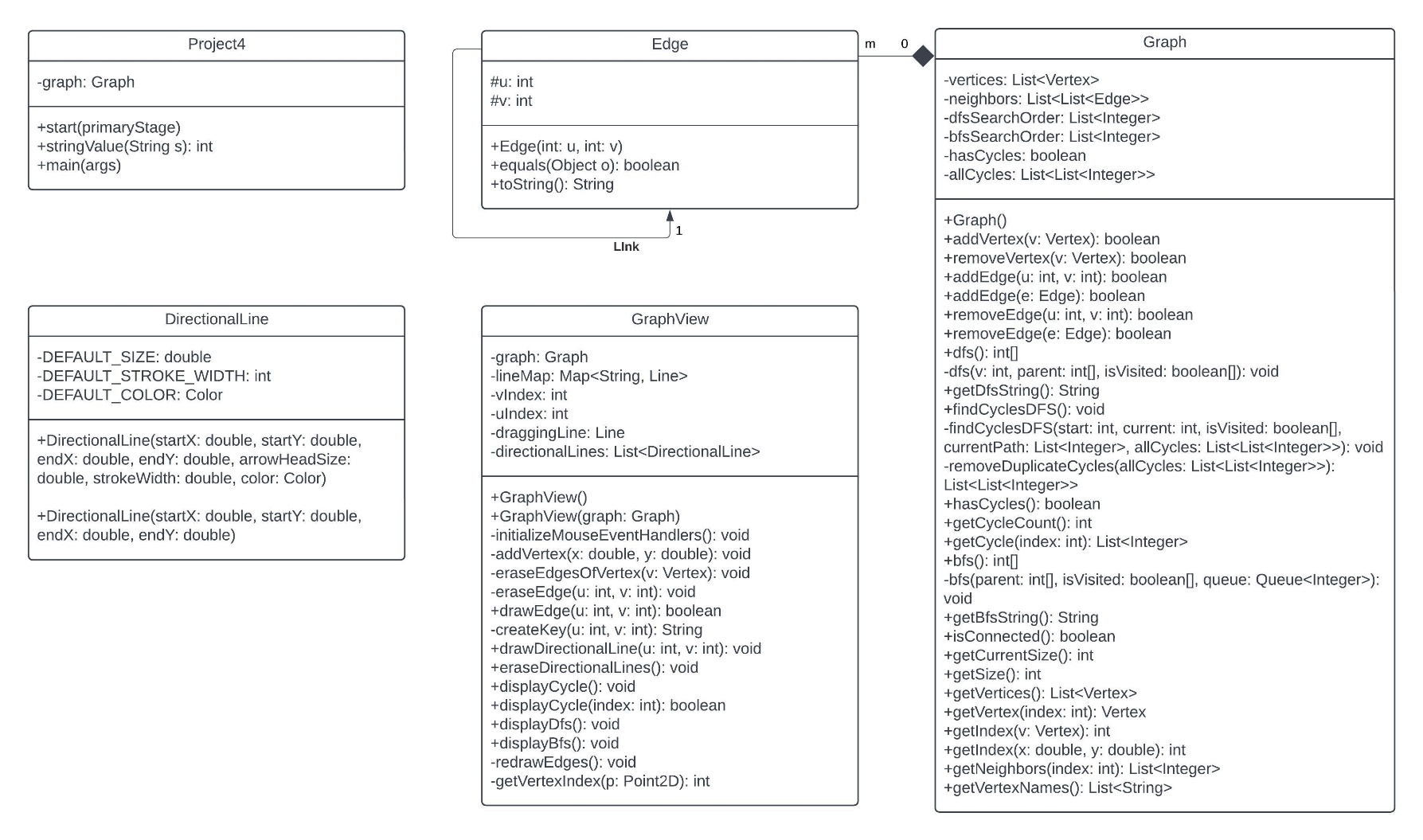
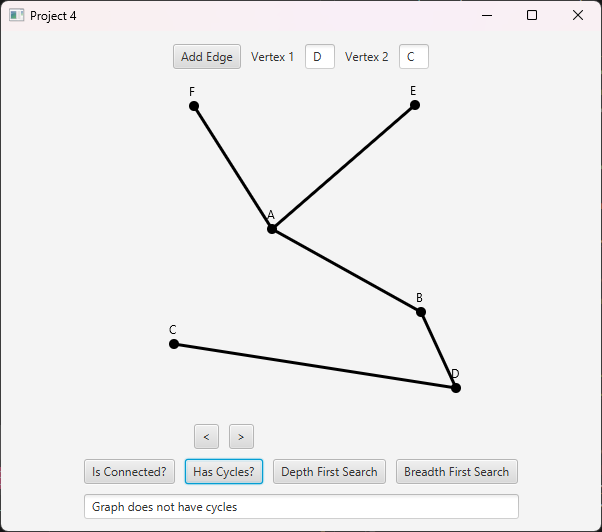
CMSC 315

Programming Project 4 – Graphs

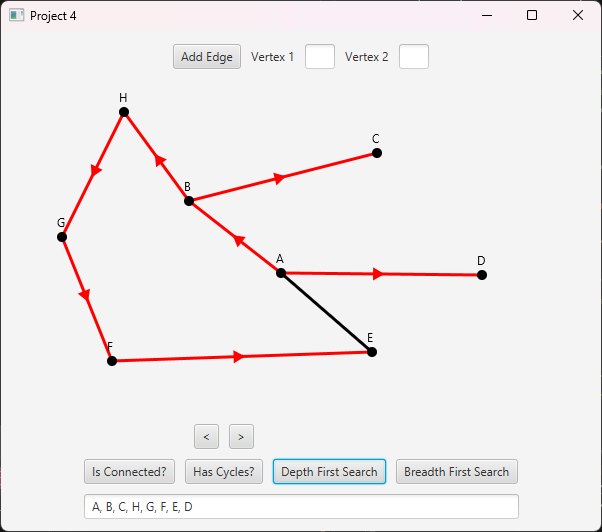
Daniel Smolsky  
  
**UML Diagram:**

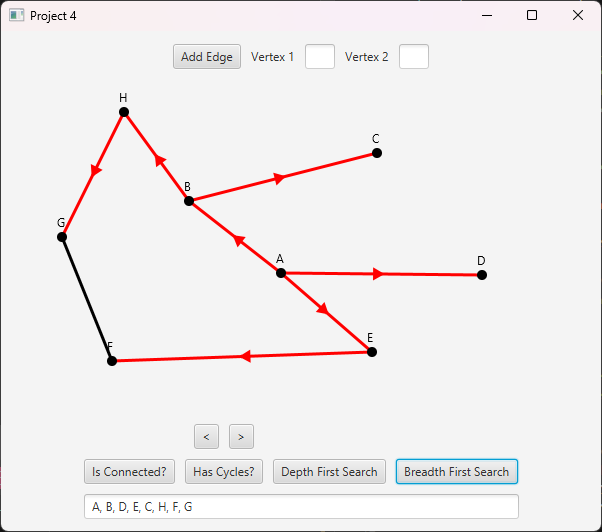
**Test Plan**

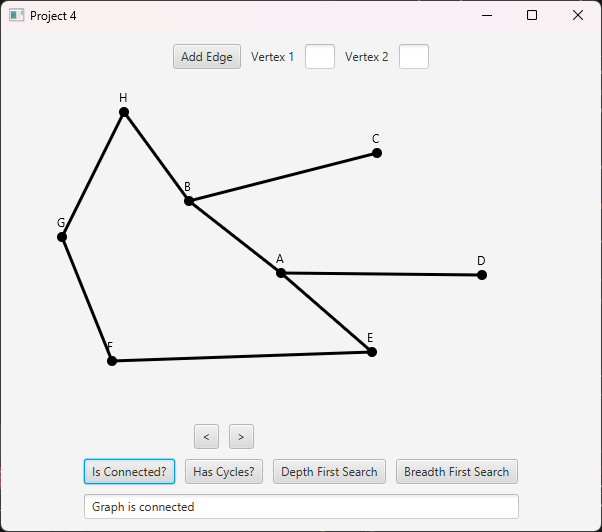
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test # | Purpose | Pos/Neg Test | Input Values | Expected Result | Pass/Fail |
| 1 | Check program matches example in project outline | Pos | Graph with 6 points | Graph does not have cycles | Pass |
| 2 | Depth first search displays correct string representation of the search and shows the path of the search | Pos | Graph with multiple points to demonstrate DFS | A, B, C, H, G, F, E, D | Pass |
| 3 | Breadth first search displays correct string representation of the search and shows the path of the search | Pos | Graph with multiple points to demonstrate BFS | A, B, D, E, C, H, F, G | Pass |
| 4 | Is Connected button returns whether or not the graph is connected | Pos | Graph with multiple connected points | Graph is connected | Pass |
| 5 | Can add vertices and edges by left clicking the pane or left click dragging from one vertex to another | Pos | Two vertices connected by dragging the mouse | Vertices are connected and edges are created | Pass |
| 6 | Can cycle through all of the possible cycle visualizations | Pos | Graph with multiple cycles | Visual representation of cycle is shown, along with the cycle # out of total available | Pass |

Test plan 1:  


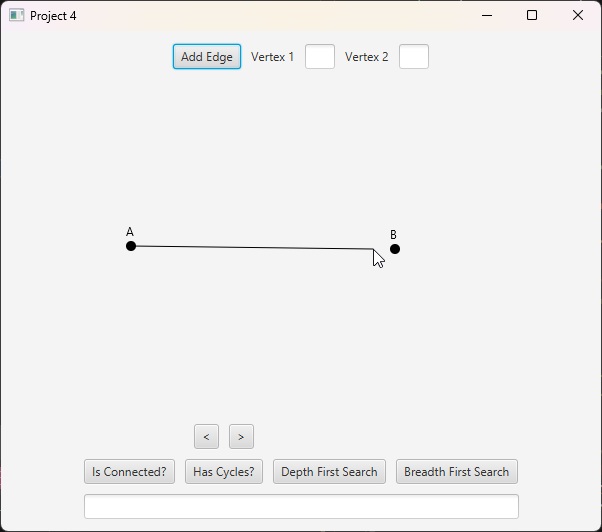
Test plan 2:



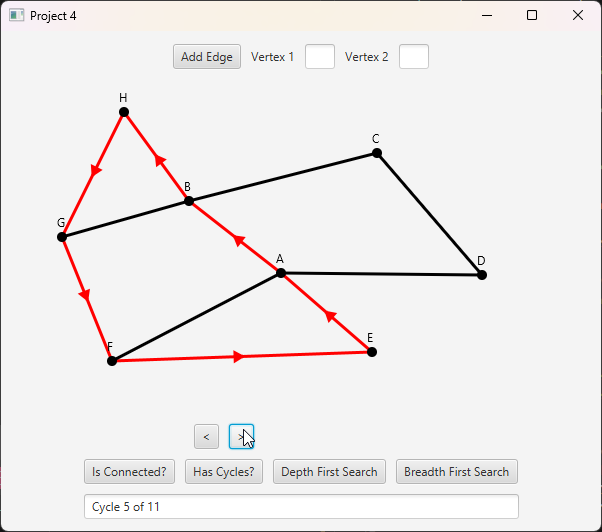
Test plan 3:  


Test plan 4:  


Test plan 5:



Test plan 6:



**Approach and Lessons Learned:**

Besides the base requirements for the project, I decided to implement a couple more features when approaching the design. I wanted to be able to remove vertices and edges by right clicking and have the resulting graph adjust to the changes, deleting any connected nodes along with them. Additionally, I wanted to be able to display the DFS and BFS paths, as well as display all of the possible cycles within the graph. In order to do this, I had to create my own DirectionalLine class, as I couldn’t find anything within the javafx class that already had this functionality. To create lists of cycles within the graph, I looked into something called Johnson’s algorithm, which is the most efficient way to implement this feature. Unfortunately, it was really complicated so I found a simpler, less efficient way that works well as long as the graph has less than 50,000 cycles, at which point it starts to run slow.

I learned a lot of new ideas during this project’s implementation. The edges within the GraphView class are stored in a map for efficient lookup and quick deletion, implementing this along with a method to create unique keys for each edge was new to me. I also had to brush up on some trigonometry to create the DirectionalLine class, specifically in the creation of arrows pointing in the direction of the line and in the right positions along the line.