Included in README:

Author: Cole Moore

Edited: 9/4/20

To run the code, open up a terminal in whatever editor you are using (I built this project in Atom, so it would probably work best with that).

Then, navigate to the ModelingPyraminx folder.

Input the command: "python main.py" into the terminal.

It should print out the moveset, the initial state of the pyraminx, and ask if you would like to randomize the pyraminx.

If you enter "y" or "Y", the program will ask you how many moves it should randomize it, and you will input a positive number.

The randomizer will then randomly apply one of the 12 moves from the moveset to the sub-pyramids until it has made the number of moves indicated by the user.

It will then make a model of the randomized pyraminx.

If you entered "n" or "N", then the program will end.

The data structure of my pyraminx is made up of combined lists in Python. The GUI is plain text; ‘r’ stands for red, ‘b’ stands for blue, ‘y’ stands for yellow, and ‘g’ stands for green:

r b y

r r r b b b y y y

r r r r r b b b b b y y y y y

r r r r r r r b b b b b b b y y y y y y y

g g g g g g g

g g g g g

g g g

g

This is the initial state of the model pyraminx. In my program, every move is made relative to the front-facing side of the pyraminx, the side that is initially blue. This is documented in my code, but in summary:

* There are 12 moves total
* The randomizer moves sub-pyramids clockwise with each turn
* When randomly choosing a number between 1 and 12, the order is the same as indicated in the moveset printed initially:
  + T stands for the top levels of the pyraminx (uppermost tip)
  + L stands for the left levels of the pyraminx (bottom left tip)
  + R stands for the right levels of the pyraminx (bottom right tip)
  + B stands for the back levels of the pyraminx (bottom center tip in the back)
  + Moves will affect sub-pyramids, starting from the chosen tip
  + 1, 2, and 3 represent how many levels of a sub-pyramid are going to be rotated in one move
  + Any moves that the randomizer makes are clockwise
    - When I program the solver, all user moves will be counterclockwise

For example, if the randomized number was 3, the randomizer would utilize the move “T3”, which moves the top three (T3) levels of the front-facing side. The program would rotate the places of the uppermost three levels facing the “front” side (blue) clockwise from the top tip down, like so:

b y r

b b b y y y r r r

b b b b b y y y y y r r r r r

r r r r r r r b b b b b b b y y y y y y y

g g g g g g g

g g g g g

g g g

g

To move the pyraminx as indicated, it is replacing elements in the list with the elements in the list that are taking its place.

I have the program separated into four files: display.py, which is responsible for the GUI, randomizer.py, which randomizes the pyraminx a select number of times based on user input, solve.py, which is how the user would manually move the pyraminx, but is implemented to only display the moveset currently for the assignment, and main.py, which ties all the other files together and would make the user input moves until the pyraminx is back in its initial state to solve it, but for this assignment, only initializes the pyraminx, displays the moveset, and randomizes the pyraminx a select number of times.

After asking if the user would like to randomize the pyraminx, the randomizer takes a user input for the number of moves it can make, then randomly chooses one of the 12 available moves that are coded in and included in the moveset. Using a for loop, the randomizer will continue to pick random moves until it has made the number of moves specified by the user, at which point a GUI of the randomized pyraminx will be displayed and the program will end.

My heuristic is:

(Total number of incorrect pieces) / (Number of faces affected per move [3]).

This is an admissible heuristic because the total number of moves to solve a randomized pyraminx will always be greater than or equal to this heuristic approximation. For example, if only one of the tips is moved, that would make the heuristic equal 1, which is equal to the number of moves needed to actually solve it. On the other hand, a large number of pieces are incorrect, the number of moves needed to solve the pyraminx would be more than the approximation from the admissible heuristic.

I did not have any experience with Python before this assignment, so I definitely learned a lot about how Python works and what its syntax is like. I learned how to utilize basic GUI in Python and how to define functions and loops, as well as importing from other files. I have also learned how to use user inputs to affect the GUI. Furthermore, I included a lot of unnecessary code to prepare for the solver, so I have learned that I need be more careful when reading the assignment.