Included in README:

Author: Cole Moore

Edited: 9/19/20

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.

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.

It will then attempt to solve the pyraminx using the heuristic I designed.

NOTE: While solving for k = 1 (k being the number of times the pyraminx was randomized), the program works; however, k >= 2 does not seem to work correctly. It is hard to tell, since the terminal usually just stops working, so I can't really test it further.

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 as such:
  + 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
  + Any moves the solver makes are 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.

In the solver, the program is using nodes to store the move and f value for each of the 12 possible moves, then finding the lowest f value and making that move on the pyraminx; this repeats until the pyraminx is solved.

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 program would use the heuristic described below to solve for the pyraminx, and main.py, which ties all the other files together and would run until the pyraminx is back in its initial state to solve it; initializes the pyraminx, displays the moveset, and randomizes the pyraminx a select number of times, then attempts to solve it. However, my computer kept crashing the program whenever I tried to go past just one move to randomize it, so I had to stop testing there.

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.

In a similar manner, the solver uses a while loop to check when the pyraminx is solved, and a for loop to use the heuristic for each move in the moveset, much like the randomizer. It continues doing this until the program is solved, at which point the program will end.

My heuristic is:

(Total number of incorrect pieces) / (Number of possible moves [12]).

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, simply due to how many pieces there actually are in a 4-level pyraminx; if the maximum number of pieces were incorrect, then it would take more moves to solve than the estimated heuristic.

I learned how to implement A\* using Python to solve a 4-level pyraminx. I also learned how to make an admissible heuristic and how to implement it into my program. In addition, I learned that my laptop is not good enough to run k >= 2 randomization without crashing.