Draft Document: HW2 GraphView Requirements Document

Requirements Analysis Document (RAD)

1. Introduction

1.1 Purpose of the system

The proposed application is "GraphView", an application that enables users to draw Planar graphs on a two dimensional canvas. A Graph consists of a set of vertices and edges constructed between the vertices. A graph is considered to be planar if it can be projected into a two-dimensional surface such that no two edges of the graph intersect each other.

The GraphView application enables users to interactively construct planar graphs using a mouse-based GUI. Graphs can be saved to, and loaded from, disk. While the graphs are being constructed, the GUI must present several perspectives on the structure, as will be described in this document.

1.2 Scope of the system

GraphView must enable users to create planar graphs interactively, and, correspondingly, it must prevent the construction of a non-planar graph.

1.3 Objectives and success criteria of the project

The success of the application depends upon meeting the following core set of objectives

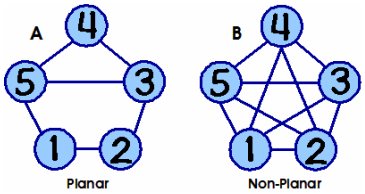
- Construct planar graphs interactively of up to 20 vertices and 50 edges. One must be able to add/remove vertices and add/remove edges.
- Construct planar graph, store it to disk, and load it up again in its identical structure
- Prevent the construction of non-planar graphs
- Show multiple views simultaneously, including GraphView, StructureView, and ValueView.

1.4 Definitions, Acronyms, and abbreviations

Graph	– a structure consisting of a set of *Vertices* and *Edges*
Coordinate	– each vertex is assigned an (x,y) coordinate where (0,0) is the upper left corner, and x increases to the right while y increases downwards.
Vertex	– a fundamental unit of a Graph that contains an (x,y) coordinate. Each vertex is labeled by a unique integer identifier within the graph.

Edge – two vertices v_1 and v_2 can be connected by an undirected edge. The edge (v_1,v_2) can also be described as (v_2,v_1).

Planar Graph – A Graph can be projected into a two-dimensional space using the (x,y) coordinates of its vertices. If no two edges in the graph intersect each other, then the graph is considered to be planar.



GraphView – A two-dimensional graph representation of a Graph, showing vertices as small circles, and edges between vertices as lines.

StructureView – A tabular view of a Graph which produces a report where each row represents a vertex of the graph and the vertices to which it is connected.

StructureView

1	2 5
2	1 3
3	2 4 5
4	3 5
5	1 4

ValueView – A tabular view of a Graph which produces a report where the information for each vertex is shown (i.e., its label and (x,y) coordinate) and the set of known edges is also displayed.

ValueView

1	32 45
2	42 45
3	45 30
4	34 8
5	20 30

Edges
12
13
…

Incident – An edge is incident to vertex v_i if that vertex is one of the vertices that form the edge's endpoints.

2. Current System

None.

3. Proposed System

3.1 Overview

The goal of GraphView is to provide a simple interactive experience for users to create, edit, and save planar graphs. This exercise will reveal a lot about the proper way to design interactive applications and will also involve the appropriate use of abstraction.

3.2 Functional Requirements

The system supports a single actor:

- The *User* launches the GraphView application to construct graphs or load ones already stored on disk.

3.2.1 User

A *User* can:

- Create a new graph
- Save a graph
- Load a graph
- Add a vertex to a graph
- Remove a vertex from a graph
- Connect two vertices in the graph
- Disconnect two vertices in the graph

3.2.2 Optional Capabilities

There are numerous extensions that one might consider for this project. You will not be graded on whether you have satisfied any of these requirements, yet I have provided them because they were in my original vision when I thought of this assignment.

- Enable one to open multiple graphs simultaneously, enabling the user to switch back and forth between each one
- Provide the ability to select a set of vertices and/or edges within the graph
- Provide a rudimentary clipboard capability, enabling user to **copy** a selection of vertices and edges within a graph and **paste** that fragment within the same graph or another graph. Also support **cut**.
- Provide ability to export a graph as an XML file, to enable interoperability with other systems.
- Provide ability to drag an existing vertex to a new location, so long as planarity constraints are not violated.

3.3 Nonfunctional Requirements

3.3.1 Usability

The GraphView application must support mouse-driven interaction as well as keyboard interaction, to broaden the user base. In industry, there is a common understanding that accessibility drives the market (think of how easy the iPod is to use, for example; it was designed to be easy to use). Thus you need to support (at least) the following modalities of interaction. If you wish to provide additional means of interaction, that is your decision. The following must be present:

Action	**Keyboard**	**Using Mouse**
Add vertex n at location (x,y)	Type “add n x y” where (x,y) is the coordinate in the graph and n is the unique label for the new vertex	After selecting the “Add Vertex” tool from command palette, user clicks on a specific (x,y) coordinate

Remove vertex n	Type “remove n”	After selecting the “Remove Vertex” tool from the command palette, user clicks on a specific vertex
Add edge between vertices u and v	Type “connect u v” to connect vertex labeled “u” with the vertex labeled “v”	After selecting the “Connect” tool, user presses the mouse on vertex “u” and then drags and releases over another vertex “v”
Remove edge between u and v	Type “disconnect u v” to remove the edge (if it exists) between the two vertices labeled “u” and “v”	After selecting the “Disconnect” tool, user presses the mouse on vertex “u” and then drags and releases over another vertex “v”

If you feel the Mouse-driven user interface is not sufficient, then your group is going to have to meet with the client (i.e., **me**) and make your pitch as to what interaction you would like to implement.

3.3.2 Reliability

Application should be able to handle small graphs of up to tens of vertices and edges.

3.3.3 Performance

User interaction should incur very little penalty, and should only degrade in relation to the number of vertices and edges are in the graph.

3.3.4 Supportability

3.3.5 Implementation

I recommend that you stick to the simple(r) Java API for graphical entities known as the Java Advanced Windowing Toolkit (AWT). If you want to learn Java Swing, then by all means go ahead, but recognize that you are going to be in for a more difficult time.

3.4 System Models

3.4.1 Scenarios

Not present

3.4.2 Use case model

The following use cases are initiated by the user:

- **Launch GraphView**
- **Create New Graph**
- **Save Graph**
- **Load Graph**
- **Draw Graph**

The following use cases are more fine-grained, and reflect the actions over the graph:

- **Select Modality**
- **Add Vertex**
- **Remove Vertex**
- **Connect Two Vertices**
- **Disconnect Two Vertices**

Along the way, there will always be three views present. The challenge of this application is to properly ensure that the views are all synchronized as the graph model changes.

4. Extended Use Cases

Some of the details in these use cases implies a specific user interface modality, which I am suggesting because of the implied simplicity in implementing it. You are free, of course, to go further than this and capture what you believe to be the better interface. I will say at the outset that I would not choose to implement the application based upon the interface use cases that I am presenting here, but I do not want to make this assignment so focused on GUI-based applications and their idiosyncrasies.

Use Case Name	Launch GraphView
Participating Actor(s)	Initiated by User
Flow of Events	1. Player executes the GraphView application 2. The application opens with a new graph “Untitled” already in place.
Entry Conditions	• None
Exit Conditions	• User is editing a new graph, tentatively called Untitled
Quality Requirements	•

Draft Document: HW2 GraphView Requirements Document

There is always a “current graph” that is being edited, so the user is never given the option to “close” graph.

Use Case Name	Create New Graph
Participating Actor(s)	Initiated by User
Flow of Events	1. Player selects “New Graph” option 2. If an existing graph is in progress, the user is given the choice to save it first 3. A new empty graph is created.
Entry Conditions	• None
Exit Conditions	• New empty graph is presented to the user.
Quality Requirements	• If the user cancels the request to “save the existing graph” then the existing graph will not be saved and the new graph will be constructed.

The current graph can be saved to a file, to be shared or loaded later.

Use Case Name	Save Graph
Participating Actor(s)	Initiated by User
Flow of Events	1. Player selects “Save Graph” option 2. User is prompted with a File Dialog asking them to select a location where the file is to be saved; cancel terminates the use case 3. Once File has been designated, then description of graph is written to file according to an XML standard format (see appendix)
Entry Conditions	• None
Exit Conditions	• File on disk is created with graph representation.
Quality Requirements	• XML-generated file should be interchangeable with other groups. • The file type “owned” by the program is “.grp” which stands for graph. Files generated by the program should have this suffix.

To load a graph, just make sure you have access to the proper file on disk.

Use Case Name	Load Graph
Participating Actor(s)	Initiated by User
Flow of Events	1. Player selects “Load Graph” option 2. If an existing graph is being displayed (and it has been modified) the user is asked

	whether they wish to save the graph first; if they choose to, then the Save Graph use case is followed. 3. User is prompted with a File Dialog asking them to select a location where the file is to be loaded from; cancel terminates the use case 4. Once File has been designated, then the graph is attempted to be loaded from the designated file; if successful, the new graph is displayed to the user.
Entry Conditions	• Desired graph exists on secondary storage
Exit Conditions	• Newly loaded graph is presented to the user
Quality Requirements	• XML-generated file should be interchangeable with other groups. Files loaded by the program should assume the suffix ".grp"

Within the application itself, there are a set of actions that are to be completed by the user. Each one, in its own way, controls the modification of the graph. There is also a "type command here" text field at the GUI bottom where users can type commands in by hand that replicate the actions to be performed via the mouse.

Use Case Name	Select Modality
Participating Actor(s)	Initiated by User
Flow of Events	1. Player clicks on one of the tool buttons (Add Vertex, Remove Vertex, Add Edge, Remove Edge) 2. The current Mode is displayed in the upper right-hand corner.
Entry Conditions	• None
Exit Conditions	• Until the mode is changed, all mouse clicks within the main canvas are interpreted according to the current mode
Quality Requirements	• When a new graph is created, the tool modality is reset to "Add Vertex"

Use Case Name	Add Vertex
Participating Actor(s)	Initiated by User
Flow of Events	1. Tool mode is "Add Vertex" and user clicks within the canvas at location (x,y); OR user types "add x y" within the command Field 2. If the (x,y) position is "too close" (within 5 pixels) to the location of an existing vertex, the request is denied and a "beep" is heard 3. The vertex is added 4. All open views of the graph are re-drawn

Entry Conditions	• None
Exit Conditions	• If desired vertex is not "too close" to existing vertices, then it is added to the graph
Quality Requirements	• None

Use Case Name	Remove Vertex
Participating Actor(s)	Initiated by User
Flow of Events	1. Tool mode is "Remove Vertex" and user clicks within the canvas at location (x,y); OR user types "remove x y" within the command Field 2. If the (x,y) position is within 2 pixels of an existing vertex (there must be only one), then that vertex is removed from the graph. If there are any edges *incident* to the vertex, then the user is prompted "Do you want to remove this node with N edges?" – a denial would cancel the use case. 3. The vertex is removed (as well as any incident edges) 4. All open views of the graph are re-drawn
Entry Conditions	• None
Exit Conditions	• If desired vertex is not "too close" to existing vertices, then it is added to the graph
Quality Requirements	• Make sure that user cannot delete two vertices by a single click; since vertices can't be closer than 5 pixels to each other, then we expect the delete restriction of 2 pixels will properly hone in on exactly one vertex. • Note that text-oriented GUI person can press "Return" or "Space" when prompted "Do you want to remove this node with N edges?"

Edges are to be added/removed using a two-step process.

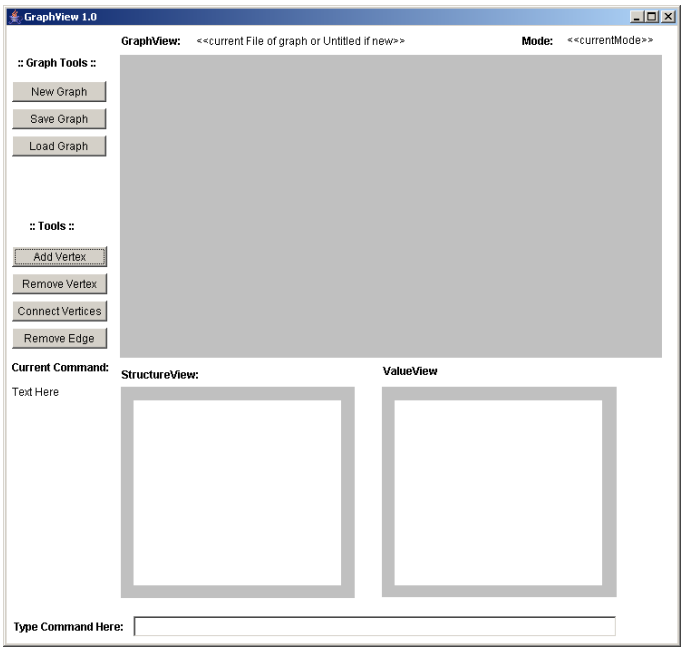
Use Case Name	Disconnect Two Vertices
Participating Actor(s)	Initiated by User
Flow of Events	1. Tool mode is "Remove Edge" and user presses the mouse at location (x1,y1) and drags to release the mouse at location (x2,y2); OR user types "remove x1 y1 x2 y2" within the command Field 2. If the (x1,y1) position is not within 2 pixels of an existing vertex, system beeps; otherwise, if the (x2, y2) position is not within 2 pixels of an existing vertex, system beeps. If there is an edge between the two selected vertices, it is removed; otherwise the system beeps 3. All open views of the graph are re-drawn
Entry Conditions	• None
Exit Conditions	• Edge between two selected vertices is removed

Quality Requirements	• None

Adding an edge is accomplished in the same way

Use Case Name	Connect Two Vertices
Participating Actor(s)	Initiated by User
Flow of Events	1. Tool mode is “Connect Vertices” and user presses the mouse at location (x1,y1) and drags to release the mouse at location (x2,y2); OR user types “connect x1 y1 x2 y2” within the command Field 2. If the (x1,y1) position is not within 2 pixels of an existing vertex, system beeps; otherwise, if the (x2, y2) position is not within 2 pixels of an existing vertex, system beeps. If there is already an edge between the two selected vertices, the system beeps; otherwise it is added. 3. All open views of the graph are re-drawn
Entry Conditions	• None
Exit Conditions	• If selected edge exists, then it is removed.
Quality Requirements	• None

5. Sample GUI



6. Appendix

XML graph is stored as following format [note: this is a draft specification and may change]. This interchangeable format should enable graphs written by one group to be loaded by another group.

```
<graph>
<vertices>
  <vertex id=”ID” x=”xCoord” y=”yCoord”/>
  ...
</vertices>
<edges>
  <edge src=”ID1” target=”ID2”/>
  ...
</edges>
</graph>
```

7. End Notes

1. [2/20/2007 12:42 PM] Detailed Use Cases still to be added.
2. [2/22/2007 2:42 PM] Full Version with all use cases, details of the XML format, and updates screen shots.
3. [3/4/2007 11:28 PM] There is an issue with the unique integer identifier for each vertex in a graph. As mentioned in class, are these labels recycled when vertices are deleted? Think of it this way: when you create a new graph with a single vertex, that vertex might have id “1”. What happens when you delete this vertex and then add a new one back in? Does the new vertex have label “2” (non-recycled) or “1” (recycled). I leave it to you to determine this.
4. [3/4/2007 11:29 PM] note that the DELTA change I will be asking of you will be announced in class on Tuesday March 6th.