AN IMPLEMENTATION IN GREENFOOT USED IN TEACHING PROGRAMMING TECHNIQUES

Adrian Runceanu, ***Constantin Brancusi University, Targu Jiu, ROMANIA***

Abstract: *In this paper we present a visual programming environment oriented, Greefoot, in which we can built it using applications like games. The approach in this paper proposes to use interactive teaching in Greenfoot. In this way the student can better understand concepts related to programming methods and especially may make changes, improvements at the implemented applications. The paper is implementing the Tower of Hanoi problem, practical application of the method of Divide and Conquer programming.*

Key words: *Programming techniques, Java, Greenfoot, Interactive development environment (IDE), Divide et Impera, Recursion.*

1. INTRODUCTION

Greenfoot is an interactive Java development environment designed primarily for educational purposes at the high school and undergraduate level. It allows easy development of two-dimensional graphical applications, such as simulations and interactive games.

Greenfoot is being developed and maintained at the University of Kent and La Trobe University, with support from Oracle. It is free software, released under the GPL license. Greenfoot is available for Microsoft Windows, Mac OS X, Linux, Sun Solaris, and any recent JVM.

Java is a programming language and computing platform first released by Sun Microsystems in 1995. It is the underlying technology that powers state-of-the-art programs including utilities, games, and business applications. Java runs on more than 850 million personal computers worldwide, and on billions of devices worldwide, including mobile and TV devices.

There are lots of applications and websites that won't work unless you have Java installed, and more are created every day. Java is fast, secure, and reliable. From laptops to datacenters, game consoles to scientific supercomputers, cell phones to the Internet, Java is everywhere!

The language derives much of its syntax from C and C++, but has fewer low-level facilities than either of them. Java applications are typically compiled to byte code (class file) that can run on any Java Virtual Machine (JVM) regardless of computer architecture. Java is a general-purpose, concurrent, class-based, object-oriented language that is specifically designed to have as few implementation as possible. It is intended to let application developers "write once, run anywhere" (WORA), meaning that code that runs on one platform does not need to be recompiled to run on another. Java is as of 2012 one of the most popular programming languages in use, particularly for client-server web applications, with a reported 10 million users.

We will present one of the most popular Java tool - Greenfoot. This envirovment is used to create animation and for creating interactive games that involve human interactions, decisions, and actions with 2D objects. Greenfoot are very useful tools for learning Java:

1. It teaches the basics of Java syntax and object orientation which makes developing desktop Java applications easier than starting from scratch.
2. Its interface is an interactive development environment (IDE) that allows you to edit source code, compile, and debug, just like in other Java IDE's.

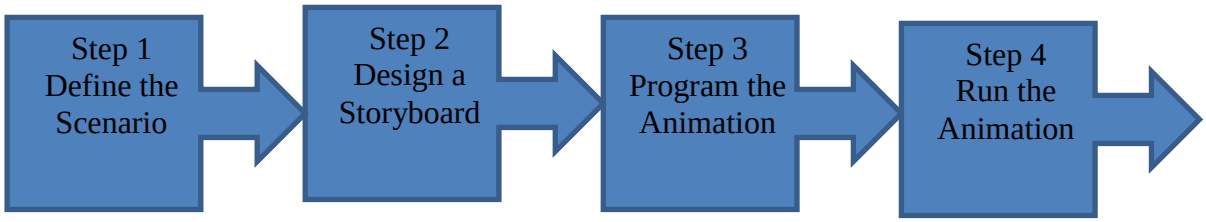
Greenfoot will help users learn to program in Java. In order to use nd Greenfoot, however, we need to learn certain skills to create animations and games. Here is a high level overview of the steps involved in creating an animation or game:

1. Define a scenario
   - What story is to be told?
   - What objects are needed?
   - What actions will take place?
2. Design the storyboard for the scenario – Visual or textual (or both)
3. Create the animation or game
4. Test

A **scenario** contains three parts:

1. **Story:** The story to tell, or game to play. For example, a flying frog will catch flies and eat them.
2. **Objects**: The objects you will use in your story. For example, a frog and flies.
3. **Actions**: All the actions the objects will take.

Animation Development Process



2. EXISTING TECHNIQUES

The main tool for programming in Greenfoot is the code editor. The code editor displays the source code for the class. The source code of a class is the code that specifies all of the properties and characteristics of that class and its objects. The programmer can command the objects in his scenario to perform tasks or answer questions by writing source code, or syntax, in the Java programming language. When selecting the Open Editor from the class's menu to see the editor window that contains the class's source code. The source code displayed defines what the objects of the class can do.

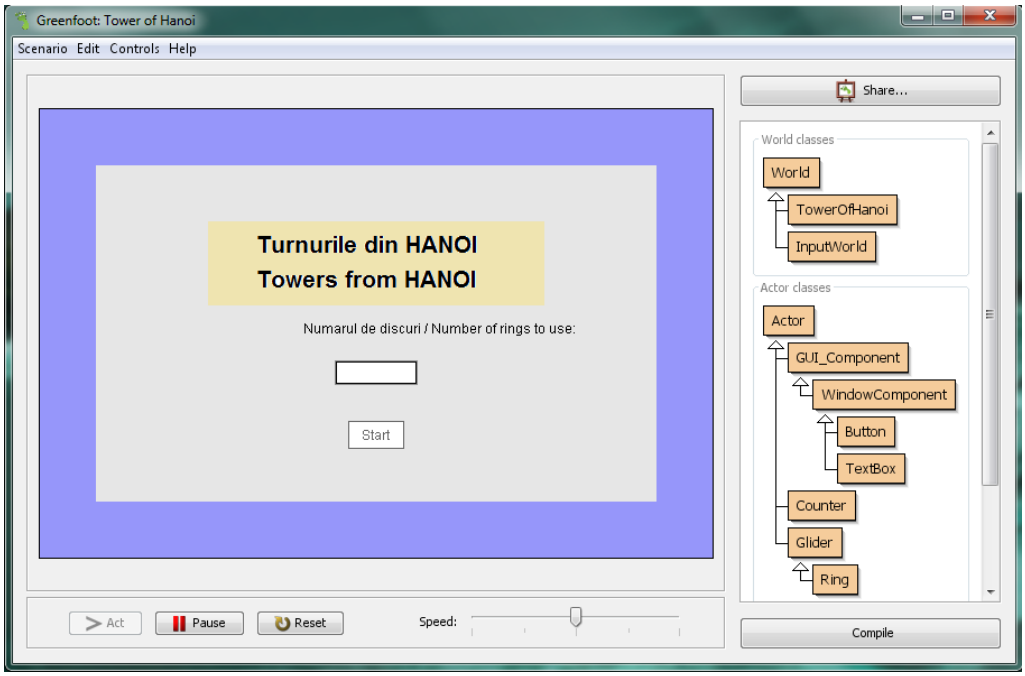


Figure 1 – the Greenfoot window

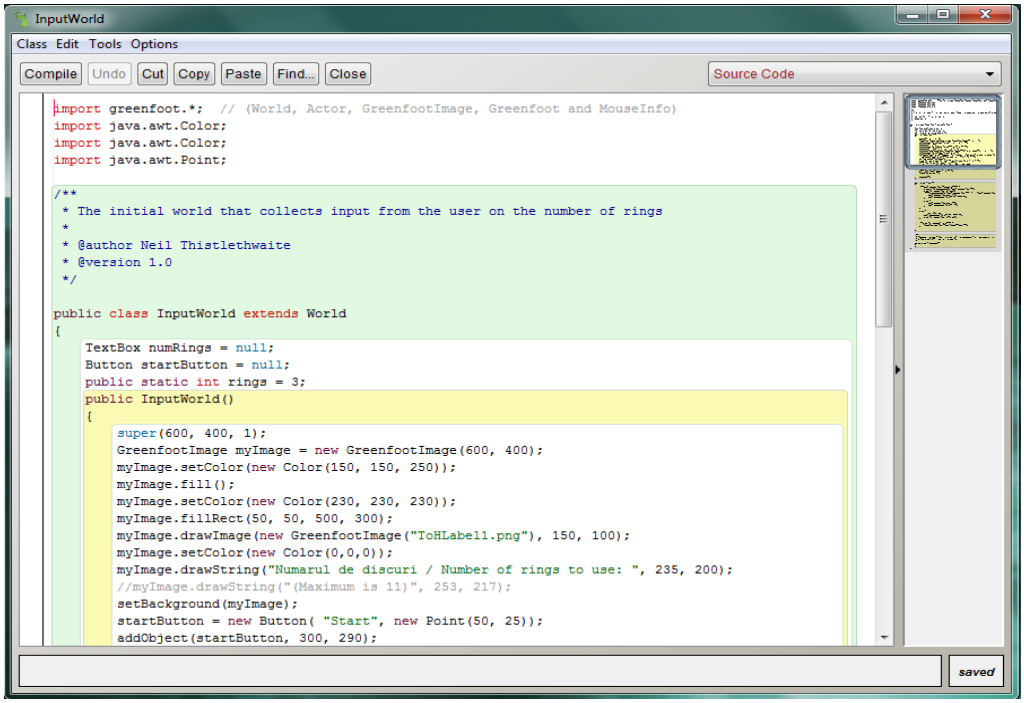


Figure 2 – Code editor in Greenfoot

In the code editor (Figure 2), the programmer can:

- Write source code to tell instances of the class how to act
- Review a class's inherited methods and properties, to understand what actions the instances are capable of taking
- Review methods created specifically for the class by the programmer who wrote the source code
- Modify existing source code to change an instance's behavior

The Greenfoot programming model consists of a World class (represented by a rectangular screen area) and any number of actor objects that are present in the world and can be programmed to act independently. The world and actors are represented by Java objects and defined by Java classes. Greenfoot offers methods to easily program these actors, including method for movement, rotation, changes of appearance, collision detection, etc.

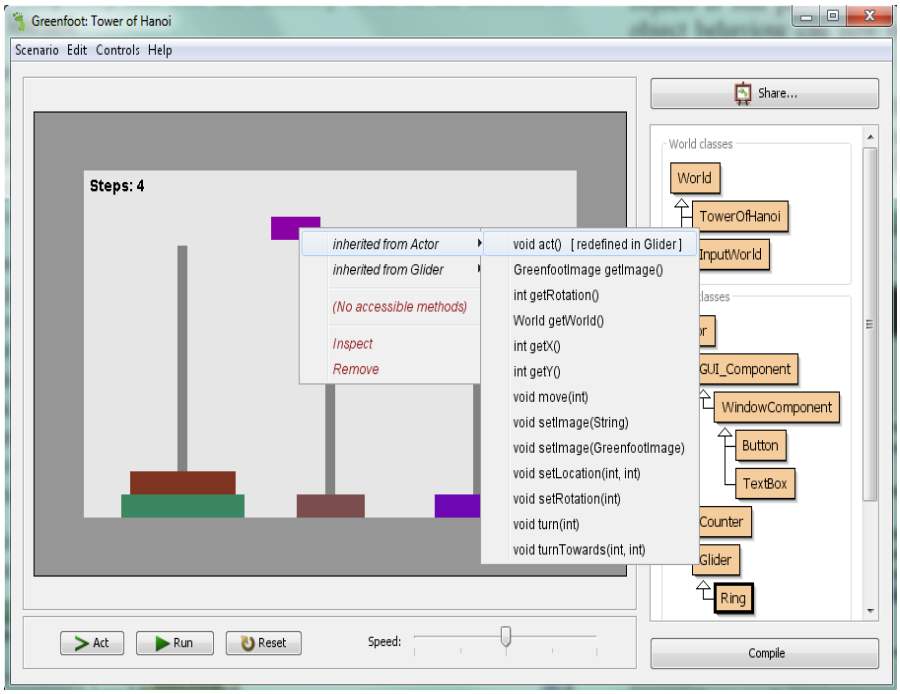


Figure 3 – Main classes in Greenfoot: World and Actor

Programming in Greenfoot at its most basic consists of subclassing two built-in classes, World and Actor (Figure 3). An instance of the world subclass represents the world in which Greenfoot execution will occur. Actor subclasses are objects that can exist and act in the world. An instance of the world subclass is automatically created by the environment.

Execution in Greenfoot consists of a built-in main loop that repeatedly invokes each actor's act method.

Programming a scenario, therefore, consists mainly of implementing act methods for the scenario's actors. Implementation is done in standard Java. Greenfoot offers API methods for a range of common tasks, such as animation, sound, randomisation, and image manipulation. All standard Java libraries can be used as well, and sophisticated functionality can be achieved.

The largest part of greenfoot's user interface is reserved for the display of the world, shown in the centre of the screen (Figure 1). It holds the greenfoot objects (two greenfoot robots, a beeper and some walls in this example). To the right of the world is a class display. Here, all classes involved in the current application are shown along with buttons to compile and create new classes. The classes are divided into **Greenfoot-World Classes** and **Greenfoot-Object Classes**.

The lower part of the window (Figure 4) holds execution controls to run, stop or single-step the simulation and a slider to control the execution speed.

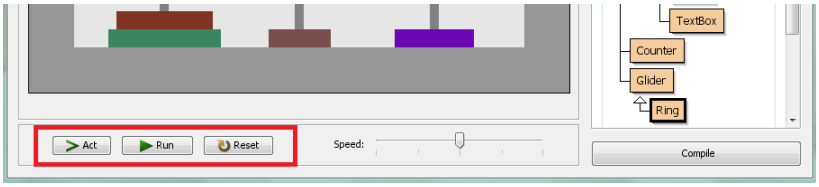


Figure 4 – The bottom part of the Greenfoot window

GreenfootObject classes are the classes that are to be visualised in the world. Their superclass – GreenfootObject – will always be shown in the class browser. The GreenfootObject class cannot be modified.
Subclasses of GreenfootObject will typically have an individual icon. This icon is shown in the representation of the class next to the class name. Greenfoot objects that do not specify an appearance have a default look defined in their superclass.
GreenfootWorld classes are classes that represent worlds. Different worlds may exist in a single project (holding, for example, different initial populations of walls and beepers).
The superclass of these – GreenfootWorld – will always be shown in the class browser. The GreenfootWorld classes have popup menus exactly like the ones described for GreenfootObject classes. When a constructor is selected for a subclass of GreenfootWorld, the new world object will automatically replace the existing world in the main view of the greenfoot user interface.

3. OUR CONTRIBUTIONS

For course Designing Algorithms we developed some applications in Java, with IDE Greenfoot, for helping students to understand programming techiniques. We try to present a lot of programming techniques: Recursion, Lists, Stacks, Queques, Divide et Impera, Bracktracking, Greedy and others.
Divide and conquer method has many practical applications. One problem is known as the Towers of Hanoi.
In brief, method consist in:
We consider three towers numbered A, B, C and n perforated disks having different diameters.
Initially all disks are placed on the tower A, in ascending order of their diameters, considering the direction of the top of the tower at its base.
To move all the disks on tower B in the same order using Tower C and the following rules:

1. at each step moves one disk
2. continuously on each individual turn disc may occur only above the smaller diameter discs.

Solving this problem is based on the following considerations logic:

- if n = 1, then A -> B is immediate move (move disc from A to B)
- if n = 2, then the sequence moves is: A -> C, A -> B, C -> B
- if n > 2 the following:
  - Move (n-1) disks A -> C
  - Moving a disc A -> B
  - Move the (n-1) disks C -> B

Note that the initial problem is decomposed into three simpler subproblems similar initial problem:

- move (n-1) disks A -> C
- move the last disk B
- move the (n-1) disks C -> B

The size of these sub-problems are: 1 n-1, n-1. These subproblems are independent because the initial tower (which are arranged discs) final tower and the tower intermediate are different.

We make a note: **H (n, A, B, C) = n moves from tower A to tower B, using tower C**
For
n = 1, we make move A -> B

n> 1, we call recursive function H (n, A, B, C) = H (n-1, A, C, B), A -> B, H (n-1, C, B, A)

Implementation of the solution in Greenfoot (Java):

```
import greenfoot.*;
import java.awt.Color;
import java.util.ArrayList;

public class TowerOfHanoi extends World
{
   int numrings = 5;
   int start = 1;
   int finish = 3;
   int oinstructionSubStep = 1;
   int instructionSubStep = 1;
   int currentInstruction = 0;
   Instruction icurrentInstruction = null;
   Counter stepCounter = new Counter("Steps:
");
   boolean finishedInstruction = true;
   Stack stack1 = new Stack(1, 150);
   Stack stack2 = new Stack(2, 300);
   Stack stack3 = new Stack(3, 450);
   Ring moving;
   ArrayList<Ring> rings;
   ArrayList<Color> colors = new
ArrayList<Color>(21);
   ArrayList<Instruction> instructions;

   public TowerOfHanoi() {
      super(600, 400, 1);
      int numOfRings = InputWorld.rings;
      numrings = numOfRings;
      rings = new ArrayList<Ring>(numrings+1);
      instructions = new
ArrayList<Instruction>((int)(Math.pow(2.0,
(numrings*1.0)))+2);
      GreenfootImage myImage = new
GreenfootImage(600, 400);
      myImage.setColor(new Color(150, 150,
150));
      myImage.fill();
      myImage.setColor(new Color(230, 230,
230));
      myImage.fillRect(50, 50, 500, 300);
      myImage.setColor(new Color(130, 130,
130));

      int width = 10;
      int halfwidth = (int)(width/2.0);
      myImage.fillRect(stack1.getXLoc()-
halfwidth, 115, width, 235);
      myImage.fillRect(stack2.getXLoc()-
halfwidth, 115, width, 235);
      myImage.fillRect(stack3.getXLoc()-
halfwidth, 115, width, 235);

      setBackground(myImage);
      addObject(stepCounter, 100, 64);
      makeInstructions();
      makeColors();
      makeRings(numrings);
      addRings();
   }

public void addRings()
   {
      for(int a = 0; a < rings.size(); a ++) {
         Ring b = rings.get(a);
         stack1.addRing(b);
         addObject(b, stack1.getXLoc(), 360 - (20
* stack1.getRings()));
      }
   }

public void makeRings(int ringsToAdd)
   {
      //All the ring sizes between 125 and 75
      double increment = (75.0/(ringsToAdd-1));
      for(int a = 0; a < ringsToAdd; a++) {
         rings.add(new Ring(colors.get(a), 125 -
(int)(a * increment)));
      }
   }

public void makeColors()
   {
      for(int a = 0; a < 20; a++) {
         colors.add(new
Color(Greenfoot.getRandomNumber(150),
Greenfoot.getRandomNumber(200),
Greenfoot.getRandomNumber(200)));
      }
   }

public void doInstructionSteps()
   {
      if(instructionSubStep == 0) {
         if(!moving.moving())
instructionSubStep = oinstructionSubStep + 1;
      }
      else if(instructionSubStep == 1) {
         // Glide to (CurrentX, 100)
         if(icurrentInstruction.getFrom() == 1) {
moving = stack1.getTopRing();
stack1.removeTopRing(); }
         else if(icurrentInstruction.getFrom() ==
2) { moving = stack2.getTopRing();
stack2.removeTopRing(); }
         else { moving = stack3.getTopRing();
stack3.removeTopRing(); }
         if(moving != null) {
moving.glideTo(moving.getX(), 100, 30); }
```

```
        oinstructionSubStep = instructionSubStep;
        instructionSubStep = 0;
      }
      else if(instructionSubStep == 2) {
        int destinationX = 0;
        if(icurrentInstruction.getTo() == 1) { destinationX = stack1.getXLoc(); }
        else if(icurrentInstruction.getTo() == 2) { destinationX = stack2.getXLoc(); }
        else { destinationX = stack3.getXLoc(); }
        moving.glideTo(destinationX, moving.getY(), 20);
        oinstructionSubStep = instructionSubStep;
        instructionSubStep = 0;
      }
      else if(instructionSubStep == 3) {
        int destinationY = 0;
        if(icurrentInstruction.getTo() == 1) {
          destinationY = 340 - (20 * (stack1.getRings())); }
        else if(icurrentInstruction.getTo() == 2) { destinationY = 340 - (20 * (stack2.getRings())); }
        else { destinationY = 340 - (20 * (stack3.getRings())); }
        moving.glideTo(moving.getX(), destinationY, 30);
        oinstructionSubStep = instructionSubStep;
        instructionSubStep = 0;
      }
      else if(instructionSubStep == 4) {
        if(icurrentInstruction.getTo() == 1) { stack1.addRing(moving); }
        else if(icurrentInstruction.getTo() == 2) { stack2.addRing(moving); }
        else { stack3.addRing(moving); }
        incrementCounter();
        finishedInstruction = true;
      }
    }

public void incrementCounter() { stepCounter.add(1); }
    public void act()
    {
      if(!(stack3.getRings() == numrings)) {
        if(finishedInstruction)
        {
          currentInstruction += 1;
          oinstructionSubStep = 1;
          instructionSubStep = 1;
          icurrentInstruction = instructions.get(currentInstruction-1);
          finishedInstruction = false;
        }
        doInstructionSteps();
      }
    }

public Instruction getInstruction(int step) { return instructions.get(step); }
    public void clearOutput()
    {
      for(int a = 0; a < 20; a++) { System.out.println(" "); }
    }

public void makeInstructions() { HTower(numrings, start, finish); }
public void HTower(int numRings, int current, int destination)
    {
      int spare = 0;
      if(current == 1) {
        if(destination == 2) { spare = 3; }
        else { spare = 2; }
      }
      else if(current == 2) {
        if(destination == 1) { spare = 3; }
        else { spare = 1; }
      }
      else {
        if(destination == 2) { spare = 1; }
        else { spare = 2; }
      }
      Instruction returnInstruction = null;

      if(numRings != 1) {
        HTower(numRings -1, current, spare);
        instructions.add(new Instruction(current, destination));
        HTower(numRings -1, spare, destination);
      }
      else {
        instructions.add(new Instruction(current, destination));
      }
    }
```

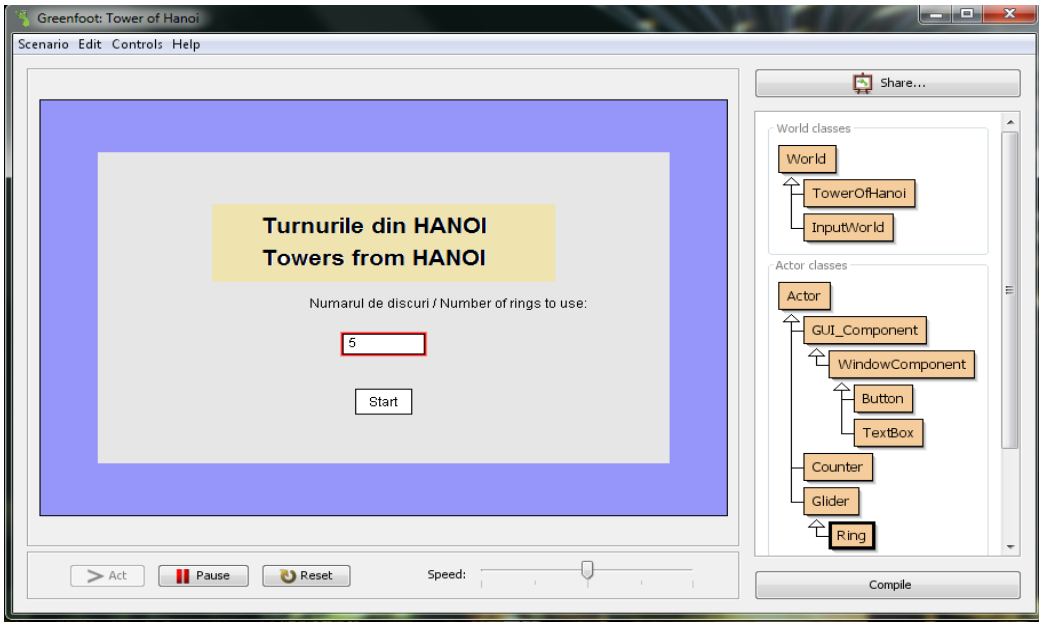


Figure 5 – Recursion with Greenfoot Towers of Hanoi Game

The game explains the movements required to transfer be made to move an entered number of discs from the first tower to the last, using as an intermediate tower the middle one. The player can move only a disc at a time, and a disc cannot be placed on top of a smaller one. Students can play the game varying the input data, which is the number of discs, just be clicking on the Act button (Figure 5).

Once the game has been played, student is requested to edit the Towers class to see the recursive function used (Figure 6). The variables origin, destination and aux correspond to the first, third and second tower respectively, and the n contains the entered number of discs for that execution. Students can now understand how a recursive function makes call to itself, and what are the parameters on every call.

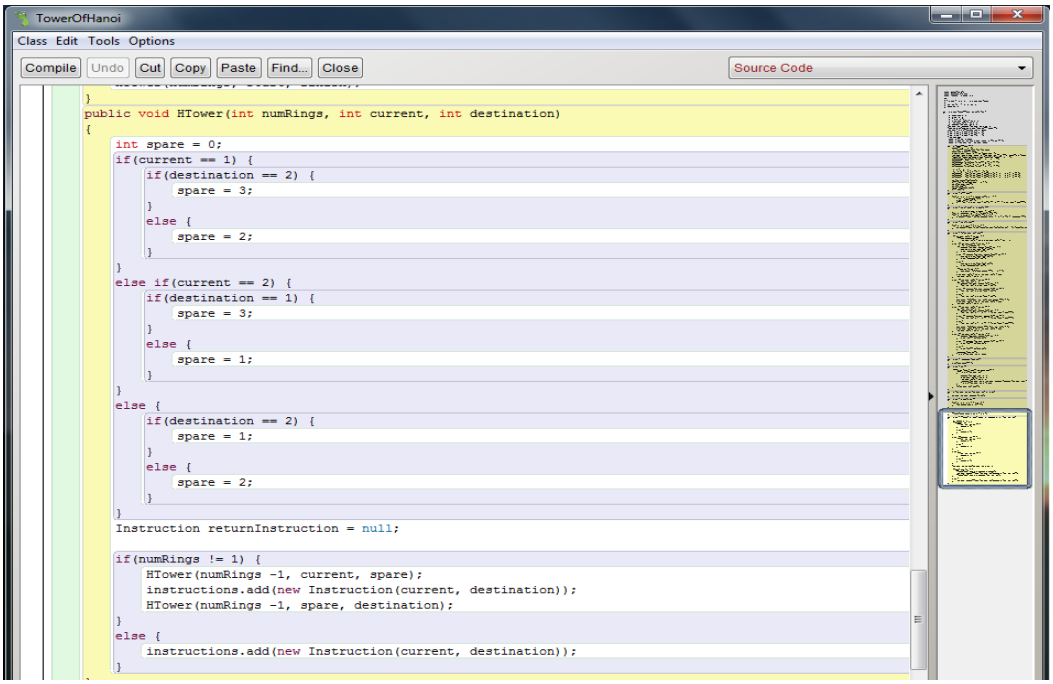


Figure 6 – Source Code for the Towers of Hanoi Game

4. CONCLUSION

Using this approach in Algorithm Design course, we got a positive result from students, using applications like games built by students under the supervision of the teacher.

The impact of using interactive development environment Greenfoot was a good one considering the possibilities of presenting graphical programming concepts with high difficulty. I will try to develop further applications in Java (programming language used for Greefoot) to give students more attractive representations and especially interactive computer programming problems.

5. REFERENCES

[1] Michael Kölling, The Greenfoot Programming Environment. Trans. Comput. Educ. 10, 4, Article 14 (November 2010), 21 pages. DOI=10.1145/1868358.1868361 http://doi.acm.org/10.1145/1868358.1868361

[2] Michael Kölling, *Introduction to Programming with Greenfoot, Object-Oriented Programming in Java with Games and Simulations*, Prentice Hall, 2010

[3] Michael Jonas, Teaching introductory progamming using multiplayer board game strategies in Greenfoot. J. Comput. Sci. Coll. 28, 6 (June 2013), 19-25.

[4] Poul Henriksen, Michael Kölling, 2004. greenfoot: combining object visualisation with interaction. In Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications (OOPSLA '04). ACM, New York, NY, USA, 73-82. DOI=10.1145/1028664.1028701 http://doi.acm.org/10.1145/1028664.1028701

[5] Raquel Hijón-Neira, Ángel Velázquez-Iturbide, Celeste Pizarro-Romero, Luís Carriço
Improving Students Learning Programming Skills with ProGames - Programming through Games system, 2013

[6] http://www.greenfoot.org

[7] http://www.runceanu.ro/adrian