

## 2 Recursive Animations and The Towers of Hanoi

Animations can be a useful tool to visualise and understand solutions to mathematical problems that can be represented graphically. In this project you will begin by creating animations of various recursive problems using the python graphical library `pygame`, in part by integrating its use with other tools that you have learnt during the course. You will then tackle the Towers of Hanoi problem and create `pygame` animations of solutions to the problem. As a further challenge you can also choose to develop one of the extensions outlined in parts 6, 7, 8, 9 below.

**Technical Note.** To run the video animations in this project you will need to work locally, i.e. using your own distribution of python or that of the University. (You cannot do this via the Noteable server.)

**Remark.** The Python, Jupyter Notebook, and pdf files mentioned below are available in the GitHub repository for this project at <https://github.com/cmh42/toh>.

1. **(core)** Animating fractal constructions gives us direct insight into how the function generating the fractal operates. You are given a `pygame` animation function `draw_sierpinski` (in the `animation_examples.ipynb` file) that draws the Sierpinski triangle. Using this function the user is able to choose the *depth* of the triangle and to stop and start the animation. You should develop this function so that the user is also able to change the speed of the animation. You should also add colours to the triangle drawing.
2. **(core)** Fractals can simulate shapes found in the natural world. One simple example of this is the construction of a recursively defined tree. The tree of depth 1 is just a trunk with three straight branches. Then given the tree  $T$  of depth  $n$  the tree of depth  $n + 1$  is the tree  $T$  where every branch has been replaced by a tree of depth 1. In Figure 1 (from the left) is such a tree of depth 1, 2, 3 and 4 and then a tree of depth 9 partially and completely drawn.

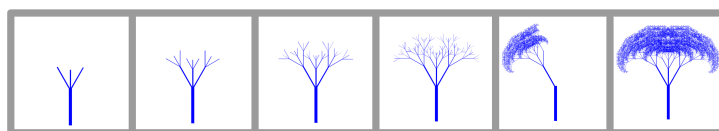


Figure 1: Recursive Tree

You should develop a `pygame` animation function that constructs a similar recursive tree. The user should be able to choose the depth of the tree and should be able to start and stop the animation and control its speed. The tree should be coloured with the trunk and branches being brown and the leaves (i.e. the last level of branches) being green.

3. **(core)** Similarly to results in Homework 3 we can show that there are interesting Julia sets with parameter  $j_p = 0.7885e^{ai}$  where  $a$  is a small non-negative real number. In fact if we allow  $a$  to vary in repeated cycles over  $[0, 2\pi]$  we are able to create a film like animation of Julia sets with this form of parameter. The images in Figure 2 are snapshots of such Julia sets.

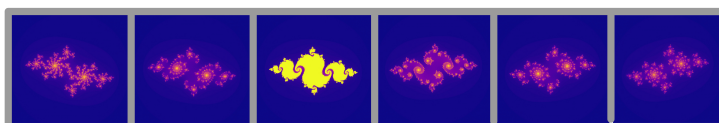


Figure 2: Julia sets

Your task is to develop a function (using the matplotlib.pyplot tools you developed for Homework 3) to generate a number (for example 200) of image files of the Julia sets with parameter  $j_p = 0.7885e^{ai}$  where the numbers  $a$  are chosen at equally spaced intervals in  $[0, 2\pi]$ . You should then develop a pygame function to display these files as a film like animation. The user should have speed and stop/start control over the animation.

4. **(core)** In the *Towers of Hanoi* problem we have three poles and  $n$  discs that fit onto the poles. The discs differ in size and are initially arranged in a stack on one of the poles, in order from the largest disc (disc  $n$ ) at the bottom to the smallest disc (disc 1) at the top. The problem is to move the stack of discs from one pole to another pole while obeying the following rules.

- Move only one disc at a time.
- Never place a disc on one smaller than it.

This problem can be solved by issuing a sequence of instructions for moving the discs in the appropriate way. We assume that the poles are arranged in a row and that each instruction to move a disc specifies its number and whether to move it left or right. We allow *wrap around*: if a disc is on the left pole an instruction to move left means to wrap around to the right pole, whereas if a disc is on the right pole an instruction to move right means to wrap around to the left pole. A solution to the Towers of Hanoi problem for 3 discs is displayed in Figure 3.

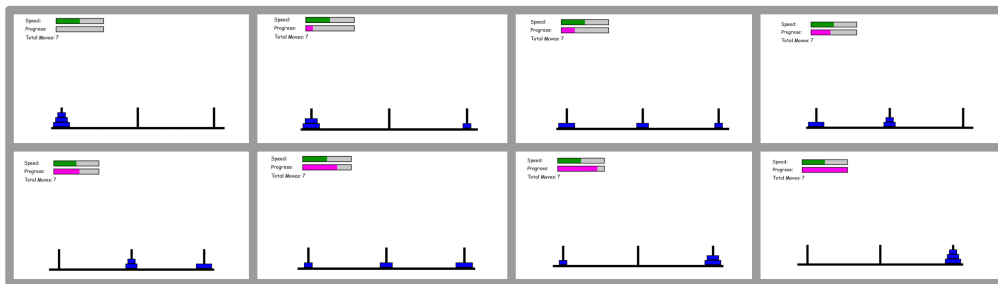


Figure 3: Towers of Hanoi with 3 discs

We can solve the problem by recursion. First we move the stack comprising the top  $n - 1$  discs to an empty pole, then we move the largest disc (i.e. disc  $n$ , the bottom disc of the starting stack) to the other empty pole. Finally we move the stack of  $n - 1$  discs onto (the pole with) the largest disc. Figure 4 illustrates this recursive approach.

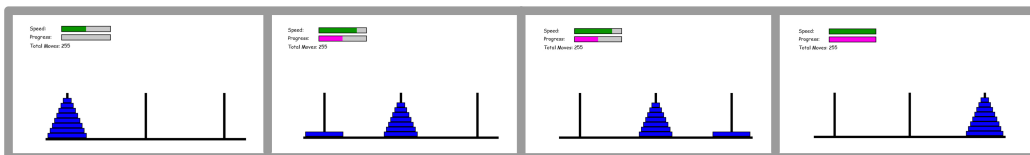


Figure 4: Solving the Towers of Hanoi recursively

Your task here is to write a function that implements this idea to compute/generate, given input  $n$ , the sequence of moves required to solve the Towers of Hanoi problem for  $n$  discs<sup>1</sup>. You should then develop a function that displays a printout of the sequence of

<sup>1</sup>To get a better idea of how to do this see the description of the Towers of Hanoi problem in the file `towers_of_hanoi.pdf` (which is an excerpt of Section 2.3 of [1]). **Suggestion.** Instead of using a boolean

configurations of a solution to the problem for  $n$  discs. (To be used for e.g. at most 8 discs. Note that this is a warm up for the graphical animation.)

5. **(core)** Using the function you developed in part 4 to compute/generate the sequence of moves for the Towers of Hanoi problem you should now develop a pygame function that gives an animation of the problem. The user should be able to control the initial number of discs, stopping/starting, and the speed of the animation. The initial number of discs may be any integer in the interval  $[1, 16]$ . (Discs should be moved directly from one pole to another, i.e. without showing any intermediate motion.)
6. **(extension)** Work out the definition for functions required to solve the two following variants of the Towers of Hanoi problem. (1) The problem where wrap around is not permitted: you can only move discs between adjacent poles. (2) The problem where, given even  $n$ , there are  $n$  discs numbered according to size (as before), but initially the discs with odd number are in a stack on the left pole whereas the discs with even number are in a stack on the right pole. The problem is to move the discs with odd number to the right pole and the discs with even number to the left pole (still satisfying the two rules given above, with wrap around allowed). Develop your pygame animation function to show solutions to these two problems.
7. **(extension)** Develop a version of your Towers of Hanoi pygame animation function from part 5 that moves the discs smoothly between the poles (i.e. now showing the intermediate motion). Once you have done this add a menu (in pygame) so that the user can choose to either run the original animation or the smooth animation.
8. **(extension)** Develop a version of the Towers of Hanoi pygame animation where a player can play the problem as a game, by moving the discs from pole to pole.
9. **(extension)** Create an animation in which the user is able to repeatedly focus in and out on any part of the Mandelbrot set. (Remember that in the tutorial of Week 4 we developed functions to display the Mandelbrot set.) To do this you will want to integrate the use of a function that generates image files (using matplotlib.pyplot) with a pygame function to control the animation.

**Note on the extensions.** In this project (i.e. project 2) you may choose to develop one extension. (Doing more will not achieve more marks: the point is to concentrate on the quality of the ideas and the design of your code in both the core sections and the extension that you choose.)

## References

- [1] R. Sedgewick, K. Wayne, and R. Dondero. *Introduction to Programming in Python: An Interdisciplinary Approach*. Addison-Wesley Professional, 2015.

---

variable to specify left or right movement you may find that a more flexible approach is to number the poles and specify the source pole, auxiliary pole, and destination pole. (Note: when reading [1] do not be confused by the fact that the authors use their own specialist libraries and functions, such as `stdio` and `stdio.writeln`.)