Project Report: Graphical Resource Allocation Graph Simulator

1. Project Overview

The Graphical Resource Allocation Graph Simulator is a GUI- grounded tool developed in Python using Tkinter, NetworkX, and Matplotlib to
visually represent and simulate resource allocation between processes and resources in an operating system. The simulator
can determine deadlocks using cycle discovery in directed graphs and helps students or professionals understand how resource operation and deadlocks work

2. Module-Wise Breakdown

Module 1: GUI Interface (Tkinter)
- Provides a simple and intuitive graphical interface.
Module 2: Graph Handling (NetworkX)
- Manages the underlying graph structure.
Module 3: Visualization (Matplotlib)
- Displays the graph visually with nodes and arrows.

3. Functionalities

- Add Resource Allocation
- Release Resource
- Deadlock Detection
- Graph Display

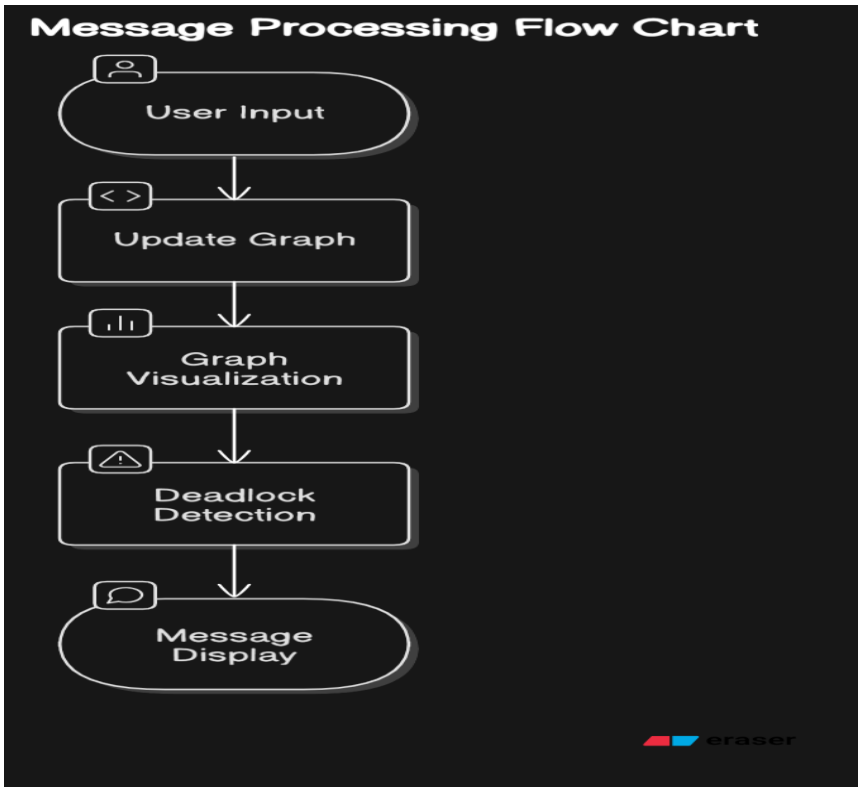
4. Technology Used

Programming Languages:
- Python 3.x

Libraries and Tools:
- tkinter
- networkx
- matplotlib

Other Tools:
- GitHub (for version control)

5. Flow Diagram



6. Revision Tracking on GitHub

Repository Name: Graphical-RAG-Simulator
GitHub Link: https://github.com/Abhinavps5/Graphical-RAG-Simulator
Revisions are manually uploaded with unique commit messages (at least 7).

7. Conclusion and Future Scope

This simulator provides a user-friendly way to understand how deadlocks can occur due to improper resource handling. It serves as a learning aid for OS students.

Future Enhancements:
- Add automatic edge coloring for deadlock paths
- Include logs for actions
- Export graphs
- Save/load sessions

8. References

- https://networkx.org/
- https://matplotlib.org/
- https://docs.python.org/3/library/tkinter.html

Appendix A. AI-Generated Breakdown

The Graphical RAG Simulator breaks down into GUI (Tkinter), Graph Management (NetworkX), and Visualization (Matplotlib). It allows the user to simulate resource allocation between processes, detect deadlocks using cycle detection, and interact through a graphical interface.

Appendix B. Problem Statement

Graphical Simulator for Resource Allocation Graphs
Create a visual tool to simulate resource allocation graphs and analyze deadlock scenarios interactively.

Appendix C. Solution/Code

```
import tkinter as tk
from tkinter import messagebox
import networkx as nx
import matplotlib.pyplot as plt

class RAGSimulator:
    def __init__(self, root):
        self.root = root
        self.root.title("Graphical Resource Allocation Graph Simulator")
        self.graph = nx.DiGraph()

        self.label_process = tk.Label(root, text="Process:")
        self.label_process.grid(row=0, column=0)
        self.entry_process = tk.Entry(root)
        self.entry_process.grid(row=0, column=1)

        self.label_resource = tk.Label(root, text="Resource:")
        self.label_resource.grid(row=1, column=0)
        self.entry_resource = tk.Entry(root)
        self.entry_resource.grid(row=1, column=1)

        self.btn_add_edge = tk.Button(root, text="Allocate Resource",
command=self.add_edge)
        self.btn_add_edge.grid(row=2, column=0, columnspan=2)
        self.btn_remove_edge = tk.Button(root, text="Release Resource",
command=self.remove_edge)
        self.btn_remove_edge.grid(row=3, column=0, columnspan=2)
        self.btn_detect_deadlock = tk.Button(root, text="Detect Deadlock",
command=self.detect_deadlock)
```

```
        self.btn_detect_deadlock.grid(row=4, column=0, columnspan=2)
        self.btn_show_graph = tk.Button(root, text="Show Graph",
command=self.show_graph)
        self.btn_show_graph.grid(row=5, column=0, columnspan=2)

    def add_edge(self):
        process = self.entry_process.get()
        resource = self.entry_resource.get()
        if process and resource:
            self.graph.add_edge(process, resource)
            messagebox.showinfo("Success", f"Edge added: {process} → {resource}")
        else:
            messagebox.showwarning("Input Error", "Please enter both process and
resource.")

    def remove_edge(self):
        process = self.entry_process.get()
        resource = self.entry_resource.get()
        if self.graph.has_edge(process, resource):
            self.graph.remove_edge(process, resource)
            messagebox.showinfo("Success", f"Edge removed: {process} → {resource}")
        else:
            messagebox.showwarning("Not Found", "Edge does not exist.")

    def detect_deadlock(self):
        try:
            cycle = nx.find_cycle(self.graph, orientation='original')
            messagebox.showerror("Deadlock Detected", f"Cycle found: {cycle}")
        except nx.NetworkXNoCycle:
            messagebox.showinfo("No Deadlock", "No cycles detected.")

    def show_graph(self):
        plt.figure(figsize=(6,4))
        pos = nx.spring_layout(self.graph)
        nx.draw(self.graph, pos, with_labels=True, node_color='lightblue', edge_color='red',
arrows=True)
        plt.show()

if __name__ == "__main__":
    root = tk.Tk()
```

```
app = RAGSimulator(root)
root.mainloop()
```