Tower of Hanoi

Michael Martin Pagaran
Bachelor of Science in Information Technology
Zone 7, Bulua
Cagayan de Oro City
09275600181
grimfiasco@gmail.com

Ella Rose Cabalatungan
Bachelor of Science in Information Technology
Kisanlu Subdivision, Iponan,
Cagayan de Oro City
09125876991
ellarosecabalatungan@gmail.com

Rogelio Ibacarra Jr.
Bachelor of Science in Computer Science
J.V. Seriña Street Carmen,
Cagayan de Oro City
09365015517
ibacarrajr.rogelio@gmail.com

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Overview

Tower of Hanoi is a 1 player mathematical puzzle game with three rods and n disks. The objective of the game is to move the entire stack to another rod, obeying the following simple rules: One, only one disk can be moved at a time. Two, each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack. Finally, no disk may be placed on top of a smaller disk.

The game provided enough challenge for programmers not only to implement the appropriate data structures and algorithms which were discussed throughout the first and second semester but also, to offer a fun game experience to its players.

1.2 Objectives

The main objective of the development of the project is to build a game by using the following data structures and algorithms: arrays, stacks, recursion, and file systems that exhibits acquired programming skills of the programmers. The following are the rest of the objectives for the development of the game.

Objectives of the project:

- The program will be able to read the inputs of the player and react accordingly during the duration the game is played.
- The program will be able to provide the player with intuitive and comprehensive game interface.

1.3 Scope and Limitation

The project was focused on the development of a Command-line interface game program by using Java.

The scope of this project includes:

- A brief tutorial on how to play the game. Subsequently, the tutorial accepts any number of disk that the user wishes to have and shows the best way to solve the problem.
- In the actual gameplay, the user can still choose the difficulty by entering the desired number of disks.
- Each move will be recorded as score. The lesser the score, the better.
- After beating the game, the user has the option to retry the whole process to improve his/her score or to enter his/her name and exit to save the current score as final.
- The scores will be stored in a text file along with their names and number of disks used.

The limitation of this project includes:

- The game does not have an 'undo' option while playing.
- The program used in-built stacks.
- JavaFX was not used in this program.

1.4 Functionalities

The functionalities of the game include:

- Game Menu
    - ➢ Tutorial
    - ➢ Start game
    - ➢ Scores
    - ➢ Game credits
    - ➢ Exit
- Three stacks as rods
- User defined N number of disks
- Manipulate the movement of disks
- Minimum moves made will serve as scores
- Retry option
- Save game

2. PROGRAM DESIGN AND IMPLEMENTATION

2.1 Pseudocode

ToH.java.

```
switch(gameMode){
        case 1: SampleHanoi.main(new String[0]);
        break;
        case 2: ManualHanoi.main(new String[0]);
        break;
        case 3: ScoreHanoi.main(new String[0]);
        break;
        case 4: HanoiCred.main(new String[0]);
        break;
        case 5: System.out.println("Closed");
        break;
```

```
            default:
            break;
        }
```

SampleHanoi.java

```
    public static void toh(int disk) {
        for(int d = disk; d>0; d--){
            rod[1].push(d);
        }
        display();
        move(disk, 1, 2, 3);
    }
    public static void move(int n, int a, int b, int c) {
        if (n > 0) {
            move(n - 1, a, c, b);
            int d = rod[a].pop();
            rod[c].push(d);
            display();
            move(n - 1, b, a, c);
        }
    }
```

ManualHanoi.java

```
    while(gameWon() == false){
            S.O.P(Ask input);
            int rodFrom = input.nextInt();
            space(1);

     while(rodFrom>3||rodFrom<0){
            space(50);
            display();
            S.O.P(Display error. Ask input);
            rodFrom = input.nextInt();
            space(1);
     }
      while(rod[rodFrom].isEmpty()){
            space(50);
            display();
            S.O.P(Display error. Ask input);
            rodFrom = input.nextInt();
            space(1);
     }
      int activeDisk = rod[rodFrom].peek();
      space(50);
      display();
      space(1);
      S.O.P(Ask input);
      int rodTo = input.nextInt();
      space(1);
      while(rodTo>3||rodTo<0){
            space(50);
            display();
            S.O.P(Display error. Ask input);
            rodTo = input.nextInt();
            space(1);
     }
      try{
            while(activeDisk > rod[rodTo].peek()){
               space(50);
               display();
               S.O.P(Display error. Ask input);
               rodTo = input.nextInt();
               space(1);
            }
     }catch(Exception e){
     }
      rod[rodFrom].pop();
      rod[rodTo].push(activeDisk);
      space(50);
      display();
      movesMade++;
      space(1);
      System.out.println("Current score: "+movesMade);
    }
```

ScoreHanoi.java

```
    try{
      File file = new File("ToH_Scores.txt");
      Scanner input = new Scanner(file);
              while(input.hasNextLine()){
                 String line = input.nextLine();
                 System.out.println(line);
              }
    catch(Exception e){
    }
      Scanner s = new Scanner(System.in);
      s.nextLine();
      space(50);
```

Display()

```
    for (int i = N - 1; i >= 0; i--) {
      String disk1 = " ", disk2 = " ", disk3 = " ";
            try {
               if(rod[num].get(i)==0){
                  disk1 = " ";
               }
              else{
                  disk1 = String.valueOf(rod[1].get(i));
               }
            } catch (Exception e) {
            }
               S.O.P(disk1+disk2+disk3);
    }
```

2.2 Data Structures and Algorithms Discussion with Code Snippets

ToH.java

ToH.java was used as the main java file. The 4 game modes are placed inside the 'switch case' statement to allow the player to have the freedom to select which option he/she prefers.

SampleHanoi.java

This part of the code was used in the tutorial part of the game. This implements the recursive method of solving the problem. It uses the in-built Stack. Three stacks were made to serve as the rods of the game. Identifiers were used to easily access the stacks.

ManualHanoi.java

This was used for the actual gameplay, where the player is given the freedom to move the disk to any possible rods. It is placed inside a 'while' loop which executes repeatedly until the game is won.

while(rodFrom>3||rodFrom<0)

while(rod[rodFrom].isEmpty()

while(activeDisk > rod[rodTo].peek()

There were no recursive methods that were used in this game mode. To prevent the user from making incorrect inputs, 'while' loops will be responsible for constantly checking the user's input.

ScoreHanoi.java

This will read all the saved scores from a text file and display it in the command prompt. Instead from retrieving data from "System.in", the Scanner will be retrieving data from the text file directly.

```
File file = new File("ToH_Scores.txt");
```

Display()
This method will be used several times during the game. This prints the contents of every rod, line by line. It will first declare three (3) String classes and by default, it will hold only 'blank' (" ").

```
String disk1 = " ", disk2 = " ", disk3 = " ";
```

These String classes will store the numbers of a certain index on each three rod. If a rod has a number greater than zero in a specific index, the number will be converted into a String data type and will be stored in the class. However, if the number in a specific index from a rod is zero, the String will assume its default content which is 'blank' (" ").

```
disk1 = String.valueOf(rod[1].get(i));
if(rod[num].get(i)==0){
        disk1 = " ";
```

The three String classes will be printed out and will move to the other index for the next line.

3. CONCLUSION

In conclusion, the development of the Tower of Hanoi program requires programming knowledge in the use of Java and skills in the application of various data structures and algorithms, but equally important is the creativity in designing the features and functions of the game considering the player's experience as well. It is important to pay equal attention to the development of the program codes and stick with the project's objectives.

Creating a mathematical puzzle game inspires both logical and creative thinking and entails continuous learning.

4. REFERENCES

Running a java program from another java program: https://bit.ly/2O0xRrr. Accessed: 2019-2-21

Tower of Hanoi recursion game algorithm explained: https://bit.ly/2C9ogcU. Accessed: 2019-2-7

Java Programming Tutorial 80 - Writing to Files: https://bit.ly/2Uvm2vt. Accessed: 2019-2-20

Java Programming Tutorial 81 - Reading from Files: https://bit.ly/2VRMqAd. 2019-2-20

5. APPENDICES

5.1 Project Proposal

CC12/CC13 Project Proposal

I. Title of the Program
Tower of Hanoi

II. Brief Description of the Program
A mathematical puzzle program having three rods and five numbers as objects. The objective of the puzzle is to move the entire stack to another rod, obeying the following simple rules:
1. Only one disk can be moved at a time.
2. Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.
3. No disk may be placed on top of a smaller disk.

Game mechanics:
· The program provides a brief tutorial on how to play the game. Later, the tutorial will accept any number of disk that the user wishes to have and will show the best way to solve the problem.
· In the actual gameplay, the user can still choose the difficulty by typing the desired number of disks.
· The game will not have an 'undo' option while playing. Each move will be recorded as score. The lesser the score, the better.

- After beating the game, the user will have the option to retry the whole process to improve his/her score or enter his/her name and exit to save the current score as final.
- The scores will be stored in a text file along with their names and number of disks used.

III. Features and Functionalities
- Game Menu
- Minimum move made will serve as scores
- Retry option
- User defined N of disks
- Tutorial/auto-solve
- 3 stacks as rods
- N of numbers represent as disks
- Scores will be saved in a text file.

IV. Possible data structures and algorithm to be used
- Arrays
- Stacks
- Recursion
- File systems

5.2 Photo Documentation/ Program Screenshots