CMSC 315 - Graph Visualizer (Project 4) Documentation

Laird, Brendan M.

University of Maryland - Global Campus (UMGC)

CMSC 315: Data Structures & Analysis

Professor Ioan Salomie

August 3, 2025

Discussion of the Approach

For Project 4, I adopted a bottom-up, iterative coding strategy combined with frequent testing. I began with defining the core data structures–the Graph class–to encapsulate essential graph operations, including adding and removing vertices, adding edges, and implementing primary graph algorithms (DFS, BFS, connectedness, and cycle detection). Starting with these fundamental methods allowed me to clearly visualize the data flow and logic necessary for a robust implementation.

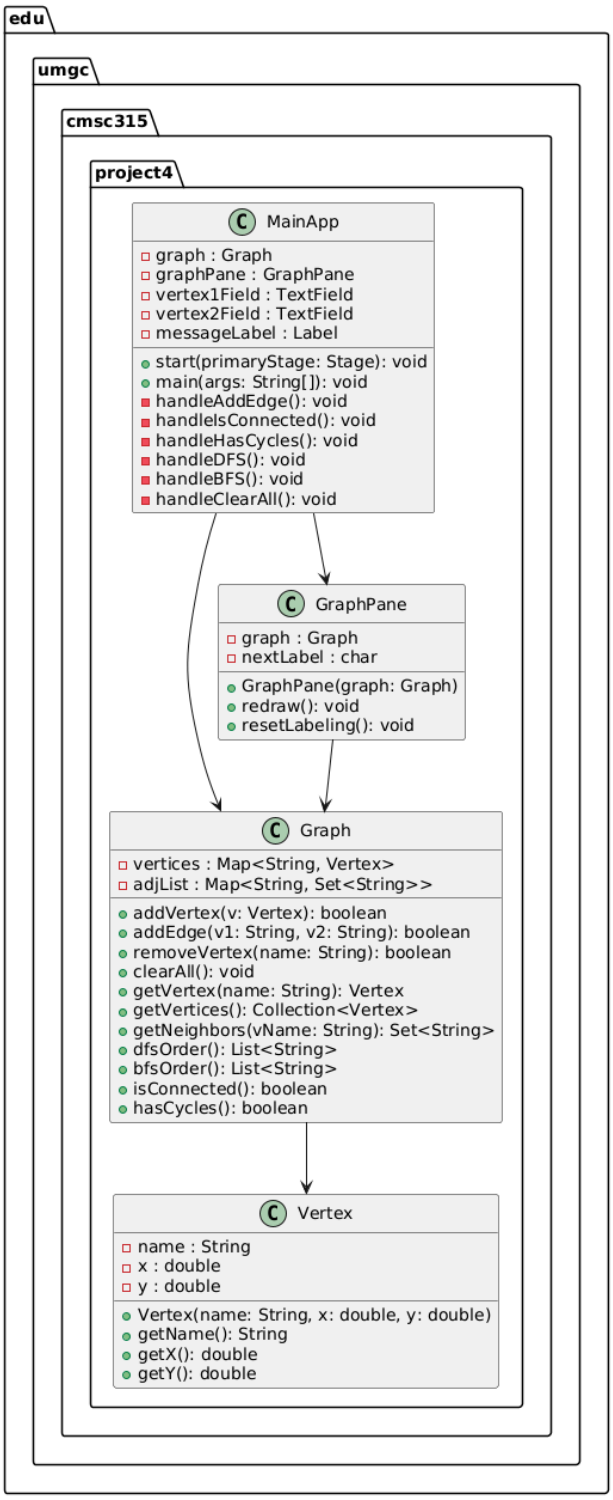
Once the underlying graph logic was functional, I incrementally developed the interactive graphical interface using JavaFX. This component, GraphPane, handled the visual representation of vertices and right click for removal–to align with standard GUI conventions, thus ensuring user-friendliness. The immediate visual feedback during these interactions significantly streamlined my debugging process.

During implementation, I identified a limitation in the initial DFS and BFS algorithms: both were rigidly hardcoded to start from vertex ‘A’. Recognizing this as a potential usability issue-particularly when ‘A’ was removed or never created–I adjusted the logic to dynamically select a suitable starting vertex. The algorithm now intelligently selects ‘A’ if available, or defaults to the alphabetically first vertex present. This enhancement maintained backward compatibility while significantly improving robustness and user experience.

I further reinforced usability by adding a clearly labeled “Clear All” button in the GUI, which resets the graph state and vertex labeling entirely. This feature proved invaluable during testing, especially when evaluating edge cases and performing repetitive test scenarios.

Overall, this iterative, bottom-up approach–with continual testing, incremental enhancements, and deliberate usability refinements–enabled the construction of a stable, intuitive, and fully functional graph visualization and analysis application.

UML Class Diagram



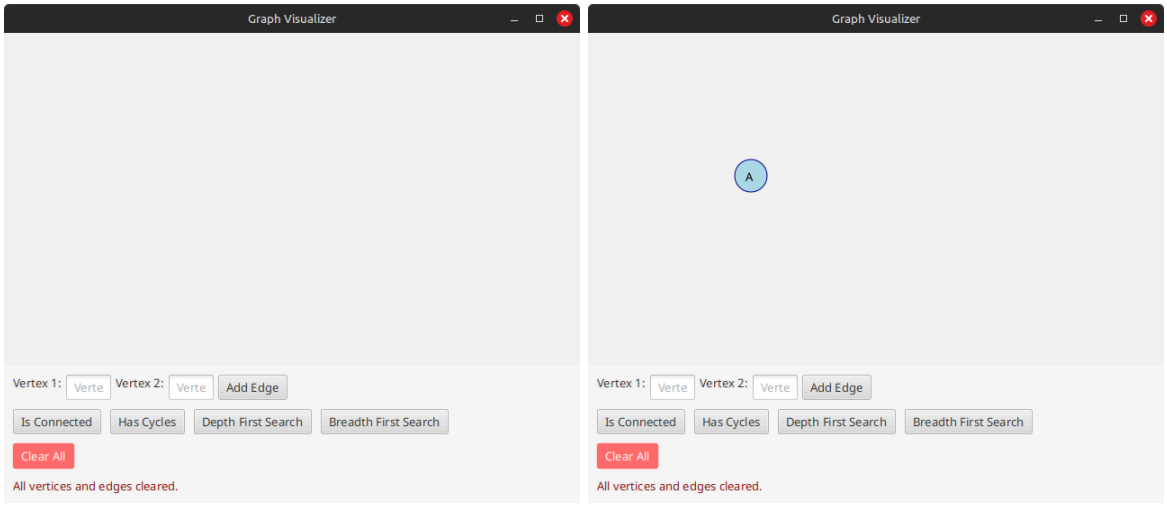
Test Plan

Test Case	**Description**	**Input**	**Expected Output**	**Pass/Fail**
TC01 - Add Vertex	Add a vertex by left clicking the pane	Left-click empty area	Vertex appears, labeled ‘A’, ‘B’	Pass
TC02 Add Edge	Add edge between two vertices	Enter ‘A’, ‘B’, click “Add Edge”	Edge Appears between A and B	Pass
TC03 - Add Invalid Edge	Try edge with a non-existent vertex	Enter ‘A’, ‘Z’, click “Add Edge”	Error, no edge added	Pass
TC04 - Remove Vertex	Remove a vertex by right-clicking it	Right-click vertex ‘A’	Vertex ‘A’ and its edges removed	Pass
TC05 - Remove Non Existent Vertex	Right-click on empty space	Right click any not on a vertex	No effect	Pass
TC06 - Self Loop	Try to add self-loop	Enter ‘A’, ‘A’, click ‘Add Edge’	Error, no edge added.	

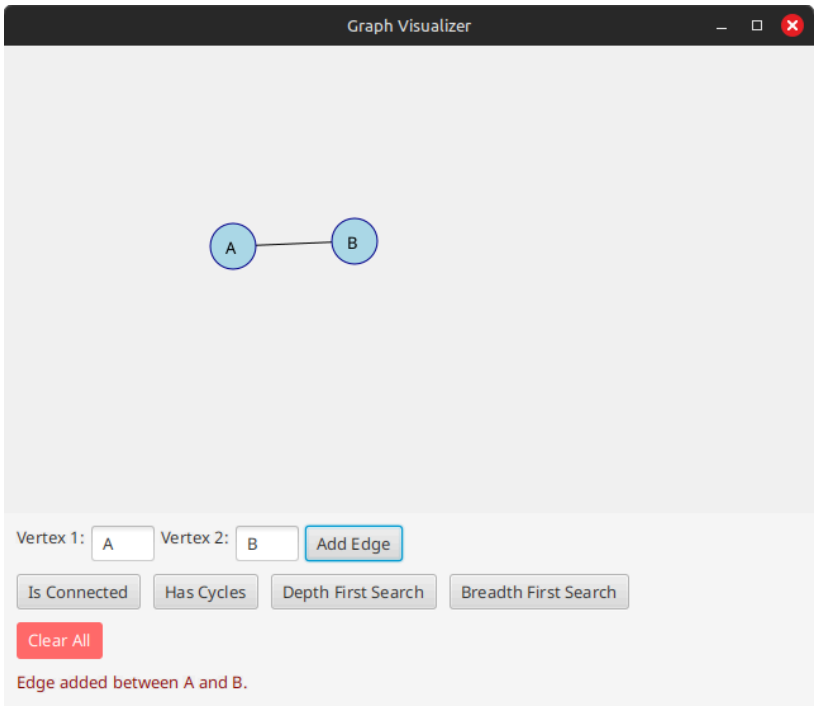
TC07 - Clear All	Clear all vertices and edges	Click “Clear All”	All graph data erased, labeling reset to ‘A’	
TC08 - Is Connected (Positive)	Check single connected component	Build connected graph, click “Is connected”	Message: “The graph is connected”	
TC09 - Is Connected (Negative)	Disconnected graph	Two unconnected sets, click “Is connected	Message: “The graph is NOT connected.”	
TC-10 Has cycles	Detect a cycle	Make a triangle, click “Has Cycles”	Message: “The graph has cycles.”	
TC-11 Has Cycle (No Cycle)	Tree/no cycles	Make a tree, click “Has Cycles”	Message: “The graph has NO cycles.”	
TC12 - DFS w/ A	DFS with A present	Vertices A, B, C, (A connected), click “Depth first Search”	Message: “DFS Order starting from A): …”	

TC13 - DFS w/o A	DFS after removing A	Remove, click “Depth First Search”	Message: “DFS Order (starting from B): …”	
TC14 BFS w/ A	BFS present with A	Vertices A, B, C, click “Breadth First Search”	Message: “BFS Order (starting from A): …”	
TC15 BFS w/o A	BFS after removing A	Remove A, click “Breadth First Search”	Message: “BFS Order (starting from B): …”	
TC16 - Max Vertices	Add More than 26 vertices	Left-click > 26 times	Only vertices A-Z appear, no more added	

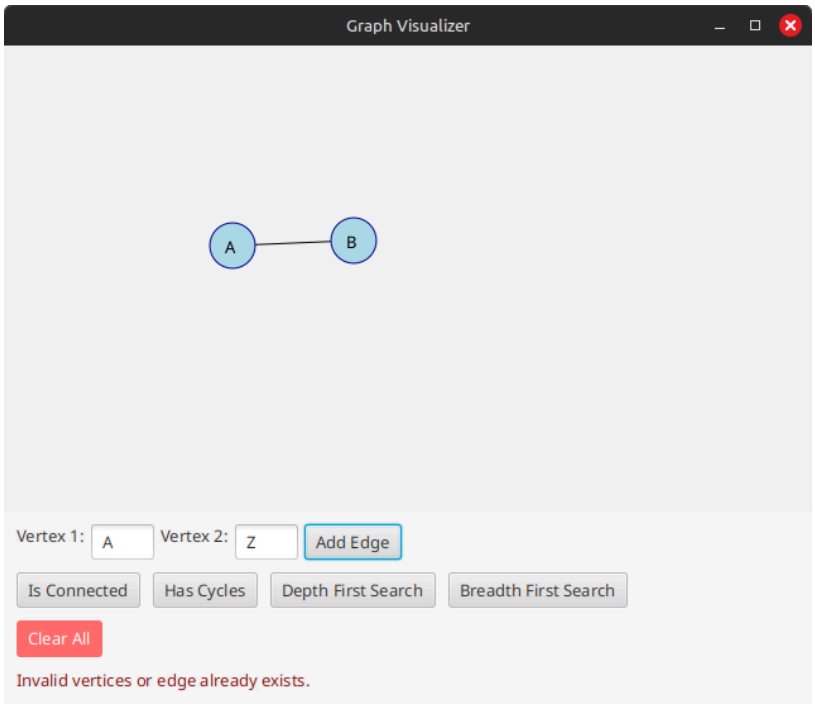
TC01 - Add Vertex



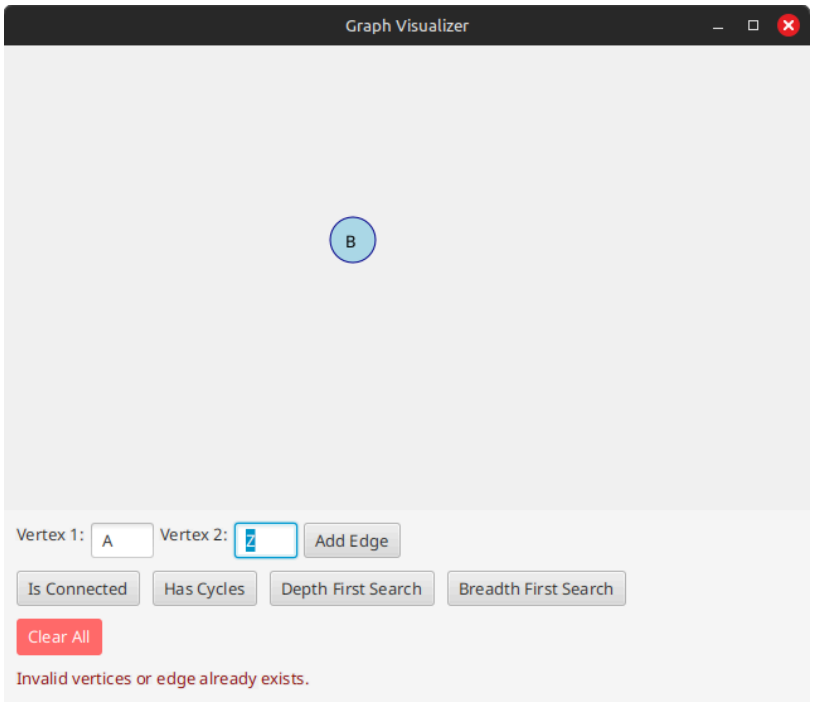
TC02 - Add Edge



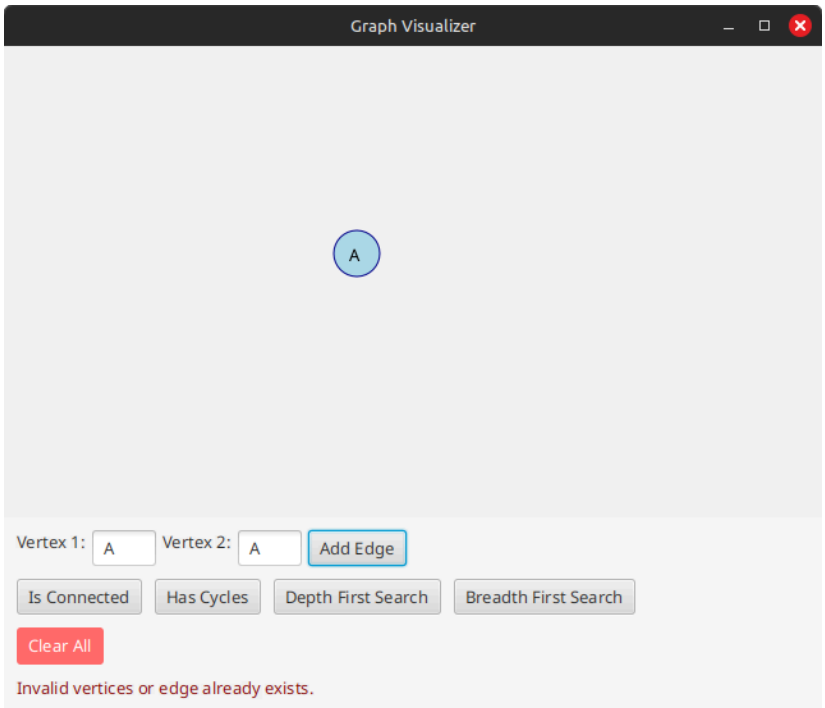
TC03 - Add Edge (Invalid)



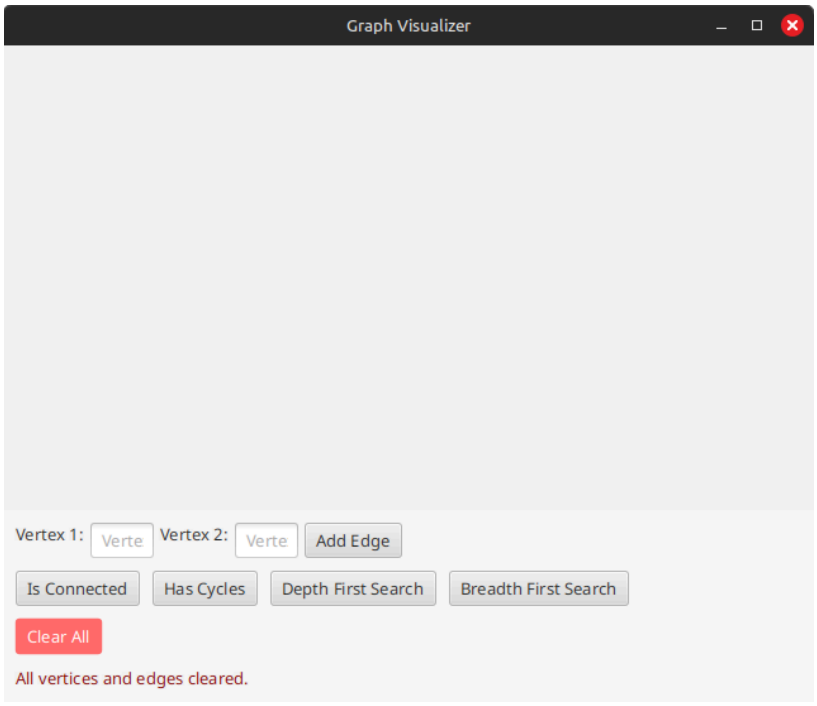
TC04 - Remove Vertex



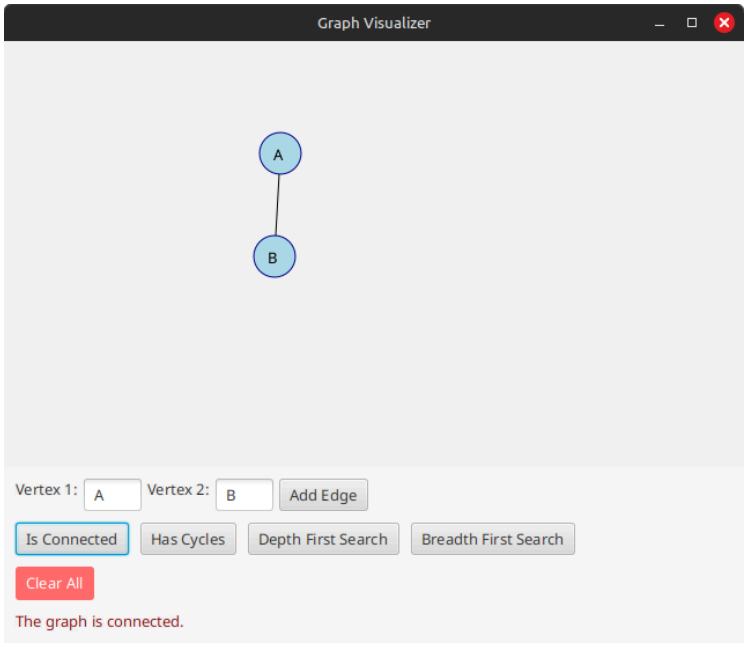
TC06 - Self Loop



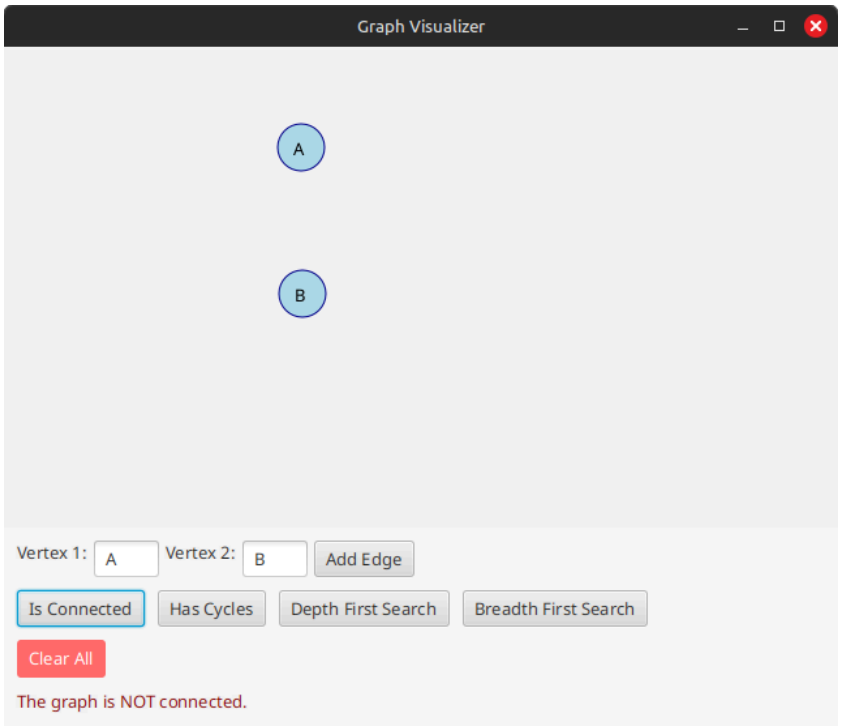
TC07 - Clear All



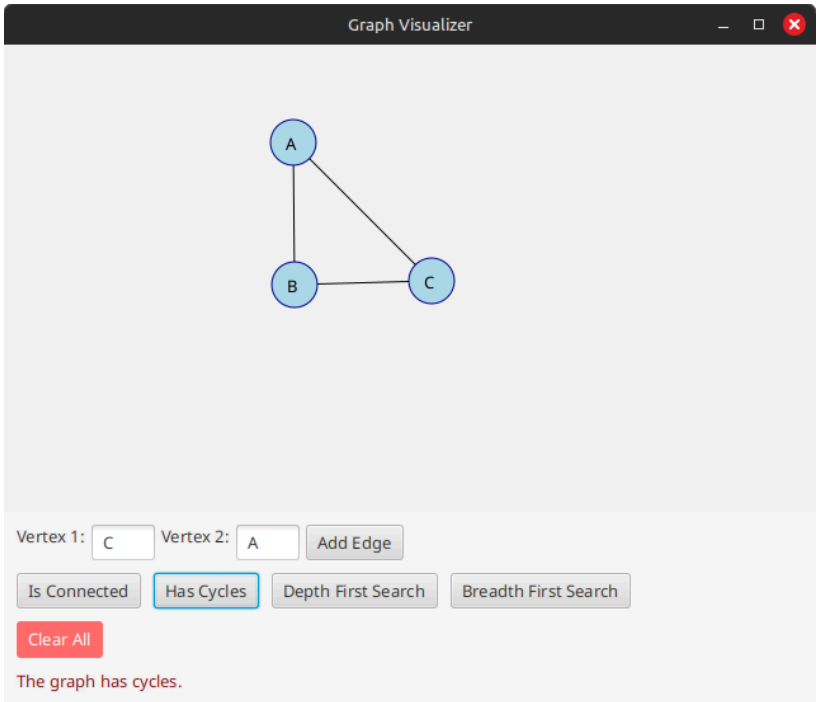
TC08 - Is Connected (Positive)



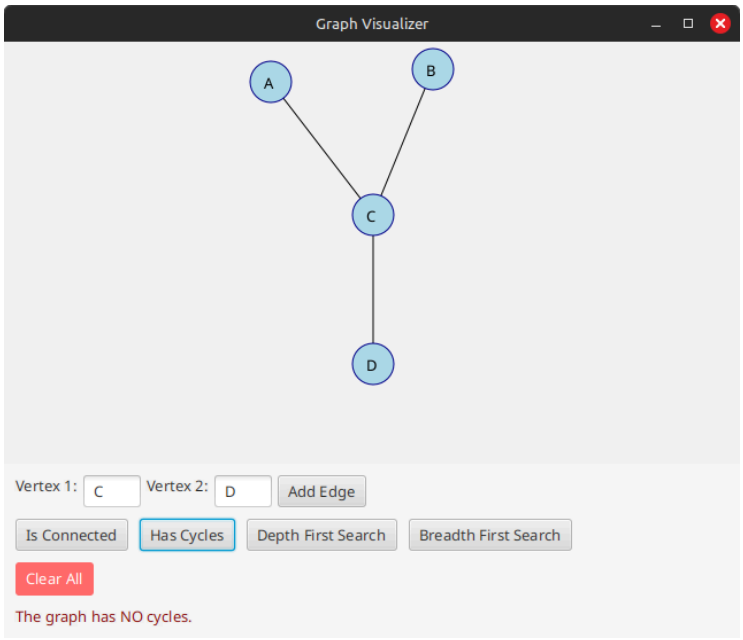
TC09 - Is Connected (Negative)



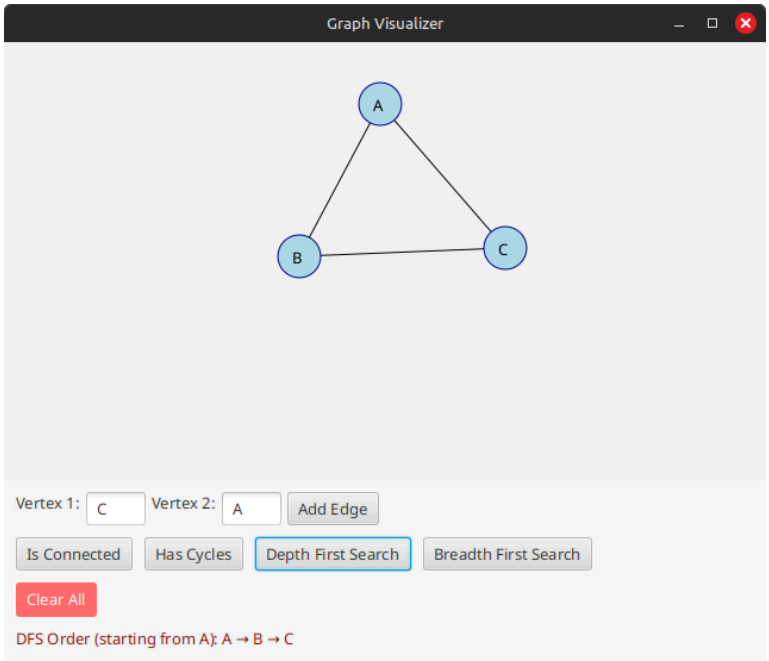
TC10 - Has Cycles (Cycle)



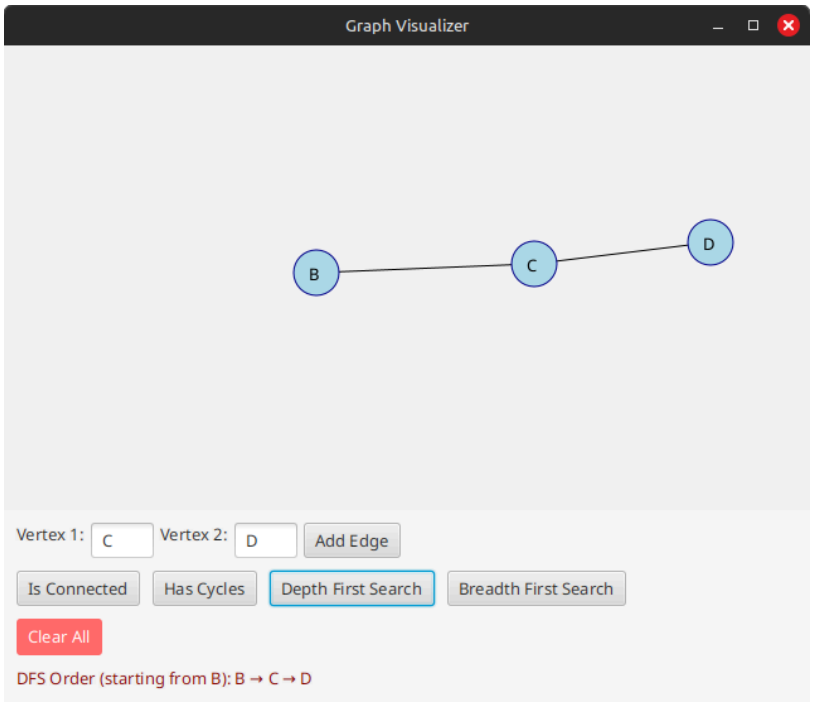
TC11 - Has Cycles (No Cycles)



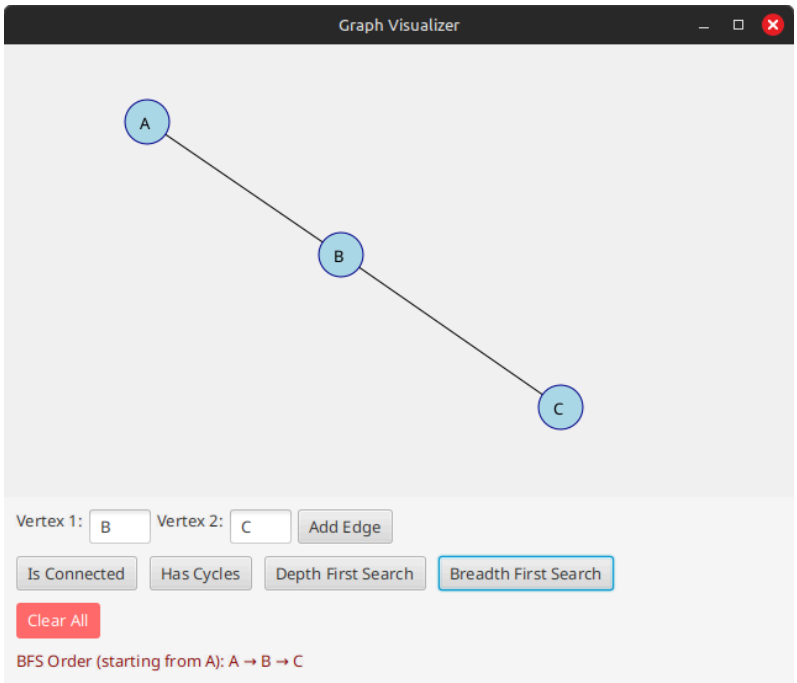
TC12 - DFS with A



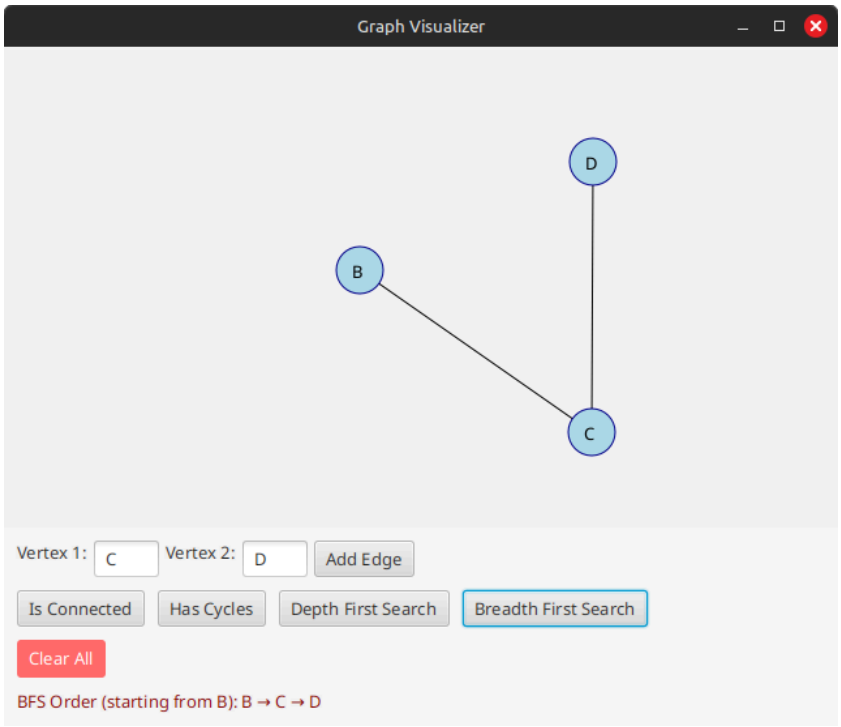
TC13 - DFS without A



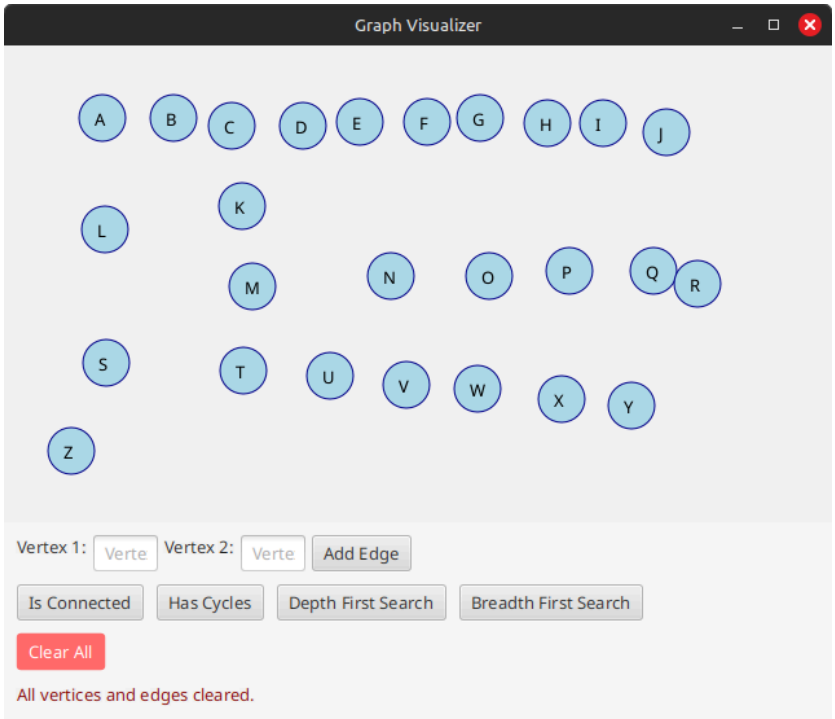
TC14 - BFS with A



TC15 - BFS without A



TC16 - Max Vertices



Lessons Learned

Through completing this project, I deepened my understanding of graph data structures and algorithms, particularly how critical the choice of data structure representation is for clear, maintainable code. Implementing the adjacency-list representation enhanced my ability to visualize relationships between vertices and simplified the implementation of essential algorithms such as depth-first and breadth-first searches.

One key insight was recognizing the importance of flexibility in algorithm design. Initially, my DFS and BFS implementations were limited by assuming a fixed starting vertex. By shifting toward a more dynamic approach—selecting an optimal starting vertex based on existing graph elements—I significantly improved the algorithms' resilience and user experience.

I also learned the value of intuitive GUI interactions. Allowing vertices to be added or removed directly through mouse clicks rather than solely via text input fields dramatically improved usability and reduced cognitive load during testing and demonstration. Additionally, providing immediate visual feedback made debugging far more efficient and engaging.

Lastly, maintaining backward compatibility while introducing improvements proved essential for smooth iterative development. This balance of enhancing functionality without disrupting existing behavior sharpened my software engineering skills and reinforced the practice of incremental, tested code changes.