Graph Theorist's Sketchpad App

Samuel Gibson

Contents

IGraphObject class

Description:

Interface class for creating graph objects (vertex, edge). Mostly contains functions regarding the visualization and interaction of the object.

Functions:

- `GetPath()` : returns a GraphicsPath for the object. A GraphicsPath is a representation of a series of lines and curves. In the context of this app, it will be the path for a vertex or edge.
- `HitTest(Point clickPos)` : returns whether the user successfully clicked on the graph object on the GUI.
- `Draw(Graphics g, Pen pen)` : draws the object onto the passed Graphics object with a specific pen. The Graphics object is a drawing surface where vertices and edges are displayed

Vertex class

Description:

Extends IGraphObject. Represents a vertex of a graph and contains useful information and functions for vertex manipulation. Getters and setters are present for the relevant fields.

Fields:

- `String id` : identifier of the vertex, ex. “V1”.
- `Color color` : the color of the vertex.
- `Bool selected` : whether this vertex is selected by the user.
- `Point point` : the coordinates of the vertex.
- `Int radius` : radius of the vertex.
- `Int parallelEdgeCount` : the number of parallel edges the vertex has.
- `List<Edge> connectedEdges` : the edges connected to the vertex.
- `List<Vertex> neighbors` : the vertices adjacent to this vertex.

Constructor:

Takes a string id, a color, a point, and a radius to create a new vertex with those parameters. By default, selected is set to false, parallelEdgeCount is set to 0, and the list of edges and vertices are initialized. When the point is passed, it is offset by the radius to ensure the vertex is placed in the center of the coordinates.

Functions:

- `IncrementParallelEdges()` : increments the parallelEdgeCount by 1.
- `Draw(Graphics g, Pen pen)` : implemented member from IGraphicsObject. Using the GetPath() method, first fills a path of the vertex, then draws another path serving as the vertex outline.
- `GetPath()` : implemented member from IGraphicsObject. Creates a new GraphicsPath object and adds an ellipse to the path with the vertex parameters. Returns the created GraphicsPath.

- `HitTest()` : implemented member from IGraphicsObject. Detects whether the user clicked on the GraphicsPath for the vertex, returning true or false.
- `Move(Point target)` : Moves the vertex from its old point to the new target point. Sets the vertex's current point to the new one.

Edge Class

Description:

Extends IGraphObject. Represents an edge of a graph and contains useful information and functions for edge manipulation. Getters and setters are present for the relevant fields.

Fields:

- `Vertex v1` : the first vertex point for the edge.
- `Vertex v2` : the second vertex point for the edge.
- `Color color` : the color for the edge.
- `Bool selected` : whether this edge is selected by the user.
- `Bool isLoop` : whether this edge is a loop.
- `Int width` : the width of the edge.
- `Int parallelEdge` : the parallelEdge number between the two vertices.

Constructor:

Takes a first vertex, a second vertex, a color, and a width to create a new edge object.

The isLoop field is determined through comparing the first and second vertex and checking if they are the same. The parallelEdge number can optionally be set, or it will be 0 by default.

Functions:

- `Draw(Graphics g, Pen pen)` : Implemented member of IGraphObject. Uses the GetPath() method to draw the corresponding path onto the graphics object with given pen.
- `GetPath()` : Implemented member of IGraphObject. Returns a new GraphicsPath object. First determines the offset to begin drawing from based off the first vertex's radius. This offset is necessary for the edge to be properly drawn from the center of the vertex. Next, the parallelOffset is determined by the DetermineParallelOffsetAmount() function. This is used for parallel edges. Finally, the BuildPath() method is used to return a new GraphicsPath based on the coordinates of the two vertices, the offset, and the parallel offset.
- `HitTest(Point clickPos)` : implemented member of IGraphObject. Returns whether the user clicked on the edge. Uses the GetPath() function and checks if the coordinates of the user's click were on top of the edge.
- `DetermineParallelEdgeOffsetAmount()` : returns the parallel edge offset, necessary for properly drawing parallel edges so they are not on top of each other. Uses the parallelEdge field to determine which parallel edge the edge is. Depending on that number, returns an increasing offset amount.
- `BuildPath(GraphicsPath path, int x1, int y1, int x2, int y2, int xOffset, int yOffset)` : returns the GraphicsPath path with the edge shape drawn on it. First checks if the

edge is a loop, and if so, adds an arc to the graphics path. Otherwise, adds a line to the GraphicsPath path according to the coordinates, and offsets.

Graph Class:

Description:

Stores the edges and vertices of the graph as well as providing statistics and manipulations of the graph. Depends upon the Edge and Vertex class. Contains getters and setters for appropriate fields.

Fields:

- `List<Vertex> vertices` : list of all vertices in the graph.
- `List<Edge> edges` : list of all edges in the graph.
- `Vertex selectedVertex` : the current selected vertex.
- `List<Edge> selectedEdges` : list of all currently selected edges.
- `Bool hasSelectedVertex` : whether the graph has a selected vertex.

Constructor:

The graph constructor takes no parameters and sets up initial values. Vertices, edges, and selected edges are initialized as new lists and hasSelectedVertex is set to false.

Functions:

- `UpdateSelectedVertex(Vertex v, bool value)` : updates the vertex with it being selected or not. If there is not already a selected vertex, the passed vertex is set as the selected vertex if true is passed. If false is passed, the passed vertex is deselected.
- `AddSelectedEdge(Edge e)` : adds a new selected edge to the selected edges list.
- `DeselectEdge(Edge e)` : removes an edge from the selected edges list.
- `AddVertex(Vertex v)` : adds the vertex v to the graph's vertices list.
- `AddEdge(Vertex from, Vertex to, Color color, int width)` : Adds a new edge to the graph. First check if the edge being added is a parallel edge, and if so, increments the vertex's parallel edge count by calling IncrementParallelEdges() on both to and from vertex. In either case, the from and to vertex's neighbors are updated to account for the edge between them, and the edge is added to the graph's edges list and the vertex's edge list.
- `CheckBipartite()` : returns whether the graph is bipartite. First, returns false if the HasLoop() method returns true. Bipartite graphs cannot have loops. Then, for each vertex in the graph's vertex list, the color is set to black. Then a bool array is used to keep track of which vertex has been visited and a while loop is used for calling PaintGraph() recursively. The condition for the while loop is whether all vertices have been visited yet by utilizing the CheckIfVisisted() method.
- `CheckIfVisisted(bool[] visited)` : checks the bool array and returns the index for the next unvisited vertex. If all vertices have been visited, returns -1.
- `PaintGraph(int start, Color color, bool[] visited)` : Depth first method of painting each graph vertex. Recursively visits each vertex while attempting to paint connecting ones alternating colors between red and blue. A graph is bipartite if it's nodes can alternatively be painted red or blue. Returns true if painting was successful, otherwise false if an attempt to paint a vertex's neighbors the same color as itself was made.

- `HasLoop()` : returns whether the graph contains a loop. Checks each edge in the graph and if the edge's isLoop value is true, returns true, otherwise false.
- `GetMatrix()` : returns an adjacency matrix of the graph. Goes through each vertex in the graph and constructs a 2D array containing the adjacency values for each vertex. If vertices are neighbors, a 1 is set, otherwise 0.
- `CountComponents()` : returns the number of graph components. Utilizes a bool array to keep track of visited vertices. Similar structure to CheckBipartite(). Calls the recursive SearchGraph() function in a while loop until all vertices have been visited. Each time SearchGraph() finishes, the count is increased by 1, indicating a component was visited.
- `SearchGraph(int start, bool[] visited)` : conducts a depth first search of the graph, finishing when a component is found. Sets the current vertex as visited in the bool array, then begins iterating through the vertex's neighbors. Each neighbor is checked and if it has not been visited yet, SearchGraph() is recursively called with the index of that neighbor. Each neighbor is visited, and its neighbors also visited until finally there are no more neighbors to visit. Once the search is complete, a component has been found.

SketchPad Class

Description:

Contains a graph object and handles user input from the GUI. Extends the Form class

Fields:

- `Graph graph` : the graph object for the application.
- `Int vertex_count` : the current number of drawn vertices.
- `Const int VERTEX_RADIUS` : default vertex radius.
- `Const int EDGE_WIDTH` : default edge width.
- `Bool moveState` : whether a vertex is being moved or not.
- Bool deleteState : whether the delete button is active.
- Vertex movingVertex : the vertex being moved.
- Point lastVertexLocation : the previous location of the moving vertex.

Constructor:

The sketchpad constructor initializes the WinForms components, creates a new Graph object, and initializes movingVertex, moveState, and deleteState as null and false.

Functions:

- `Panel1_Paint()` : Event for when the graph panel is redrawn. Draws all the current graph objects onto the screen such as vertices and edges
- `Panel1_MouseClick_1()` : Event for when the user clicks on the graph panel. Depending on the selection mode and where the user clicked, the appropriate function from the Graph object is called, such as creating an edge. The graph's information is also updated and displayed on the screen when a new object is created or deleted by calling UpdateGraphInfo().

- `TryGetVertex(Point pCurrent)` : checks each vertex in the graph object and calls it's HitTest() function to check if the location the user clicked had that vertex. Returns the vertex or null if failed.
- `TryGetEdge(Point pCurrent)` : checks each edge in the graph object and call's it's HitTest() function to check if the location the user clicked had that edge. Returns the edge or null if failed.
- `TryDeleteObject(Point point)` : calls TryGetVertex() and TryGetEdge() at the location the user clicked and deletes the object from the graph if it's not null. Appropriate edges are also deleted when deleting a vertex. The graph's information is also updated with UpdateGraphInfo().
- `GraphPanel_MouseDown()` : event for when the user presses down on the mouse over the graph panel. If the left mouse button was pressed down, movingVertex is set to the return value of TryGetVertex() at the location of the click. If movingVertex is not null, the lastVertexLocation is also set from movingVertex. Field moveState is also set to true.
- `GraphPanel_MouseMove()` : event for when the user moves the mouse over the graph panel. If moveState is true and the mouse button being held is the left button, a new point is created where the user stops the mouse movement. This point is then set as lastVertexLocation and movingVertex's Move() function is called with the new point. The graph panel is then redrawn the update the location of the vertex.
- `GraphPanel_MouseUp()` : event for when the user releases the mouse button. If moveState is true and the mouse button released is the left button, moveState is set to false and movingVertex set to null.
- `DeleteButton_Click()` : event for when the user clicks the delete button on the interface. Updates the deleteState and allows deleting graph objects.
- `Paint_button_Click_1()` : event for when the paint button is clicked on the interface. Goes through each selected object in the graph and updates the color with the color currently selected by the user. Afterwards, clears the selected edges of the graph and redraws the graph panel.
- `Select_color_button_Click()` : event for when the user clicks the select color button. Opens a new paint color dialogue and sets the chosen color.
- `UpdateGraphInfo()` : updates all relevant information on the interface about the graph. Based on the number of graphs and vertices in the graph, displays the count for each. If a vertex is selected, displays the degree of that vertex by getting the number of edges connected to that vertex. Calls the graph's CountComponents() method and displays the results. Gets the adjacency matrix from the graph's GetMatrix() method and displays the result. Finally, redraws the graph panel to update the information.
- `Clear_all_button_Click()` : event for when the user clicks the clear all button. Resets the graph's fields and creates a new graph object. Calls UpdateGraphInfo() to refresh the display.
- `Bipartite_test_button_Click()` : event for when the user clicks the bipartite test button. Calls the graph's CheckBipartite() method and displays the results. Redraws the graph panel.

Implemented Features:

1. Graphical display of vertices and edges
2. Input of vertices and edges
3. Able to reposition vertices while maintaining adjacencies
4. Deletion of vertices and edges
5. Parallel edges
6. Loops
7. Information about numbers of vertices and edges
8. Information about degrees of vertices
9. Information about components
10. Show whether a graph is bipartite
11. Adjacency matrix info

How to Use:

Create new vertex: left click on graph panel.

Select object: right click on object.

Create new edge: Select a vertex, left click on the vertex to connect the edge to.

Select color for new object: Click the paint brush button to set the current color for new objects.

Paint existing object: Select the object to paint. Click the paint button to paint it the current color.

Move vertex: click and drag a vertex to a new location.

Create loop: Select a vertex. Left click the same vertex to form a loop.

Add parallel edge: Select a vertex that has an edge. Left click on the same endpoint vertex to create another edge between the two.

View vertex degree: Select a vertex. The degree is displayed on the right denoted deg(V).

Delete object: Click the delete button to toggle deleting. Left click an object to remove it. Click the delete button again to disable deleting.

Check if Bipartite: Click the run test button underneath the Bipartite label. Results are displayed.