'The Towers of Hanoi'
User Manual
Produced by Lewis Haley

Initial Set-up

In order to run this program it is first necessary to modify the code slightly to allow it to run on your user name login.

Please follow these instructions before you try to run the program.

1) Open the source code file ("towers.py")
 in a text editor.
2) Go to line 46.

3) Change the '/nccaba1/i7989478' to the directory where the folder is saved.

i.e. cgstaff, nccaba1, bapublic, etc, then asarafop, avanner, i7990443, etc.

4) Save the source code file ("ctrl+s").

This needs to be done because the program imports a font from a directory within the folder provided and the file path must be changed for it to load in a different user's login directory.

Starting Program

Once this has been done, open terminal. Navigate to "The_Towers_Of_Hanoi" folder within terminal. Then type in "clear", followed by "python towers.py". This will run the program.

Keyboard Commands/Instructions

The program will give you step by step instructions as you progress, however here is a general guide to the keyboard controls.

- If an image window appears, you can press "q" to continue with the game (alternatively, click on the upper-right hand cross to close the window as you would normally).
- To move to the next line of text, press return.
- When asked to enter something, enter *exactly* what has been asked of you.
- If you enter something that doesn't conform to what has been asked, you will be re-prompted to input until your input conforms.

To stop and exit the program at any time, simply press "ctrl+c" simultaneously on the keyboard and you will exit back to the regular terminal.

Objective

The objective of this game is to move a set of disks from one pole to another, whilst at all times making sure that a larger disk is never placed on a smaller disk. A more precise description is given to you upon starting the program.

'The Towers of Hanoi'
Documentation
Produced by Lewis Haley

Accreditation

This program is based on a ‘pygame’ script I discovered online at:
`http://tuxradar.com/content/code-project-tower-hanoi-python`
‘Pygame’ is a programming language based on python but which can deal with interactive and updatable images. In other words, and as the name suggests, it can be used to write code for interactive, real-time games, or other programs that require direct user-interface interaction.

Main algorithms

The main algorithms used in this program are if statements which determine the current locations of the disks on the poles by analysing the ‘stacks’ list. They work in conjunction with a “for” loop which cycles between the 3 poles. Essentially, it works by saying, “Are the disks set up like this? If yes then draw them here.”

Also used a lot are while loops that are used as a check to make sure the user entered the required inputs. The only way to break out of the loop is to put in the right input.

The recursion function ‘autoHanoi()’ works by calling itself inside its own body, and each time it subtracts 1 from the current number of disks (on the pole) until there is just 1 left, at which time it performs list operations on ‘stacks’ to move a disk from 1 pole to the next. Each time a move is made the current image is shown to the user and then disappears, acting like a pseudo-animation.

The PIL aspect of this project uses ‘for’ loops to create gradient effects. These work by drawing the full shape first, then drawing the same shape with a slightly different colour and slightly not as far as the first shape. This is controlled by the ‘i’ in the for loops.

Flow of Control

As mentioned above, there are lots of loops and ‘if’ statements used in this program. There are also numerous defined functions that are called throughout the program. The ‘autoHanoi()’ function uses recursion, where the function calls itself. There are no non-structured commands used, i.e. breaks or continues.

Step-by-Step Program Walkthrough

Lines 1-8:

- Tells program to load python keyword library.
- Imports Image, ImageDraw and ImageFont libraries.
- Creates a new image, (red, green, blue, alpha), 500 by 700 pixels in size, called ‘img’.
- Creates a draw object on ‘img’, called ‘draw’. This allows PIL to use draw commands to edit the image.

Lines 10-38:

- Defines procedure called ‘background’, requiring no arguments to be passed in.
- For loops are used to create a gradient effect when drawing the sky, sun and ground.
- Each time the for loop cycles, the shape that is drawn is slightly smaller, and the fill colour is slightly different, creating a gradient effect.
- More for loops are used to create similar gradient effects to draw the three poles. The three poles are identical, so the draw commands for each can be put inside the same loop.
- Yet another for loop is used to draw ‘shadows’ from the poles. The start and end coordinates are increased asymmetrically using multiplication and so making the lines drawn non-parallel.
- This set of commands are grouped under the ‘background’ procedure so that each time a new move is completed the background can be redrawn, which ‘covers up’ the previous image and layout of the disks. It is always the same, and so does not require arguments. This is a prime example of using a procedure to eliminate use of repeated code. Nothing is returned to the caller when this is called.

Lines 40-53:

- Another procedure requiring no arguments is set up. Unlike the ‘background’ procedure, this one is not called repeatedly; it is only used to create a pleasant aesthetic for the ‘front screen’ of the game.
- However, the code it contains was put into a procedure so that in can be seen subsequently to the ‘background’ procedure.
- Firstly a variable is set up called ‘FONT’, which contains the file path to the .ttf font file. (this file path has to be changed by the user prior to running the program, as explained in the User Manual)
- ‘font1’ variable is set up, using ImageFont library. This sets up the font and size.
- Text is then drawn on the draw object – shadows are drawn first so they appear underneath the actual text.
- ‘font2’ variable is set up. Same font is used but a smaller size used for the subtitle on the image.

Lines 55-100:

- More for loops to draw gradients on the disks to be moved around in the game.
- ‘disk1’ is largest disk, ‘disk5’ the smallest, each disk gets arithmetically smaller.
- Each disk takes 2 arguments – offset and height. They control the x and y coordinates of where the disk will be drawn respectively.

Everything prior to this point has merely been creating and defining the images that will appear and move about in the game. From line 106 onwards, the program is adapted from the aforementioned website and controls the movements and other events in the game.

Lines 113-153:

- ‘polesSetup’ procedure defined, taking 1 argument – ‘numdisks’, i.e. number of disks being used in the game (this is a user inputted value initialised later on).
- Variable ‘offset’ initialised and set at 50 (pixels). This is used to dictate the x coordinate of where the disks will be drawn.
- Sequence of ‘if’ and ‘elif’ statements. Which statements are carried depend on ‘numdisks’ argument. When ‘numdisks’ is 1, only ‘disk1’ procedure is called; when ‘numdisks’ is 2, ‘disk1’ and ‘disk2’ are called, etc.
- Each ‘if’ statement sets up a list called ‘stacks’, which contains information on the location of each disk. Essentially, ‘stacks’ is a list that contains 3 more lists, each dedicated to a different pole.
- Also set up is a list called ‘WIN’. The user must make it so that ‘stacks’ is identical to ‘WIN’, and dictates when the game is over and that the user has won.
- For each ‘if’, both ‘stacks’ and ‘WIN’ are returned to the caller.

Lines 155-291:

- As stated in the code, these procedures greatly reduce the number of lines of code. There is nothing complex to them, and they are incredibly repetitive.
- The purpose of the procedures is to determine where there are disks in the current setup of the poles and to subsequently draw those disks.
- They achieve this by using an ‘if’ statement for every combination of disks possible for each value of ‘numdisks’.
- Each take arguments ‘x’ – which acts as the pole number (0, 1, 2), and is brought forward from the ‘for loops’ in the next section; and offset, which is the distance needed to put the disks onto the adjacent pole.

Lines 293-333:

- Utilises aforementioned procedures to draw the disks onto the correct poles as per the current set up of the list ‘stacks’.
- There is an ‘if’ statement for each possible value of ‘numdisks’.
- Subsequently, all the required ‘drawDisk*()’ procedures are called for that ‘if’ statement.
- At the end of each loop, the offset variable has 200 added to its value. This means that if any disks are drawn in the next loop they are drawn on the next poles to the left.

Lines 335-342:

- A simple procedure called when it is time to show the user an image.
- First, the image window is shown, then a time function is called that pauses the program for 2 seconds, then a Linux command is run that closes all ‘eye of gnome’ windows (open image windows).
- Closing the image windows automatically mean the user doesn’t have to do it themselves which keeps the program running smoothly.

Lines 345-371:

- Uses recursion to solve the Towers of Hanoi puzzle - edited from author Ari Sarafopoulis' "towersOfHanoi.py" in the '09_RecursionR.zip' on myBU.
- The base action for this occurs when there is only 1 disk left on the peg, and removes the last disk from the peg and puts it onto another.
- Then background is redrawn and the disks drawn before image being shown. The ‘show()’ function makes each image appear then disappear as the moves are made by the program. This makes something like an animation sequence appear, although technically it is not an animation sequence.

Lines 374-456 and 592-616:

- Now the program really starts, the user is talked through the premise of the game, etc. Game sleeps for 1 second between each line of text.
- 2 variables are set up, 1 for a while loop and one for a nice little Easter Egg. ☺
- While progRunning is 1 (true), ask user if they want to play the game or watch it being played. If the user doesn’t put enter either ‘PLAY’ or ‘WATCH’, the ‘idiot’ counter increases by 1 and they user is re prompted to enter ‘PLAY’ or ‘WATCH’. If the user refuses to enter ‘PLAY’ or ‘WATCH’ 5 times, they are insulted for be an imbecile.
- Anyway... 2 ‘if’ statements depending on user input: user plays the game or user watches game be completed.
- In either case, another variable is set up called ‘comply’, while loop is set up for comply = 0 and the user is asked how many disks they want to use in the program. If the user enters a number not between 1 and 5, they are re-prompted for the input. When input complies, comply = 1.
- PolesSetup() is run with the user input of ‘numdisks’ as an argument, returning the lists ‘stacks’ and ‘WIN’ to variables of the same name.

Lines 458-590:

- User is given instructions on how to play the game. Move counter is started.
- While loop: while the ‘stack’ list is not the same as the ‘WIN’ list.
- User prompted for a move input. Then ‘if’ statements run for each viable move the user can make: pole 1 to pole 2 or 3, 2 to 1 or 3, 3 to 1 or 2. If the user enters an invalid input, they are given a message and re-prompted for an input.
- Likewise, ‘if’ statement runs but if the stack they are moving from is empty message given and re-prompted.
- Otherwise, item popped from list into variable pop, then pop is inserted into ‘stacks’ in the desired position. After each valid move, move count is increased by 1.
- Once user completes game, ‘stacks’ will equal ‘WIN’, so the while loop breaks out and winning message is given.
- User asked if they want to play again (given option to play or watch if yes). If yes, progRunning while loop continues,; if no, while loops ends and program goes to line 631 for goodbye message.

Line 619-628:

- Runs autoHanoi() program as described earlier.
- A few lines of text to user, including how many moves it took.
- User asked to if they want to play again, as per above.

PROBLEM

For some unknown reason, when the program is running through the game automatically using the recursive function the images shown do not correspond to the ‘stacks’ list. The same function ‘drawDisks()’ that is called to draw the disks onto the background that appears in the user-run game part of the program is called in the recursive section, and in the game section the function works fine. I do not know why the disks are not drawn correctly, because the list ‘stacks’ is altered the right why by the program, as can be seen when the program is running. The ‘stacks’ list is printed on screen – you can see that it is changing in the correct way and yet the ‘drawDisks()’ function is not producing the correct images. ‘Stacks’ is not edited when it enters the function. I can only assume it has something to do with the recursive element of the ‘autoHanoi()’ function.