Spencer Cooperman

CS 463

9/10/2023

Representing the Megaminx Project

How to Run the Code

In the folder where this document is, is a python file called Representing_the_Megaminx.py. Either open up the file and run it in the terminal within the editor or instead just go in through a terminal and navigate to where the file was downloaded where you can run the command 'python .\Representing_the_Megaminx.py'. The code will then run showing the faces of the megaminx after 50 random rotations as well as showing how many pieces are out of place.

Data Structure

To create the data structure for the megaminx, I created a 12x10 array where each element in the array had a 1x2 array. The 12 in the 12x10 array is for the 12 faces and the 10 is for the 10 pieces that surround the core piece. This means that every row in the array corresponds to each face on the megaminx. Each element in the row has a 1x2 array which represents the location of the cubbie on the megaminx and the color of the cubbie.

The code for my data structure of the megaminx is shown below:

```
megaminx = [[[0, 0], ['white']], [[0, 1], ['white']], [[0, 2], ['white']], [[0, 3],
['white']], [[0, 4], ['white']], [[0, 5], ['white']], [[0, 6], ['white']], [[0, 7],
['white']], [[0, 8], ['white']], [[0, 9], ['white']]],

            [[1, 0], ['blue']], [[1, 1], ['blue']], [[1, 2], ['blue']], [[1, 3],
['blue']], [[1, 4], ['blue']], [[1, 5], ['blue']], [[1, 6], ['blue']], [[1, 7],
['blue']], [[1, 8], ['blue']], [[1, 9], ['blue']]],

            [[2, 0], ['red']], [[2, 1], ['red']], [[2, 2], ['red']], [[2, 3],
['red']], [[2, 4], ['red']], [[2, 5], ['red']], [[2, 6], ['red']], [[2, 7], ['red']],
[[2, 8], ['red']], [[2, 9], ['red']]],

            [[3, 0], ['green']], [[3, 1], ['green']], [[3, 2], ['green']], [[3, 3],
['green']], [[3, 4], ['green']], [[3, 5], ['green']], [[3, 6], ['green']], [[3, 7],
['green']], [[3, 8], ['green']], [[3, 9], ['green']]],

            [[4, 0], ['purple']], [[4, 1], ['purple']], [[4, 2], ['purple']], [[4,
3], ['purple']], [[4, 4], ['purple']], [[4, 5], ['purple']], [[4, 6], ['purple']],
[[4, 7], ['purple']], [[4, 8], ['purple']], [[4, 9], ['purple']]],

            [[5, 0], ['yellow']], [[5, 1], ['yellow']], [[5, 2], ['yellow']], [[5,
3], ['yellow']], [[5, 4], ['yellow']], [[5, 5], ['yellow']], [[5, 6], ['yellow']],
[[5, 7], ['yellow']], [[5, 8], ['yellow']], [[5, 9], ['yellow']]],
```

```
            [[[6, 0], ['light blue']], [[6, 1], ['light blue']], [[6, 2], ['light
blue']], [[6, 3], ['light blue']], [[6, 4], ['light blue']], [[6, 5], ['light
blue']], [[6, 6], ['light blue']], [[6, 7], ['light blue']], [[6, 8], ['light
blue']], [[6, 9], ['light blue']]],

            [[[7, 0], ['light yellow']], [[7, 1], ['light yellow']], [[7, 2], ['light
yellow']], [[7, 3], ['light yellow']], [[7, 4], ['light yellow']], [[7, 5], ['light
yellow']], [[7, 6], ['light yellow']], [[7, 7], ['light yellow']], [[7, 8], ['light
yellow']], [[7, 9], ['light yellow']]],

            [[[8, 0], ['pink']], [[8, 1], ['pink']], [[8, 2], ['pink']], [[8, 3],
['pink']], [[8, 4], ['pink']], [[8, 5], ['pink']], [[8, 6], ['pink']], [[8, 7],
['pink']], [[8, 8], ['pink']], [[8, 9], ['pink']]],

            [[[9, 0], ['lime green']], [[9, 1], ['lime green']], [[9, 2], ['lime
green']], [[9, 3], ['lime green']], [[9, 4], ['lime green']], [[9, 5], ['lime
green']], [[9, 6], ['lime green']], [[9, 7], ['lime green']], [[9, 8], ['lime
green']], [[9, 9], ['lime green']]],

            [[[10, 0], ['orange']], [[10, 1], ['orange']], [[10, 2], ['orange']],
[[10, 3], ['orange']], [[10, 4], ['orange']], [[10, 5], ['orange']], [[10, 6],
['orange']], [[10, 7], ['orange']], [[10, 8], ['orange']], [[10, 9], ['orange']]],

            [[[11, 0], ['grey']], [[11, 1], ['grey']], [[11, 2], ['grey']], [[11, 3],
['grey']], [[11, 4], ['grey']], [[11, 5], ['grey']], [[11, 6], ['grey']], [[11, 7],
['grey']], [[11, 8], ['grey']], [[11, 9], ['grey']]],

            ]
```

The colors for each of the faces is also stored in a 1x12 array which is used for displaying the core piece on each face. The code for that data structure is:

```
colors = [['white'], ['blue'], ['red'], ['green'], ['purple'], ['yellow'], ['light
blue'], ['light yellow'], ['pink'], ['lime green'], ['orange'], ['grey']]
```

GUI

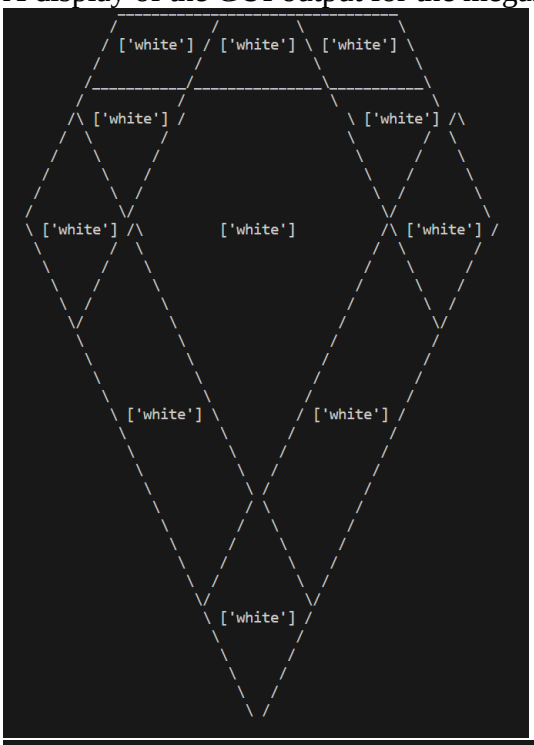
To run the code above and have it displayed to a GUI output, run the code below. It is an ASCII representation of a megaminx. While it is not the prettiest representation of the megaminx, it still shows the location of all of the cubbies on it.

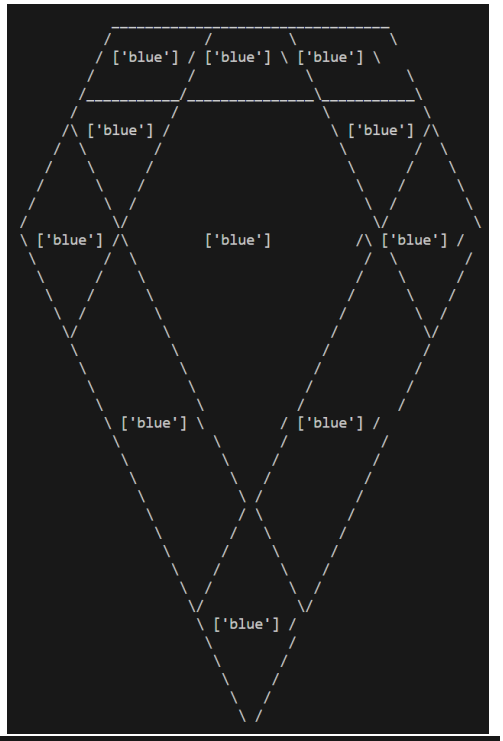
```
def display_megaminx(row):

    print('            _____________________________')

    print('          /          /        \\          \\')

    print('         /', megaminx[row][0][1], '/', megaminx[row][1][1], '\\',
megaminx[row][2][1], '\\')

    print('        /          /            \\          \\')
```

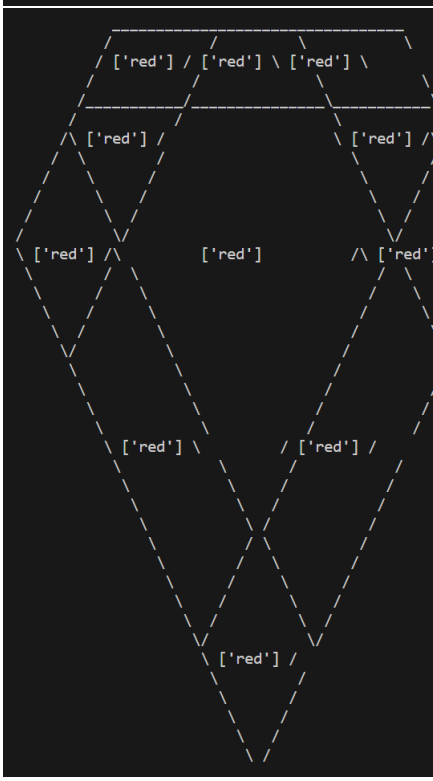
```
    print('        /__________/_______________\\__________\\')
    print('       /          /                \\          \\')
    print('      /\\', megaminx[row][9][1], '/                  \\',
megaminx[row][3][1], '/\\')
    print('     /  \        /                    \\        /  \\')
    print('    /    \      /                      \\      /    \\')
    print('   /      \    /                        \\    /      \\')
    print('  /        \  /                          \\  /        \\')
    print(' /          \/                            \\/          \\')
    print(' \\', megaminx[row][8][1], '/\\        ', colors[row], '         /\\',
megaminx[row][4][1], '/')
    print('  \\        /  \\                          /  \\        /')
    print('   \\      /    \\                        /    \\      /')
    print('    \\    /      \\                      /      \\    /')
    print('     \\  /        \\                    /        \\  /')
    print('      \\/          \\                  /          \\/')
    print('       \\          /\\                /          /')
    print('        \\        /  \\              /          /')
    print('         \\      /    \\            /          /')
    print('          \\    /      \\          /          /')
    print('           \\', megaminx[row][7][1], '\\        /', megaminx[row][5][1],
'/')
    print('            \\      /  \\      /          /')
    print('             \\    /    \\    /          /')
    print('              \\  /      \\  /          /')
    print('               \\/        \\/          /')
    print('                \\        /\\         /')
    print('                 \\      /  \\       /')
    print('                  \\    /    \\     /')
    print('                   \\  /      \\   /')
```

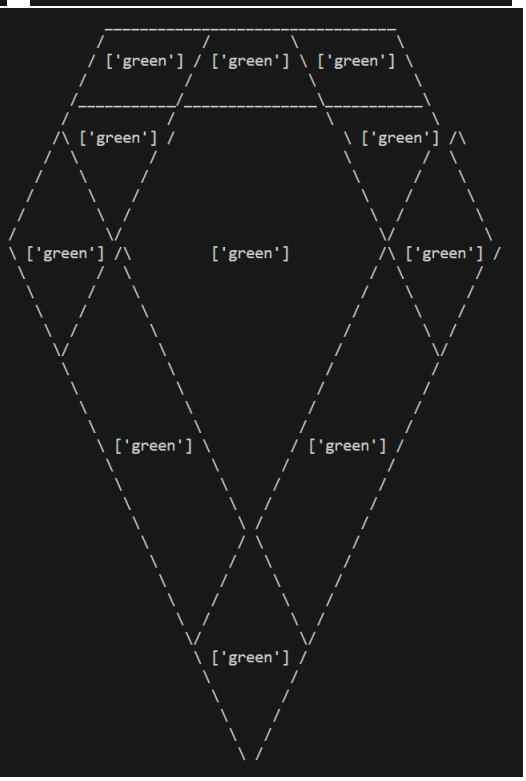
```
print('                  \\  /         \\  /')
print('                   \\/           \\/')
print('                    \\', megaminx[row][6][1], '/')
print('                     \\         /')
print('                      \\       /')
print('                       \\     /')
print('                        \\   /')
print('                         \\ / \n')
```

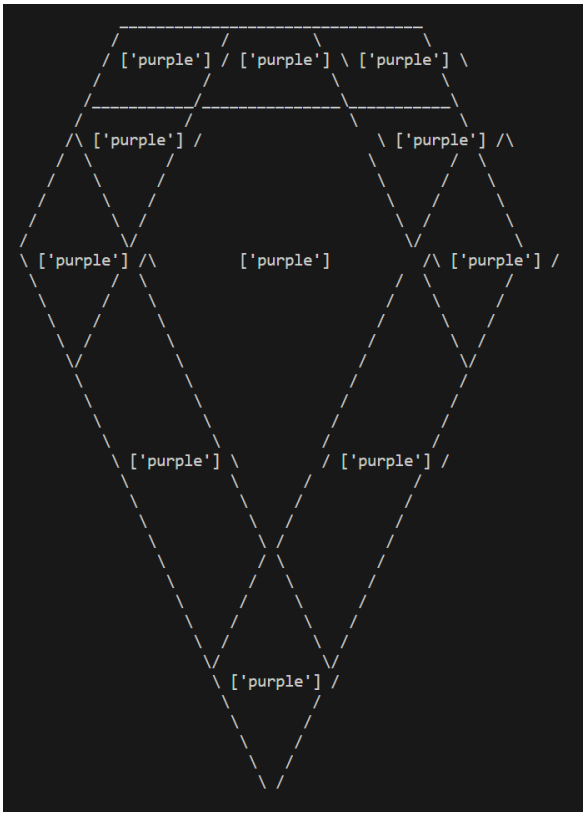
This function to display a face of the megaminx is in a for loop that loops through all 12 faces on the megaminx and prints a visual representation of each face as it goes through the loop. A display of the GUI output for the megaminx is shown below:

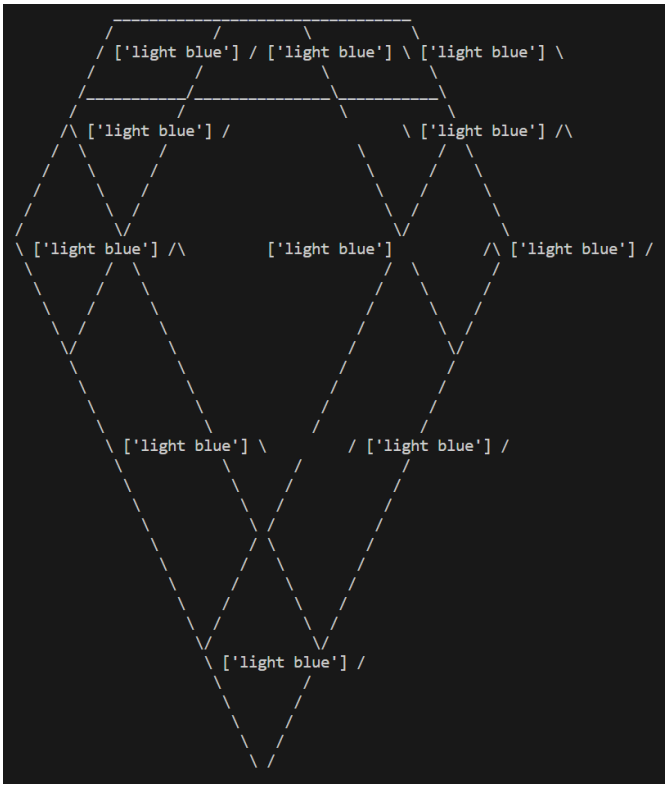


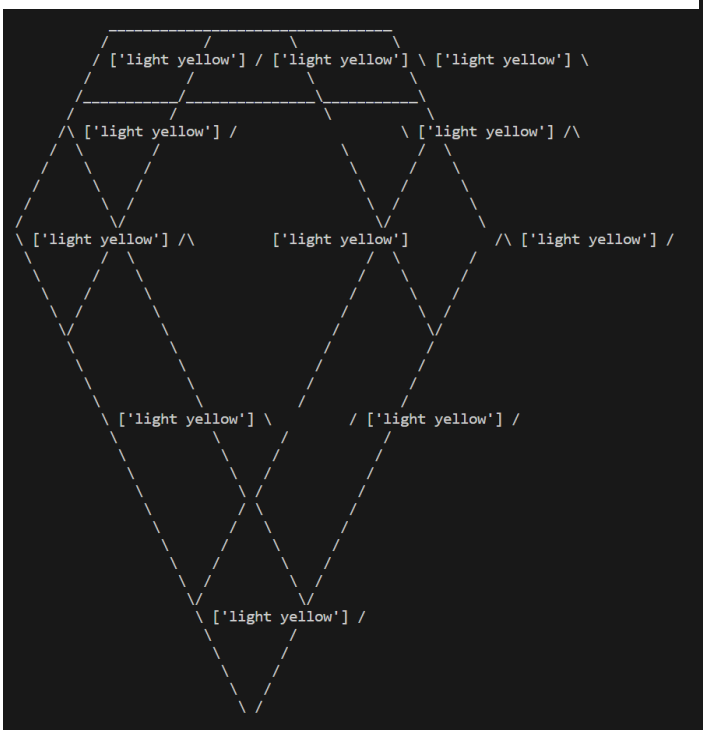


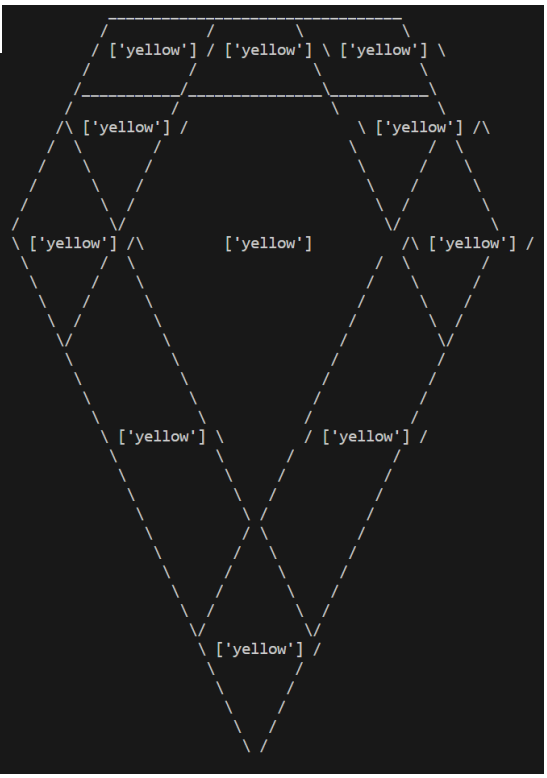


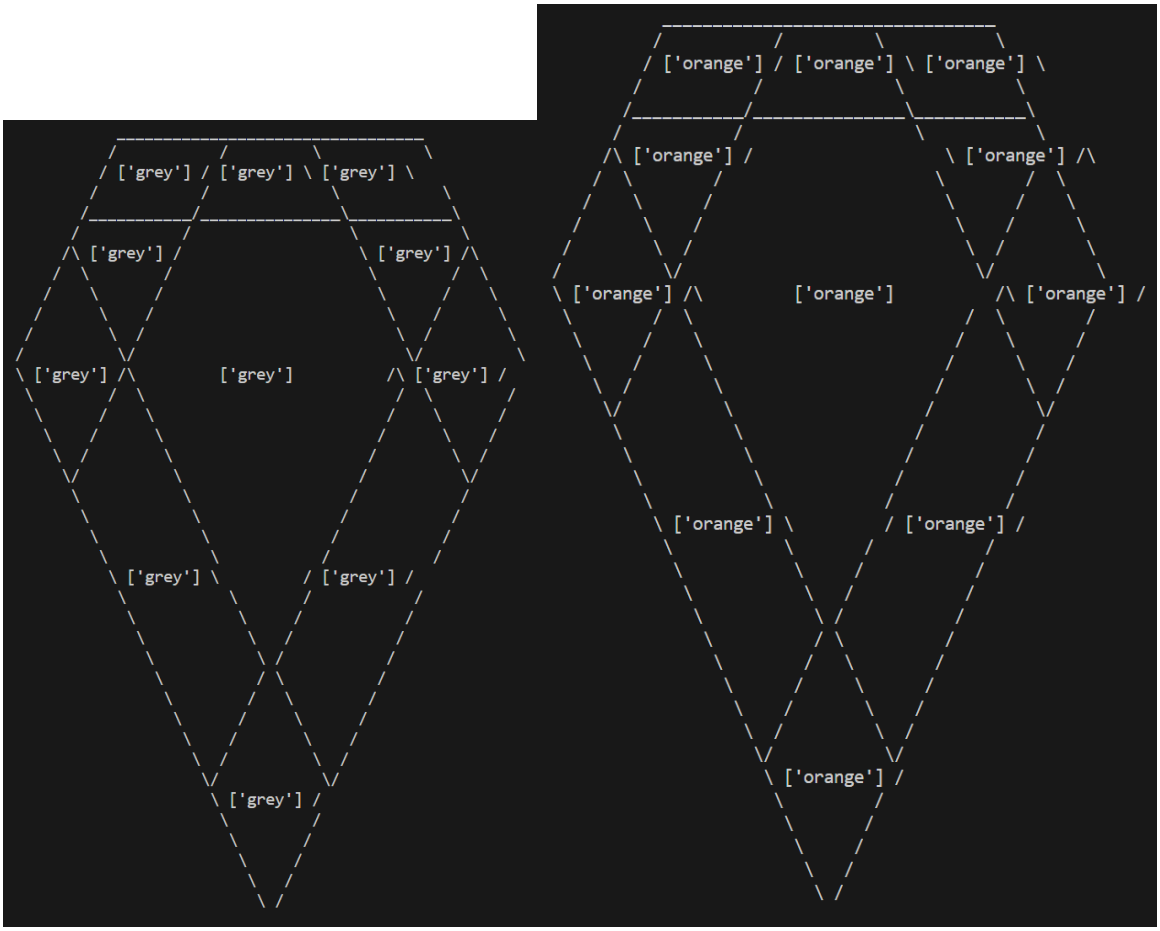


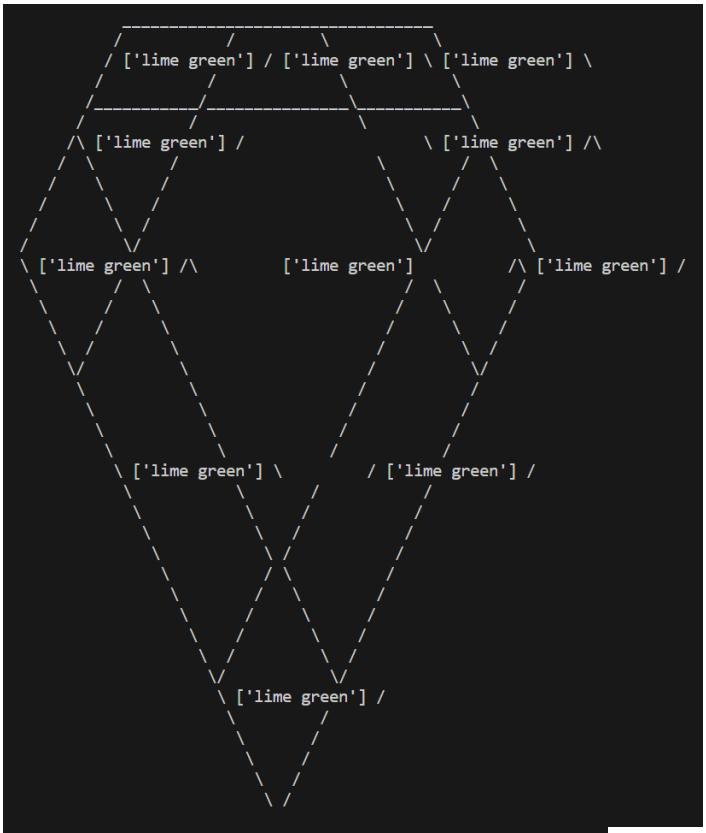

['purple']
['purple']
['purple']
['purple']
['purple']
['purple']
['purple']
['purple']
['purple']
['purple']
['purple']
['purple']
['purple']
['purple']
['purple']

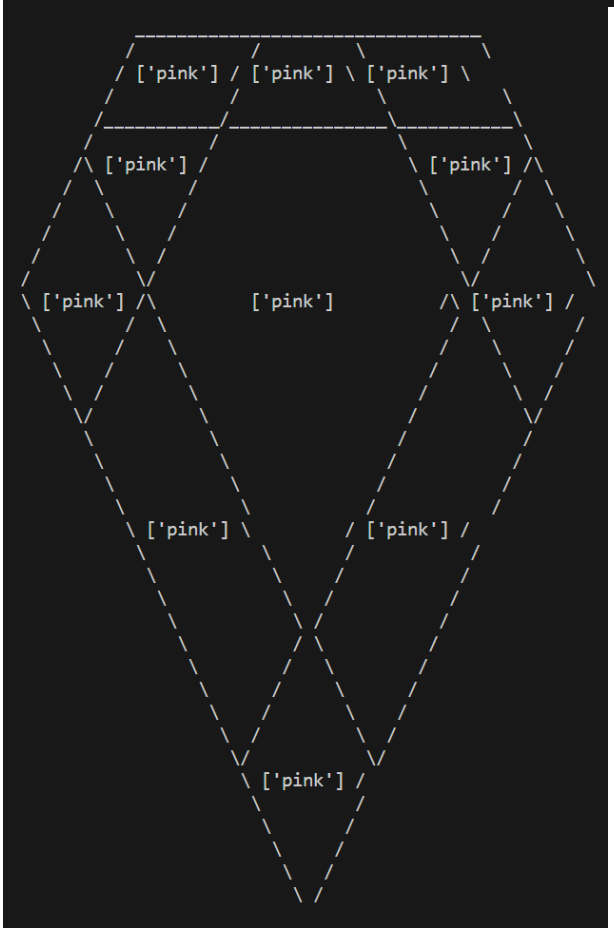

['light blue']
['light blue']
['light blue']
['light blue']
['light blue']
['light blue']
['light blue']
['light blue']
['light blue']
['light blue']
['light blue']


['light yellow']
['light yellow']
['light yellow']
['light yellow']
['light yellow']
['light yellow']
['light yellow']
['light yellow']
['light yellow']
['light yellow']
['light yellow']


['yellow']
['yellow']
['yellow']
['yellow']
['yellow']
['yellow']
['yellow']
['yellow']
['yellow']
['yellow']
['yellow']


['grey'] ['grey'] ['grey']
['grey'] ['grey'] ['grey'] ['grey']
['grey']
['orange'] ['orange'] ['orange']
['orange'] ['orange'] ['orange'] ['orange']
['orange']
['orange']


['lime green'] ['lime green'] ['lime green']
['lime green'] ['lime green'] ['lime green'] ['lime green']
['lime green'] ['lime green']
['lime green']


['pink'] ['pink'] ['pink']
['pink'] ['pink'] ['pink'] ['pink']
['pink'] ['pink']
['pink']

Randomizer

The randomizer used to randomize the megaminx is a function that takes in a given number of rotations and makes that many random rotations on the megaminx. There is a random number generator deciedes whether the turn will be a clockwise rotation or a counterclockwise rotation. It then uses another random number generator to pick a random number between 0 and 11 to choose which face it will rotate. It does this as many times as specified by the user input parameter.

The code for the randomizer is:

```
def randomize(rotations):

    rotation = 0

    while rotation < rotations:

        direction = random.randint(0,1)

        print('Rotation: ',rotation)

        if direction == 0:

            rotate_clockwise(random.randint(0,11))

        else:

            rotate_counterclockwise(random.randint(0,11))

        rotation += 1
```

And to run the randomizer, the user is prompted to enter the number of rotations they would like the randomizer to make. If the user wants the randomizer to make 10 turns, the the user simply needs to enter 10 when prompted for an imput.

```
numberOfRatations = input('How many turns would you like the randomizer to
make?')

randomize(int(numberOfRatations))
```

Heuristic

The heuristic that I will be using in this project is A*. This is a best-first search that finds the optimal path to a given solution. A* is admissible because it never overestimates the cost to reach the solution.

For the megaminx problem, A* will look at the given state and record how many moves the given state is away from completion. This total distance is called the Manhaton distance. It then looks at every possible move that it can take and looks at those moves Manhaton distances of those moves. It then chooses the move that results in the smallest Manhaton distance and does

this process until it reaches the solution. Because it looks at all possible moves and picks the one that results in the smallest Manhattan distance, it will not overestimate the number of moves it takes to find the solution

Learning Outcomes

Through doing this assignment, I learned how to represent a megaminx in code and be able to make rotations on it. In order to make this work, I needed to make sure that I could account for all of the pieces of information on the megaminx as well as keep track of which elements of the megaminx are connected to each other. I also learned to break certain parts of the code that could be used in multiple places into functions so that I could just call the function instead of manually typing several lines of code over and over again.

Who Did What

For this project, I did all of the work by myself. I talked to my peers that sit near me in class during in class discussions to get ideas on data structures and how we might do the rotations. Asides from the in class discussions, all of the work in this project is my own