Programming Lab: Towers of Hanoi

Data Structures and Advanced Programming Due: 2/28/20

Objectives

- To write a program using stacks.
- To review recursion.
- To explore a problem that is interesting in its own right.

Introduction

In this lab, we'll be writing a Java program to simulate the famous Towers of Hanoi puzzle. The game was originally invented by Édouard Lucas in 1883, but Lucas's puzzle has since given rise to many variants. Lucas's game begins with n disks arranged on a post from largest (on the bottom) to smallest (on the top). There are two empty posts to the right of the post that contains the stack of disks. The goal of the game is to move the entire stack of disks to the right most post in as few moves as possible, subject to the following movement restrictions:

1. Only one disk may can be moved at a time.
2. Only the top disk of any given pile can be removed from the pile. It must be placed on the top of another (possibly empty) pile.
3. Larger disks may never be placed on top of smaller ones.

There is an elegant recursive algorithm for solving this problem. Specifically, to solve the Towers of Hanoi problem with n disks, you must first move the top $n-1$ disks to the middle pile. Then, you move the largest disk to the last pile. Finally, you move the middle pile onto the largest disk. Using this strategy, the 3 tower version of the Tower of Hanoi can be solved in $2^n - 1$ moves.

For more information on the Towers of Hanoi problem (including details about how to solve it with binary operations and for an interesting connection to the Sierpinski gasket) , have a look at the Wikipedia article on it: `https://en.wikipedia.org/wiki/Tower_of_Hanoi`.

Program Requirements

For your final project of the term, you must develop a graphical Towers of Hanoi game using the objectdraw library. You may work with a partner on this project and will be graded based on the following criteria:

- At least 3 disks and 3 towers appear on the screen in the correct orientation when the game launches (10 points).

- Disks can be smoothly dragged around the screen and moved between posts (10 points).
- Only the top most disk can be removed from a post (10 points).
- Disks automatically move back to their prior position if the user attempts and invalid move (10 points).
- The game correctly detects when the disks are in a winning configuration (8 points).
- The game keeps track of the number of moves the player takes (8 points).
- The game displays custom victory messages based on how close to optimal the player's solution was (8 points).
- The player can choose to play with between 3 and 7 disks (6 points).
- The game allows the user to undo their previous move (6 points).
- Each disk is a different color (6 points).
- The game includes a reset button that clears the user's score and restores the towers to their original state (3 points).
- The game supports unlimited undos (3 points).
- Disks snap into place on a post and are always neatly aligned (3 points).
- The program supports saving a game that is in progress and writes out a saved file (2 points).
- A saved game can be passed in as a command line argument and reloaded (2 points).
- Victory message includes an animation of some sort (2 points).
- Game includes a flag that suppresses its GUI and causes it to run, with basic functionality, in CLI mode (1 point).
- Game includes a training mode that alerts players when they make a suboptimal move (1 point).
- Game includes an auto-play mode that demonstrates the optimal solution (1 point).

Up to 10 additional points will be awarded for extra features not listed in the requirements above (provided that these extensions are documented in the code). The more difficult the code extensions, the more points they will be worth. Some possible extensions include supporting additional posts or adding modes to play the "adjacent pegs" or bicolor versions of the game.

Thought Questions

Please include your answers to the following thought questions in a comment at the top of the file that contains your `WindowController`.

1. Prove by induction that the recursive solution to the 3 post Tower of Hanoi problem requires $2^n - 1$ moves.
2. This lab asked you to develop a program that supported both a graphical and a command line interface. Were you able to reuse any code for these two versions of the game? If so, what about your design allowed you to do so? If not, what about your design necessitated different code bases? (If you didn't get around to making a CLI, answer this question in terms of what you would have done.)
3. What is the ABACABA pattern, and how does it relate to the Towers of Hanoi?

Submission

You should submit your code by checking it into your GitHub repository in a folder called **TowersOfHanoi**. Your source code should reside in an `src` folder within the project folder. The `src` folder should contain all of the files and resources necessary to compile and run your game using the following bash commands while within the TowersOfHanoi directory:

```
javac -cp ./src:./src/objectdrawV1.1.2.jar ./src/*.java
java -cp ./src/:./src/objectdrawV1.1.2.jar TowersOfHanoi
rm ./src/*.class
```

Note that you need to include the objectdraw library inside your src folder to make the above commands work correctly. As ever, you should detach the following page from this handout and affix it to a printed copy of your code, which you must turn in at the start of class on the project due date.

Data Structures, Towers of Hanoi

Name: _____

Honor Code: _____

Criterion	Points
Correct starting layout	___ / 10
Drag and drop disks	___ / 10
Only top disks move	___ / 10
Enforces legality of moves	___ / 10
Detects win condition	___ / 8
Counts moves	___ / 8
Custom victory messages	___ / 8
Supports 3-7 disks	___ / 6
Can undo a single move	___ / 6
Different color disks	___ / 6
Reset functionality	___ / 3
Unlimited undos	___ / 3
Alignment on post	___ / 3
Save functionality	___ / 2
Load functionality	___ / 2
Animation at game end	___ / 2
Command line interface	___ / 1
Training mode	___ / 1
Auto-play mode	___ / 1
Thought Questions	___ / 15
Total:	**___ / 115**