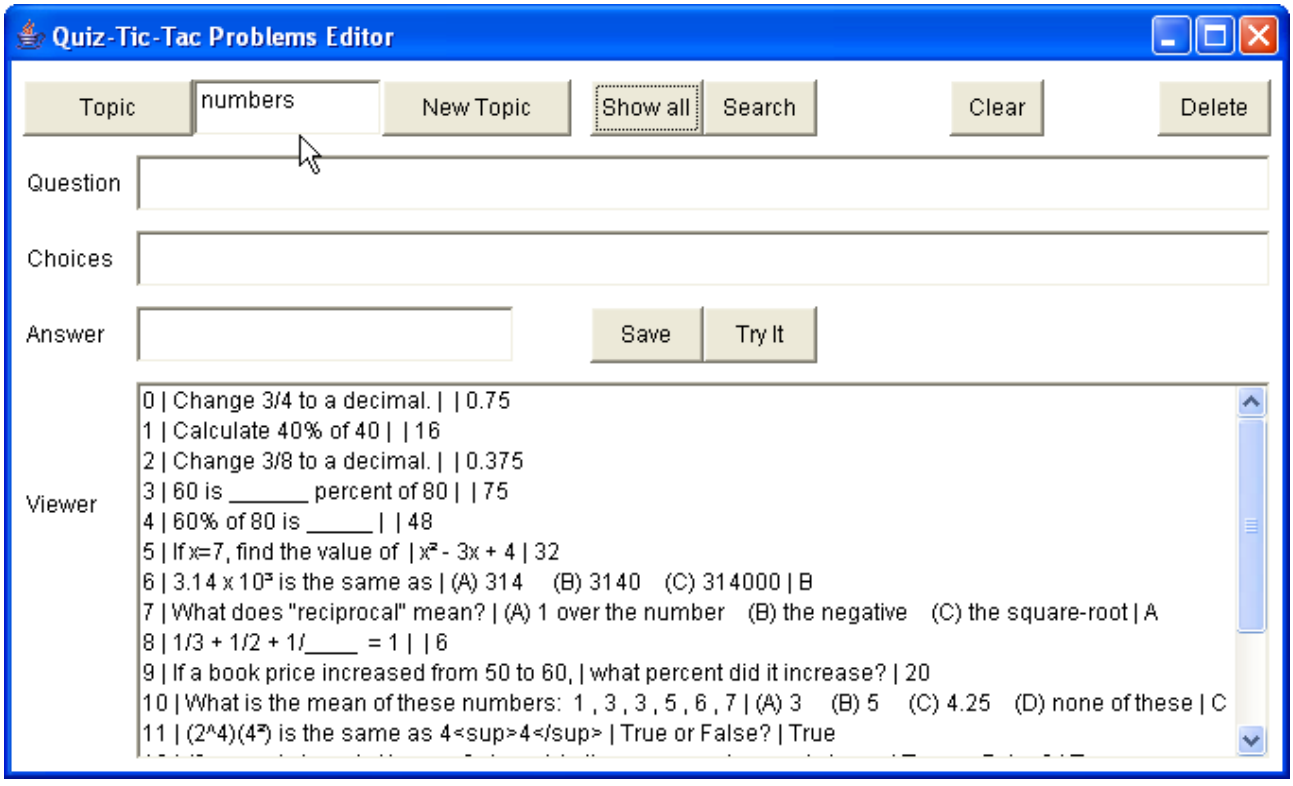
(E13) Then clicking [Show All] shows that the file is empty.



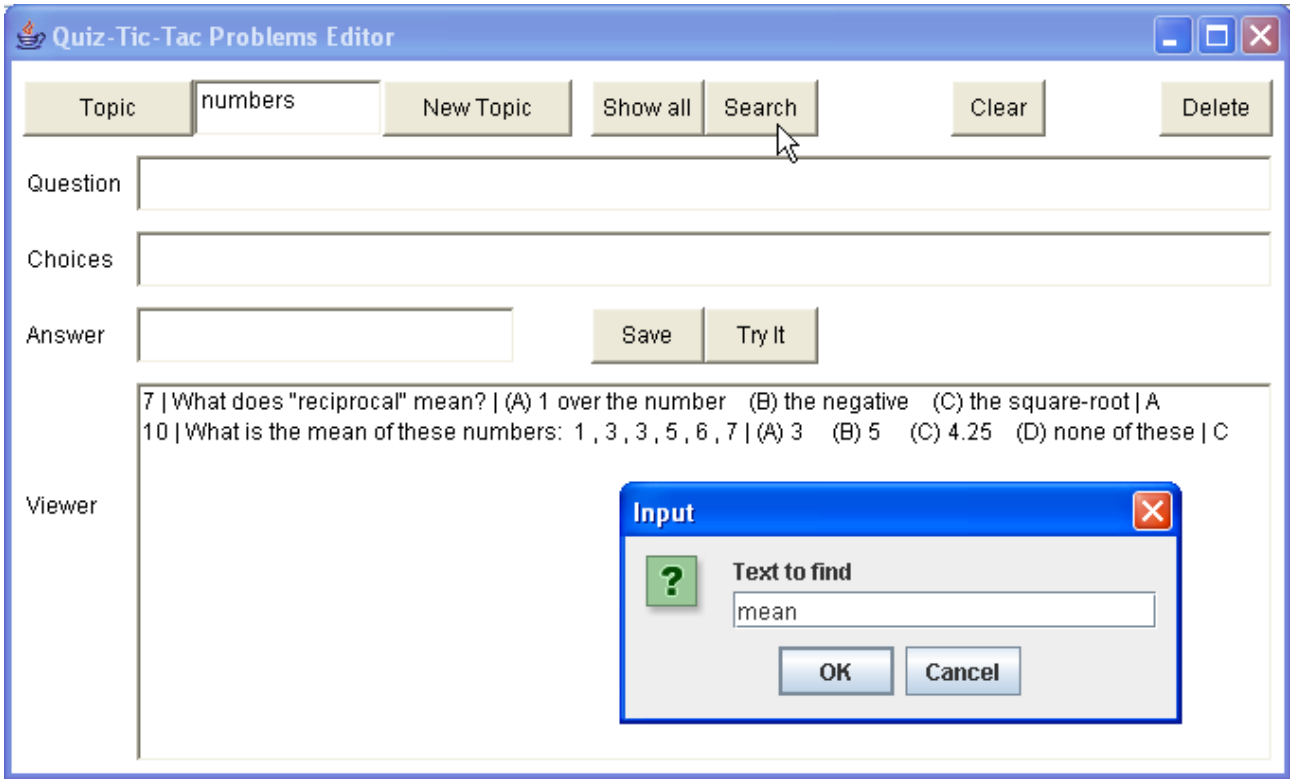
(E14) Now she adds a couple problems.



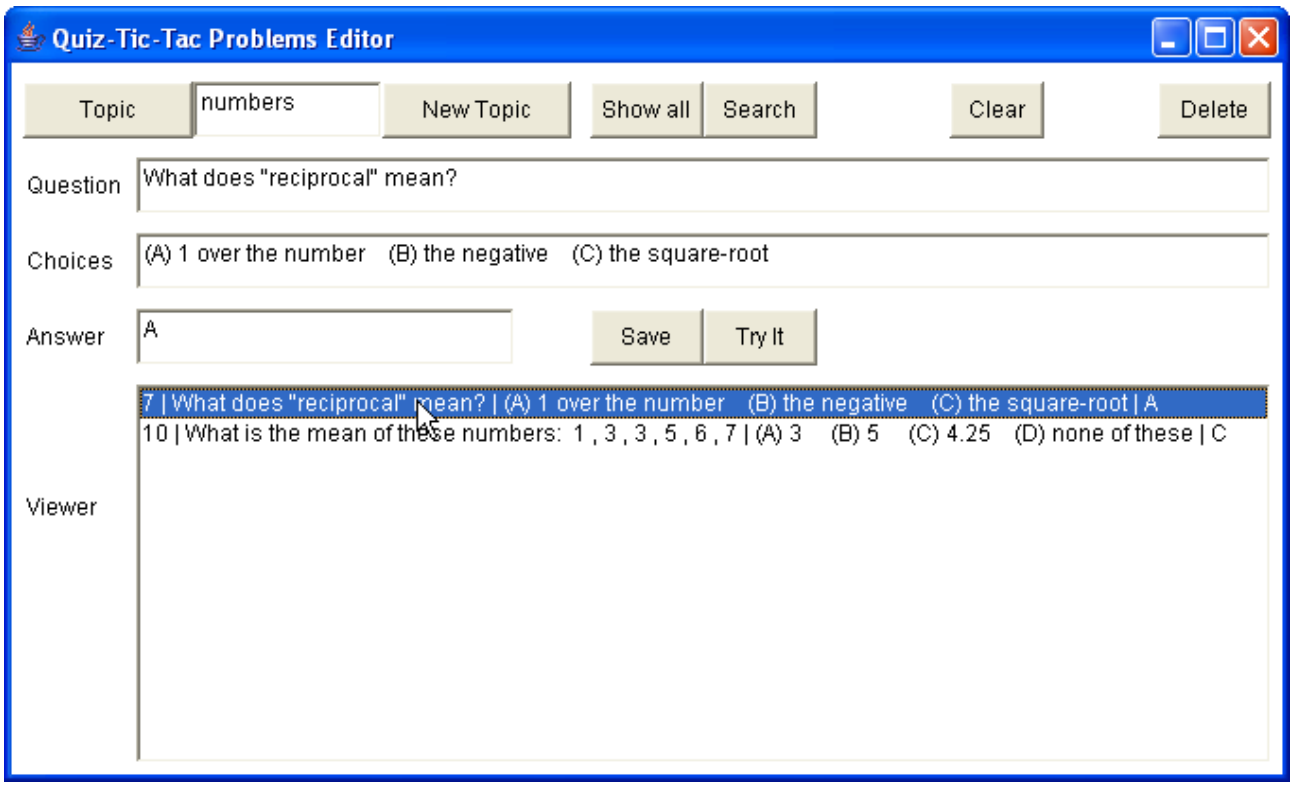
(E15) She decides to look back at another existing file - numbers.



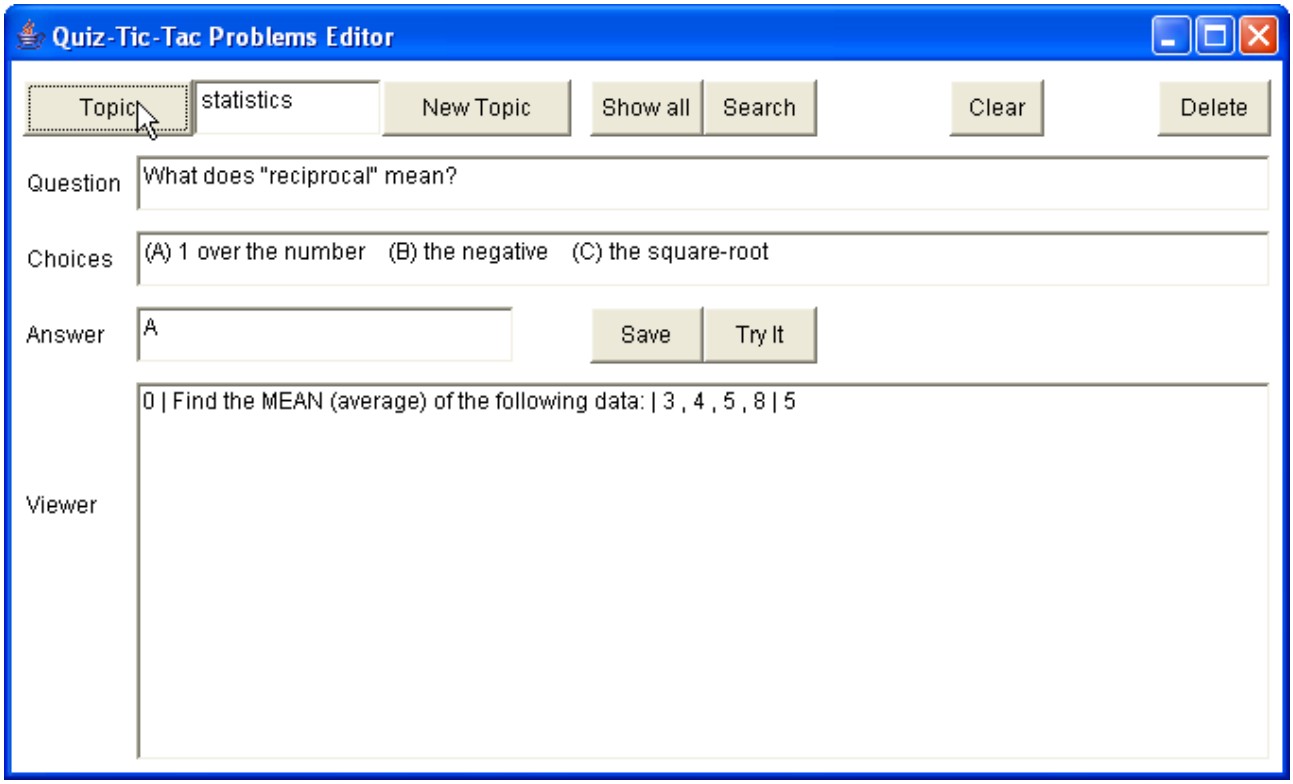
(E16) She decides to search for "mean", to see whether there are already some statistics problems in this file.



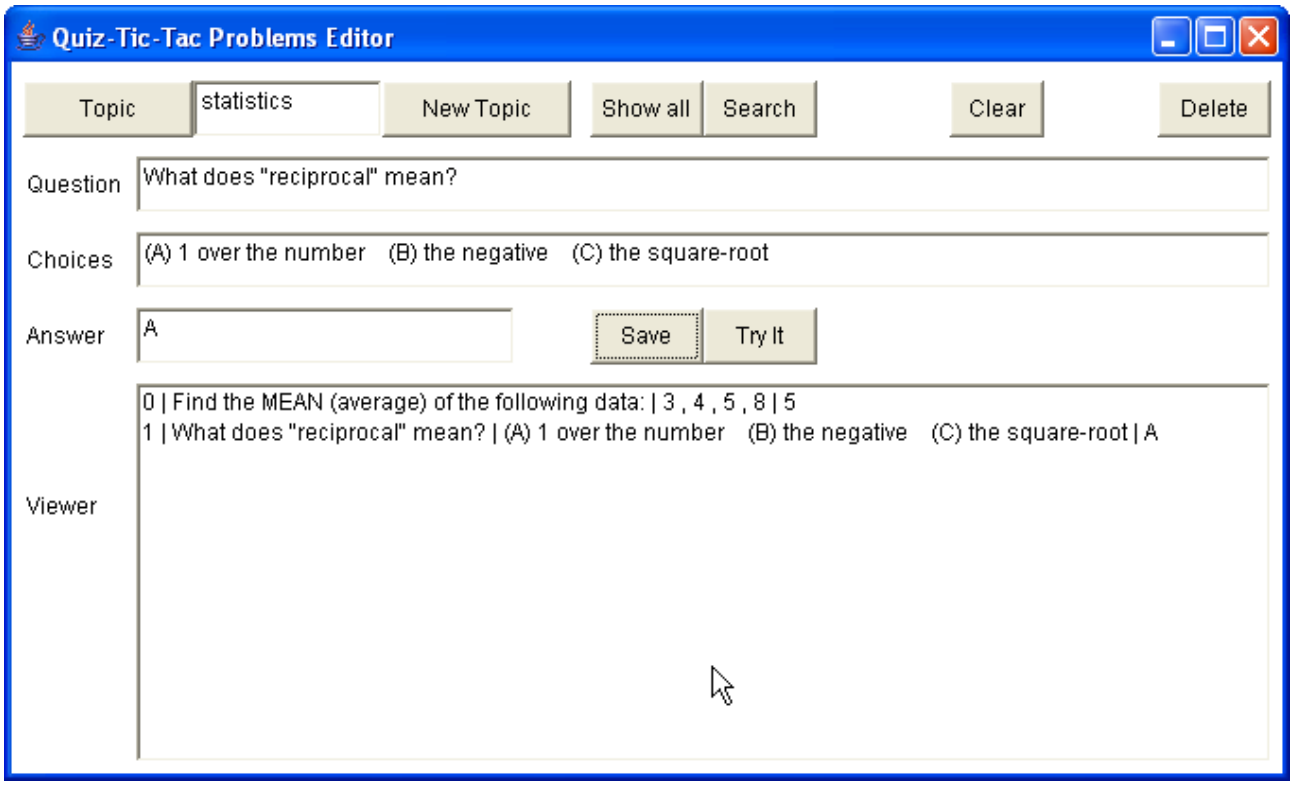
(E17) There are 2 mean problems here. She decides to copy these into the Statistics file. This is a bit cumbersome - click on a problem, then change the topic, then save the problem, then change the topic back to numbers.



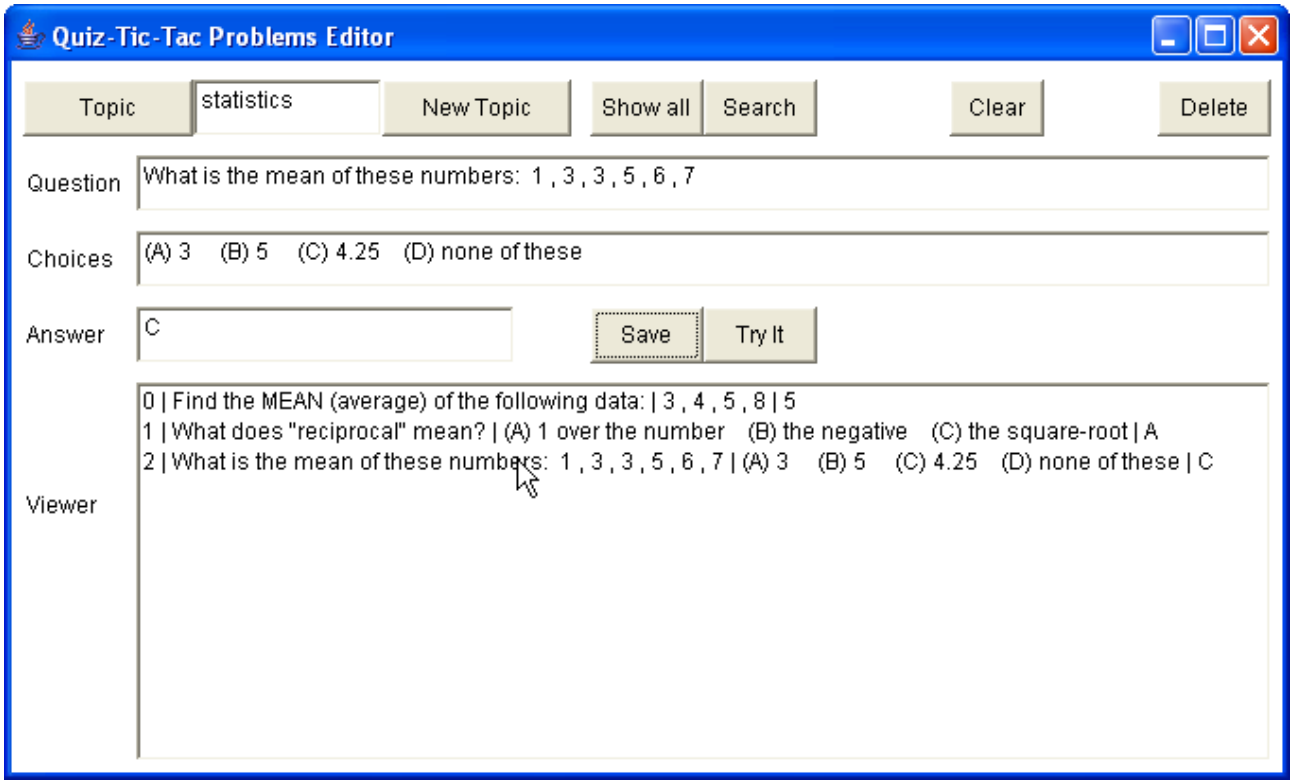
(E18) Now change topics to the Statistics file.



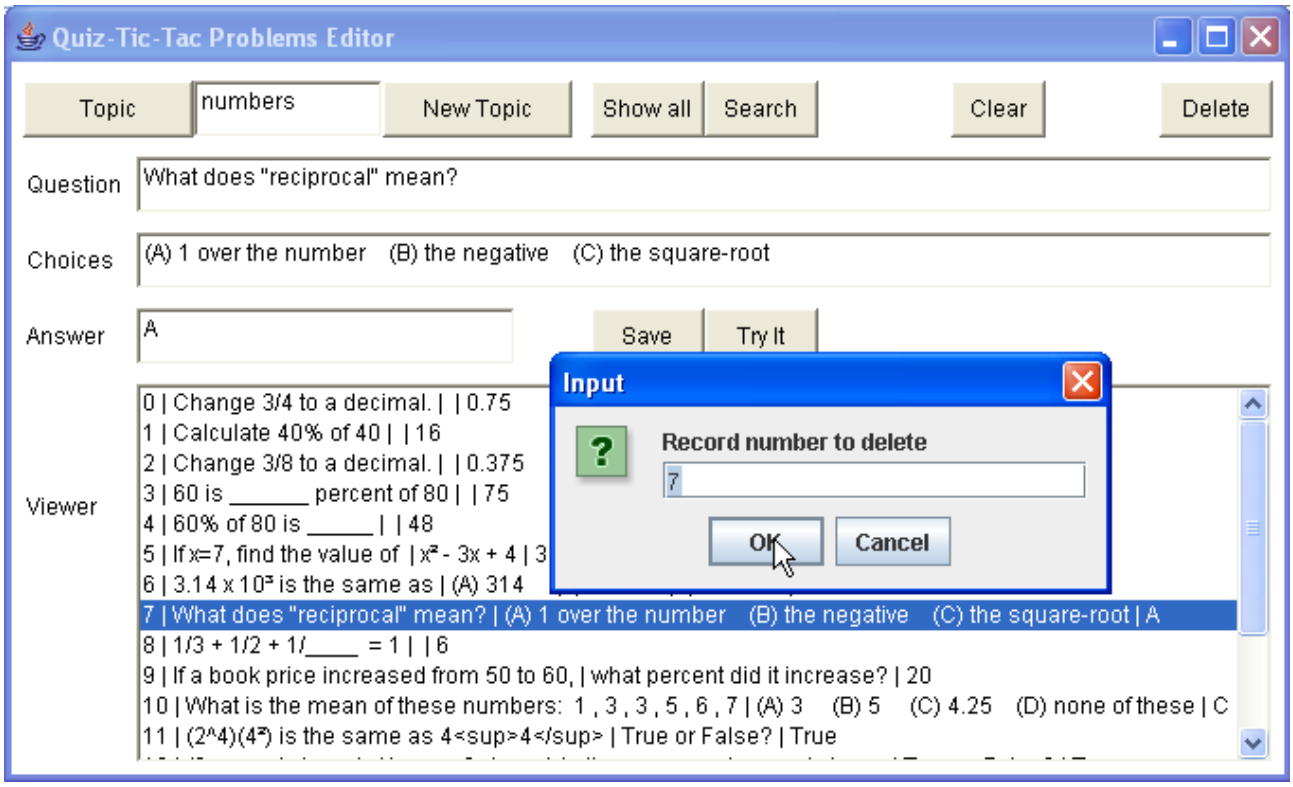
(E19) Click the [Save] button. The problem is saved into the file and the viewer is refreshed.



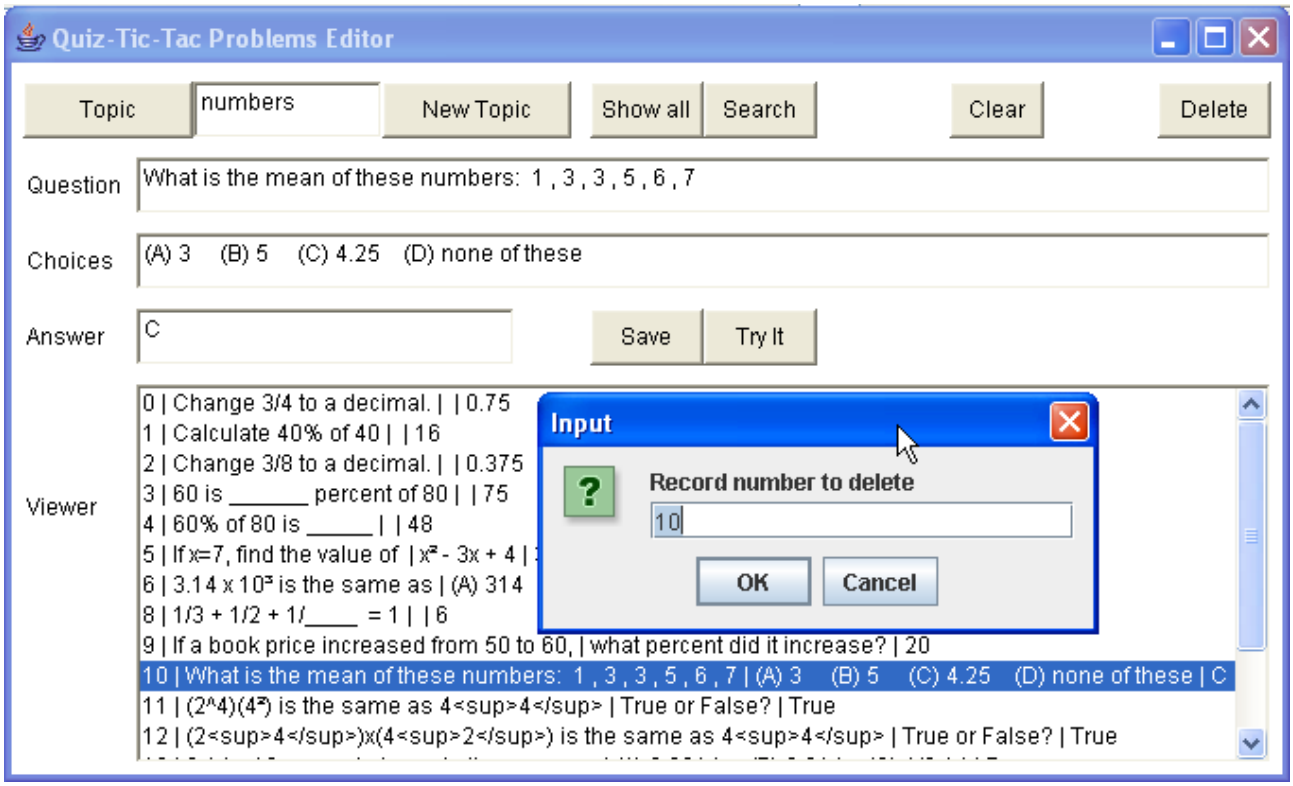
(E20) Change back to Numbers and repeat to copy the second "mean" problem.



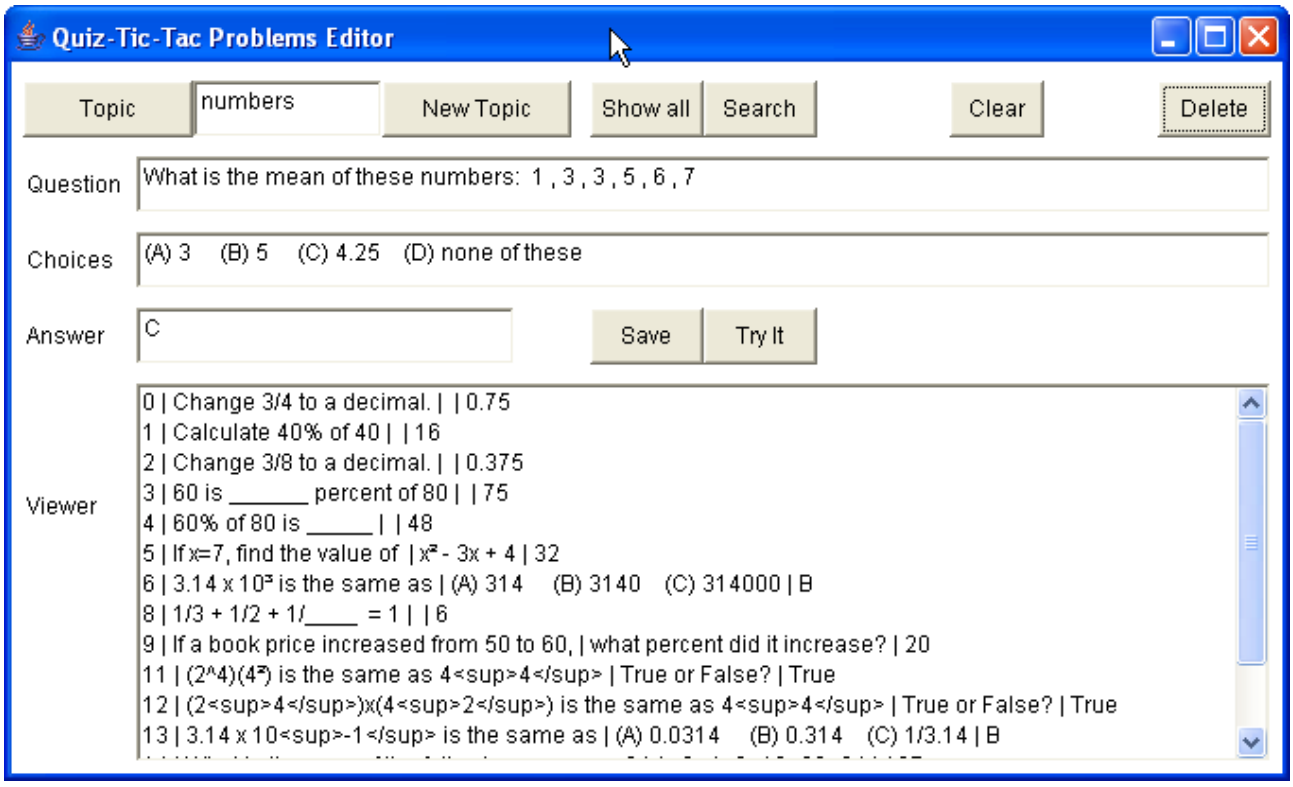
(E21) Now she can switch back to Numbers and [Delete] the "mean" problems.



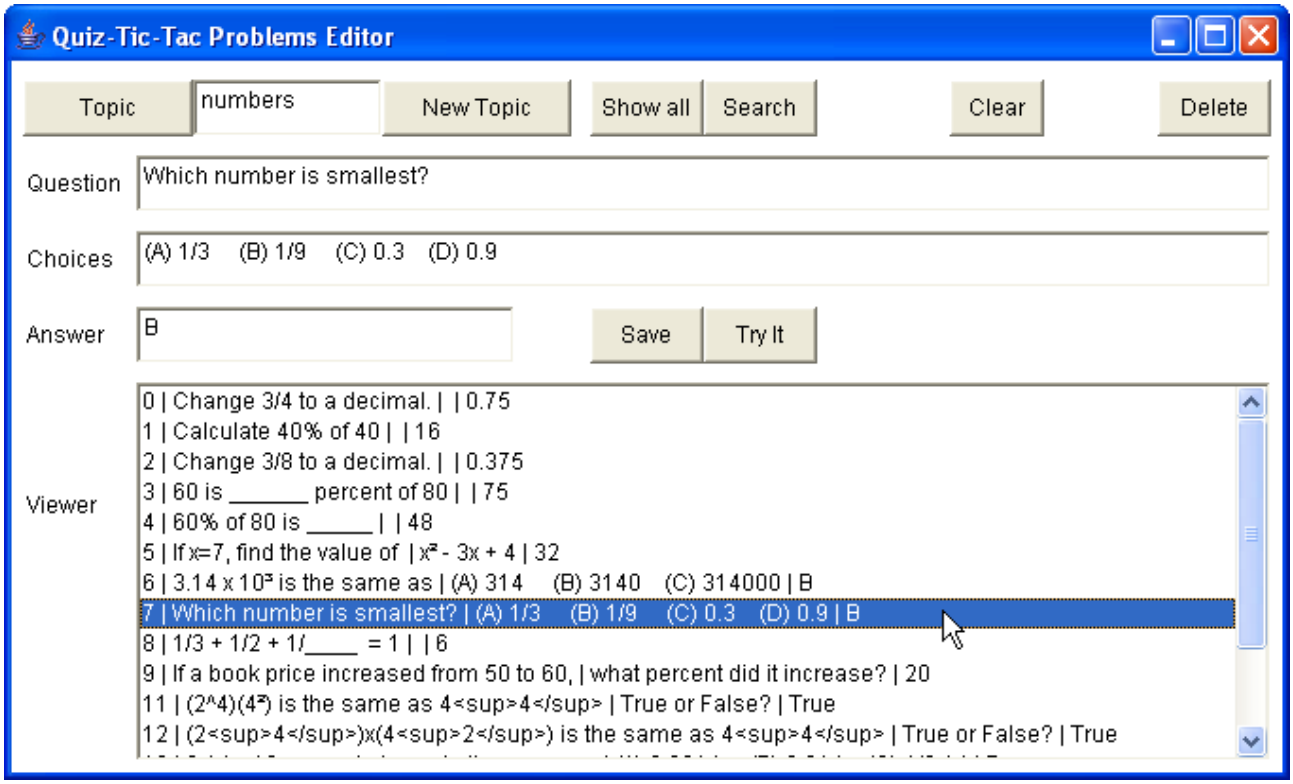
(E22) And delete problem #10.



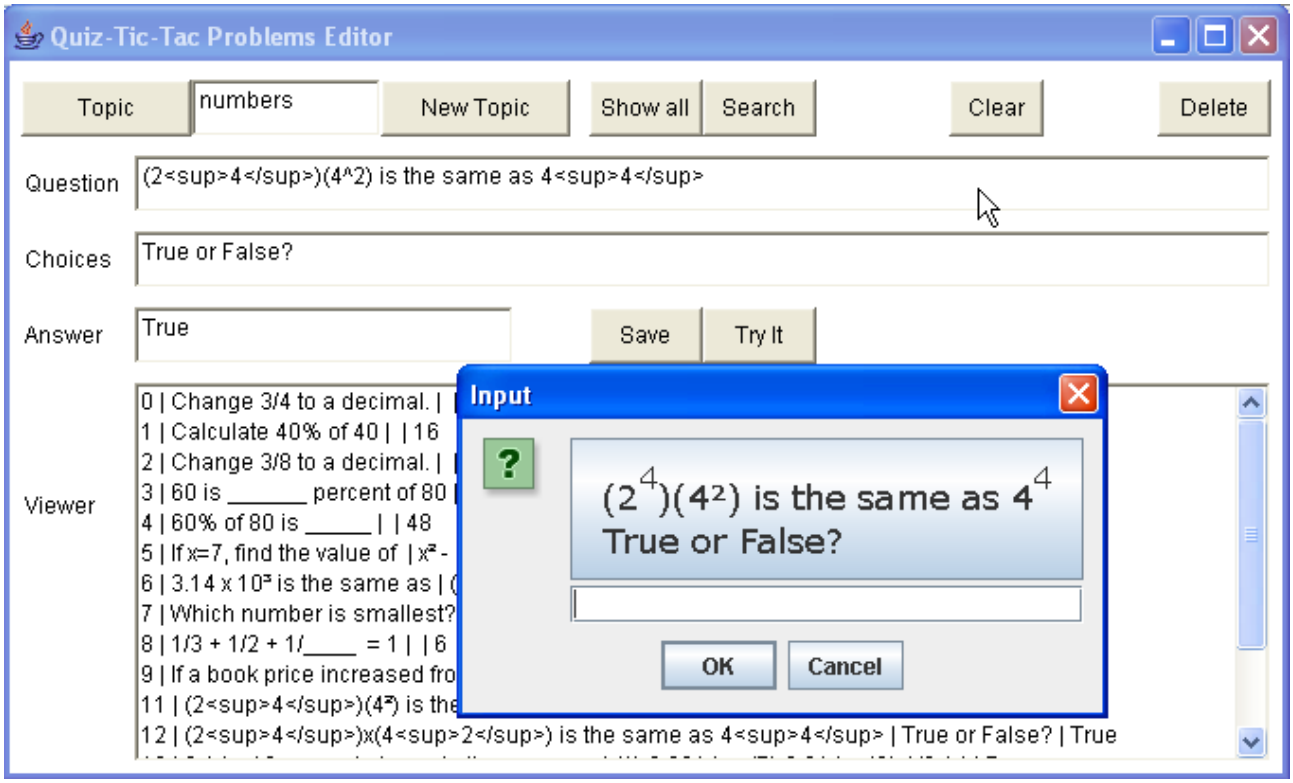
(E23) Now the file has a couple of empty records at positions #7 and #10.



(E24) The next new problem in the Numbers file will be saved automatically in record #7.



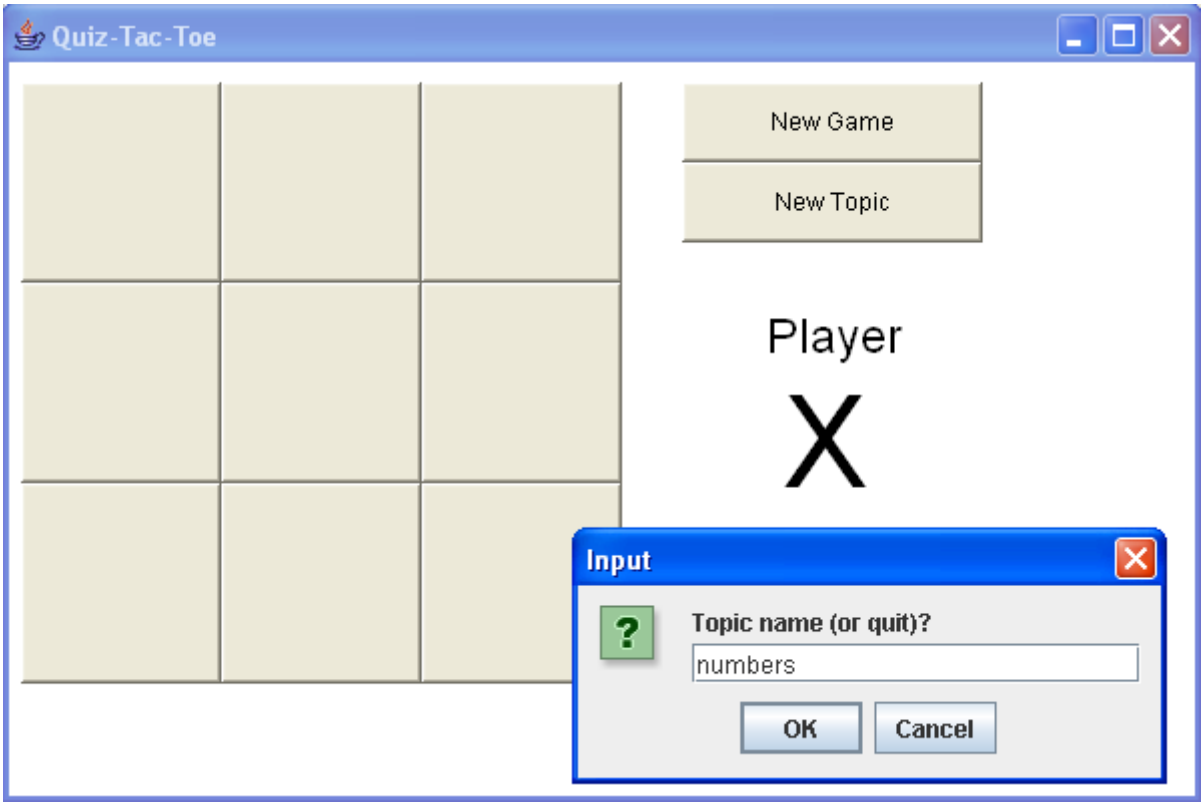
(E25) Notice that HTML formatting tags can be used in the problems to make other exponents besides squared and cubed, by using the <sup> superscript tag. This is automatically supported by the JButton control used in the output method.



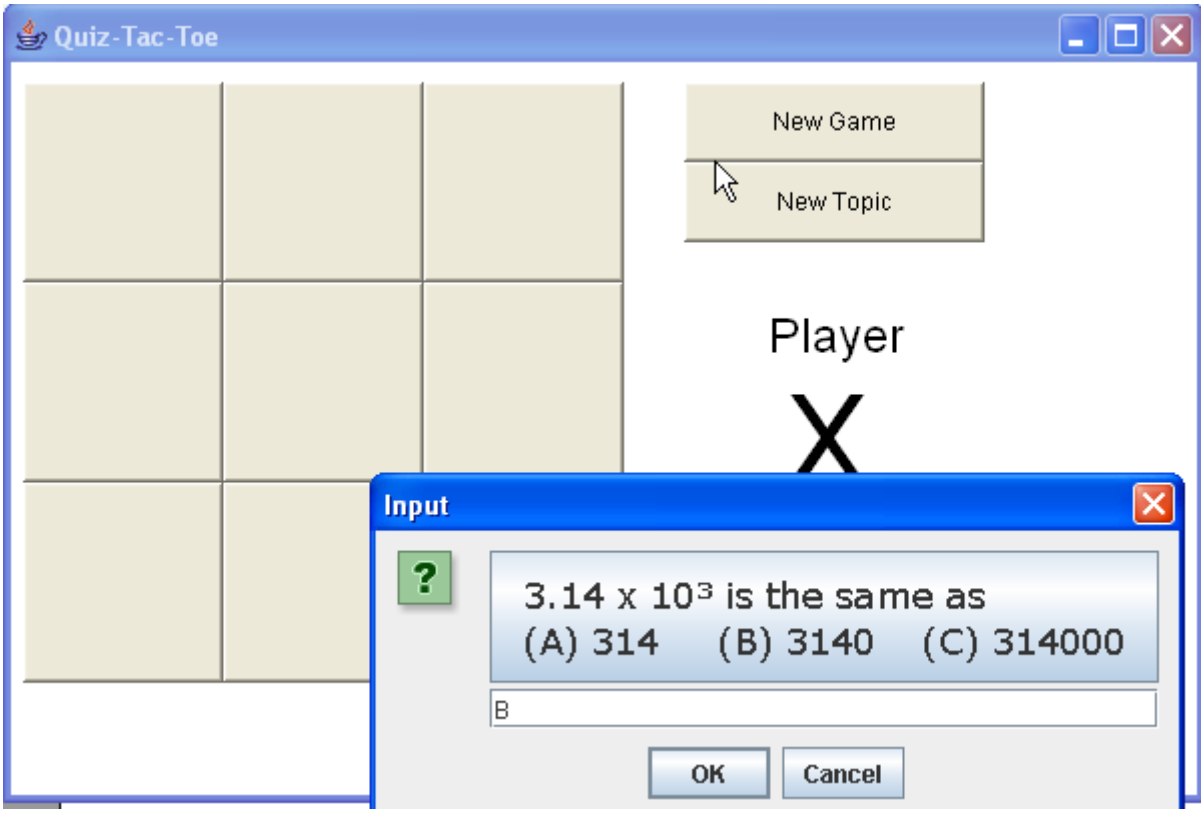
Notice it's probably best to not mix these up, as the ^2 squared looks different than a ². But Ms Fizz didn't really like the idea of typing HTML tags. Maybe a different teacher will like it.

Game - Typical Run for Students' Game Interface

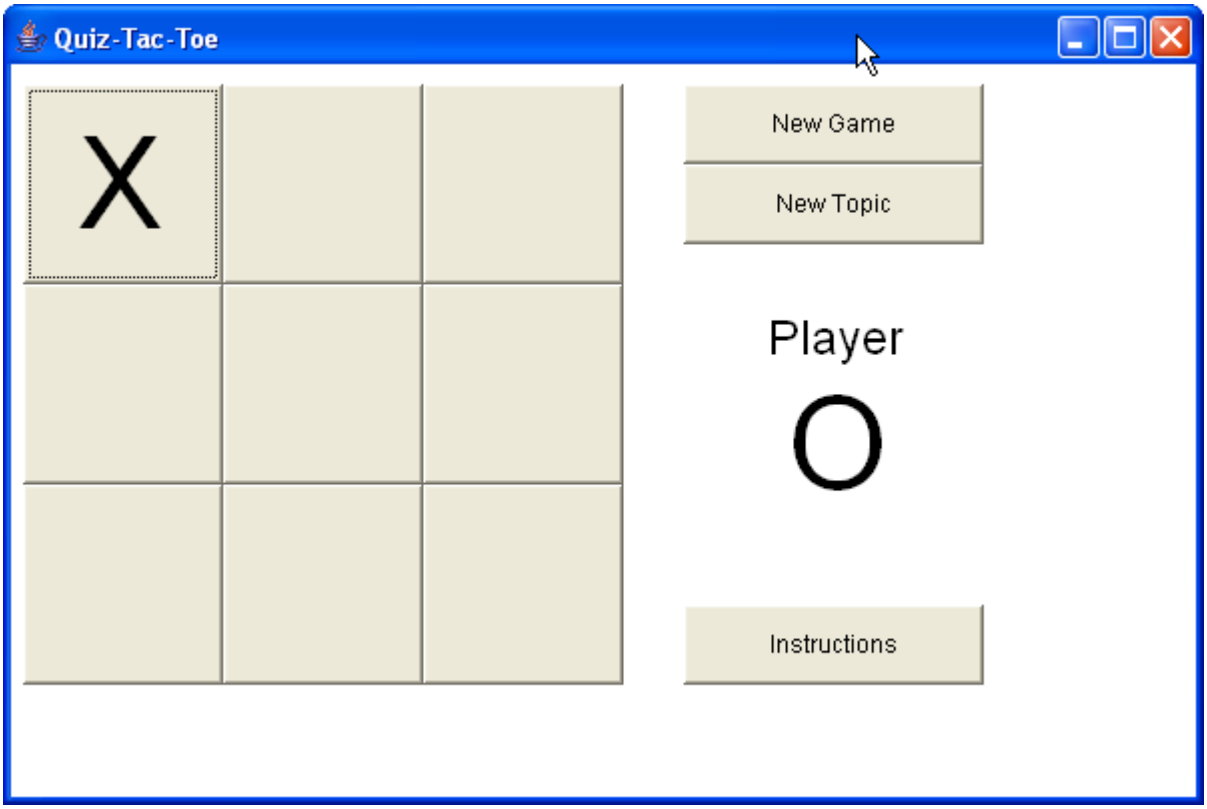
(G1) Now there are a couple topic files, so students can play the game. When the game starts, they must type the name of a topic (numbers, quadratics, or statistics).



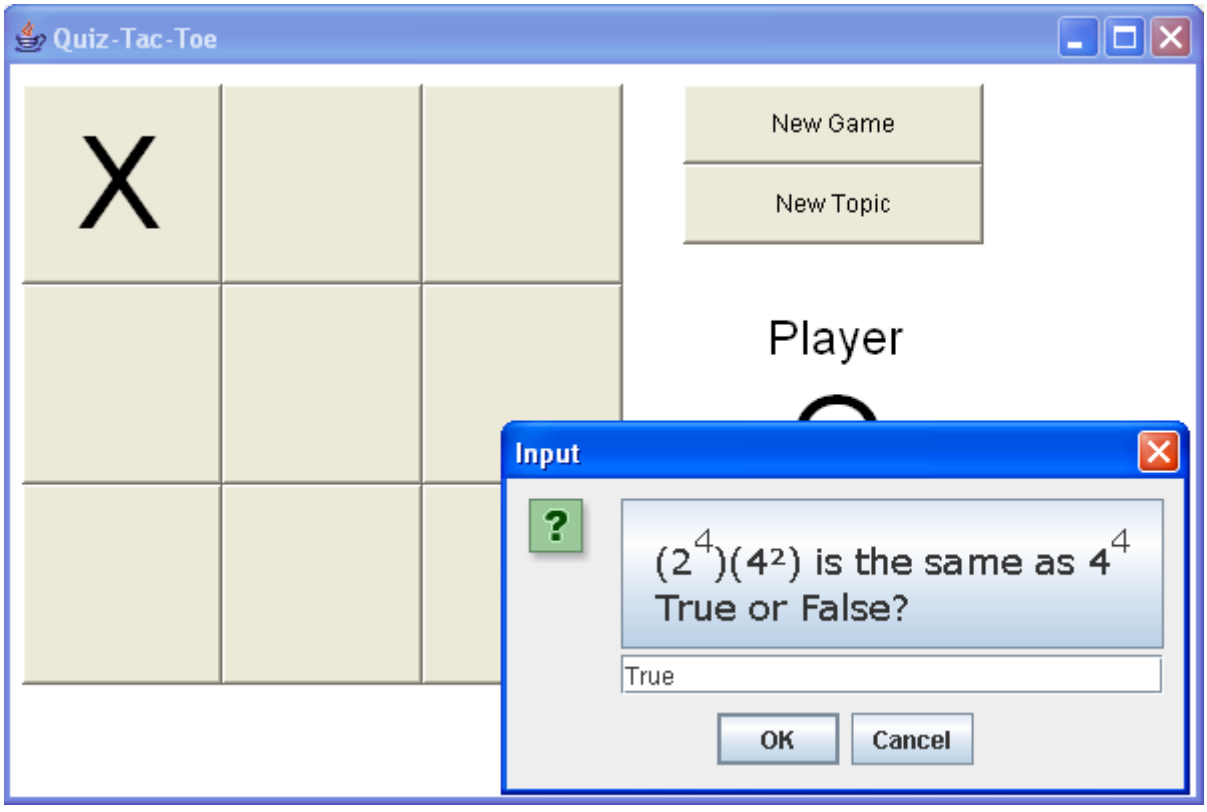
(G2) Now the board is blank and it's X's turn. X clicks on the top-left corner.



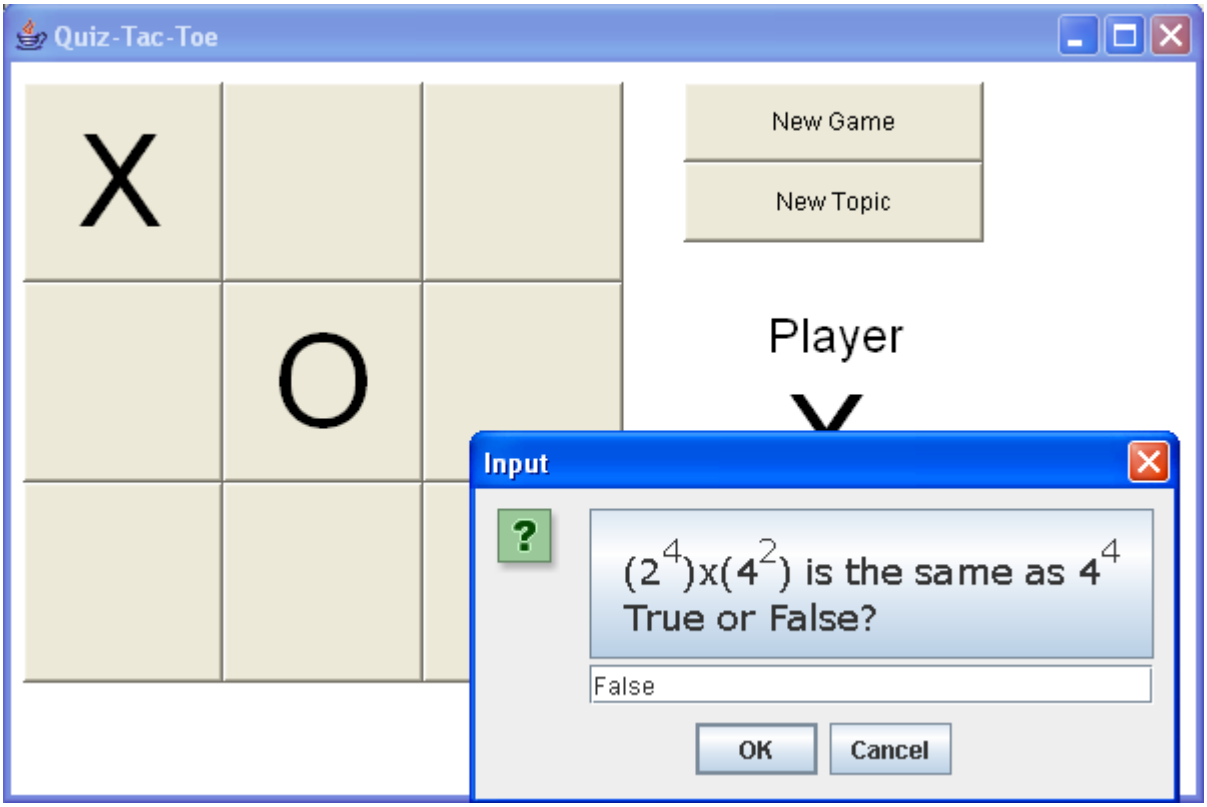
(G3) X answered correctly, so an X is drawn in the top-left square.



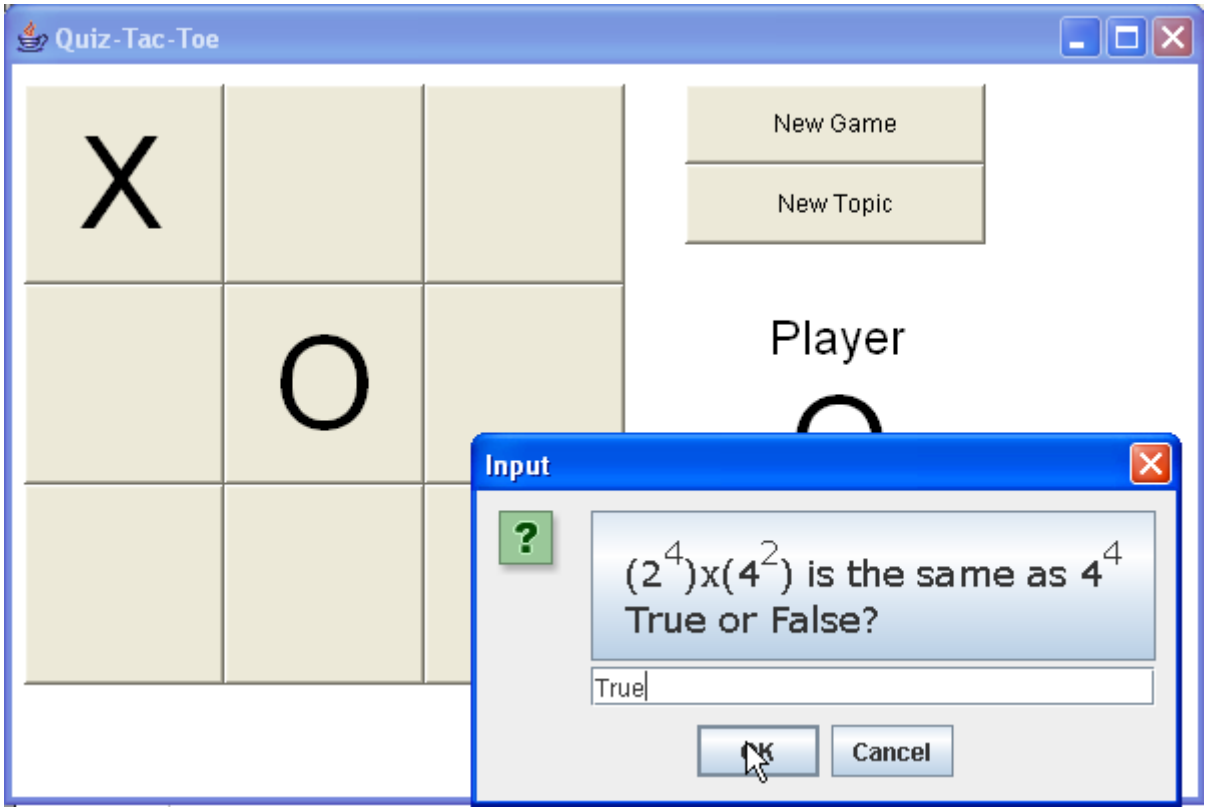
(G4) Now O tries for the middle square.



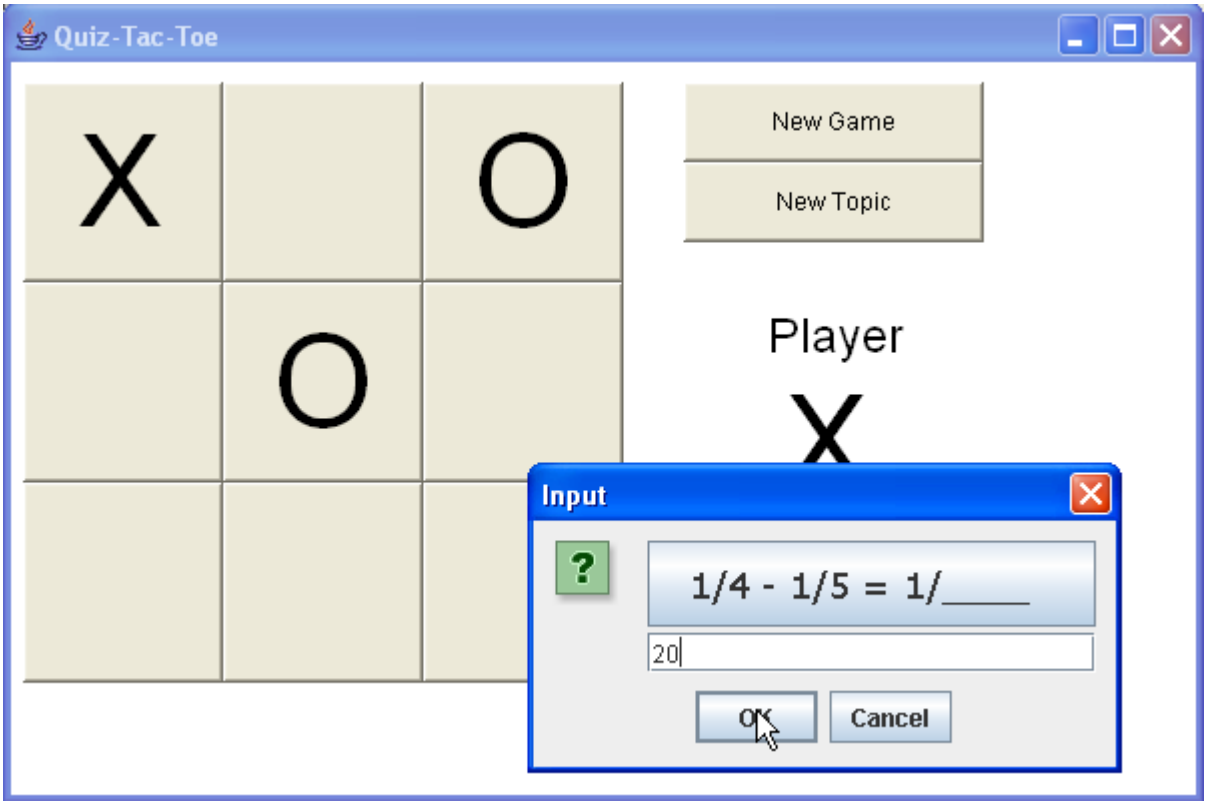
(G5) O's answer is correct, so he gets an O in the middle square. Now X tries the top-right corner.



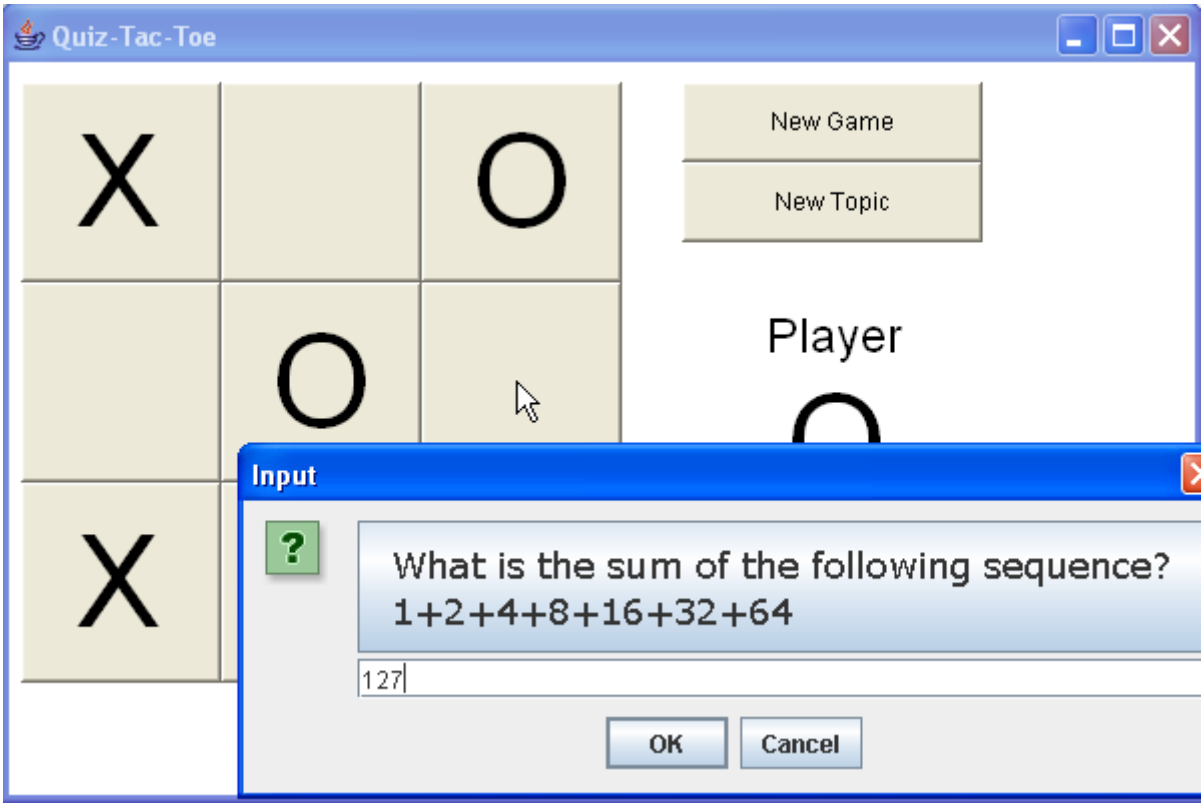
(G6) X's answer was incorrect, so he doesn't get an X in the corner. Now O tries the same square - pretty easy, since it's a true/false question.



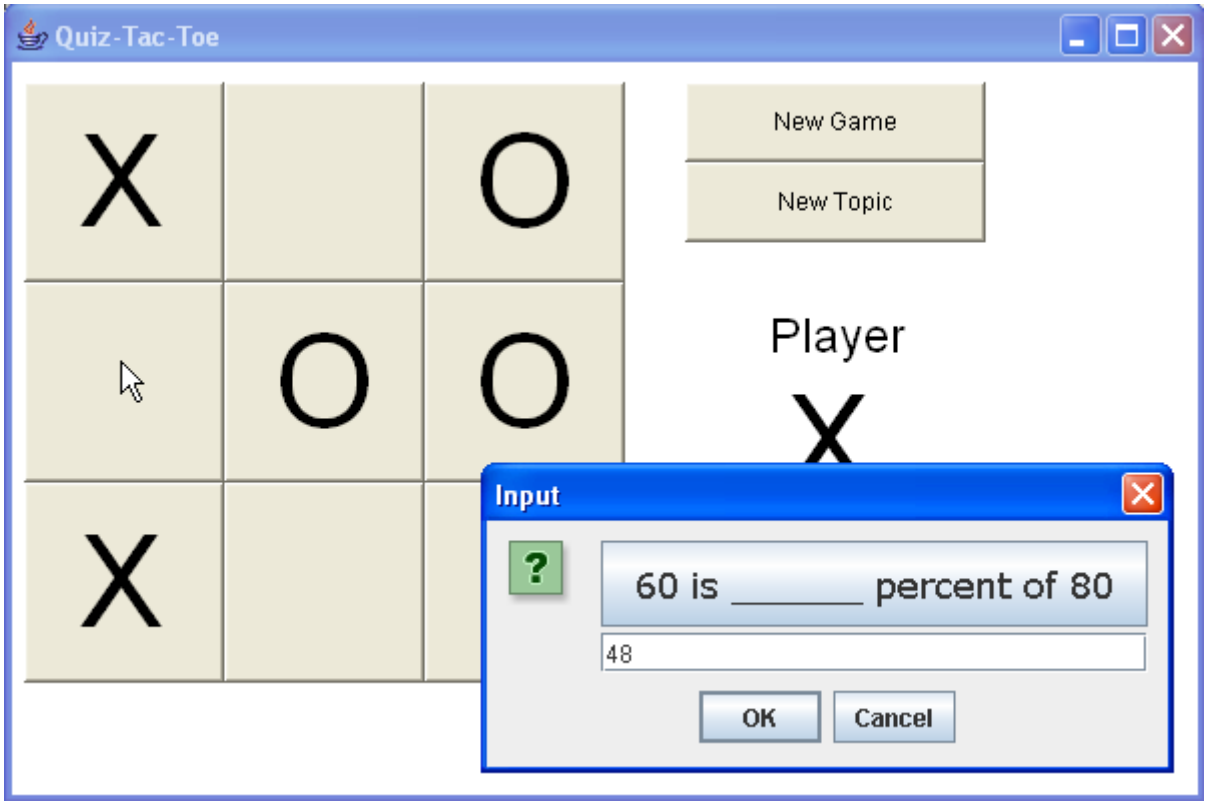
(G7) O got the right answer, so he gets an O in the corner. Now X goes for the bottom left.



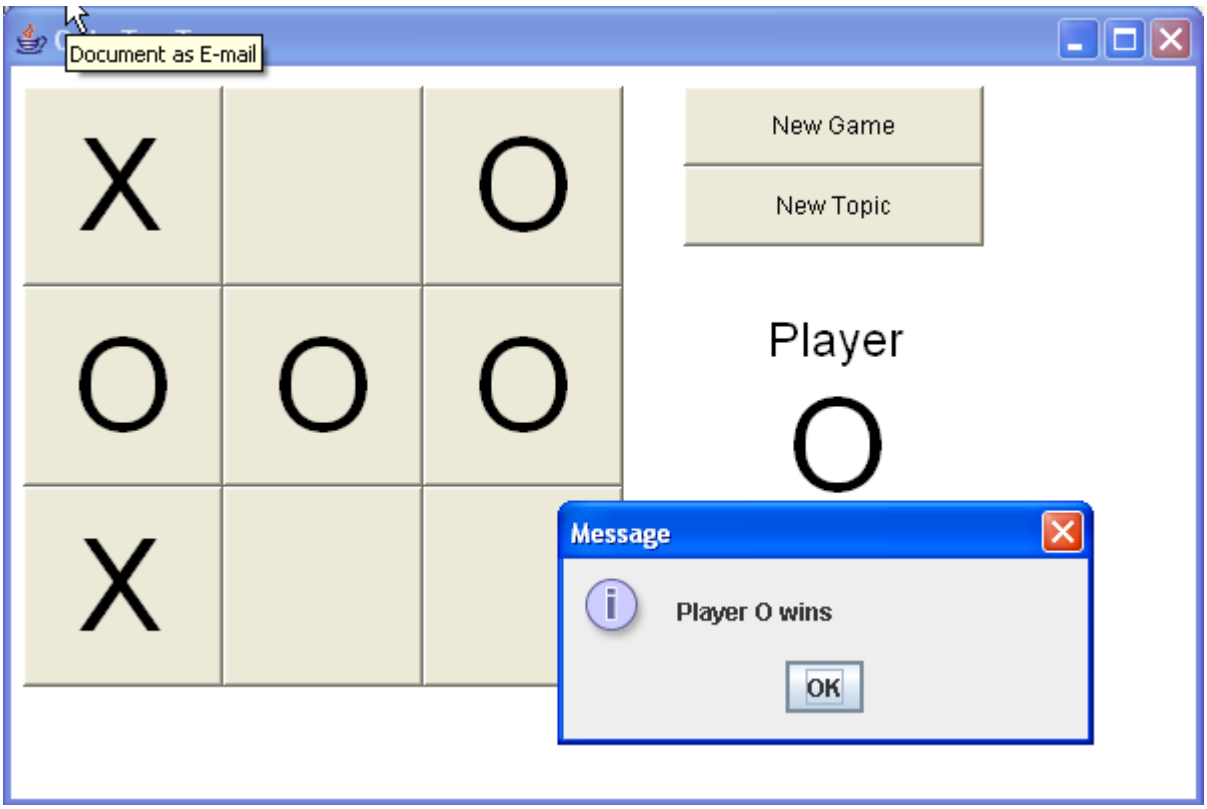
(G8) X answered correctly. Now O tries the right middle.



(G9) O had the correct answer and gets the square. But X goes for the middle-left square.

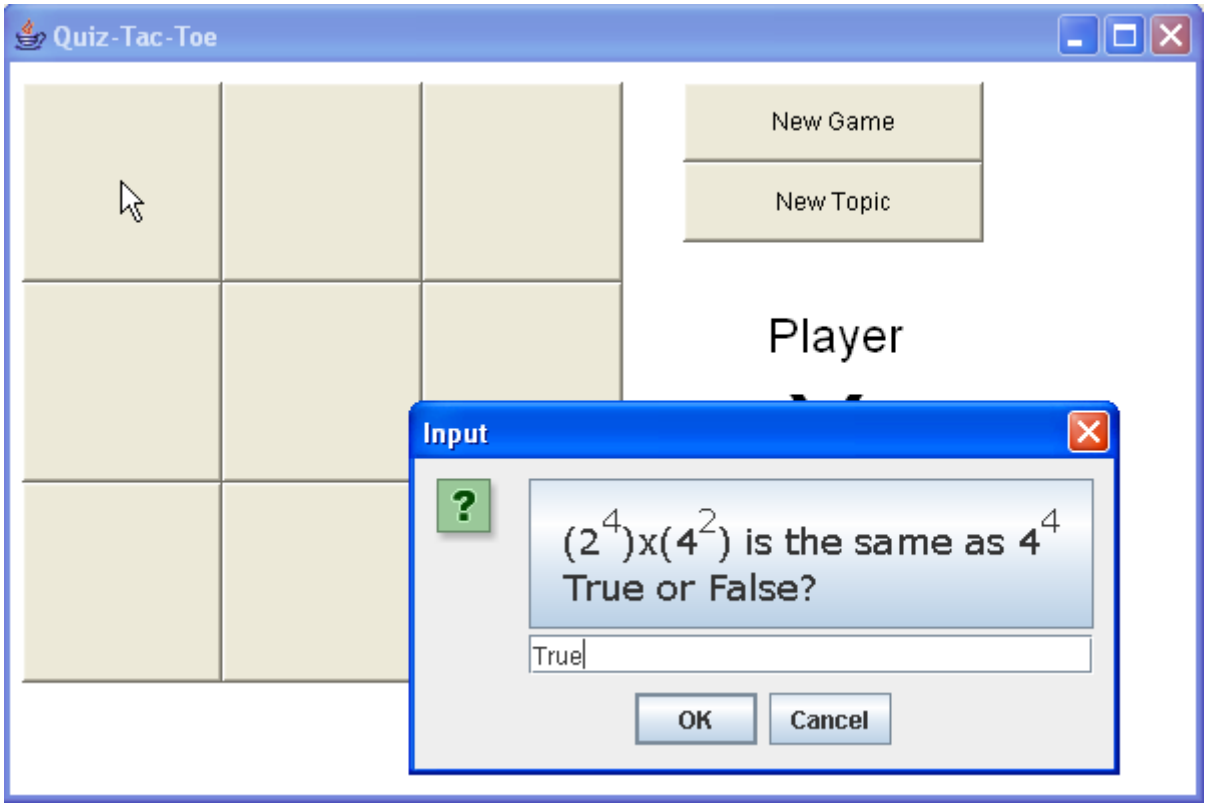


(G10) But the answer was wrong. Now O tries for the same square, and gets the answer correct.

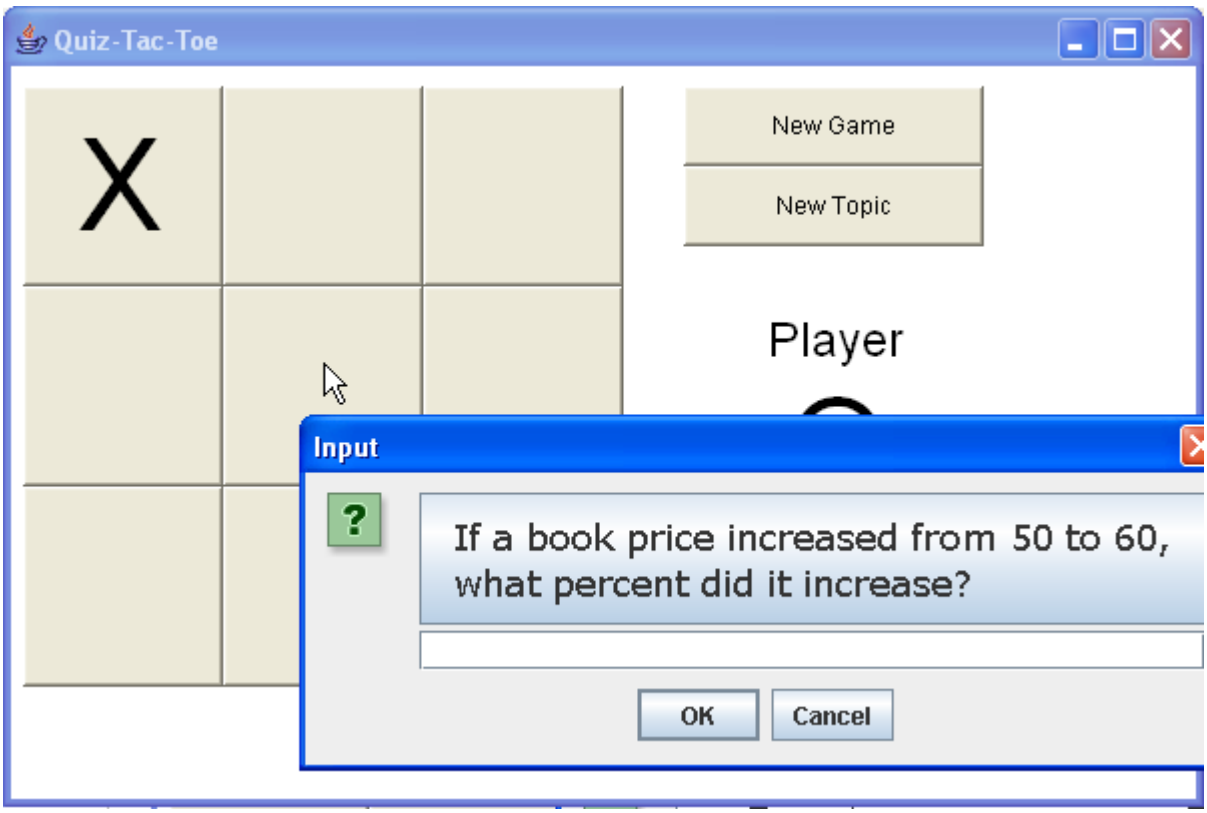


Player O wins the game!

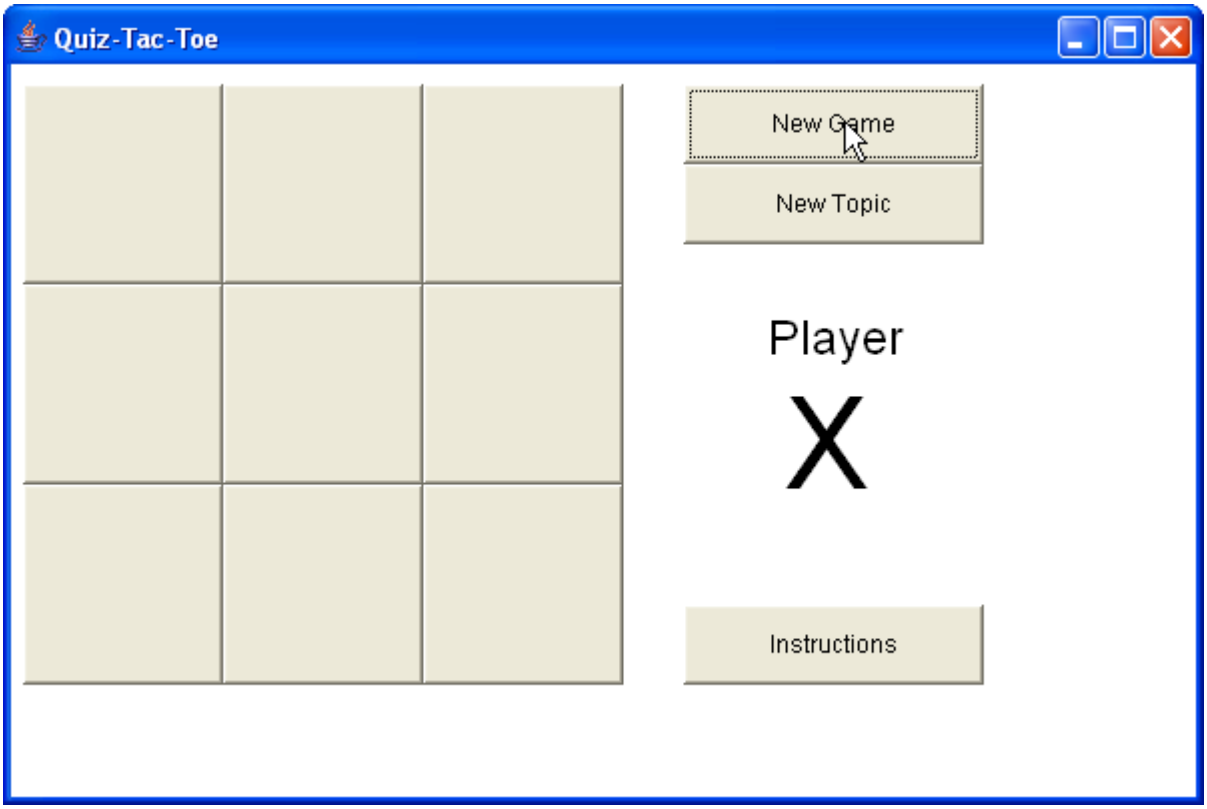
(G11) They start a new game. Now there are different questions in the squares.



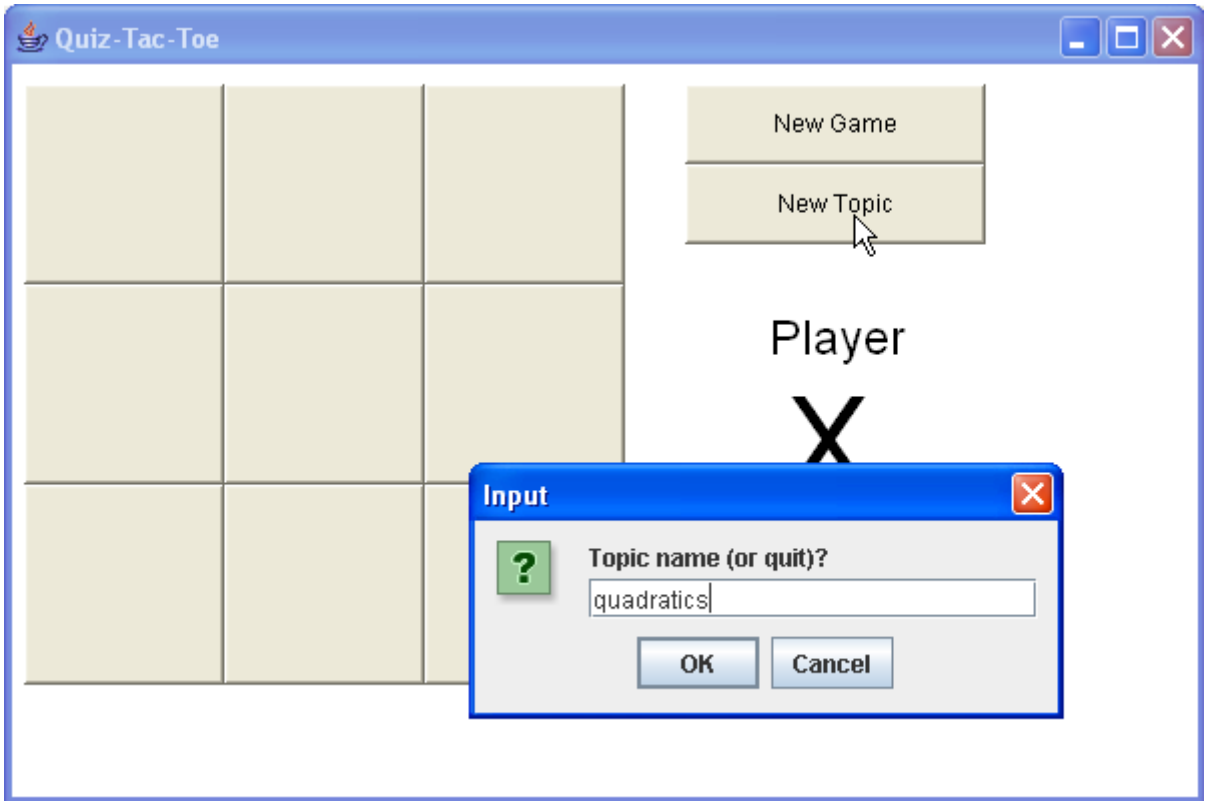
(G12) O tries the middle square.



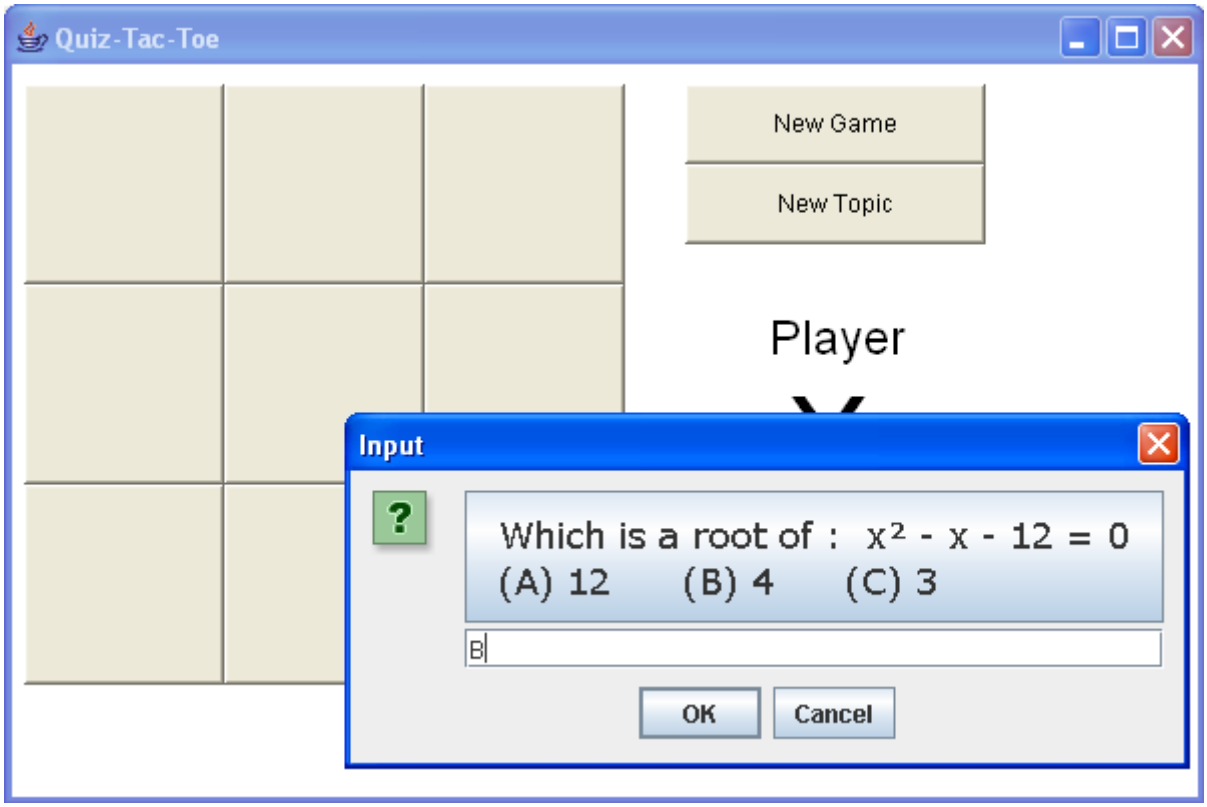
(G13) Now player X decided to start a new game by clicking [New Game].



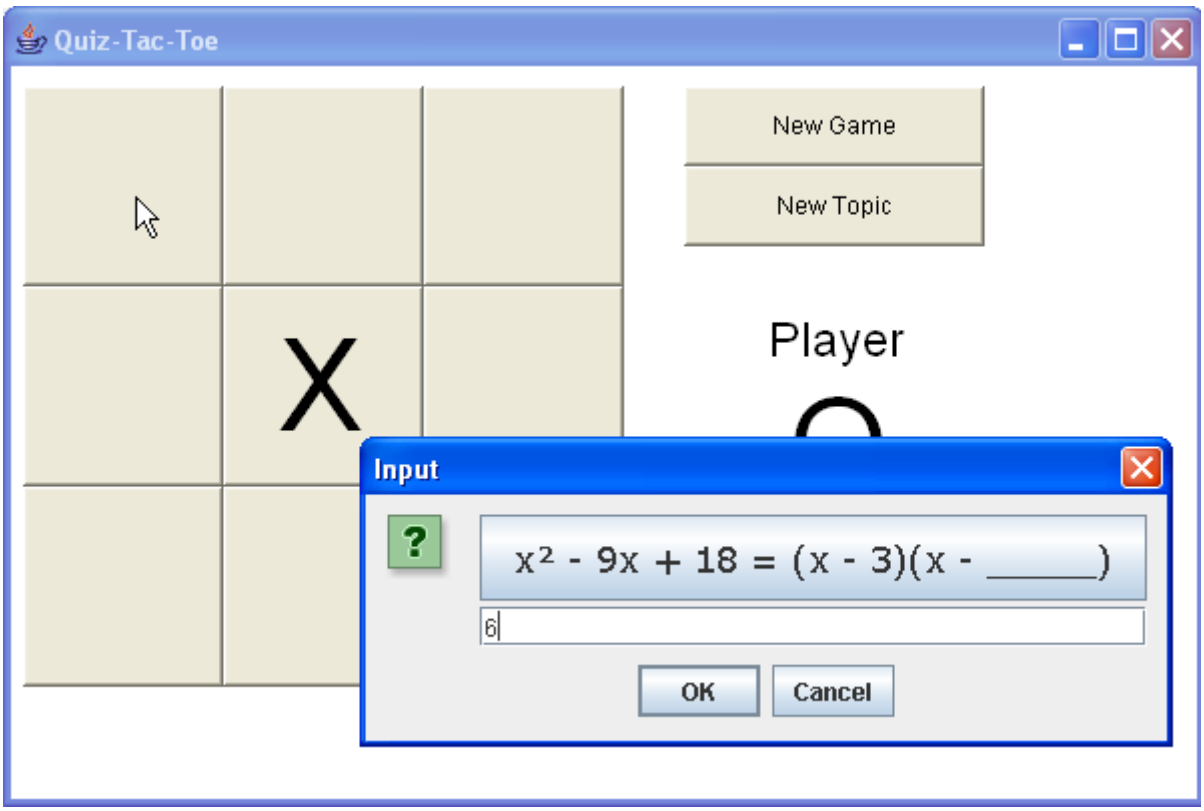
(G14) Pressing the [New Topic] button will also start a new game, but allows the players to select a different topic.



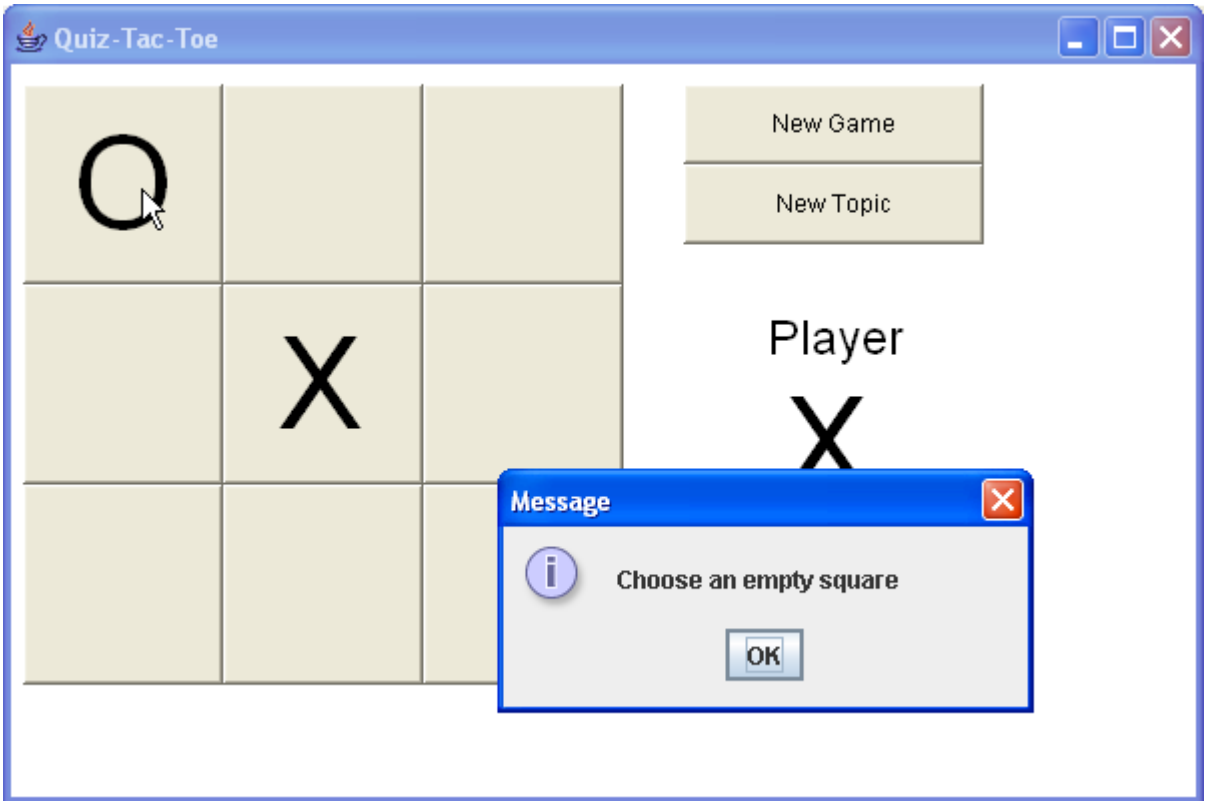
(G15) Now the problems are a bit harder.



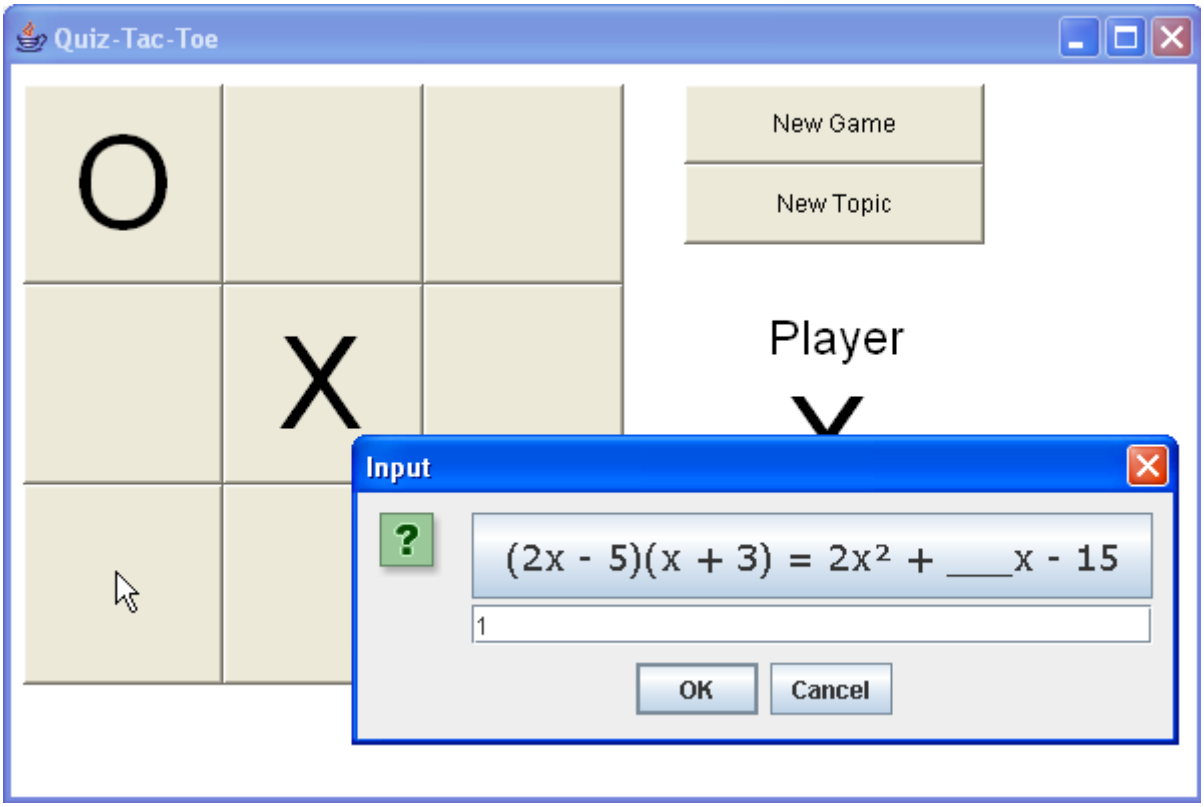
(G16)



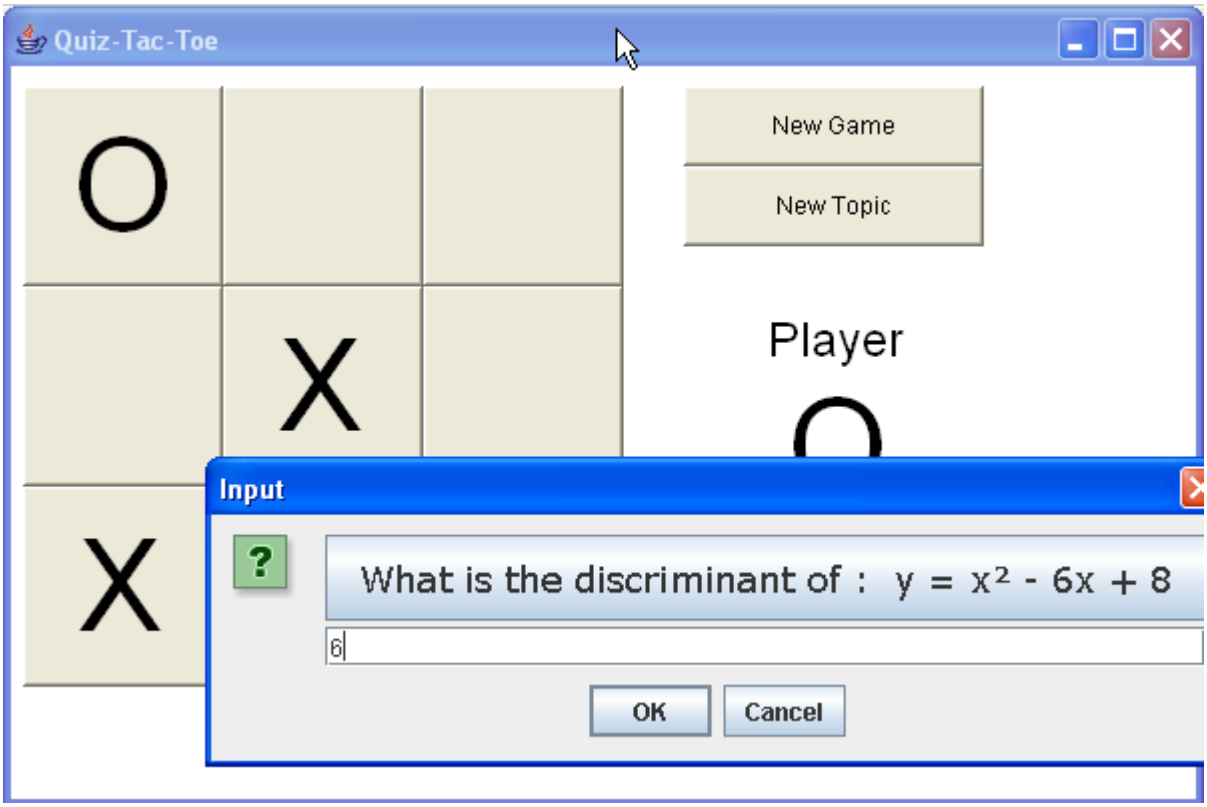
(G17) If X tries to cheat and click on a square that's already taken (top left), the move is rejected.



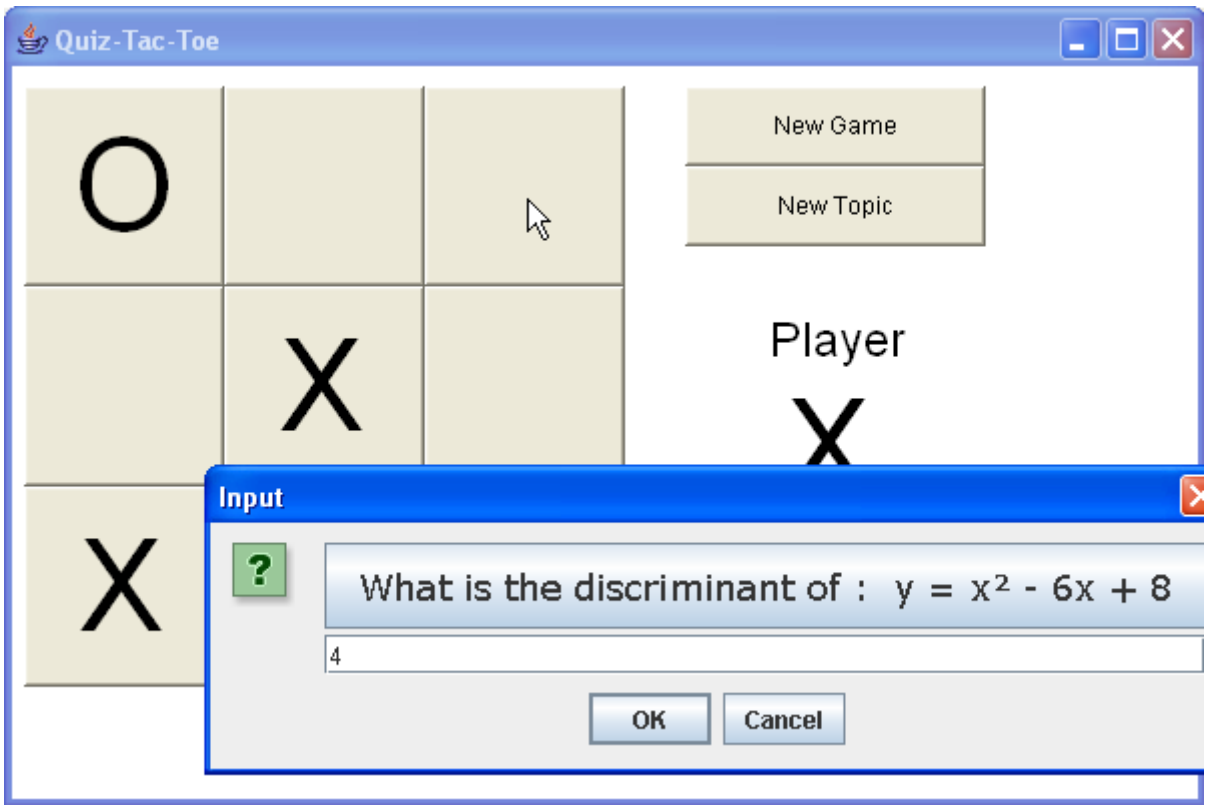
(G18) X is allowed to choose a different square - this time bottom left.



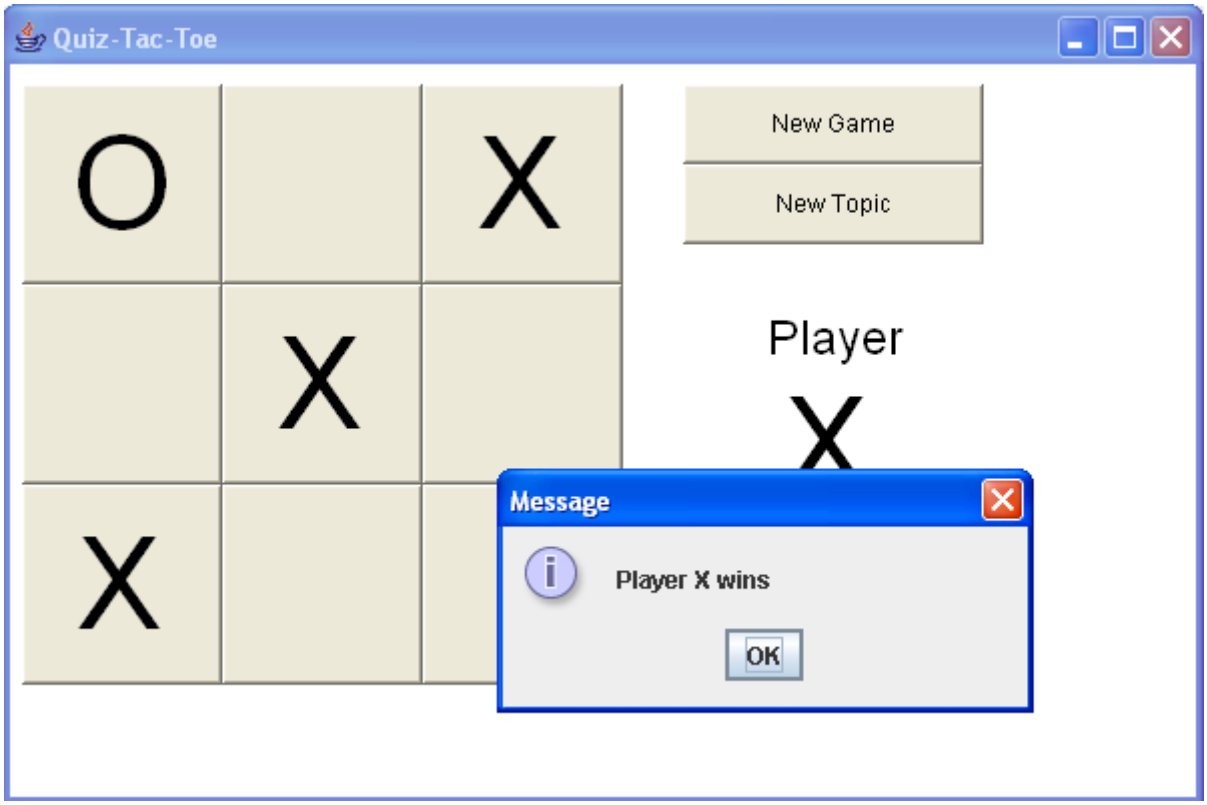
(G19) Now O must play top-right, but answers incorrectly.



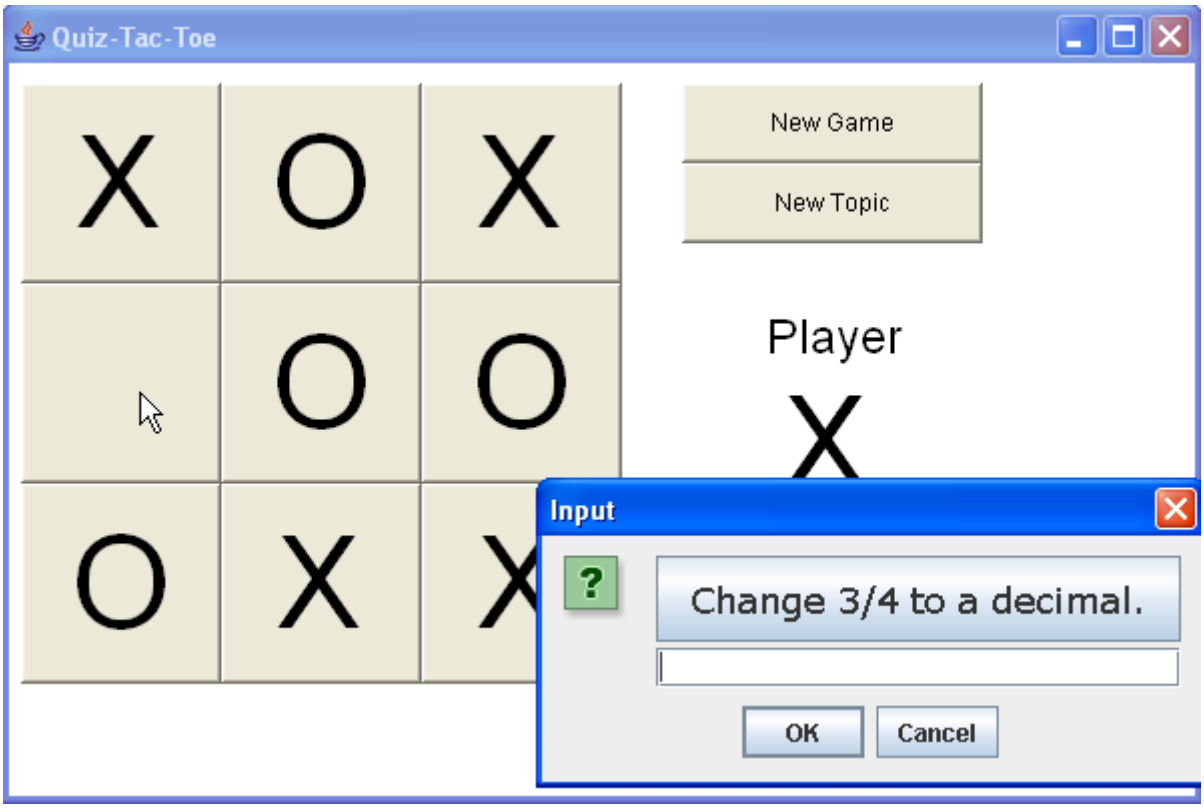
(G20) O's answer was incorrect, so X can take the top-right square and win with a correct answer.



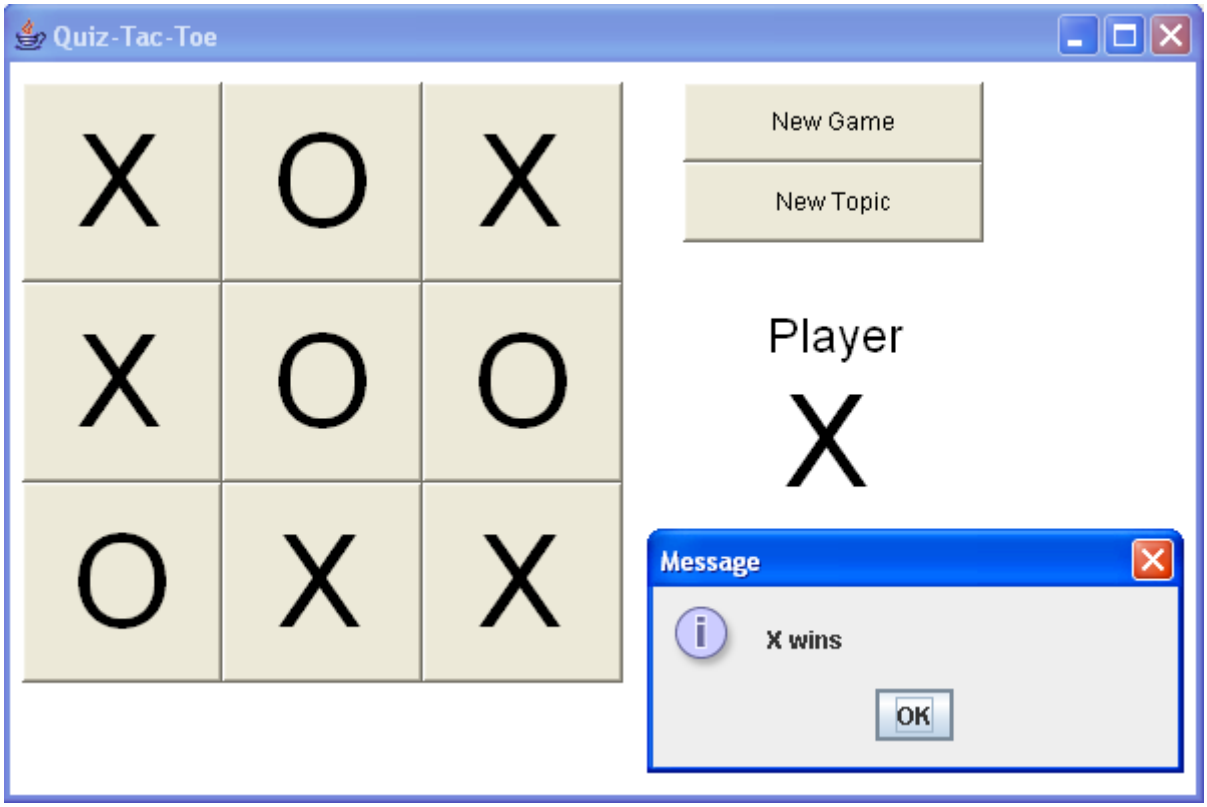
(G21) X wins along the diagonal.



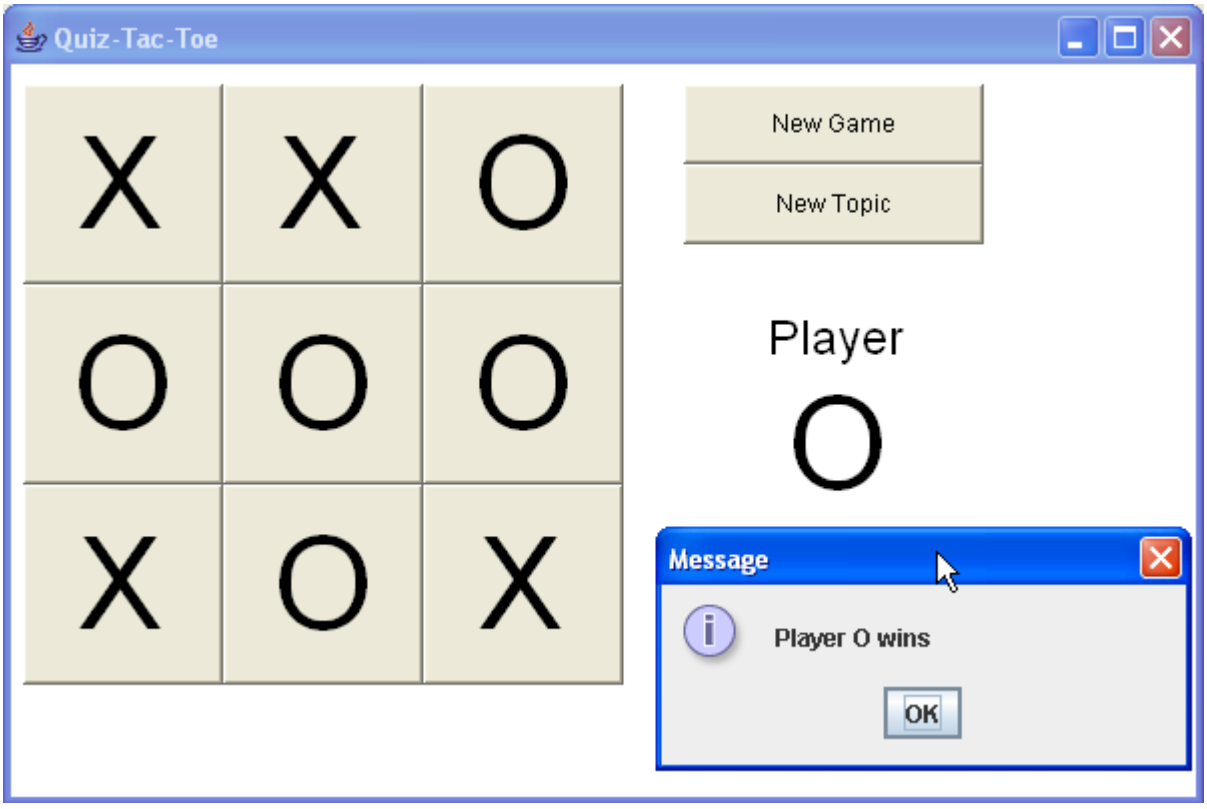
(G22) In a different game, it is going to be a tie. If X answers correctly, then X will win by having more squares (5 to 4).



(G23) It's a tie, but player X wins because they have more squares.

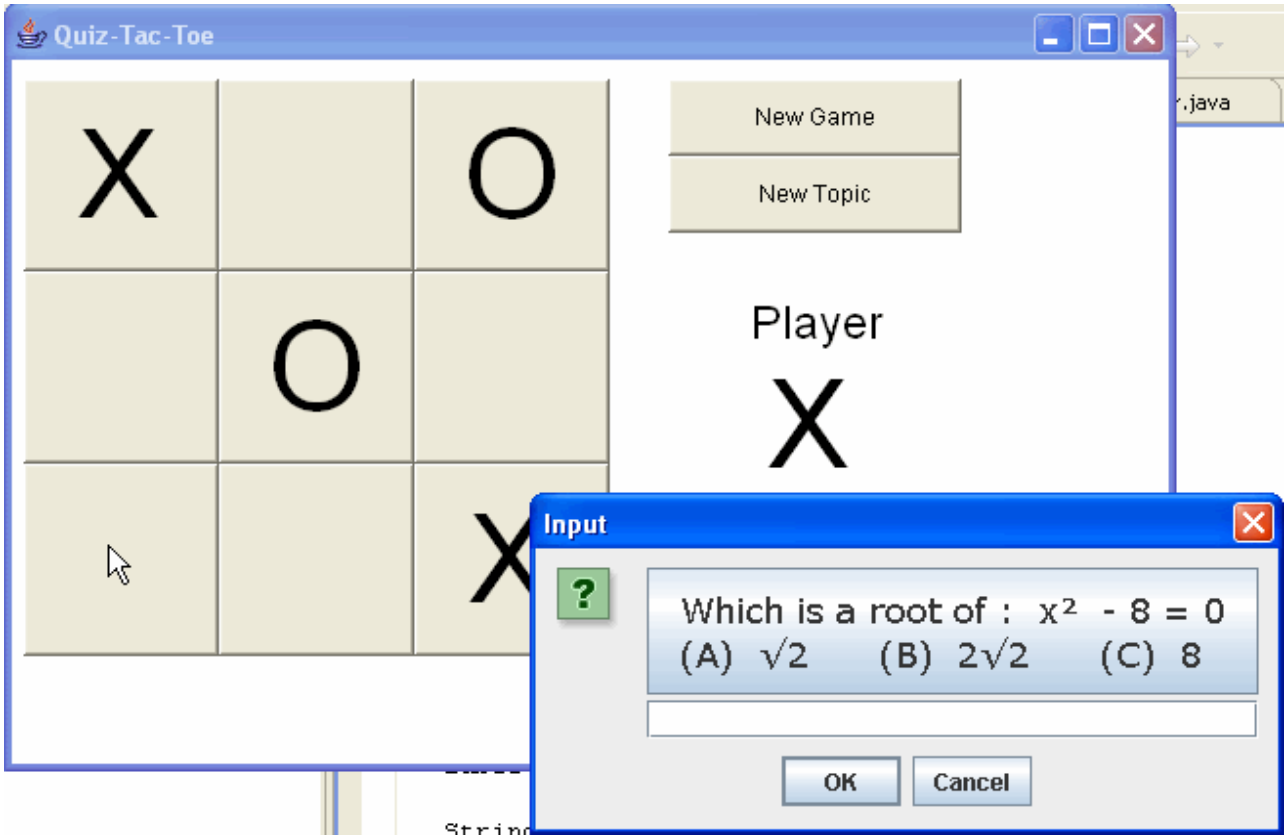


(G24) This rule makes the game very unfair - it's much easier for X to win. But they still need to have correct answers. The players should take turns going first. Here is a tie where O wins.



(G25) If a player clicks the [Instructions] button, they see the following web-page:

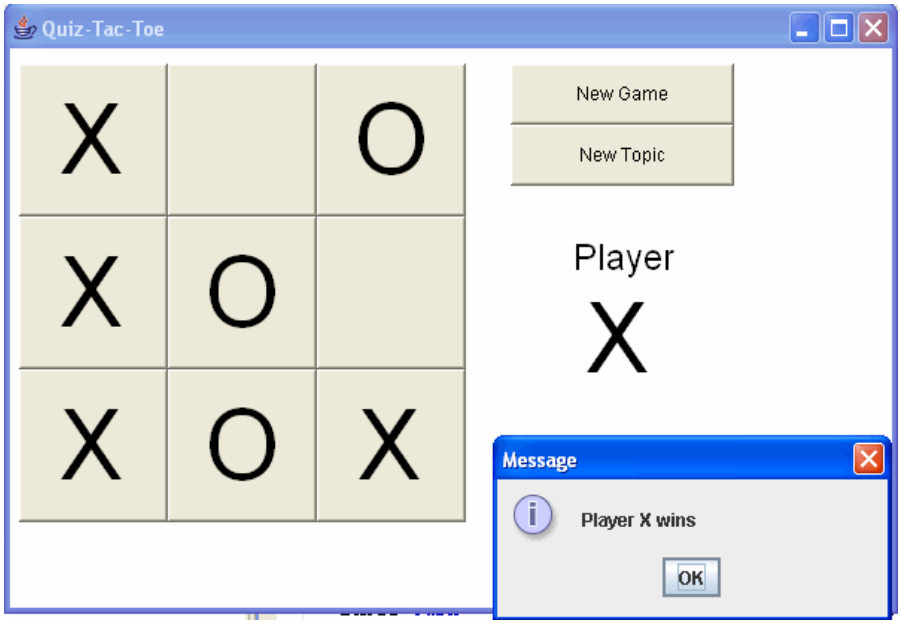
Quiz-Tac-Toe - Instructions



Quiz-Tac-Toe is a math practice game for middle and high school students.

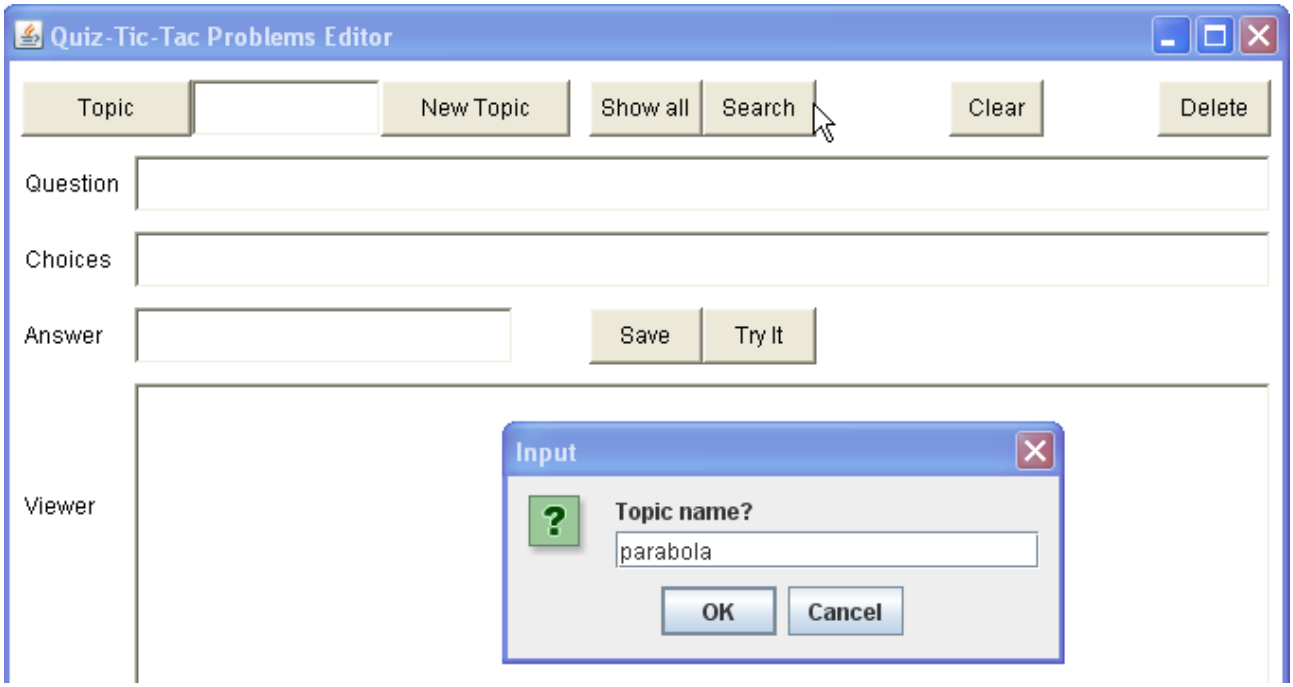
Quiz-Tac-Toe is like a normal Tic-Tac-Toe game, but each time you click a square, you must solve a math problem. If you answer correctly, your X or O is placed in the square. Otherwise the square remains blank.

Winning follows normal Tic-Tac-Toe rules - 3 in a row in any direction.

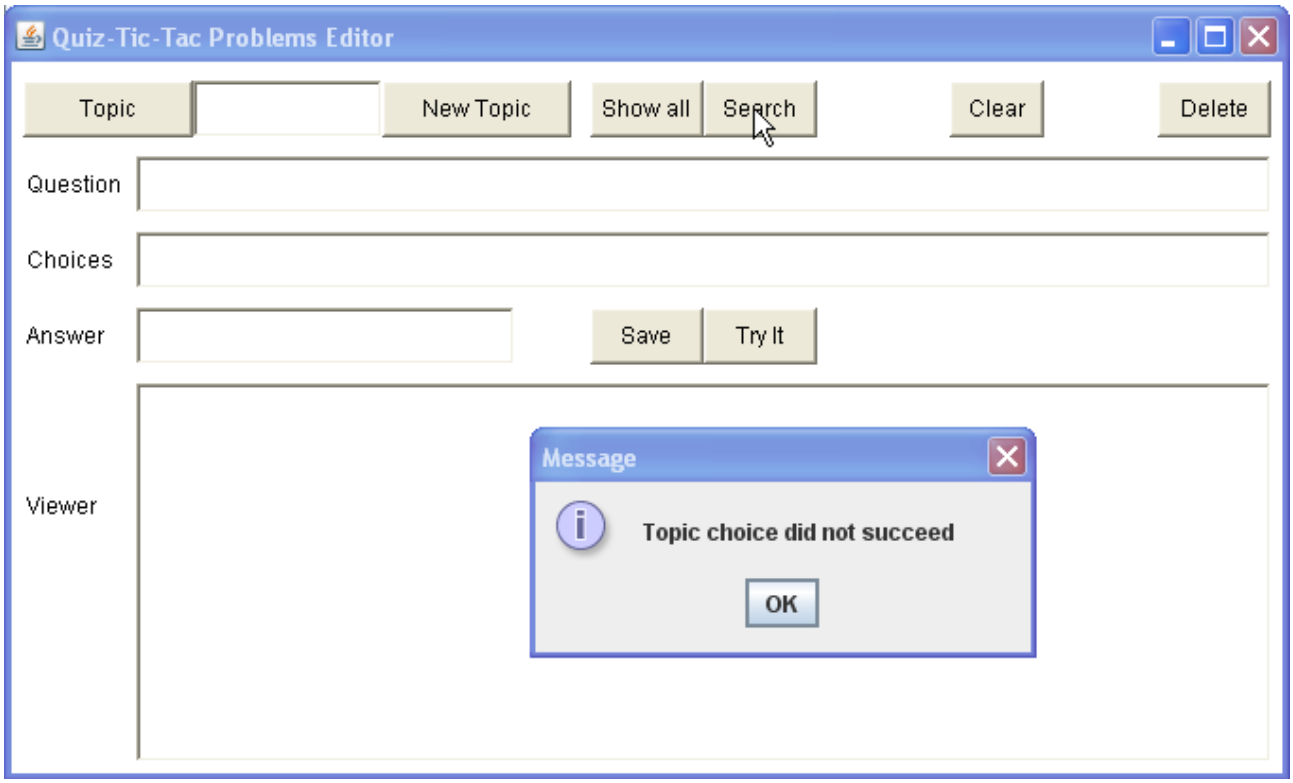


Testing Reliability and Error-Handling

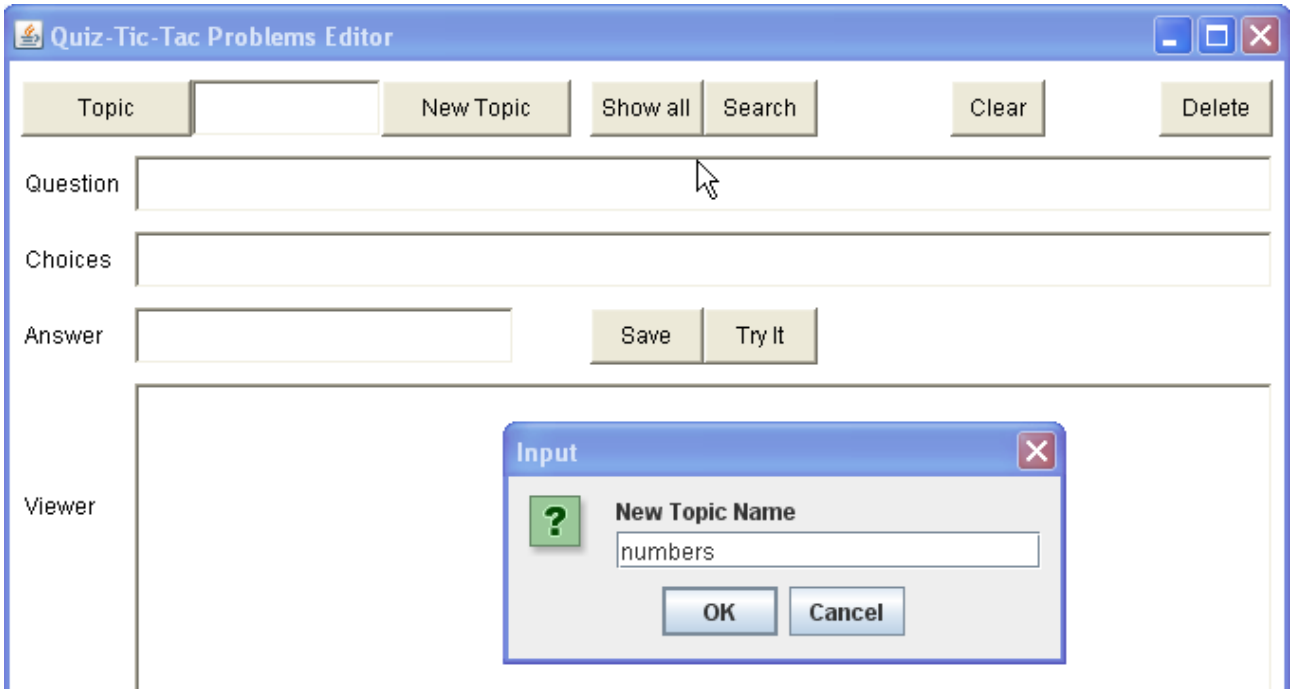
(T1) Teacher types a topic name which does not exist (no file) - it is rejected.



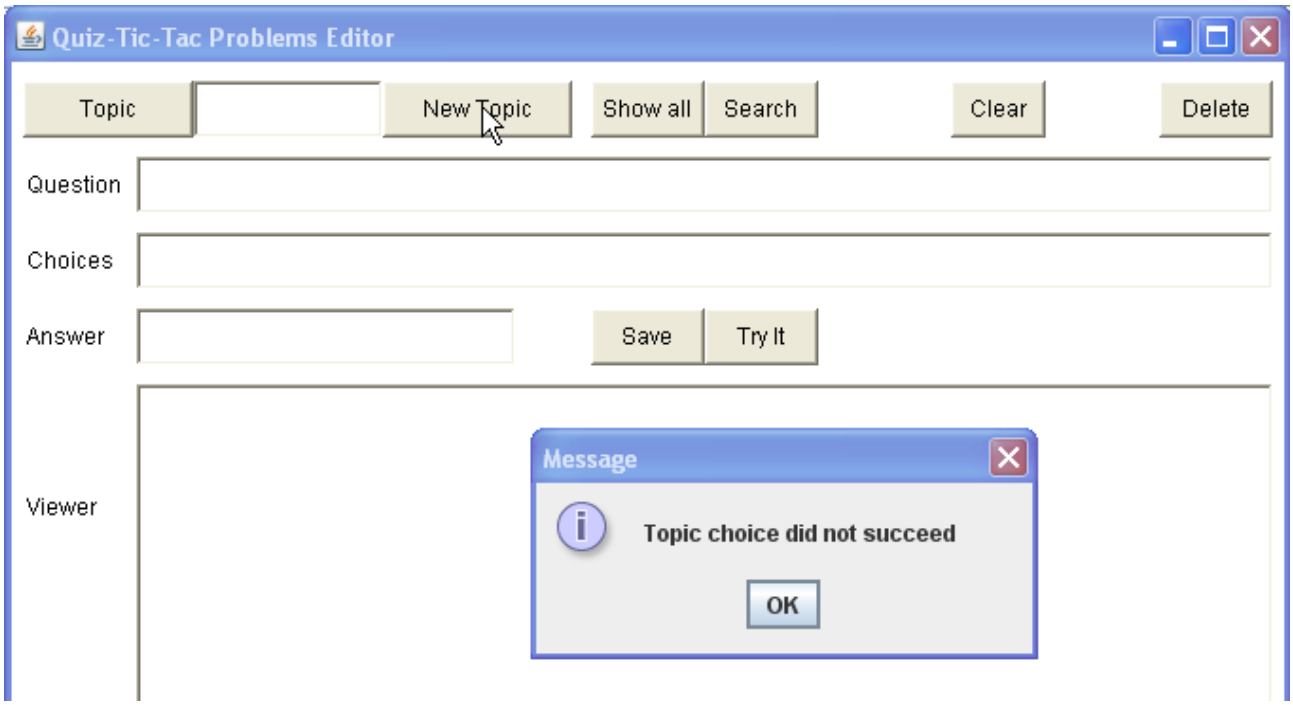
(T2) Rejected.



(T3) Teacher tries to create a [New Topic], but it is rejected because the topic already exists.

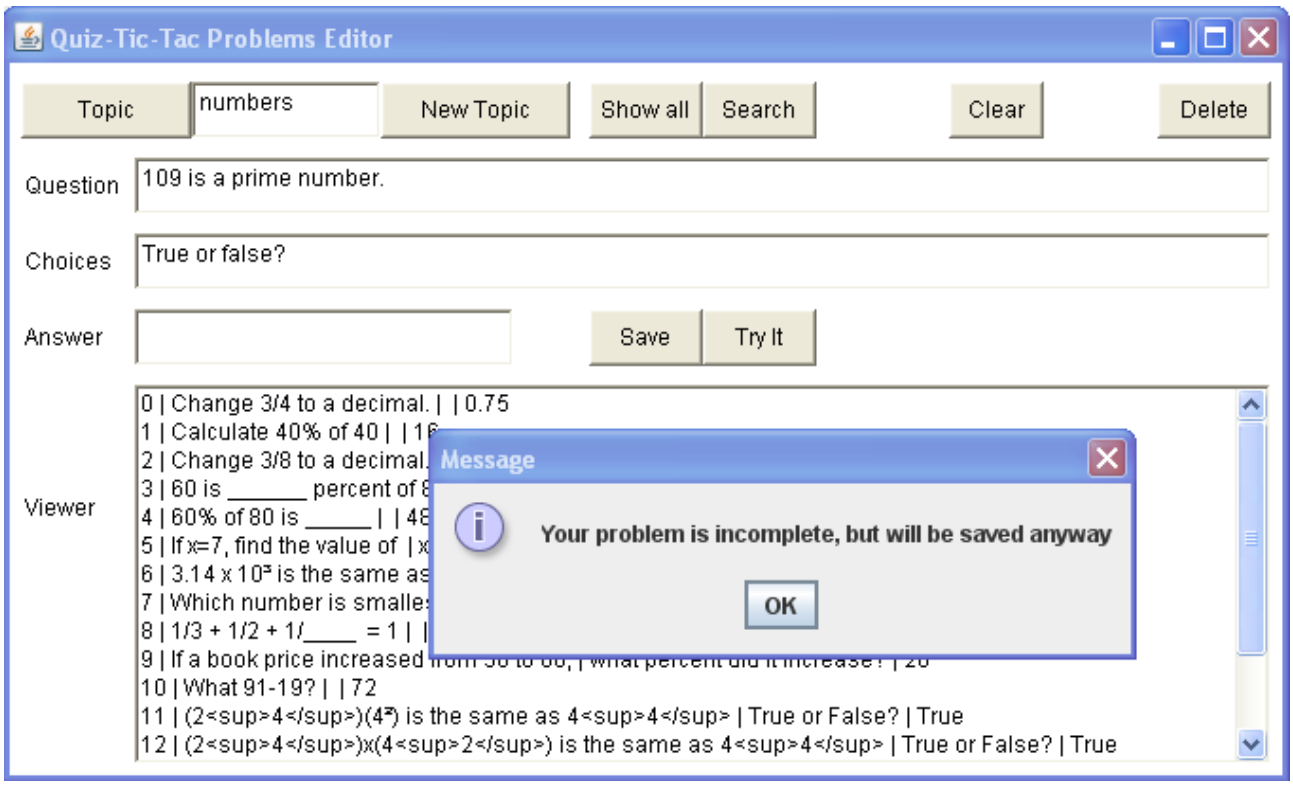


(T4) Topic is rejected.

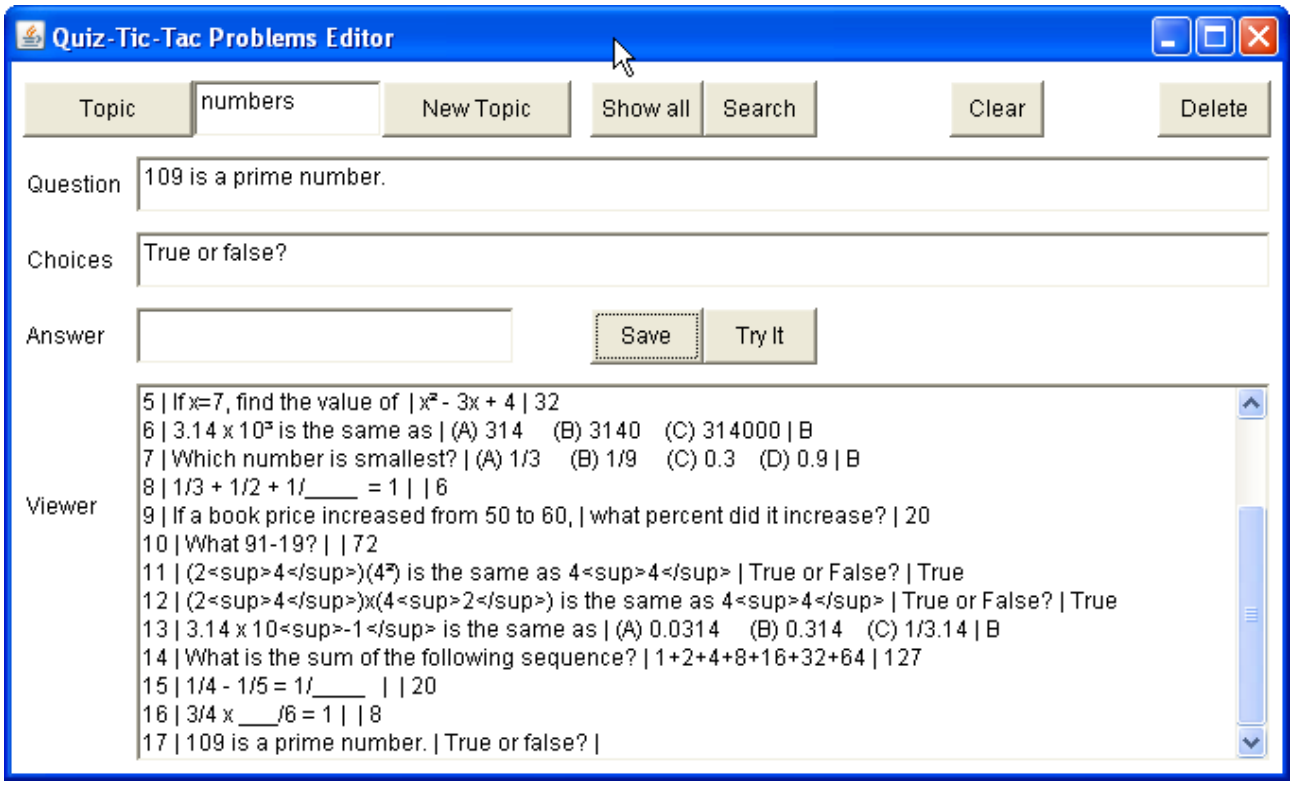


This error message should probably be changed to be more specific - e.g. “Topic already exists”

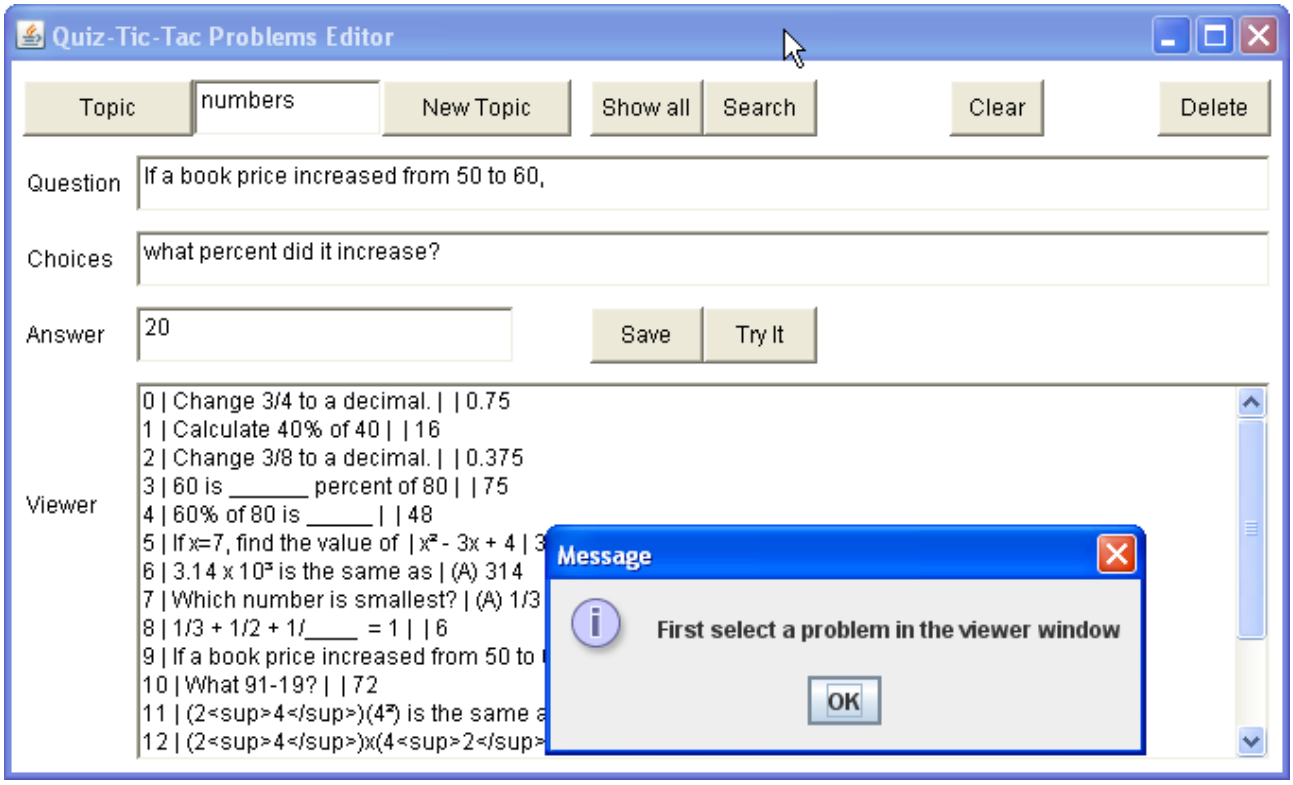
(T5) Teacher attempts to save a problem without an answer. A warning is printed, but the problem still gets saved.



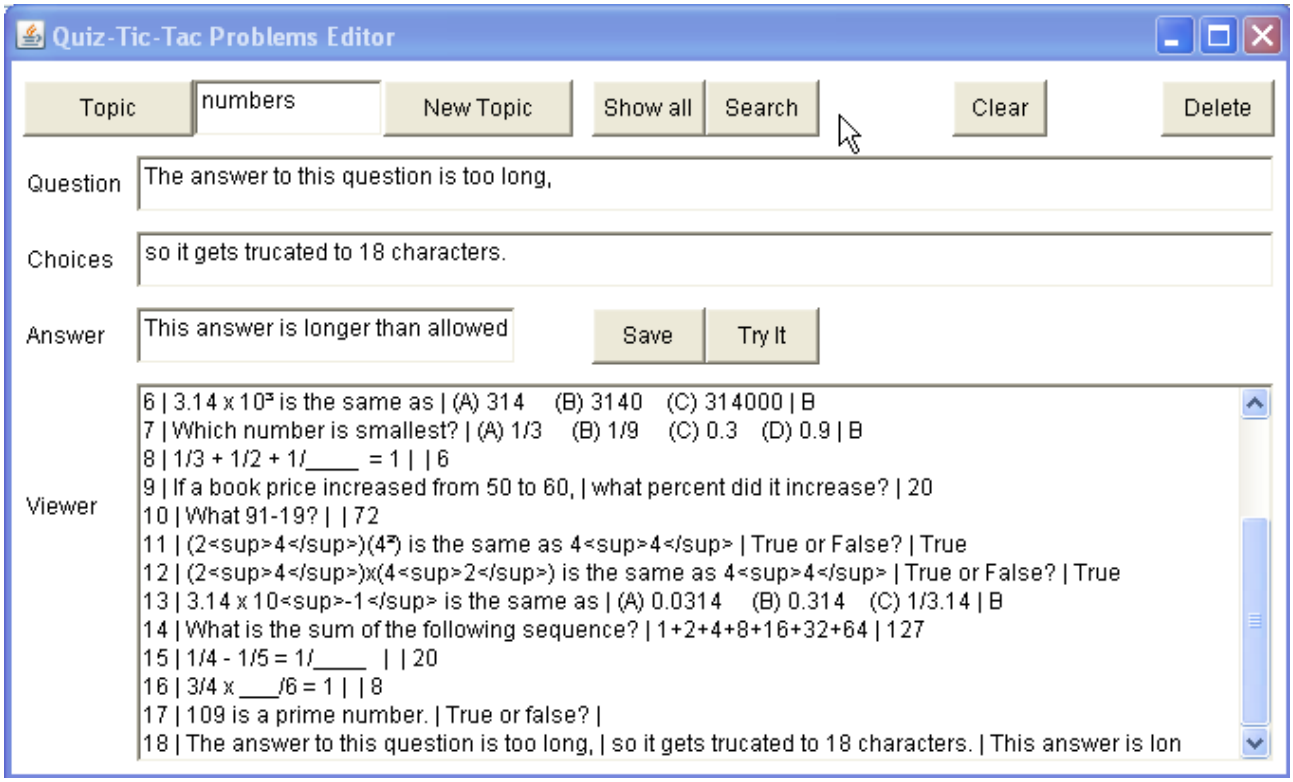
(T6) Notice that the problem was indeed saved.



(T7) The teacher clicks [Delete] before selecting a problem - an error message appears.



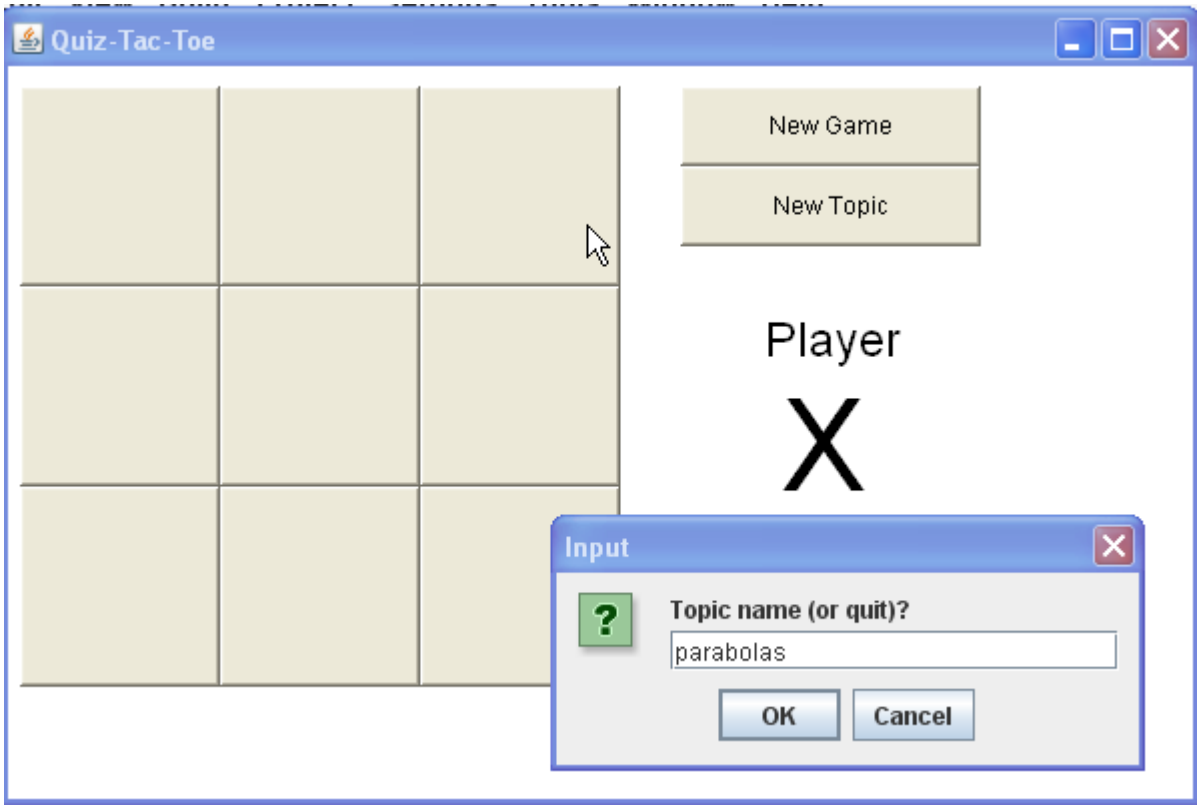
(T8) If the answer is longer than 18 characters, it is shortened to 18 characters before saving.



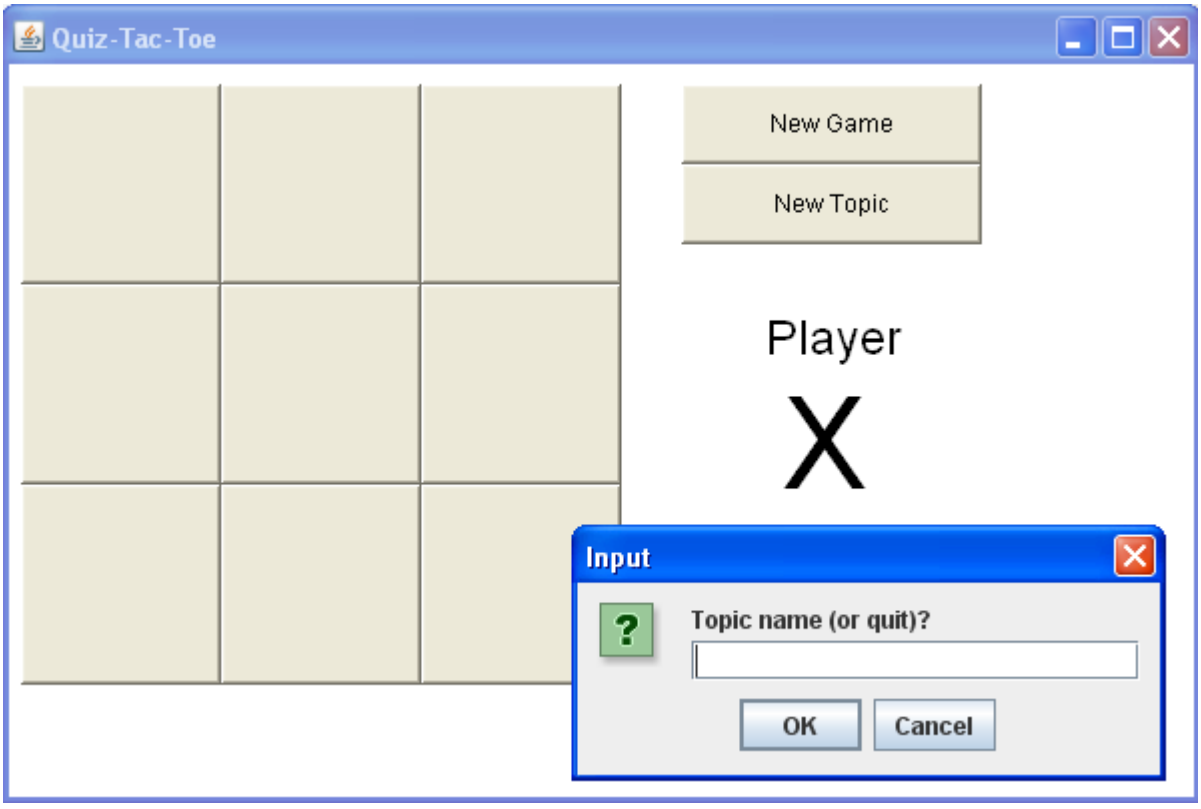
The question and choices are limited to 88, so the WriteUTF fits into 90 bytes.

This works okay, and prevents corruption, but should include a warning message for the user.

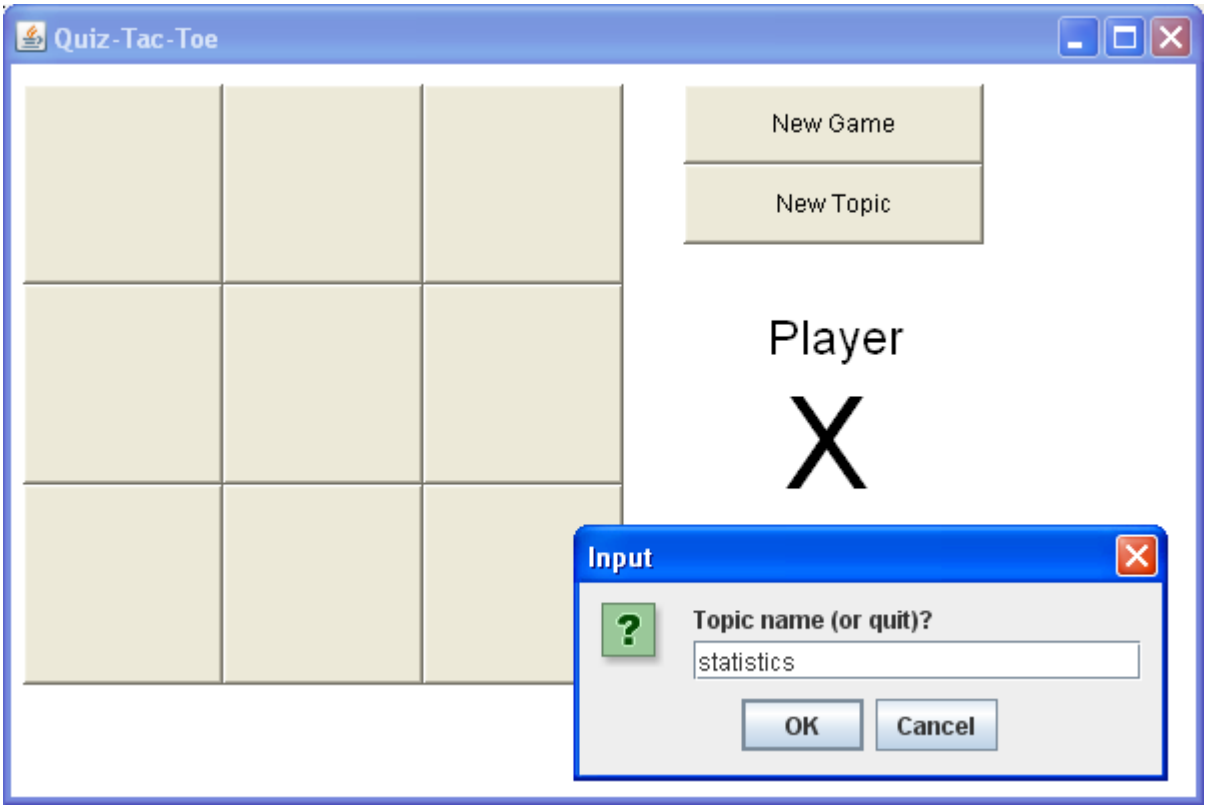
(T9) In the GAME module, very little can go wrong. The only problems that occur are when the students type a topic name that doesn't exist, or the topic file contains fewer than 9 problems.



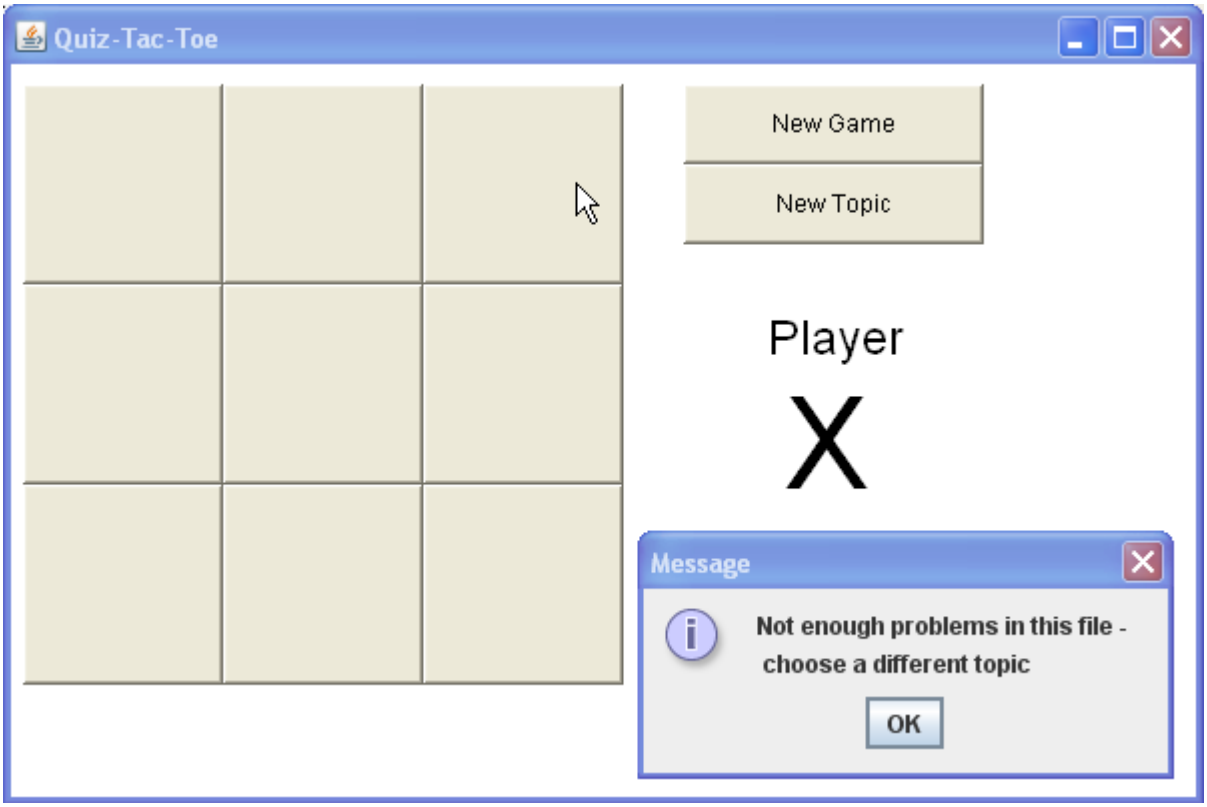
(T10) Topic does not exist - it is rejected.



(T11) The statistics file does not contain enough problems, so an error message is displayed.

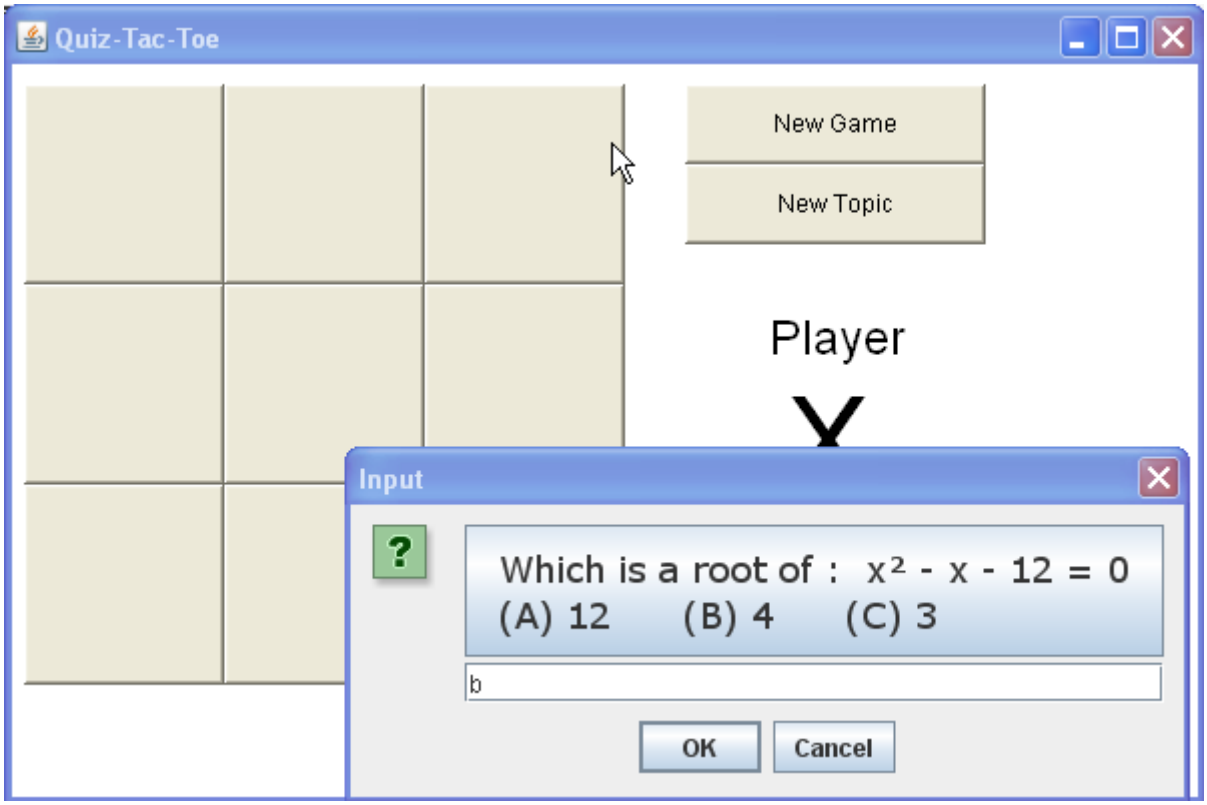


(T12)

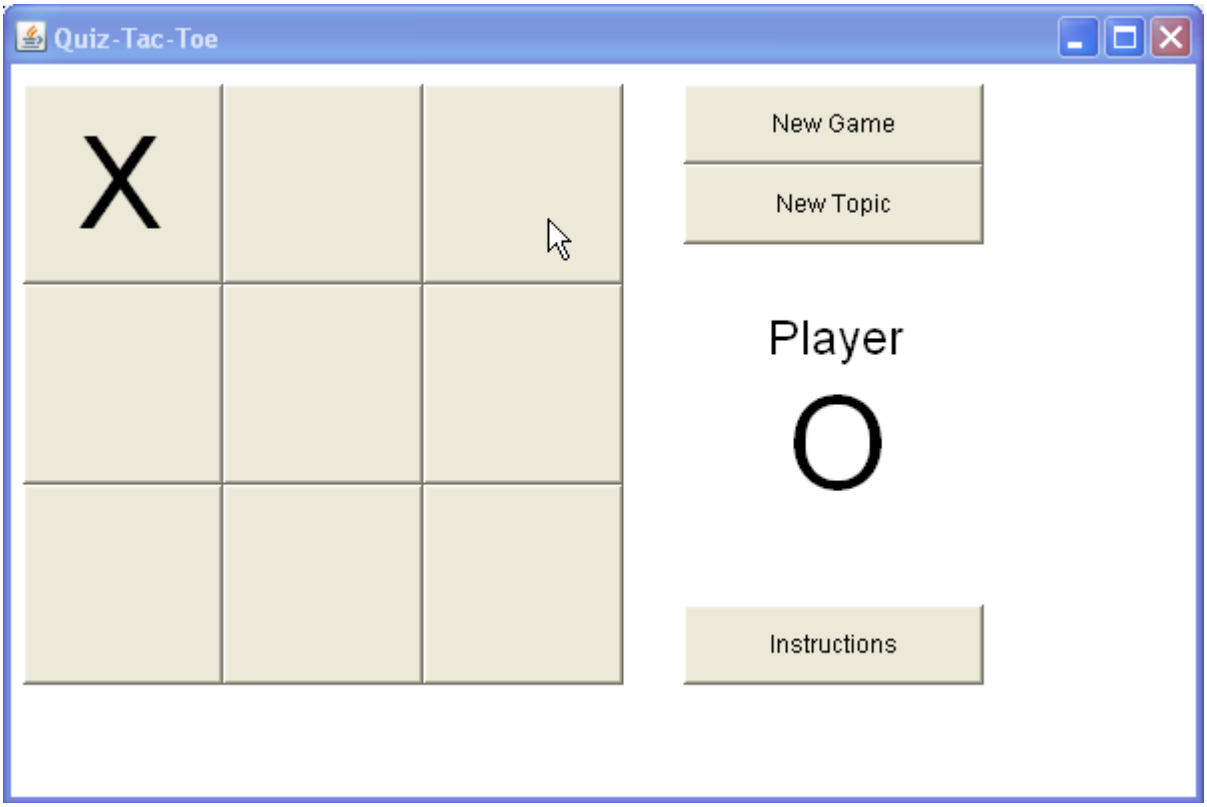


But after this error message, there is no topic at all. So the board is empty but does not respond to clicking. It would be better if the program would ask again for a topic - but the students can click the [New Topic] button to continue, so it's not a disaster.

(T13) Answers are not case-sensitive

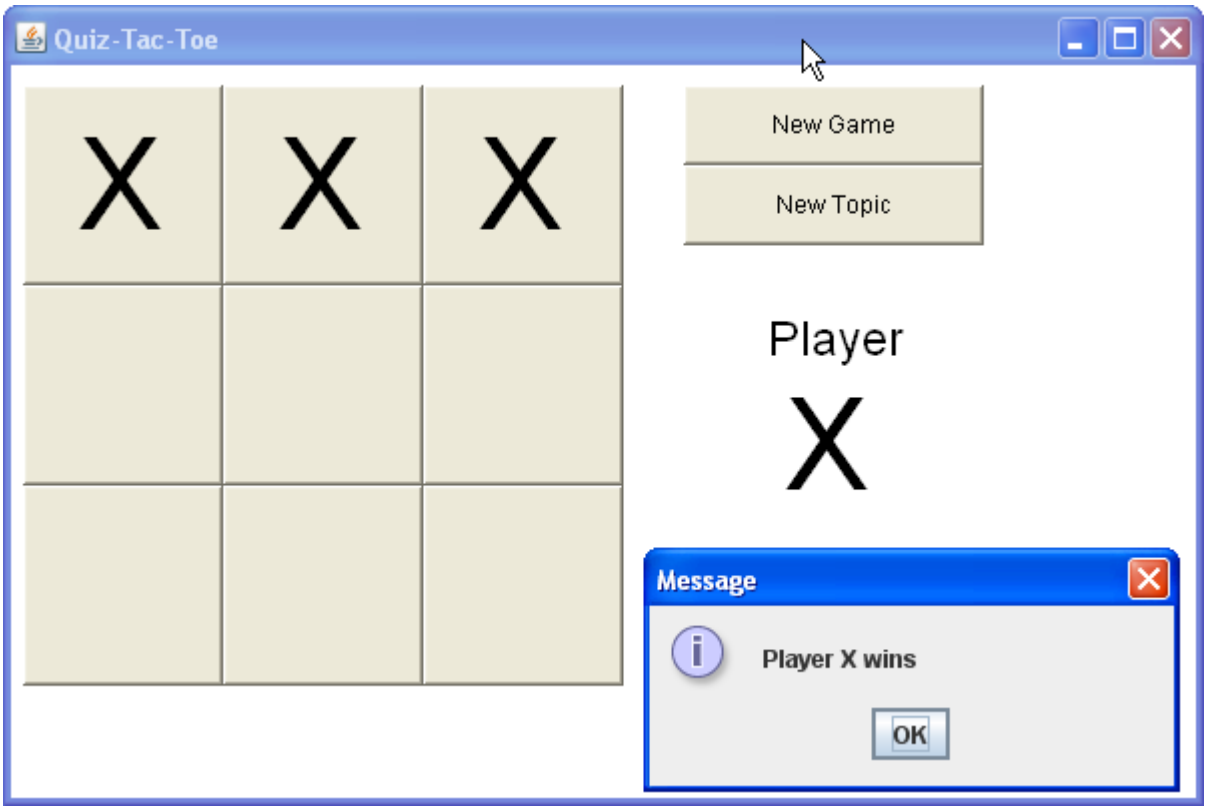


(T14) The b answer was correct (didn't need B).

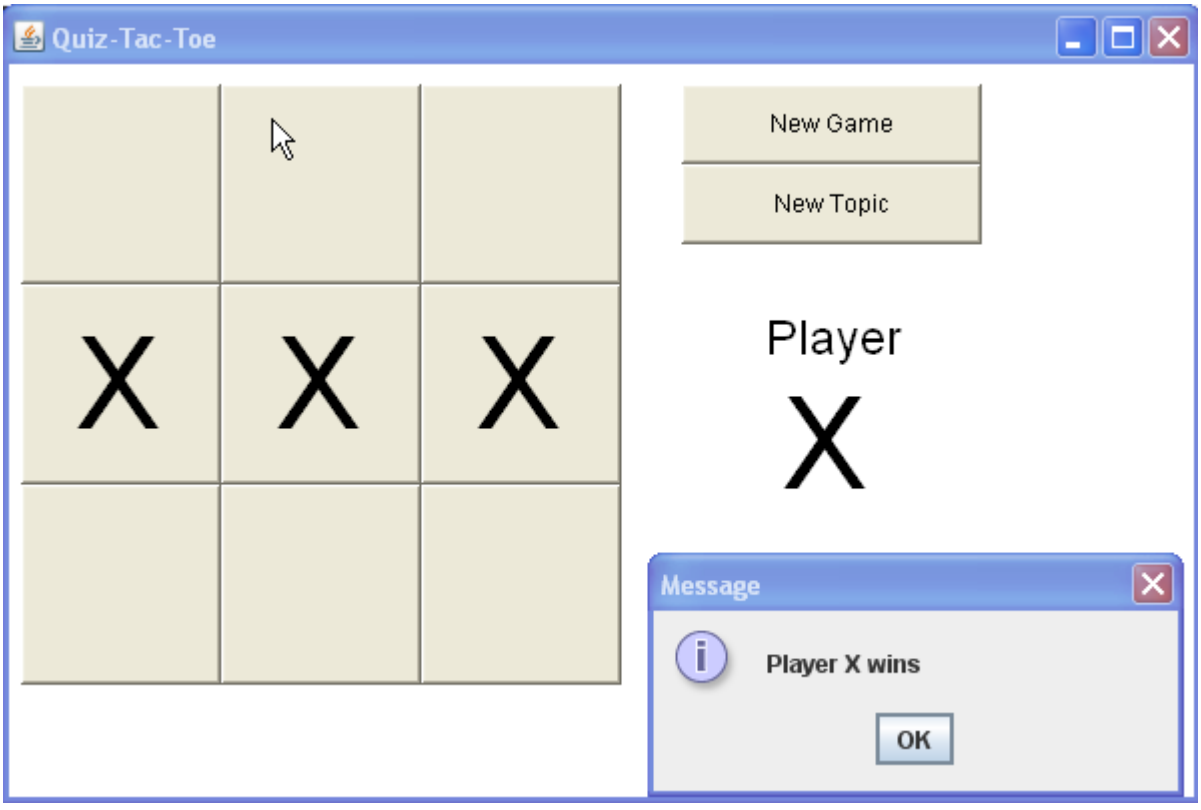


Following are tests showing that all 3-in-a-row directions are recognized by the program.
They are shortened games - O always answers incorrectly.

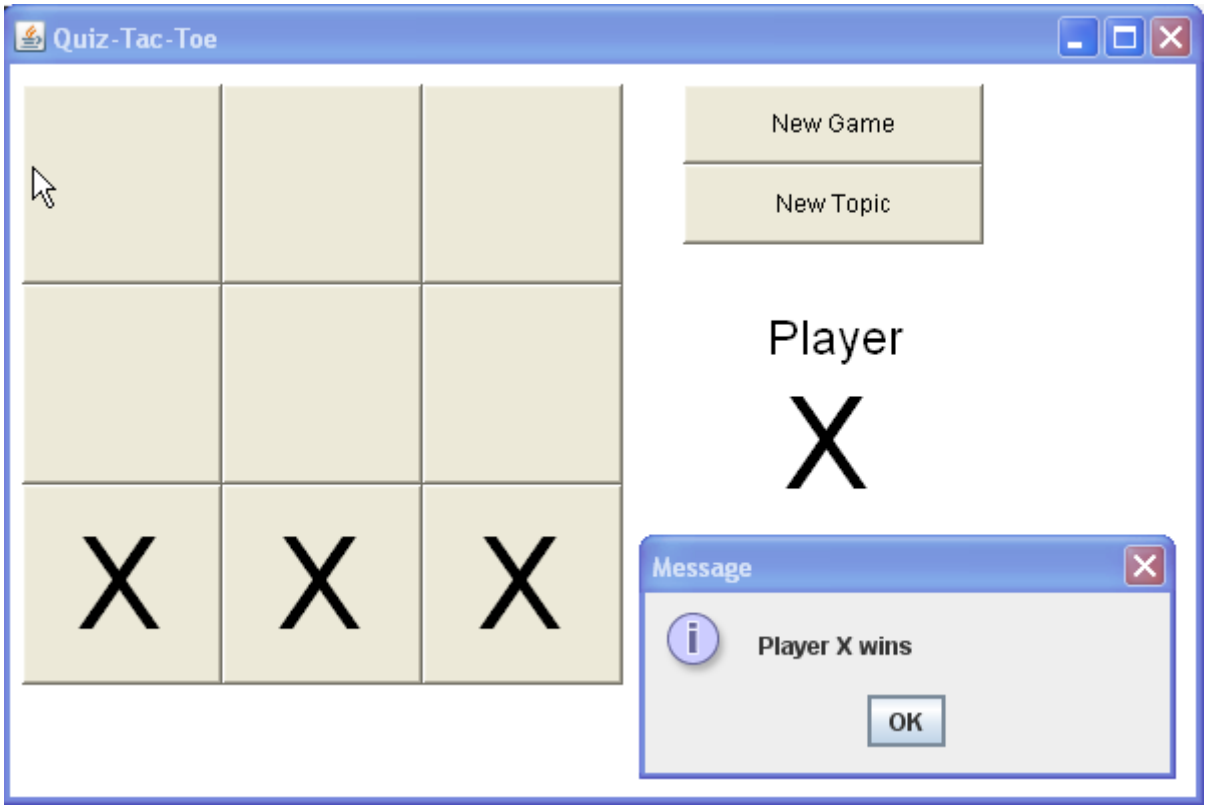
(T15) Top-Row



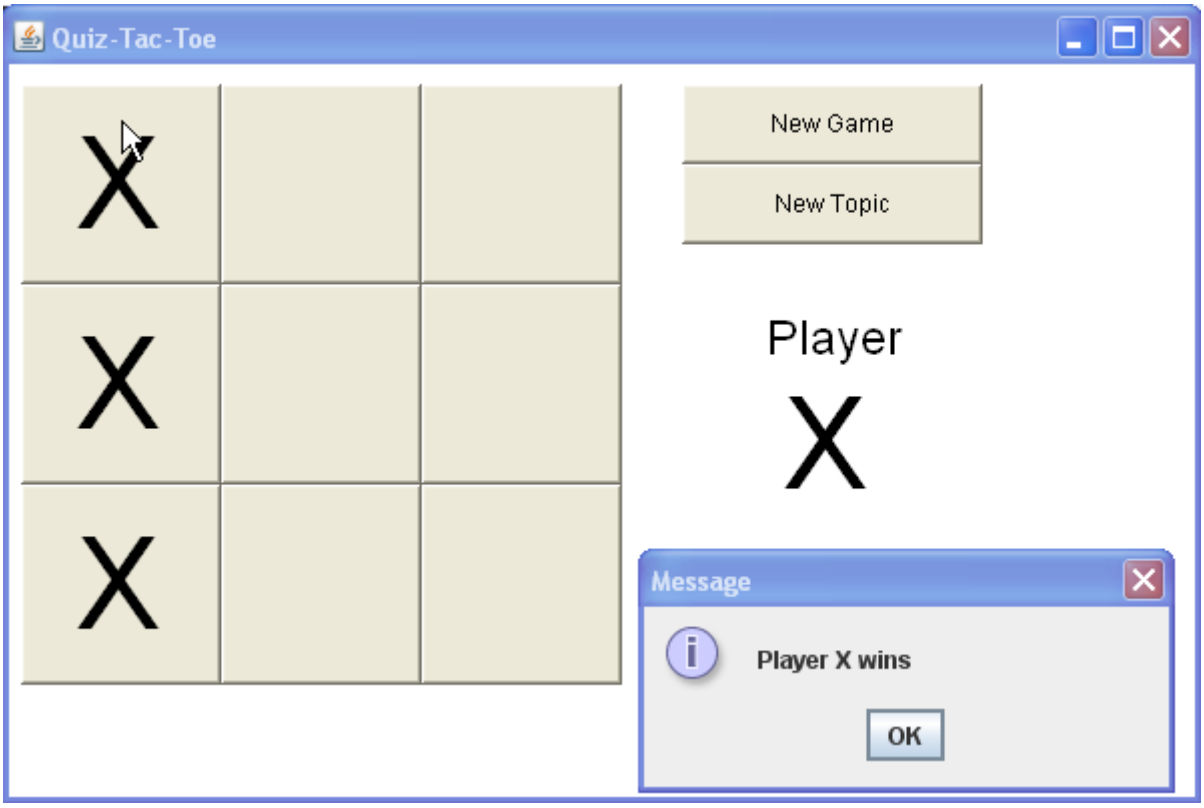
(T16) Middle Row



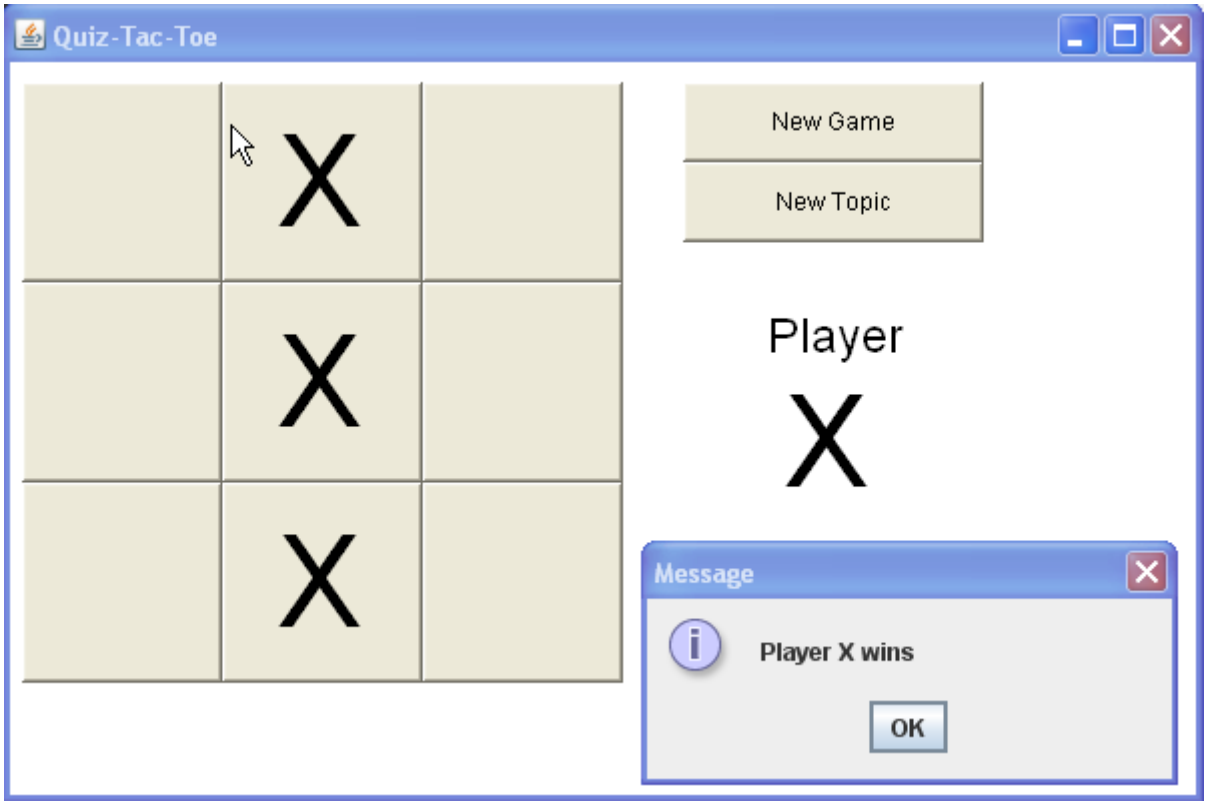
(T17) Bottom row



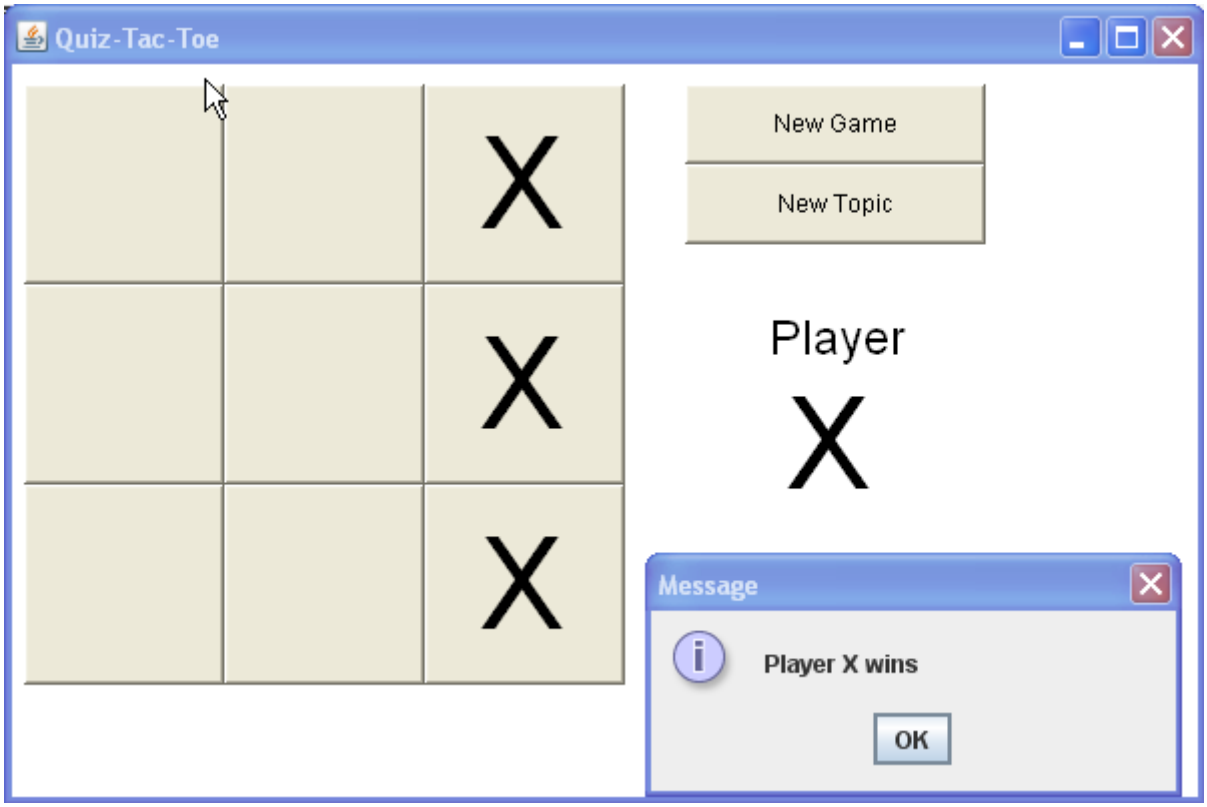
(T18) Left column



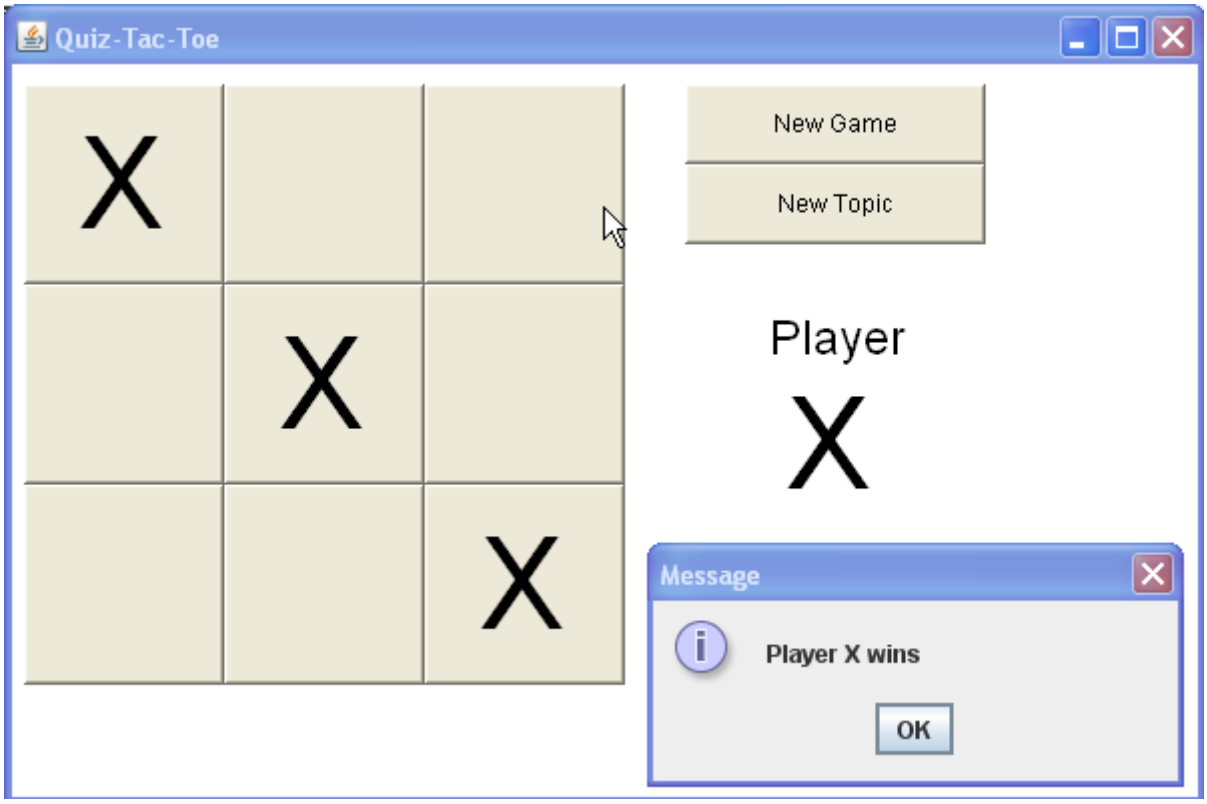
(T19) Middle Column



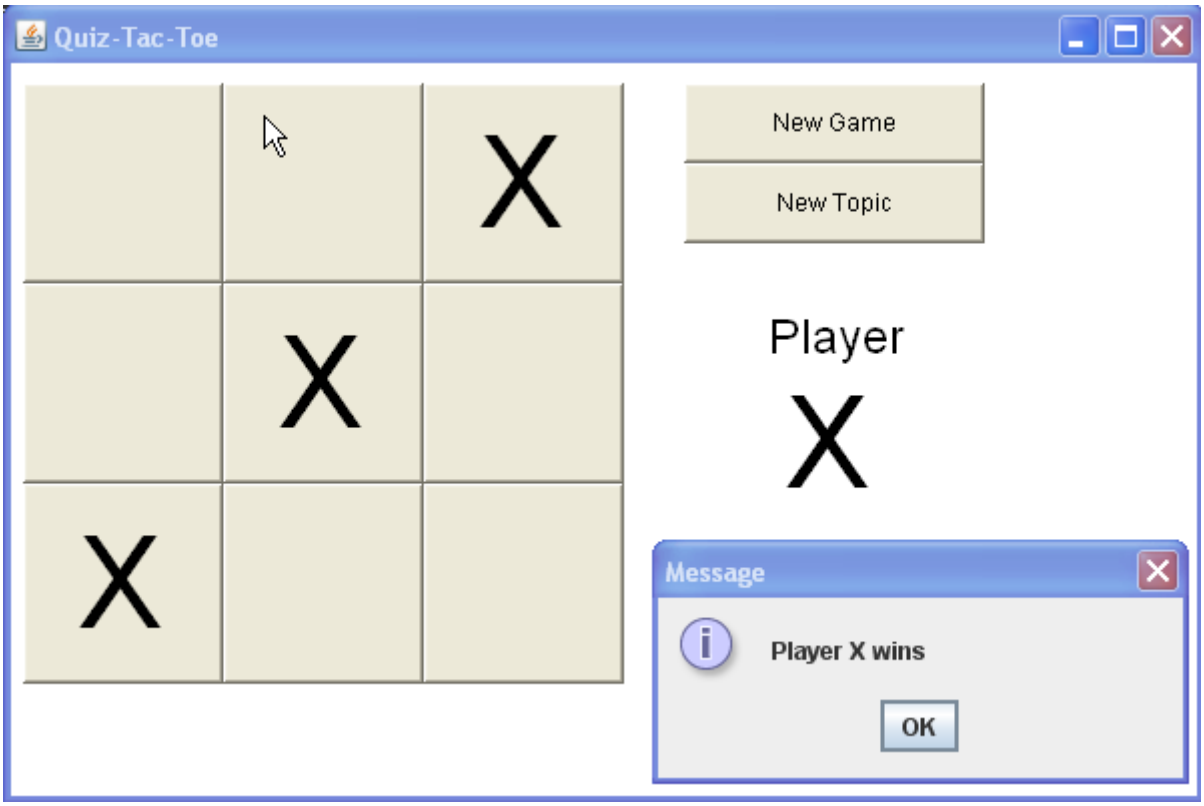
(T20) Right Column



(T21) One diagonal



(T22) The other diagonal



D2 - Evaluating Solutions

Did the program work?

The program worked very well.

- The teacher can enter new problems and answers easily, using keyboard short-cuts for special math symbols.
- The GUI editor interface makes it easy to review, edit, and delete old problems.
- The students' game interface is simple to use and easy to understand.
- The game follows standard Tic-Tac-Toe rules, so it's easy to play.
- Each game randomly selected a different set of questions.
- Questions are short and quickly answered.
- There were no significant run-time errors.

As shown in documentation section C4, all the criteria for success were addressed successfully.

Various data sets

The program (both editing and playing the game) worked correctly for various topics, with no significant run-time errors. If the topic file is too short (under 9 questions), the game module refuses to start the game, as it needs 9 questions. This possible error was handled correclty.

The design easily allows handles both multiple-choice and full-work answers (including true/false) so the system allows flexibility in the type of question.

Limitations

Mathematical Notation

The most significant limitation is in the presentation of mathematical notation. For example, fractions appear as "3/4" instead of ¾ or $\frac{3}{4}$. Since the questions are generally short and relatively simple, this is not a huge problem. But more complex algebraic expressions such as $\frac{x^2-8x+12}{x-6}$ cannot be written sensibly, so algebraic fraction problems cannot be set.

No images

The teacher originally asked about including images (diagrams) in the questions. This was not achieved. Questions consist only of text and special symbols.

Text-Only Input

The program only accepts simple text input. This is fine for the game module, but a bit limiting for the Problem Editor module. Teachers can type HTML markup code, but this is too difficult for most teachers. Some teachers might be more comfortable using a word-processor than the simple text-editor.

Additional Features

Future versions of the program should include the possibility of putting images in the problems. Although the current version permits HTML code to be used, and thus an <image> tag, it seems like it would be possible to include images. But this is difficult and the formatting of the question is difficult to control.

An improved version of the program could allow WYSIWYG editing in a larger box, including bold, italic, and images.

It would be nice to allow proper mathematical formula notation. This might be possible through the use of an equation editor, like the tool in MS Word. But this is incompatible with HTML, it's not clear how to link it with a Java program, and the results could not be stored in the RandomAccessFile. So this feature is unlikely to be added in the near future. When we looked at existing online quiz software, this problem also existed in all the programs we saw. The best alternative seemed to be some Flash software, but we did not find anything that had a usable problem editor for the teacher to use.

Appropriateness of the Initial Design

The initial design was quite complete and led easily to the finished program. Large parts of the functional prototype were used as the basis for the finished Game module. The modular design with 3 classes worked very well. The user interviews were very helpful, leading to a clear and complete set of goals that led to a usable program.

Alternative Approaches

The teachers and students at our school have become accustomed to web-based solutions. That allows the students to use the system at home, without installing any software. This Java program cannot be set up to run “on-line” (web-based) because browsers will not open a RandomAccessFile. So even if this program were rewritten as an Applet instead of an Application, it would still not function on a web-site.

It appears that many web-based systems use MySQL and ASP pages to implement data-base functionality in the WWW. Apparently JSP (Java Server Pages) allow a similar functionality. I'm not familiar with any of these technologies, so I don't know whether this sort of change would be appropriate.

Simple Java applications will run from a web-site (without conversion to Applets) as long as there is not database access required. This could be set up by including the questions inside the program, as was done in the prototype, but this greatly limits the flexibility of the system – teachers would not be able to add and change the problems easily. So this approach is probably a dead-end.

Some students won't like the Tic-Tac-Toe game, so it would be sensible to consider other quiz environments. Some possibilities seen in other software are: a race, flying targets, a more conventional test situation, etc. It would be especially useful if the teacher could work on one set of questions and the students could use the same data in a variety of quiz environments, choosing the game they like the best. It should be reasonable straightforward to add other games that use the same database.

D3 - Including User Documentation

Introduction

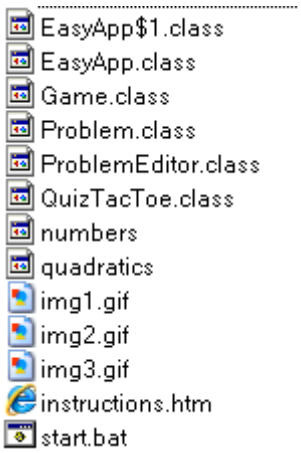
QuizTacToe is a traditional Tic-Tac-Toe game that requires the players to answer quiz questions in order to place an X or O. The question sets can be changed (by teachers) and chosen (by students) to provide appropriate review and practice for classroom tests. The Tic-Tac-Toe environment makes review a bit more fun than traditional study, and the competition with another student motivates students to do their best.

System Requirements

QuizTacToe has been tested on the Windows XP / PC platform, using Java version 1.5 . So it should run correctly in Windows as long as Java is installed. The installation instruction are written for the Windows platform. The program might run on other platforms as long as Java is installed, but this has not been tested, and the installation and start-up instructions will be different on those platforms.

Installation

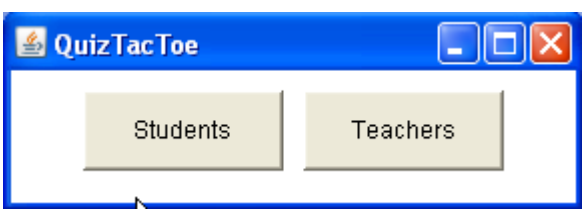
1. Create a folder (e.g. QuizTacToe) on a hard-disk or on a LAN server.
2. Copy the distribution files into the folder. This includes all the following files:



3. Ensure that all users have appropriate rights to the folder.

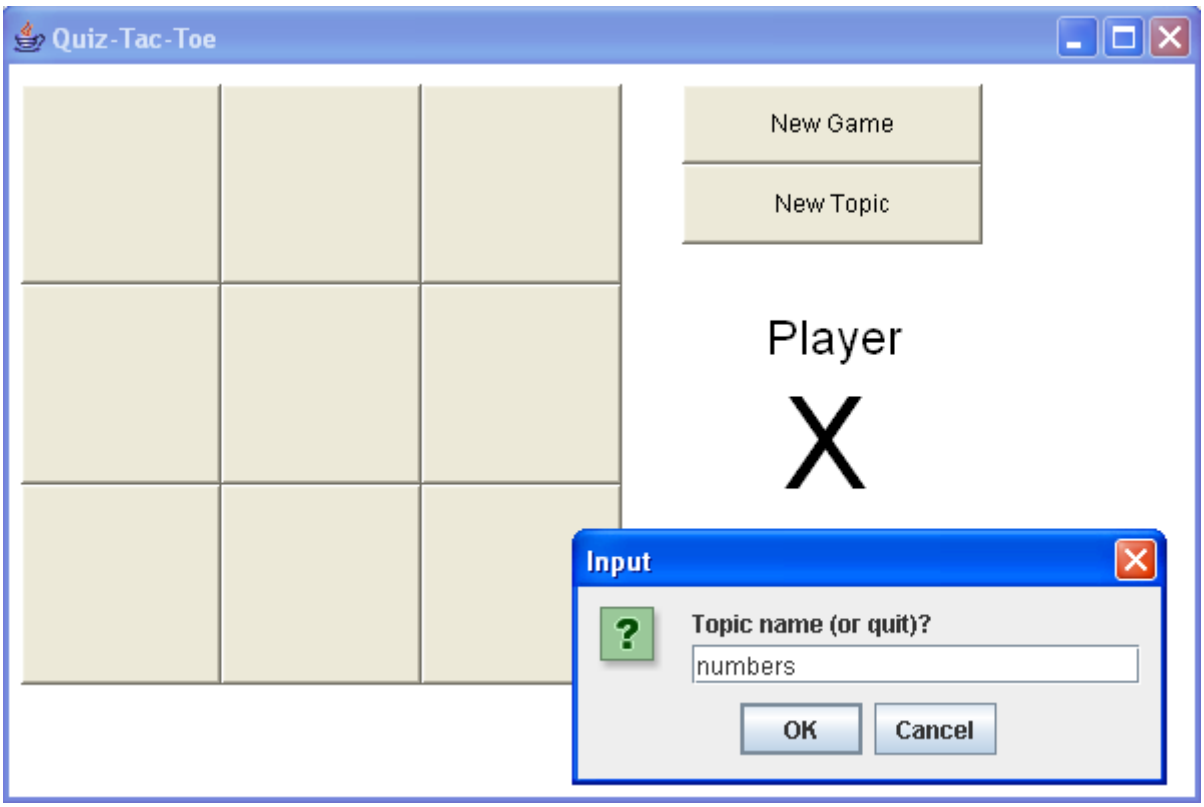
Starting the Program

Execute the **start.bat** file to run the program. You will see this splash screen:

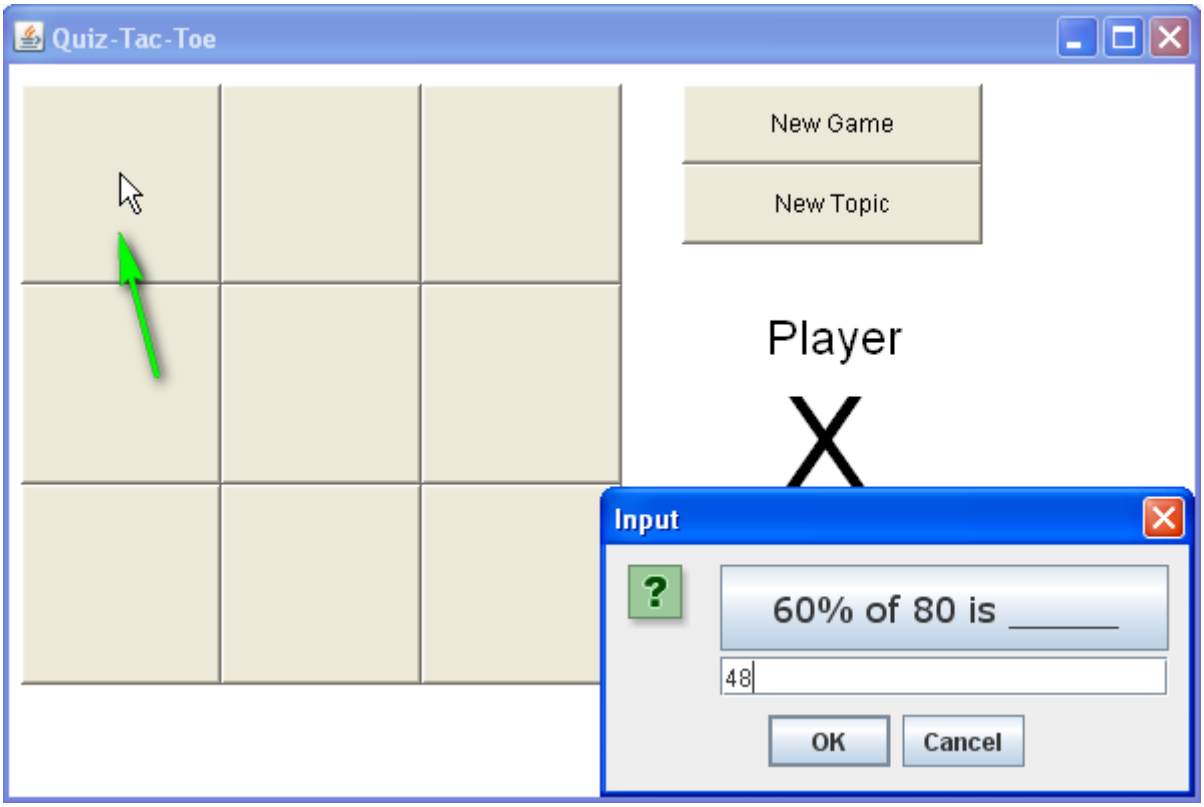


Click on [Students] to start the game, or [Teacher] to edit the problem data files.

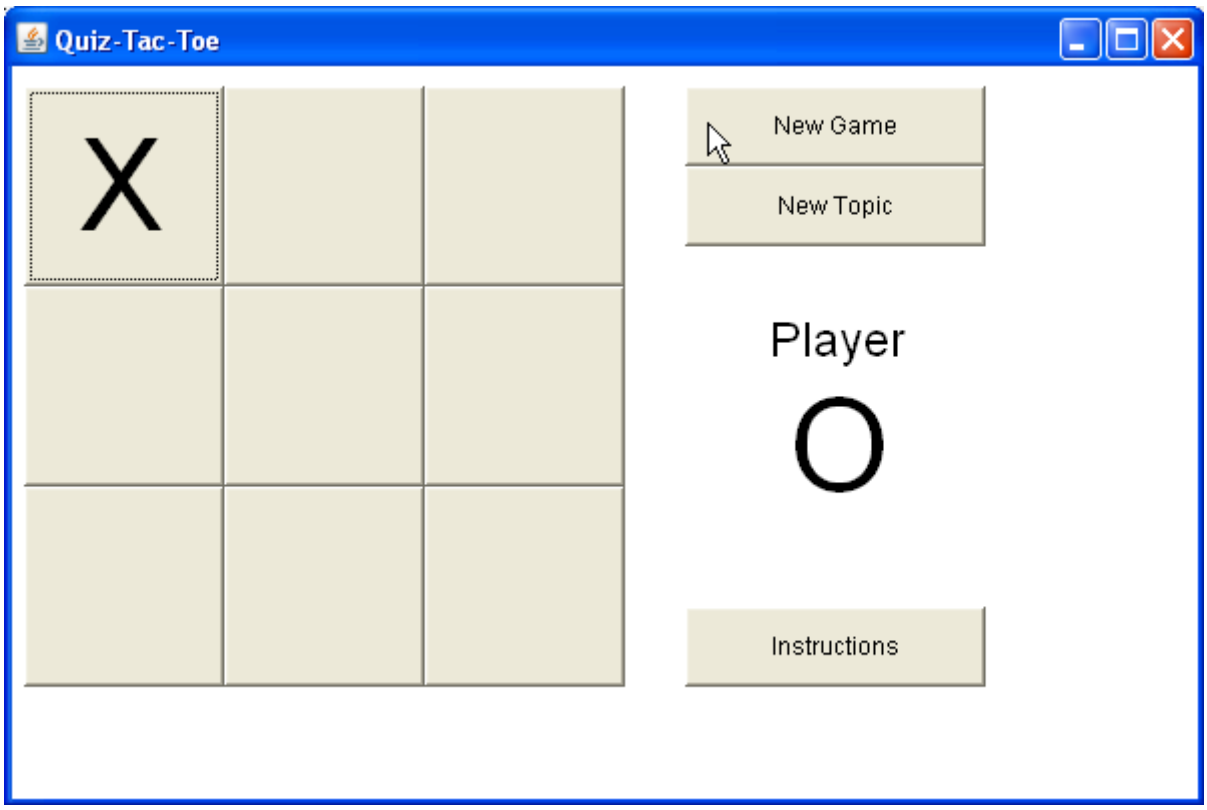
Playing the Game - When the game starts, players must type the name of a topic. The distribution include sample topics: **numbers** and **quadratics.**



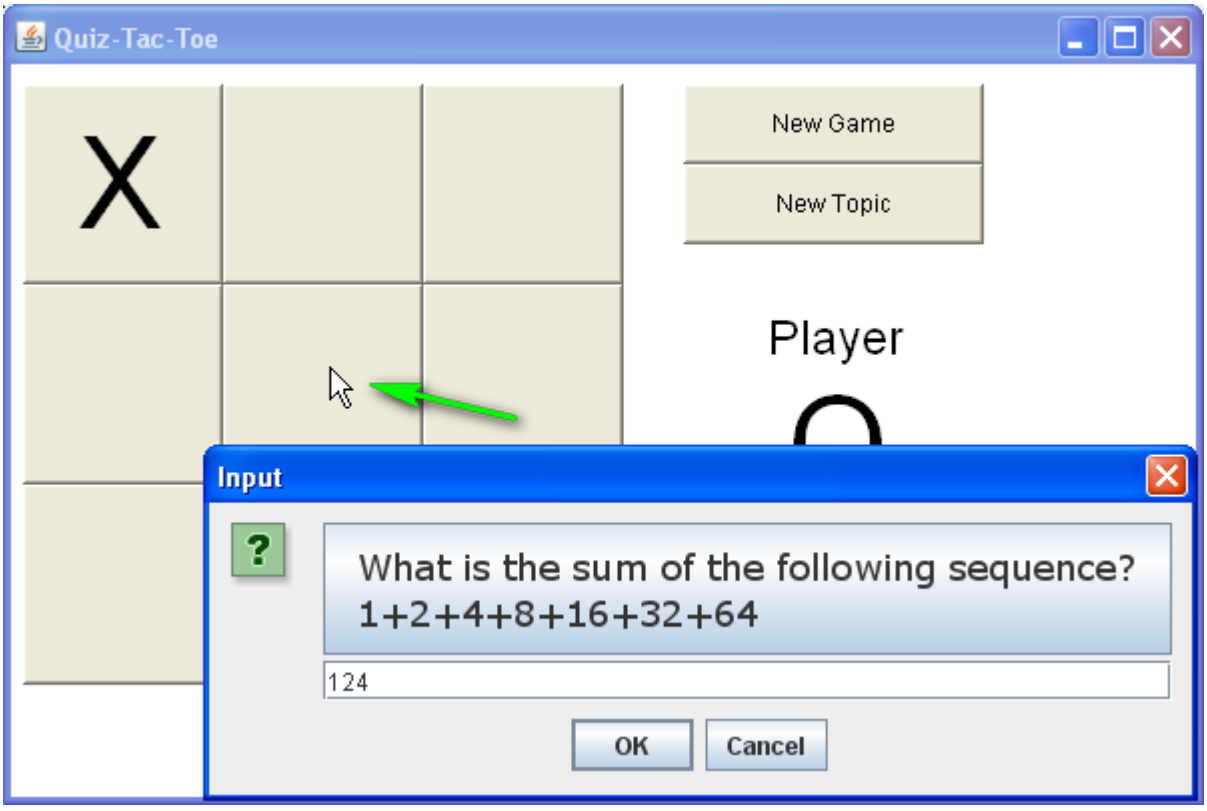
Choose a Square - Player X clicks on one of the squares to start the game. A question appears.



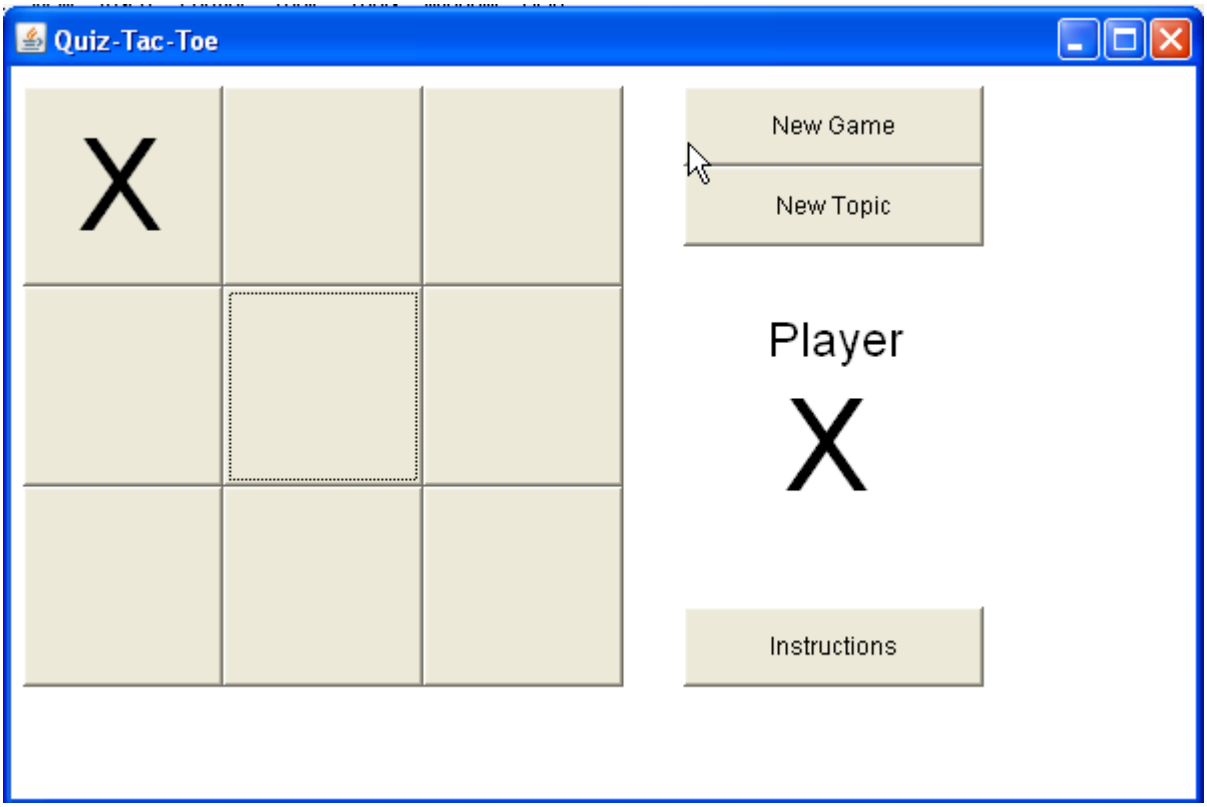
If Player X answers the question correctly an X appears on the board. If the answer is incorrect, he loses his turn.



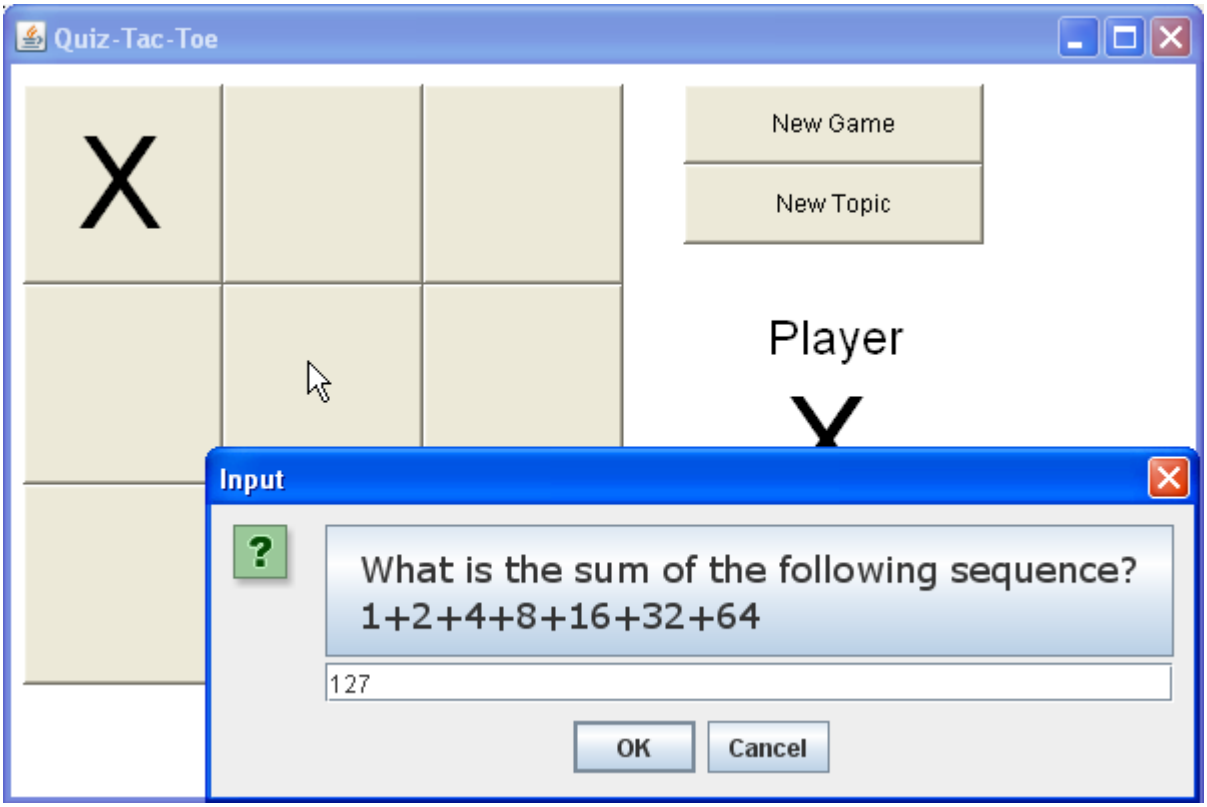
Now player O clicks on a square and answers a question.

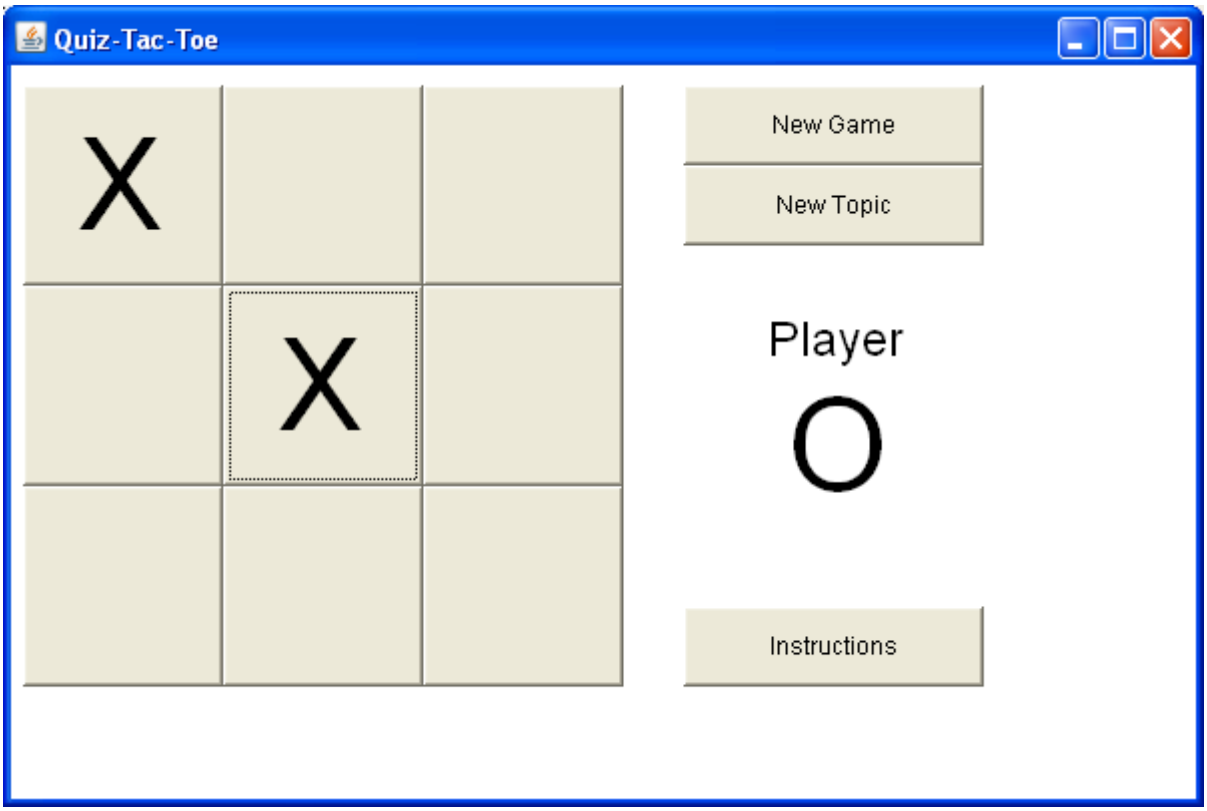


If the answer is incorrect, O loses his turn.

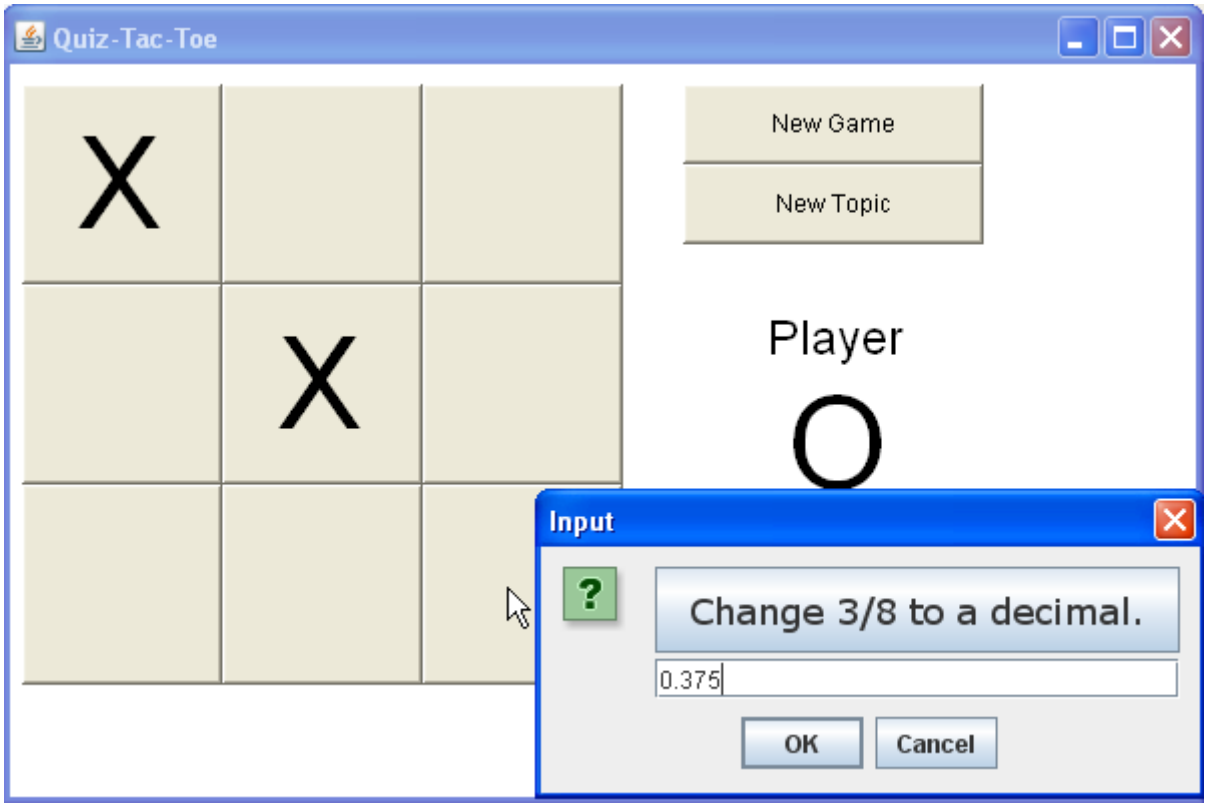


Play continues, alternating X and O turns until one player has three in a row.

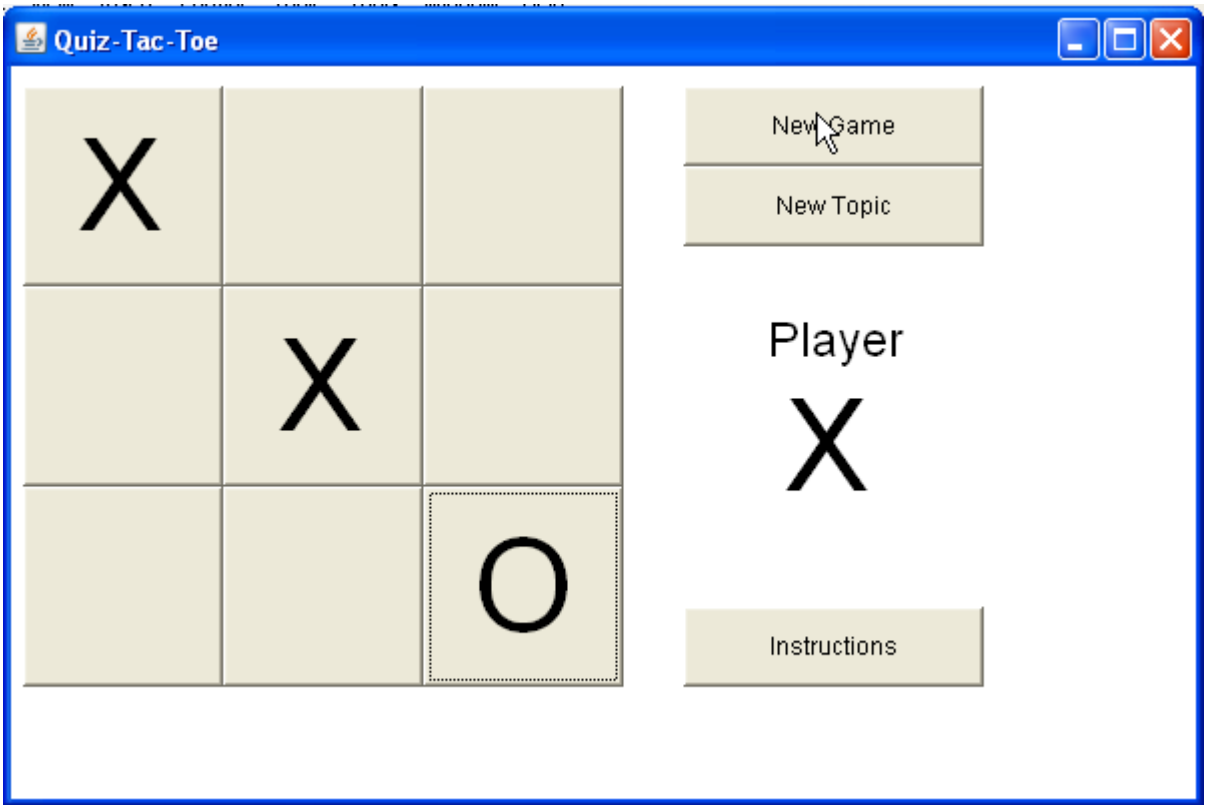




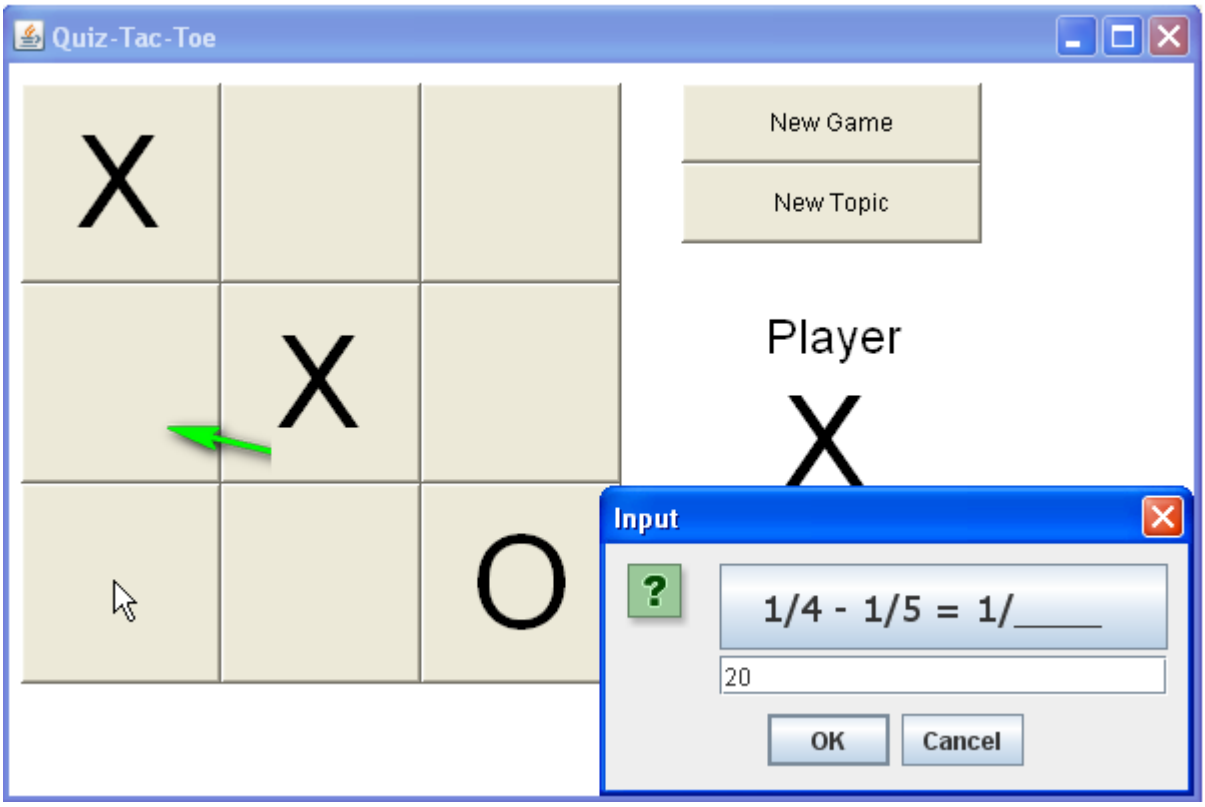
Player O attempts to block the 2 X's on the diagonal.



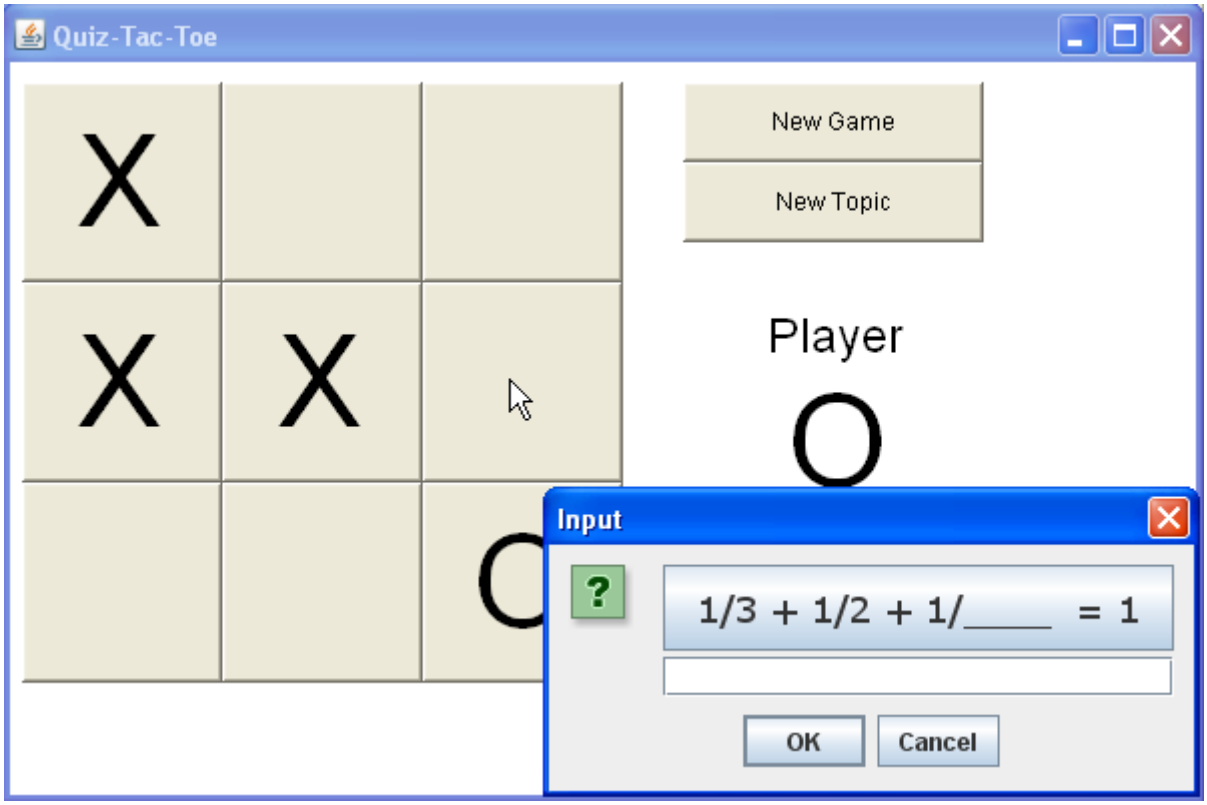
O answers correctly and blocks X.



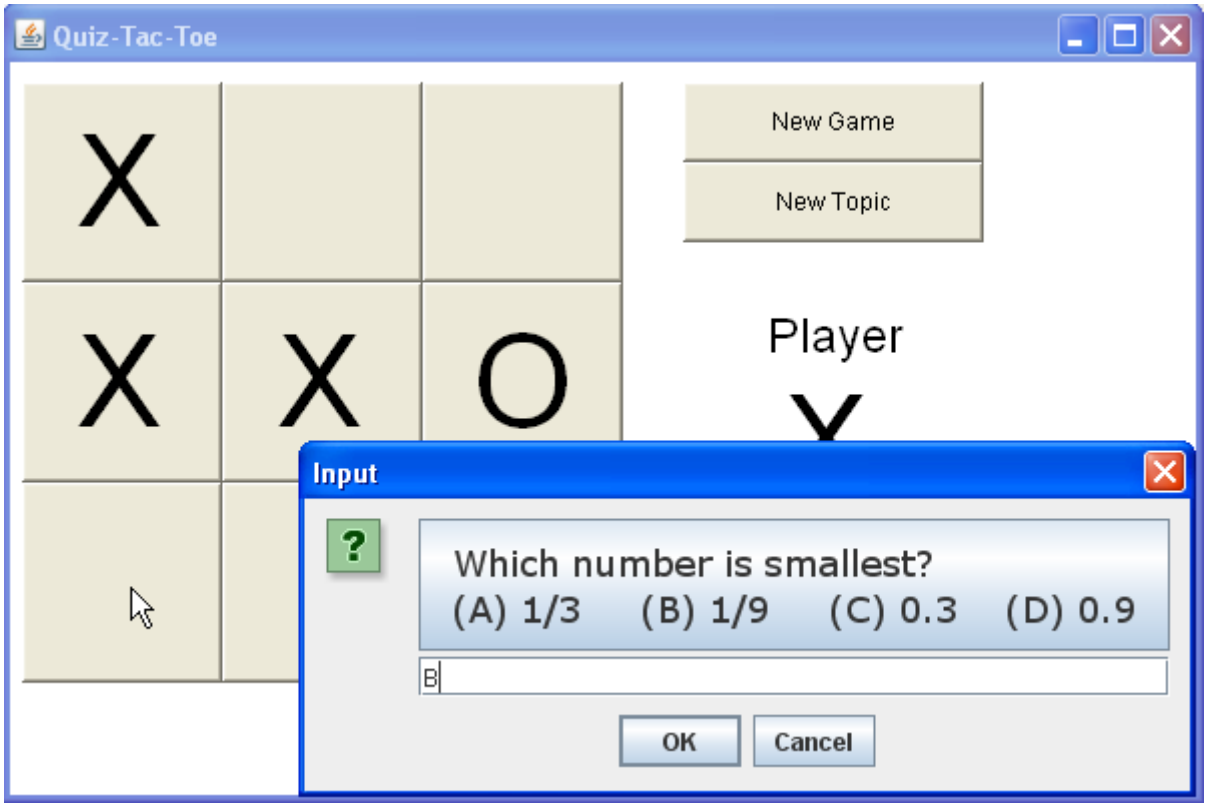
X goes for 2 sets of 2 in a row.



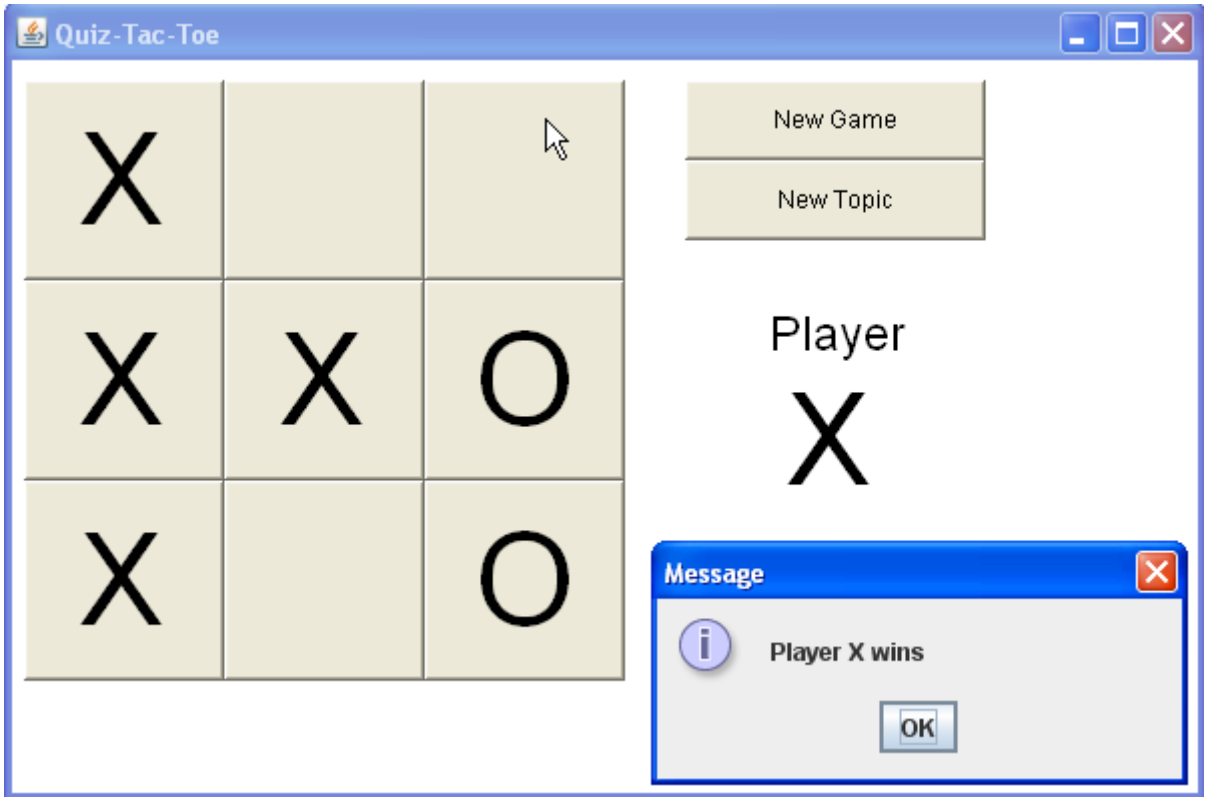
Success! Things look bad for player O. But he tries to block the middle row.



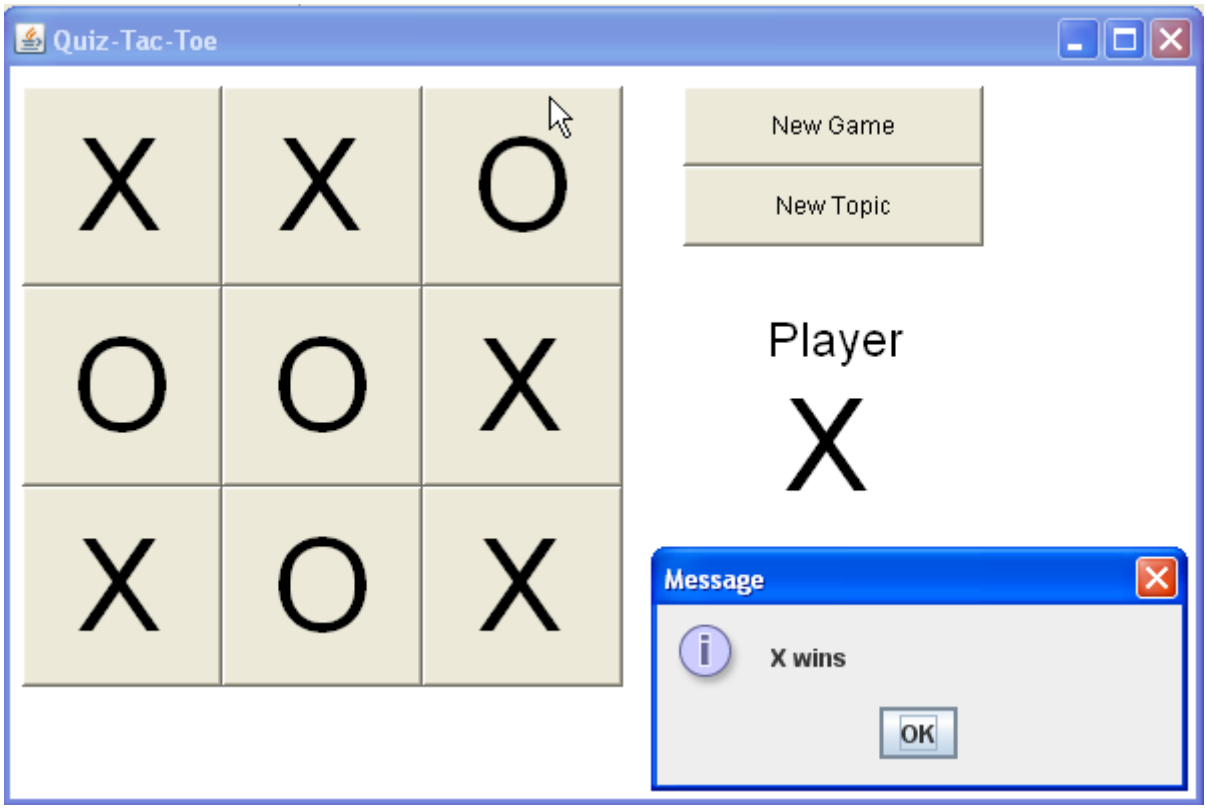
Player O answers correctly. Now X goes for the win in the left column.



Player X wins!



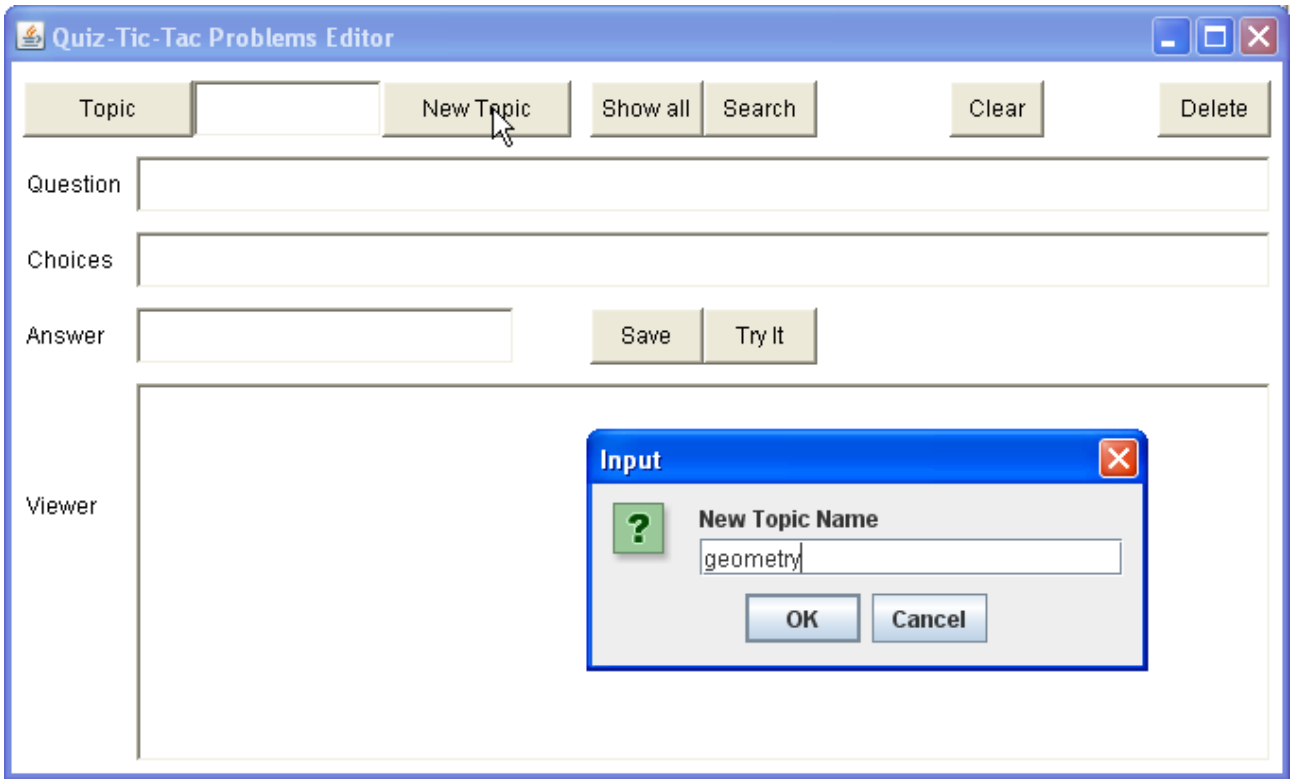
If the board gets full, with no 3-in-a-row, the player with the most squares wins.

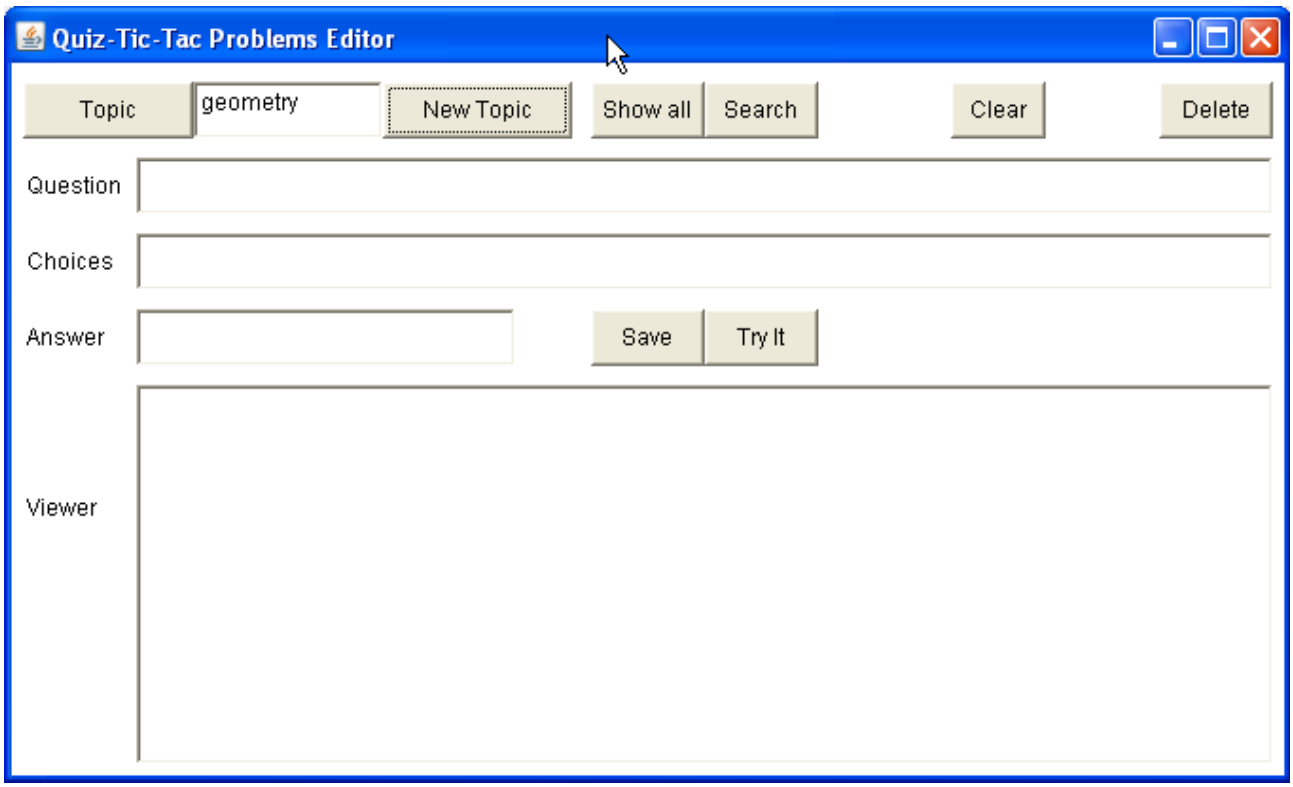


Question Sets

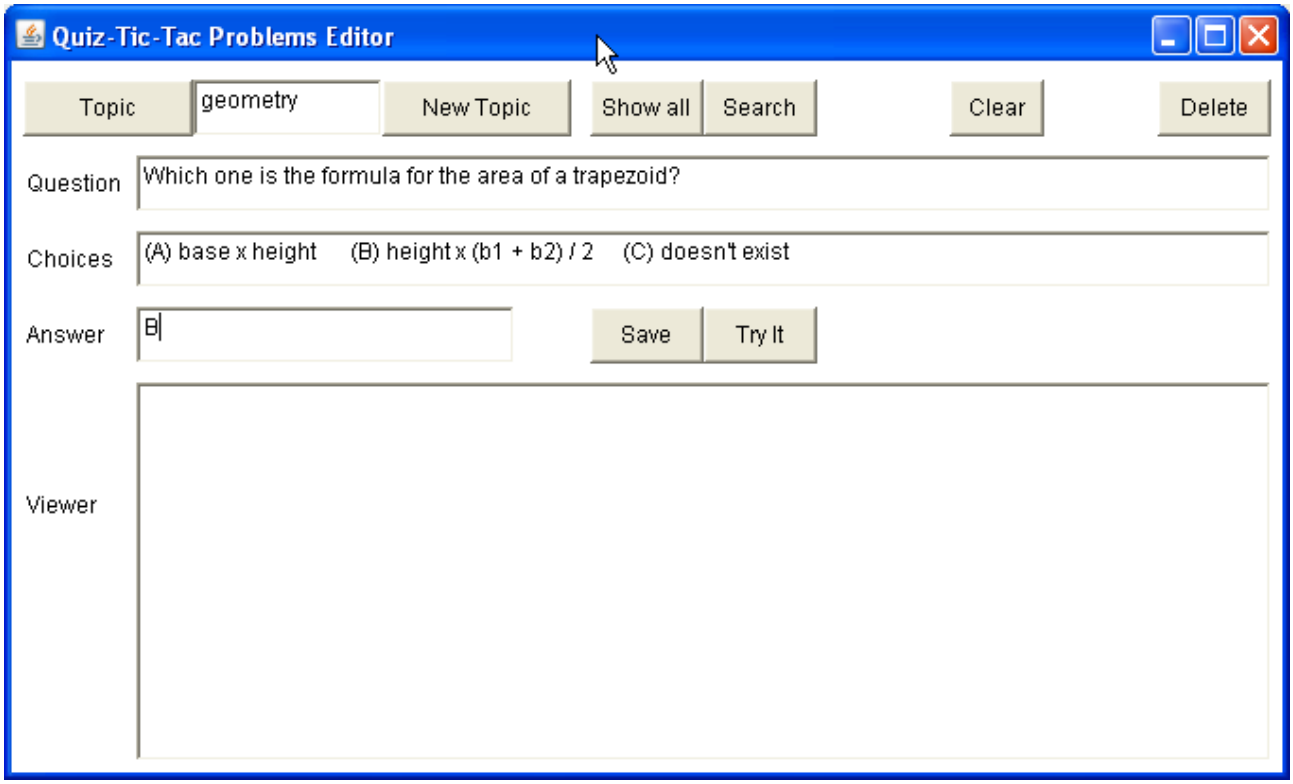
The distribution files include 2 sample question files: **numbers** and **quadratics**. But teachers will want to add more questions. Teachers should click on the [Teachers] button to start the Problem Editor module.

New Topic - the teacher clicks on [New Topic] to create a new (empty) problem set.

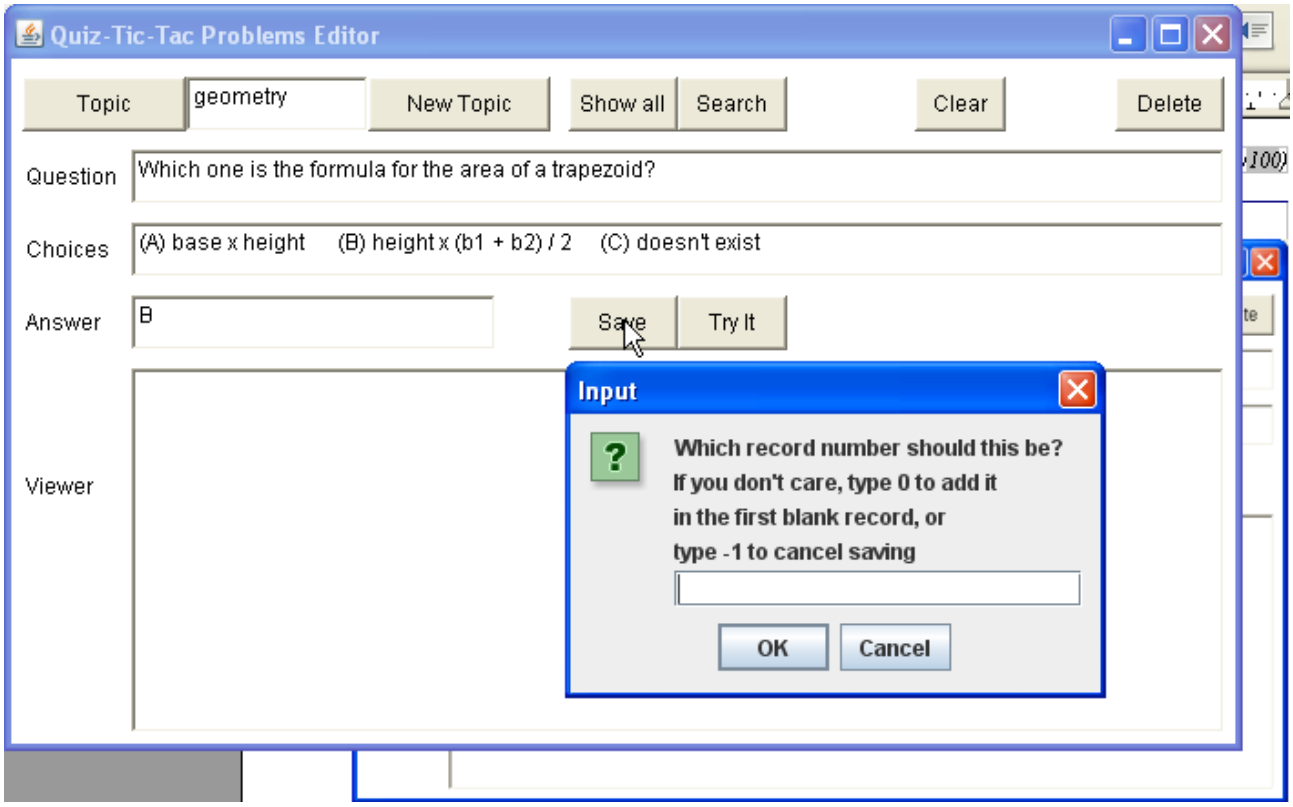




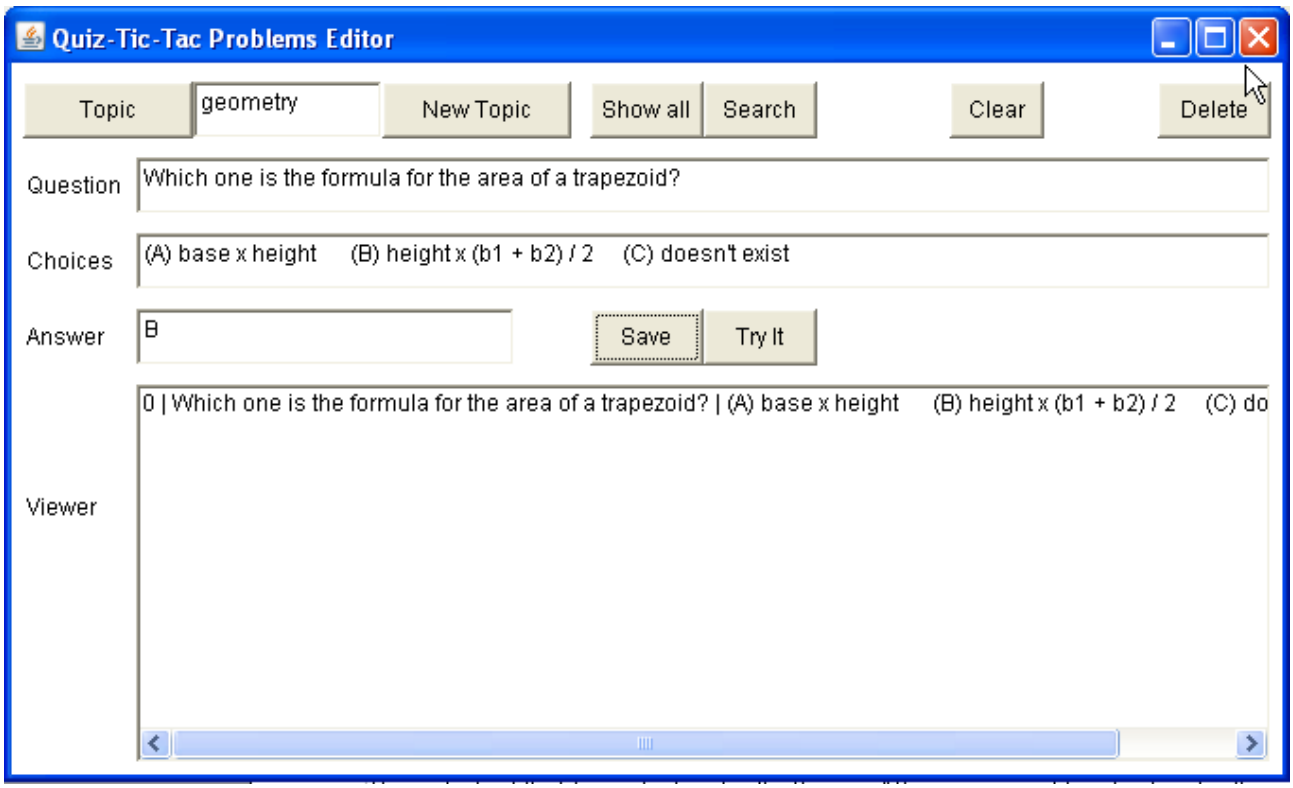
Type a new problem by filling in the Question, Choices, and Answer boxes.



Then click [Save] to save the problem - it is automatically added into the **geometry** file. Leave the record number blank (unless you are replacing an old problem).

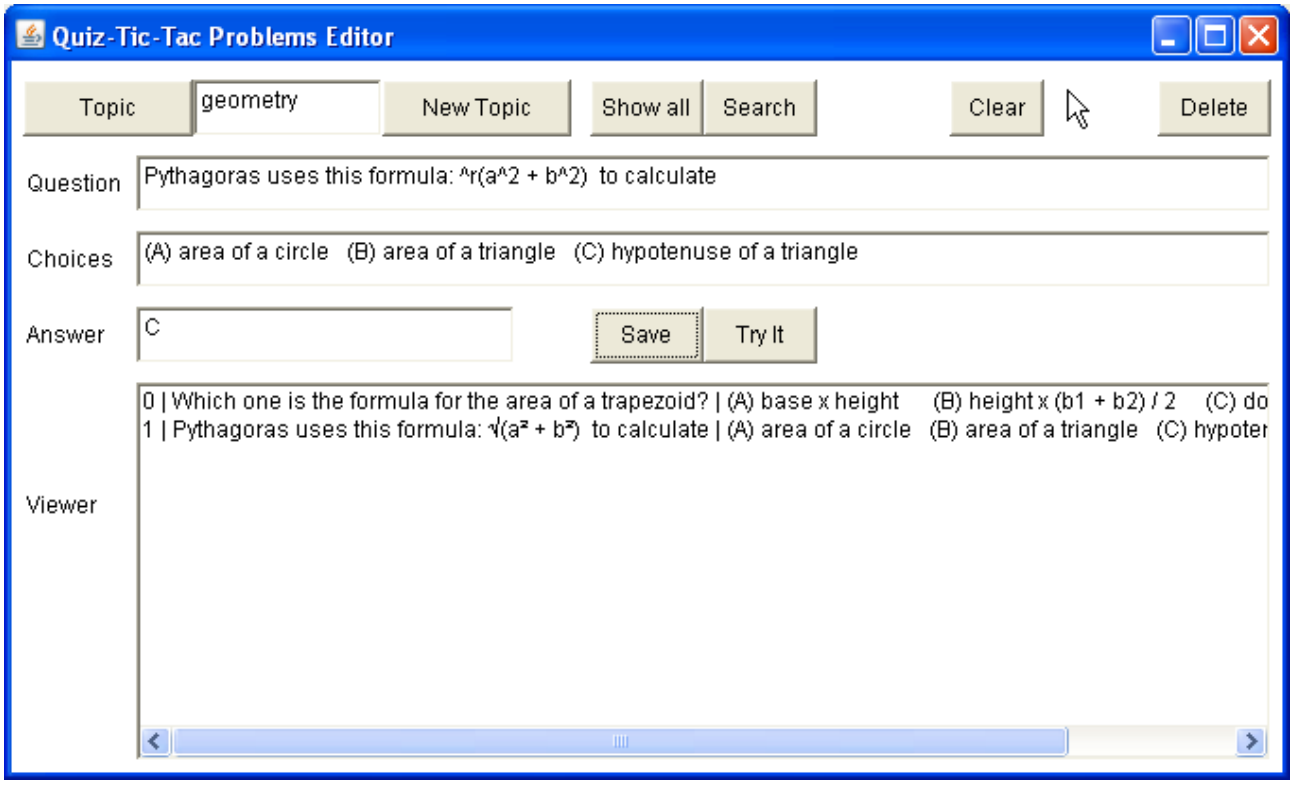


The contents of the file are displayed in the Viewer. After saving a problem it automatically appears in the viewer.



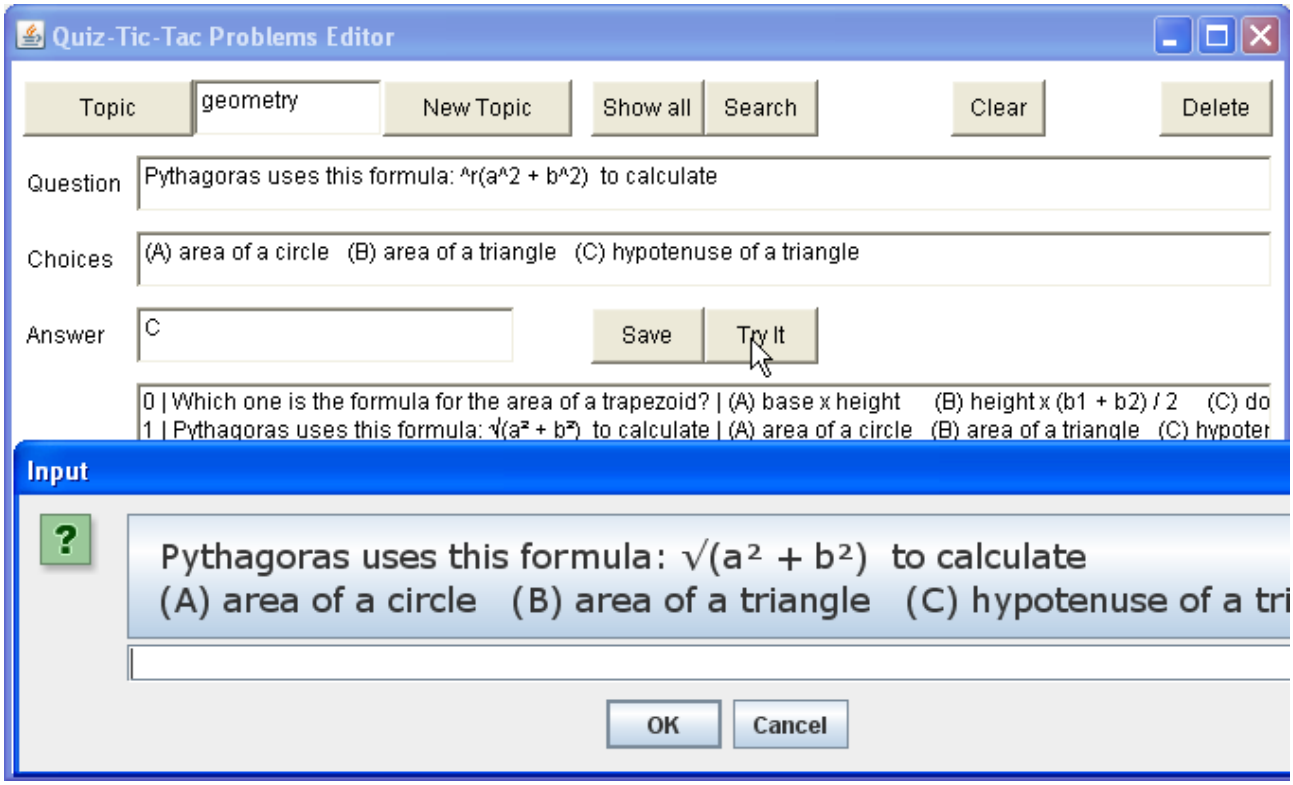
Some math symbols can be typed by using keyboard shortcuts:

^2 makes **squared** ^3 makes **cubed** ^r makes a **square-root sign**



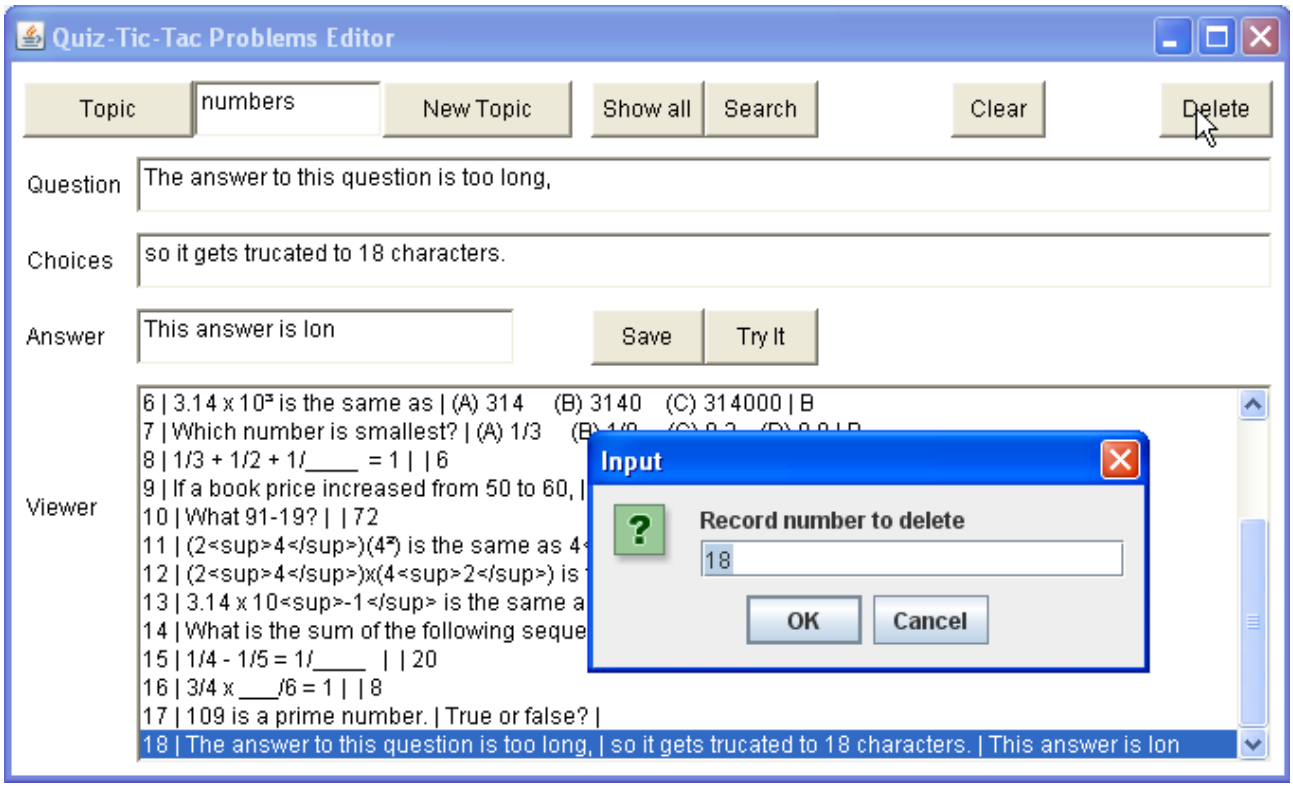
In the Game, the keyboard short-cuts will be presented as proper mathematical symbols.

The teacher can click [Try It] to see a "preview" of the problem, including proper math symbols.



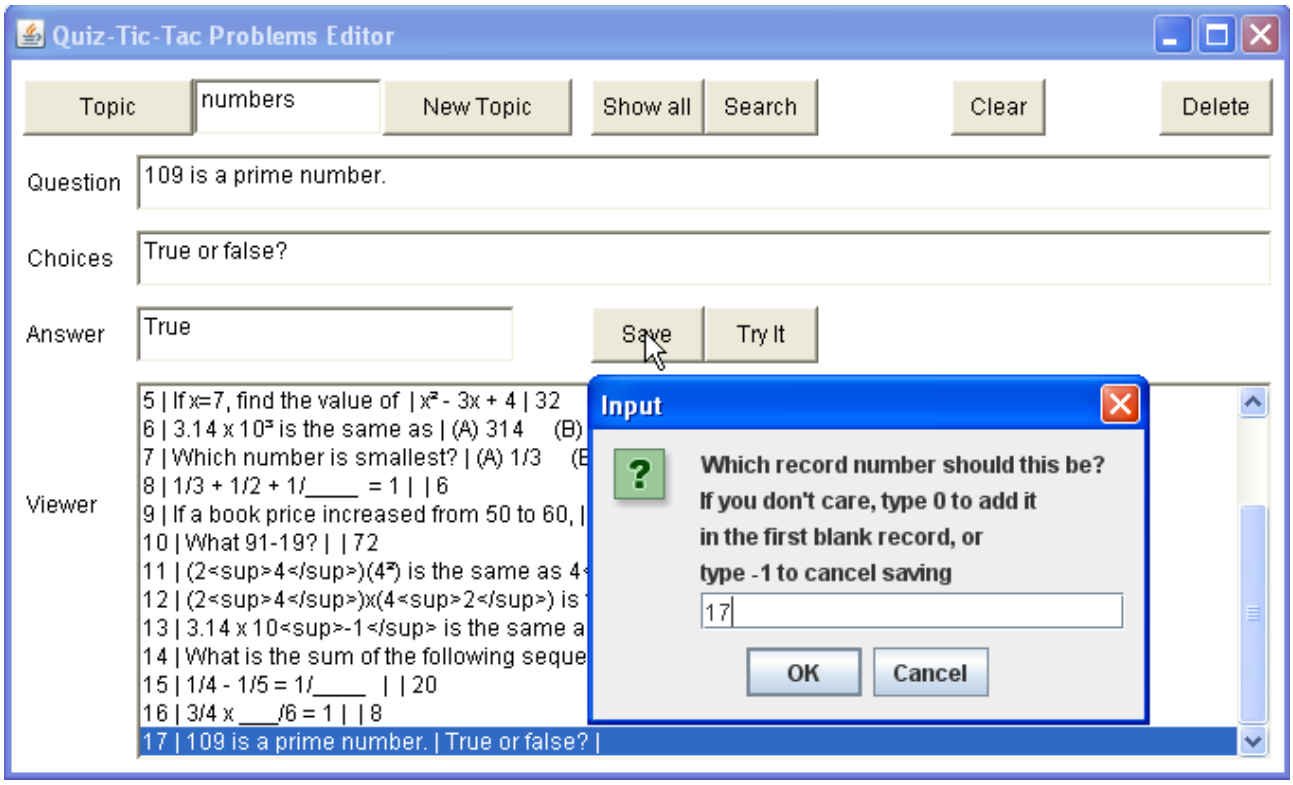
The problems all collect in the file and the entire list appears in the Viewer. The teacher can change and delete problems later, by clicking on a problem in the Viewer list.

Deleting a Problem



Editing a Problem

To edit a problem, click on the problem, make changes in the editing boxes, and then save the problem in the same record numbers.



Topics

The program does not maintain (or show) a list of topic files. So the teacher needs to tell the students the names of all the topic files that exist.

It probably makes sense to have several sets of topics - one for each class that will be using the program. In this case, there will be a folder for each class, containing the topic files for that class. In this case, each folder must also contain all the java .class files, as the program only opens files that are stored in the same folder with the program.

LAN Issues

If various folders are stored on a LAN server, the IT support staff should be asked to ensure that students have access rights for the folders.

There may also be some issues about installing the correct version of Java on the client PCs. Ask the IT staff to ensure that the latest version of Java is correctly installed on all the student PCs.

Mastery Factors

Mastery Factor	***Evidence***
Arrays	Button[] squares, Problem[] problems in Game class
User-defined objects	Problem class, used in Game class for problems array (lines 22, 115, 175, etc), used in ProblemEditor class for problem variable (lines 22, 137, 151, etc)
Objects as data records	Problem class (same as above) . This saves 3 data items - question, choices and answer, and implements appropriate get. and set. accessor methods.
Simple if..then..	many places, especially Game actions method (lines 53-107) and ProlemEditor saveProblem method (lines 146-190)
Complex if..then..	many places, especially Game checkFull method (lines 196-221)
Loops	many places, especially Game getRandomProblems method (lines 123-154)
Nested Loops	Game getRandomProblems method (lines 143-154)
User-defined methods	many
User-defined methods with parameters	many, especially Problem.replace method(lines 98-108)
User-defined methods with return values	many, especially ProblemEditor.firstNumber (line 304-324)
Sorting	------ not done -------
Searching	ProblemEditor.search (line 208)
File i/o	RandomAccessFiles in Problem class - loadProblem (line 137) and saveProblem (line 113)
Additional libraries	AWT GUI interfaces in Game and Teacher modules, especially Buttons for tic-tac-toe board in Game, and List box for problem viewer in ProblemEditor
Sentinels or flags	many methods return boolean flags signaling whether the method was successful or not.