Spencer Cooperman
CS 463
9/17/2023

A* on the Megaminx

Data structures

The data structure that I used to create the megaminx was a dictionary that contained all 12 faces as well as a tuple of all adjacent faces for each of the 12 faces. Throught the code, lists were created with the values associated with the cubies on each of the faces. These lists were then modified as faces were rotated to acount for the new position of the cubies. The main datastructure used in the search algorithm was the priority queue. This queue allowed the searching algorithm to look at the children nodes that returned the lowest f-value so it could find the solution.

Heuristic

The heuristic that I used in this assignment was take all of the cubies that are out of place and divide it by 15 rounded up. The total number of cubies that are out of place is divided by 15 because when a face moves, there are 5 faces that are rotating and each face has 3 pieces. This means there can be at most 15 cubies that are out of place after 1 rotation. If there are 15 cubies that are out of place, then we know that the megaminx is aproximately 1 turn away from compleation. There can be at most 120 cubeis that are out of place so that would mean that the heuristic value would be 8. Since that is the most amount of cubies that you can have out of place, 8 would be the max heuristic value while 1 is the lowest. Which ever child state has the lowest heuristic value is in theory closser to being solved than the other children states. A heurstic value of 0 would mean that the megaminx is solved in which case the solution has been found.

Description of code

Because many megaminx objects are created in this program, a megaminx class was created so that each megaminx object that was in the code had the same properties and it was easy to use. Every megaminx object had a dictionary of the origional colors of every cubie on each face as well as several functions to allow things such as finding and rotating a face, calculating heuristic values, displaying the megaminx, cloning, and creating children.

The code is nested in a double for loop. The first loop starts at 3 and goes to 20 and this loop provides the k-value which is used to randomize the megaminx. The second loop runs 5 times and creates and solves 5 megaminx's for each k-value. The first thing done in the main body of code is the megaminx, priority queue and variables are initialized. Once this is done, the starting megaminx is randomized. To do this, the k value from the first loop is passed into the randomize function where a random face is chosen and is rotated k times. To rotate the face, I used list splicing to move the last 2 nodes to the beginning or the first to nodes to the end depending on whether the face was being rotated clockwise or counterclockwise. Since the randomize function only rotates, clockwise, it will always move the last 2 nodes to the beginning. This works because when the face is rotated, all the face value will go down by 2 so if the last 2 elements are moved to the front, it is the same as subtracting 2 from all the nodes in the list. After this is done, the starting index for each of the adjacent faces is found which is then used to find the adjacent edges. These are both found by using list comprehension and then just like how the face was rotated, list splicing is used to modify the face values.

After the megaminx is randomized, it is added to the priority queue and the then the search for the solution begins. To find the solution to the megaminx, the while loop gets the first thing in the priority queue and removes it. It then checks to see if it is solved. If it is solved, it records the route it took to get to the solution and records it. If the megaminx is not solved, then it expands the node and creates 12 children. This process of checking nodes for compleation and creating children repeats until a solution is found

In the create children function, the children are deep copies of the megaminx which are created using list comprehension. It makes a clone of the megaminx for each face in the original dictionary and then it rotates a face on the newly created child corresponding to the face in the dictionary. The heuristic value is then calculated and then added with the depth to create the f-value. This f-value and the child are then added into the priority queue. After the function is called, a counter goes up by one to record the number of times a node was expanded.

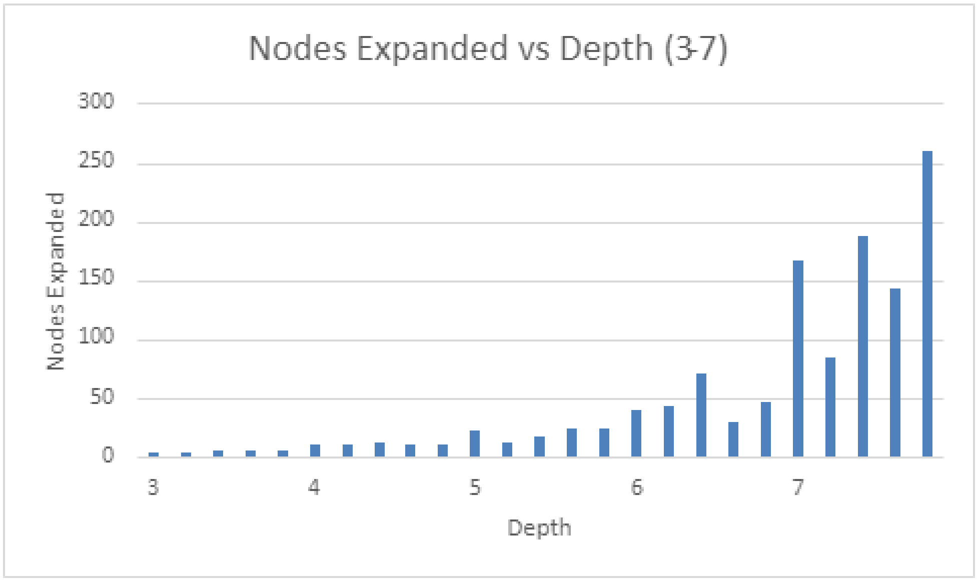
After A solution has been found, the results of the algorithm are displayed. First, the origional steps to randomize the megaminx are displayed. This is followed by the path the A* algorithm took to solve the megaminx. After that, the number of times a node in the tree is expanded is shown followed by the depth of the tree.

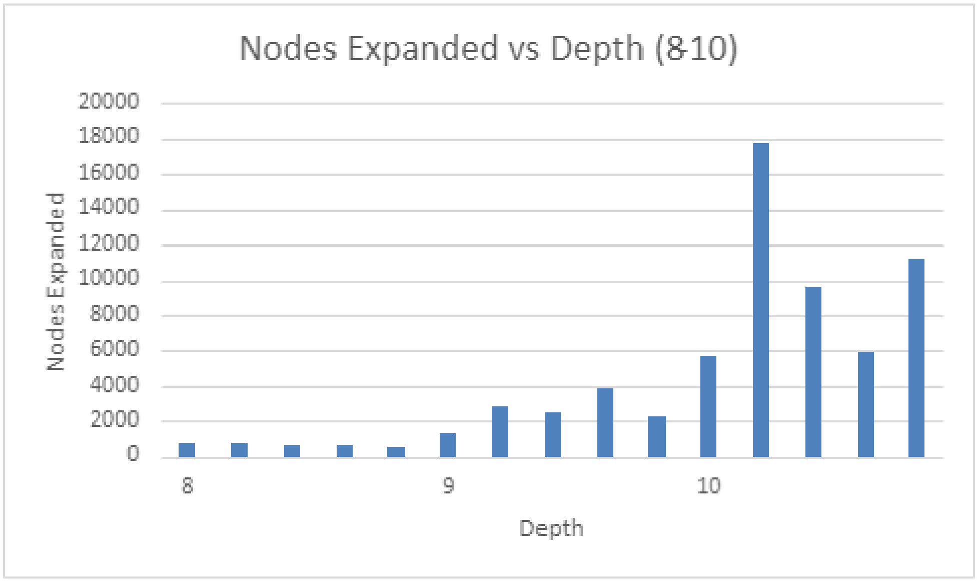
How to run the code

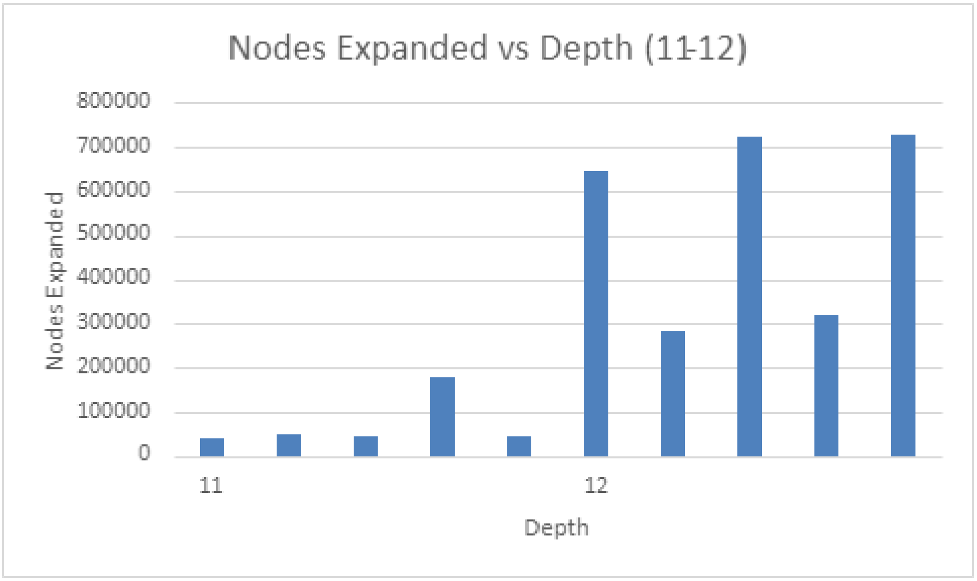
To run the code, open up whichever terminal you use and go to where the folder that has this document in it. It is called, “A on the Megaminx”, and it contains a python script also called, “A on the Megaminx”. To run the code, enter “python3 ‘.\ A on the Megaminx.py’” into the terminal and the code will run.

Graphs

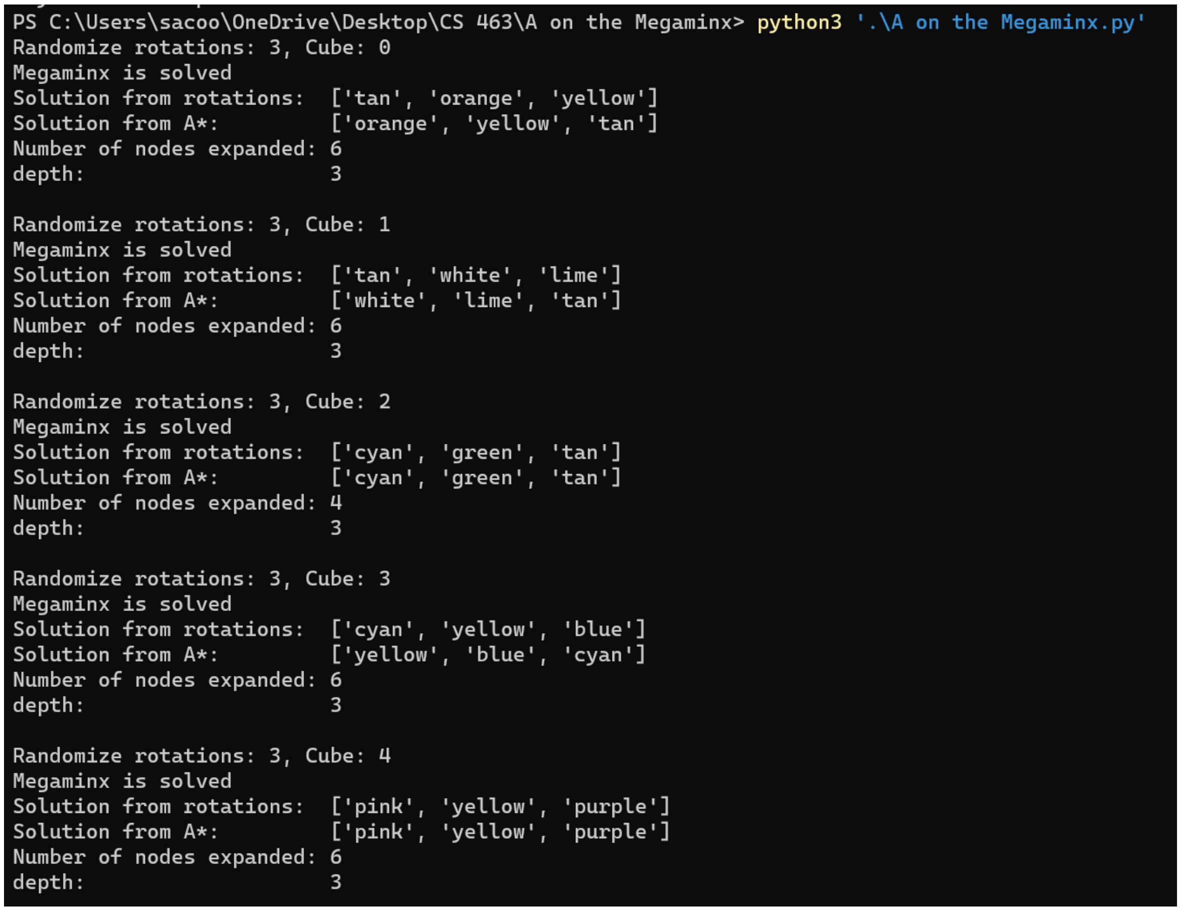
Below are graphs showing the relationship between the depth of the tree and the number of nodes that are expanded






Nodes Expanded vs Depth (11-12)
800000
700000
600000
500000
400000
300000
200000
100000
0
Nodes Expanded
11
12
Depth

Below is what the output of the code looks like but also the rlationship between the origional order in which the megaminx was randomized to the order in which it was solved. Note, the screenshots were taken after the graphs were made so the number of nodes expanded in the graphs will not exactly match the number of nodes explanded in the screenshot of the code output.



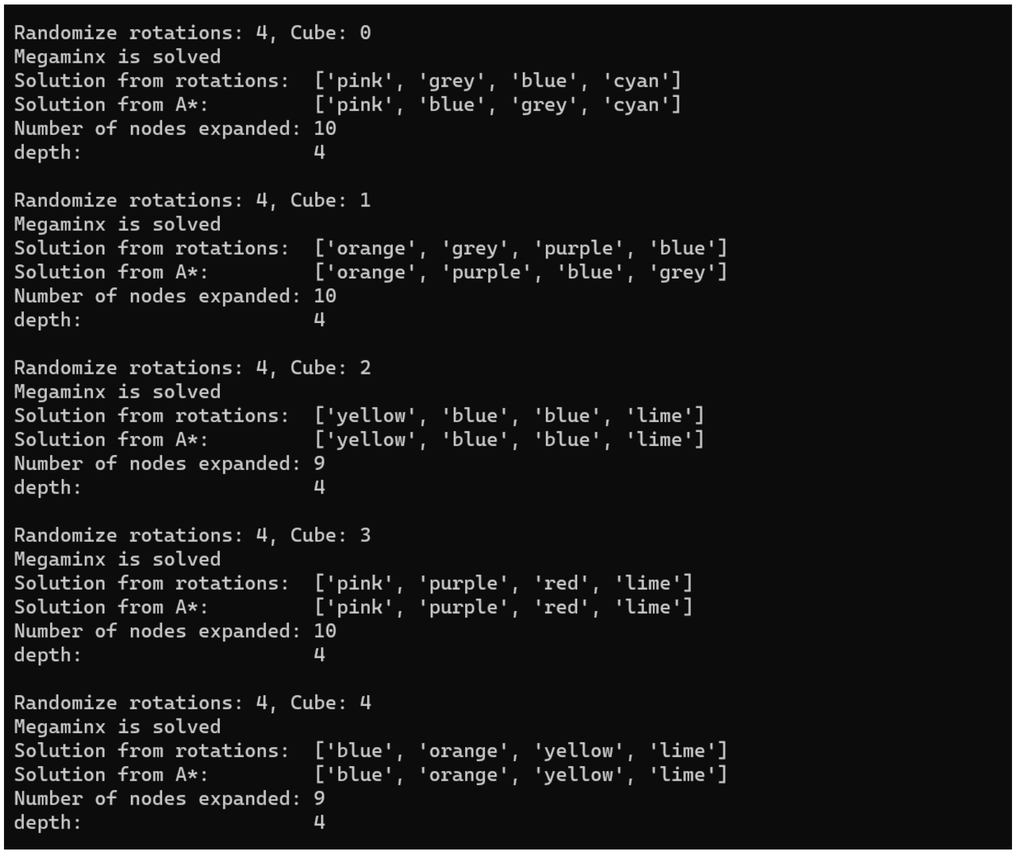
```
PS C:\Users\sacoo\OneDrive\Desktop\CS 463\A on the Megaminx> python3 '.\A on the Megaminx.py'
Randomize rotations: 3, Cube: 0
Megaminx is solved
Solution from rotations:  ['tan', 'orange', 'yellow']
Solution from A*:         ['orange', 'yellow', 'tan']
Number of nodes expanded: 6
depth:                    3

Randomize rotations: 3, Cube: 1
Megaminx is solved
Solution from rotations:  ['tan', 'white', 'lime']
Solution from A*:         ['white', 'lime', 'tan']
Number of nodes expanded: 6
depth:                    3

Randomize rotations: 3, Cube: 2
Megaminx is solved
Solution from rotations:  ['cyan', 'green', 'tan']
Solution from A*:         ['cyan', 'green', 'tan']
Number of nodes expanded: 4
depth:                    3

Randomize rotations: 3, Cube: 3
Megaminx is solved
Solution from rotations:  ['cyan', 'yellow', 'blue']
Solution from A*:         ['yellow', 'blue', 'cyan']
Number of nodes expanded: 6
depth:                    3

Randomize rotations: 3, Cube: 4
Megaminx is solved
Solution from rotations:  ['pink', 'yellow', 'purple']
Solution from A*:         ['pink', 'yellow', 'purple']
Number of nodes expanded: 6
depth:                    3
```

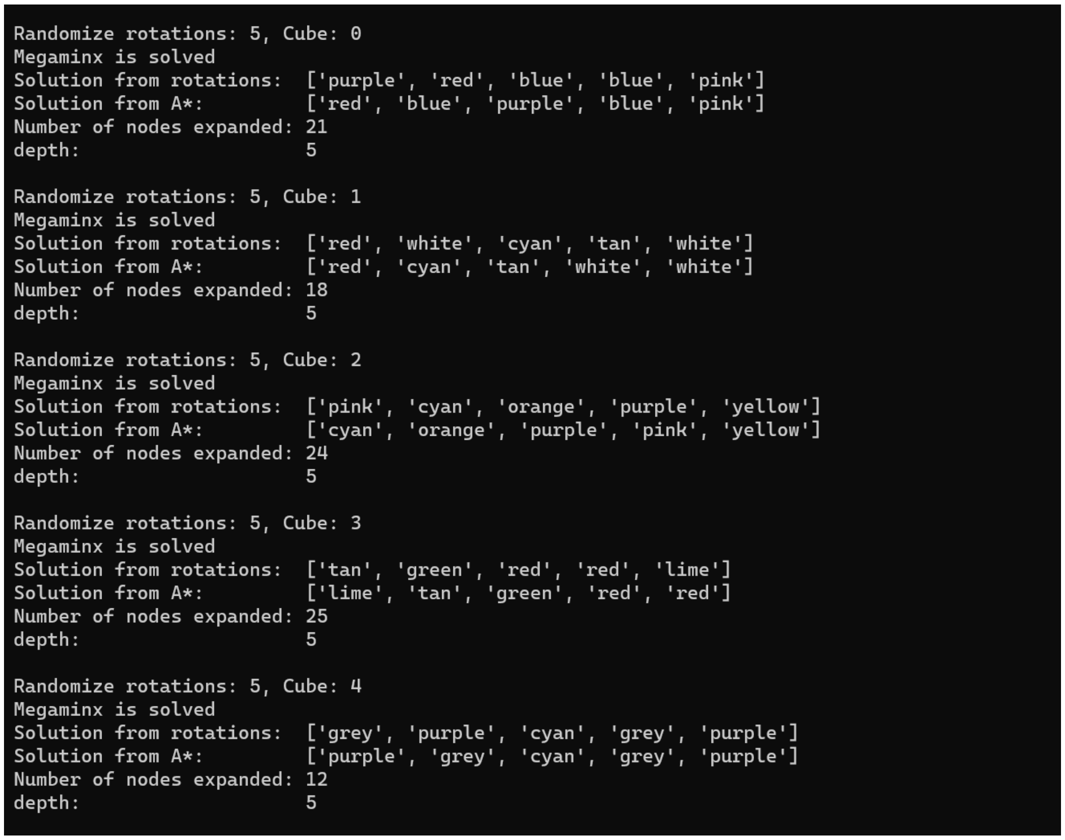

```
Randomize rotations: 4, Cube: 0
Megaminx is solved
Solution from rotations:  ['pink', 'grey', 'blue', 'cyan']
Solution from A*:         ['pink', 'blue', 'grey', 'cyan']
Number of nodes expanded: 10
depth:                    4

Randomize rotations: 4, Cube: 1
Megaminx is solved
Solution from rotations:  ['orange', 'grey', 'purple', 'blue']
Solution from A*:         ['orange', 'purple', 'blue', 'grey']
Number of nodes expanded: 10
depth:                    4

Randomize rotations: 4, Cube: 2
Megaminx is solved
Solution from rotations:  ['yellow', 'blue', 'blue', 'lime']
Solution from A*:         ['yellow', 'blue', 'blue', 'lime']
Number of nodes expanded: 9
depth:                    4

Randomize rotations: 4, Cube: 3
Megaminx is solved
Solution from rotations:  ['pink', 'purple', 'red', 'lime']
Solution from A*:         ['pink', 'purple', 'red', 'lime']
Number of nodes expanded: 10
depth:                    4

Randomize rotations: 4, Cube: 4
Megaminx is solved
Solution from rotations:  ['blue', 'orange', 'yellow', 'lime']
Solution from A*:         ['blue', 'orange', 'yellow', 'lime']
Number of nodes expanded: 9
depth:                    4
```

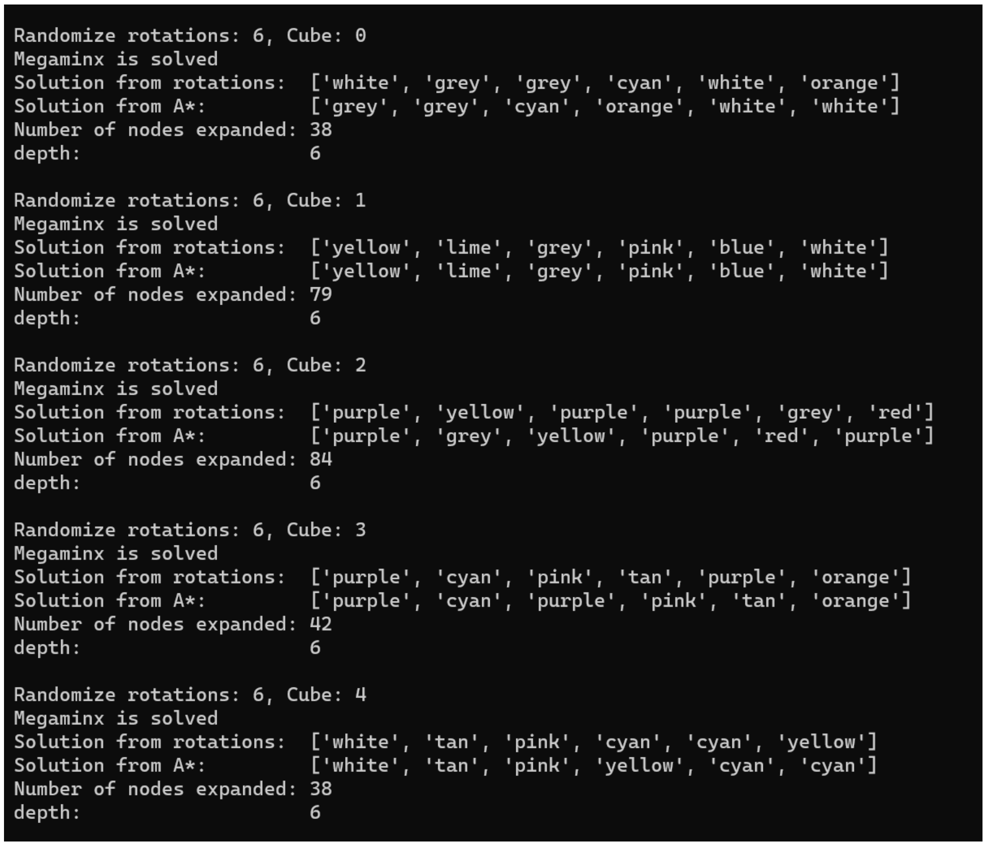

```
Randomize rotations: 5, Cube: 0
Megaminx is solved
Solution from rotations:  ['purple', 'red', 'blue', 'blue', 'pink']
Solution from A*:         ['red', 'blue', 'purple', 'blue', 'pink']
Number of nodes expanded: 21
depth:                    5

Randomize rotations: 5, Cube: 1
Megaminx is solved
Solution from rotations:  ['red', 'white', 'cyan', 'tan', 'white']
Solution from A*:         ['red', 'cyan', 'tan', 'white', 'white']
Number of nodes expanded: 18
depth:                    5

Randomize rotations: 5, Cube: 2
Megaminx is solved
Solution from rotations:  ['pink', 'cyan', 'orange', 'purple', 'yellow']
Solution from A*:         ['cyan', 'orange', 'purple', 'pink', 'yellow']
Number of nodes expanded: 24
depth:                    5

Randomize rotations: 5, Cube: 3
Megaminx is solved
Solution from rotations:  ['tan', 'green', 'red', 'red', 'lime']
Solution from A*:         ['lime', 'tan', 'green', 'red', 'red']
Number of nodes expanded: 25
depth:                    5

Randomize rotations: 5, Cube: 4
Megaminx is solved
Solution from rotations:  ['grey', 'purple', 'cyan', 'grey', 'purple']
Solution from A*:         ['purple', 'grey', 'cyan', 'grey', 'purple']
Number of nodes expanded: 12
depth:                    5
```

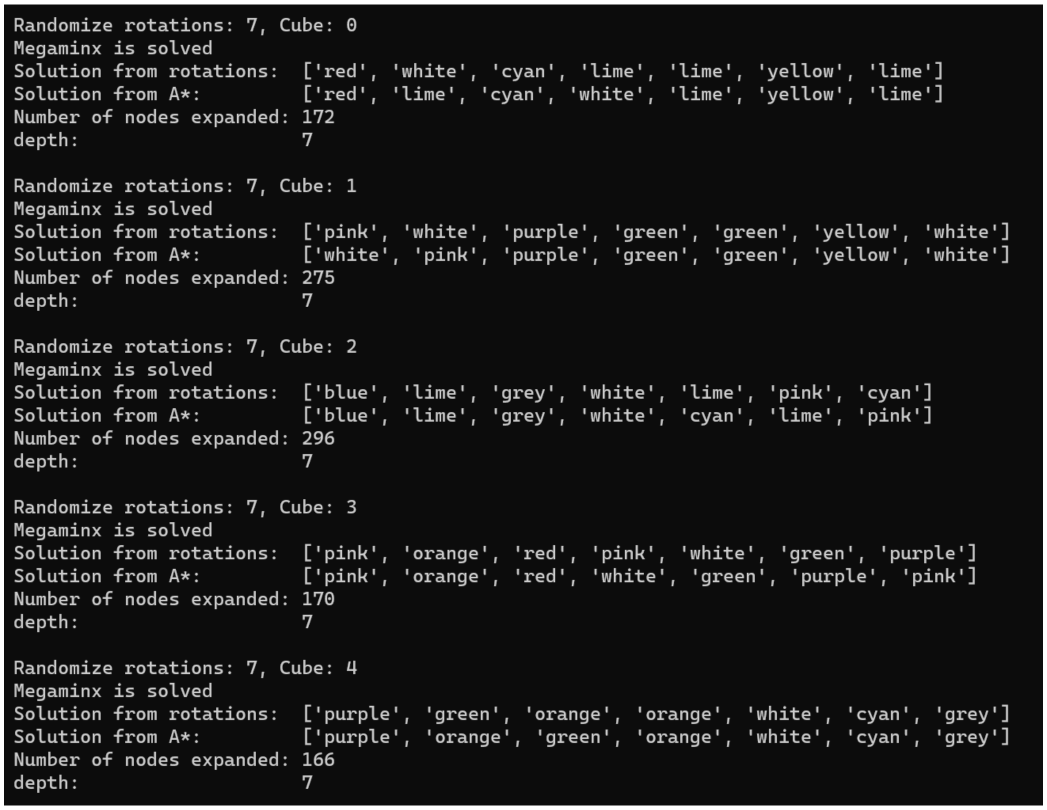

```
Randomize rotations: 6, Cube: 0
Megaminx is solved
Solution from rotations:  ['white', 'grey', 'grey', 'cyan', 'white', 'orange']
Solution from A*:         ['grey', 'grey', 'cyan', 'orange', 'white', 'white']
Number of nodes expanded: 38
depth:                    6

Randomize rotations: 6, Cube: 1
Megaminx is solved
Solution from rotations:  ['yellow', 'lime', 'grey', 'pink', 'blue', 'white']
Solution from A*:         ['yellow', 'lime', 'grey', 'pink', 'blue', 'white']
Number of nodes expanded: 79
depth:                    6

Randomize rotations: 6, Cube: 2
Megaminx is solved
Solution from rotations:  ['purple', 'yellow', 'purple', 'purple', 'grey', 'red']
Solution from A*:         ['purple', 'grey', 'yellow', 'purple', 'red', 'purple']
Number of nodes expanded: 84
depth:                    6

Randomize rotations: 6, Cube: 3
Megaminx is solved
Solution from rotations:  ['purple', 'cyan', 'pink', 'tan', 'purple', 'orange']
Solution from A*:         ['purple', 'cyan', 'purple', 'pink', 'tan', 'orange']
Number of nodes expanded: 42
depth:                    6

Randomize rotations: 6, Cube: 4
Megaminx is solved
Solution from rotations:  ['white', 'tan', 'pink', 'cyan', 'cyan', 'yellow']
Solution from A*:         ['white', 'tan', 'pink', 'yellow', 'cyan', 'cyan']
Number of nodes expanded: 38
depth:                    6
```

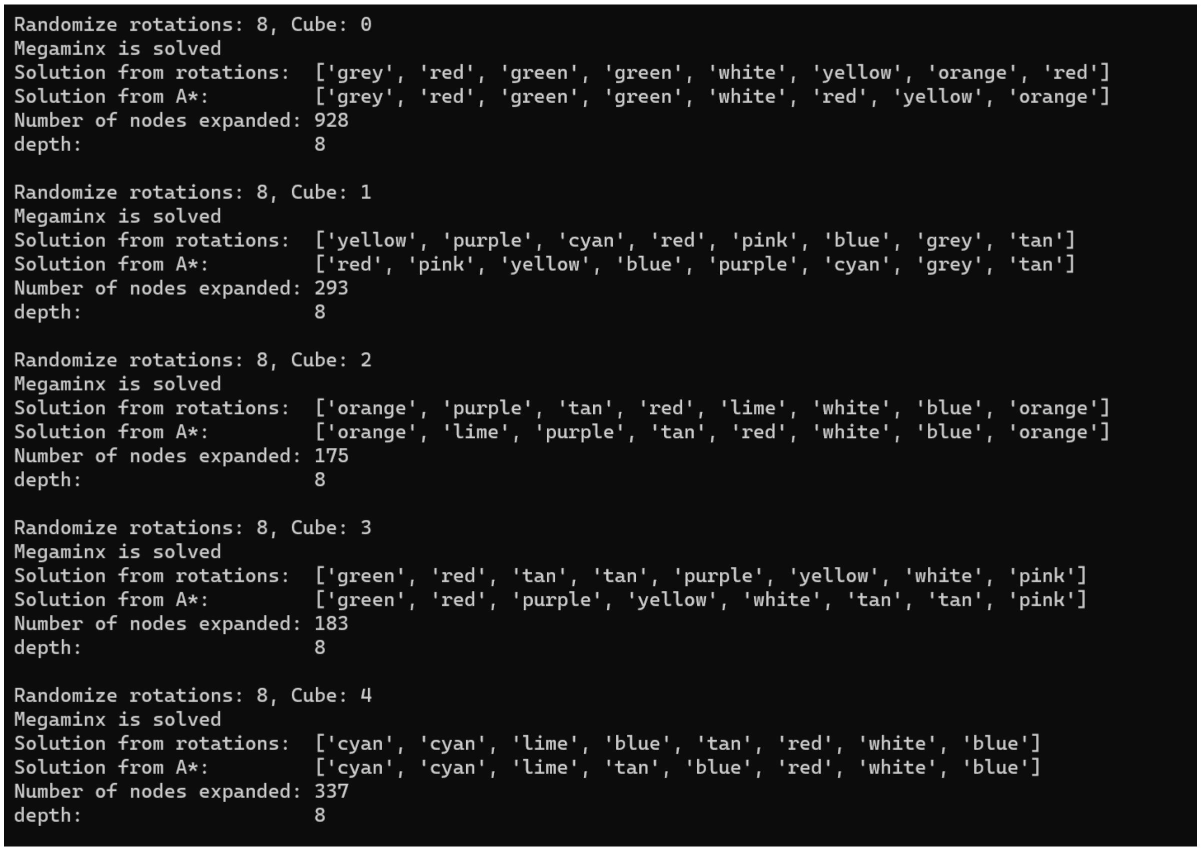

```
Randomize rotations: 7, Cube: 0
Megaminx is solved
Solution from rotations:  ['red', 'white', 'cyan', 'lime', 'lime', 'yellow', 'lime']
Solution from A*:         ['red', 'lime', 'cyan', 'white', 'lime', 'yellow', 'lime']
Number of nodes expanded: 172
depth:                    7

Randomize rotations: 7, Cube: 1
Megaminx is solved
Solution from rotations:  ['pink', 'white', 'purple', 'green', 'green', 'yellow', 'white']
Solution from A*:         ['white', 'pink', 'purple', 'green', 'green', 'yellow', 'white']
Number of nodes expanded: 275
depth:                    7

Randomize rotations: 7, Cube: 2
Megaminx is solved
Solution from rotations:  ['blue', 'lime', 'grey', 'white', 'lime', 'pink', 'cyan']
Solution from A*:         ['blue', 'lime', 'grey', 'white', 'cyan', 'lime', 'pink']
Number of nodes expanded: 296
depth:                    7

Randomize rotations: 7, Cube: 3
Megaminx is solved
Solution from rotations:  ['pink', 'orange', 'red', 'pink', 'white', 'green', 'purple']
Solution from A*:         ['pink', 'orange', 'red', 'white', 'green', 'purple', 'pink']
Number of nodes expanded: 170
depth:                    7

Randomize rotations: 7, Cube: 4
Megaminx is solved
Solution from rotations:  ['purple', 'green', 'orange', 'orange', 'white', 'cyan', 'grey']
Solution from A*:         ['purple', 'orange', 'green', 'orange', 'white', 'cyan', 'grey']
Number of nodes expanded: 166
depth:                    7
```



```
Randomize rotations: 8, Cube: 0
Megaminx is solved
Solution from rotations:  ['grey', 'red', 'green', 'green', 'white', 'yellow', 'orange', 'red']
Solution from A*:         ['grey', 'red', 'green', 'green', 'white', 'red', 'yellow', 'orange']
Number of nodes expanded: 928
depth:                    8

Randomize rotations: 8, Cube: 1
Megaminx is solved
Solution from rotations:  ['yellow', 'purple', 'cyan', 'red', 'pink', 'blue', 'grey', 'tan']
Solution from A*:         ['red', 'pink', 'yellow', 'blue', 'purple', 'cyan', 'grey', 'tan']
Number of nodes expanded: 293
depth:                    8

Randomize rotations: 8, Cube: 2
Megaminx is solved
Solution from rotations:  ['orange', 'purple', 'tan', 'red', 'lime', 'white', 'blue', 'orange']
Solution from A*:         ['orange', 'lime', 'purple', 'tan', 'red', 'white', 'blue', 'orange']
Number of nodes expanded: 175
depth:                    8

Randomize rotations: 8, Cube: 3
Megaminx is solved
Solution from rotations:  ['green', 'red', 'tan', 'tan', 'purple', 'yellow', 'white', 'pink']
Solution from A*:         ['green', 'red', 'purple', 'yellow', 'white', 'tan', 'tan', 'pink']
Number of nodes expanded: 183
depth:                    8

Randomize rotations: 8, Cube: 4
Megaminx is solved
Solution from rotations:  ['cyan', 'cyan', 'lime', 'blue', 'tan', 'red', 'white', 'blue']
Solution from A*:         ['cyan', 'cyan', 'lime', 'tan', 'blue', 'red', 'white', 'blue']
Number of nodes expanded: 337
depth:                    8
```

Learning outcome

Through this assignment, I learned how to code more efficiently by finding beter ways to have code do the same thing in less lines. I also learned more about classes and how they can be used to simplify the code so that it is easier to preform the same operations on many different copys of an object. When starting this assignment, I knew visualy what this program would look like but implementing it was much more difficult than I had thought. One area where I ran into a lot of problems was with the priority queue. I couldn't seem to get it to work and I found a few different ways people had implemented it but I managed to get it to work correctly and was able to use that to allow the search algorithm to look at the nodes that were on the top of the priority queue.

Who-did-what

I did this project by myself but I did get my heuristic from another classmate through the discussion assignment. I will credit them in the comments section.