I chose to implement the Misra & Gries edge coloring algorithm, a constructive algorithm of Vizing's theorem. This algorithm colors a graph using $\leq \Delta + 1$ colors. The user is able to build a graph in the GUI, run the algorithm, then edit the graph and do it all over again.

The Misra & Gries edge coloring algorithm is very similar to Vizing's proof. Each iteration of the algorithm picks an uncolored edge $uv$ to color.

- First a maximal fan $F[1:k]$ of $u$ is created, and the first vertex in the fan is $v$ (so the first fan edge is the one we want to color).
  The fan for the algorithm is similar to the fan in Vizing's proof. We keep adding vertices to the fan until we can't preserve the fan definition. The algorithm doesn't use the term 'missing' and instead uses the term 'free' to describe a color that is not incident to a vertex.
  Just like in Vizing's proof, the fan has $k$ neighbors of $u$ and $k-1$ colors used.
  In the algorithm, the color of edge $(F[i+1], u)$ is free on vertex $F[i]$. Unlike Vizing's proof, we don't keep track of a specific missing color on $F[i]$ and make sure it isn't equal to the missing color of $u$.
  The fan used also doesn't have the extra requirement that Vizing's proof has, checking that $F[i]$ does not miss any color already missed by $F[j]$ $(j < i)$.

- Next we find a free color $c$ on vertex $u$ and a free color $d$ on vertex $F[k]$ (the last vertex in the fan). This is like the missing colors of $u$ and $F[k]$ that Vizing's proof keeps track of.

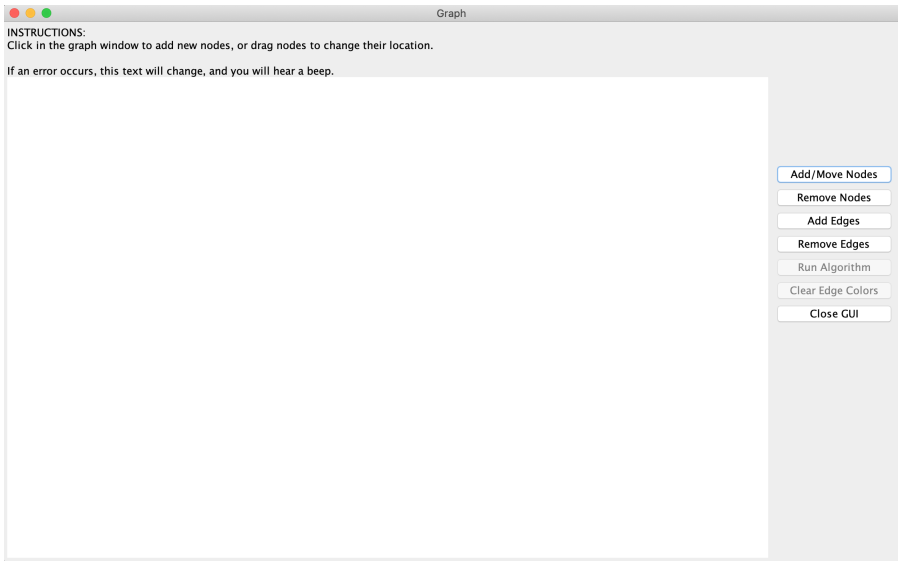
- Like in Vizing's proof, we look for an alternating path of these two colors $c$ and $d$ that touches $u$, so then we can swap colors on the path.

- Next, we find a subset of fan $F$ to rotate. This subset is fan $F' = [F[1]...w]$. Before we swap colors on the path, we have two cases. Either a fan edge had color $d$ or no fan edge had color $d$. The paper explains how to choose $w$ in each case.
  Rotating a fan is like the chain reaction in Vizing's proof. The edge color of $(F[i], u)$ changes to be the edge color of $(F[i+1], u)$. The last fan edge $(w, u)$ gets uncolored.

- Last, we color $(w, u)$ with color $d$.

How to run the code:

- First compile with `javac *.java`
- Run GUI with `java GraphGUI.java`




What the GUI looks like when you start.

Details

- I created an undirected graph that doesn't allow for duplicate edges or self-loop edges. The original paper also has these two edge restrictions on the graph.
- Instructions and error messages are displayed in the GUI.
- I randomly pick RGB colors to display in the GUI. Since some edge colors can be similar, I also show edge color labels (c0, c1, ...)
- In the GUI, you can only run the algorithm if there is an edge in the graph. You can run the algorithm on a disconnected graph.
- In the GUI, you can only clear the edge colors after the algorithm has been run.

- When the graph gets complicated, it can be difficult to know which edge labels go with which edge. So I run a test to see if the edge coloring is valid and then I print the result out in the instructions of the GUI.

- When building a graph, you can add nodes, change the location of nodes (edges will be moved with the node), add edges between two nodes, remove edges, and remove nodes.

Dfficulties encountered:

- Understanding the fan definition was tricky from the wikipedia page. The lecture was helpful for that.

- Understanding what a free color was took a couple of attempts while I was writing that section of code.

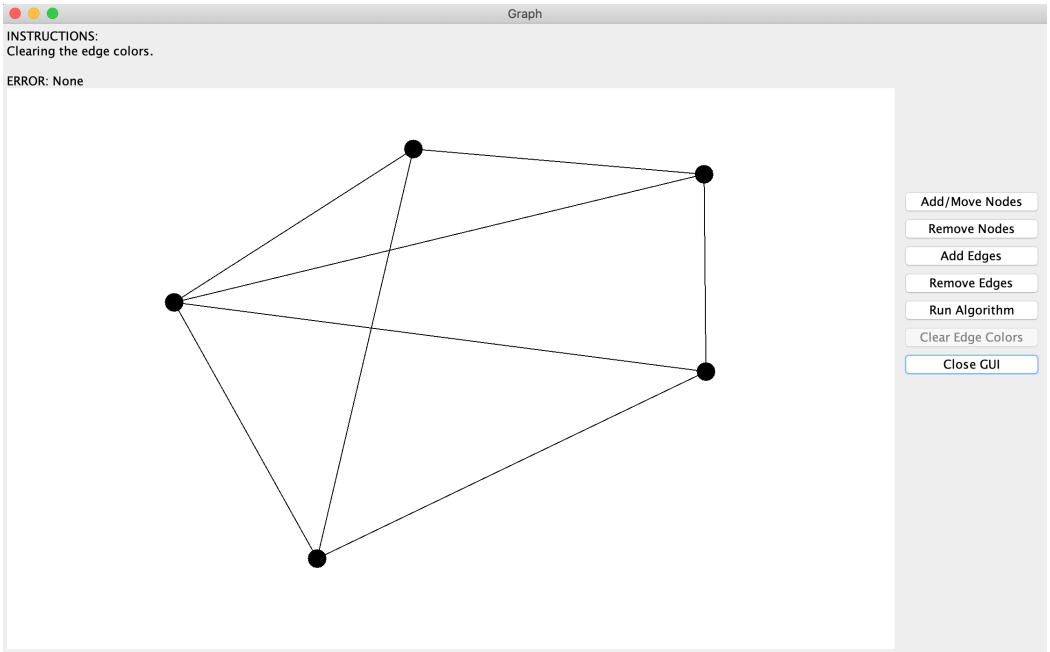
- How to find the path to invert using a BFS strategy. I had written code for BFS from another class so I reused some of that.

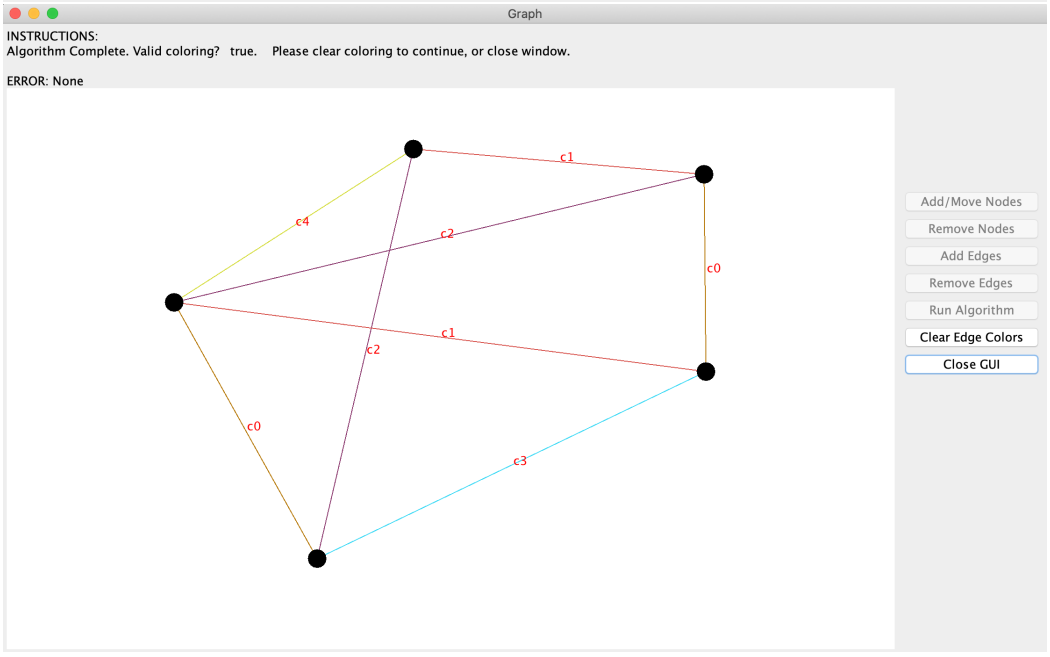
- I used $K_5$ and $K_6$ to continue testing my code as I made changes. I was having trouble getting the algorithm to work. I originally had a different idea for finding the $F'$ fan by just looking at the pseudocode. But then I read more of the Wikipedia page and it tells you how to select the fan for the different cases that could arise.

- During my debugging I also changed how I find the max fan. Originally I went through all the neighbors once and added them to the fan if it passed the condition. After reading the paper, I changed the function to keep a list of nodes not in the fan and keep trying until no nodes can be added.

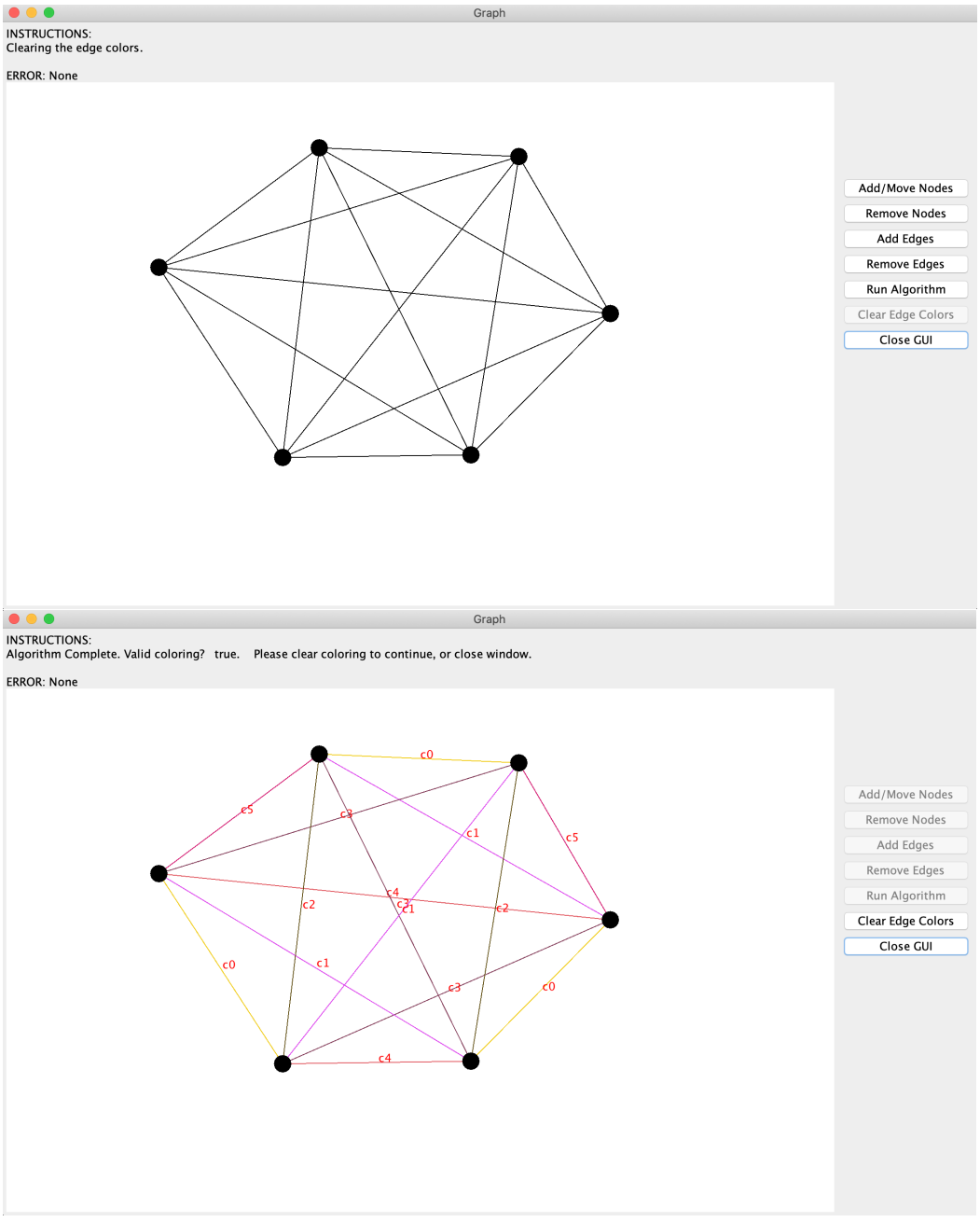
It seems like the algorithm finally works.

Experiment #1: A simple example before and after coloring. The algorithm produced a valid coloring.

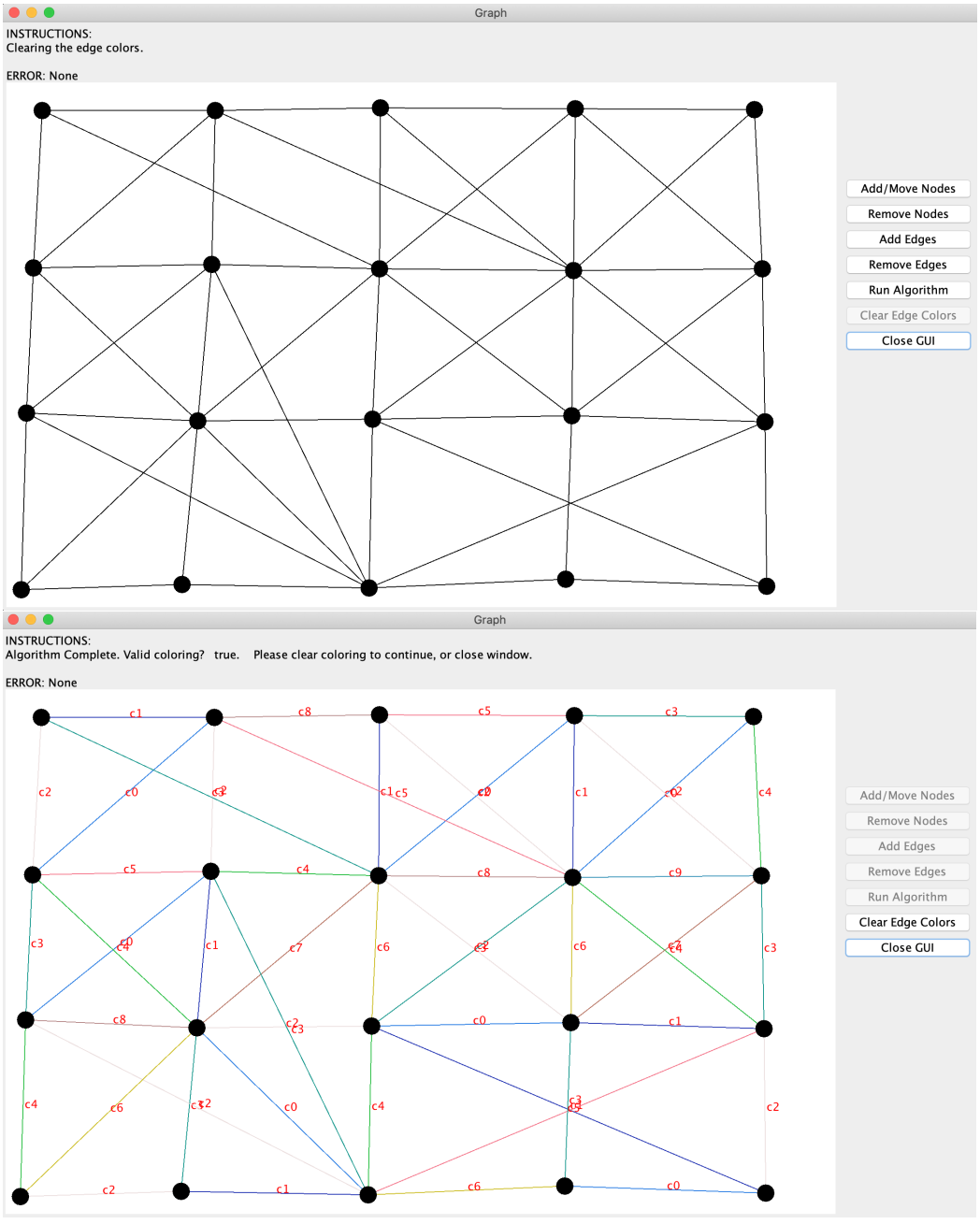


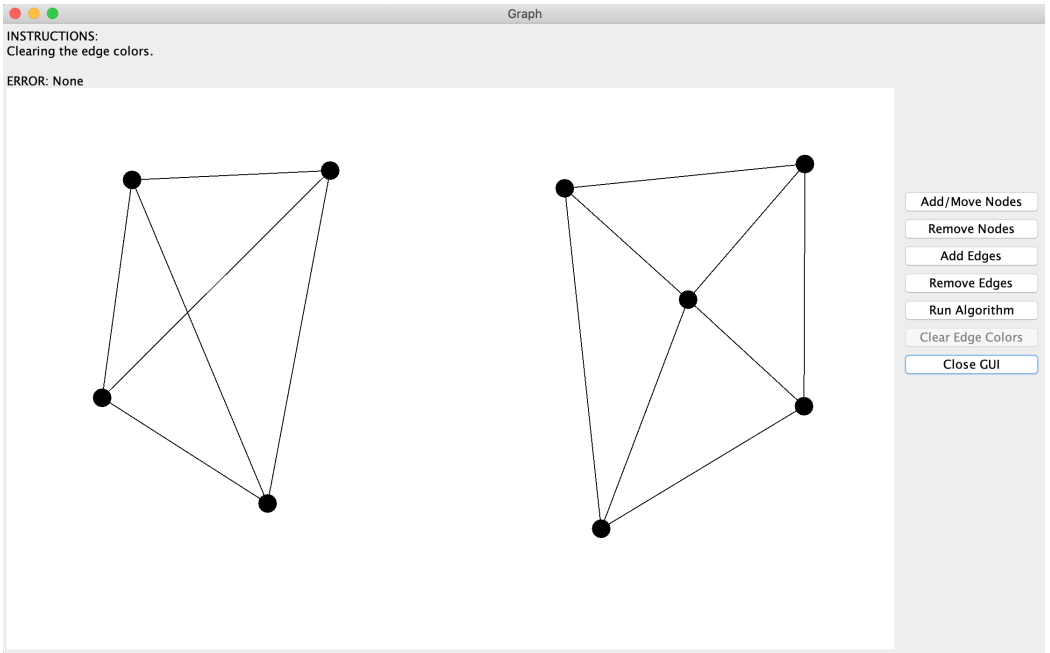
Experiment #2: $K_6$ before and after coloring. The algorithm produced a valid coloring.

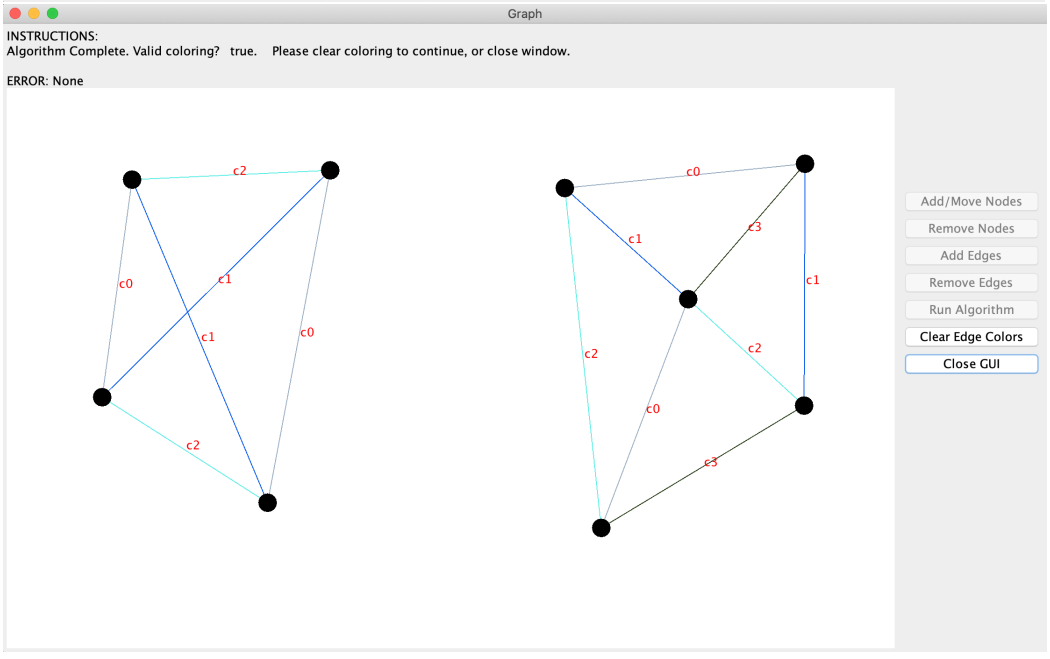


Experiment #3: A bigger graph before and after coloring. The algorithm produced a valid coloring.

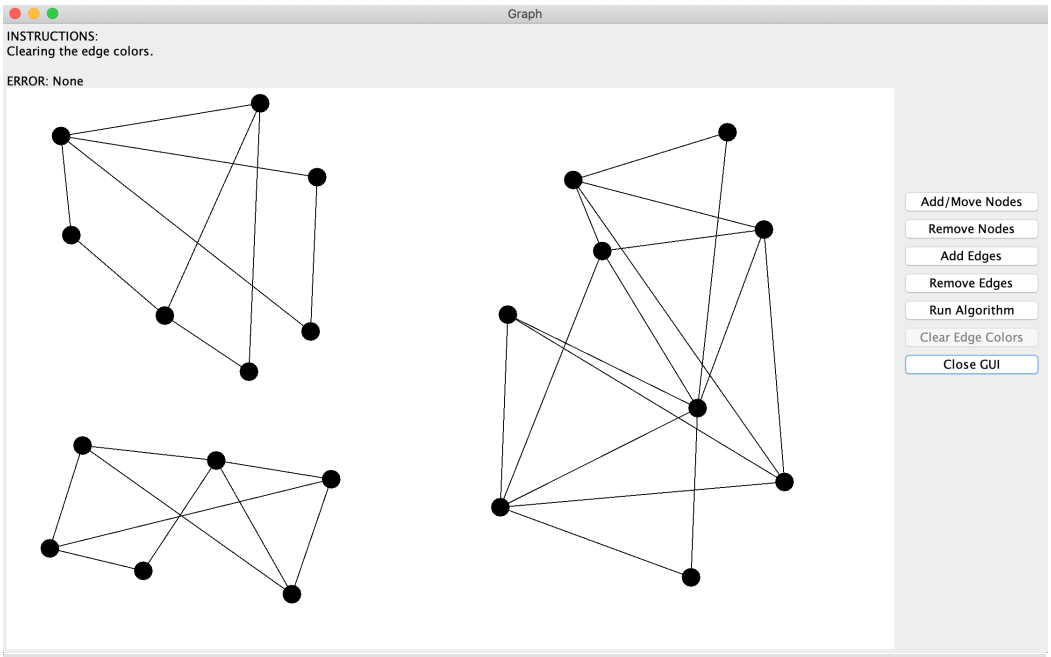


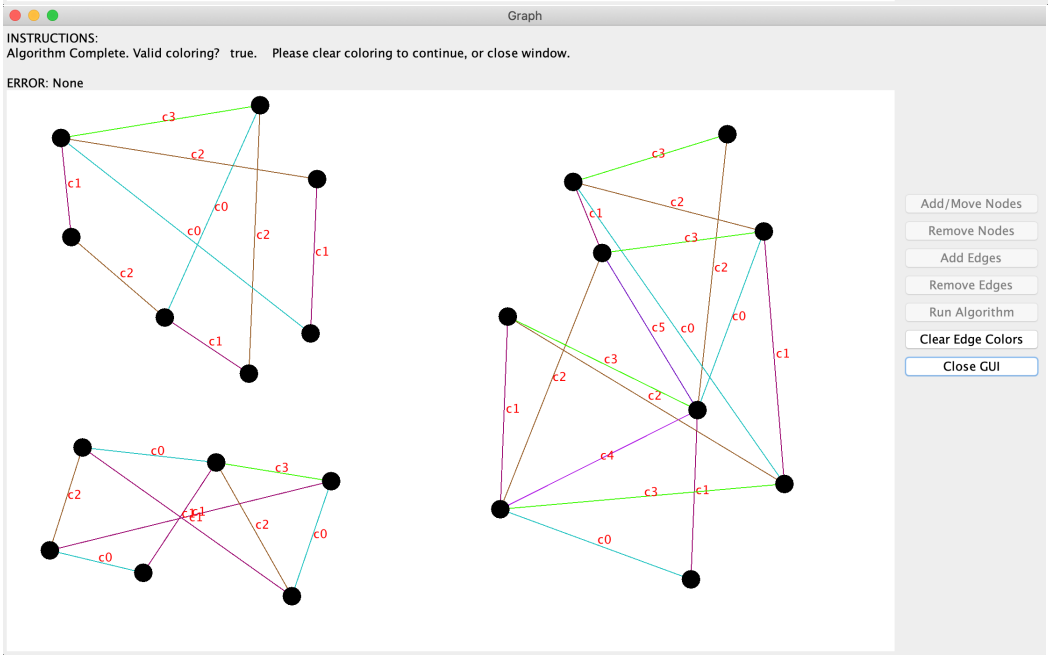
Experiment #4: A disconnected graph before and after coloring. The algorithm produced a valid coloring.



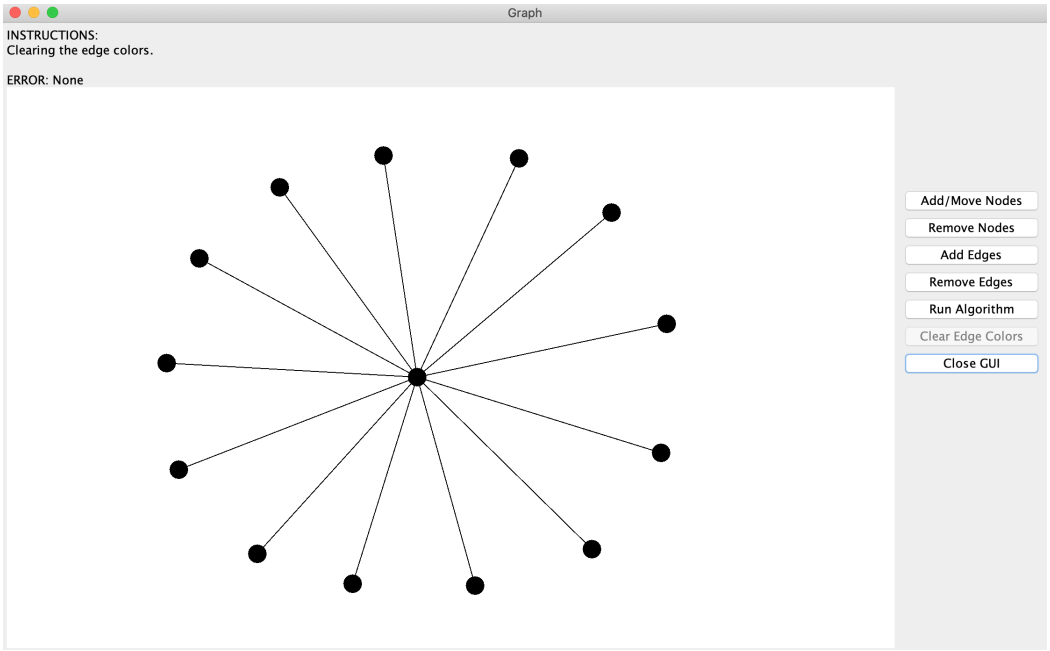



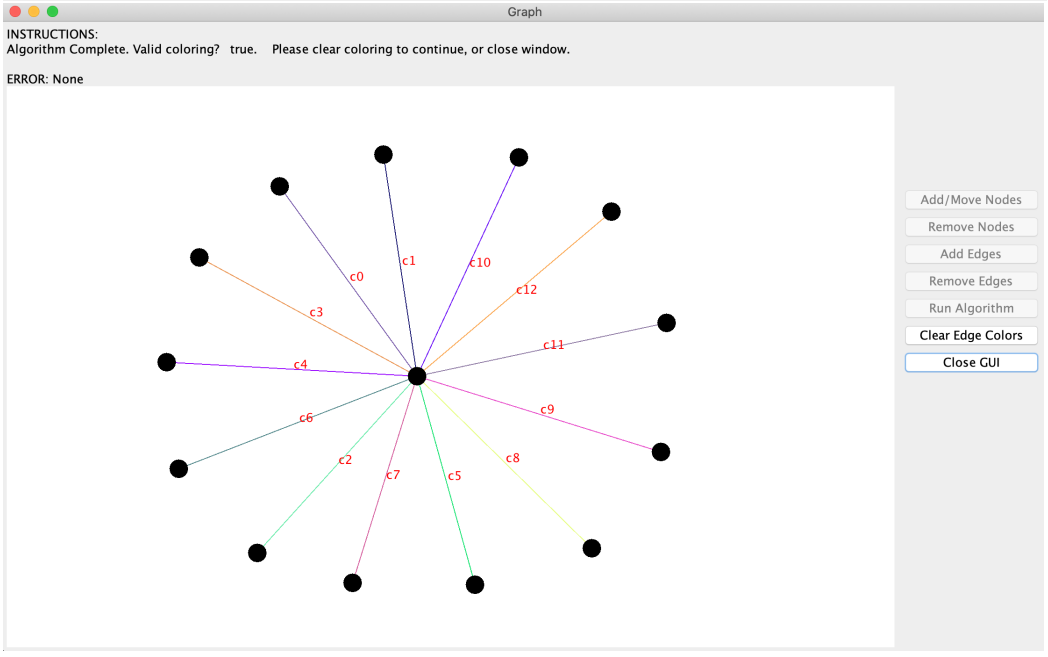
Experiment #5: A more complex disconnected graph before and after coloring. The algorithm produced a valid coloring.



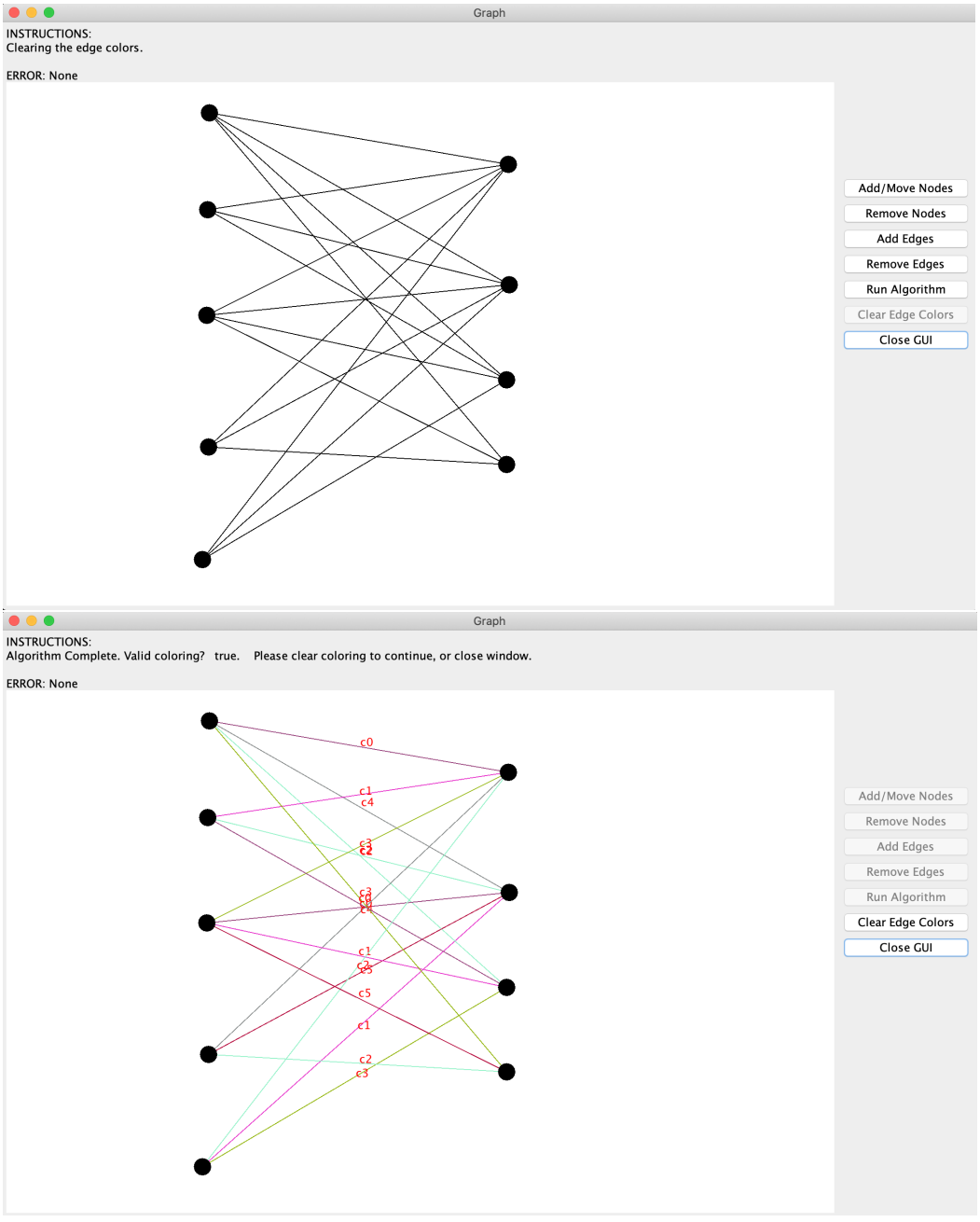



Experiment #6: A star graph before and after coloring. The algorithm produced a valid coloring.

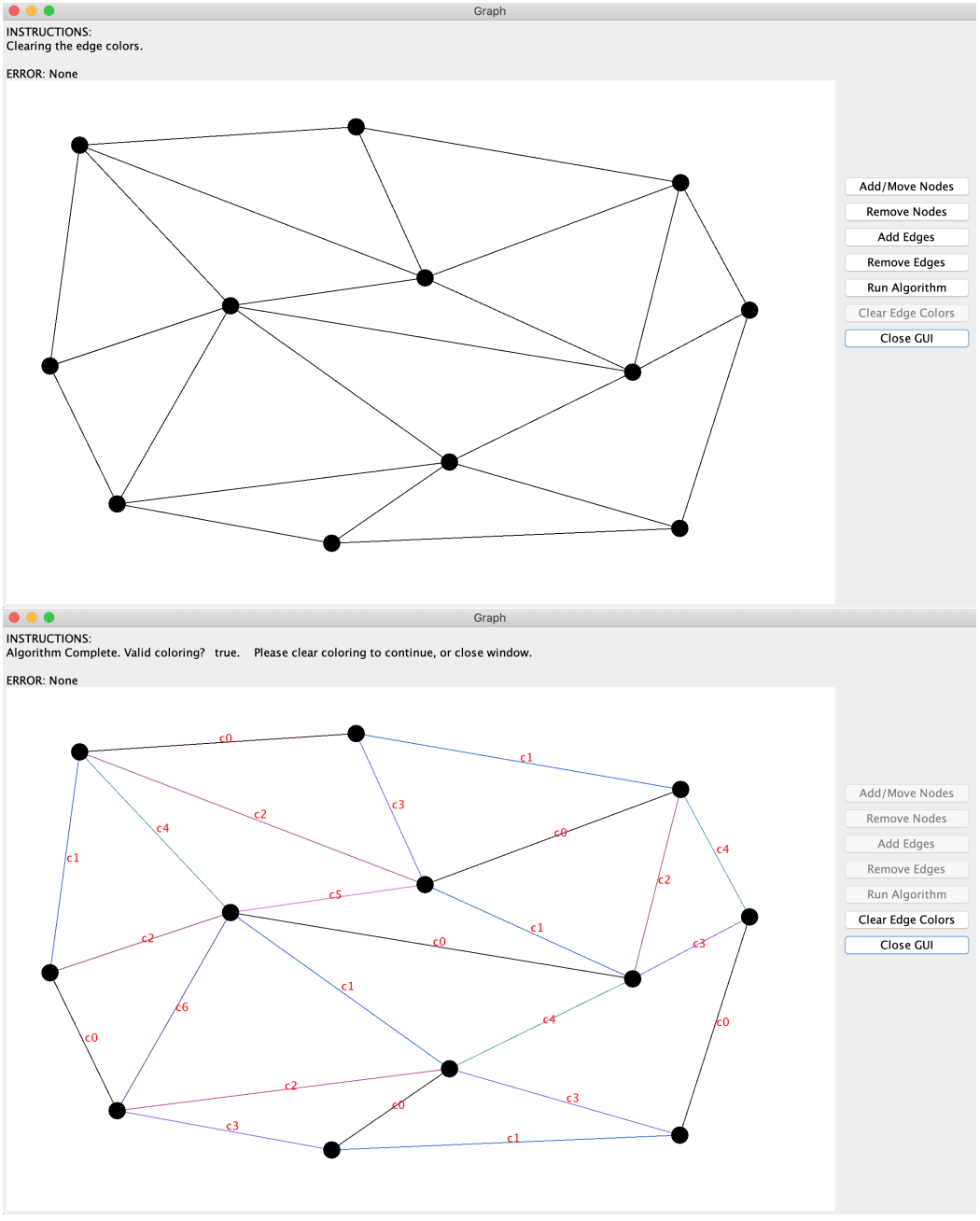


Experiment #7: A bipartite graph before and after coloring. The algorithm produced a valid coloring.



Experiment #8: A planar graph before and after coloring. The algorithm produced a valid coloring.



Sources:

- Wikipedia: https://en.m.wikipedia.org/wiki/Misra_%26_Gries_edge_coloring_algorithm
- Original paper published by Elsevier: https://www.sciencedirect.com/science/article/pii/002001909290041S
- Original paper on Jayadev Misra's UT Austin webpage: https://www.cs.utexas.edu/users/misra/psp.dir/vizing.pdf